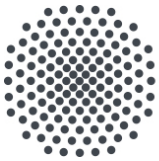

Universität Stuttgart

University of Stuttgart
Universitätsstraße 38 D–70569 Stuttgart
Faculty of Computer Science, Electrical Engineering and Information Technology
Institute of Parallel and Distributed Systems

Master's Thesis

Cost Optimization for Data Placement Strategies in an Analytical Cloud Service

submitted by

Muhammad Usman Saleem
registration number: 2915070

Examiner:

Prof. Dr.–Ing. habil. Bernhard Mitschang

Supervisor:

Dipl.–Inf. Felix Beier

Stuttgart, 17. Oktober 2016

Abstract

Analyzing a large amount of business-relevant data in near-realtime in order to assist decision making became a crucial requirement for many businesses in the last years. Therefore, all major database system vendors offer solutions that assist customers in this requirement with systems that are specially tuned for accelerating analytical workloads. Before the decision is made to buy such a huge and expensive solution, customers are interested in getting a detailed workload analysis in order to estimate potential benefits. Therefore, a more agile solution is desirable having lower barriers to entry that allows customers to assess analytical solutions for their workloads and lets data scientists experiment with available data on test systems before rolling out valuable analytical reports on a production system.

In such a scenario where separate systems are deployed for handling transactional workloads of daily customers business and conducting business analytics on either a cloud service or a dedicated accelerator appliance, data management and placement strategies are of high importance. Multiple approaches exist for keeping the data set in-sync and guaranteeing data coherence with unique characteristics regarding important metrics that impact query performance, such as the latency when data will be propagated, achievable throughputs for larger data volumes, or the amount of required CPU to detect and deploy data changes. So the important heuristics are analyzed and evolved in order to develop a general model for data placement and maintenance strategies. Based on this theoretical model, a prototype is also implemented that predicts these metrics.

Acknowledgements

I would like to thank Mr. Felix Beier who supervised me during my thesis and provided very useful tips and reviews at each stage and Prof. Dr. Bernhard Mitschang for examining my thesis and providing me an opportunity to work under his supervision. A limitless acknowledgment for my family, who was a continuous source to build my motivation during the complete tenure of my thesis and to my friends who helped me to resolve my confusions I had during my work. Finally, special thanks to Ms. Eva Strähle and Mr. Timo Schweizer for helping me in administrative issues.

Eidesstattliche Erklärung / Affirmation

Hiermit versichere ich,

Muhammad Usman Saleem, geboren am 26. August 1988 in Pakistan,
dass ich die vorliegende Masterarbeit mit dem Titel

Cost Optimization for Data Placement Strategies in an Analytical Cloud Service

selbständig verfasst und ausschließlich die angegebenen Quellen und Hilfsmittel verwendet habe. Alle von mir aus anderen Veröffentlichungen übernommenen Passagen sind als solche gekennzeichnet.

Hereby I,

Muhammad Usman Saleem, born on 26. August 1988 in Pakistan,
affirm that I have written this master's thesis with the title

Cost Optimization for Data Placement Strategies in an Analytical Cloud Service

on my own without making use of any supporting resources, that have not been mentioned and that literal citations from literature and also the using of thoughts from other authors have been cited accordingly within the thesis. I am aware of the fact that a wrong declaration can have legal consequences.

Stuttgart, October 14, 2016 Muhammad Usman Saleem

Trademarks

IDAA, InfoSphere CDC, DB2, DB2 Universal Database, DB2 for z/OS, IBM dashDB, System Z, IBM are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java, MySQL are trademarks of Oracle in the US, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Contents

Acronyms

AaaS Analytics as a Service

API Application Programming Interface

AWS Amazon Web Services

BI Business Intelligence

BR Bulk Refresh

CapEx Capital Expense

CDC Change Data Capture

CE Cost Effectiveness

DBMS Database Management System

DML Data Manipulation Language

DWH Data Warehouse

ETL Extract, Transform, Load

FASMI Fast Analysis of Shared Multidimensional Information

FPGA Field Programmable Gate Array

FR Full Refresh

IaaS Infrastructure as a Service

IBM International Business Machines Corporation

IDAA IBM DB2 Analytics Accelerator

IO Input/Output

IoT Internet of Things

IR Incremental Refresh

JCL Job Control Language

JMS Java Message Service

JSON JavaScript Object Notation

JSP Java Server Pages

MOM Message Oriented Middleware

MQ Message Queue

MQI Message Queue Interface

MQM Message Queue Manager

NIST National Institute of Standards and Technology

OLAP Online Analytical Processing

OLTP Online Transactional Processing

OpEx Operating Expense

OSI Open Systems Interconnection

PaaS Platform as a Service

QM Queue Manager

REST REpresentational State Transfer

SaaS Software as a Service

SOA Service Oriented Architecture

SP Stored Procedure

XML Extensible Markup Language

List of Figures

List of Tables

Chapter 1

Introduction

Analytical cloud service or data analytic software as a service is a cloud-based analytical service which is fed with data and it provides back analytics. This service means a major shift of infrastructural paradigm from on-premise to cloud, where resources, components, and infrastructure are relocated from on-premise to the remote cloud-based location. Before that cloud-based analytical on-line services could get their popularity, organizations and customer were used to host complex data warehouse infrastructure for their in-house businesses, which involves a lot of deployment and management. And they need to spend a tremendous amount of money as well to get this infrastructure always up and running. This entire infrastructure, although it is not their mainstream business, becomes an important unit that needs to be maintained at any cost to run OLAP (Online Analytical Processing) queries for enterprise planning and decision making to see how their business is expanding. And here come these cloud-based on-line analytical services into play which let businesses to grow without hosting business decision support systems, data warehouses, data marts etc, because these on-line services offer everything and even many more options to the users to run their on-line queries and make important business decisions and not hosting even a single component of this complex infrastructure.

Data acquisition and placement is the backbone of this infrastructure because data gets updated every fraction of seconds or even milliseconds in transactional databases. Not updating this transactional data continuously into on-line data service backing an analytical application can lead to aged OLAP reporting which in return can become the basis of wrong business decisions in mission critical applications. In a traditional approach, where data warehouse is hosted and maintained solely by the customers themselves with mainstream business components, many good to very good heuristics are being in practice that are developed over the years to provide near real-time data transfer and reporting. But when it comes to an on-line analytical service that is hosted on cloud and shifts the burden of hosting and maintenance a data warehouse in-house to cloud providers, a lot more factors are involved that should be kept in mind whose poor configuration can lead to poor performance of the overall system while acquiring data from

transactional databases on customer side and placing it to on-line databases. A re-thinking and (maybe) re-factoring of these data placement heuristics are needed to make them again efficient and cope with the current situation.

These days AaaS (Analytics as a Service) is getting its popularity because of very efficient and easy-to-access public cloud offerings. AaaS is an emerging concept where cloud providers not only store customer's data but also provide an infrastructure to perform analytics on it. Customers can avail these analytical services without hosting a single component on their infrastructure and continue to focus on their core businesses. It also assists companies and IT departments to re-model their expenditures because cloud computing and "as-a-Service" services dramatically move CapEx (Capital Expense) to OpEx (Operating Expense). Although it is not limited, companies with less capital investment can take huge advantage out of cloud computing and related technologies and can spend more money on their revenue-generating processes [1]. The capital cost of analytics infrastructure can be converted into operating cost by using AaaS model. It also provides an on-demand BI (Business Intelligence) or cloud BI that involves delivery of BI applications to end users from a hosted location. They are scalable and makes start-up easier and less expensive.

If companies host large and very sophisticated analytics infrastructure on their premises, they have to cope with its installation, administration as well as setup. A full dedicated team consisting of consultants and technical members is required to deal with such an enormous architecture, installations of hardware and software components and setup issues. In the case of on-premise analytics, companies should pay continuously to keep such a complicated and gigantic setup of components running to perform analytics. In contrast to this traditional approach, cloud providers are responsible for managing entire infrastructure in off-premise analytics. They have technical and well-skilled teams who are responsible to keep it running without impacting their customers. AaaS model of analytics is an easy to use and cost-effective way of performing analytics which lets core business grow exponentially with right decisions.

1.1 Motivation

According to analytics in the cloud report [2] published in 2015, data acquisition is one of the core components of analytical environments and the second most popular choice of respondents of analytical projects. Efficient data acquisition is of paramount importance for fast and near real-time replication of data. This efficiency is only possible if all related heuristics are carefully analyzed while developing an infrastructure of a replication system. These heuristics involves but not limited to performance metrics such as latency, throughput, CPU usage, cost-effectiveness, and pricing. These heuristics assist system architect to come up with an optimized solution for the entire system.

As an example, few systems are described here where efficient data replication from data sources to central data repository which is used to run OLAP queries and take important run-time decisions, is very critical. Consider a traffic management system which manages city traffic by taking input from surveillance units located in different regions of the city. Each unit is installed with a video camera that constantly records video of its surroundings and stores it to a local storage. Although the main purpose of these cameras is to assure security but can also be used to feature a traffic management in that region. The new system, capable of featuring traffic management, consists of a data collector component that collects data from the local storage of surveillance units and send it to a central repository where a personal can view and monitor the current situation of traffic in different regions of the city and based on his/her observation can issue special instructions to enhance traffic situation in a particular region. Obviously, this system will only be effective if data collector can collect real-time data from local storage of units. Instructions lead by a personal by working on an obsolete data does not make any sense.

Applications of near real-time replication systems are not limited. They are becoming a part of almost every important business. To motivate the readers of this thesis, some more examples are presented where these systems are of utmost importance. Take an example of electricity production and distribution system, electricity can be generated from many sources such as wind, water, sun, atom or coal and added to the central electric grid station. From there it can be further distributed to different states and regions based on their demands. The production capacity of a source, as well as actual consumption capacity of a region, are usually not constant. The central grid station will only be able to distribute electricity to different regions appropriately if it knows the actual consumption need along with the production capacity of all individual sources. This task is performed by replication components deployed both on source and consumption regions. Again efficiency of the replication component is very important to meet actual requirements. The same system can also be used to view the performance of each production unit along with time dimension which will give an idea to decision makers about their next investment.

As few useful real-life examples have already been presented, many other applications also require these replication systems to work effectively. In the market of stock exchange, for instance, data flow in real time between different services requires very fast and efficient data movement techniques to keep stock buyers and sellers up-to-date with their current stock values. And this fast data movement and synchronization between different on-premise and off-premise services heavily depend on how a system is structured. There are already quite good methodologies available for data acquisition from an on-premise implementation and transferring it to another on-premise application, but very few work is done so far to get an enormous amount of data from on-premise services and moving it in near real-time to a cloud environment and an in particular environment which hosts an analytical service.

Data Acquisition Transfer Models

This section discusses which data acquisition and transfer models are available that are worth to consider here. As per requirements and architecture, each system requires a unique data acquisition and transfer model. Fundamentally they can be categorized into four basic models as discussed in analytics in the cloud report [2] and are listed here:

1. On-Premise to On-Premise Data Transfer
2. Cloud to Cloud Data Transfer
3. Cloud to On-Premise Data Transfer
4. **On-Premise to Cloud Data Transfer**

In the first data transfer model, data from in-house source components is replicated to in-house target components. In the second approach, source and destination data storage both are located in a public cloud, and data is transferred inside a cloud or from one cloud to another cloud. The third option is not commonly used for analytical purposes but in case if an application is already running on a cloud and needed to be integrated with a local application.

The fourth model is the focal point of this thesis because it deals with hosting transactional data sources locally by customers and one or more central repositories in the form of data warehouse on a public cloud. Which enables them to use cloud analytical and BI services as well. This model keeps target database in sync with the source database and is vital for nowadays businesses to flourish by reducing their upfront costs and paying attention to just their main business units and players.

1.2 Contribution

This thesis frames a cost model of the replication systems. Although these systems are present in the market with quite a big success from last many years, a change of computing paradigm asserts a need of re-work and possibly re-shaping. This emerging computing paradigm, also known as cloud computing, eases a business to cut down its entire or some part of IT infrastructure by allowing it to host by a third party. If entire IT infrastructure is moved to a public cloud, then business is accessed via services and interfaces and customers pay only for their usage. In some situation, it worths to have some part of IT infrastructure hosted on-premise and some on a cloud. In this scenario, data should flow to and from between different services. The study of

this data movement is the core of this thesis, specifically in the case where data moves from an on-premise service to an analytical service on the cloud.

When a replication system is taken into account, many questions arises related to its efficiency and performance. The work done in this thesis answers them. How efficiently a customer can move her data from on-premise data sources to cloud platform consisting of analytical services considering in mind time, replication cost, data rate or throughput, CPU usage and disruption aspects, etc. How many possibilities or approaches a customer has to clone her in-house operational storage to cloud based data services that will be used later to provide near real-time OLAP queries back to the client. What would be the smartest and intelligent move out of these available approaches to enjoy best-of-breed functionality?

In the course of this thesis, a generic data replication model is developed which can be applied to any data replication strategy along with a model to estimate performance and financial aspects which can be used to evaluate a chosen data replication technique with the help of latency, CPU usage, throughput, etc. The impact of data compression, decompression and transformation of data before transferring it on the wire is also taken into account while considering data synchronization. A web tool is implemented which will calculate values of these metrics to estimate the performance of selected strategy quickly. In the end, a particular data transfer technique will be chosen, this model will be applied to evaluate its performance metrics with developed tool, and evaluation results will summarize this work with a comparison of estimated and original values of parameters.

1.3 Structure

The complete work is classified into different chapters, each of which explains a particular aspect of this thesis. The classification of this thesis into various chapters will help the reader to quickly grasp entire work by reading the whole or just a part. If a user is confident enough to have background knowledge needed to catch further technical details, he/she can just overlook it and move on to next chapters. The sequence in which the work is divided and presented help even a novice reader to get valuable information from this thesis.

Classification

The thesis work is classified into following chapters:

Chapter 1 – Introduction: This chapter provides a brief introduction of the thesis along with motivation and contribution to the specified problem.

Chapter 2 – Background and Technology Review: provides essential technical skills that prepare the reader to grasp further knowledge and concepts presented.

Chapter 3 – Related Work: an overview of past and related work is summarized and reader is made well-acquainted and conversant with missing work.

Chapter 4 – Conceptual Model and Design: presents a conceptual model of the replication strategies and how to evaluate them with performance metrics.

Chapter 5 – Implementation: provides a tool that implements our conceptual model and a replication system on a real-time environment for a given use case.

Chapter 6 – Evaluation: chapter evaluates the implemented replication system with the help of performance metrics.

Chapter 7 – Conclusion and Outlook provides conclusive words and a short summary of the overall chapters and future work.

Chapter 2

Background and Technology Review

This chapter explains some background of key concepts and technologies that reader should have to understand concepts and ideas presented in following chapters of this thesis. Some topics are covered with just basics, and wherever applicable the external pointers are provided to refer to.

2.1 Data Warehouse

DWH (Data Warehouse) is a central database that is deployed for analytical purposes. It is fed by many heterogeneous relational or non-relational source databases. A data warehouse can provide an overview of all data sources that are feeding it. Data is extracted from its data sources, processed, cleaned, transformed and then loaded into DWH. It also facilitates to execute data mining algorithms to provide insights of data, which enables to get valuable information from data about a particular business. By running mining techniques companies find out hidden patterns in the day-to-day transactional data and it opens up more opportunities and paths for the business to expand [3].

Consider an example of a big supermarket that runs hundreds of subsidiaries across a country and even worldwide. Each subsidiary has its personal database system where user transactions and sales information are stored. This single transactional database system does not provide any valuable information about the overall business and future trends. Instead, data is extracted from each subsidiary and loaded into a shared repository called DWH which is essentially a reflection of entire data in all relevant subsidiaries. The data in this joint repository or DWH is used to mine out hidden and useful information about the future business extension and to make better decisions.

2.1.1 Architecture

Fig. 2.1 shows basic components of a data warehouse architecture. Data is extracted from data systems which consist of different types of storage systems including relational database systems, legacy data storage, simple file-based or excel sheets. Here comes the real challenge to extract data from totally heterogeneous storage systems. It is then transformed and loaded into a data warehouse. This process is shortly named as ETL (Extract, Transform, Load). It involves multiple operations on data, i. e., processing, cleansing, aggregation, normalization, inconsistency resolution, etc., before it is loaded into DWH. Once data is in DWH, it presents a full aggregated and clean picture of data from all operational or legacy data sources which can be used for business analytics as well as data mining to figure out important business decisions. Another important component involves in DWH architecture is data warehouse manager that monitors source databases and ETL operations [4].

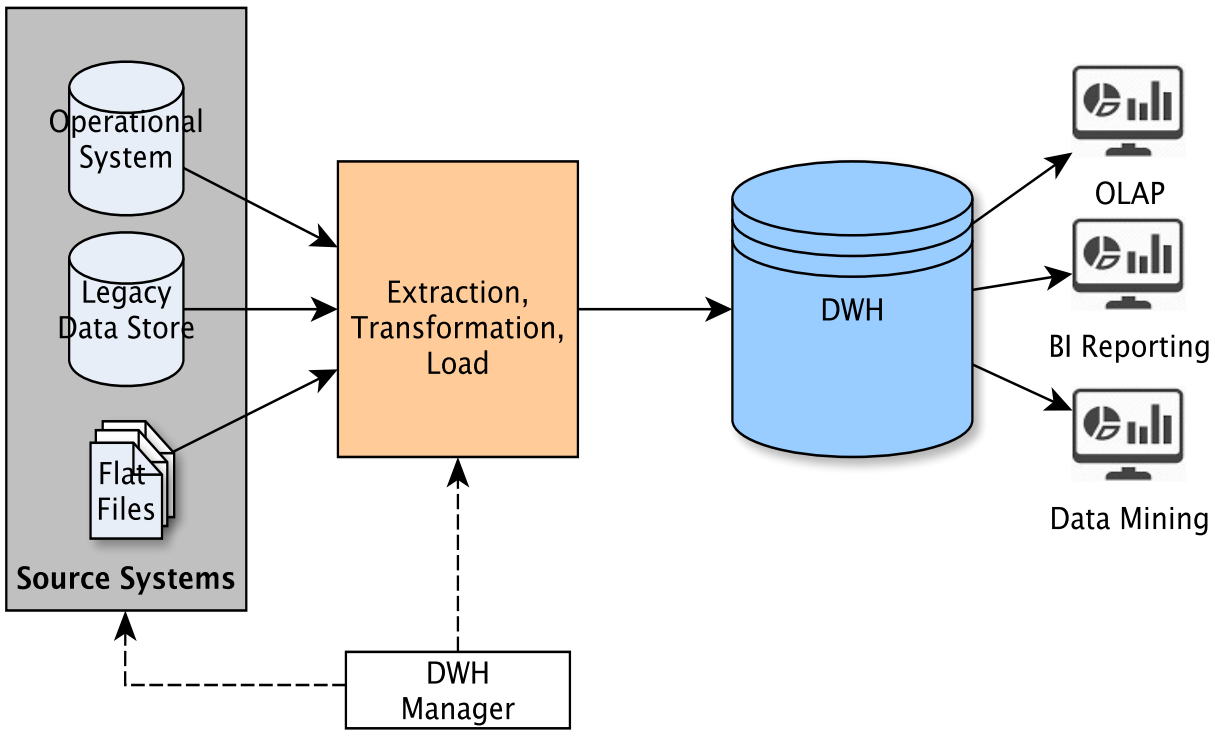

Figure 2.1: Data Warehouse Architecture

2.1.2 Online Transaction Processing

OLTP (Online Transactional Processing) is a mode to process data in a database in real-time and without any delay noticeable to front-end applications. In OLTP database queries are gen-

erally short, simple, do not overburden database system and therefore be processed as fast as possible. They are not supposed to return too much of data from the database. DML (Data Manipulation Language) queries, i. e., insert, update, and delete are characterized as OLTP queries because they require very fast query processing. Such kind of query processing mode is used in enterprise applications for their daily operational processes. Generally, OLTP systems are meant to handle a large number of users, so they should be able to manage enormous requests concurrently [5].

