CLOUD-TO-CLOUD DATA TRANSFER PARALLELIZATION FRAMEWORK VIA SPAWNING INTERMEDIATE INSTANCES FOR SCALABLE DATA MIGRATION

CALVIN BOEY MUN LEK

MASTER OF SCIENCE (COMPUTER SCIENCE)

FACULTY OF INFORMATION AND
COMMUNICATION TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN
FEBRUARY 2019

CLOUD-TO-CLOUD DATA TRANSFER PARALLELIZATION FRAMEWORK VIA SPAWNING INTERMEDIATE INSTANCES FOR SCALABLE DATA MIGRATION

By

CALVIN BOEY MUN LEK

A dissertation submitted to the Department of Computer and Communication Technology,
Faculty of Information and Communication Technology,
Universiti Tunku Abdul Rahman,
in partial fulfillment of the requirements for the degree of
Master of Science (Computer Science) in
February 2019

ABSTRACT

CLOUD-TO-CLOUD DATA TRANSFER PARALLELIZATION FRAMEWORK VIA SPAWNING INTERMEDIATE INSTANCES FOR SCALABLE DATA MIGRATION

Calvin Boey Mun Lek

As enterprises are increasingly embracing the practice of multiple clouds federation, scalable data transfer between cloud datacenters is important from the standpoint of cloud consumers. Many existing works are done from the service provider perspective, requiring insights into the datacenter operations which are not available to the cloud consumer. In this dissertation, a data transfer framework that allows cloud consumers to circumvent the bandwidth limitation by spawning intermediate nodes and perform parallel transfer through many-to-many nodes is proposed. However, the effectiveness of such approach depends on many factors such as the time required to spawn new nodes, and bandwidth between the nodes. The objective of this work is to investigate the limitation and potential of the cloud-to-cloud parallel transfer (CPT).

Firstly, all the components needed in the parallel data transfer is identified and modelled. Based on the transfer time and cost models, the circumstances where parallel transfer is worthy is identified. Then, a few optimizations are proposed, namely pipelining and network data piping to increase the data transfer throughput. Pipelining enables each stages of the parallel transfer to work concurrently while network data piping reduces the time spent on dividing files into chunks. Secondly, selected cloud Virtual Machines (VM) are benchmarked. Based on the observed behavior, pre-testing and VM-type selection techniques are proposed. Pre-testing utilized nodes top performing nodes while VM-type selection utilize suitable VM type and sizing. Thirdly, the CPT is implemented and tested on Amazon EC2. The adapted CPT for transfer between Hadoop clusters is also tested. The results showed that the transfer time of CPT is not only lesser than DistCp, but also has a lower cost – up to 8x in certain scenario.

ACKNOWLEDGEMENTS

I am deeply grateful to my main supervisor, Dr. Ooi Boon Yaik for encouraging me to embark on the journey of pursuing a Master's Degree by research. He has also supported me and provided guidance towards my research work and the completion of this dissertation. I would also like to express by gratitude to my co-supervisor, Dr. Liew Soung Yue for providing valuable feedbacks on the methodology employed and reviewing the various writings.

Being a part-time postgraduate student while being employed full-time has been challenging yet fulfilling. I would like to thank my employer, Hilti Group and my managers, Dr. Christoph Baeck and Mr. Ng Eng Siong for being tremendously supportive and understanding as I occasionally need to take time off from work.

Finally, I would like to thank my parents and family for encouraging me to further my studies. My family is my source of strength and inspiration to not only strive for excellence in whatever I do, but to also enjoy the journey.

APPROVAL SHEET

This dissertation entitled **"CLOUD-TO-CLOUD DATA TRANSFER PARALLELIZATION FRAMEWORK VIA SPAWNING INTERMEDIATE INSTANCES FOR SCALABLE DATA MIGRATION"** was prepared by CALVIN BOEY MUN LEK and submitted as partial fulfillment of the requirements for the degree of Master of Computer Science at Universiti Tunku Abdul Rahman.

Approved by:

____
(Dr. Ooi Boon Yaik) Date: _______________
Main Supervisor
Department of Computer Science
Faculty of Information and Communication Technology
Universiti Tunku Abdul Rahman

____
(Dr. Liew Soung Yue) Date: _______________
Co-supervisor
Department of Computer and Communication Technology
Faculty of Information and Communication Technology
Universiti Tunku Abdul Rahman

SUBMISSION SHEET

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITY TUNKU ABDUL RAHMAN

Date: _______________

It is hereby certified that ___Calvin Boey Mun Lek__(ID No: __15ACM06528_) has completed dissertation entitled _"CLOUD-TO-CLOUD DATA TRANSFER PARALLELIZATION FRAMEWORK VIA SPAWNING INTERMEDIATE INSTANCES FOR SCALABLE DATA MIGRATION"_ under the supervision of __Dr. Ooi Boon Yaik _-(Supervisor) from the Department of Computer Science, Faculty of Information and Communication Technology, and __Dr. Liew Soung Yue __(Co-Supervisor) from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my dissertation in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

____

(Calvin Boey Mun Lek)

DECLARATION

I, ___Calvin Boey Mun Lek__ hereby declare that the dissertation is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTAR or other institutions.

____

(CALVIN BOEY MUN LEK)

Date ______________________

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

2D	Two-dimensional
AWS	Amazon Web Services
CDN	Content Distribution Network
CPT	Cloud-to-Cloud Parallel Transfer
CSP	Cloud Service Provider
DC	Datacenter
DN	Hadoop Datanode
FTP	File Transfer Protocol
HDFS	Hadoop Distributed File System
HTTP	Hyper Text Transport Protocol
IaaS	Infrastructure-as-a-service
ISP	Internet Service Provider
MQTT	Message Queue Telemetric Transport
NN	Hadoop Namenode
OS	Operating System
P2P	Peer-to-Peer
PFTP	Parallel File Transfer Protocol
SCP	Secure Copy Protocol
SLA	Service Level Agreement
TaaS	Transfer-as-a-service
TCP/IP	Transmission Control Protocol/Internet Protocol
TD	Transfer Daemon
TM	Transfer Manager

VM	Virtual Machine
vs	Versus
WAN	Wide Area Network

CHAPTER 1

INTRODUCTION

The rapid growth of cloud services such as Infrastructure-as-a-service (IaaS) enables cloud consumer to rent Virtual Machines (VM) from the Cloud Service Provider (CSP) by paying a fixed rate on per unit time basis (e.g. per minute). The rate depends on the amount of CPU, memory, storage, and network resources allocated (“AWS | Amazon EC2 | Pricing,” n.d.)(“Pricing - Linux Virtual Machines | Microsoft Azure,” n.d.)(“Google Compute Engine Pricing | Compute Engine Documentation,” n.d.). One of the advantages of public cloud computing is the capability to geo-distribute the application and data across multiple datacenters located around the world. However, existing IaaS cloud-to-cloud data transfer solutions for cloud consumer suffers from a major shortcoming when performing data transfer between a source VM to a destination VM, the maximum throughput of the transfer is constrained by the bandwidth allocation set by the cloud service provider. Solutions such as breaking down large files into smaller pieces and transferring them in parallel by opening multiple simultaneous connections in order to maximize the bandwidth utilization will not be able to circumvent the bandwidth allocated by the cloud service provider for a VM.

There are specific cases where urgent data transfer has to be completed in the shortest time possible and the cost of transfer is secondary. For example, in the event of disaster recovery and the priority is to have data transferred from the backup site within a short period of time. In addition to that, enterprises today are generating data in large volume which could potentially be caught in cloud vendors lock-in (“Dealing with cloud storage service providers: Avoiding vendor lock-in,” n.d.) which is naturally a scenario favored by many cloud service providers. There are also times when enterprises are federating the workloads and data across multiple cloud datacenters – sometimes federating the data across different cloud providers. As such it is of great importance for enterprises to ensure that they can freely and securely move files across datacenter in a timely manner.

To the best of our knowledge, there is currently no way to scale VM-to-VM data transfer within the public cloud environment that can be performed solely by cloud consumer. Many existing large-scale data transfer requires insights and intervention of datacenter operators. Although cloud service providers increasingly offer data transfer services (“Azure Import/Export,” n.d.)(“Batch Cloud Data Transfer | AWS Snowball,” n.d.), however, not all the needs of the cloud consumer are met. Therefore, scalable data transfer solution that is easily implemented by cloud consumer are getting more and more important.

Since the number of IaaS resources on many public clouds are virtually unlimited, therefore this work proposed a technique to speed up transfer via spawning intermediate nodes (i.e. VM) and aggregate their bandwidth by performing a many-to-many nodes data transfer.

1.1 Problem Statement

As the maximum throughput of the transfer is constrained by the bandwidth allocation per VM set by the cloud service provider, there is opportunity to aggregate bandwidth via parallel transfer. The concept of parallel transfer is to divide the data and transfer the parts simultaneously across multiple channels. Although the concept of scaling parallel transfer via spawning intermediate nodes sounds simple on the high-level, its effectiveness is affected by many factors.

a) The process of parallel transfer incurs additional overhead and may not improve performance in certain cases. Identifying the effectiveness of a parallel transfer upfront before the actual transfer often requires meticulous performance benchmarking and calculation.

b) There are factors that are unique to public clouds such as cost and throughput variability which affect the efficiency and the effectiveness of parallel transfer. This is because different cloud service provider will have different charging model and different virtual machines (VM) performance.

c) Existing parallel transfer approaches such as GridFTP and DistCp have yet to take advantage of cloud elasticity. They are not designed to perform on-demand scaling. These techniques are designed to maximize the utilization of bandwidth of existing infrastructure and are not designed to scale the data transfer to through cloud elasticity.

1.2 Objectives

This aim of this goal is to create a framework to assist cloud consumers to identify the potential and limitation of parallel transfer between clouds and to estimate the performance and cost effectiveness upfront before performing parallel transfer. The framework is hereon addressed as Cloud-to-Cloud Parallel Transfer (CPT). Therefore, the research objectives of this work can be summarized to the following: -

1. To model and identify the limiting factors of scaling cloud-VM to cloud-VM parallel transfer via spawning intermediate nodes.
2. To validate and enhance the model by identifying the influencing factors of implementing the CPT model in the public cloud environment (i.e. AWS) and only using insights within the reach of cloud consumers.

3. To devise an actual CPT solution based on the model and compare it with state-of-the-art parallel file transfer techniques such as Hadoop Distributed File System (HDFS)'s DistCp in terms of performance and cost.

1.3 Project Scope and Assumptions

In general, the research focuses on identifying the factors and designing a framework for cloud-VM to cloud-VM parallel transfer via spawning intermediate nodes. The project and research scope are defined as below:

1. Low-level network and data transfer protocol configuration such as TCP/IP tuning is not the focus of this work. However, note that any improvement made to these areas will result in better performance as our framework relies on these typical transfer protocols.
2. The research focus on VM-to-VM data transfer across different cloud datacenter. Live application migration is not part of the study.
3. Limited to data manifested as objects residing on the operating system (OS) filesystem. For example, database system is not considered under this study. Semantic in object and storage such as file permission and metadata will also not be taken into consideration.
4. The work excludes finding the point of diminishing return as the maximum network throughput is below the maximum disk throughput achievable.

In this research work, the following assumptions were made:

1. The cloud provider limits network throughput per VM, and there is no limit on the account or tenant level.

2. There is more than sufficient resources in the VM pool managed by the cloud provider. It is always possible to acquire VMs on-demand – request is never denied.

1.4 Research Contribution

The major contributions of this research are as follows: -

1. The proposed approach introduced the technique of spawning intermediate nodes in order to circumvent bandwidth allocation and ultimately reducing the time taken for bulk data transfer across cloud DCs for cloud consumers. The work introduces a highly scalable and cost effective parallel transfer method and model.
2. Based on the derived model and understanding the various factors involved (e.g. cost, throughput) in implementing the proposed approach in a cloud environment, a framework named Cloud-to-Cloud Parallel Transfer (CPT) is developed to ease the end-to-end process.
3. The proposed framework is implemented on the AWS public cloud platform and compared against sequential transfer as well as DistCp - a parallel transfer in the Hadoop cluster environment. CPT demonstrated that:

- Better scalability than other solution – reduced transfer time and at a lower additional cost incurred.
- The framework is adaptable to various kind of existing cluster or application.

1.5 Organization of Dissertation

The remainder of this dissertation is organized as follows. In Chapter 2, a thorough literature review is presented in order to justify the research design. In Chapter 3, the development of models of the CPT is described. Chapter 4 is dedicated to understand the relevant behaviors of public cloud and analyze the CPT model developed in Chapter 3. Chapter 5 describes the detail implementation of the CPT framework and Chapter 6 presents the experimental and results of the proposed CPT on a real cloud. Chapter 7 concludes our work.

CHAPTER 2

LITERATURE REVIEW

The chapter provides both an overview of current state-of-the-art in the area of data transfer and in-depth analysis of works that are closely related to ours. The chapter is organized as follows. Section 2.1 provides an introduction to general data transfer and data transfer across datacenters. Section 2.2 and 2.3 presents cloud data transfer from a different perspective. Section 2.4 describes related work in the area of cloud VM bandwidth and pricing. At the end of each section, a summary is provided to compare the pros and cons of the technique. Section 2.5 summarises the chapter.

2.1 Introduction and Inter datacenter data transfer

In the area of cloud computing, there are two main parties in the ecosystem, namely the cloud service provider (CSP) and the cloud consumers. The CSP manages the software and hardware infrastructure by exposing the consumable services via a unified interface. The cloud consumers are charged according to their usage. The two common pricing models are pay-as-you-go model and contractual upfront payment for reservation of resources. CSP often employ multi-tenancy architecture such that multiple consumers (customers) share a common infrastructure.

Therefore, data transfer across different datacenters can also be categorized into 2 categories based on the perspective, namely the CSP perspective and the consumer perspective. Figure 2.1 depicts the 2 categories and the corresponding related works.

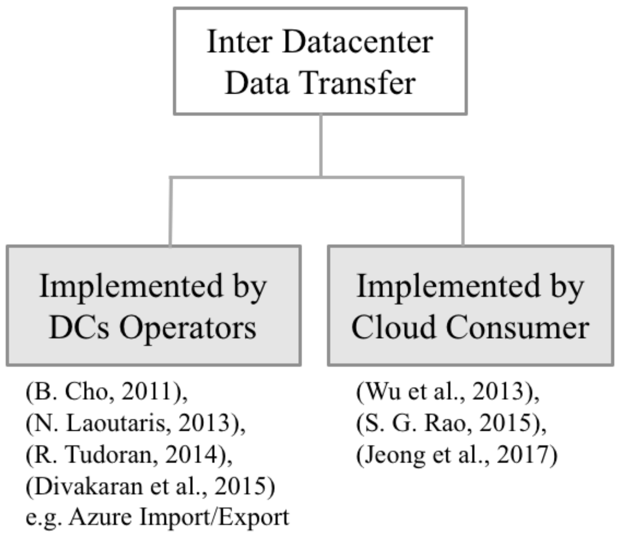

Figure 2.1 Classification of Inter Datacenter Data Transfer based on perspective

There are many works done by the research community for cost-effective bulk data transfer. Most of the existing work focused on reducing the 95th percentile of bandwidth usage as this is how the Internet Service Provider (ISP) charges the CSP. However, the number of research work from the cloud consumer's perspective is much lesser. This work is classified into the cloud consumer's perspective category.

As briefed in the chapter's introduction, the chapter is divided into 3 major sections. Firstly, data transfer that is implemented by the DCs Operator. Secondly, works that can be implemented by the cloud consumer. Thirdly, cloud VM bandwidth and pricing.

2.2 Implemented by DC Operator / Cloud Service Provider

NetStitcher (Laoutaris et al., 2011) uses multi-hop and multi-path for data transfer between datacenters. The system contains a volume prediction module as bandwidth is assumed to exhibit a periodic behavior. It is able to infer available future bandwidth and adapts to estimation errors and failures. In a later work, the same authors propose a technique for transferring bulk data that are delay tolerant through ISPs by utilizing the already-paid-for off-peak bandwidth resulting from percentile pricing and diurnal traffic patterns (Laoutaris et al., 2013). These techniques are excellent for DC operators as it potentially results in significant cost savings when performing data transfer. However, both of these techniques require insight into the network conditions which is only limited to cloud service providers. Such benefit is mostly enjoyed by the DC Operator, and not the cloud consumer.

Tudoran et al. (2014a) describe Transfer-as-a-Service (TaaS), allowing cloud service provider to consolidate customers' data transfer request and route it through a common and managed infrastructure. The users of multi-site or federated clouds benefit from increased data transfer throughput while service providers potentially enjoy the decreased energy consumption due to the consolidation effort. Three scenarios were presented; multi-route user transfers where the user controls the degree of parallelism, asymmetric TaaS approach where intermediate node is only present on one endpoint (DC on one side of the transfer) and symmetric TaaS approach where intermediate nodes are present on both source and destination DCs. Similarly, Divakaran and Gurusamy (2015) explored algorithms and pricing strategies for bandwidth guarantees in the clouds. Both the work above requires the CSP implementation and jurisdiction, which does not help cloud consumer to scale its transfer especially when the demand is high and other consumers are willing to pay more for guaranteed service level agreement (SLA).

Exploring a similar idea, the same authors then proposed a data management solution directed for applications running across geographically distributed locations which offers consistent and predictable transfer time and cost (Tudoran et al., 2014b). It utilizes multi-pathing and multi-hop path splitting involving multiple DCs. The system builds and adapts models for the cloud infrastructure on-the-fly to efficiently optimize and schedule the data transfer process. Each node consists of three modules, the decision manager (DM), transfer agent (TA) and monitoring agent (MA). The decision manager coordinates data transfers, either through direct paths or using multiple intermediate datacenters. The transfer agent performs the transfer and provides the option to exploit network parallelism while the monitoring agent monitors the cloud environment and reports to the decision manager. Our system implementation shares a similar architecture – the separation between data transfer process and control process.

In order to attract consumers to deploy their application in the public cloud, cloud platforms such as Amazon Web Service (AWS) and Microsoft Azure offer services related to data transfer. An example is the AWS Import/Export ("AWS Import/Export - Cloud Data Transfer Services," n.d.) and Azure Import/Export ("Azure Import/Export," n.d.) service which allows consumers to transfer data into or out of AWS and Azure respectively using physical storage appliances. This high-throughput option may be suitable for massive amount of data (e.g. one-time migration into the cloud) but it comes at very poor latency as the transfer overhead falls in the range of days or weeks. Another example of CSP provided transfer service is the AWS S3 (i.e. object storage) acceleration which allows consumers to download and upload data by an optimized network path that is servicing the various content distribution network (CDN) endpoints. However, this is not applicable for transfer between VMs.