2.1.3 Online Analytical Processing

OLAP is a query processing mode which is supposed to manipulate large to very large queries that can return a tremendous amount of rows from multiple tables. These queries are usually used in analytical applications. They are very complex and long which can include a lot of join operations on different tables. Therefore these queries are not supposed to return data in real-time, but they should expose an interactive environment. The processing time of such a complex queries is also large. OLAP systems, in contrast to OLTP systems which stores transactional data, store usually historical and archived data that is specifically intended for reporting purposes. OLAP queries are not issued on a normal relational model of the database. Instead, OLAP systems use a multidimensional model to store data in a database. The data in these systems is not stored in traditional normalized schemas, instead, in special purpose schemas like star, snowflake schema, etc. Such a schemata consist of fewer tables, but each table contains a lot of data. Users of such systems are generally managers and analysts who want to get valuable information from historical data about their business. When comparing OLAP products, FASMI (Fast Analysis of Shared Multidimensional Information) test can be used which evaluates them on different factors [6, 7].

2.1.4 Summary

Data warehouse is the common repository which stores data from all or respective transactional data sources and makes the entire data available for analytical tasks, BI reporting, and data mining. To have a basic knowledge of data warehousing along with analytics is helpful to the reader to understand advanced concepts presented in this thesis. In introductory chapter Sec. 1 it is already mentioned that our focus is to analyze data replication strategies, which transfer data from on-premise services to off-premise analytical cloud service, for performance. This analytical service uses an underlying data warehouse to execute OLAP queries. Therefore it is beneficial to the reader to have basic concepts of these terminologies to grasp overall idea of the picture.

2.2 Cloud Computing

Cloud computing is a computing paradigm where resources are shared between multiple tenants to save the cost of infrastructure or computing resources. A shared set of resources which include servers, storage, network, computational power, services, applications, etc., are made available to different tenants. This type of computational model helps customers to reduce their infrastructure and upfront cost by using pay-as-you-go model provided by cloud providers. This model helps customers to focus on mainstream business.

According to the NIST (National Institute of Standards and Technology) definition [8], Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. .

2.2.1 Service Models

Cloud computing offers respective services in three different models, i. e., IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service) also known as cloud computing stack. This stack categorizes a broad range of services (networks, servers, applications, storage, or middleware services) built on top of one another and offered under the name of cloud. Each category of this stack targets a particular set of services to a unique set of users including end-users, developers, project managers, and more. For example, IaaS covers a set of hardware and software components that powers the availability of networks, storage, servers, operating systems through cloud. PaaS is the set of services that gears application developers to quickly develop and deploy a specific application using development platform services. And SaaS includes services which offer a set of finished or ready-to-use applications to the end-users [9]. Fig. 2.2 depicts different layers of service model. This cloud computing stack is explained further in following sections.

2.2.1.1 Infrastructure as a Service

This is a very primary form of cloud computing service model, which makes available infrastructure like CPU, memory, storage, or network, etc., as an on-demand service. Instead of purchasing entire hardware and software infrastructure, the tenants or customers hire it from some cloud provider in a public, private, or hybrid (a combined mode of public and private cloud provisioning) mode. Each vendor, who is offering IaaS, also provide an interface through

which infrastructure resources can be accessed. For example, Open Stack[1] is such an open source software that can manage a pool of computing resources and makes them available using OpenStack API (Application Programming Interface). AWS (Amazon Web Services)[2] is such an example whose infrastructure includes a wide variety of computational resources like servers, storage, databases, or networking services, etc [10, 11].

2.2.1.2 Platform as a Service

It is a service model of cloud computing in which cloud providers offer mainly development tools as a service for a quick development and easy deployment of an application. The services include generally development tools for the developers, web servers, already deployed databases, messaging servers, etc. Customers can hire these platform resources according to their needs. Such services help customers to cut down their finished product or application costs enormously by saving a lot deployment time. Developers can start developing an application without taking a headache of installing and deploying development tools first and can get scalability advantages as well by quickly running multiple instances of a service according to the needs. IBM Bluemix[3] provides a number of platform services covering different areas of businesses, i. e., databases, analytics, mobile, IoT (Internet of Things), network, etc [10, 11].

2.2.1.3 Software as a Service

SaaS vendors offers finished products or applications as on-line services to their customers or users. Cloud providers take care of back-end infrastructure, platform needed to run on-line services along with the application itself. End-users get an advantage of a ready-to-use application

[1] https://www.openstack.org/
[2] https://aws.amazon.com/
[3] http://www.ibm.com/cloud-computing/bluemix/

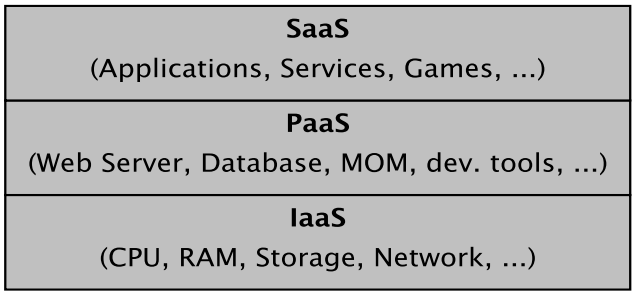

Figure 2.2: Cloud Computing Service Model [10]

for which they pay for its usage. In this cloud computing model, customers simply hire these services from their vendors and start using them. It provides an opportunity to businesses to reduce their cost by cutting down infrastructure and platform from their administration hands. For example, Google offers many applications as a service which includes Google Drive, office suites, etc [10, 11].

2.2.2 Deployment Models

These days cloud computing paradigm offers a lot of different options to the vendors as well as customers to benefit from it. According to NIST [8] there are four models available to deploy cloud architecture which is common to all service models and these are:

1. Public cloud
2. Private cloud
3. Hybrid cloud
4. Community cloud (out of scope)

2.2.2.1 Public Cloud

A shared pool of services is made available to multiple tenants. In public cloud deployment model, any number of tenants can access cloud, and a service or an IT resource can also be shared among multiple tenants. A tenant does not know how many other tenants are sharing same resource. This public cloud model helps to reduce the cost for an individual tenant because of this sharing mechanism. Typically not all tenants access a same resource at the same time which makes sharing possible, but if it so then cloud provider needs to scale that particular IT resource or service so that user do not face any performance issue. For example, many users can access a resource at the same time in business hours that will lead to the performance bottleneck. A cloud provider can leverage more instances of that resource in those busy hours and then reduce them again when they are not accessed by too many tenants. Fig. 2.3 gives a clear understanding of a public cloud being accessed by multiple tenants along with a resource [10, 11].

It is a good idea for a cloud provider to determine the business hours of each tenant, and then share a resource with those tenants whose business hours are different. For example, if two tenants A and B want to access a storage resource with their extreme business workloads at time t_1 and t_2 and these times are mutually exclusive then it is a right decision to let both tenants to share this resource. But if they both want to access that resource with t_1 and t_2 that are same then let tenant A to access some other instance of that resource, so that tenant B does not face any performance issue.

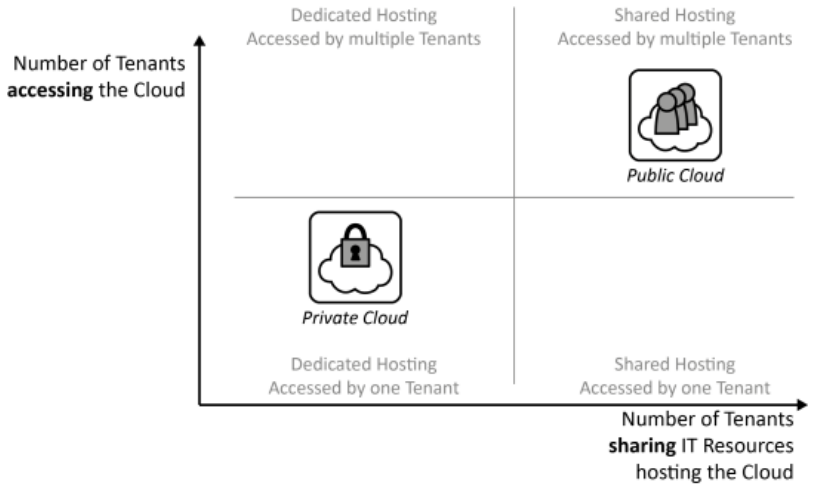

Figure 2.3: Public Cloud [11]

2.2.2.2 Private Cloud

Sometimes companies want to have a cloud infrastructure in their premises to access it dedicatedly and to have their application along data inside their firewall to avoid any security leaks that can arise due to a public cloud. The private cloud solves this problem, in which services are not shared among multiple tenants. Instead, only one tenant access a cloud environment along with its all services. Unlike public cloud which provides a shared resource to multiple tenants, a private cloud is dedicated to a single tenant or organization and offered services are also dedicated. No other tenant has access either to the private cloud or the provided resources. It is a very secure cloud deployment model and avoids any cloud computing security issues because it is deployed inside of an organization firewall. It is very suitable for the enterprises or organizations who have unpredictably high load, security concerns and can not afford any performance related issues. On the other hand, tenants have to pay more for this cloud model because resources are not shared with anybody. The responsibility to manage and control falls solely on an individual tenant, although there are some vendors, for example, Rackspace [4] who claims to take management responsibility as well. It is entirely possible to have few business units running on a public cloud and few on a private cloud. There are few more flavors available for the private cloud that is out of scope for this thesis and are not discussed here.

Fig. 2.3 provides an overview how a private cloud is accessed between tenants. The number of tenants who are accessing private cloud is one, and the number of tenants who are accessing an IT resource within a cloud is also one. It is obvious from the diagram that private cloud deployment model is entirely a dedicated environment for an individual tenant or an organization where no one else has access either to the cloud or to the resources, which makes it very secure environment [10, 11].

[4] http://www.rackspace.co.uk/cloud/private

2.2.2.3 Hybrid Cloud

Hybrid cloud is a cloud computing deployment model in which a tenant or an organization hosts some of the resources in its premises while some other resources are hosted by some external vendor who is offering cloud services publicly. It is a mix of above mentioned two cloud deployment models, i. e., public cloud, and private cloud. Some tenants or organizations do not want to manage all their IT infrastructure but very mission critical components. This cloud computing model facilitates those tenants to have some of their infrastructure (which is not so significant regarding security or performance) running on a public cloud and also to have own private cloud where they can run and manage important components, for instance, some business processes which a corporate do not want to expose to others or important data that should not reach into other hands in any circumstances [10, 11].

NIST defines this deployment model as a composition of two or more different cloud infrastructures (private, community, or public) that remain unique entities but are connected together by standardized or proprietary technology that enables data and application portability [8].

2.2.3 Summary

Cloud computing is the rapidly emerging technology which allows IT resources and components to run on the infrastructure of a third-party or cloud provider and let tenants use their services according to their needs. This computing paradigm helps companies to hire IT services from cloud providers while focusing on their mainstream business. Different data transfer approaches of an analytical service, which is running as-a-service on a cloud, are analyzed for their performance and monetary metrics. To feed the data warehouse of this analytical service, transactional data sources are running on the tenants-premises. The complete picture is the perfect example of cloud computing, and the reader should have a brief knowledge of this paradigm and related terminologies. This section helps the reader to understand basics of cloud computing.

2.3 Message Oriented Middleware

MOM (Message Oriented Middleware) is a middleware that is more commonly used to transfer messages synchronously or asynchronously in a distributed environment. It allows loose coupling between different nodes or components and makes communication possible between applications running on heterogeneous platforms, different operating systems, and network protocols. Various components are loosely coupled in a way that they can communicate with other

components without knowing the location of other components, independent of message production or consumption time, communicate format or protocol and last but not least independent of platform or environment. If these characteristics hold for a system, then connected components can operate on 'Send and Forget[5]' approach. A sender can send a message to other recipient components without caring whether the recipient is listening or not, the recipient is accessible via the same location or been moved. The sender will hand the message to MOM and MOM will take care of that piece of information to successfully deliver it to the recipient. Once the message is delivered to MOM, it will determine itself when and where to forward it. MOM acts as a mediator between a sender and a receiver, assists them to know very little about each other and still makes it possible to communicate. An arbitrary number of senders or receivers can connect with this middleware to participate in the communication. Along with all its benefits, it has some drawbacks as well. MOM will be unable to deliver messages if it gets crashed due to some incident and it will destroy communication between all its connected components. This drawback can be resolved by having a backup instance [12, 14].

2.3.1 Communication Modes

Communication with message based systems or MOM is possible in two distinct modes, i. e., Synchronous and Asynchronous which are explained further in following paragraphs:

Synchronous communication [13] blocks the sender component till the receiver respond it back. The sender sends a message to a middleware instead of direct receiver and waits till it is received, processed and responded back by the receiver. The sender cannot continue to process any further task or send more requests until already sent request is responded. There are many design alternatives available which can be applied to use synchronous mode of communication according to the requirements [14]. The sender is blocked until a message get **a)** received by the middleware on the sender's side **b)** received by the middleware on the receiver's side **c)** successfully delivered to the receiver or **d)** processed and responded by the receiver.

Asynchronous mode of communication [13] is a non-blocking communication mode in which sender can continue to work after sending a request or message to a receiver via middleware. It will not wait for a response from the receiver. However to accept a response from the receiver at some later point in time the sender needs a callback which alerts it whenever a response arrives from the receiver end. Or sender can use polling mechanism as well to get any response message from the receiver. Fig. 2.4 shows a clear difference between both modes of exchanging information.

[5] http://www.enterpriseintegrationpatterns.com

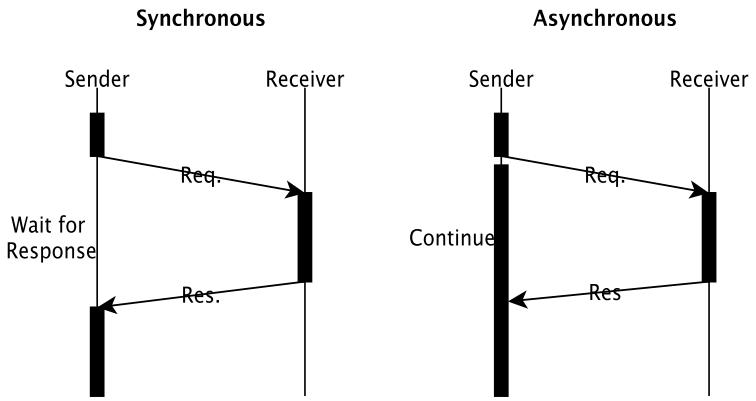

Figure 2.4: Synchronous vs Asynchronous Communication [13]

2.3.2 Message Structure

Before explaining two flavors of MOM, it is important to understand the structure of the message that flows between sender and receiver. Each message essentially can be broken into three parts, i. e., header, properties, and body. The header is the required component because it defines the route of the message as well as the structure of body itself. Properties consist of optional fields, which are added by the client as per needs. The body is the part of the message where payload goes, or it contains actual data that should be sent to other party [12].

2.3.3 Point to Point Messaging System

Point to point (P2P) messaging systems are an important paradigm of MOM where a sender sends a message to the middleware, and only one receiver will consume it, although multiple receivers or consumers can listen for incoming messages. There are multiple patterns to implement this system which are discussed below:

- One-to-One, involves only one producer and one consumer
- One-to-Many, one producer sends a message, whereas multiple consumers can listen but only one will receive a message
- Many-to-One, multiple producers can generate and send messages but only one receiver will consume all of them
- Many-to-Many, which involves multiple producers and multiple consumers

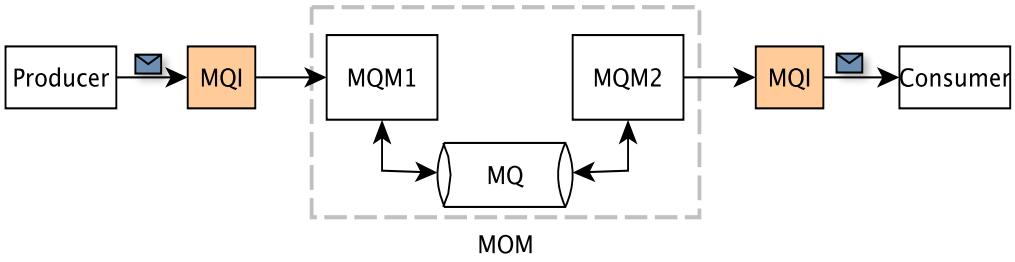

Figure 2.5: Point to Point Messaging System

Fig. 2.5 explains a very brief overview of P2P messaging system architecture [12, 14]. A message-oriented middleware can have multiple MQMs (Message Queue Managers), which can be accessed by clients with the help of MQI (Message Queue Interface).

The producer that produces a message accesses MQM through MQI. It specifies a queue to which message is targeted. And consumer retrieves this message and consumes it by gaining access to MQM also through MQI. MQI is an interface that provides methods that can be used to connect and disconnect a program to an MQM, give access to a specific queue, put and get a message from a specific queue, release associated resources, change attributes and properties of a queue, etc.

To convey a message between producer and consumer an MQ (Message Queue) is used as a media . It can operate in two ways, i. e., Persistent and Nonpersistent. The management of the messaging system is owned by MQM or QM (Queue Manager) which is a kind of database that provides an environment to manage it as DBMS (Database Management System) manages database system. MQM is used to create and delete a queue, update attributes, or properties of a queue, start and stop a queue, manage security, and performance, etc. A queue manager can have local message queues which can be used between a sender and receiver connected to the same MQM. MQM can also forward messages to other queue managers if the receiver is not listed within its environment.

2.3.4 Publish Subscribe Messaging System

There are multiple use cases in which a message should be sent to multiple recipients instead of only one, i. e., if multiple consumers are interested in the same piece of information. In order to cover these cases, a different message paradigm is available called publish-subscribe or in short Pub/Sub messaging system. For such scenarios, P2P systems (although they can be used) are not efficient and should not be used. In Pub/Sub messaging system, a producer is named as publisher and consumer as subscriber. A publisher can publish a piece of information which is then delivered to all recipients or subscribers who are interested in getting that information.

Any number of subscribers can subscribe to get particular information. Let us take an example of a stock market where many users or parties are interested to know the current stock value of a specific company. As soon as the stock values of a company are published, they are delivered to all interested parties. There are patterns available which also allow filtering a message to get more specific information. For example, if the stock values of three companies are published as one message, but subscribers are not interested in receiving stock values of all three companies. The message will be filtered by the messaging system and then delivered to the recipients. These systems are best suited for information-driven applications. In Pub/Sub systems, messages are also known as notifications. In the context of publish/subscribe systems, messages and notifications are used interchangeably.

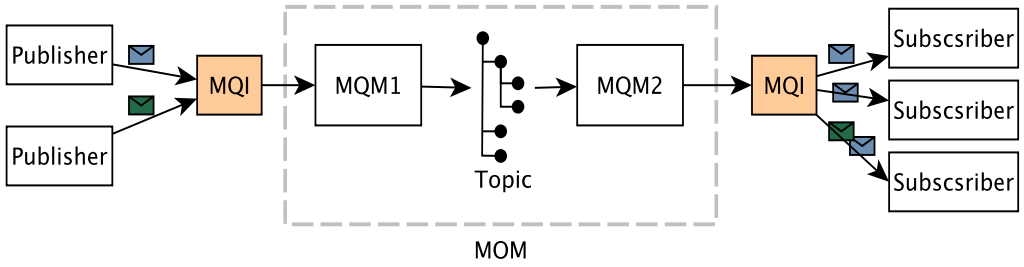

Figure 2.6: Pub/Sub Messaging System

Fig. 2.6 presents an overview of the architecture of publish/subscribe based messaging system [12, 14]. It consists of a publisher that can publish messages to a specific topic in the messaging system after registering to that topic. It accesses the messaging system through MQI. And a program named subscriber can retrieve messages from the messaging system sent by publisher. Before it can receive any message, it should subscribe to a specific topic or a sub-tree of a topic. This is called subscription. As soon as a message is published by a publisher on a specific topic, it is filtered (if applicable) and forwarded to all subscribers whose subscription matches with the incoming message. Like publishers, subscribers also access pub/sub messaging system via MQI.