Cho and Gupta (2011) propose a technique to perform optimal transfer that minimizes transfer latency within a financial cost constraint. Efficient binary search methods are developed to solve this problem. Algorithms for planning bulk transfer via internet and shipping networks are also explored in their previous work (Cho and Gupta, 2010). The study solves the NP-Hard problem of data transfer from different sites to a common sink over the network (i.e. internet) or physically shipping the storage device. The transfer plans take consideration of transfer cost, shipping cost, transfer time (i.e. over the internet or estimation of time spent unpacking and plugging the disk). The focus in these works is on delay tolerant data and requires the ability to export data from datacenter to physical storage. Again many of the needed information is not available to cloud consumers.

The works covered in this section are all primarily concerned about balancing data transfer cost and subsequently performance. NetStitcher, Tudoran's TaaS and Cho's proposal's primary QoS indicator is cost savings without compromising of transfer deadlines. As for the transfer services provided by the CSP, the main purpose is to help customers migrate large volume of data into the cloud by shipping physical storages. The primary QoS is data integrity and secondary QoS is throughput but measured in days. Most of the works utilized multi-hop and multi-path technique, which is one of the technique utilized by our work with one difference – no DC operator's involvement.

2.3 Implemented by Cloud Consumer / Tenant

There are several other studies that consider the perspective of cloud service consumers. For example, SPANStore (Wu et al., 2013) provides an abstraction layer so that cloud consumer can distribute the data across various geo-distributed object storage from different cloud platforms. The work focus on providing a replication scheme that minimizes data duplication and provides a global view of storages. It can be fully implemented by a cloud consumer. However, the work is only applicable to object storage services and do not focus on data transfer between VMs.

CloudMPcast (Garcia-Dorado and Rao, 2015) optimizes the cost of bulk data distribution between cloud datacenters while ensuring end-to-end transfer time is within the specified deadline. Two characteristics of the public cloud is utilised; transfer cost depends on the location of source and destination datacenters, and discounts are offered when the customer exceeds certain volume threshold per datacenter. The method involves routing data transfer via intermediate hops sited in different datacenter. Our work is distinguished by the ability to scale the data transfer (i.e. reducing the time taken to complete data transfer) rather than reducing cost as much as possible.

Similar to our work, Jeong et al. (2017) explored the idea of multi-stream TCP transfer across paths constructed from relay points sited across multiple cloud datacenters. The intention is to search for less congested paths and utilizing multiple paths. The experiment results demonstrated that it is indeed possible to aggregate bandwidth using multiple paths which can be fully implemented by cloud consumers. However, the authors relied on setting up fixed relay points and have yet to study the effects of spawning ad-hoc relay points during the lifetime of the data transfer.

In addition to the few methods outlined here that takes the view of data transfer from a cloud consumer's perspective, the cloud consumer can utilize many other generic data transfer techniques described in the next subsections.

2.3.1 Typical transfer protocol

Conventionally, the well-known protocols such as Hyper Text Transport Protocol (HTTP), File Transfer Protocol (FTP) and Secure Copy Protocol (SCP) are used for point-to-point data transfer. A point-to-point transfer is where a single source exchange data with a destination host and is responsible for managing the data transmission without the presence of intermediate nodes. There are many research work on optimizing these point-to-point transfers (Lu et al., 2005; Yildirim et al., 2016). The usability of the aforementioned methods to reduce transfer time of large volume of data across cloud datacenters is restricted.

As the size of the network increases and the size of application data processed, advancement to data transfer techniques is growingly important. Therefore, data transfer between source and destination such as Rsync ("rsync(1) - Linux man page," n.d.) is introduced, which uses rolling checksum comparison to reduce repetitive data transfer. Also, adaptive data transfer is introduced such as Dsync (Pucha et al., 2008) that avoids resource contention by detecting back-pressure, as well as detecting similarity at a different hierarchical level. Unfortunately, in a cloud environment where VM-to-VM data transfer throughput is often limited by either the physical connection or soft limit set by cloud service providers, therefore usability of the aforementioned methods to reduce transfer time of large volume of data in VM-to-VM data transfer is limited.

In Figure 2.2, this literature review classifies data transfer approaches to 2 broad categories; typical – point-to-point and parallel transfer. From the figure, only parallel transfer techniques can be used to circumvent the data transfer limits set by cloud service providers. The subsequent sections will focus on parallel transfer techniques that can be used to circumvent the data transfer throughput limits per VM set by cloud service providers.

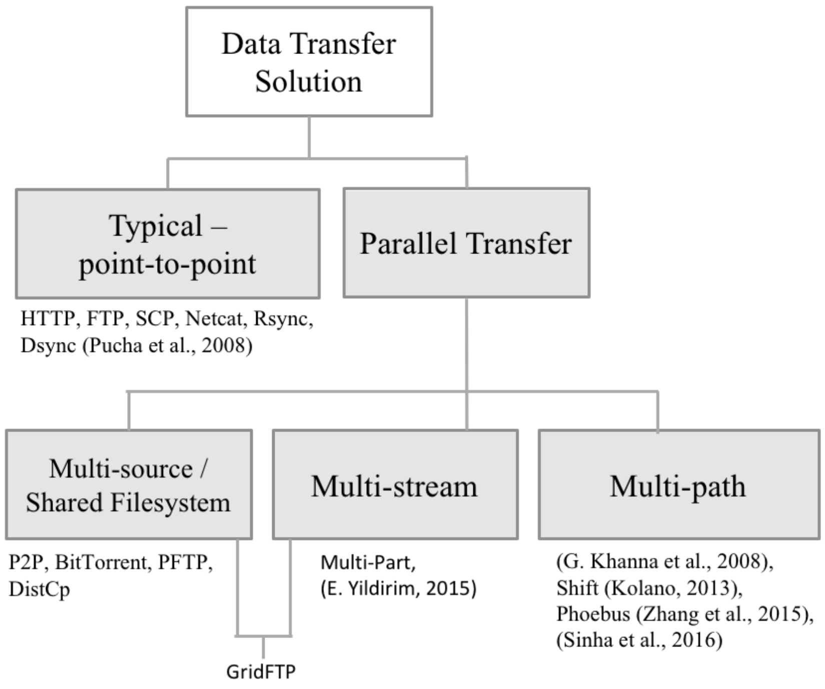
Data Transfer Solution
Typical – point-to-point
Parallel Transfer
HTTP, FTP, SCP, Netcat, Rsync, Dsync (Pucha et al., 2008)
Multi-source / Shared Filesystem
Multi-stream
Multi-path
P2P, BitTorrent, PFTP, DistCp
Multi-Part, (E. Yildirim, 2015)
(G. Khanna et al., 2008), Shift (Kolano, 2013), Phoebus (Zhang et al., 2015), (Sinha et al., 2016)
GridFTP

Figure 2.2 Classification of data transfer solution based on parallelism capability and parallelization technique

2.3.2 Parallel Transfer

One way to circumvent the limit is to chunk large files and perform data transfer across multiple channels simultaneously providing that the overheads for chunking the files into smaller chunks and combining them at the destination is minimal. This technique is practiced in applications that use a Peer-to-Peer (P2P) file sharing protocol such as BitTorrent (“Incentives Build Robustness in BitTorrent,” n.d.), but it is not suitable for a one-off VM-to-VM data transfer.

Multi-part (Hacker et al., 2004) download is a common technique used in off-the-shelf download accelerator manager. The mechanism is to break down large files into smaller segments and downloading them in parallel by opening several simultaneous connections, achieving significantly higher downloading speed. This technique circumvents server-side limitations that restrict bandwidth which fairly allocates bandwidth to each connection equally. Unfortunately, this technique is not suitable to improve VM performance as cloud service provider has set the network bandwidth limit of a VM.

In distributed computing, parallelism in data movement mechanism is often used, especially in the application of high-performance computing. For example, the work presented by Bhardwaj and Kumar (2005), a parallel file transfer protocol (PFTP) is introduced as a concept for data transfer using multiple parallel data paths between clusters. The PFTP protocol makes use of a Parallel File System to stripe data across multiple storage nodes in order to reduce disk I/O bottleneck, which enables transfer via multiple TCP connections simultaneously. GridFTP (Allcock, n.d.) extends the FTP protocol with new features such as partial file access and striping for parallelism. The Globus Toolkit (Allcock et al., 2005), which offers GridFTP as a data movement mechanism, includes widely adopted software packages for implementing grid-based applications. GridFTP is not intended for performing one-time-off point-to-point transfer as it only handles file transfers between GridFTP instances. Unfortunately, the proposed parallel data transfer solutions above are not designed for one time of data transfer in cloud computing environment because these solutions often require complex setups and are not suitable for dynamic infrastructure where VMs are added and removed very often from time to time.

Shift (Kolano, 2013) is a framework for Self-Healing Independent File Transfers that replaces sequential transfer with highly parallel transfer model. Transfer clients are spawned to allow maximum performance when resources are underutilized and put to sleep during overutilization to prevent resource contention. Similar to the PFTP, it requires a common storage accessible by all the transfer clients. In an aspect, the work is quite similar to ours as Shift allows spawning of intermediate nodes to support the transfer. However, the author did not explore the opportunity and the implication of the proposed solution in a public cloud environment.

In the area of data transfer between Hadoop HDFS based clusters, DistCp ("Apache Hadoop Distributed Copy – DistCp Guide," n.d.) is by default included in most Hadoop Framework distribution and it is a de facto standard HDFS data transfer tool. It relies on the uses MapReduce to perform data transfer – distribution, error handling and recovery, and reporting. DistCp is massively parallel using multiple nodes, however, it does not have a built-in capability to scale beyond the initially configured infrastructure. Another method is to utilize GridFTP to perform Wide Area Network (WAN) transfer between HDFS cluster (Liu, 2013; Amin et al., 2011). In order for GridFTP to work with HDFS, a suite of tools has to be setup – FUSE for a POSIX-like interface, BeStMan (Sim, 2009) server as storage resource manager and the GridFTP server. Although the GridFTP is a hugely popular tool among the Grid computing community, it's usage in the HDFS cluster is limited, this is likely due to the equal performance but at a much convenience setup offered by DistCp.

Several research studies tackled the issue of optimizing data transfer across the WAN by using overlaying network – a technique similar to parallel transfer. For instance, two optimization mechanisms for multi-pathing and multi-hop path splitting were proposed by Khanna et al. (2008) to improve the performance of file transfer over WAN. Multi-pathing is the technique of chunking data at the source and transferring it across several overlaying paths. Multi-hop path splitting involves multi-hopping though intermediate nodes instead of relying on a trivial direct connection between the source and the destination node. The author, however, did not study the potential of using this technique to scale the transfer beyond the original infrastructures.

The Phoebus propose an infrastructure deployment for improving data transfer across WAN by using a session layer protocol and gateways in the data distribution network (Kissel et al., 2011; Ramakrishnan et al., 2010). The work by Zhang et al. (2015) explores the feasibility of deploying Phoebus for data transfer between cloud datacenters. Although the performance improvement can be seen in certain cases, however, the biggest setback is the need of setting up quite a number of intermediate nodes and complicated overlaying network across WAN. Besides, the work is focused on reducing latency for critical application and did not focus on bulk data transfer.

Sinha et al. (2016) describes routing data transfer between Storage-as-a-Service (SaaS) such as Dropbox ("Dropbox," n.d.) and Google Drive ("Google Drive - Cloud Storage & File Backup for Photos, Docs & More," n.d.) through intermediate nodes in order to mitigate bottlenecks. CoCloud (E et al., 2018) describes a cloud-to-cloud file collaboration framework for users sharing files across different SaaS. These two works are not directly applicable to VM-to-VM data transfer, however, it shares a similar concept of utilizing intermediate nodes and proxies to bridge the gap between different cloud platform or region.

In summary, parallel data transfer primarily provides improved data transfer throughput and secondly better fault tolerance. Hence, the primary QoS of parallel transfer is throughput. Multi-source or shared filesystem technique is excellent for increasing the data transfer throughput for shared cluster environment or many-to-one transfer. The mechanism makes the assumption that data is already replicated across multiple storage nodes which makes it not suitable for typical one-off data transfer. Multi-stream technique allows circumvention of server-side bandwidth allocation. This method is however not effective for cloud-VM to cloud-VM data transfer which our work is addressing. Multi-hop and multi-path technique, which our work also employs, transfers data across multiple newly discovered paths.

2.4 Cloud VM bandwidth and pricing

As part of the work on proposing bandwidth guarantee in the public cloud environment, the authors in explored state-of-the-art on VM pricing and bandwidth. It is found that for VMs in the same cloud, it's possible that the average bandwidth of a cheaper VM surpasses the bandwidth of a more expensive VM. From the work, it also shows that price does not scale linearly to the network performance. This is likely due to the fact that cloud pricing is derived and set by the cloud provider based on a mixture of various metrics such as CPU, memory, storage, and network. This makes it challenging for cloud consumer to pick the VM type/size that best fit the specific use case in terms of meeting the performance requirement yet at the minimum cost without benchmarking and performance testing.

Detailed information about cloud performance is often kept secret by the cloud provider for security and commercial reasons (Raghavan et al., 2007; Mogul and Popa, 2012). Service Level Agreements (SLA) typically only describe the performance guaranteed (e.g. network bandwidth) vaguely – without precise figures, which customers have no choice but to rely on this qualitative information. Partly it could be due to the best-effort basis of quality of service (QoS) provided by the cloud service provider. Such limited performance information weakens the ability of the consumer to understand the impact on their application such as on data transfers (Wang and Ng, 2010). As part of benchmarking the network throughput in AWS and Azure respectively, it is found that network performance is stable across the lifetime of the VM (Persico et al., 2015; Persico et al., 2017; Scheuner and Leitner, 2018). However, the average network bandwidth allocated for each VM varies even with the same VM type/size (Ou et al., 2013; Gilani et al., 2015). This behavior makes it especially challenging for cloud consumers as the network performance is dependent on when the VM is spawned. These authors recommend testing VM's network performance and discarding the VM if the performance is below the known average.

2.5 Discussion

In the area of data transfer in cloud computing, techniques that have to be implemented by DC Operator / Cloud Service Provider may benefit both CSP and Cloud Consumers. Similar to our work, techniques that can be implemented by Cloud Consumers do not have any significant impact on the CSP. In the area of data transfer solution, various parallel transfer techniques are prevalent and widely used. The methods multi-hop and multi-path are particularly interesting for us as it allows transferring using newly discovered paths, providing the opportunity to bypass limitation set by the CSP.

In summary, based on the presented discussion in this section, there is yet a complete parallel cloud-to-cloud data transfer approach for cloud consumers. Most of the presented studies focus on optimizing data transfer from the cloud service provider perspective and little work is done from the cloud consumers' perspective. Achieving an optimized data transfer rate is indeed a challenging prospect in diverse cloud environments because of the diversity of the deployed hardware, software and quality-of-service agreements. This work addresses this challenge by leveraging cloud elasticity. We first investigate the potentials and limitations of parallel data transfer and then propose a mechanism that utilizes instance-to-instance pipelines. To the best of our knowledge, no other work has proposed similar mechanism that allocates and de-allocates cloud instances to control the performance and cost of data transfer.

CHAPTER 3

PROPOSED SOLUTION

3.1 Cloud Parallel Transfer Concept

In IaaS cloud offering, it is usual that every spawned instance in the cloud are given a specific amount of processing power, memory, storage and network bandwidth. Despite charges are often by the pay-as-you-use model, the maximum amount of allocated processing power, memory, storage and network bandwidth are capped. This work proposes a way to circumvent the network bandwidth limitation by spawning additional instances and share the aggregate bandwidth known as cloud-to-cloud parallel transfer (CPT). In a public cloud platform, the network bandwidth is limited either by the cloud service provider or physical limits of the particular instance.

In this work, the terms virtual machine, instance and node are used interchangeably and are all referring to the same thing – machine that the cloud consumer can rent.

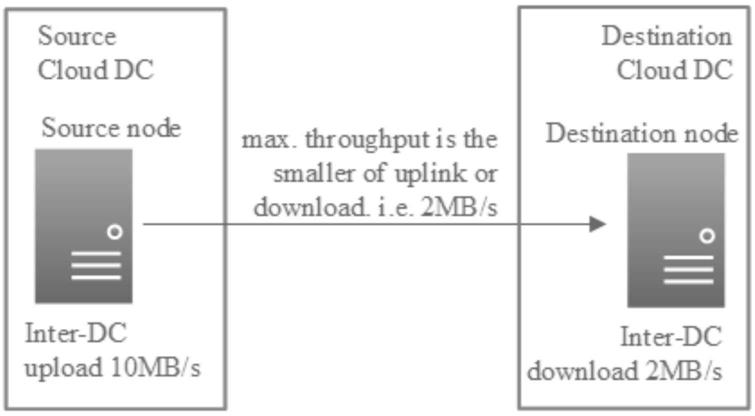

Figure 3.1Maximum throughput of typical point-to-point sequential transfer

Figure 3.1 demonstrates that the total transfer speed between two datacenters is bounded by both the sender's (i.e. source node) maximum throughput and receiver (i.e. destination node) maximum throughput. The maximum throughputs are often different not only between different cloud vendors but also between datacenters of the same vendor.

Figure 3.2 shows that the network bandwidth from different instances can be potentially aggregated through spawning additional intermediate instances. For example, the maximum bandwidth for a pair of instances, from the source instance to the destination instance, is 50mbps, by having 3 pairs of such instances, the bandwidth can be multiplied by 3. Thus, in theory, it is possible to increase the bandwidth by *n* times using *n* pairs of instances.

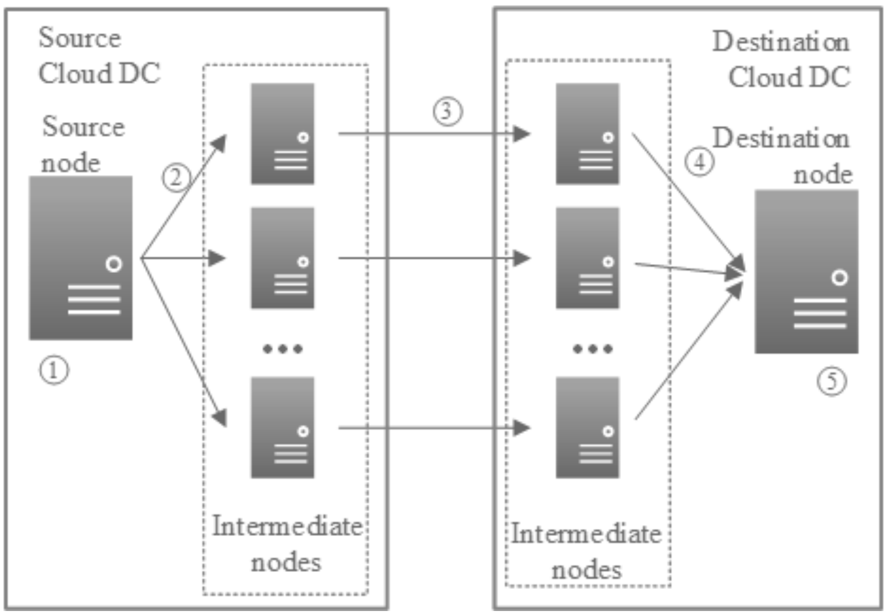

Figure 3.2 A high-level overview of the parallel transfer mechanism

As shown in Figure 3.2, the source node is the node where the data to be transferred is originally residing on. Destination node refers to the target node that the data should be transferred to. Intermediate nodes refer to nodes that are logically between the source and the destination node which allows the data transfer to be parallelized.

However, to perform data transfer over the aggregated bandwidth of multiple pairs of instances is not without limitations. For instance, it is important to note that the speed of the source instance splitting the file into chunks for the intermediate instances could be a limiting factors, as well as the speed of combining the files back at the destination instance can be another limiting factor that render the bandwidth aggregation technique worthless. Therefore in this chapter, a parallel transfer model will be created to identify all the factors influencing the performance of CPT. Then, the costing model is derived to understand the factors impacting the financial cost that is incurred when performing CPT transfer.

3.2 The workflow of the proposed CPT transfer technique

The Cloud-to-Cloud Parallel Transfer (CPT) leverages aggregate bandwidth between the source and destination nodes located in different cloud datacenters by making use of intermediate nodes. These intermediate nodes can either be newly spawned instances or existing underutilized instances.

Based on Figure 3.2, these are the sequence of events in brief, for a CPT transfer from the source node to destination node.

1. In the source node, the consumer initiates transfer of a file(s) from source to destination node. Then, limited network throughput testing and forecast based on the models will be conducted automatically so that a decision on the number of pairs of intermediate nodes to be spawned, p can be made.

   On the source node, the manager spawns p new (or use available) VM instances (hereafter addressed simply as parallel instances or intermediate nodes) in source V_{sj} and destination V_{dj} , j = [1..p] for each cloud datacenter. That is, a total of $2p$ instances are spawned and used as intermediate nodes.

   The file to be transferred is split into n equal sized chunks C_j, j = [1..n], where n is arbitrarily and $n >= p$.

2. Once step 1 is completed, each chunk is transferred from the source node to its corresponding intermediate nodes, i.e. chunk C_j transferred from V_s to V_{sj}.
3. Once the intermediate node in the source DC received the chunk from the source node, the chunk will then be transferred to its corresponding intermediate node in the destination DC, i.e. chunk C_j transferred from V_{sj} to V_{dj}. Note that there is bandwidth limitation imposed by CSP on each connection of an instance.
4. Once the intermediate node in the destination DC received the chunk, the chunk will be transferred to the destination node, i.e. chunk C_j transferred from V_{dj} to V_d.
5. As the chunks are received on the destination node, the utility will merge the pieces to reconstruct the original file(s).
6. Step 1 to 5 repeats until all the chunks have been transferred and the files have been reconstructed. The destination node notifies the manager (i.e. in the source node) that the transfer has completed. The transferred files are verified by matching the checksum. .

3.3 Modelling the performance of CPT

3.3.1 Foundation

In order to keep the work from being overly complicated and to keep it within the scopes of our studies, the number of intermediate nodes in the destination DC will be equal and match in a 1-to-1 manner to the source DC. This simplification does not affect the generality of the model which can, in future work accommodate the case where the number of intermediate nodes in source and destination DCs is not the same i.e. m-to-n mapping. The components of parallel data transfer time are depicted in Eq. 3.1.

CPT transfer time,

$$T_{CPT} = T_v + T_d + T_e + T_c \tag{3.1}$$

The following explains all the components.

a. VM Setup time (T_v)

The VM setup time consists of time for VMs allocation, provisioning, and starting up of all the intermediate nodes. In today's public cloud IaaS platform, VMs can typically be provisioned within few minutes after the request is received, sometimes even within a few ten seconds. However, it must be noted that VM setup time is a significant factor in the functionality of the CPT. Unfortunately, from a cloud consumer's perspective, there is almost nothing that can be done.

b. Data Distribution Time (T_d)

In the proposed model, there are two actions that happen in the same DC, files are chunked and then transferred from the source node to the intermediate nodes. The transferring process has to begin immediately after the split of a particular chunk is completed in order to reduce the overall data distribution time. This means there is no time waiting for the overall file splitting process to complete. However, care must be taken as the process of both splitting and transferring may utilize (i.e. reading and writing into) the same disk which affects throughput due to I/O contention.

c. Data Transfer Time (T_e)

This time component refers to the transfer time from the intermediate nodes of the source DC to intermediate nodes in the destination DC. There are multiple straightforward ways to shorter the duration of this stage, such as using more powerful VMs (i.e. VM with higher network throughput) or increasing the number of pairs of intermediate nodes.

d. Data Consolidation Time (T_c)

Data consolidation time is the amount of time taken for the file reconstruction process to complete. It includes the time of file chunks transfer from intermediate nodes to the destination node within the same DC. This stage is heavy on disk operation as data is first read and then rewritten back into the filesystem. Normally, data consolidation time correlates with the data distribution time. An example, decompression have to be performed during the consolidation stage if compression was performed during the distribution stage. However, this work assumed that the data is already and cannot be further compressed such as high definition multimedia data.

Based on the aforementioned illustration, the CPT total transfer time is represented by the following equation. The transfer speed is the average of all intermediate nodes in both source and destination datacenters. In the remaining of our work, we assumed that the individual transfer speed of each intermediate nodes is identical. The effective transfer speed is the lesser of network throughput and disk throughput.

The equation for the total time taken of CPT is given below. The notations are given in Table 3.1.

Table 3.1. Notations in the CPT time taken equation

Notation	Description
T_{CPT}	CPT transfer time (s)
T_{CPT+}	CPT w/ pre-testing transfer time (s)
T_{sq}	Sequential transfer time (s)
T_v	Stage “VM setup” time (s)
T_{pt}	Stage “Pre-testing” time (s)
s	Total Transfer size (MB)
v_s	Split throughput (MB/s)
v_i	Internal transfer throughput (MB/s)
v_e	External transfer throughput (MB/s)
p	No. of intermediate node pairs

$$T_{CPT} = T_v + \frac{s}{v_s} + \frac{s}{v_i} + \frac{s/p}{v_e} + \frac{s}{v_i} + \frac{s}{v_s} \qquad (3.2)$$

As an example, Figure 3.3 depicts an example timeline of each component for a CPT transfer with 2 pairs of intermediate nodes, p=2.

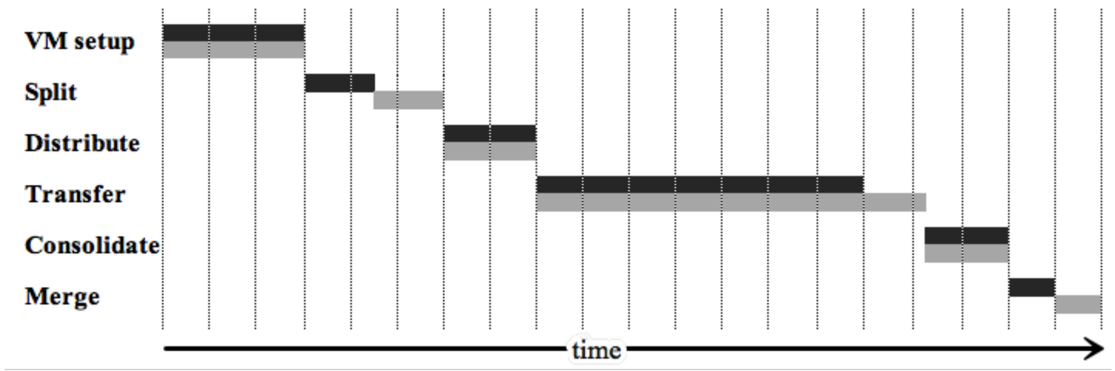

Figure 3.3 Example timeline of basic CPT using 2 pairs of intermediate nodes (p=2)

3.3.2 Introducing Pipeline into the CPT Technique

The Eq. 2 showed that the model can be further improved by using a pipelining technique. In order to make the transfer more worthwhile, the stages such as distribute, transfer and consolidate of file chunks can take place concurrently. As an example, Figure 3.4 depicts the example timeline of CPT transfer with pipelining (p=2).

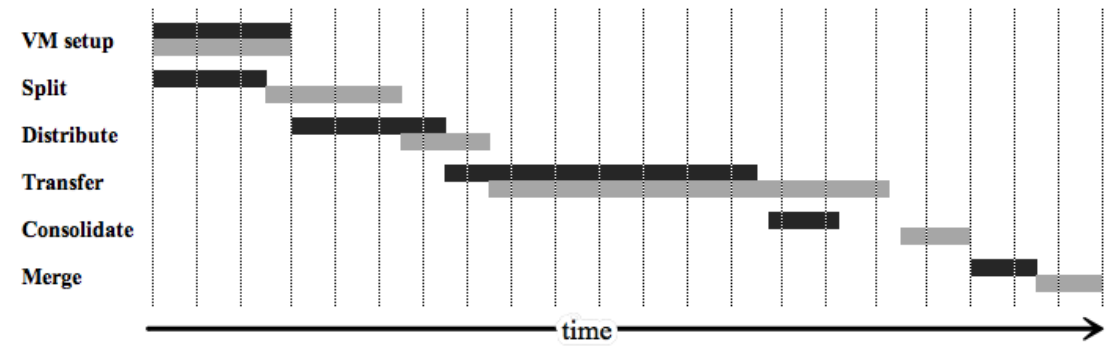

Figure 3.4 Example timeline of CPT transfer using 2 pairs of intermediate nodes (p =2) and with pipelining

For instance, submitting a request for VM spawning and splitting of files can be started simultaneously. Then, once the first data chunk is ready, it will be transferred to the first intermediate node in source DC without waiting for the second chunk (which will be transferred to the second intermediate node). Similarly, once the first chunk is completed, it will be transferred to its respective intermediate node in destination DC. By doing so, we ensure that there is no waiting for transfer since another pair of instance is available. However, the exception is during the file merging process, which can only begin when the transfer of all file chunks is completed. In this model, the total parallel transfer time is influenced by the larger of VM setup or file splitting time. This further refines Eq. 3.2 to Eq. 3.3.

$$T_{CPT} = \max(T_v\,,\ \frac{s}{v_s}) + \frac{s/p}{v_i} + \frac{s/p}{v_e} + \frac{s/p}{v_i} + \frac{s}{v_s} \qquad (3.3)$$

3.3.3 Reducing I/O time with Network Data Piping

From the Equation 3.3, I/O time should be further reduced to improve the parallel transfer efficiency. Therefore, the data splitting and merging stages should be performed virtually. For example, in the “splitting” stage, the physical splitting of the file is not necessary as the file to be transferred can be read from disk at arbitrary location and sent over the network. Similarly, instead of having to wait for all the data to be available before starting the merging process, the merge operation can be eliminated as the data is put in place as part of the network transfer. This can be achieved using tools such as Netcat (“nc - arbitrary TCP and UDP connections and listens - Linux man page,” n.d.). As an example, Figure 3.5 depicts the possible timing sequence of parallel transfer (p=2) with pipelining and network data piping.

Such an approach not only allows better concurrency, but can also off-load the disk IO operation to the network. This should significantly reduce the total parallel data transfer time.

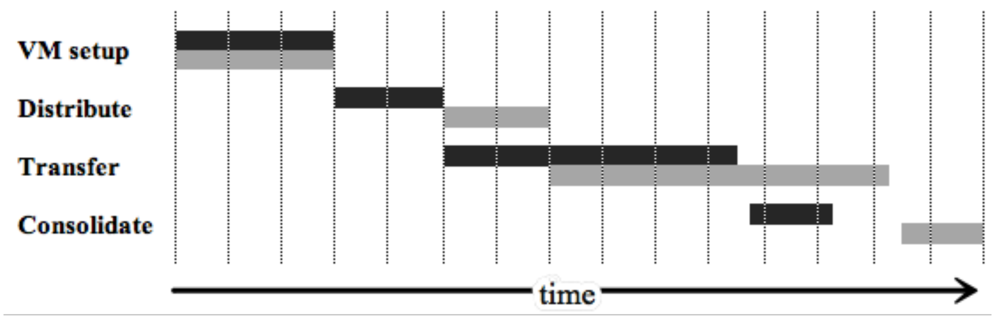

Figure 3.5 Example timeline of CPT using 2 pairs of intermediate nodes (p=2), and with pipelining and network data piping

Therefore, with network file merging, eq. 3.3 can be reduced to the following:

$$T_{CPT} = T_v + \frac{2s}{v_i} + \frac{s/p}{v_e} \qquad (3.4)$$

3.4 Impact towards financial cost of CPT

Utilizing cloud resource comes at a financial cost. Therefore, by spawning additional resources for data transfer, additional operation cost is incurred. Typically, cloud providers do not only charge for compute resources in the event of spawning VMs, but there are also implicit costs such as storage, provisioned disk I/O performance, and network interface. After the transfer is completed, the resources (i.e. intermediate nodes) are deallocated, hence no continuous charges.

Sequential transfer cost, $C_{sq} = s \, . \, C_{eg}$ (3.5)

Table 3.2 Notations in the CPT cost equation

Notation	Description
C_{sq}	Cost of sequential transfer
C_{CPT}	Cost of CPT transfer
C_{CPT+}	Cost of CPT w/ pre-testing transfer
C_{eg}	Egress traffic cost (cent / MB)
C_{ig}	Ingress traffic cost (cent / MB)
C_v	Cost of VM per unit time (cent / s)
n	No. of intermediate node

Assuming internal transfer, C_{ig} is free of charge which is true for most cloud services, CPT Transfer Cost,

$$C_{CPT} = (T_{CPT} \, . \, 2p \, . \, C_v) + (s \, . \, C_{eg}) \qquad (3.6)$$

Since n = 2p, and $C_{sq} = s \, . \, C_{eg}$,

$$C_{CPT} = (T_{CPT} \, . \, n \, . \, C_v) + C_{sq} \qquad (3.7)$$

From the Equation 3.7, since the CPT transfer cost is a summation of the extra processes and sequential transfer cost, it is not possible to reduce the cost of parallel transfer to be lower than the sequential transfer. However, the parallel transfer potentially provides better throughput but comes at an increased in financial cost. This will be studied in greater details in the next few chapters.

3.5 Conclusion

The proposed CPT transfer involves 3 main stages: distributing the data to intermediate nodes, transferring them in parallel across WAN and then subsequently reconstructing the data at the destination node. The model of CPT transfer time shows that VM start-up time and the various network throughput is an important factor of CPT. Finally, the cost model showed that CPT transfer comes at an increased in financial cost. In short, network throughput of the VMs and cost of running the VMs are the main factor in determining the effectiveness of the proposed parallel transfer technique.

In a nutshell, corresponding to the first research objective, this chapter modelled and identified the limiting factors of scaling cloud-to-cloud data transfer via spawning intermediate nodes.

CHAPTER 4

UNDERSTANDING THE CLOUD AND MODEL ANALYSIS

The purpose of this chapter is to understand the characteristics of the public cloud in terms of performance and cost, and then uses the collected performance and cost values to analyze the two CPT models described in Chapter 3, one pertaining to CPT data transfer time and the other on the CPT data transfer cost.

This work uses AWS cloud services. This is because according to Gartner's reports in the year 2018 ("Magic Quadrant for Cloud Infrastructure as a Service, Worldwide," n.d.), AWS is the world market leader in Cloud Infrastructure as a Service.

The tests conducted and results explained in the first half of the chapter sets the stage for understanding the network throughput behavior of a typical public cloud platform. Then, the models are furthered explored, and the CPT framework and process flow are described. The CPT framework proposed 2 techniques – VM-type selection and pre-testing to optimize the performance and cost.

4.1 Cloud's Network Performance and Charges

In order to benchmark the inter-network capacity, this work selected the two further apart AWS datacenters; transfer between the Oregon and Ireland region. The intra-network are tested within Oregon. The network throughput of various type of general purpose EC2 instances within and between these 2 datacenters are measured. They are depicted as internal and external transfer respectively.

The test is performed with iPerf ("iperf - Linux man page," n.d.) with 3 streams and for a duration of 2 minutes each. For each test, EC2 of identical type is spawned in the respective AWS region to carry out the test. The experiment was performed 3 times a day for continuously 2 days (23rd-24th April 2018), then the average is taken (total of 6 readings). The results are presented in Figure 4.1 and 4.2.

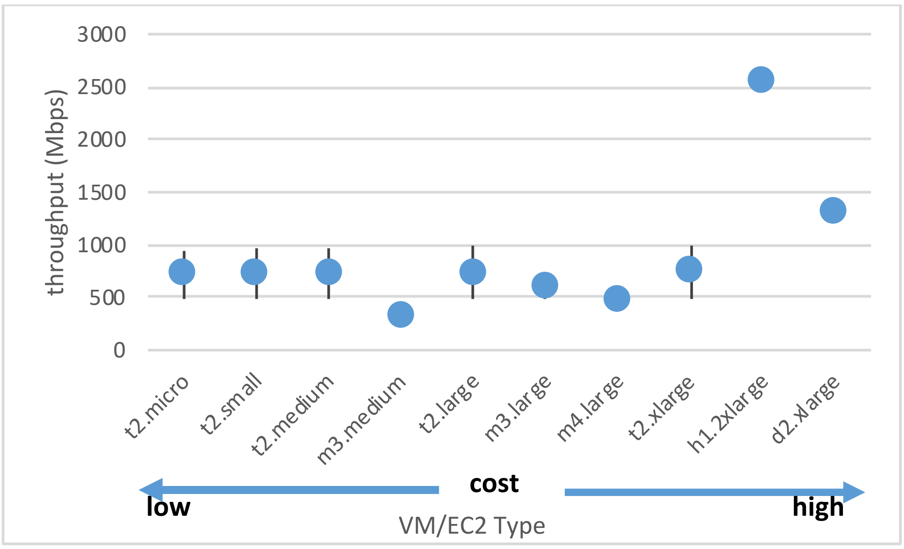

Figure 4.1Average, Minimum and Maximum Intra-DC network throughput observed for the various EC2 type

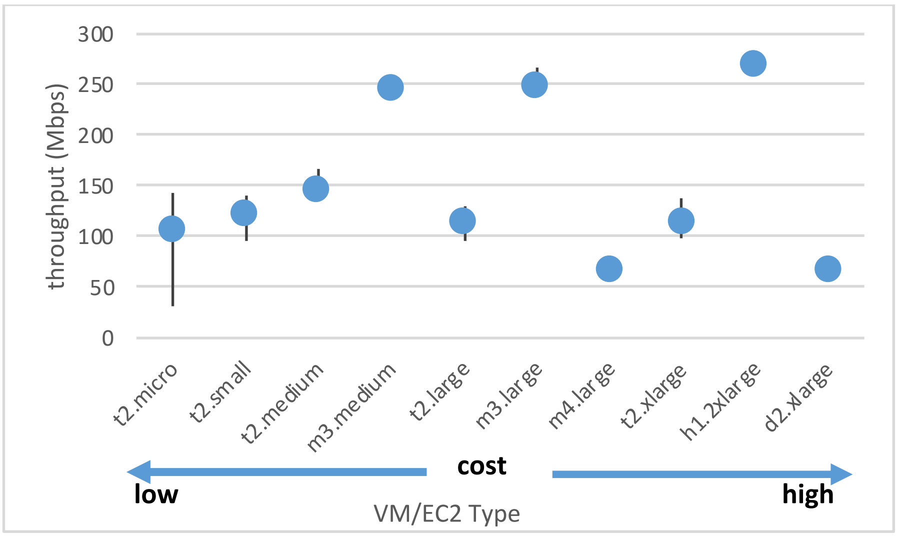

Figure 4.2 Average, Minimum and Maximum Inter-DC network throughput observed for the various EC2 type.