A topic is an entity in Pub/Sub system that categorizes the messages or notifications published by publishers. A tree-based hierarchy represents a topic, where a sub-tree is also a (sub)topic and represents a specific categorization of information. The subscribers subscribe to these topics or sub-topics to get notifications according to their interests that are published on them.

2.3.5 Summary

Message-oriented middleware is commonly used for conveying information or messages in a distributed environment. This middleware can also be used to transfer data from on-premise

source database to an off-premise target database. Therefore reader should have a good knowledge of this middleware along with its two communication modes, i. e., synchronous and asynchronous and knowledge presented in this section is enough for the reader to get ready for the next chapters.

2.4 Data Replication Strategies

This section describes available data replication strategies which can be used to copy data from source to a target database. Depending upon requirements, any one of them or all can be used simultaneously to keep source and target in synchronization. Two of them have been extensively discussed over the years in the software industry, incremental refresh, and bulk load. These both strategies differ in concept, design and usage applications. Incremental refresh as the name suggests, transfer changes made in transactional source databases incrementally one after another. Bulk load, in contrast to incremental refresh, is totally different in concept which loads data in bulk to replicate. These two main data replication methodologies are explained here in more details.

2.4.1 Incremental Refresh

In many cases analytical applications need to generate business or analytical reports over fresh data and therefore require real-time or near real-time data replication. Such applications can not work with bulk or full data loading strategy, as loading entire data to refresh target database takes hours and usually executed after specific intervals or in out-of-business hours so that no transaction source gets affected. For these sensitive applications, IR (Incremental Refresh) comes into play and helps them to keep data in synchronization with their source databases. The basic idea behind incremental refresh is to capture changes from data sources as soon as they are made and send them to target end. The concept is totally different than bulk load refresh where full tables or partitions are captured and sent. As only changes are captured in case of incremental refresh, data to be transmitted is really small. This small size of data makes this replication strategy very efficient for time critical applications. But there are few design considerations that should be taken into account while choosing incremental refresh. One of them is to discover and capture changes as soon as they are applied in the source database.

Fig. 2.7 shows basic components that are involved in incremental Refresh.

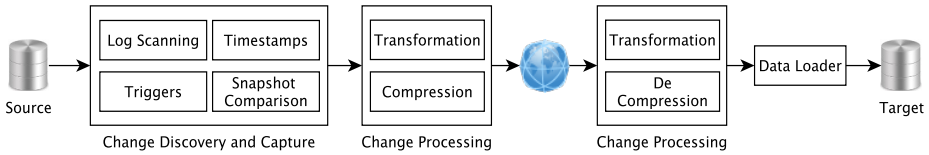

Figure 2.7: Incremental Refresh Components

2.4.1.1 Change Discovery and Capture

Discovering changes in data source incrementally are of utmost importance in the incremental refresh. There are some change discovery techniques which try to find out recent changes in data sources with minimal impact on performance because these techniques can put extra load on the transactional database. Selecting an optimal change discovery technique is very important in the case of incremental refresh. Following are the most common change discovery and capture techniques:

- Log Scanning
- Triggers
- Timestamps
- Snapshot comparison

In following paras, they are explained with further details.

2.4.1.1.1 Log Scanning Some but most renowned DBMS system like IBM DB2, Oracle Database, MySQL, etc. generates transactional logs for backup and restoration purposes. Every DML operation that is part of a transaction or not is written to these logs. If some failure occurs or DBMS system suffers a crash, there could be chances of data loss. To avoid such losses and to display ACID properties, DBMS systems uses these internal logs to restore last stable position. If these logs are supported by the database system, then they can be used to detect and discover current changes in the source databases [15]. Log scanning provides a lot of benefits i. e., changes can be detected without interrupting original database schema so there will be no load on the transactional database in case of reading log files. There is no need to change source schema to detect these changes. However, source database system should provide read access to its internal transactional logs. Otherwise, log scanning is not possible.

The industry has accepted this technique widely because of its intervention-less data capture ability. Many enterprises have developed products which are based on log scanning. For example, IBM Infosphere Change Data Capture [16], which can scan transactional log files produced

by DB2 database, capture changes and apply them to target end. Along with IBM, Oracle [17] and Microsoft, etc., also makes this technique available as change data capture to their customers. Oracle CDC (Change Data Capture) product operates in two modes i. e., Synchronous CDC and Asynchronous CDC. The asynchronous modes capture change data from log files once changes have been committed to the source database.

2.4.1.1.2 Triggers can also be used to discover changes in the source database. They are stored programs which are automatically executed upon some action. They work on action vs. reaction principle. Whenever there is a DML operation on the database, a corresponding reaction or trigger is performed. They have the capability to copy and write data to some other table or file as a reaction to DML (Insert, Update, Delete) operations. For example, if an insert statement is fired upon source database, triggers will be executed and copy all data of this insert statement to another place, a table, or a file. This sort of mechanism can also be used to synchronize source data and target data. Any change is detected and written by triggers in a copy location, from where it can be moved silently to target end [15]. But using triggers have some performance issues as well. First, they put an extra burden on the database system to fire corresponding triggers every time a DML statement is issued. There can be performance issues because of using triggers. Secondly changes needed to be made in the source schema to use triggers, if source schema does not already have them.

This type of change discovery and capturing methodology is implemented by Oracle CDC in the form of synchronous mode. The synchronous mode uses triggers on the source database to capture change data. It claims to have no latency because of continuous and real-time data discovery from source data. But this synchronous mode implies extra overhead on source database because of triggers and can be a cause of performance issue [17].

2.4.1.1.3 Timestamps is another approach to find changes in the database. The concept [18] of this approach is to assign a time-stamp field to each row and then compare this field to discover new changes. Whenever a new row is inserted, updated or deleted in the database, time of insertion, update or deletion is added to this field. This time-stamp field of each row is compared with the time at which changes were captured last time. If there are any recent rows found, they should be copied and replicated to target end. The drawback of this approach is the scanning time. Each time replication necessary, a full scan over all the rows is required to detect recent changes in the database. Similarly, if the schema does not already have this column in required tables, it should be modified to hold time-stamp field.

2.4.1.1.4 Snapshot Comparison supports replication by finding differentials between multiple snapshots taken at different times. It works on the principle of taking a periodic snapshot of source data. These snapshots are actually files containing data of source tables. When replication is required, a new snapshot is taken from source data and then compared with a previous

snapshot. If there are some differentials found, that means source data has been changed which can be detected and discovered based on some heuristics or algorithms. One such algorithm is Window algorithm [19] which operates on two snapshots and maintains a moving window of records in main memory and find out recent changes based on differentials found in two snapshots.

2.4.1.2 Change Processing

Data, once it is read and captured from source databases, is in its raw format and not clean. It can not be sent to target database without its intermediate transformation and cleansing. Before that data is transmitted over the wire, it should be processed to meet the requirements of the target database. In this stage, data is cleansed and transformed to an acceptable format for the data warehouse.

2.4.1.2.1 Transformation is the process to perform a series of steps or operations on the data before it is loaded into the target database. Usually data, after reading from data sources, is in raw format which is not suitable to apply directly to the target. In case, data is coming from multiple sources and goes into common data warehouse for analytics there are a lot of conflicts expected due to an independent database schema. This transformation [20] process resolve these conflicts or introduce new values by operating different steps on the data. It performs essential transformations on data, for instance, encoding conversions, translation of encoded values, derivation of new aggregated values, joining of data, format modification, type and value conflict resolution, etc.

2.4.1.2.2 Compression is an optional part of incremental refresh but if used/enabled helps to lower overall size of data being sent on wire or fiber and hence lowers the latency of the system and increases real-time data replication performance. Any suitable compression technique can be applied here keeping in mind that compression time is less than transfer time of data in its raw form that is without compression. If compression process is taking more time than transfer time, then it can become a bottleneck for the performance of the system. After this part, data is sent over the wire to the target system.

2.4.1.3 Data Loader

Once data arrives at target system, it is time to apply this data to corresponding target databases respectively data systems. This responsibility is performed by data loader located at target side. It takes incoming change data and applies it to relevant target tables. It also has the capability to forward it to some message queuing system.

2.4.2 Bulk/Full Load Refresh

If data that is to be transferred is very huge and is counted in GBs and TBs, incremental refresh based approach fails to replicate data in real-time and therefore should not be used for such a large amount of data. However, there is another data replication strategy available that does this job pretty efficiently than incremental refresh, and that is called BR (Bulk Refresh) or FR (Full Refresh). It has a capacity to successfully transfer huge chunks of data from source system to target system. It is a good approach to use bulk load refresh if a transactional source database contains a huge amount of data and is being replicated for the first time to the data warehouse or target system. Once initial replication is completed, leverage IR system to synchronize further changes in source database with target database in near real-time.

Tables that are needed to be archived for future reference and are very big in size should be copied using bulk loading. This technique is much faster than incremental refresh when replicating bulk amount of data. But there are some drawbacks as well. For example when bulk load exports data from a source system, performance of database goes tremendously down because it copies data from physical tables. In contrast incremental refresh has no performance effect on source system because it reads log files. Due to its performance effects, bulk load based refresh can not be applied so often to keep source and target systems in sync. Another drawback of using such a system is that even if there are very few changes in the table made, the entire table will be replicated that will put an extra burden on network traffic.

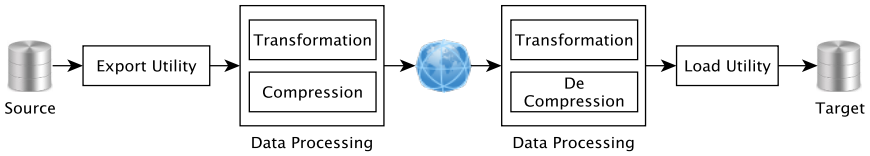

Figure 2.8: Bulk Load based Refresh

Fig. 2.8 shows an overview of the architecture of this replication system. First of all data is exported from data sources using export utility, processed for cleansing and transformation and then loaded into target database using load utility. It is obvious that many components of this architecture like transformation and compression are common to IR and are already explained in Sec. 2.4.1. The details on missing components can be found below.

2.4.2.0.1 Export Utility It is a program or a process that is used in case of a full refresh to export data from source databases. With the help of this export utility, data can be unloaded from changed tables or partitions. Before running this program, a connection is established to the database from which data is to be unloaded.

2.4.2.0.2 Load Utility is a program that is used to load a huge amount of data (which is unloaded from source database using export utility) into the target database. A connection is required with target database. Once the connection is established with respective database, data can be loaded into it in corresponding tables and partitions.

2.4.3 Summary

Incremental and load refresh are the two very commonly used techniques to perform data synchronization between source and target data sources. The main focal point of this work is to optimize these techniques using performance metrics to replicate data from on-premise transactional data source to a data analytical cloud service. The reader should have a sound knowledge of both techniques because they are the base of our work presented in the conceptual model.

2.5 Performance Metrics

This section explains important performance metrics that are relevant to our design concept. Based on these measures performance of a data replication system can be calculated, whether it is working fine or does it need some more optimization. Analysis of system with these metrics enables to debug its particular part. In the following chapters, a mathematical model is constructed which can be emplaced to measure these factors for different components of a replication system.

2.5.1 Latency

Latency, denoted as L in our system, in its very basic definition is the time delay between the cause and effect of some physical change in the system [21]. Latency describes how long does it take for a data packet to reach from on point to another. To get optimal performance from the system, the focus should be to decrease this measure to its maximum extent. High latency of some component depicts more time a component takes to finish its job. Instead of analyzing entire system as a unit to calculate latency, it is measured by splitting a system (i. e., a replication system) into smaller units or components. This approach will enable system architect to analyze and debug each component in more details and assist in finding out which component is the bottleneck for efficiency.

In case of incremental refresh, it is the time delay in measured unit of time, i. e., seconds or milliseconds between a change(s) is logged in log file at source side and then applied to target database that is a DML (insert, update or delete) query make some changes in transactional

source database during OLTP and database system in returns updates log file which is then duplicated to target system. In an ideal case, latency should be minimum to realize near real-time replication of data or synchronization of source database and target databases. Each component in incremental refresh adds its overhead towards latency, so it is of utmost importance to make every single component in this technique clever enough to process its job efficiently.

Similarly, latency in bulk/full load based data refresh system is the time taken by this process when it was first started to capture a table or partition from source database till it completes its job by copying entire table or partition to target database. Usually latency of bulk refresh is much higher than its counterpart because of the fact that in case of bulk load changed tables or partitions are entirely copied to target, and that constitutes to enormous amount of data in contrast to incremental load which synchronizes only changed rows and hence drops a lot of overhead while discovery and capture process.

2.5.2 CPU Usage

CPU Usage, represented as U is the amount of time measured in clock ticks or seconds that a process takes while being executed by central processing unit (CPU). The time in which a process is idle, i. e., waiting for some IO (Input/Output) resource or user input, or even when it is blocked by high priority processes and waiting in a queue for its turn to use processor again, etc., should not be included in CPU usage. In such situations, the process is idle, while doing nothing and waiting for some event to occur or its turn to come for execution. CPU usage is what that processor takes to run all instructions of a process. As of latency, CPU usage is also a very important factor to measure the performance of a system, especially replication of data is a very CPU hungry process. Therefore it is important to measure CPU utilization for each component of incremental refresh as well as full load. The cost model of CPU usage determines its value not only for the entire system but individually for every component as well. The cost model presents a mathematical model that will help to ease this analysis and assists to debug any replication system, as mentioned earlier, in more details. Another important point is that CPU utilization, usage and time means the same thing and will be used interchangeably.

2.5.3 Throughput

Throughput R is the amount of data that is processed or transferred by a system in the given interval of time. It is the measure that determines the capacity of the system to capture, process and apply changes. How fast is the system to synchronize both source and target parties is measured based on throughput. In the case of incremental refresh, it is the rate of changes at which they are collected from source and are applied to target databases in a specific amount of time.

For bulk or full refresh replication, throughput is defined as the rate at which all changed partitions or tables copied and replicated to target end. Like incremental refresh, additional components like transformation and compression also determine this rate. Based on this metric one can calculate that given a replication system how fast synchronization process between source and target should be finished, and whether such a system is capable of handling required amount of load.

A mathematical model that will be developed in the course of this work will help to measure throughput for a replication system.

2.5.4 Cost Effectiveness

The cost-effectiveness of the overall system tells how efficient and useful is a system given a certain set of execution conditions and input parameters. The already calculated parameters are applied to calculate CE (Cost Effectiveness) of our system. All three aforementioned core parameters, i. e., latency, CPU usage, and throughput contribute towards its measurement.

The mathematical model presented in the cost model helps to calculate cost-effectiveness once core measures for the system are calculated. Based on CE the feasibility of the replication system can be determined and can be estimated if it is beneficial to use it further under given input circumstances. After estimation, if cost-effectiveness turns out to be below some threshold then the system might need some new orientation in the architecture or change in input to make it more feasible.

2.6 Monetary Metrics

This section provides an overview of financial aspects of using replication strategies and tries to figure out the factors which affect them. These metric(s) belong to a different dimension than performance metrics but dependent on them along with the architecture of the system. The inclusion of unnecessary components in the system can badly impact monetary measures, and these affects are taken into consideration in this section.

Data replications system depending upon requirements needs a lot of infrastructures, and that requires money. This infrastructure cost gets multiplied when scalability is also a requirement for a stable system. Pricing tells that how much funds are needed for the entire system. This measure will give the customers a rough estimate that how much they need to pay for such a system given a particular set of changes in case of incremental refresh or how much money they need to spend for replication of changed tables or partitions in case of a bulk load.

In our case, replication system is much complex where source end is hosted with on-premise infrastructure and target end is hosted on off-premise infrastructure using some public cloud services. Each public cloud service provider have their own pricing model. Some providers charge bills based on the time that how long a system is running, while other based on data usage and or based on CPU usage. With our pricing model, the customer can make a rough estimation for each component of the system as well as for the entire system. Total pricing is dependent upon performance metrics, i. e., latency, CPU usage, throughput. For example, if for a given replication system and change dataset, latency is extraordinarily high, the system should run for a longer period and that impacts pricing.

Chapter 3

Related Work

This chapter presents technical work that is already done and somehow directs to the contribution of this thesis. As it is already described in the introductory chapter that replication strategies are not new, they have been used extensively to transfer data from on-premise transactional source database to on-premise target database. The architecture of IT resources is changing tremendously due to the arrival of cloud computing paradigm where companies can get rid of their entire IT infrastructure by hiring IT services from cloud providers, or they can have a mix of their infrastructure along with some borrowed resources from cloud providers, see Sec. 2.2. If companies use on-premise transactional data sources while hiring analytical services including data warehouse from cloud providers, there is a need to replicate data from on-premise transactional data sources to off-premise analytical services. In this chapter, existing technical contributions that try to optimize data replication techniques with the help of performance metrics, are presented. If some technical work evaluates any replication strategy either incremental or full refresh or both for performance, this chapter will try to cover it and find out if there are any missing metrics that are important to our contribution.

Companies that hire cloud resources or services have to pay for their usage. This thesis is also contributing towards the monetary aspect of replication strategies and in this chapter the existing work that compasses financial outlook of using data replication techniques is found out. In short, any existing technical work, paper, writing or thesis that attempts to evaluate or optimize these data replication technique (while applying them to on-premise, off-premise or hybrid infrastructure) using performance or monetary heuristics is the part of related work for this thesis. In Sec. 1.2 it is mentioned that compression and transformation of data while replicating it from an on-premise data source to off-premise target service is also considered, so any technical work or contribution that measures the performance of these processing units is also mentioned.

3.1 Performance Metrics Modeling

Data in source transactional databases is continuously updated with very high speed and it should be transferred to target databases with the same rate in order to keep data warehouse fully in synchronized state. Data replication strategies, i. e., incremental and full refresh are the backbones of transferring data and the synchronization state largely depends on the performance of these strategies. Otherwise, analytics or BI reporting performed on data warehouse will be outdated and might result in antiquated business decisions. A summarized overview of some technical contributions that demonstrate the evaluation and modeling of performance metrics with regard to incremental and full refresh is presented here.

3.1.1 Performance Estimation of Incremental and Full Refresh

To calibrate a replication system for optimized performance a very nice white paper [22] is formulated by IBM (International Business Machines Corporation) which discusses performance affecting factors in detail for most common data transfer techniques, i. e., incremental and full refresh. This article takes CPU time and latency into account and describes how one can calculate these two parameters while tuning their system. For example, in case of incremental load or bulk load, they have measured total CPU time to replicate data in bytes and provides nice formalization of CPU time calculation. For incremental update as well as bulk load, after measuring net volume of changes using NUM_CHANGED_ROWS * AVGROWLEN, calculation of total CPU time is described in the article. For incremental update, estimating total number of inserts, updates and deletes is very important to estimate actual CPU time. Similarly, in case of full refresh, it is very important to detect which tables or partitions have been changed over time, because only these modified partitions and tables needed to be replicated. And assessing the total size of altered table or partition is also very important to actually measure CPU time.

Although this paper provides valuable information to measure CPU time, it does not deal with factors affecting this time. There are multiple stages involved to carry out data replication using incremental or full refresh strategies, e. g., fetching data from data sources, transformation, compression, loading data, etc. It does not dealt with questions like how much CPU time is needed for individual step involved in carrying replication, how to figure out which particular step is becoming a bottleneck for the system.

This article [22] goes one step further to calculate latency of the system. As suggested in that article product specific latency is included in the XML (Extensible Markup Language) code returned by a specific stored procedure. If customers are using this IBM product, they can easily detect current system latency without any hassle. Again further details are missing this detection procedure. Latency is actually dependent upon multiple factors involved in different steps. There is no information provided how to calculate latency for individual step. Which

factors will affect latency? What can be the best possible set of configurations to get an optimal solution and high performance for the system? The paper also discusses in summarization part about throughput for different refresh mechanisms, but details are missing. How to detect throughput for every single component and which measurements can be taken to improve it?

Another important scientific paper [23] compares and evaluates the efficiency of available and well-known open source ETL tools. Authors takes following performance indicators into account in order to present their results:

- Execution Time
- CPU Utilization
- Memory load

The metrics mentioned above for different ETL tools are evaluated, and results are presented in graphical manners. For some open source ETL tools, if only performance comparison is needed this paper can be referenced. But this paper neither includes measurement of above-mentioned parameters nor present any formula to calculate these metrics. A fully comprehensive and extensive study of these parameters is needed to be conducted to present them in a formalized approach so that results can be used to gauge data loading techniques for performance factors such as latency, CPU Utilization, Throughput, etc. My work will dive into these details and compass maximum possible metrics. Measurement of these parameters become more important when one is using near real-time or zero latency data replication techniques for a zero-latency data warehouse architecture as presented in this [24] piece of work.

Another attempt is made by Eccles [25] in his doctoral work on pragmatic development of service-based real-time Change Data Capture (it is an approach similar to incremental refresh to data integration that is based on the identification, capture, and delivery of the changes made to enterprise data sources [26]) in order to evaluate his implementation approach with already existing conventional approaches. In chapter 6 of this report, Eccles describes and measures three important performance metrics, i. e., 1) transaction rate, 2) capture latency, 3) CPU usage. He provides a calculation of transaction rate as the part of his Ph.D. work but does not conclude a specific formula that can be used to measure this transaction rate. As far as latency is concerned, Eccles only calculates capture latency which he define as the time measurement between two points of references: a start time and an end time, where the start time, he defines as, is the point at which change is made and the end time is the point at which a change reflects to change table. In his work, he also presents some concrete equations to measure capture latency which are not capable of calculating latency for each separate component in case the system is needed to be debugged for performance reasons. Similarly, CPU utilization is defined and calculated for his work, but never formulated to measure it for an individual component.

Another piece of work is presented by Pareek [27] in his industrial paper as part of Oracle. In this paper, he measures some key non-functional features of CDC-like system throughput, IO overhead, and CPU utilization. As part of his work, he measures these metrics and represent them with graphs for comparison but does not present any formula or methodology to actually calculate them.

A framework is already presented in [28] to analyze data freshness. Data freshness describes how old is the data with respect to the user perspectives? How older are the current reports generated over data warehouse from its transactional data sources? As part of his contribution, he comes up with many definitions to measure data freshness of the system. Using these definitions, the quality dimension of a system can be analyzed. But still, there is a need that how these definitions should be applied to the system? For example, this paper discusses some freshness factors like currency and timeliness but does not explain anything about the calculation of these metrics.

A lot more contributions are presented along with aforementioned works that all geared towards the same topic, but none of them is so discrete that actually depicts the measurement of performance factors. One such attempt is presented by Raitto [29] to measure latency while discussing Oracle Change Data Capture product. But he in his work does not explain how latency is measured? There is a gap between using data placement strategies, analyzing and optimizing them according to the needs. Few authors go along this dimension but end up analyzing only the part that is relevant to their work.

3.1.2 Performance Estimation of Data Compression

Data compression is extensively explored in many technical contributions because it can reduce data in size and possibly increase performance of different systems in many cases. For example, it helps to optimize the access time of websites by compressing its contents especially the content with MIME-Type [30–33]. And some technical work encompasses improvement of throughput of IO forwarding with data compression [34]. These technical writings differ from this thesis where it tackles data transfer from transactional databases to target database located on cloud. It deals with the performance of compressing different changes made in the database over the time.

3.1.3 Monetary Aspect of Replication

While transferring data from on-premise transactional data source to an off-premise analytical cloud service using available data replication strategies, pricing also becomes very important factor to measure. The cloud services are not free of cost; rather they are billed per their time

and usage. In the case of hiring data replication services from cloud providers, another factor that plays towards pricing is how much data is being transferred per unit time.

To estimate the cost of using these replication services, all cloud providers have their own pricing model along with corresponding pricing calculators[1] [2] which can be used to measure actual cost of the replication system.

3.2 Summary

Although a substantial amount of work is already done in this area of performance measurement and cost optimization of data replication strategies, but a more concrete, well elaborated and sophisticated approach is needed which can be applied to boost overall system efficiency by detecting and measuring individual component's performance metrics. Many solutions, as already presented in this Sec. 3 try to focus on these metrics but somehow fail to elaborate actual measuring approaches. In short, a comprehensive study is needed, which is the focal point of my thesis, to provide an answer to the following few concerns for most common data replication techniques:

1. How much latency is being added by individual component of the system?
2. How much CPU is required by a particular component?
3. What is the throughput of each component?
4. How to calculate cost effectiveness of the system?
5. How to calculate price of a selected replication system?
6. How to optimize system components with respect to price?
7. ...

A concrete solution to these questions depends on a system as well as upon individual component; however, we will try to come up with a more generic solution that can be applied to any data replication strategy and can be used to enhance system efficiency by removing performance bottleneck in an individual component. For example, my contribution includes handy tools to calculate performance metrics and tells that which component is causing high latency, more CPU utilization, and less throughput.

[1] https://cloud.google.com/products/calculator/
[2] https://calculator.s3.amazonaws.com/index.html

Chapter 4

Conceptual Model and Design

This chapter is the heart of this thesis. First of all, a layered architecture of a system with synchronization in mind is presented which gives an idea to the reader about overall system architecture. This architecture covers almost all essential components of a system including transactional source database, replication, communication, target database, etc. In the next step, the important and relevant performance metrics are defined, because the replication approaches are evaluated on the basis of these metrics. In the next section, we will present a common and unified approach of two already in Sec. 2.4 explained data replication methods. Using this general approach, performance evaluation of replication strategies can be much simplified. Each component of this approach along with the affect of performance metrics is explained here. For each component of this unified model, performance metrics are calculated with examples where applicable. Last but not least, monetary evaluation of this system is explained as well.

4.1 Architecture of Analytical Systems

This section describes a layered architecture for a general analytical system which basically explains necessary components in each layer on the source as well as target side. It is the common architecture of an analytical system that involves operational data sources, analytical service including a target database or data warehouse, a replication strategy to move data from source to target end, and/or communication methods between both sides. The idea presented here can be applied or generalized as per needs to any analytical system that involves a replication strategy. The two models of this architecture and related components of this system are explained below. First of all the source components or layers are explained, then communication layer and then finally target components. This direction is same as of data movement from source to target end.

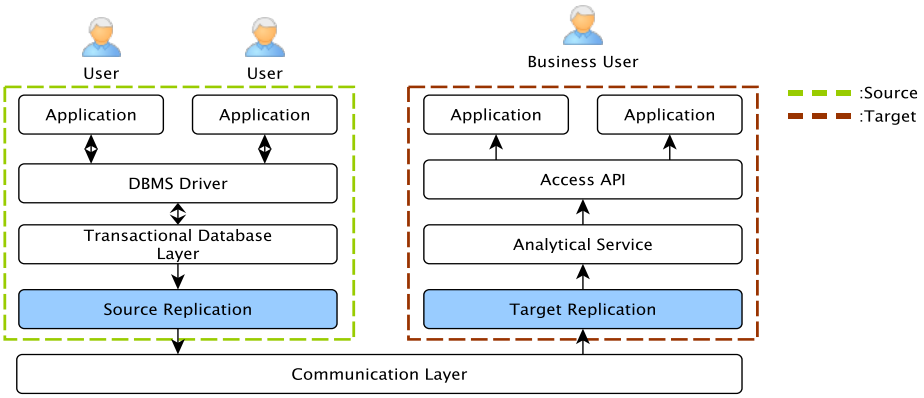

Figure 4.1: General Architecture of Analytical System

4.1.1 On-Premise Analytics Model

This is a traditional approach of performing analytics on data using on-premise analytics infrastructure. The hardware and software components needed for analytics are kept with the same infrastructure where transactional data sources are located and are managed by the host itself. Being on on-premise, it enjoys the benefits of the local network that makes replication much simpler and faster from data sources to data warehouse on which analytics are performed. Fig. 4.1 depicts different components or layers of this system:

4.1.1.1 Application Layer

Application layer hosts all applications that provide business functionality to the users. It can be a desktop or web application depending upon business requirements. Users can log into the applications and perform day-to-day transactions. The application layer is directly connected to the operational database for storage. The connection between application and database is handled via special purposed APIs also called DBMS drivers.

4.1.1.2 Transactional Database Layer

This layer represents all operational or transactional databases that are directly connected with the applications to store application, user, or daily transaction data. It can be a relational or non-relational database system with support for data replication. For example, in the case of bulk load, the database system should allow copying full table or partition. And in the case of the incremental refresh, database system should write some transactional logs which are the basic source of data for incremental replication approach.

4.1.1.3 Source Replication

Source replication contains all components of replication system that are required at source end to gather data from data sources. Incremental refresh requires a log reader that reads logs of transactional databases along with parser to parse these log files. For bulk refresh, an unload utility is required to copy data from respective table or partition. After data is gathered from source databases, it is sent to communication layer.

4.1.1.4 Communication Layer

It is responsible for carrying data from source end to target end. Any communication, as per requirements, can be used to transfer data. A bare TCP connection, synchronous or asynchronous message-based replication and/or web services can serve as the data carriers. The selection of communication protocol or middleware is based on the requirements, type of replication and the amount of data being transferred.

4.1.1.5 Target Replication

This is the layer that consists of components essential to perform data synchronization/replication at the target end. The main responsibility of these components is to take data from communication layer, process it further and then apply it to target database, data warehouse or forward it to some other service. For incremental refresh, it will apply incoming changes into respective target tables, and in case of a bulk update, a full table or partition is refreshed in target database.

4.1.1.6 Analytical Service

Analytical service maps to the components needed to perform analytics, workhorses to process OLAP queries including data warehouse. This service is connected with analytical applications with an access interface. The applications or clients can send OLAP queries via this interface to analytical service. The service processes these queries and returns the query results. The queries that these service processes are generally very big and complex which process a lot of data.

4.1.1.7 Target Application

Target application is the layer consisting of applications or clients used to display OLAP query results and interprets raw data in graphics so that the business users can get valuable information from it and take critical business-relevant decisions. Using these clients, analytical queries can be forwarded to analytical service and the results are then displayed with graphics and in a comprehensive format which allows deciphering future business trends and decisions.

4.1.2 Analytics-as-a-Service Model

AaaS facilitates to use analytics benefits without hosting a single component needed to gear analytics on an on-premise infrastructure and allows to perform analytical query execution at the cost of its usage price, see chapter 1 for more details. In AaaS model, OLAP queries along with data from transactional data sources hosted on local on-premise infrastructure are sent to an analytical service hosted by a public cloud provider, which then executes these analytical queries on this data and sends back its results to the clients. This kind of service provisioning is the part of hybrid cloud deployment model that we have discussed in detail in Sec. 2.2. In hybrid cloud deployment model, a mix of public and private cloud infrastructure is used so that clients or tenants can avail public services while keeping very important and business critical IT components in their private sphere.

The architecture of analytical systems presented in Fig. 4.1 for traditional private deployments can be generalized for AaaS model as well. The only difference is the premises where this infrastructure is deployed. In the case of traditional or on-premise deployment, both source, as well as target stages, are deployed privately and customer itself is responsible for its deployment and maintenance. While in the case of AaaS model, target components (that include replication services at target end, analytical service, data warehouse, etc.) are managed by some public cloud provider and source components (including transactional data sources, replication services at source, etc.) are still hosted on private premises. The communication layer that is responsible for carrying data from source end to target end can be a private network or can be shared among other tenants as per needs. So there is not much difference between AaaS model of analytics and traditional approaches, and it lies in where these models are hosted.

Now that we have basic knowledge of system architecture used to dump data differentials to target database from source database, it is time to dive deep into details to evaluate replication systems using performance metrics.

4.2 Replication System Modeling

Incremental refresh and full refresh are two prevalent replication strategies that are frequently used to transfer data between source and target systems. The goal in this section is to model these systems in a way that later they can be evaluated using performance metrics. The fundamentals of both replication methodologies is to transfer data, so they share some common components. First of all, this commonality between two replication strategies is determined and then a general model is developed which can be used to tackle any replication system. Later in this section, a pattern is explained that can be used to assess each stage of our general model. This pattern is applied to evaluate each stage using performance metrics.

4.2.1 Unified Pipeline Model

In this sub-section, a model is explained that consists of essential components for data replication. This model serves as the basis to assess performance metrics on each stage. Although the basic approach and idea to replicate data from source to target is different in incremental refresh as well as bulk refresh, but they use common set of components in their infrastructure. By using this commonality property, a unified replication system architecture is developed which should serve for our requirements to evaluate individual component. The very first requirement of a data replication or synchronization system is to gather changes from source system and this functionality is common in any kind of such system. Replication systems usually also make some transformations over the captured data and based on requirements data should be compressed to save network bandwidth. The final step in all data transfer system is to apply incoming changes into target database. Based on these assumptions, a set of common components is concluded which is the intersection of components of IR and BR systems.

Set of IR components = {Source DB, Log Files, Data Reader, Transformation, Compression, Transmission Service, De-compression, Data Loader, Target DB}

Set of BR components = {Source DB, Table, Partitions, Data Reader, Transformation, Compression, Transmission Service, Decompression, Data Loader, Target DB}

Set of common components = {Source DB, Data Reader, Transformation, Compression, Transmission Service, Decompression, Data Loader, Target DB}

Fig. 4.2 refers to such a generic system which is totally based on common components. It consists of different paths for a synchronization system. Each path depicts a fully capable replication system with some functionalities enabled or disabled. This path-based generic replication system enables a customer to choose a particular path as per their requirements.

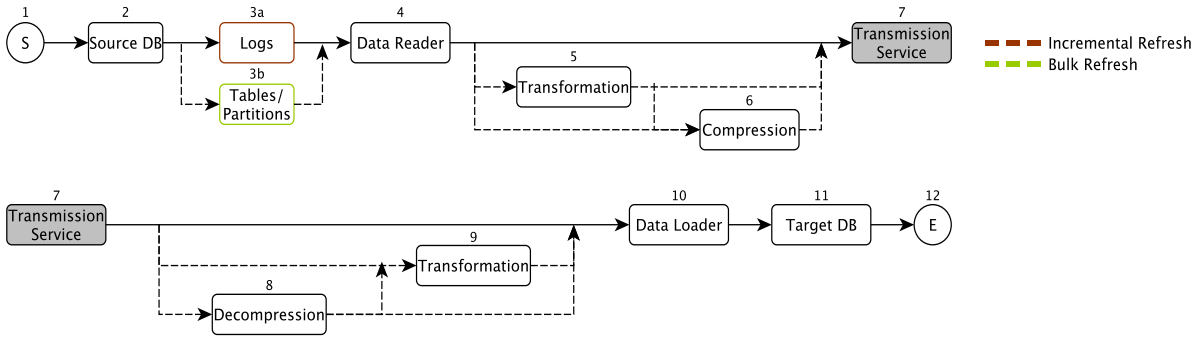

Figure 4.2: Unified Pipeline Model

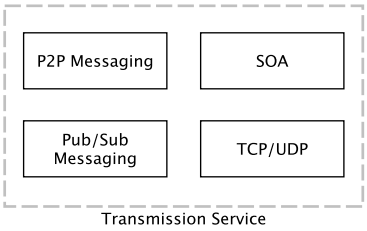

Figure 4.3: Transmission Service

In Tab. 4.1, the possible paths for incremental refresh based on unified pipeline model are determined, where mode determines the capabilities of the system. In basic mode, only replication is enabled keeping other processing components like transformation and compression disabled. Transformation enabled system indicates that system is also capable of performing certain transformations on the data along with its synchronization ability which is the default mode. When compression should be used to save network bandwidth, compression enabled mode can be employed which leverages compression algorithms in place. And the last mode of the system is the full mode, which is supposed to enable all available capabilities including replication, transformation, and compression. It should be noted that transmission service as in Fig. 4.3 also provide many options like SOA (Service Oriented Architecture), message based systems or pure TCP/UDP, which can be used as a data carrier. Where SOA based carrier is not of our interest but other transmission services are taken into consideration for the mathematical model. To determine possible paths for bulk/full load based refresh, change 3a to 3b in the calculated routes in Tab. 4.1 and rest is similar.

4.2.2 System Evaluation Pattern

This section elaborates the evaluation pattern that is applied to each stage of unified pipeline model of replication systems and to measure their performance. This pattern is chosen to keep

No.	Path Mode	Route
1	Basic	1, 2, 3a, 4, 7, 10, 11, 12
2	Transformation enabled	1, 2, 3a, 4, 5, 7, 9, 10, 11, 12
3	Compression enabled	1, 2, 3a, 4, 6, 7, 8, 10, 11, 12
4	Full	1, 2, 3a, 4, 5, 6, 7, 8, 9, 10, 11, 12

Table 4.1: Replication System Paths

our evaluation procedure identical for all stages or components of above-mentioned unified replication model. It is beneficial for the reader to grasp the idea of modeling each component to evaluate it for performance. Fig. 4.4 is the pictorial overview of this pattern. It is explained in following paragraphs with more detail.

The core of the component is its logic, which can be mapped to the functionality of that component. The logic tells what is the purpose of this component and what is being processed. It can further contain some small operating units or workhorses to perform a particular task and algorithm that is being in use. The logic of each stage is fed by input data (in our case it is replication data or changes that should be transferred from source to target end), and once it is processed, logic produces output data. This logic along with its input and output data is termed as data flow model. It should be noted that the output data of one stage becomes the input data of next stage in Unified Pipeline Model. This way, data or changes pass through each component, processed and reached the final stage that is target database. Logic is also dependent upon some component-specific parameters which help to process input data. These parameters can be some configuration factors needed for selected algorithm. For example, the logic of Compression stage is to compress the input data with some chosen compression algorithm and produces compressed data as output. Parameters include compression rate, compression rate, and algorithm specific configurations, etc.

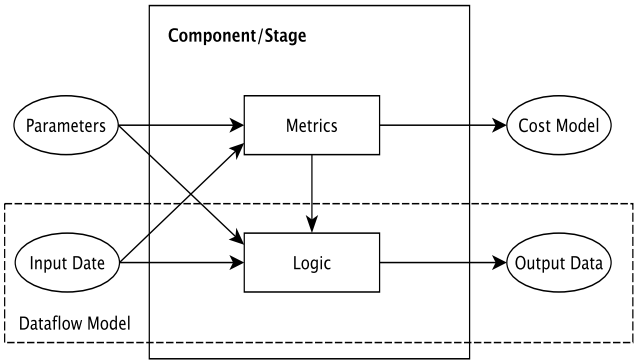

Figure 4.4: Evaluation Pattern

The next important part of this pattern are metrics which affect the performance of current processing component. In cost evaluation, these metrics play an important role and should

be determined. A pre-defined set of these metrics is consisting of latency, CPU usage, and throughput. Each stage is evaluated with this pre-defined set of metrics and a cost model is produced as a result. This cost model is the set of mathematical formulas, equations, or in some cases black box that determines theoretical values of metrics.

Parameters defined in this pattern can be further divided into two categories, specification and calibration parameters. The specification parameters are the one that are specific to that logic of the component, or algorithm used, for example, compression rate or compression ratio. On the other hand, calibration are those parameters which are not related to logic or algorithm, but they do impact logic, algorithm or their performance, i. e., platform or environmental configuration parameters. For a specific environment, calibration parameters can be same but their impact for different algorithm can differ. These all parameters should be determined and categorized accordingly to further proceed the evaluation of a particular component.

To summarize it, a pattern is developed, which is applied to each stage and component of unified pipeline model of replication systems to get cost model of that component. With this pattern, each component can be evaluated with the same procedure and a set of rules. Once the evaluation of each component in the pipeline is performed using system evaluation pattern, they are assembled to get overall system evaluation.

4.3 Replication System Performance Modeling

In this section and the following subsections, each component of unified pipeline model (see Sec. 4.2.1) of replication systems is explained with more details along with corresponding mathematical models for performance metrics as discussed earlier. This model can be used to estimate the performance of each component of a replication system.