Each bar depicts the average, maximum and minimum of the averages. Consistent to observations made by other authors such as in (Hu et al., 2018), the intra-DC bandwidth exceeds the inter-DC by several factor. On the other hand, it can be seen that the network throughput variability for t2 instance class is relatively large while the throughput of the remaining types is quite consistent. The result is consistent to the official network performance classification provided by AWS (Table 4.1). However because AWS does not provide service level agreement (SLA) for network throughput, the network rating is given very vaguely.

Table 4.1 also presents the cost of running the respective VMs, sorted by the cost ascendingly. As seen, the price of VMs does not directly correlate to the network throughput. It is possible to spend a fraction of the cost and still get a better network throughput. This proposed technique is further explored in section 4.3.

Table 4.1 Price and Spec of running VM/EC2 on AWS (April 2018, AWS Oregon)

Type	vCPU	Memory (GiB)	Network	Price per Hour ($)
t2.micro	1	1	Low-Moderate	0.0116
t2.small	1	2	Low-Moderate	0.0230
t2.medium	2	4	Low-Moderate	0.0464
m3.medium	1	3.75	Moderate	0.0670
t2.large	2	8	Low-Moderate	0.0928
m4.large	2	8	Moderate	0.1000
m3.large	2	7.5	Moderate	0.1330
t2.xlarge	4	16	Moderate	0.1856
h1.2xlarge	8	32	Up to 10 Gbps	0.4680
d2.xlarge	4	30.5	Moderate	0.6900

AWS does not offer network optimized VM per se. That is to say, VM with better network throughput also comes with more CPU and memory resources, and has a much higher cost. We observed that this VM offering model is generally true for many other cloud providers such as Azure ("Azure Linux VM sizes - General purpose," n.d.) and Google Cloud ("Egress Throughput Caps | Compute Engine," n.d.). For this work, we included h1.2xlarge as it is the most affordable VM with AWS network rating of "up to 10Gpbs".

Table 4.2 depicts the result (from the extended test conducted above) of intra-DC network throughput between VMs of a different type. It is observed that when transferring between VMs of different type, the network throughput is the smaller of both. The download and upload bandwidth seem to be capped equally without any favor.

Table 4.2 Network throughputs between VMs/EC2s of a different type

From | To (Mbps)	t2.micro	m3.medium	m4.large	h1.2xlarge
t2.micro	718	331	601	721
m3.medium	323	320	323	326
m4.large	595	326	593	597
h1.2xlarge	735	327	597	2540

Similar characteristics are also observed by the same authors in two separate work (Persico et al., 2015; Persico et al., 2017) – throughout this preliminary tests, the network throughput during the lifetime of the VM is stable. There is no significant fluctuation which means that the resource allocated to the VM is not changed throughout the lifetime of the VM.

The impact of network throughput on the overall performance and cost of the CPT is discussed in the next chapter.

4.2 Model Analysis

Before jumping into the model validation, it must be understood that speedup is defined as S(s, p) = T_{sq} / T_{CPT}

Where time taken for sequential transfer, $T_{sq} = \frac{s}{v_e}$

In order for CPT to have better performance than sequential transfer, the CPT time must be smaller than the sequential data transfer time, i.e. S > 1.

4.2.1 Performance of various CPT and its optimizations

In this section, CPT transfer against sequential transfer is carried out based on the models. In this sub-section, a hypothetical case where the all the intermediate, source and destination nodes are of the same VM type. This means that the network bandwidth is similar across all the VMs.

Based on the preliminary tests carried out on Amazon EC2 cloud type t2.small, the VM startup time varies between 81 and 95 seconds with an average of 87.8 seconds. This demonstrates the performance volatility when requesting a cloud service. In this section, an arbitrary value for VM set-up time, i.e. 90 seconds is used. Also based on a separate test, the split and the merge speed, which are input/output intensive activity, are normally equal. Table 4.3 includes the parameters as follows: 65 MB/s for split throughput, 90 MB/s for internal transfer throughput and 15MB/s for external transfer throughput.

Table 4.3 The numerical figure used in the models

Variable	Numerical figure used in the models
T_v	90 s
v_s	65 MB/s
v_i	90 MB/s
v_e	15 MB/s

Feeding the values in table 4.3 into the equations 3.2, 3.3 and 3.4 results in Figure 4.3 to 4.6. Figure 4.3 shows the transfer time against total data transfer size of basic CPT for different number of intermediate node pairs. Basic CPT with p=2 and p=4 has larger transfer time than sequential transfer, which means performance is worse off. For p=8, the performance is still worse than sequential transfer when total data transfer size is below 10GB. This also demonstrates that CPT is more effective when the data transfer size increases.

Figure 4.4 and 4.5 shows the transfer time of CPT with pipelining and CPT with pipelining and network data piping respectively. Unlike the basic CPT (Figure 4.3), both the optimizations have resulted in better performance even when the number of pairs of intermediate node is low, p=2 in this case. Other than the lower transfer time compared to basic CPT, the general trend and observation remain the same.

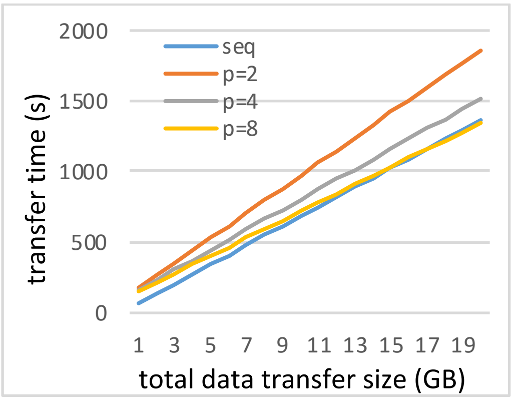

Figure 4.3 Time taken of various number of instance pair for basic CPT.

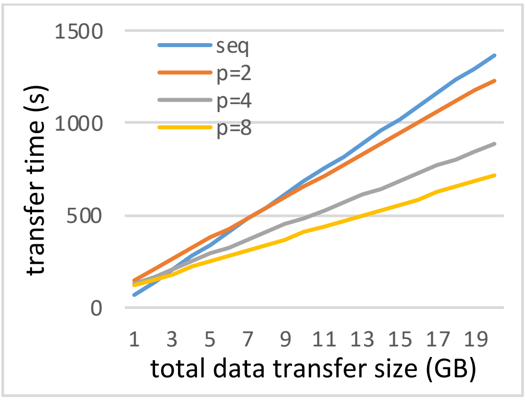

Figure 4.5 Time taken of various number of instance pair for CPT with pipeline and network data piping.

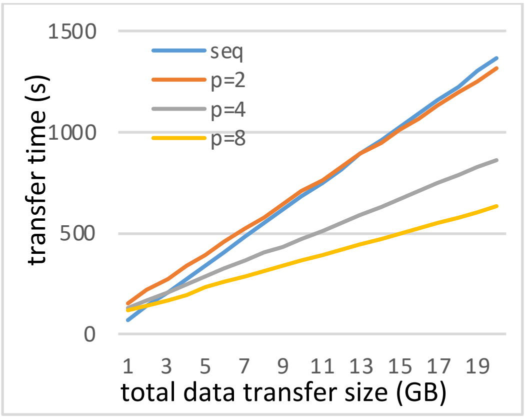

Figure 4.4 Time taken of various number of instance pair for CPT with pipeline.

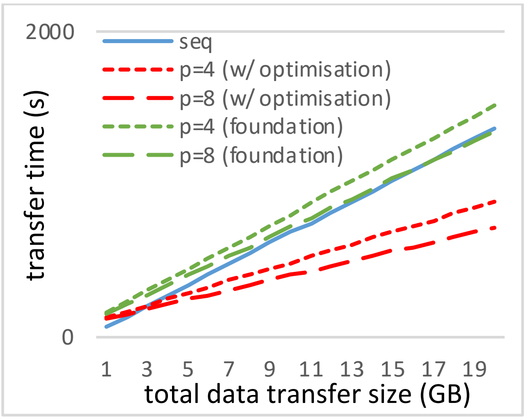

Figure 4.6 Comparing the performance of basic CPT and CPT with optimization.

Figure 4.6 is a combination of Figure 4.3 and 4.5. It compares the basic CPT (i.e. foundation) against CPT with pipelining and network data piping (i.e. optimized CPT). In the figure, it can be seen that the transfer time of basic CPT with p=4 is larger than sequential. This means worse off performance. Optimized CPT with p=4 has a slight improvement. As expected, increasing the number of intermediate nodes pair to 8 give a far smaller transfer time than sequential transfer.

As can be seen from the figures, each improvement proposed increases the overall performance of CPT. Hence, optimized CPT is used for the remaining of the work – implementation of CPT. The work on basic CPT and CPT with pipeline ends at this sub-section.

4.2.2 Performance and Financial Cost Incurred

This section is a continuation of the previous sub-section, where the cost of the CPT transfer is explored here. The equation of transfer time for optimised CPT and cost of CPT (equation 3.4 and 3.7 respectively) is fed with the values mentioned in Table 4.4 to compute the CPT cost. The price is based on the actual average pricing of AWS Oregon and Ireland (as of April 2018).

Table 4.4 Cost of AWS t2.small per second billing

Variable	Numerical figure used in the models
C_{eg}	\$0.02 / GB
C_{ig}	\$ 0
C_v	\$ 0.241 / hour (per second granularity)

For sequential transfer and the different number of intermediate nodes, three key metrics are closely explored:

- Cost against total data transfer size
- Throughput against total data transfer size
- Throughput per cost against total data transfer size

And for each of the metric, the impact on small and large total amount of data transfer is studied. The results are shown in Figure 4.7 to 4.9 and Figure 4.10 – 4.12 respectively and the observations are outlined below. Do take note that the x-axis in the figures is not linear hence the apparent curve.

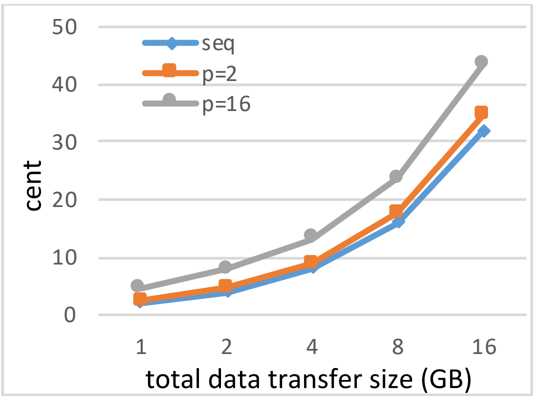

Figure 4.7 Cost vs transfer size (small total data transfer)

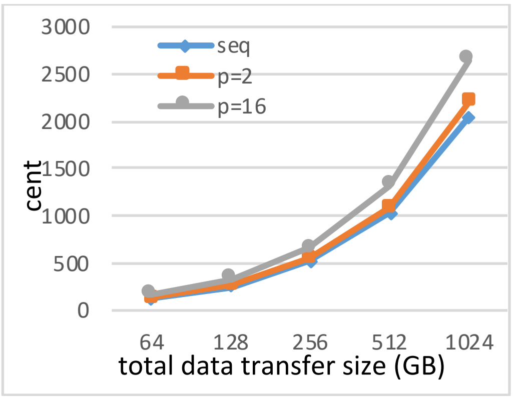

Figure 4.10 Cost vs transfer size (large total data transfer)

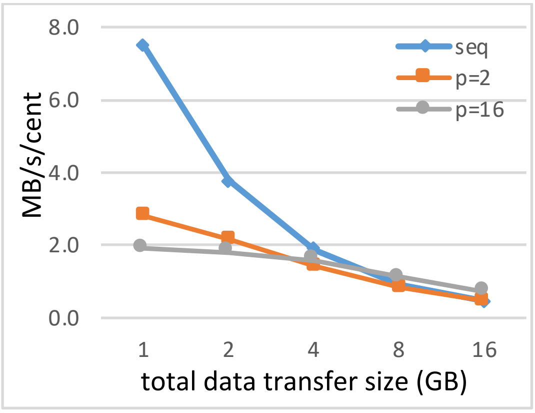

Figure 4.8 Throughput per cost vs transfer size (small total data transfer)

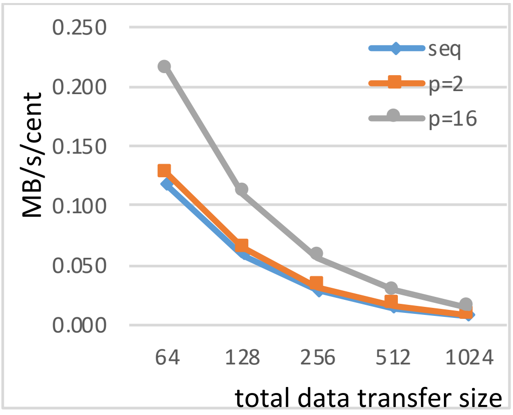

Figure 4.11 Throughput per cost vs transfer size (large total data transfer)

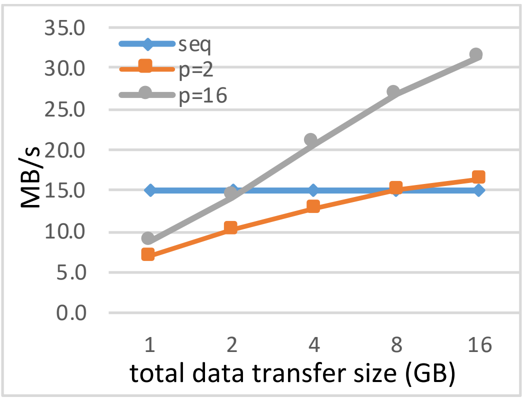

Figure 4.9 Throughput vs transfer size (small total data transfer)

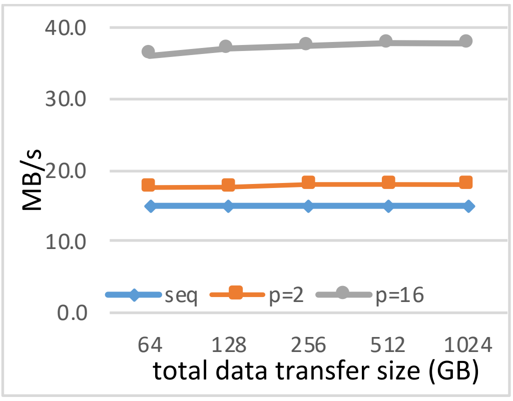

Figure 4.12 Throughput vs transfer size (large total data transfer)

As the total size of data transfer increases, both sequential and CPT cost increases linearly (Figure 4.7 and 4.10) and the throughput per cost decreases linearly (Figure 4.8 and 4.11). The reason for the decrease in throughput per cost is previously described in section 4.2. The increase in the cost of running the VM is much higher than the increase in network throughput of the VM. As seen in Figure 4.11, as the total amount of data transfer increase, CPT results in better throughput per cost as compared to sequential transfer, and the performance differences are even larger when the number of intermediate nodes is high.

As depicted in Figure 4.9 and 4.12, the throughput of sequential transfer remains the same as the only factor is the external network throughput. As for CPT, throughput increases until it reaches the maximum throughput depending on the number of intermediate nodes employed. Hence, it is observed that for large total amount of data transfer, the throughput for CPT seems constant.

4.2.3 Impact of the DC throughput ratio on CPT

As the CPT's core technique is to parallelize file transfer across multiple paths, the ratio of internal to external network throughput certainly affects the feasibility of the transfer. It is crucial that the internal network throughput (Vi) exceeds the external throughput (Ve) by a certain factor. Figure 4.13 depicts the graph of speedup versus the ratio of internal to external transfer speed.

In order to understand the impact of the ratio of Vi to Ve on the speedup, using the model, we set the Ve=10 mbps and increment the Vi. Figure 4.13 depicts the graph of speedup versus the ratio of internal to external transfer speed for transfer of 5GB. It can be observed that the speedup increases as the number of instance pairs is increased.

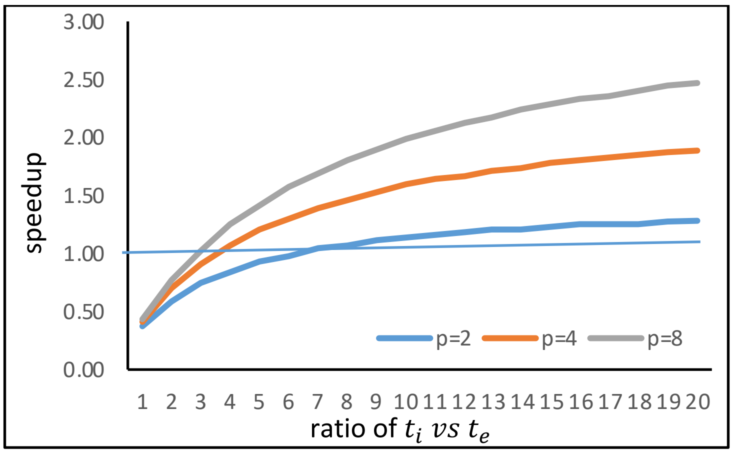

Figure 4.13 Speedup vs Ratio of Internal to External network throughput.

From the figure, it can be observed that the higher the ratio of internal to external network throughput, the higher the speedup. CPT transfer results in poorer performance when the ratio is below a certain threshold. The exact ratio depends on the transfer size and number of pairs of instances.

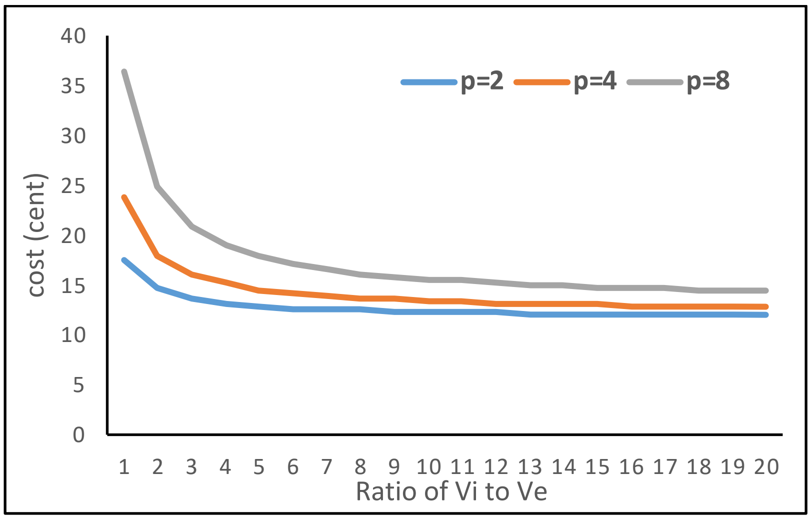

Figure 4.14 Cost vs Ratio of Internal to External transfer speed.

Figure 4.14 depicts the cost against the ratio of internal to external network throughput. As the ratio increases, the additional cost incurred for performing CPT decreases. This is because as the ratio increases, the time taken for the transfer is reduced and hence the cost incurred for CPT transfer reduces more than proportionately.

Hence, the higher the ratio of internal to external network throughput, the better it is for CPT both in terms of better performance and lower cost.

4.2.4 Impact of VM type on the cost of CPT

In the previous sections, the performance and cost are explored based on the AWS t2.medium machine type. As already described in section 4.1, the cost of running the VM is not proportional to the network throughput of the VM. In this section, the impact of using "network optimized VM" is studied.

The AWS EC2 type h1.2xlarge which has AWS official network rating of "up to 10 Gbps" is selected as a comparison. Here, the case is where the source and the destination node (h1.2xlarge) are of different VM type as the intermediate nodes (comparing h1.2xlarge vs t2.small). Table 4.5 and 4.6 are the values that are used.

Table 4.5 Numerical figure used in the models (h1.2xlarge)

Variable	Numerical figure used in the models
T_v	90 s
v_i	315 MB/s
v_e	30 MB/s

Table 4.6 Cost of AWS h1.2xlarge per second billing

Variable	Numerical figure used in the models
C_{eg}	\$0.02 / GB
C_{ig}	\$ 0
C_v	\$ 0.468 / hour (per second granularity)

Figure 4.15 depicts the throughput per cost ratio of CPT with different VM type as intermediate nodes. As seen, using h1.2xlarge as intermediate nodes results in smaller

throughput per cost ratio than t2.small. Hence, VM selection is important, to pick VM type favorable for improving the transfer but at the lowest cost.

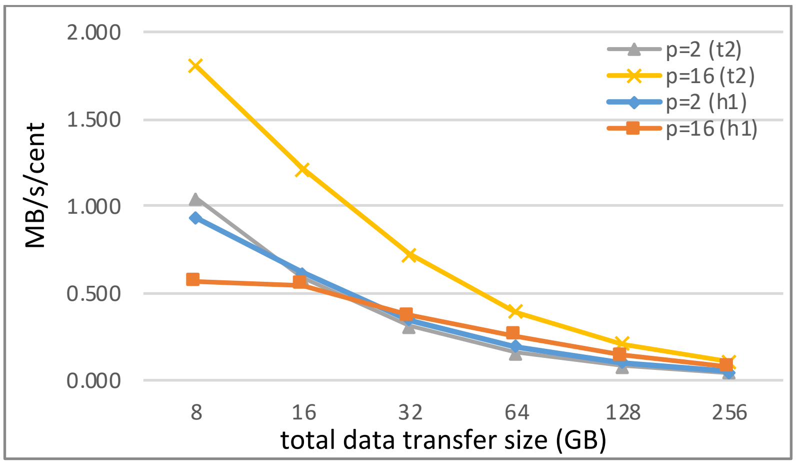

Figure 4.15 Throughput per cost vs transfer size for CPT of different VM type

4.2.5 Impact of pricing granularity on the cost of CPT

Depending on the pricing mechanism used by the service provider, charging granularity affects the overall price incurred. Pricing granularity varies across different cloud and time-to-time cloud service providers revise the scheme. Running *n* VMs for the duration of *m* minute (where $m < 60$) will incur *n* instance hour. Example, in hourly block charging, using the VM for 5 minutes will be charged the same price as using it for an hour.

In the following, we investigate the impact of pricing granularity on the overall cost of CPT transfer. Figure 4.16 depicts the total cost without egress charges vs data transfer size based on different pricing granularity. As seen, pricing granularity has a noticeable impact on the total cost of CPT.

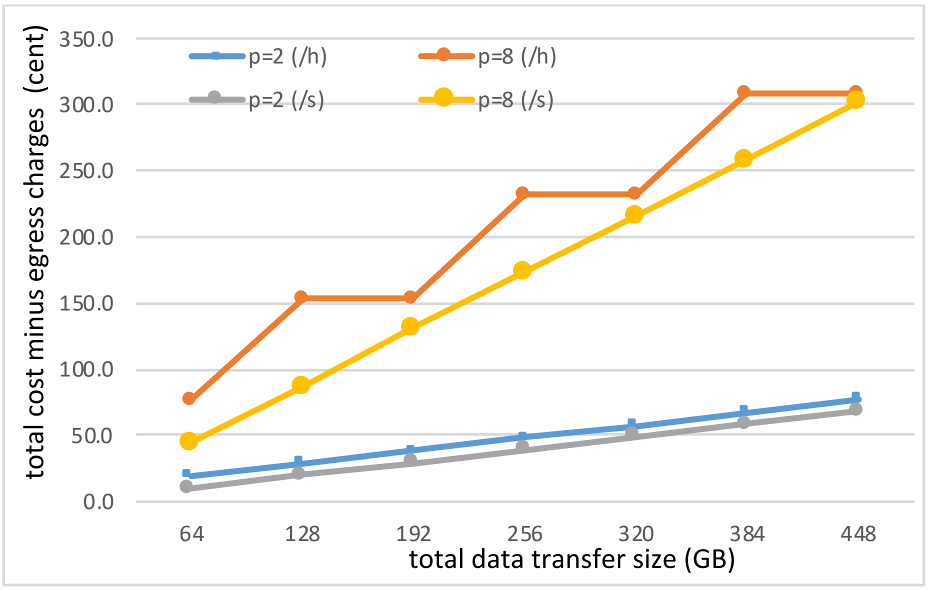

Figure 4.16 Total cost vs transfer size for different pricing granularity

As seen in the figure, the total cost difference between per hour and per second billing is not very significant when the number of intermediate nodes is lower. When the number of nodes deployed is higher in the case of per hour pricing granularity, more "wastage" is incurred as the cloud consumer is charged the full hour although CPT only utilized a fraction of the hour. For example, the cost addition of utilizing 1.5 hours but paying for 2 hours is more significant than utilizing 5.5 hours but paying 6 hours although it's just 30 minutes short in both cases.

This also explains the ladder-like cost incurred when p=8(/h). This is because the total time of transfer has increased, and the additional price difference to ceiling price (e.g. rounded to the next hour) is spread. Hence, the smaller the pricing granularity, the lower the overall cost of CPT transfer.

4.3 CPT Framework

Section 4.2 has enabled a good understanding of the impact of various factors towards the performance and cost of CPT transfer. In this section, an end-to-end process flow for CPT transfer is proposed, hereafter addressed as CPT framework. The proposed framework is cloud-agnostic and so, is generally applicable to typical public cloud platforms like AWS, Azure and Google Cloud.

Two techniques; VM-type selection and pre-testing are proposed to address the network throughput characteristic discussed in section 4.1.

4.3.1 VM-type selection

As described in the section 4.1, cost of running the machine does not linearly correlate to the network performance. Price of renting a VM type is based on a combination of several resources capacity such as CPU, memory and network throughput. Choosing a more powerful machine of double the price with double the CPU and memory capacity does not mean double the network capacity. This is an important area to explore as based on the models, the two factors that significantly affect the CPT is network throughput and price.

Here is an example of the impact of the deployment of two different VM type of similar network throughout but at a cost difference. VM1 – Small has much lesser resources (CPU, memory and storage) but equal network throughput as VM2 – Large. Deploying VM1 – Small as the intermediate nodes result in comparable performance but at a much lower cost. Hence, VM selection is important to ensure an effective CPT transfer.

The CPT is extended with the proposed feature to pick the most favorable machine type for the transfer. Based on actual data on network performance (a table with a list of VM type and its corresponding cost, inter-DC and intra-DC network bandwidth) is available, CPT will select the VM type with higher external network performance to cost ratio. This allows the overall transfer to be at a comparable performance but at a lower cost.

The network performance data should be provided up-front (e.g. taken from test result of previous transfer) before the transfer begins, otherwise, additional time is required for testing. As this information can be reused again across multiple transfers, in the remaining of this work we exclude the time taken for testing in the overall time taken for CPT transfer.

4.3.2 Pre-testing

Network performance varies even when picking machines of same class/type. This pattern is also observed in another researcher's work at (Persico et al., 2017). The work further describes killing machines that are performing below par. Inspired by this technique, we propose the following enhancement to CPT. Utilizing machines when its performance is on the upper side while killing machines that are not performing at an optimum level.

The CPT is designed to perform limited testing at the start of the transfer, we call this the pre-testing stage. If the transfer requires x number of intermediate nodes, the CPT will spawn 2x number of machines. Each of the machines will be tested for both its inter-DC and intra-DC network bandwidth, then, the machines at the bottom half the performance will be killed as it is not optimal.

Some may argue that more machines should be spawned for pre-testing. There is, of course, the possibility that more than half of the initial machines were not performing optimally. However, when conducting the test in section 4.1, we observed that less than half of the machines spawned are performing below average.

Besides, as the network throughput is relatively unchanged throughout the lifetime of the VM (see section 4.1), the models remain simple as network performance fluctuation do not need to be taken account of. Deciding the VM based on the early network throughput is sufficient.

The equation for CPT with pre-testing is as below:

$T_{CPT+} = T_{pt} + T_{CPT}$ (4.1)

If pre-testing is involved, the cost of the pre-testing stage is given as

$C_{pt} = T_{pt} \,.\, 2n \,.\, C_v$ Hence,

$C_{CPT+} = n \,.\, C_v \,(T_{CPT} + 2 \,.\, T_{pt}) + C_{sq}$ (4.2)

4.3.3 End-to-end process flow of CPT

The Figure 4.17 shows the CPT framework – flow chart for the events leading up to the CPT transfer.

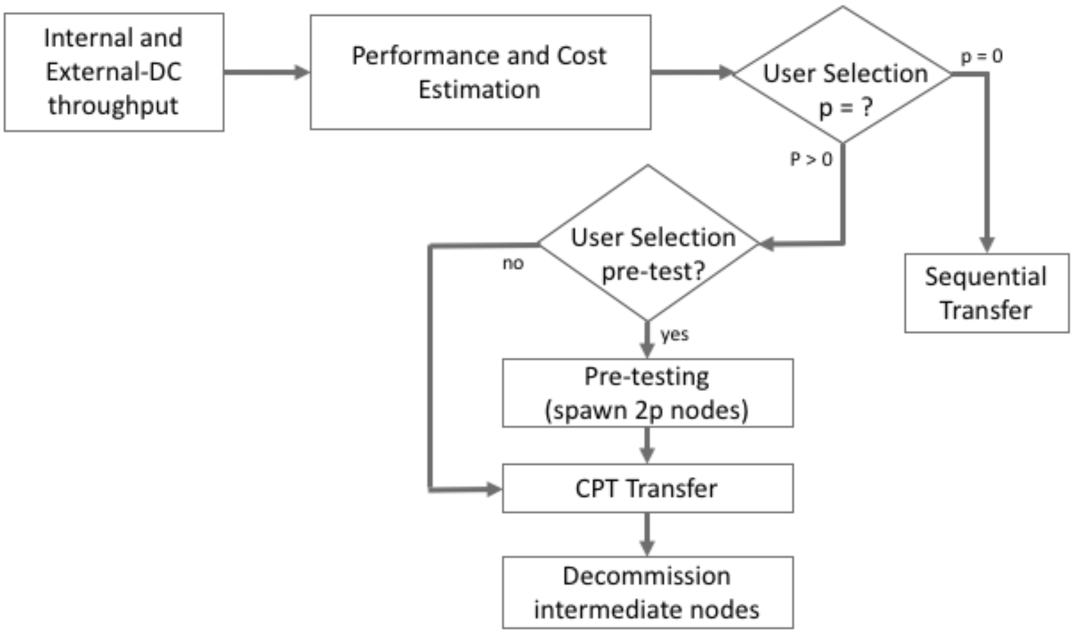
Internal and External-DC throughput
Performance and Cost Estimation
User Selection p = ?
p = 0
P > 0
Sequential Transfer
User Selection pre-test?
no
yes
Pre-testing (spawn 2p nodes)
CPT Transfer
Decommission intermediate nodes

Figure 4.17 Process flow leading to the CPT Transfer

At the very beginning, VMs cost and its respective internal and external DC throughput are to be provided. If not readily available, a separate test is performed. Then, based on the performance and cost estimation for different numbers of intermediate nodes, p, the user select the desired performance. Based on the decision, the transfer is done sequentially (i.e. a normal transfer without CPT) if p=0 or using CPT if p>0. Next, based on the performance estimation a recommendation is provided to the user if pre-testing is likely to yield benefit. If the transfer time is estimated to be short, pre-testing is not recommended as overhead is high. It is the users decision to go with the recommendation or overrule – taking the risk. Once pre-testing is performed (if needed), the CPT transfer begins. In the event any chunk failed the checksum, the chunk will be resent. If any intermediate node fails during the transfer, the failed node will be replaced with new intermediate node. CPT will resume transferring but there is a performance impact. Once the data transfer completed successfully, the intermediate nodes are decommissioned.

4.4 Conclusion

Based on the model analysis that have been covered in this chapter, the table 4.7 provides a summary of the variables' impact towards CPT.

Table 4.7 Respective variables' impact towards CPT

No.	Variable	Impact towards CPT	Control
1	Internal transfer throughput, v_i	The larger the better. Besides, the higher the ratio of v_i to v_e the better.	Dynamic, varies according to VM type and CSP
2	External transfer throughput, v_e		
3	VM setup time, T_v	The smaller the better.	
4	Ingress traffic cost, C_{ig}	Same as sequential transfer. The smaller the better for consumer.	Usually zero, set by CSP
5	Egress traffic cost, C_{eg}		Usually fixed, set by CSP
6	Cost of VM per unit time, C_v	The smaller the better.	
7	Pricing Granularity	The smaller the charging block, the lower the total cost.	

This first part of the chapter showed that the cost of renting the VM does not linearly correlate to the network performance. Two important lessons are learned from the network throughput test. Firstly, the average network throughput of a VM varies across different occasion. Secondly, no significant fluctuation to network throughput throughout the lifetime of the VM.

Then, from the model analysis, we see that CPT with pipelining and network data piping outperforms CPT without both the optimizations. Hence the optimized CPT will be implemented and experimented in the subsequent chapters. It is important to note that the ratio of Vi to Ve is important as it has a large impact on the speedup of the CPT. If the internal network throughput is not sufficiently more than the external i.e. ratio is low, performing CPT transfer will result in worse performance than sequential transfer. Assuming that cost is not a constraint, the maximum speedup is achieved when the ratio of internal to external network throughput approaches infinity and the number of intermediate nodes deployed approaches infinity.

However, with the cost model, it shows that the efficiency (throughput per unit cost) will diminish over time. The cost models also showed that the smaller the pricing granularity, the cheaper the overall cost of CPT with performance being equal.

Finally, the CPT framework is proposed. The end-to-end process involving preparation work, execution of CPT and post-CPT steps are described. Two techniques; VM-type selection and pre-testing were proposed to optimize the performance and cost.

In a nutshell, corresponding to the second research objectives, this chapter has not only identified the influencing factors of implementing CPT model in the public cloud environment, but also provided enhancements which are critical to improving the effectiveness of CPT.

CHAPTER 5

SYSTEM IMPLEMENTATION

In this chapter, the implementation of each component and process which makes up the complete proposed CPT framework using the AWS cloud. In addition to that, the implementation of CPT for data transfer between Hadoop's HDFS cluster is also described. The implementation of CPT for data transfer between Hadoop's HDFS cluster is described here because the proposed CPT will be compared against Hadoop's DistCp in the subsequent chapter.

5.1 CPT Implementation

As the model in section 4.2 has shown that optimized CPT the best performance, we implement this version of CPT for the remaining of our work. The implementation is done on AWS EC2 to prove the concept of the proposed CPT.

All the scripts are written in Perl due to the out-of-the-box support on most Linux distros and compatibility on Windows-based machine – allowing portability across different platforms. For our implementation on AWS, all the EC2 (i.e. VMs) are running Ubuntu Linux 16.10. Password-less SSH connections are set up so that the VMs can authenticate and communicate with each other and perform the data transfer.

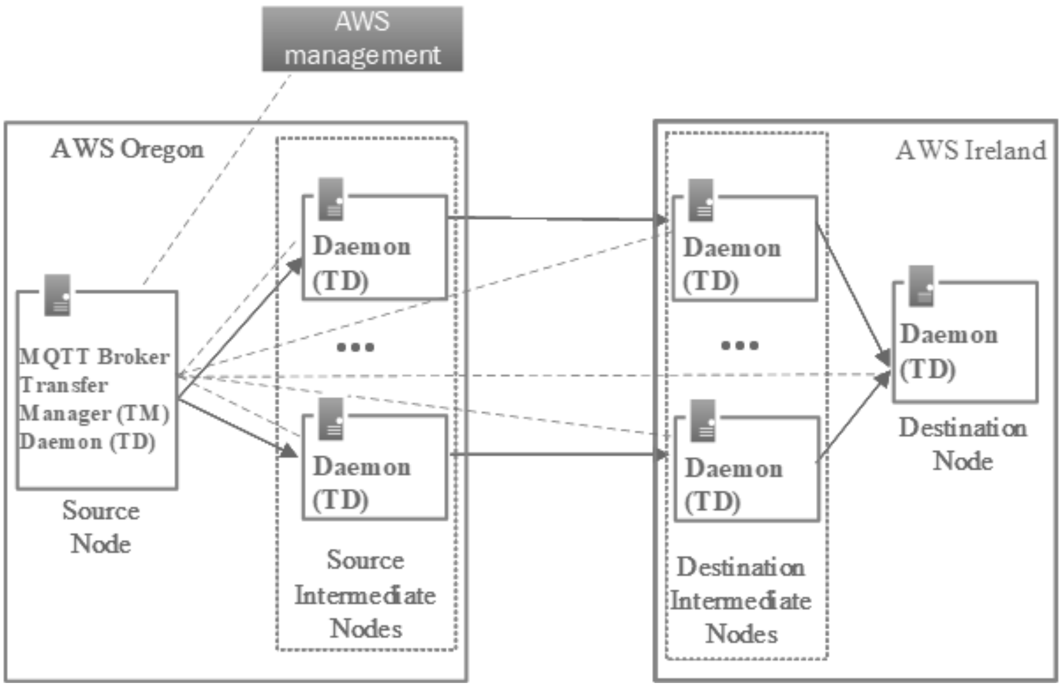

Figure 5.1 Control and data flow of CPT implementation on AWS

Figure 5.1 depicts the implementation involving two different AWS region. The solid lines represent the data flow (i.e. data transfer) while dotted lines denote control/command channel (two ways). All the forecast and decisions are made by the Transfer Manager (TM). All control communication is made by the daemons via the centralized Message Queue Telemetric Transport (MQTT) broker running as part of the TM. The Transfer Daemons (TD) are responsible for initiating the transfer based on received instructions and periodically reporting to the TM. This allows for real-time monitoring of the transfer and reactions to failure as soon as possible (e.g. one of the intermediate nodes failed or chunk corrupted). The TM is also responsible for communicating with the cloud APIs for commissioning and decommissioning the VMs.