4.3.1 Data Reader

Data reader is the backbone of replication systems. The core functionality of this component involves to read data and put it in some stream where it can be used for further processing. In the case of full refresh based synchronization, it copies changed tables or partitions from data source and make available them to other applications for loading. In the case of incremental update it observes the log files and capture data if a new change occurs and written in the logs. Performance depends on its change discovery and capture ability. If data reader is not configured carefully, it will append latency while replicating changes to target end. It should be monitored continuously for its working.

Fig. 4.5 explains the concept of a log reader. Presume an existing data source that should be added to target analytical service for the first time and never synchronized before. If data source already contains an enormous amount of data, it is a good idea to replicate already existing data using bulk refresh in out-of-business hours to minimize any transactional interruption. Bulk refresh can synchronize existing large amount of data (entire tables and partitions) in a source system with target in comparatively short time as compared to incremental refresh. On the other hand, small amount of changes can be replicated in near real-time using incremental refresh later. As shown in figure data from time t_0 to t_x that consists of all previous data, is replicated using one-time load strategy. From time t_{x+1} onwards incremental refresh is leveraged. At this time log reader will continuously observe log file and capture any new changes as soon as they appear. DBMS writer denotes a component which writes new changes to log files as they appear in physical tables following DML operations and log reader tries to catch up to read these new changes. There can be a lag between when changes are written to the log file and when they are captured which is called offset. It determines how much behind is the log reader from the writer. If offset increases, latency will automatically increase because it will take then longer to capture recent change. Focus should be to minimize this offset to decrease latency.

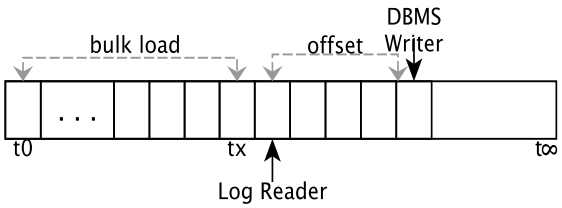

Figure 4.5: Log Reader

In the case of bulk refresh, data reader is a program or a script that can copy data from all changed partitions and tables. One such example is UNLOAD utility from IBM utility suite [35]. This utility unloads data from one or more source objects, i. e., tables or partitions to sequential data sets or HFS files. It must be run on the source systems.

4.3.1.1 Parametrization

In this step, all important parameters that affects performance of data reader are determined including specification and calibration parameters. They depend on the method or algorithm that is used to read changes from the data source. This set of parameters as P_{rdr}. Also find out how each parameter in P_{rdr} affects performance of data reader and include it in its cost model. One such parameter is the total number of changes in the source database denoted as C_n with each change having an average size expressed as C_s. Also determine the $offset$ especially in the case of IR system. Another important parameters to determine is the average *reading time*

(RT) of one change which is used to estimate latency later on. The $offset$ however, can be assumed to be zero in case of FR. So, P_{rdr} is constituted of $P_{rdr} = \{C_n, C_s, RT, offset\}$.

The last thing to note here is that C_n, in case of IR, is equal to total number of actual changes in the source database. While, in case of BR, it is equal to total number of rows in all changed tables subject to replication.

4.3.1.2 Data Flow Model

Here we will present data flow model of data reader component as defined in Sec. 4.2.2. Data reader reads data or changes from the source database and makes it available for next stages. In the case of an incremental refresh, log files of a transactional source database system are the input data for data reader, and for full refresh complete tables or partitions that have been changed work as its input.

Total output data of data reader needed to be transferred to the target system is proportional to the total number of changes made in the source system. Based on these total number of changes, total data size for transfer can be estimated. In the case of incremental refresh, formula 4.1 is used to calculate the total size of changes (total data size for transfer) that needed to be captured from the data source and should be applied to target. It is defined as the product of the total number of changes made in the source system and the average size of a change in that system and is viewed as the total number of changes that are being replicated.

$$S = C_n \times C_s \tag{4.1}$$

where C_n is the total number of changes, C_s is the average size of a change and S is the total size of data that should be transferred or total size of all changes in case of IR.

In the case of full/bulk refresh, data is much higher than incremental refresh because all tables or partitions that have been changed in source system should be replicated. Total size of data for transfer can be calculated using formula 4.2:

$$S = \sum_{t=1}^{n} S_t \tag{4.2}$$

where t represents a table in the source system, n is the total number of changed tables, S_t represents the total size of table including changed partitions respectively. Using the above equation, the total size of data for transfer S can be calculated in case of bulk refresh.

As the incremental refresh and full refresh are modeled in a single unified pipeline model (see Sec. 4.2.1), a mapping between total data that is sent in case of IR and total data that is sent in

case of FR is figured out. Total data for transfer in case of FR can also be viewed as changes similar to IR. Formula 4.3 refers total data for transfer as the changes to replicate.

$$S = (C_n \times C_s) + O_{fr} \tag{4.3}$$

again C_n is the total number of changes made in all tables or partitions (subject to replication) of source database or all data sources, C_s is the average size of a change and S is the total size of data that should be transferred. As it is already explained that in the case of full refresh, entire table or partition is replicated even if there is only one change made, so O_{fr} represents a constant overhead data of bulk refresh strategy that becomes the part of original changes and can be calculated using equations 4.2 and 4.3. Last thing here to note is that the total size of changes, i. e., S is the data that comes out of data reader stage in out pipeline model and it can also be expressed as $S_{out,rdr}$ which becomes the input of next stage.

4.3.1.3 Cost Model

The performance of data reader component is determined in this cost model using a pre-defined set of metrics, i. e., latency, CPU usage, and throughput.

4.3.1.3.1 Latency is the time needed by log reader to read a set of changes from the source database and make them available for further components. It is highly dependent upon the offset that lies between log reader and DBMS writer, i. e., how behind reader is currently from the writer. To determine latency, the offset should be determined first which is calculated based on the current position of reader and writer. Many product vendors provide this facility to its customers to get this offset. For instance, the current position of log reader in IBM Infosphere CDC [36] can be determined by setting its particular configurations. The following formula can be used to determine latency:

$$L_{rdr} = offset + (C_n \times RT) \tag{4.4}$$

where RT is the average reading time of one change. Products that implement incremental refresh functionality to perform replication like Infosphere CDC [22], Microsoft SQL Server [37], etc., provides interfaces which determine latency for customers.

4.3.1.3.2 CPU Usage is the utilization of CPU that log reader requires to read a set of changes, parse them and make them available for next steps or transmission. In our model, it is indicated as U_{rdr}. It can be determined using the management console of replication products like Infosphere CDC [22], Microsoft SQL Server [37], etc. Another approach determining this utilization is presented in IBM white paper [22]. Use following formula to measure CPU usage for entire data:

$$U_{rdr} = C_n \times U_{rdr/change} \tag{4.5}$$

4.3.1.3.3 Throughput , in the case of data reader, is defined as the rate at which it reads changes from data sources, denoted as R_{rdr}. Similar approaches as presented to calculate latency and CPU usage for log reader can be applied to measure throughput, i. e., management console of the replication product and the methodology presented in IBM white paper [22]. If latency is known, then throughput can be found using equation 4.6.

$$R_{rdr} = \frac{C_n}{L_{rdr}} \tag{4.6}$$

4.3.2 Transformation

Transformation is the process in which a series of operations are applied to the data or changes after they are read from data reader. The purpose of this component is to cleanse the data before it is applied to target. It is required whenever the schema of target database does not match with the schema of the source database. In the case of incremental refresh, it should transform all incoming changes in real-time to give a seamless replication, while in the case of load refresh it should be capable of handling a huge amount of changes and transforming them.

4.3.2.1 Parametrization

In this step, determine all factors or parameters that are important for transformation module. Transformation includes depending upon requirements, multiple operations, each of them has a unique impact on input data. These parameters, for instance, include the total number of operations that are part of transformation, how much each operation has impact on the size of input data, etc., and calibration parameters.

4.3.2.2 Data Flow Model

Input data of transformation module is equal to what is coming from data reader component. So it can be modeled using the following equation:

$$S_{in,trans} = S_{out,rdr} \tag{4.7}$$

and the output transformed data ($S_{out,trans}$) is the data coming out of transformation module. The size of output data might not be equal to the size of input data because of transformation operations, i. e., data cleansing, key-value mapping, format amendment, string manipulation, etc. These transformation operations define the size of output data. As these operations are arbitrary, the cost model of this module is acting as a black box.

4.3.2.3 Cost Model

In this step, performance evaluation of transformation stage is performed to determine latency, CPU usage and throughput.

4.3.2.3.1 Latency is the time that transformation process takes to perform all operations on the incoming data or changes. It depends upon how much time each step or operation is needed to perform a particular transformation. It is modeled using following equation:

$$L_{trans} = X \tag{4.8}$$

where X denotes a user estimated input value for latency of transformation component.

4.3.2.3.2 CPU Usage is defined as the total utilization of CPU needed by transformation process to perform all operations. It can be modeled using following formula:

$$U_{trans} = X \tag{4.9}$$

where X denotes a user estimated input value for CPU usage of transformation component.

4.3.2.3.3 Throughput , in the case of transformation, is the rate at which it can process a particular set of changes. It is denoted as R_{trans}. The transformation process involves multiple operations that are needed to perform required cleansing on data, throughput of this process is dependent on these operations and can be measured by the rate of an operation that is slowest of all. Using 4.10 throughput of this component can be found.

$$R_{trans} = X \tag{4.10}$$

where X denotes a user estimated input value for throughput of transformation component.

4.3.3 Compression

Compression is a method or a process to reduce the total size of data that is needed to be transferred to reduce network traffic and to utilize network bandwidth as efficiently as possible. The reduction in the total size of data means that total transfer time is also reduced. There are two types of compression **a)** lossy data compression and **b)** lossless data compression. In lossy compression, some data can be lost during the process of compression and is not recoverable again in case of decompression. This technique is frequently used for reducing the size of images. One such example is JPEG images whose size is much lesser than their original counterparts.

In contrast to lossy, lossless compression assures data quality along with compression. No data is lost during lossless compression, although in this case, the compression rate is much lower than lossy techniques. Lossy compression is suitable to use for multimedia data that mainly consists of images, audio, and videos, etc. If loss of data quality is not affordable at any cost, the lossless technique should be used. A hybrid approach can also be applied if data consists of important data along with multimedia data where later can be compressed using lossy compression algorithms.

There are many algorithms available in the industry that can be applied according to the chosen lossy or lossless strategy. Two main factors that depend on the performance of these algorithms are compression rate and speed. The algorithms that claim high compression rate usually work at lower speed and vice versa. In replication system, it should be kept in mind that compression time should not exceed the total transfer time required to convey data in its original format. While choosing compression for data replication, select among the available fast compression algorithms like LZ4.

4.3.3.1 Parametrization

In this step, all necessary parameters that do affect cost model should be determined. In this work only those parameters are found out that are common in almost all compression algorithm namely compression rate and compression ratio. Compression Rate denoted as V_c is the rate at which a selected compression algorithm can compress the data, and Compression Ratio CR is the ratio between uncompressed size and compressed size of data. It defines the size of output data after compression. The set of these compression parameters can be named as with P_{cmp} which should also include other specification and calibration parameters that affect performance of compression module. If compression algorithm is lossy, the factor of information loss should be determined if it affects the cost model. So, P_{cmp} is $P_{cmp} = \{V_c, CR\}$

4.3.3.2 Data Flow Model

Input data for compression component can come either directly from the output of data reader or transformation. The size of input data for this stage can be determined using the following equation:

$$S_{in,cmp} = S_{out,x} \tag{4.11}$$

where $S_{out,x}$ is the size of output data of previous component. The size of compressed data is denoted as $S_{out,cmp}$ and is dependent on compression ratio CR of the algorithm. It can be determined by using following formula:

$$S_{out,cmp} = \frac{S_{in,cmp}}{CR} \tag{4.12}$$

where *CR* as already defined is the compression ratio. Compression algorithm with high *CR* usually takes longer to compress data and directly impact on latency as well as CPU utilization. The suitable choice of algorithm is very important to keep the overall pipeline within effectiveness.

4.3.3.3 Cost Model

Here in this part, performance metrics, i. e., latency, CPU usage and throughput are determined for compression module.

4.3.3.3.1 Latency of compression module in above mentioned unified pipeline model can be measured using following equation:

$$L_{cmp} = \frac{S_{in,cmp}}{V_c} \tag{4.13}$$

where L_{cmp} is the total latency caused by compressing a particular data set, $S_{in,cmp}$ is the total size of input data and V_c is the compression speed or compression rate of the compression algorithm. The compression speed can differ substantially depending upon which algorithm is applied to compress data. For instance, LZ4 is a real-time compression algorithm that claims to yield output at very fast rate. It also depends on how much compression ratio *CR* is required, which quantifies that how many times data is compressed using data compression algorithm. A high CR means that data is compressed to a higher degree. If an algorithm is set to achieve high *CR*, it will negatively impact on latency.

4.3.3.3.2 CPU Usage Compression algorithms usually are very CPU hungry. They consume a lot of processing power and depending upon the compression ratio one want to achieve; they can cause a performance issue.

CPU usage (U_{cmp}) of compression is computed as a product of the total size of data and CPU usage for average change size C_s. Applying following formula, CPU utilization for $S_{in,cmp}$ can be calculated:

$$U_{cmp} = C_n \times U_{cmp/change} \tag{4.14}$$

The CPU utilization of each compression algorithm is unique and can differ how they are configured to use based on speed vs. compression ratio.

4.3.3.3.3 Throughput of compression module highly dependent on selected scheme of the compression algorithm. It is defined as the number of changes per second that can be compressed and can be calculated as:

$$R_{cmp} = \frac{V_c}{C_s} \quad \text{[changes/sec]} \quad (4.15)$$

if $V_c = 400MB/s$, average change size $C_s = 100KB$ then throughput of such a compression algorithm is 4000 changes per second. This means such an algorithm can compress 4000 changes (each of 100 KB) per second.

4.3.4 Transmission Service

Transmission service is the basis of replicating data from one source end to target end. There are many options available that can be used as the part of transferring data. These services have a huge impact on replication system, especially when source and target are present in different regions for example in the hybrid cloud. Therefore it is very important to measure performance parameters for them. The following sections present a mathematical model from measurement of performance metrics for each available transmission service. The data can be transferred using a transmission service that uses a middleware like MOM or just TCP/UDP protocols that are explained below.

4.3.4.1 TCP/UDP based Transmission

In this section, the performance effect of transmission service is measured while using only bare TCP/UDP protocols to send data from source to target. Using the system evaluation pattern, first of all, the essential parameters are found out which can have impact on further explained data flow and cost model.

4.3.4.1.1 Parametrization When using TCP/UDP protocols to transfer data, following are the few common delays that contribute towards transmission service latency (L_{tms}). Only important parameters are determined here.

Propagation delay also denoted as D_{prop} is the time that light signal takes to propagate from one point to another point. As discussed in Cisco white paper [38], the light signal does not travel at its full speed but at 122,000 MPS when it passes through glass or some copper material. So the light signal adds a propagation delay of around 8.2 μs per mile or 0.82 ms per 100 miles.

Each data packet is processed by few routers on the way to its destination and each router also adds some processing and serialization delay which are known as *processing delay* (D_{proc}) and

serialization delay (D_{ser}) respectively. A worst case processing delay per hop, with hardware assistance, is about 25 μs. If the Internet connection is 1 Gbps between source and target, then serialization delay is about 12 μs for 1.5 KB of packet and 8 ms for 1 MB of a packet. These measures are borrowed from above mentioned Cisco white paper.

Routers may not be able to handle incoming data packets because of their high input rate and they may get congested. To avoid any packet lost each router is configured with a queue where the packets are stored in case of congestion, that adds extra delay to the transmission latency and known as *Queuing delay* or D_{queue}.

Another delay that of utmost importance to discuss here is *transmission delay* [39] or D_{trans} which is the time taken to push all packet's bits over the wire. It is the delay caused by the data rate or transmission rate (W) of the connection.

Futhermore, each layer of OSI (Open Systems Interconnection) stack [40] adds some header fields along with original data. This overhead is totally neglected in our calculations. To get more accurate results, header size of each OSI layer should also be considered. This overhead is denoted with O_h and will use it to represent OSI overhead in the rest of the work, wherever applicable and is the sum of the size of all respective OSI layer's (physical, data link, network, transport layer, etc.) headers, see equation 4.16.

$$O_h = \sum_{i=1}^{n} S_h \tag{4.16}$$

where S_h is the size of the header of a particular OSI layer, n is the total number of layers that are contributing to this overhead and O_h is the total overhead size that is added to each packet.

The last parameter of our interest is PL_{ff}, named as Payload Fit Factor. It determines, how many packets of TCP are required for a change to carry. It should be determined as well in advance. For example, if $PL_{ff} = 0.5$ then it means a TCP packet can carry two changes and if $PL_{ff} = 2$ then it means a change requires two TCP packet to be fit into. So the set of parameters of transmission service, i. e., P_{tms} is comprised of $P_{tms} = \{D_{prop}, D_{proc}, D_{ser}, D_{queue}, D_{trans}, W, O_h, PL_{ff}\}$.

4.3.4.1.2 Data Flow Model Input data of transmission service is equal to the output data of previous stage, i. e., either of data reader, transformation or compression plus the size of OSI headers. The size of input data is expressed as $S_{in,tms}$.

$$S_{in,tms} = S_{out,x} + O_h \times C_n \times PL_{ff} \tag{4.17}$$

where $S_{out,x}$ is the size of output data of previous component, O_h is the total overhead per transmission packet, C_n is the total number of changes and PL_{ff} is the change fit factor. And output data denoted as, $S_{out,tms}$ can be determined using following equation:

$$S_{out,tms} = S_{out,x} \tag{4.18}$$

It is exactly equal to the size of output data of previous stage, because OSI headers will be removed once data arrived at target end.

4.3.4.1.3 Cost Model Once transmission service is parametrized and data flow model is determined, cost evaluation can be done as follows:

Latency of TCP/UDP based transmission service is defined as the time taken to transfer all data from one end to anther end and can be determined, based on already found parameters, as follows:

$$L_{tms} = D_{prop} + D_{proc} + D_{ser} + D_{queue} + D_{trans} \tag{4.19}$$

where L_{tms} is the total latency of transmission service, and D_{trans} is calculated as:

$$D_{trans} = \frac{S_{in,tms}}{W} \tag{4.20}$$

where W is the transmission rate.

CPU Usage is defined as the utilization of CPU required to transmit the complete set of changes and is denoted as U_{tms}. It is highly dependent on network hardware or network interface card used which is responsible for transferring data over the wire. Calculate it using:

$$U_{tms} = C_n \times U_{tms/change} \tag{4.21}$$

Throughput is defined as the rate at which TCP/UDP based transmission service is able to transfer data, denoted as R_{tms} and can be measured using either of the following equations:

$$R_{tms} = \frac{W}{C_s + O_h} \quad \text{[changes/sec]} \tag{4.22}$$

$$R_{tms} = \frac{C_n}{L_{tms}} \quad \text{[changes/sec]} \tag{4.23}$$

where W is the transmission rate as already defined in parametrization section. Here it is assumed again that each TCP or UDP packet is carrying one change.

4.3.4.2 Message Queue based Transmission

These days MQ based services are used very commonly to transfer data from source databases to target database. These MQ based services keep across the globe distributed databases as well as data warehouses in synchronization by applying changes in near real-time. Because of their high number of applications and ever-growing demand, it is necessary to discuss their performance as per already defined metrics. If somebody wants to leverage message-based transmission service as data replication strategy and willing to mount his/her system with best

of breed configurations, the possible methodologies to measure the performance of MQ service and its impact on overall system by looking into latency, CPU utilization, throughput, etc., are discussed below.

The industry standard benchmark for evaluating the performance of enterprise messaging systems, i. e., SPECjms2007 [41] is the base for our model, because this is the first industry-standard benchmark for evaluating the performance of MOM servers based on JMS (Java Message Service). The results of this benchmark can be viewed in this report [42]. Although the performance measures substantially dependent on a particular product, still the results of this report can be used as a benchmark to tune a system accordingly.