In summary, the CPT consists of the TM and TD. TM coordinates the entire transfer while TD merely execute the instructions and reports the progress of transferring the respective chunks. The TM consist of the following components:

- Initial VM-type throughput tester – perform intra-DC and inter-DC network bandwidth test for the different VM types, so that the most suitable VM type can be decided.
- VM-type selector – selecting the most suitable VM type for use as intermediate nodes.
- Speedup and Cost Estimator – estimation of the speedup and financial cost of CPT based on the models.
- Pre-tester – perform network bandwidth test (all VM of the same type) and only use top-performing VMs as intermediate nodes.
- CPT coordinator – initiating and keeping track of the chunk transfers between all the involved nodes

The roles and implementation of the components are discussed in the subsequent sections.

5.1.1 DC Throughput & VM-type Selection

As described in the section 4.3 CPT framework, before the CPT transfer even begin, specific information on the VM startup time, internal and external DC throughput has to be provided. Table 5.1 shown below, is the information needed for an optimized transfer.

Table 5.1 List of information collected during the network throughput test

No.	Item	Example
1	VM Type	m5.large
2	Internal Throughput (Mbps), v_i	250
3	External Throughput (Mbps), v_e	80
4	Price ($/hour)	0.78
5	Startup time (s), T_v	55

Item 1 and 4 are an example of information related to a cloud instance provided by the cloud provider (https://pricing.us-east-1.amazonaws.com/offers/v1.0/aws/index.json), which is query-able via API. As there may be a price difference of running the same VM type in source and destination, the average of the two is calculated since our CPT implementation is performing 1-to-1 mapping.

As for item 2 and 3, the cloud consumer has to perform the necessary test. If this information is readily available, testing can be performed before the transfer begin. The data can be collected and used for future transfer. As time goes by, as more and more transfers are performed, the average will be closer to reality.

The pseudocode in Table 5.2 describes the algorithm used to populate the data for a transfer for the first time. The input is a 2-dimensional array with VM type and price obtained from the cloud provider. The procedure then spawns the VMs and perform the iPerf test accordingly. The output is the VM type with its price and corresponding internal throughput, external throughput and VM startup time collected from the test.

Table 5.2 Pseudocode for benchmarking VM network performance

```
Input:  2D Array (VM Type and its corresponding Price)
        Duration of test to perform, x
Output: 2D Array (VM Type and its corresponding Price, Ve, Vi, Tv)

Procedure:
FOR each VM type in Array
        Spawn 2 VM (VMs1, VMs2) of VM type in source DC
        Spawn 1 VM (VMd1) of VM type in destination DC
WHILE VMs are NOT ready (WAIT)
Record VM startup time and update Array
FOR each VM type in Array
        Connect to VMs1 and initiate network throughput test to VMs2 for x
                duration
WHILE x duration NOT elapsed (WAIT)
FOR each VM type in Array
        Connect to VMs1, retrieve result (internal throughput) and update Array
        Connect to VMs1 and initiate network throughput test to VMd1 for x
                duration
```

```
    Decommission VMs2
WHILE x duration NOT elapsed (WAIT)
FOR each VM type in Array
    Connect to VMs1, retrieve result (external throughput) and update Array
    Decommission VMs1 and VMd1
RETURN Array
```

Table 5.3 shows the example starting array where only the VM type and Price is listed. Table 5.4 shows the updated example array after all the necessary information needed for forecasting the CPT transfer is furnished.

Table 5.3 Starting Input Array prior to network throughput test

VM Type	Price
t2.micro	0.0116
t2.small	0.0230
t2.medium	0.0464
…	…

Table 5.4 Array updated with VM information necessary for CPT transfer

VM Type	Price	Ve	Vi	Tv
t2.micro	0.0116	107	718	83
t2.small	0.0230	124	724	82
t2.medium	0.0464	146	721	85
…	…	…	…	…

As discussed in section 4.2.3, VM type with the best external throughput to cost ratio is used, provided that the internal network throughput exceeds the external throughput of the source and destination VM. The pseudocode for VM selection is as below, continuing from above:

Table 5.5 Pseudocode for VM selection

```
Input:  2D Array (VM Type and its corresponding Price, Ve, Vi, Tv)
        VM type of source and destination machine
Output: The most suitable VM Type (for intermediate nodes)

Procedure:
FOR each VM type in Array
    COMPUTE cost per unit internal throughput and update table
```

```
SORT table ascending cost per unit internal throughput
FOR each VM type in list of VM type
    IF VM type internal throughput > 7 x source machine's internal throughput
        RETURN VM type
RETURN "no suitable VM"
```

5.1.2 Speedup and Cost Estimation & User preference

In order for the user to decide if CPT should be performed and how many intermediate nodes should be deployed, an estimated time and cost has to be calculated. The calculation is performed based on the 2 models derived in chapter 3, with all the necessary information obtained from the previous stages (from section 5.1.1). Below is the pseudocode of getting the estimated time and cost for different values of *p*, the number of pairs of intermediate nodes.

Table 5.6 Pseudocode for estimating CPT transfer time and cost

```
Input:  2D Array (VM Type and its corresponding Cv, Ve, Vi, Tv)
        Egress Cost, Ceg
        Total data transfer size, s
        Number of intermediate nodes for CPT transfer, p
        VM type of source and destination machine
Output: Estimated time and cost

Procedure:
WHILE
    IF p=0
        Estimated Time, T <= s / Ve of source/destination machine
        Estimated Cost, C <= s * Ceg
    ELSE IF p>0
        Estimated Time, T <= Tv + 2 * s / Vi  + s * p / Ve
        Estimated Cost, C <= (s * Ceg) + (T * 2 * p * Cv)
    End if
end loop
RETURN Array of T, C
```

Below is the example outputs showing all the estimated time and cost depending on the number of intermediate nodes. Speedup refers to the sequential transfer time divided by the respective CPT transfer time.

Table 5.7 Example of array of performance and cost estimation

p	Estimated Time (s)	Estimated Cost ($)	Speedup
0 (no CPT)	545	0.160	-
1	713	0.219	0.764
2	521	0.203	1.046
3	493	0.201	1.105
…	…	…	…

Based on the table the user selects the number of pairs of intermediate nodes for the CPT transfer. The exact number of intermediate nodes that will be spawned is described in the next section.

5.1.3 Pre-testing & VM spawning

The number of VM to be spawned depends on 2 decision made by the end user. Firstly, the desired number of pairs of intermediate nodes as described in the previous section. Secondly, whether pre-testing is required. When the pre-testing is employed, the TM is responsible for spawning twice the number of intermediate nodes necessary, otherwise, the exact number is spawned. In the former, the bottom half performing VM will be destroyed after the pre-testing phase.

Pseudocode for spawning VMs with pre-testing stage is as shown in table below.

Table 5.8 Pseudocode for spawning VMs with pre-testing

```
Input:  VM Type
        Source DC
        Destination DC
        Number of intermediate nodes for CPT transfer, p
        Duration of pre-testing, x
Output: List of intermediate nodes (p VMs spawned)
```

```
Procedure:
Initialize 2D Array (VM ID, Ve, Vi)
Spawn 2p VM (VMsx) of VM type in source DC
Spawn 2p VM (VMdx) of VM type in destination DC
FOR each i in 2p iterations
        Connect to VMsi and initiate network throughput test to VMdi for x
                duration
WHILE x duration NOT elapsed
FOR each i in 2p iterations
        Connect to VMs1, retrieve result (external throughput) and update table
SORT table ascending throughput
FOR each i in p iterations
        Decommission VMsi and VMdi
```

If no pre-testing, the exact number of VMs is spawned without need any test. The pseudocode as shown below:

Table 5.9 Pseudocode for spawning VMs without pre-testing

```
Input:  VM Type
        Source DC
        Destination DC
        Number of intermediate nodes for CPT transfer, p
Output: List of intermediate nodes (p VMs spawned)

Procedure:
Spawn p VM (VMsx) of VM type in source DC
Spawn p VM (VMdx) of VM type in destination DC
```

5.1.4 CPT

The transfer coordinator virtually splits the file(s) into arbitrary number of equal sized chunks, and the total number of chunks must be more than the number of pairs of intermediate nodes. In our implementation, we set the number of chunk to 3 times the number of pairs of intermediate nodes. The study of the impact of varying the number of chunks is beyond the scope of this work. Once the virtual splitting is done, the transfer from the source node to source intermediate nodes is initiated. The transfer daemon in the source node executes the transfer.

The transfer daemon in each of the nodes executes the transfer; monitor and restarts the transfer if there is any failure and initiates the next transfer based on instructions from the transfer coordinator.

The daemon in the source intermediate nodes monitors the transfer between source intermediate nodes and destination intermediate nodes. Once a particular chunk is received in the destination intermediate node, the daemon in the destination node will immediately relay the transfer to the destination node.

The daemon in the destination node will stitch all the chunks together to re-form the original file(s). Then, the daemon gets the checksum of the file(s) and informs the transfer coordinator that the transfer is completed. If the final checksum matches the CPT transfer is considered done. The baton is handed back to the Transfer Manager to decommission the intermediate nodes.

Both the transfer coordinator and transfer daemons are implemented with an asynchronous methodology. The event-driven architecture allows the immediate reaction of events which may happen in a span of short time. The pseudocode of the transfer coordinator is as below.

Table 5.10 Pseudocode of the transfer coordinator

```
Input:  Source DC
Destination DC
        List of intermediate nodes
        List of file(s)
        N=3p
Output: NULL

Procedure:
PUT all source int. Node ID to srcint queue
PUT all destination int. Node ID to dstint queue
Virtually split file(s) into N size chunk
PUT chunk ID into src_send queue
Set count <= 0
```

```
WHILE
    IF src_send queue not empty
        free srcint node <= SHIFT srcint queue
        chunk <= SHIFT src_send queue
        Init transfer of chunk from src to free srcint node
    ELSE IF count eq. N AND get checksum match
        Transfer completed
        Return
    ELSE Restart entire transfer
    end if
end loop

EVENT: src node sent to src int. node completed
    free dstint node <= SHIFT dstint node
    inform src int. node to send to free dstint node
    IF src_send queue NOT empty AND srcint NOT empty
        free srcint node <= SHIFT srcint queue
        chunk <= SHIFT src_send queue
        Init transfer of chunk from src to free srcint node
end if

EVENT: src int. sent to dst int. node completed
    lookup hash table and inform dst int. node to send to dst node
    put src int. node into vacant queue

EVENT: dst int. node sent to dst node completed
    PUT dst int. node into dstint queue
    count <= count + 1
```

The pseudocode of the transfer daemon is as below:

Table 5.11 pseudocode of the transfer daemon

```
Input: Node ID (for identification and reporting)
Output: NULL

Procedure:
WAIT for instruction

IF receive instruction to Prepare
    LISTEN on network port and prepare for incoming chunk
end if

IF receive instruction to Start
    START chunk transfer to next node
end if
```

```
WHILE (every 10 seconds)
        REPORT progress to transfer coordinator
        IF chunk transfer completed
                Restart daemon and WAIT for instruction
        end if
end loop
```

The table below shows an example 2-dimensional hash (lookup table) keeping track of the virtual chunks and its transfer status. The "start KB" and "end KB" marks the beginning and end of each chunk. The "stage" indicates the stage at which the chunk is currently at; 1 → in-flight between source to intermediate node, 2 → in-flight between source int. node to destination int. node, 3 → in-flight destination int. node to destination node. Status indicates the percentage transferred for the particular chunk in the stage. "Start time" marks the starting time of the stage – for accountability purpose.

Table 5.12 2D Array storing virtual chunk information and transfer status

ID	File Name	Start KB	End KB	Stage	Status (%)	Start Time (Epoch)
0	/file01	0	920000	2	5	1527508391
1	/file01	920001	1840000	1	70	1527508512
2	/file01	1840001	2600000	-	-	-
3	/file02	0	800000	-	-	-

The Table 5.13 below shows an example 2-dimensional hash keeping track of all the intermediate nodes, source node and destination node. The ID is unique for each VM and the naming convention indicates its role. Internal IPs are used for intra-DC transfer while external IPs are for inter-DC (WAN) transfer. The chunk column keeps tracks of the respective chunk ID that has been sent and are in progress of sending by the respective VM.

Table 5.13 2D Array storing information of intermediate nodes

Node ID	Int. IP Address	Ext. IP Address	Chunk (out)
src	172.168.1.2	52.12.12.20	0, 1, 2, 3
srcint01	172.168.1.20	53.230.2.11	0, 2
srcint02	172.168.1.15	33.32.23.230	1, 3
dstint01	192.168.2.120	52.0.2.110	0
dstint02	192.168.2.10	55.12.120.234	1
dst	192.168.2.99	32.45.23.120	NA

5.2 Implementation of CPT on Hadoop's HDFS

In this section, the adaption and implementation of CPT for data transfer between Hadoop Clusters are described here. Instead of an invasive approach where a redesign or tempering of the cluster-ware solution is needed, the technique describes the adaptation needed by CPT transfer with minimal changes needed. This is not the only way to utilize the techniques outlined in this work, but it's the simplest implementation. This is as a continuation of our work and a proof that the proposed solution is flexible enough for application in many domains within the scope of cloud-to-cloud data transfer.

The HDFS is the default file system in Hadoop. It typically consists of a Namenode (NN) which provides the namespace and multiple Datanodes (DN) which provides the distributed storage element. All access to the file has to be first queried via the NN which stores the metadata. It is not unusual that the NN is also designated as the Hadoop Master Node while the DN is the Hadoop Worker Nodes.

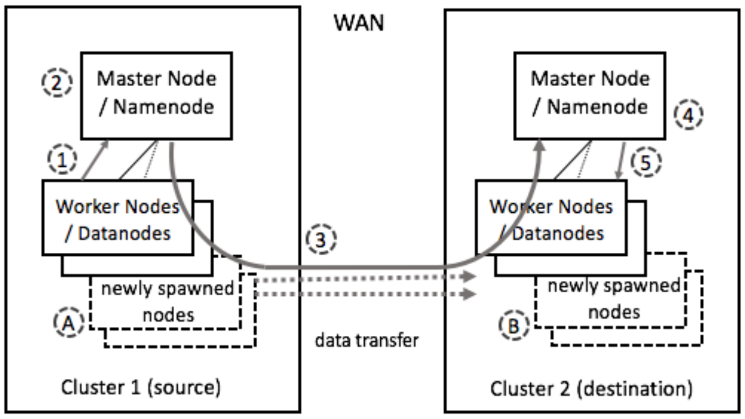

Figure 5.2 Adapting CPT for HDFS cluster transfer

Figure 5.2 depicts the brief component diagram of two typical HDFS cluster implemented with CPT. The solid lines with arrow depict the data flow of transferring from source cluster to the destination cluster using the adapted CPT framework. The original worker nodes of the cluster also serve as intermediate nodes. As the CPT is designed to work with transferring of file on native OS filesystem, usage in the context of other cluster-ware requires additional steps as described below in sequence and in reference to the diagram:

A. If needed, spawning of intermediate nodes (additional nodes not already part of the original cluster) on both source and destination DCs.

1. Data is exported from source HDFS cluster into the local OS filesystem of the source cluster's NN.
2. Depending on the size distribution of the files, it may have to be bundled (tar) into a single file residing on the source cluster, so that it can be broken into equally sized chunk for CPT transfer.
3. The file is transferred to the destination cluster's NN using CPT. On top of the additional nodes spawned in step A, all the DNs of each of the cluster respectively are utilized as intermediate nodes to aid the transfer.

4. On the destination cluster's NN local OS filesystem, the transferred files are then reconstructed to give the original files.
5. The files are imported into the destination HDFS cluster.

B. Decommissioning the intermediate nodes after the transfer is completed.

Realistically speaking, from performance point-of-view, it is not the best idea to have the data transferred twice internally, first in step 1 and 5 and then second time as part of step 3. However, such approach is proposed in order to keep the complexity (of integrating our parallel transfer for cluster-ware) low otherwise additional effort is required to keep track of the files individually which is not the focus of this study.

$$\text{Transfer time, } T_{HCPT} = T_E + T_{CPT} + T_I \qquad (5.1)$$

Table 5.14 Notations for time components of adapting CPT transfer for HDFS

Notation	Description
T_{HCPT}	Adapted CPT for cluster-ware
T_E	Time taken for data export to source master node OS filesystem
T_{CPT}	CPT transfer time from src. to dst clusters
T_I	Time taken for data import to destination cluster specialized filesystem

In short, the high-level concept and implementation of performing CPT transfer between the Hadoop Cluster are the same. There are only 3 differences, that are:

1. In addition to spawning new VMs to serve as intermediate nodes, existing worker nodes in the HDFS cluster may also serve as intermediate nodes for the CPT transfer.

2. Additional step before the CPT transfer to export and import the files from the HDFS filesystem to the local OS filesystem and vice versa respectively. Figure 5.3 shows the example timeline of CPT transfer for HDFS cluster.
3. Equation 5.1 (section 5.2, above) instead of Equation 3.4 (section 3.3.3) is used for the estimation of transfer time.

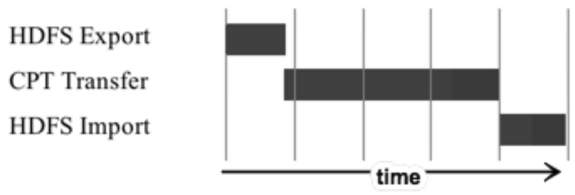

Figure 5.3 Example timeline of CPT transfer between HDFS cluster.

5.3 Discussion

As the framework is cloud-agnostic, the implementation described in this chapter can also be implemented on other major IaaS cloud platforms such as Azure and Google Cloud. For example, AWS provides a feature called "placement group" that allow several resources to be placed on the same physical host (proximity) or somewhere nearby. The implementation did not make use of such feature as it's cloud-specific, and hence avoided. Should we have used this feature, the result would have been [positively] different.

5.4 Conclusion

In summary, this chapter showed the detailed implementation of CPT on the AWS public cloud platform. The chapter has also demonstrated a quick and potentially effective adoption of the CPT transfer for transfer across Hadoop HDFS cluster.

CHAPTER 6

EXPERIMENT RESULT

This chapter discusses the experimental result of implementing and testing CPT on the AWS platform. This chapter is divided into 2 parts, first, compares CPT with sequential transfer, and second, evaluates CPT for HDFS cluster.

6.1 Comparing CPT transfer time to sequential transfer

This section describes the experimental setup and results for CPT comparing to sequential transfer. The experiment results for CPT with (all except in section 6.1.3) and without (only in section 6.1.3) pre-testing is also described.

6.1.1 Experimental Setup

All the experiment is conducted on AWS infrastructure located in two locations: US Oregon and EU Ireland. The US region is the data source while the EU is the destination in which we want to transfer the data to. In this section, all the intermediate, source and destination nodes use are type t2.medium (2vCPU, 4GiB memory, 50GB network disk). The Ubuntu Linux VM image included the pre-configured SSH authentication so that each nodes can communicate between each other without needing additional configuration, also known as “password-less authentication”.

The test data to be transferred are Linux ISO images downloaded from repositories (public domain) and truncated to the precise size needed. The reason for doing this instead of generating arbitrary files is to save time preparing files that cannot be further compressed. In the next few sub-sections, the result of CPT transfer is shown and analysed.

6.1.2 Performance of CPT

As the model in the previous chapter has shown the promising result for CPT, an understanding of whether the result can also be achieved in real-world conditions is needed. Hence, we first tested out the CPT transfer of a single file of varying sizes and with 2, 3 and 4 intermediate nodes. Figure 6.1 depicts the result in a graph of total time taken to complete the transfer against the total size transferred.

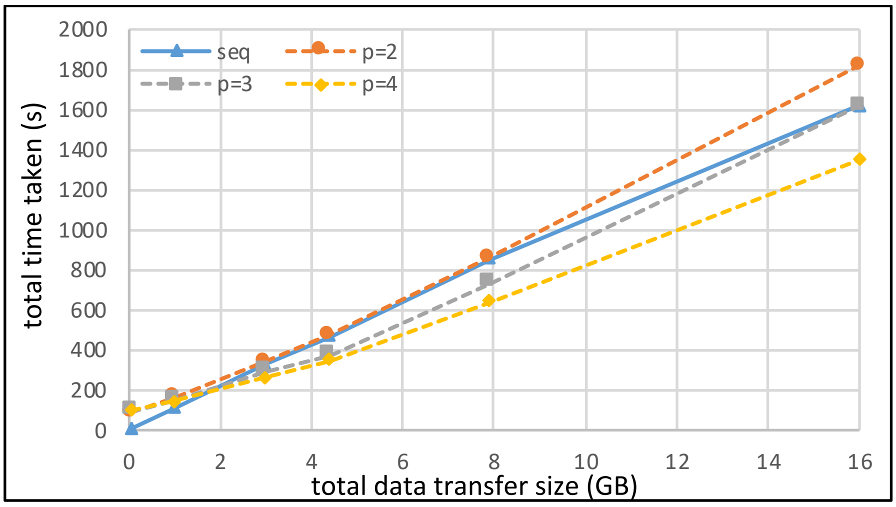

Figure 6.1 Total time taken of CPT as compared to Sequential transfer (lower transfer time better)

CPT using 3 or less pairs of intermediate nodes resulted in poorer performance compared to sequential transfer. CPT using up to 4 pairs of intermediate nodes begins to perform better than sequential transfer when the data transfer size exceeds 3GB. CPT with 2 pairs of intermediate nodes result in longer transfer time for transfer of between 8 and 16GB. As transfer of 16GB is reached, even CPT with 3 pairs of intermediate nodes. CPT with 4 pairs of intermediate node results in shorter transfer time compared to sequential transfer once the total transfer size exceeds 2GB.

It is observed that increasing the number of pairs of intermediate nodes results in a decreased in the total time taken to complete the transfer. This is expected as increasing the number of intermediate nodes increases the aggregate bandwidth.

In order to better quantify the performance of CPT compared to sequential transfer, Figure 6.2 depicts the result in a graph of speedup against transfer size. The figure also shows both the result from actual experiments compared to forecasted result. Forecasted result is calculated from the model as part of the CPT framework.

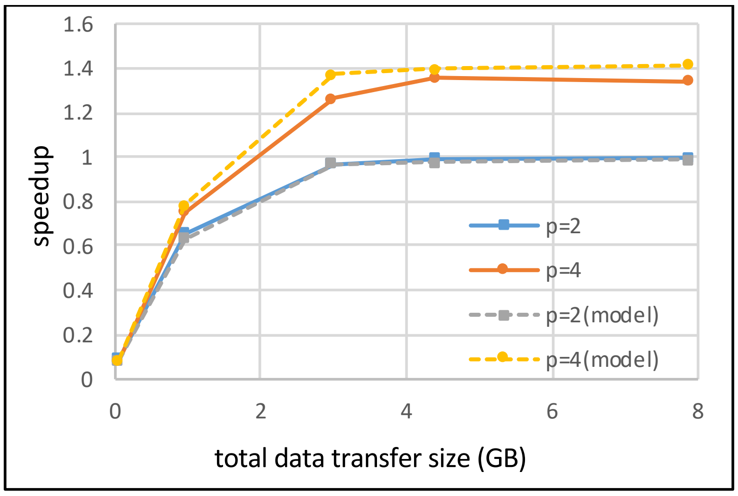

Figure 6.2 Speedup for CPT (experiment vs model)

Firstly, it can be seen that the achieved speedup did not differ much compared to our model. This is good as the model is critical in forecasting the time taken for the transfer based on the known factors (i.e. VM startup time, internal and external throughput).

Secondly, the general observation from the experiment is consistent to the models – speedup is low for small total data size transfer, but increases as the total data size transfer is increased. For an 8GB transfer using 4 pairs of intermediate nodes, we are able to achieve a speedup of roughly 1.4x. This also translates to 25% less time compared to sequential transfer. As expected, for any transfer below 8GB using 2 or 4 pairs of intermediate nodes, speedup cannot be achieved.

It is clear from the figure that out of the 2 cases, speedup > 1 is only archived when the number of pairs of intermediate nodes is 4 and with the two optimization - pipelining and network data piping.

Summary of the general observations are as follow:

1. Speedup is low for transfer of small total size transfer (i.e. 1GB in our experiments). Benefits of the CPT is more significant for larger total size transfer. From our experiment, the data size has to be larger than 4GB.
2. Increasing the number of pairs of intermediate nodes results in increased performance (i.e. higher speedup). However, the performance improvement has a diminishing return, where increasing the number of intermediate node results in lesser improvement gain than the previous addition.

6.1.3 CPT with pre-testing

In this section, CPT with 60 seconds of pre-testing is tested. A 1:1 ratio - 30 seconds of testing intra-DC and 30 seconds inter-DC bandwidth is used. At the end of the minute, the intermediate nodes are ranked (first based on the inter-DC, then the ratio of intra-DC to inter-DC), and the bottom half performing VMs are discarded. The remaining half is used for the CPT transfer. The experiment is repeated 3 times a day at different time of the day for over the course of 4 continuous days (Feb 2018) – collecting a total of 12 set of results.

Figure 6.3 and 6.4 depicts the result – average, minimum and maximum. Most importantly, it can be seen that the performance variability when pre-testing is lower, resulting in more consistent transfer time. For an 8GB transfer, no pre-testing results may differ up to 15% between different attempts while pre-testing differs by less than 5%.

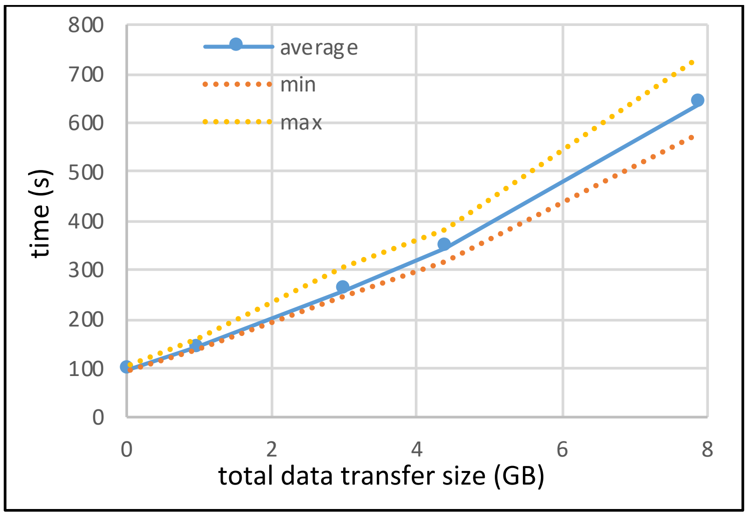

Figure 6.3 The upper and lower bound of CPT (p=4) performance without pre-testing.

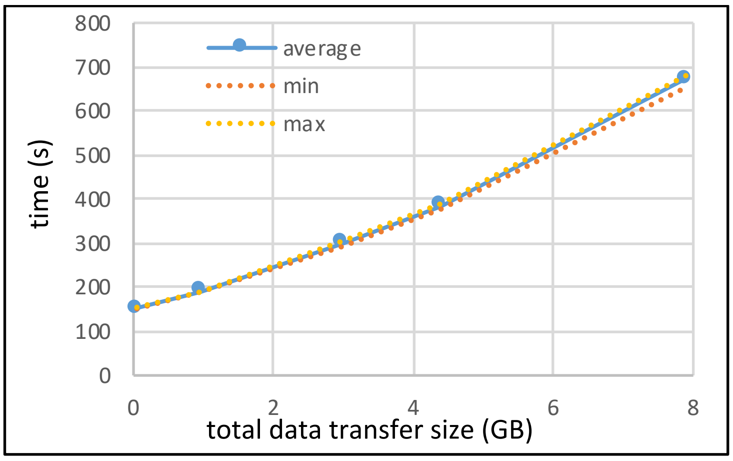

Figure 6.4 The upper and lower bound of CPT (p=4) performance with pre-testing.

Despite the advantage of pre-testing, pre-testing comes with a 60 seconds overhead – making it not suitable for smaller transfer as the overhead is a huge proportion of the total transfer time. As the transfer grows larger (i.e. more than 8GB in our experiment), the overhead of pre-testing becomes insignificant and pre-testing becomes more worthwhile as results in performance improvement.

6.2 Comparing CPT for HDFS with DistCp

This section describes the experiment setup and result for CPT on HDFS cluster comparing to DistCp transfer. DistCp is the state-of-the-art for parallel transferring data between HDFS clusters and CPT is adapted to benchmark against DistCp.

6.2.1 Experimental Setup

The setup on the AWS public cloud platform for CPT + HDFS experiment is slightly different from the previous subsection due to the specific requirements of HDFS. Two separate but identical clusters are set up with the support of Hortonworks Data Cloud for AWS ("Hortonworks Data Cloud for AWS," n.d.), one in US Oregon region, and one in the EU Ireland region.

Each cluster consists of 3 nodes, 1 Namenode (NN, also acts as the Hadoop Masternode) and 2 Datanodes (DN, also acts as the Hadoop Workernode). All nodes are type m4.large (2vCPU, 8GiB Memory) and 2 disks of 50GB each. Replication factor is set at 2 so each DN has a full copy of the data. Except for the replication factor, the Hadoop cluster has default parameters from Hortonworks. All the nodes are configured with public IPs which is necessary for both DistCp and CPT. Besides, reverse DNS lookup is enabled for DistCp to work using the native HDFS protocol. Let's consider this as the base setup.

Each of the experiment is performed by transferring an increasing number of 256MB and 512MB files separately with DistCp and CPT. Example, 12x 512MB means 12 files of 512MB each, giving a total transfer of 6GB. The files are generated using the Linux dd tool ("dd(1): convert/copy file - Linux man page," n.d.) and are imported into the HDFS cluster for the experiments.

In order to avoid conflict and resources contention which impacts the accuracy of the experiments, all the virtual machines are exclusively allocated to our experiment. The Hadoop cluster is not running any jobs not pertaining to our experiment.

The first experiment is conducted on the base setup – no spawning of intermediate nodes besides the original 2 Datanodes. The second experiment is with spawning 2 additional intermediate nodes (m4.large) per cluster. The third experiment is similar to the second except using t2.micro (i.e. a cheaper VM) with CPT only because DistCp comes together with HDFS does not have enough resources to be deployed in t2.micro instances that only has 1 vCPU and 1 GB Memory which is far below the requirement to run Hadoop.

6.2.2 Results of CPT and DistCp on HDFS cluster

The first experiment, Figure 6.5 shows the transfer time between 2 HDFS cluster using DistCp and CPT when no additional intermediate nodes are spawned. As shown, the time taken for CPT is slightly better than DistCp even with additional time is spent on additional stages introduced to make CPT work with HDFS. It can be observed that time taken for DistCp to complete the transfer is in "steps". This is because the Hadoop cluster can only execute a pre-defined number (e.g. 7 for the default configuration on m4.large EC2 type) of map task at a time. Unlike the DistCp, time taken for CPT increases almost linearly as the total transferred file size increases.

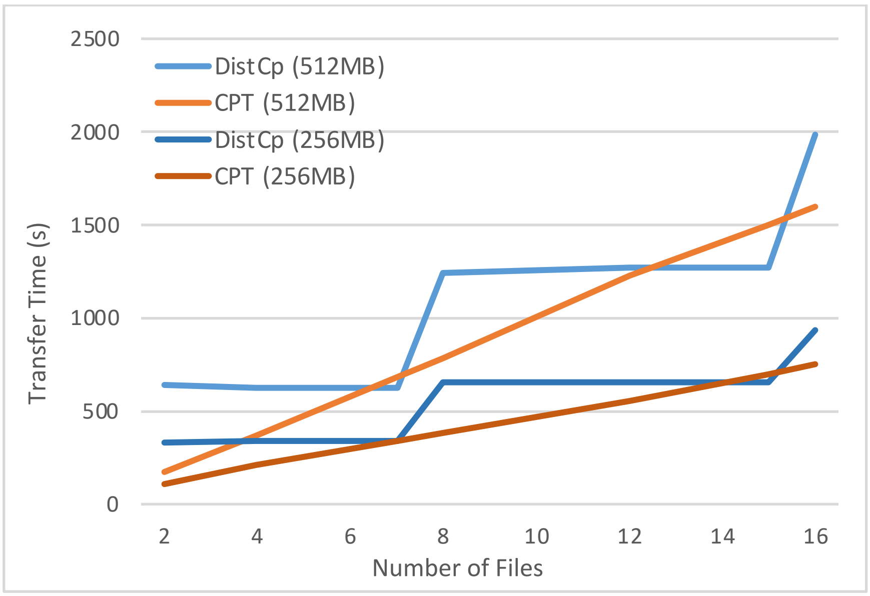

Figure 6.5 Transfer time of DistCp and CPT for various total size.

As a consequence of adapting CPT for HDFS, additional time is required for the stages HDFS import and export (Figure 6.6). However, for the case of 2x 512MB transfer, the time spent on this stages are not overwhelming significant, hence the CPT is able to provide equal performance as DistCp.

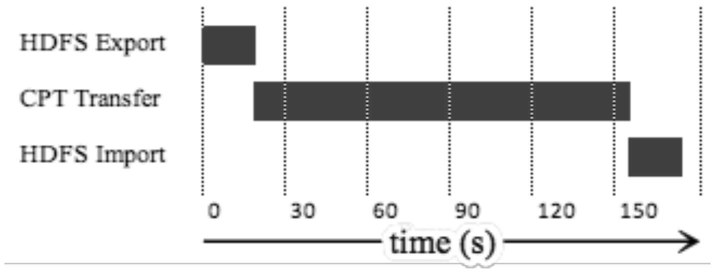

Figure 6.6 Timeline of 2x 512MB CPT transfer (no spawning) for HDFS.

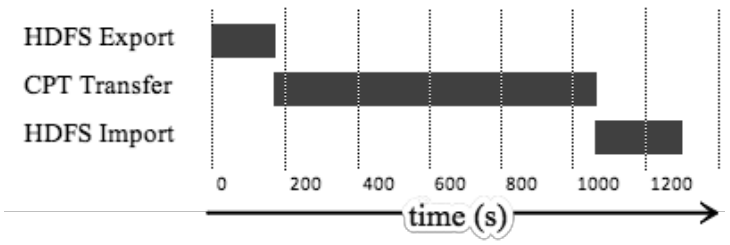

Figure 6.7 Timeline of 12x 512MB CPT transfer (no spawning) for HDFS.

Figure 6.7 depicts the time spent of each stage when transferring 12x 512MB files with CPT. As the total size to transfer has increased, it can be noticed that the overheads which cannot be parallelized (i.e. HDFS import and export) is a higher proportion of the total time.

Next, Figure 6.8 shows the result of transfer with spawning 2 additional intermediate nodes without pre-testing. As expected, both DistCp and CPT demonstrated better performance when additional intermediate nodes are spawned. However, just comparing the results of scaling with m4.large VM type does not show that CPT or DistCp scales better than one another. In this regards, they are of equal.

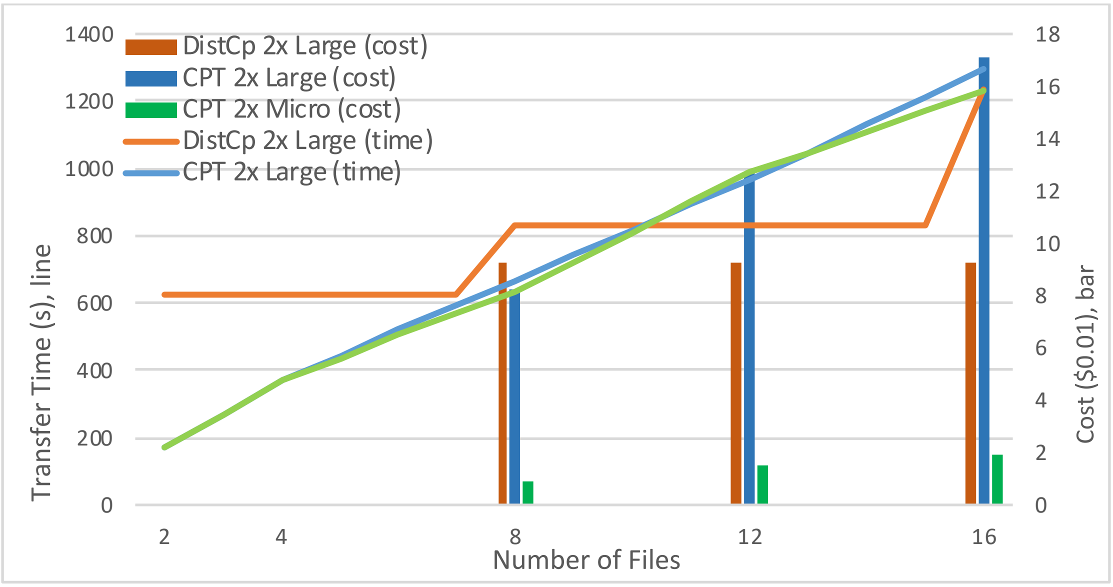

Figure 6.8 Transfer time and cost of two additional pairs of instances for DistCp and CPT

As no additional nodes were spawned when the number of files is 2 and 4, the transfer time for both CPT and DistCp is similar to the result in the previous experiment (Figure 6.5). The reason, as described earlier in the CPT framework, Transfer Manager forecasted that the transfer time will not decrease and hence decided not to spawn any intermediate nodes. No additional cost is incurred as no intermediate node is used.