4.3.4.2.1 Parametrization Message-oriented middleware, as the name describes is a middleware and require additional configuration. MOM products usually require a server running on some hardware, so parameters that are important in this middleware should be analyzed, and an impact on data and cost model should be determined.

As message based applications deal with only messages, it is necessary to convert incoming data changes into messages and M_n denotes the total number of messages required to carry all changes. Each message has a fix size denoted as M_s. Another important parameter here to discuss is the average size of header fields and properties fields added by MOM in each message and is denoted as O_m . These header fields and properties have already been discussed in Sec. 2.3. This parametrization section is common both in point-to-point and publish subscribe messaging transmission and will not be discussed in the case of pub/sub message transmission section.

The last parameter is PL_{ff}, as already known is named as Payload Fit Factor. It determines, how many packets of TCP are required for a message, in this case, to carry. It should be determined as well in advance. So the set of parameters, i. e., P_{tms}, in case of messaging consists of $P_{tms} = \{M_n, M_s, O_m, PL_{ff}\}$.

4.3.4.2.2 Data Flow Model Before going further, it is mandatory to calculate total number of messages that should carry total data coming from previous component. And it depends on how much data a message should carry, by assuming it to be equal to a change size C_s:

$$M_n = \frac{S_{out,x}}{C_s} \tag{4.24}$$

Input data, denoted as $S_{in,tms}$ of messaging middleware is equal the size of output data of the previous stage in our pipeline model plus the size of OSI headers and the size of message headers. As transformation and compression stages are optional, if they are not used data will directly come from data reader module, otherwise from transformation or compression stage. It

is dependent on size of individual message, which can be determined using following equation:

$$M_s = C_s + O_m + O_h \times PL_{ff} \tag{4.25}$$

where C_s is, as already defined, the average size of a change or tuple which becomes the size of payload in a message, O_m is the average size of header fields and properties fields added by MOM in a message, O_h is the overhead that is introduced by OSI stack, PL_{ff} is the payload fit factor and M_s is the size of an individual message. Finally, the total size of all messages can be determined by using either of following two equations:

$$S_{in,tms} = M_n \times M_s \tag{4.26}$$

$$S_{in,tms} = S_{out,x} + M_n \times (O_m + O_h \times PL_{ff}) \tag{4.27}$$

where $S_{in,tms}$ is the total size of data for all changes that should be sent over the wire in case of messaging, $S_{out,x}$ is the size of output data of previous stage. Similarly, the size of output data is measured using:

$$S_{out,tms} = S_{out,x} \tag{4.28}$$

$S_{out,tms}$ is exactly equals to the size of output data of previous stage, because OSI and messaging headers will be removed once data arrive at target end.

4.3.4.2.3 Cost Model Here cost evaluation of P2P messaging transmission system with the help of pre-defined metrics, i. e., latency, CPU usage and throughput is determined.

Latency for a message queuing system based transmission service is very important to measure. It is the time taken by all messages to reach from source end to target. And this is done by using following formula:

$$L_{tms} = L_{mq/msg} \times M_n \tag{4.29}$$

where $L_{mq/msg}$ is the mean latency of a message of size M_s in MQ system and L_{tms} is the latency of transmission service to send all messages or all incoming changes.

Example: Fig. 4.6 shows results of a point to point (P2P) MQ system to measure performance metrics. Let us assume $S_{in,tms} = 1000 \times 10KB = 10MB$, $M_s = 10KB$ and $L_{mq/msg} = 1.1ms$ as shown in report results. Applying formula 4.29 results $L_{tms} = 1.1ms \times 1000 = 1.1s$. This simple example shows that total latency of 1.1 second is introduced by P2P MQ system for total 1000 changes. These results can differ depending upon which messaging system is being in use as a transmission service.

CPU Usage is calculated based upon the usage for one message. Let $U_{mq/msg}$ is the CPU usage per message, then total CPU usage for transmission service U_{tms} can be estimated using following formula:

$$U_{tms} = U_{mq/msg} \times M_n \tag{4.30}$$

Let us take an example for determining CPU usage as well based on results accumulated by SPECjms2007. As per report, CPU time per message $U_{mq/msg}$ increases linearly with the message size but CPU time per KB payload of a message becomes stable at about 10 KB of message size. For a message of size 10 KB, average CPU time is about 1.1ms. Assuming $S_{in,tms} = 1000 \times 10KB = 10MB$ and $M_s = 10KB$ results total CPU usage to be 1.1s for all 1000 changes. This example shows how CPU usage can be calculated for entire data changes.

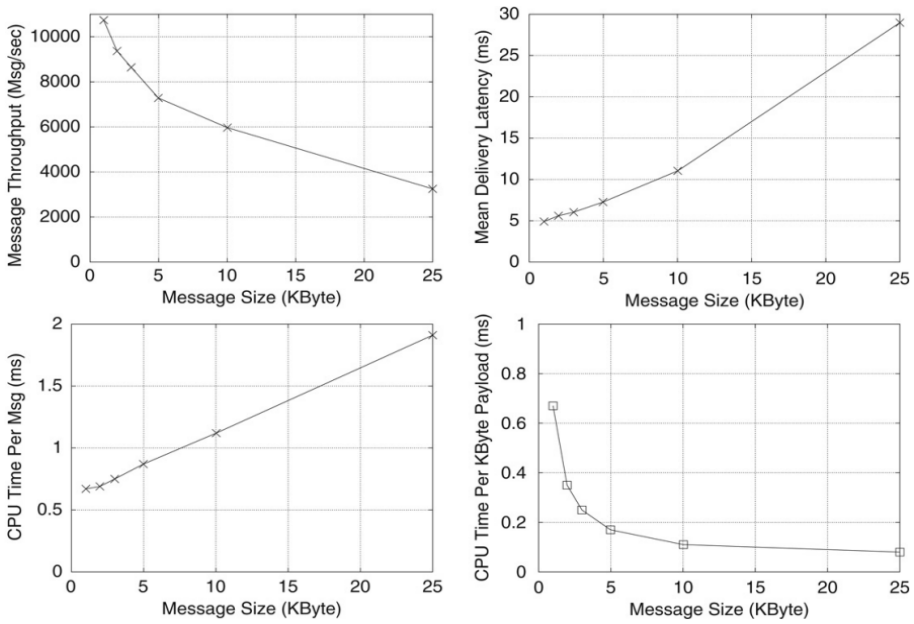

Figure 4.6: JMS 2007 Specification Results

Throughput It is time to discuss now how throughput for such a system can be calculated. Throughput denoted as R_{mq} of a message based queuing system is measured in messages per second that an MQ system capable of delivering to recipients. As only one change or tuple is embedded in one message, so the throughput of transmission service in terms of the number of changes is also same.

However, if change average size is large and should be broken into small pieces to fit into a message, then R_{tms} can be calculate using formula 4.31 provided below:

$$R_{tms} = \frac{R_{mq} \times M_s}{C_s} \quad \text{[changes/sec]} \tag{4.31}$$

Example: Taking $M_s = 10KB$, $C_s = 10KB$ and $R_{mq} = 6000msgs/sec$ (message throughput value taken from SPECjms2007 report) result in a throughput of 6000 changes/sec. That means such a system is capable of delivering 6000 changes (each of 10 KB in size) per second. If we take average size of a change to be 20 KB then throughput in terms of changes per second reduces to 3000. In short, such a MQ system is able to transfer $6000 \times 10KB = 60MB$ of data from source system to target system.

4.3.4.3 Pub/Sub based Transmission

Publish/Subscribe system shortly known as Pub/Sub or PS system is another paradigm of MOM or MQ systems to efficiently deliver thousands of messages per second to target party. In contrast to P2P messaging systems, PS systems are based on topics to which any number of publishers or subscribers can register. Multiple publishers can publish messages to a particular topic which are then received by all registered subscribers. How is such a system useful in our case? Multiple data sources in our system will act as publisher which should publish their data changes in the form of messages to a particular topic which are then delivered to registered subscribers or target database systems. A topic is a special queue in MQ systems which is able to receive messages from any number of publishers and deliver them to any number of subscribers.

There is no need to re-invent the wheel in case of PS system. As the core of these systems is same, i. e., these systems also carry data in the form of messages as like P2P systems, so parametrization, data flow and mathematical cost model developed in Sec. 4.3.4.2 can be used as is for pub/sub messaging transmission service without any modifications.

4.3.5 Decompression

Decompression is the process of restoring data that was compressed before using some compression algorithms. Based on compression ratio applied, decompression algorithm yields unique performance. Usually, decompression speed of almost all compression algorithms (like LZ4) is much higher than compression speed. So it will not cause any throughput bottleneck.

4.3.5.1 Parametrization

Exactly like compression, find out important parameters that have an impact on decompression module. These parameters come from required configuration of decompression algorithm along with the environment in which it is running. One important factor that is common in all algorithms is decompression rate, the rate at which selected algorithm can bring compressed data to its original form. It is represented as V_d. Similarly, other configuration factors that are specific to algorithm should be figured out and their impact on our cost model. So the set of parameters is comprised of $P_{dcmp} = \{V_d, CR\}$.

4.3.5.2 Data Flow Model

Data flow model of this stage defines its input and output stream of data. Input data, as already discussed, is dependent on output data of the previous stage. The size of input data can be determined as:

$$S_{in,dcmp} = S_{out,x} \tag{4.32}$$

Similarly, the size of output data, which is dependent on compression ratio CR used earlier in compression module, can be determined using following equation:

$$S_{out,dcmp} = S_{in,dcmp} \times CR \tag{4.33}$$

Definitely, the original information can only be retrieved back if lossless compression algorithm is used.

4.3.5.3 Cost Model

4.3.5.3.1 Latency of decompression module in above mentioned generic replication system can be measured using following equation for incremental refresh:

$$L_{dcmp} = \frac{S_{in,dcmp}}{V_d} \tag{4.34}$$

where V_d is the de-compression rate or speed of the selected algorithm, $S_{in,dcmp}$ is the compressed data, and L_{dcmp} is the decompression latency.

4.3.5.3.2 CPU Usage of decoding or decompressing compressed data can be measured the same way as it is calculated for compression. Assume U_{dcmp} is the CPU utilization, and it represents the total CPU usage for decompressing $S_{in,dcmp}$ into its normal format.

4.3.5.3.3 Throughput of decompressed determines how much faster compressed data can be decompressed. It is described as the number of changes per second that can be decompressed given decompression speed or rate of selected algorithm and can be calculated as:

$$R_{dcmp} = \frac{V_d}{S_{in,dcmp}} \times C_n \qquad \text{[changes/sec]} \tag{4.35}$$

where V_d is the decompression rate, $S_{in,dcmp}$ is the size of compressed data, and R_{dcmp} is the decompression throughput in changes per second.

4.3.6 Data Loader

This component is responsible for applying incoming data changes to the respective database and tables. Likewise data reader, this component is also made generic unit. In both replication strategies, the same component is referred that is responsible for applying changes to the target end. Although technically it would be totally different design and implementation for each replication strategy.

It is a program or process, in the case of an incremental refresh, that can write continuously incoming changes to the target database without causing unnecessary delay. For example, in IBM InfoSphere change data capture [43] a specific component called "Apply Agent" on target engine performs this task. The mapping between data and corresponding tables or databases is determined by metadata. In contrast to incremental refresh, a specially tuned program should be able to write a huge amount of changes to the target database in case of load refresh. And for this replication strategy, IBM provides a program named as "UNLOAD" utility [35] that is responsible for applying a huge amount of changes to target database.

4.3.6.1 Parametrization

As like other modules, in this step specification and calibration parameters are found that can impact the performance of data loader. One such parameter is average *loading time* of one change into the target database. Another parameter is the $offset$ which is the time span that exists between the time when the change reached data loader component and when it is finally applied to the database system. So the set of parameters is comprised of $P_{ldr} = \{LT, offset\}$.

4.3.6.2 Data Flow Model

The basic functionality of data loader is to load or apply data to the target database; it does not perform any action that can lead to altering the total size of incoming data to data loader. The size of input data $S_{in,ldr}$ of data reader module is equal to the size of output data of the previous stage, and size of output data $S_{out,ldr}$ of data reader is equal to the size of input data $S_{out,ldr}$.

$$S_{out,ldr} = S_{in,ldr} = S_{out,x} \tag{4.36}$$

4.3.6.3 Cost Model

Following is the cost model of the data loader module which is dependent of data flow model and parameters already found.

4.3.6.3.1 Latency in the case of data loader, can be rephrased as the time period that this component takes to apply a set of incoming changes to the target database in respective tables and partitions and is denoted as L_{ldr}. There are different methods to measure this time. An easy way to determine it using the management console of a particular product that is used for replication process [43] or it can also be measured using the same approach as described in IBM technote [22]. It can also be measured using the following equation which is similar to one presented in data reader section:

$$L_{ldr} = offset + (C_n \times LT) \tag{4.37}$$

where LT is the average loading time of one change to target database.

4.3.6.3.2 CPU Usage can be defined as the utilization of CPU that data loader needs to apply a set of changes to the target database. It is denoted as U_{ldr} and can be determined the same way as latency either using management console of the product or as described in white paper [22]. If usage for one change is known then use following formula:

$$U_{ldr} = C_n \times U_{ldr/change} \tag{4.38}$$

4.3.6.3.3 Throughput is defined as the rate at which data loader can apply changes to the target database and denoted as R_{ldr}. The management console of a particular product that is used for replication usually provides its information; otherwise, it can be measured as presented in IBM white paper [22]. If the latency of data loader is known, then throughput can be found using equation 4.39.

$$R_{ldr} = \frac{C_n}{L_{ldr}} \tag{4.39}$$

4.3.7 Complete Cost Model

As it is already detrmined, how performance evaluation can be made for each individual stage or component of the unified pipeline as presented in Sec. 4.2.1. Here the idea to connect each stage together is presented and find out overall cost model of complete pipeline. Cost model of each individual module is sufficient to evaluate that particular stage, but cost evaluation of complete system is only possible if each single module is connected with other modules to get a full picture of performance.

4.3.7.1 Latency

Latency of complete pipeline is the sum of latencies found in each individual stage and can be found using following equation.

$$L = L_{rdr} + L_{trans} + L_{cmp} + L_{tms} + L_{dcmp} + L_{ldr} \tag{4.40}$$

where L is the full latency of complete pipeline or replication system, L_{rdr} is the latency found in data reader stage, L_{trans} is the transformation stage latency, L_{cmp} is the compression latency, L_{tms} is the latency of transmission service, L_{dcmp} is the decompression latency and L_{ldr} is the latency of data loader.

4.3.7.2 CPU Utilization

Like latency, CPU utilization of complete system is also the sum of CPU usage calculated at each individual stage of pipeline and can be measured using following formula:

$$U = U_{rdr} + U_{trans} + U_{cmp} + U_{tms} + U_{dcmp} + U_{ldr} \tag{4.41}$$

where U is the CPU utilization of complete pipeline and other parameters are already defined above.

4.3.7.3 Throughput

Unlike latency and CPU usage, throughput of complete pipeline is the minimum value of throughput found in each individual stage.

$$R = \min(R_{rdr}, R_{trans}, R_{cmp}, R_{tms}, R_{dcmp}, R_{ldr}) \tag{4.42}$$

where R is the total throughput of the system.

4.4 Cost Effectiveness

In this section, the effectiveness of replication system is determined. Although, this metric is not the core of this thesis but the unified pipeline model described in Sec. 4.2.1 is so generic that it can be even applied to this aspect, without changing anything in the model, just by identifying the core parameters.

In general terms, CE tells that how practical and effective is a system given certain set of execution conditions and input parameters and given a threshold it can be used to analyze if current replication system is acceptable and suitable to use. The core performance parameters are used to calculate CE of the replication system. All three aforementioned parameters, i. e., Latency, CPU Usage and Throughput, contribute towards the calculation of CE. Once these three factors are in hand, formula 4.43 can be used to calculate CE.

$$CE = \frac{1}{U \times L} \times R \tag{4.43}$$

Where U is the sum of total CPU usage of every component in our unified pipeline model, L is the sum of total latency, and R is the lowest value of throughput from all components. Throughput has a direct relationship to CE, if higher is the throughput more will be the cost-effectiveness. On the other hand, latency is inversely proportional to CE that means high latency will have a negative impact on the system because it will decrease CE. Similar to latency, CPU usage is also indirectly related to effectiveness. While developing a model for change replication system, an attempt should be made to increase throughput and decrease latency and CPU usage so that it results in an overall high CE.

4.5 Replication System Monetary Modeling

As Analytics-as-a-Service deployment is structured into hybrid model where transactional data sources, as well as source replication components, are deployed on customer premises and analytical service, data warehouse and corresponding target replication services are managed by cloud provider; there is a need of such a model that can be applied to assess financial aspects of this hybrid system, that can analyze the IT components that are deployed on the private premises of customer, analytical services that are hired from a public cloud provider and communication services used to transfer data from data sources to target end. This section is dedicated to analyzing financial aspects of infrastructure that leverages replication systems and on-line analytics.

4.5.1 Pricing

In this part, a mathematical model for pricing is developed that helps to calculate the overall cost of using on-line data analytical service consisting of a hybrid model of deployment. The architecture presented in Fig. 4.1 is used as the base of the pricing model, while this architecture elaborates how the benefits of on-line data analytical service can be enjoyed by keeping data sources with on-premise infrastructure. As already explained in Sec. 4.1 that transactional data

sources and corresponding replication components are hosted with tenants private IT infrastructure, that are connected with a communication layer to analytical service hosted by a public cloud provider. Following equation models the cost of such a system.

$$P = P_s + P_{com} + P_t \tag{4.44}$$

where P_s is the total price of hosting data sources along with required components of replication strategy on on-premise infrastructure, P_t is the total cost of using analytical services along with replication components from a public cloud provider AaaS, P_{com} is the total cost that is used to transfer a particular set of changes from source to target and P is the total price of the overall system. where P_s is the total price of hosting data sources along with required components of replication strategy on on-premise infrastructure, P_t is the total cost of using analytical services along with replication components from a public cloud provider AaaS, P_{com} is the total cost that is used to transfer a particular set of changes from source to target and P is the total price of the overall system.

Equation 4.44 is prepared keeping the hybrid model of on-line analytical service keeping in mind which allows to model the complete architecture of this system for price. P_s, P_t and P_{com} can be further de-composed and analyze where these unit prices in our unified pipeline model of replication strategies (see Sec. 4.2.1) develop. This model breaks down a complete replication system into different components, each of them affects these costs. P_s depends on the components that are located on source side and develops from the usage of source databases and usage of source data replication services such as data reader, compression, transformation etc, P_t is, on the other hand, dependent upon target side and develop from usage of target replication services such as de-compression, transformation, data loader and analytical services explained in Sec. 4.1 and Sec. 4.2.1. P_{com} is the price that has to be paid for network service used to carry and transfer data from source to target.

In the case of hybrid deployment of AaaS, the price of replication services depends on their usage. To charge their customers, different cloud providers have unique pricing models. Usually, these cost models take CPU usage of an application, storage space, network traffic, running time, additional services and features and maintenance, etc., into account while developing them. This model uses already calculated performance parameters, i. e., latency, CPU usage, throughput to calculate the total cost of the system. Among different cloud providers, two pricing models are very common, pay-per-use and subscription based.

4.5.1.1 Pay-per-Use based Pricing

Pay-per-Use[1] is the billing model in which customers pay for what they use. There is no basic or minimum amount that users need to pay monthly. This kind of model is suitable for those

[1] http://cloudtweaks.com/2012/04/cloud-computing-cloud-pricing-models-part-4/

customers who thinks their applications and services will run for a limited amount of time per day. This way they can save money for not using cloud infrastructure rest of the time. This model is used by many public cloud providers including IBM Bluemix[2]

This pricing model can be broken into three categories, i. e., 1) storage price, 2) network price, 3) computing price. Storage price is defined as the cost that is needed to be paid for the usage of storage services like database etc. Network price is the cost that is applied to the usage of network services for transferring data between source and target. And computing price is the cost that is paid for the usage of computation services, for instance, CPUs and RAM, etc. These prices can be calculated with the help of following high-level equations:

$$Storage\ Price = \frac{S}{Mbyte} \times [Price\ per\ Mbyte/month\ storage] \tag{4.45}$$

Where S is the total size of changes (divided by $Mbyte$ to make sure it is in megabytes) as defined in our unified pipeline model of replication strategies and the rest of equation is pretty much self-explanatory. If there is any storage service used, 4.45 can be used to estimate its pricing.

$$Network\ Price = \frac{S}{Mbyte} \times L_{tms} \times [Price\ per\ Mbyte/sec\ network] \tag{4.46}$$

where S is the total size of changes that is needed to be transferred, L_{tms} is the total time that is needed to transfer data using transmission service and the rest of equation is clear. One important thing to remember here is that total size of changes can be decreased due to compression as discussed in Sec. 4.3.3 and can be increased due to the overhead of communication protocols, see Sec. 4.3.4. These factors affect total size of data that is loaded on the wire and have an impact on network cost.

Last but not least, the compute price can also be computed by using following equation:

$$Compute\ Price = U \times [Compute\ Price\ per\ sec] \tag{4.47}$$

where$U$is the total CPU utilization already defined, and compute price is the sume of price of using computational resources, like CPU and memory etc.

The price for data replication system can be measured using this model once the total change/load size S, CPU usage, throughput and latency are calculated as discussed in above chapters. Here CPU usage will determine how much CPU is required by our replication system. Throughput will let us know that what kind of data transfer service is needed, latency can be used to predict for how long each service is running and S will depict how much data is stored in the cloud. Based on these assumptions, the theoretical values of performance metrics can be given as input

[2]https://console.ng.bluemix.net/pricing/

in pricing calculators to measure total cost of the system. These calculators are already provided by almost all famous cloud providers like Amazon Web services[3], IBM Bluemix[4], Microsoft Azure[5] etc.

4.5.1.2 Subscription based Pricing

The customer can get a subscription based account as well to avail cloud services. In this model, they need to pay a minimum amount each month and can get some subscription discounts from the providers. If usage exceeds than the subscription amount, they will be charged separately. Subscription-based pricing is not of our interest, as the focus is on the pay-per-use model. It is not further explained, but the reader is definitely encouraged to get more information to have a knowledge of different types of pricing models and to use it according to the needs.

4.6 Summary

To summarize the concepts presented in this chapter, there are two intensively used replication strategies which perform their task pretty efficiently. A unified pipeline architecture of the system is developed that can be utilized for any of these replication strategies. In that unified model, those components which perform common responsibility irrespective of which strategy is in play are identified. The model that is presented here offers many different configurations to the system architect to mold the replication system according to the needs. Then we have evaluated our model for performance and determined how the performance metrics like latency, CPU utilization, and throughput can be measured. And how the effectiveness of the overall system can be supervised by calculating the cost-effectiveness of the system. Last but not least, this chapter explains the financial aspect of this model as well and points out how a replication system can be made price effective.

[3] https://calculator.s3.amazonaws.com/index.html
[4] https://console.ng.bluemix.net/?direct=classic/#/pricing/cloudOEPaneId=pricing&paneId=pricingSheet
[5] https://azure.microsoft.com/en-us/pricing/calculator/

Chapter 5

Implementation

Sec. 2.4 describes data replication strategies i. e., incremental refresh and full or bulk refresh, in detail. Incremental refresh is used to orchestrate data in near real time and provides a fully updated view of the source system, whereas bulk refresh loads entire table or partition and replicate it to the target system. Bulk data refresh is configured to run after a specific interval because of probably very huge data in source tables. Sec. 2.4 also provides an overview of different modes in which these techniques can be used to perform data synchronization. A conceptual model is also presented in the previous chapter that provides a mathematical formulation of performance metrics. This model can be used to evaluate the entire replication system to figure out performance holes and for optimized provisioning of data. These metrics include latency, CPU usage, throughput, cost effectiveness of the system and pricing.

This chapter mainly consists of two parts, one of them presents a use case that employs one of the two already discussed data synchronization methods and a specific operational mode with industrial tools and technologies and also describes how performance parameters can be measured at each stage. The second part provides an implementation of a program that leverages aforementioned conceptual mathematical model to measure performance metrics and provides a handy tool to evaluate the performance of the entire system. The results of the first part of this chapter, which mainly focus on determining different performance indicators such as latency, throughput, etc., from different components of the system, will become an input to the second part. And results of the second part help architects to assess the replication system and to remove any bottlenecks.

5.1 Overview of IDAA

IDAA (IBM DB2 Analytics Accelerator) is a high-performance appliance from IBM. The main purpose of this product is to accelerate response time of analytical queries by reducing their

processing cost. It combines IBM Netezza, zEnterprise, and DB2 for z/OS technologies to present a super fast and reliable solution to very complex, high processing and CPU utilization demanding and very slow data intensive OLAP queries. With the help of IDAA, the results of complex OLAP queries over data warehouse, business intelligence, and analytics can be generated from hours and days to seconds and minutes. It enables DB2 database with additional capabilities to process any analytical and data warehousing workloads along with transactional workload which is the usual functionality of DB2 database. Acceleration geared DB2 database act as a hybrid product which can handle any sort of query including transactional as well as analytical workloads.

Fig. 5.1 referred to main components of accelerator in its traditional implementation. It includes an IDAA plugin to DB2 for z/OS, Netezza technology acting as the workhorse of this product and data studio to provide an application interface to the users. IDAA needs a full replica of operational data subject to the acceleration that is stored in DB2, and this replica is provided either by full load or incremental refresh discussed in next sections. If in the course of time data on DB2 tables changes, this should also be replicated to IDAA data service. In order to support fast and near real-time replication of data from operational database i. e., DB2 to a dedicated highly available analytical data service hosted by IDAA itself, a private and dedicated fiber network connection is used with a capacity of 10 Gb/s. Mainly two OSA network cards are used to create a physical, highly available and redundant network. In this configuration, IDAA compensates itself the loss of the main network connection. The accelerator plugin that becomes

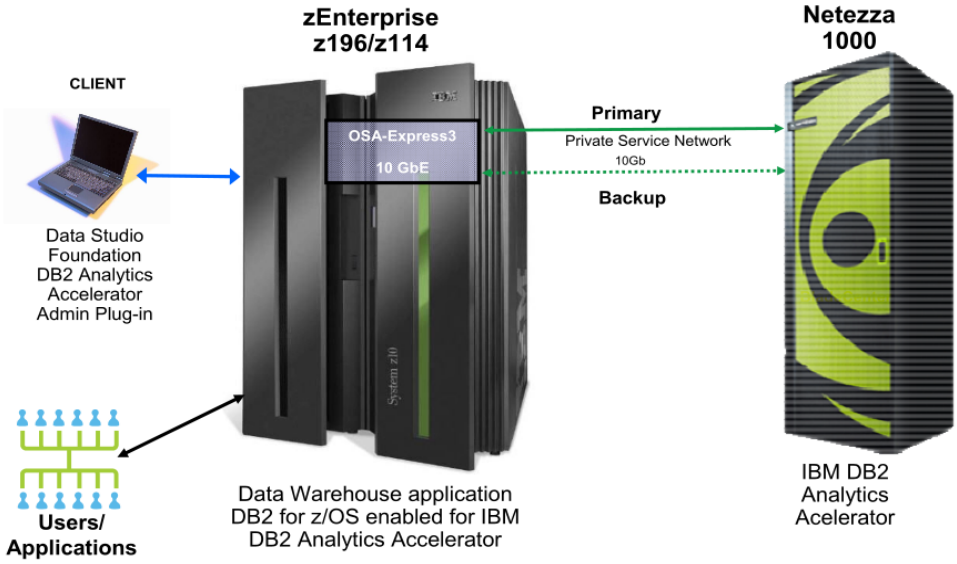

Figure 5.1: IBM IDAA Product Components [44]

the part of DB2 database is responsible for decision making of query routing, data replication, and some other tasks. Accelerator machine or Netezza box consisting of Netezza hardware processes analytical queries by scanning tables on all of its disks in parallel utilization full power of FPGAs (Field Programmable Gate Arrays). The accelerator decomposes analytical

queries into small portions also called snippets, before they get executed on all active disks. The computing power results in a scan rate of around 9GB/s or 36GB/s with compression.

Netezza technology brings database, processing power and storage together in a compact system to gear up analytical processing. It is a purpose-built system to integrate data warehousing, business intelligence and analytics needs as one single appliance. It has the ability to execute very long and complex OLAP queries very quickly by leveraging processing power of built-in FPGAs. [45]

Data Studio provides a programming interface to the accelerator to do management tasks. For example to tell DB2 which tables and partitions are subject to acceleration, to set the configuration of built-in accelerator needed in full load or incremental update functionality or query routing management, etc. Under the hood, data studio uses DB2 for z/OS stored procedures to communicate with IDAA. [44, 46, 47]

In traditional deployment model of IBM DB2 Analytics Accelerator, each component of its architecture as shown in Fig. 5.1 is on-premise along with mainframe transactional data source, i. e., DB2 for zOS. Customers have to buy a complete package of IDAA, i. e., appliance comprising IDAA and analytics back-end database including hardware once, whereas the mainframe comprising of DB2 for zOS has a pay-per-use model. In the next section, It is shown how IDAA appliance can be moved to the cloud while keeping OLTP data source on-premise to make IDAA more flexible in terms of its usage. This implementation model will make IDAA deployment more easier and more affordable to the customers who want to have this appliance also as the pay-per-usage model.

5.2 IDAA on Hybrid Cloud

As it is already discussed in chapter 1 the basic idea of this work is to analyze cost model of data replication on an on-line analytical service. This section explains a use case of IDAA deployment on a hybrid cloud by moving analytics appliance to off-premise infrastructure while transactional source systems are still running traditionally as on-premise. It also explains the architecture of IBM accelerator a bit in more detail and presents an idea to move data from on-premise transactional source system to off-premise IDAA appliance.

The main purpose of IDAA is to accelerate analytical and BI related queries which it executes on a completely separate database (a thorough copy of DB2 tables and partitions meant for acceleration). This separate copy database is maintained and updated on a regular basis by another IBM product called Infosphere CDC which comes along with IDAA. This data replication service synchronizes data from DB2 operational storage to IDAA analytical database service in near real-time.

Fig. 5.2 depicts an overview of the architecture of IDAA when deployed in a hybrid cloud environment and main components used to provide incremental update feature. The following subsections explain briefly the responsibilities of each component and then goes on further to incremental update functionality that is of utmost interest in for this work.

5.2.1 On-Premise Accelerator Plug-in for DB2 for zOS

This is the part of accelerator that operates on DB2 for zOS and consists of many subcomponents. This plug-in enables DB2 for zOS to enjoy power and benefits of the accelerator by connecting seamlessly DB2 with IDAA and providing access to the DB2 users to use accelerator. It assists DB2 users to issue SQL queries including OLTP and OLAP to the same interface. The responsibility to differentiate between queries, whether they belong to OLTP category or OLAP and whether they should be processed by DB2 local query execution runtime engine or be forwarded to the accelerator is owned by this plug-in itself. The sub-components that it comes with are discussed below.

5.2.1.1 IDAA Stored Procedures

They provide an application interface to the accelerator. These SPs (Stored Procedures) are written with DB2 stored procedures to give administrative access. Each SP provides a unique functionality. When a function using IDAA studio is invoked, a specific SP is called. These stored procedures can not only be called from IDAA studio but also from a command line interface or JCL (Job Control Language). They provide functionality that is related to tables and accelerator. IBM DB2 analytical accelerator store procedures provide an interface to manage tables subject to acceleration, add and remove tables, administer query optimization process, enable and disable replication processes that start or stop mirroring of tables defined and/or manage sub-systems for example data sources, replication agent systems and more.

5.2.1.2 Query Optimizer

It offers a hybrid approach to support both transactional and analytical workloads and takes decisive power to differentiate between different types of SQL queries in its own hands. It knows where to forward incoming query for processing. Whenever a user posts a query it is forwarded to query optimizer for decision making, and based on its heuristics it takes the decision whether the query is a transactional query and should be handled locally or it is an analytical query. If the query is transactional or OLTP it is forwarded by the optimizer to DB2 local query runtime engine, otherwise query is offloaded to IBM DB2 analytical accelerator to

enjoy its limitless power of executing analytical queries. The optimizer is intelligent enough to recognize when it is beneficial to offload the query. Once the OLAP query is processed by the accelerator, the results are handed back to DB2 for zOS to provide a unified interface to the application users. What is happening behind the scene is completely transparent to the users. [48]

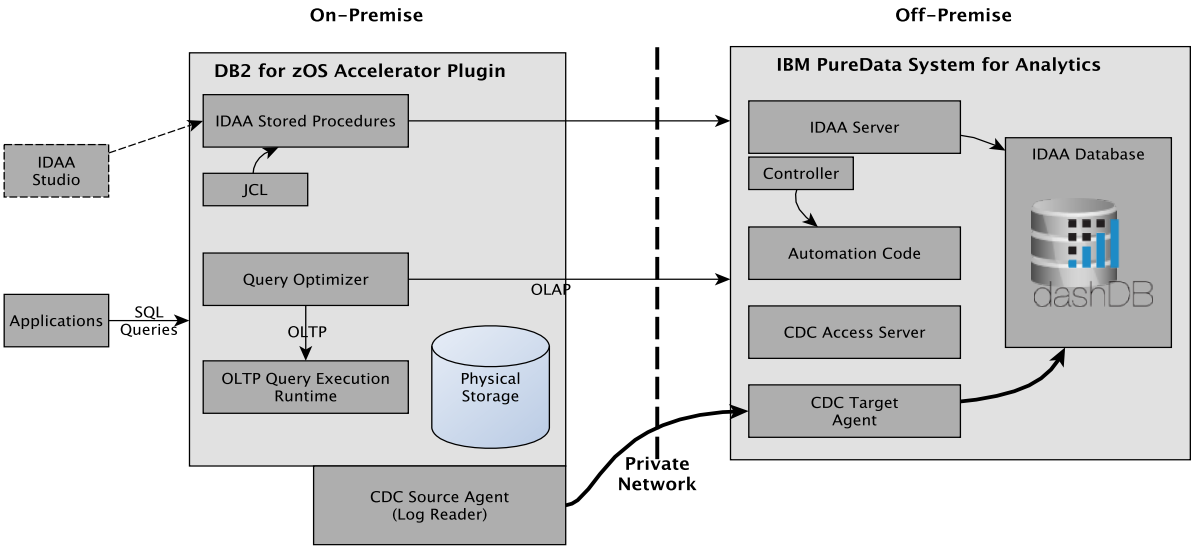

Figure 5.2: IDAA Appliance on Hybrid Cloud

IBM CDC is not the only option to replicate data from DB2 for zOS database to IDAA analytical database. Full table refresh or table partition refresh can also be used whenever there is a need to replicate entire table or table partition. This strategy is also known as UNLOAD-based refresh which uses basically DB2 UNLOAD utility to transfer data to the accelerator. [22]

5.2.2 Off-Premise IDAA Appliance

This is actual accelerator machine which hosts different components along with servers and target database. In a traditional deployment, accelerator appliance along with its hardware and software components is kept on-premise, but in IDAA's cloud deployment model, it is kept off-premise to facilitate customers and tenants to use it according to pay-per-usage model. It consists of all essential components necessary to gear on-line analytics. It also consists of off-premise back-end database to save data coming from on-premise transactional data sources. Analytical queries that are forwarded to IDAA appliance by on-premise accelerator plug-in run over this data, results are compiled and then handed back to it.

5.2.2.1 IDAA Server

IDAA server also named as accelerator server is an important component of on-premise IDAA appliance which communicates and interacts with DB2 for zOS. It also manages and distributes analytical or OLAP queries, that are forwarded by query optimizer process running on on-premise accelerator plug-in along DB2 for zOS, and performs system tasks required for on-line analytics [49].

5.2.2.2 IDAA Database

It is the storage unit of IBM DB2 analytical accelerator where it keeps data to use it for analytical purposes later. Data from DB2 for zOS is copied using change data capture and stored in IDAA database which is used by accelerator during the processing of analytical queries. DashDB serves the basis of IDAA database storage. It is a fully managed data warehouse solution by IBM, which integrates analytics as well along with high performance and scalability. Data from different source systems can be in its relational form with special data types such as geospatial data can be stored and then analyzed using simple SQL queries or built-in analytics. IBM dashDB also provides an option for data mining, comes up with ready-to-use data mining algorithms such as k-means clustering, decision tree clustering, linear regression, etc., and enables a faster insight of the data. It is integrated with IBM Netezza technology to perform in-database analytics [50].

5.2.3 Data Replication using CDC

Change data capture or in short CDC is an IBM product which implements incremental refresh replication approach to transfer recent changes found in source database log files to target database in near real-time. As of incremental refresh, CDC consists of source capture agent which reads transactional logs, parses them and sends them to target system where an apply agent is running and continuously listening which apply incoming changes to the respective tables and partitions in the target database. IBM DB2 Analytics Accelerator also makes use of CDC to transfer data from transactional source database, i. e., DB2 for zOS to the target back-end database that is connected to the IDAA appliance. IDAA uses data found in back-end database to carry out OLAP operations and prepares results. Below an idea is provided for using CDC with IDAA in its cloud deployment model and explain its few important components.

5.2.3.1 CDC Source Agent

Source agent, sometimes also termed as an apply agent is an important component of change data capture which is designed to gather data in near real-time from DB2 for zOS which stores transactional data, and send it to IBM DB2 Analytics Accelerator database (IBM dashDB in the case of IDAA cloud deployment mode). It is a part of another product of IBM named Change Data Capture that is specially tuned to synchronize source and target database systems. As soon as the transactional data sources get changed, the recent changes are taken and applied to the corresponding target. Unlike loading a complete table or a partition after a specific interval of time, CDC reads transactional logs that are created and updated by source database systems for redundancy and backup assurance. It helps to improve the overall performance of the ETL process by decreasing latency and near real-time replication of the data[1].

As IDAA is the integration of DB2 for zOS, processing power of Netezza technology and storage, it also makes use of the IBM CDC to take advantage of its near real-time data replication capability. CDC source agent as a sub-component of IDAA reads log files of DB2 database and sends them to apply agent. As the application user can exclude any table from acceleration, so it reads only those tables that are the part of acceleration.

As it is already mentioned in previous chapters that DB2 database writes every modification in the transactional data to its log files for backup and recovery needs. To read these files, CDC source agent comes up with a log reader appliance. Log reader is made intelligent enough to scan and read only relevant changes from tables and columns. Reading the full set of changes increases the size of data to be sent otherwise. It also keeps track of logs that have already been read in order to avoid duplications. It is an always-on appliance that reads log files continuously to read recent changes as soon as they appear in logs. Log reader pushes these changes in raw format into transactional queues which are useless in their current form for the target system unless they are processed to get useful data. Therefore, log parser processes these raw logs and parses them to get relevant data out of them that should be sent. It then stores extracted useful data in a staging storage and become available to be copied after commit and sent to the target system.

5.2.3.2 Communication Data Network

There are multiple approaches in which communication network can be tuned for IDAA to replicate data from DB2 for zOS to dashDB found in the off-premise public cloud. These include a dedicated private TCP/IP network connection or a shared TCP/IP network connection. These connections are usually backed up by another network connection which is enabled by accelerator itself in case of failure in primary connection. As in traditional deployment of

[1] https://www-01.ibm.com/software/data/replication/products.html

IDAA, the most commonly used communication channel is to have a private dedicated network connection so the same approach is used here to transfer data over wire in current deployment of IDAA on cloud. A dedicated private fiber network with a capacity of 10 GB is used to transfer data from source end to target. This network is very high speed, durable and highly available. These networks are fully managed by IDAA in order to provide seamless service to the DB2 for zOS users.