For transfer of lesser than 8 files (i.e. 8 pieces of 512MB each), the utilization of t2.micro for CPT gave similar performance as m4.large but with more than 75% cost savings over both CPT with t2.large and DistCp. At transfer of 12 files, CPT resulted in longer transfer time compared to DistCp. At transfer of 16 files, CPT with t2.micro resulted in similar transfer time as DistCp while t2.large took a longer time to complete the transfer. As expected, CPT with t2.micro is able to match the performance CPT with t2.large while incurring much lower cost.

For DistCp, introducing additional nodes increases the number of map task running in parallel – and hence, reduces the total time taken to complete the transfer. However, DistCp requires running certain Hadoop components which comes with higher resource requirement and is not able to run on lower end machines. It is not possible to run DistCp with t2.micro.

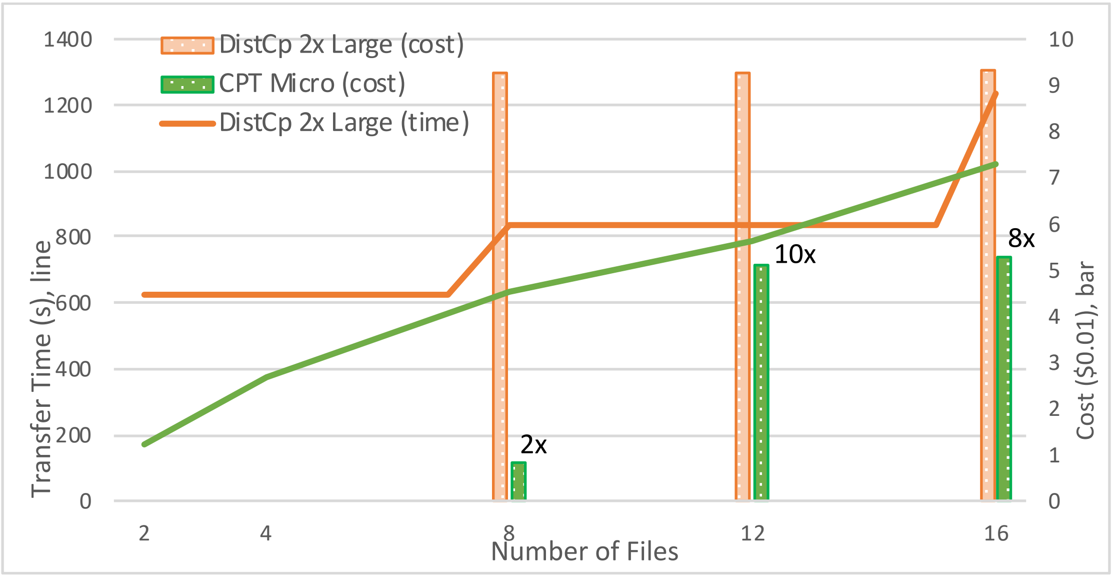

Figure 6.9 Transfer time of CPT with p<=10 to match DistCP's transfer time.

Figure 6.9 shows the result when running CPT with a variable number of t2.micro intermediates nodes in order to at least match the performance of DistCp i.e. CPT transfer time is kept as similar to DistCp as possible, the number of intermediate nodes for CPT is varied. As shown in the figure, 2 pairs of intermediate nodes (denoted as 2x) were used during the transfer of 8 pcs of 512mb files. As the number of files increases, the number of intermediate nodes required for CPT to match DistCp increases. It is observed that this is particularly true when the number of files is right before the next "step" of DistCp – the number of files to transfer is divisible by the number of files that can be handled by DistCp at a time in parallel.

Besides, it is also observed that adding many nodes to CPT only brings diminishing marginal improvement to the performance as it approaches the saturation point (where all the part that can be parallel its time spent is closer to 0).

Overall, the experiments show that CPT outperforms DistCp and comes with the flexibility of using running with lower end machines which often has much higher network performance to cost ratio.

6.3 Summary

The work described in this chapter has validated the CPT models and demonstrated that CPT is able to reduce the transfer time compared to sequential transfers. It is also learnt that speedup is low for transfer of small total data size, however, benefits of CPT get significant for larger file transfer. The exact number depends on the other factors such as internal and external network throughput.

The result also showed that as the number of pairs of intermediate nodes increases the transfer time reduces. Besides, pre-testing results in more predictable performance but incur additional time overhead. The result of CPT on HDFS cluster demonstrated that CPT outperforms DistCp. The transfer time of CPT is not only lesser than DistCp, but also has a lower cost – up to 8x in certain scenario.

In a nutshell, corresponding to the third research objectives of our work, the results validated that it is possible to scale cloud-to-cloud data transfer that is fully implemented by cloud consumer by spawning intermediate nodes. The implemented solution outperforms existing state-of-the-art data transfer techniques.

CHAPTER 7

CONCLUSION

This chapter summarizes the research work, provides a conclusion to the dissertation and describes the future work.

7.1 Revisiting the Objectives

This research studied data transfer between cloud VMs sitting across different datacenters. The existing de-facto approach is point-to-point transfer performed by cloud consumer where the bandwidth of the VM is capped by cloud provider. The dissertation proposed Cloud Parallel Transfer (CPT), the technique of parallelizing data transfer across intermediate nodes that are spawned specifically for the purpose of scaling data transfer. The first objective of the dissertation is to model and identify the limiting factors of scaling data transfer via aggregating bandwidth of intermediate nodes. The transfer time and financial cost models are introduced as a basis for estimating the performance and cost of the parallel transfer. The process flow is derived and two optimizations network data piping and pipelining were introduced to reduce the total transfer time.

The second objective is to validate and enhance the model for implementation on a public cloud. A short study on the VMs offering, pricing model and network throughput behavior is made. Here, two techniques are proposed; VM-type selection and pre-testing. Based on the test conducted on AWS cloud, the derived models are studied in order to understand the various factors involved in implementing the proposed approach in a cloud environment. From this research, a framework for end-to-end parallel transfer is proposed. The framework covers transfer time and cost estimation based on the models, and high-level fault tolerance when intermediate node fails.

The third and final objective is to implement CPT and compare it with existing data transfer solution. The dissertation describes the implementation of the proposed framework, then, by experiments, the performance and financial cost of the proposed framework is compared to existing sequential transfer. The result demonstrated that unlike typical methods such as sequential transfers, CPT is able to circumvent the network bandwidth allocation of the VM.

The proposed transfer is adapted and compared to state-of-the-art parallel file transfer for Hadoop environment – DistCp. The result showed that CPT is able to reduce the transfer time for cases when the number of files is low. DistCp starts to perform better when the number of files increases. CPT is also compared to the modified DistCp which can scale by adding nodes to the cluster, CPT not only performs better but also with cost reduction by multiple factors.

In a nutshell, this research aims to explore the feasibility and challenges of scaling cloud-to-cloud VM data transfer by circumventing the network allocation of VMs. The study is made by modelling the proposed solution, implementing the proposed framework and performing experimental analysis. Meeting the research objectives has led to major contributions in the area of utilizing intermediate nodes to improve data transfer throughput. The work produces a working cloud-to-cloud data transfer based on the designed parallel transfer framework. The solution does not require any cloud provider's insight.

7.2 Limitation and Future Work

An issue that comes with this framework is the increased in operational complexity as file chunks and intermediate nodes have to be managed during the lifetime of the transfer. Besides, the framework is only applicable when the cloud provider limits the network bandwidth on VM level. If the bandwidth is limited based on other metrics such as per tenant or per account, the technique to aggregate bandwidth as described in this work will not work.

In future work, m-to-n mapping of intermediate nodes should be explored in order to increase the efficiency in cases where similar VMs in source and destination DCs is not possible i.e. different cloud platform. Another potential area for further exploration will be to use other forms of elastic cloud storage services or microservices as intermediate nodes. This removes the need for managing the life of VMs as intermediate nodes and potentially further reduces the transfer cost.

REFERENCES

Allcock, W., n.d. GridFTP: Protocol Extensions to FTP for the Grid [Online]. Available at: http://toolkit.globus.org/alliance/publications/papers/GFD-R.0201.pdf [Accessed: 1 May 2016].

Allcock, W. et al., 2005. The Globus Striped GridFTP Framework and Server, in: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, SC ’05. IEEE Computer Society, Washington, DC, USA, pp. 54–60.

Amin, A. et al., 2011. High Throughput WAN Data Transfer with Hadoop-based Storage. J. Phys. Conf. Ser. 331, 052016. pp 1–1.

Apache Hadoop Distributed Copy – DistCp Guide [Online], n.d. Available at: https://hadoop.apache.org/docs/current/hadoop-distcp/DistCp.html [Accessed: 30 April 2018].

AWS | Amazon EC2 | Pricing [Online], n.d. Available at: https://aws.amazon.com/ec2/pricing/ [Accessed: 28 September 2016].

AWS Import/Export - Cloud Data Transfer Services [Online], n.d. Available at: https://aws.amazon.com/importexport/ [Accessed: 10 October 2017].

Azure Import/Export [Online], n.d. Available at: https://azure.microsoft.com/en-us/services/storage/import-export/ [Accessed: 15 October 2017].

Azure Linux VM sizes - General purpose [Online], n.d. Available at: https://docs.microsoft.com/en-us/azure/virtual-machines/linux/sizes-general [Accessed: 7 February 2018].

Batch Cloud Data Transfer | AWS Snowball [Online], n.d. . Amaz. Web Serv. Inc. Available at: https://aws.amazon.com/snowball/ [Accessed: 20 May 2018].

Bhardwaj, D., Kumar, R., 2005. A parallel file transfer protocol for clusters and grid systems, in: E-Science and Grid Computing, 2005. First International Conference On e-Science and Grid Computing. pp. 254.

Cho, B., Gupta, I., 2011. Budget-constrained Bulk Data Transfer via Internet and Shipping Networks, in: Proceedings of the 8th ACM International Conference on Autonomic Computing, ICAC '11. ACM, New York, NY, USA, pp. 71–80.

Cho, B., Gupta, I., 2010. New Algorithms for Planning Bulk Transfer via Internet and Shipping Networks, in: Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference On. pp. 305–314.

dd(1): convert/copy file - Linux man page [Online], n.d. Available at: http://linux.die.net/man/1/dd [Accessed: 26 September 2016].

Dealing with cloud storage service providers: Avoiding vendor lock-in [Online], n.d. Available at: http://searchcloudstorage.techtarget.com/tip/Dealing-with-cloud-storage-service-providers-Avoiding-vendor-lock-in [Accessed: 3 October 2015].

Divakaran, D.M., Gurusamy, M., 2015. Towards Flexible Guarantees in Clouds: Adaptive Bandwidth Allocation and Pricing. IEEE Trans. Parallel Distrib. Syst. 26, pp. 1754–1764.

Dropbox [Online], n.d. . Dropbox. Available at: https://www.dropbox.com/ [Accessed: 30 August 2017].

E, J., Cui, Y., Wang, P., Li, Z., Zhang, C., 2018. CoCloud: Enabling Efficient Cross-Cloud File Collaboration Based on Inefficient Web APIs. IEEE Trans. Parallel Distrib. Syst. 29, pp. 56–69.

Egress Throughput Caps | Compute Engine [Online], n.d. . Google Cloud. Available at: https://cloud.google.com/compute/docs/networks-and-firewalls#egress_throughput_caps [Accessed: 3 June 2018].

Garcia-Dorado, J.L., Rao, S.G., 2015. Cost-aware Multi Data-Center Bulk Transfers in the Cloud from a Customer-Side Perspective. Cloud Comput. IEEE Trans. On PP, pp. 1–1.

Gilani, M., Inibhunu, C., Mahmoud, Q.H., 2015. Application and network performance of Amazon elastic compute cloud instances, in: 2015 IEEE 4th International Conference on Cloud Networking (CloudNet). pp. 315–318.

Google Compute Engine Pricing | Compute Engine Documentation [Online], n.d. . Google Cloud. Available at: https://cloud.google.com/compute/pricing [Accessed: 29 April 2018].

Google Drive - Cloud Storage & File Backup for Photos, Docs & More [Online], n.d. Available at: https://www.google.com/drive/ [Accessed: 30 July 2018].

Hacker, T.J., Noble, B.D., Athey, B.D., 2004. Improving throughput and maintaining fairness using parallel TCP, in: IEEE INFOCOM 2004. pp. 2480–2489 vol.4.

Hortonworks Data Cloud for AWS [Online], n.d. Available at: https://hortonworks.com/products/data-platforms/cloud/aws/ [Accessed: 2 January 2018].

Hu, Z., Li, B., Luo, J., 2018. Time- and Cost- Efficient Task Scheduling across Geo-Distributed Data Centers. IEEE Trans. Parallel Distrib. Syst. 29, pp. 705–718.

Incentives Build Robustness in BitTorrent [Online], n.d. Available at: http://www.bittorrent.org/bittorrentecon.pdf [Accessed: 27 February 2015].

iperf - Linux man page [Online], n.d. Available at: https://linux.die.net/man/1/iperf [Accessed: 8 October 2017].

Jeong, K., Figueiredo, R., Ichikawa, K., 2017. On the Performance and Cost of Cloud-Assisted Multi-path Bulk Data Transfer, in: 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). pp. 186–193.

Khanna, G. et al., 2008. Multi-hop Path Splitting and Multi-pathing Optimizations for Data Transfers over Shared Wide-area Networks Using gridFTP, in: Proceedings of the 17th International Symposium on High Performance Distributed Computing, HPDC ’08. ACM, New York, NY, USA, pp. 225–226.

Kissel, E., Swany, M., Brown, A., 2011. Phoebus: A System for High Throughput Data Movement. J Parallel Distrib Comput 71, pp. 266–279.

Kolano, P.Z., 2013. High Performance Reliable File Transfers Using Automatic Many-to-many Parallelization, in: Proceedings of the 18th International Conference on Parallel Processing Workshops, Euro-Par'12. Springer-Verlag, Berlin, Heidelberg, pp. 463–473.

Laoutaris, N., Sirivianos, M., Yang, X., Rodriguez, P., 2011. Inter-datacenter Bulk Transfers with Netstitcher, in: Proceedings of the ACM SIGCOMM 2011 Conference, SIGCOMM '11. ACM, New York, NY, USA, pp. 74–85.

Laoutaris, N., Smaragdakis, G., Stanojevic, R., Rodriguez, P., Sundaram, R., 2013. Delay-tolerant Bulk Data Transfers on the Internet. IEEEACM Trans Netw 21, pp. 1852–1865.

Liu, W.-L., 2013. Cloud Storage Performance and Security Analysis with Hadoop and GridFTP. Master's Project, San Jose State University, USA.

Lu, D., Qiao, Y., Dinda, P.A., Bustamante, F.E., 2005. Modeling and Taming Parallel TCP on the Wide Area Network, in: Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International. pp. 68b-68b.

Magic Quadrant for Cloud Infrastructure as a Service, Worldwide [Online], n.d. Available at: https://www.gartner.com/doc/reprints?id=1-2G2O5FC&ct=150519&st=sb [Accessed: 2 July 2018].

Mogul, J.C., Popa, L., 2012. What We Talk About when We Talk About Cloud Network Performance. SIGCOMM Comput Commun Rev 42, pp. 44–48.

nc - arbitrary TCP and UDP connections and listens - Linux man page [Online], n.d. Available at: https://linux.die.net/man/1/nc [Accessed: 22 Febrary 2017].

Ou, Z., Zhuang, H., Lukyanenko, A., Nurminen, J.K., Hui, P., Mazalov, V., Ylä-Jääski, A., 2013. Is the Same Instance Type Created Equal? Exploiting Heterogeneity of Public Clouds. IEEE Trans. Cloud Comput. 1, pp. 201–214.

Persico, V., Botta, A., Marchetta, P., Montieri, A., Pescapé, A., 2017. On the performance of the wide-area networks interconnecting public-cloud datacenters around the globe. Comput. Netw. 112, pp. 67–83.

Persico, V., Marchetta, P., Botta, A., Pescape, A., 2015. On Network Throughput Variability in Microsoft Azure Cloud, in: 2015 IEEE Global Communications Conference (GLOBECOM). pp. 1–6.

Pricing - Linux Virtual Machines | Microsoft Azure [Online], n.d. Available at: https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/ [Accessed: 29 March 2018].

Pucha, H., Kaminsky, M., Andersen, D.G., Kozuch, M.A., 2008. Adaptive File Transfers for Diverse Environments, in: USENIX 2008 Annual Technical Conference on Annual Technical Conference, ATC'08. USENIX Association, Berkeley, CA, USA, pp. 157–170.

Raghavan, B., Vishwanath, K., Ramabhadran, S., Yocum, K., Snoeren, A.C., 2007. Cloud Control with Distributed Rate Limiting. SIGCOMM Comput Commun Rev 37, pp. 337–348.

Ramakrishnan, L., Guok, C., Jackson, K., Kissel, E., Swany, D.M., Agarwal, D., 2010. On-demand Overlay Networks for Large Scientific Data Transfers, in: Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference On. pp. 359–367.

rsync(1) - Linux man page [Online], n.d. Available at: http://linux.die.net/man/1/rsync [Accessed: 27 February 2016].

Scheuner, J., Leitner, P., 2018. A Cloud Benchmark Suite Combining Micro and Applications Benchmarks, in: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE '18. ACM, New York, NY, USA, pp. 161–166.

Sim, A., 2009. Berkeley Storage Manager (BeStMan). Available at: https://sdm.lbl.gov/bestman/docs/bestman-overview-091006.pdf [Accessed: 27 April 2017]

Sinha, S., Niu, D., Wang, Z., Lu, P., 2016. Mitigating Routing Inefficiencies to Cloud-Storage Providers: A Case Study, in: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 1298–1306.

Tudoran, R., Costan, A., Antoniu, G., 2014a. 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. pp. 51–56.

Tudoran, R., Costan, A., Wang, R., Bouge, L., Antoniu, G., 2014b. Bridging Data in the Clouds: An Environment-Aware System for Geographically Distributed Data Transfers, in: Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium On. pp. 92–101.

Wang, G., Ng, T.S.E., 2010. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center, in: 2010 Proceedings IEEE INFOCOM. pp. 1–9.

Wu, Z., Butkiewicz, M., Perkins, D., Katz-Bassett, E., Madhyastha, H.V., 2013. SPANStore: Cost-effective Geo-replicated Storage Spanning Multiple Cloud Services, in: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP '13. ACM, New York, NY, USA, pp. 292–308.

Yildirim, E., Arslan, E., Kim, J., Kosar, T., 2016. Application-Level Optimization of Big Data Transfers through Pipelining, Parallelism and Concurrency. IEEE Trans. Cloud Comput. 4, pp. 63–75.

Zhang, M., Kissel, E., Swany, M., 2015. Using phoebus data transfer accelerator in cloud environments, in: Communications (ICC), 2015 IEEE International Conference On. pp. 351–357.