5.2.3.3 CDC Target Agent

CDC Target agent or apply agent runs as a standalone appliance running along IBM DB2 Analytics Accelerator to apply incoming changes from CDC source agent. As already mentioned, IDAA makes use of IBM Change Data Capture as a standalone product for data synchronization between source and target systems. Once data arrives at target end, CDC apply agent applies this data to the corresponding target tables in IBM dashDB respectively in other data systems of the accelerator. Apply agent with the help of metadata updates target database tables, files in case of file based databases and/or any other type of database system. Metadata helps apply agent to map incoming data to relevant tables and columns. Like CDC source agent, target agent is also an always-on appliance along IDAA so that data can be synchronized in near real-time as soon as they appear in the transaction database.

5.2.3.4 CDC Access Server

CDC Access Server is a monitoring and configuration service mainly responsible for management and administration of IDAA replication process which is enabled by IBM change data capture. It manages incremental update settings, captures and applies agent processes, CDC, subscriptions, table mappings etc. In a stand-alone CDC installation, CDC access server is directly accessible from administrative GUI, which allows to define table mappings (i. e., which tables should be replicated), start and stop replication process, add or remove subscription (a connection that is established to mirror data from source data system to a target data system. It contains information of the data that is replicated and how data can be applied to the target. In order to speed up the process of replication, multiple and parallel subscriptions can be used. CDC staging store usually saves data only in main memory if all subscriptions are currently active to speed up mirroring process. But if some subscription is not active, then it can save data persistently as well) and monitor replication process.

For the sake of a unified access to the accelerator and all its integrated components, this admin GUI of access server is integrated with IDAA administration process and all necessary responsibilities are handled by automation code that calls underlying APIs. Additional features are provided in IDAA studio to access server. As already mentioned, all IDAA studio administrative functionality is provided with DB2 stored procedures, so provision of access server in IDAA studio is also executed using these SPs.

5.3 A Web-based Application for Modeling Data Replication Costs

As already mentioned at the start of this chapter that implementation consists of basically two major parts. In the first part, i. e., Sec. 5.2 a deployment model of IBM DB2 Analytics Accelerator on a hybrid cloud is discussed, while in the second part a web based application of cost model that is developed in chapter 4 is described in detail. The idea of developing this web-app is to allow application users to quickly evaluate their replication system once they have necessary data in hand. More details are provided below.

5.3.1 Class-level Architecture

Fig. 5.3 shows the class diagram of a web application that is developed in the course of implementation of the cost model of an individual component in our unified replication system. The class diagram is constituted of many classes, each carrying unique responsibility in the implementation of the web application. A JSP (Java Server Pages) based client interacts with a web server, which provides a RESTful interface for client interaction. Once the server is requested, it calls component factory to get each individual module and then distributes respective parameters to them. Later it contacts each component to get values of latency, CPU usage, and throughput compiles them and sends them back to the client. Further details can be found in next sections.

5.3.1.1 JSP based Client

It is a client that is written using Java server pages (JSP) to interact with web-server using RESTful interface. It facilitates user to enter corresponding values of modeled parameters for each individual module (see Sec. 4.2.1) and to send a request to the web server to process this request to evaluate all modules and measure values of performance metrics. It is also able to get back the response from the web server, parse the incoming values and display corresponding values on the web page. In the class diagram of our application, JSP based client is modeled with a role named Client which decipher all these functionalities.

Along with this web application, a sample client is also developed(see Fig. 5.5) that can be used, in its very restrictive mode, to send a POST request to the web server via RESTful service. After selecting a particular operational mode, i. e., basic, transformation enabled, compression enabled, or full mode and performance metrics, i. e., latency, CPU usage, throughput, cost effectiveness, or price, the user needs to enter global and respective local parameters to each component. Upon hitting *Calculate* button, client prepares a JSON (JavaScript Object Notation) from entered parameters, sends a POST request to the server, parses response and displays returned data in corresponding sections.

5.3.1.2 RESTful Interface

The web server provides an interface that is written on the basis of REST (REpresentational State Transfer) based services. It implements a POST method of RESTful service that can be accessed by our JSP based client or any client that complies with this interface. This interface accepts parameters in JSON format, parse them and forward them to the server for further processing. Once results are compiled by the server, they are returned back to the client with the same JSON format but with different entries. The class "CalcMetrics" in class diagram maps to this RESTful interface.

5.3.1.3 Component Factory

In this class, factory pattern [51] is implemented to get each individual component of the unified replication system. Initially, all components are saved in an EnumValues class (a class that stores all components in Java enum structure and can be extended easily with new components in future). The required component using its corresponding enum value is passed to *getComponent()* method of this factory class which then returns the requested component back. Once a component is in hand it can be used to measure performance metrics, i. e., latency, CPU usage and throughput of that particular component. In our class diagram, this factory pattern is mapped to ComponentFactory class.

5.3.1.4 Component

In this component class, each individual component or module is implemented. They are the workhorses of our web application because each of them encapsulates the logic of its own cost model. The web application with the help of RESTful service distributes respective parameters to these components, which are then processed and compiled results of an individual module are sent back. Each individual component is implemented with Singleton [51] pattern which ensures a unique global instance of a particular component. In order to get instance, *getInstance()* method is called. No matter how many times *getInstance()* is called, each time global instance is returned. Using this component instance, further processing can be done to measure and get cost model. Each component also implements an interface, named Components in our class diagram, to offer a common set of methods.

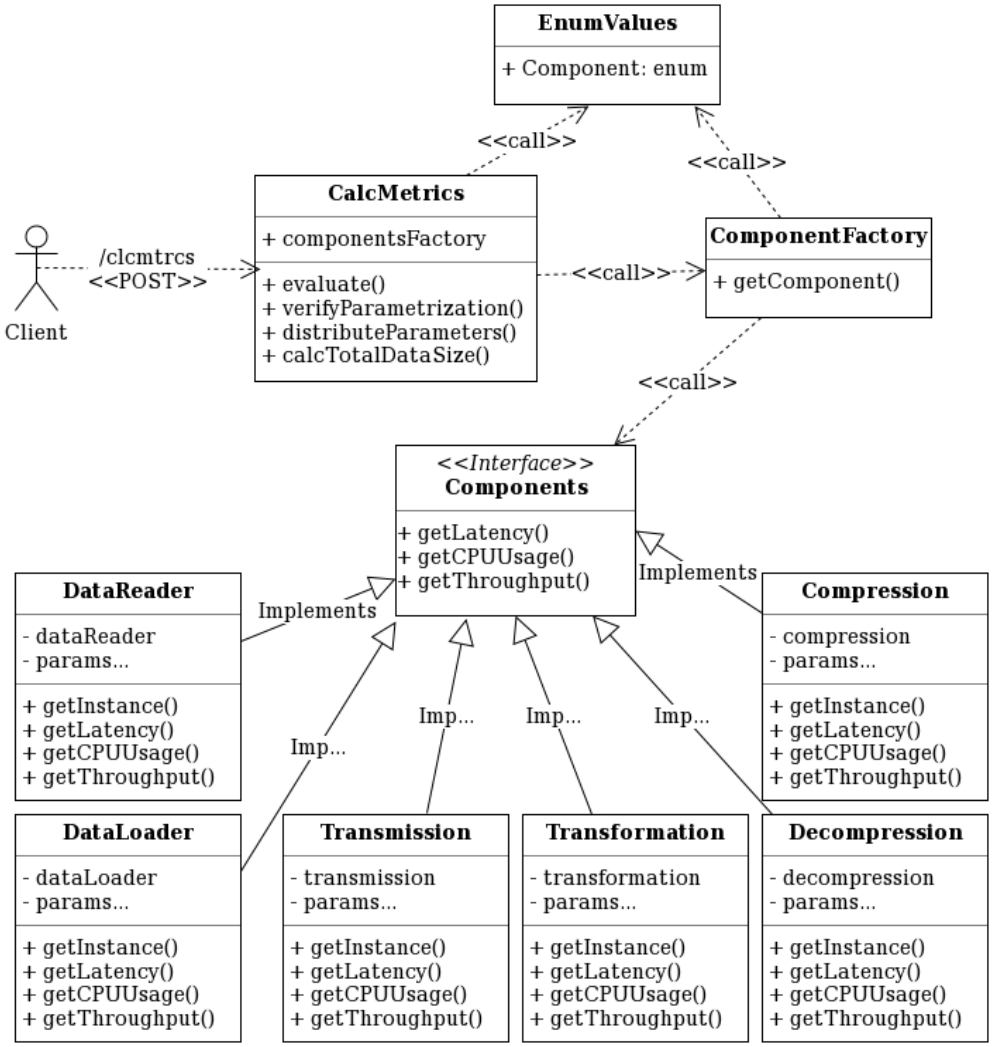

Figure 5.3: Class Diagram of the Application

5.3.2 Role Interaction

Fig. 5.4 shows the sequence diagram of our web application, explains the interaction between different roles of web-app and illustrates the sequence of flow of different interactions that are followed by a client request for a cost model of a particular replication system. The requests include necessary parameters in JSON format that are parsed and distributed by the application; results are compiled and sent back to the client.

The application is divided into four major roles, i. e., client, RESTful API, component factory, component. It starts with a request from a client asking for a cost model of the system with given specifics. These specifics or parameters are sent to the application in a pre-defined JSON format in the request body. This request is a POST method implemented and made available by RESTful service role. The RESTful service accepts this incoming request via POST method

and parses its body to get required parameters needed for an individual component to measure performance metrics. This service is also acting as a controller between client requests and components, which can also be made as a separate role. But for now, this service itself is responsible for managing all controller activities. Once the request is parsed, it asks the component factory to provide a requested component. The component factory invokes *getInstance* method of that particular component and returns it back to the service. The service then distributes respective parameters to the component and then asks for each individual performance metrics value. Once it is done with one component, it asks for another component from the component factory and repeats the same steps as mentioned for the first component. At the very end, RESTful API or service compiles all results together, bundles them again in a pre-defined JSON format and sends them back to the client as a response body.

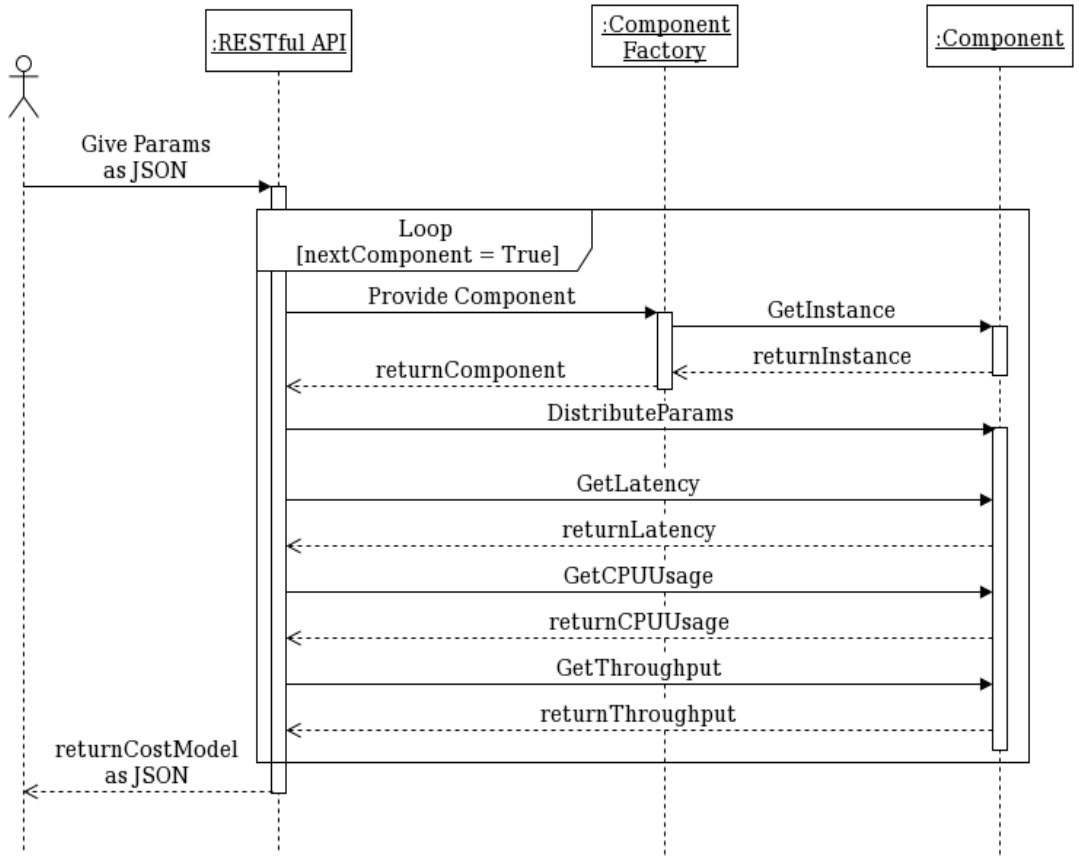

Figure 5.4: Sequence Diagram of the Application

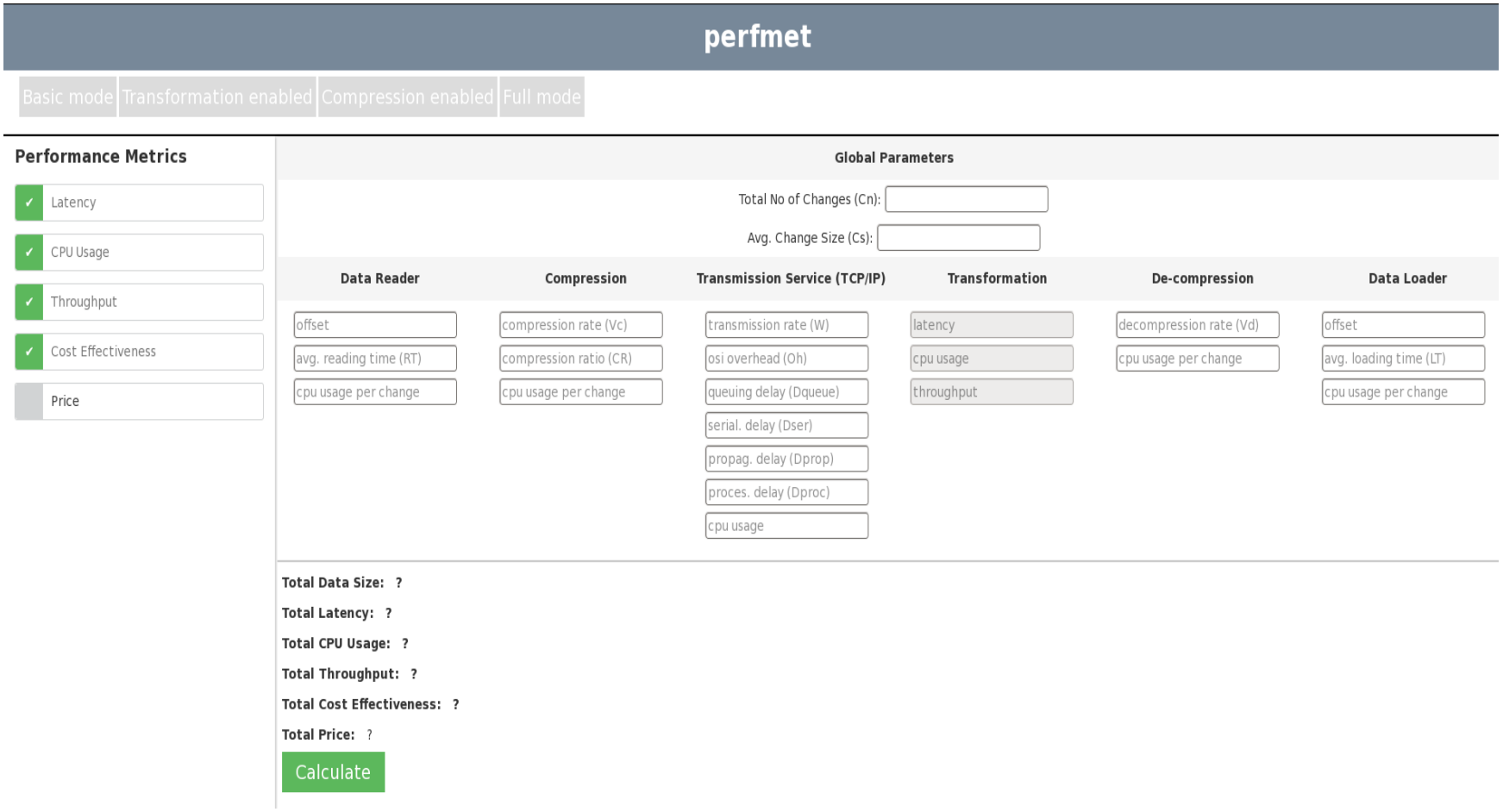

Figure 5.5: Web Application Interface

Chapter 6

Evaluation

In this chapter, the conceptual model developed in Chap. 4 is evaluated to test performance of replication strategies based on a test system. A very basic form of this conceptual model is implemented in Chap. 5. A test system with source and target databases is developed where data in the source system should be replicated to target system, and during this replication the important performance parameters are measured according to our conceptual and implemented model.

6.1 Test System Setup

Fig. 6.1 shows an overview of the test system that is used in the evaluation process. The main purpose of this test system here is to verify and determining the cost model of the basic route of unified pipeline model (refer to Sec. 4.2.1), simulating an IDAA-like setup. As at the time of writing this thesis, IDAA was under development and its use case presented in Sec. 5.2 can not be directly evaluated. But the test system presented here is a reflection of IDAA use case and can be mapped to any such a scenario. And the replication strategy that is applied in the test system is similar to Load Refresh. However, this system can be easily extended to equip with other routes in unified pipeline model and any other replication methodology.

This system consists of mainly two sub-systems, i. e., source and target systems. Each system is hooked up with MySQL database which acts as transactional source database and target database correspondingly. In order to transfer data from source database to target database, Export utility of MySQL is used which exports data from source database into a SQL file, which is then imported into target database using Import utility of MySQL database. To transfer SQL file (containing data of source database) from source system to target system, client/server socket paradigm is used. A server socket on source system sends this SQL file to client socket on target system using TCP/IP protocol.

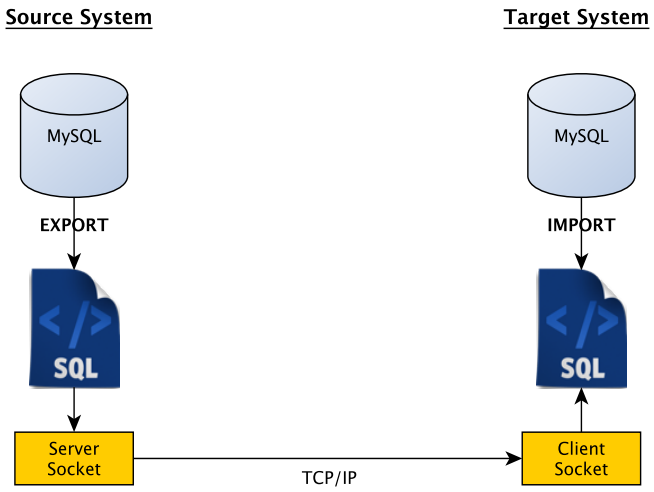

Figure 6.1: Evaluation Setup

6.1.1 System Configurations

As already mentioned, the system consists of two systems/machines, one running source database and the other running target database. The source machine is running with red hat enterprise Linux and powered with Intel core i7 processor at 2.7 GHz technology and a 16 GB of RAM. The target machine is a Fedora workstation that is powered with Intel dual core processor at 2.00 GHz technology and a 2 GB of RAM. It should be noted that the test system setup presented in Fig. 6.1 is not at all optimized according to underlying hardware because the basic purpose of our setup is to evaluate our conceptual model rather than optimizing a particular product for performance.

On source machine, MySQL running in a Docker[1] container is used, while on target machine MySQL database is directly installed. The latest version of MySQL available at the time of writing this thesis is used, i. e., 5.7.x. A temporary database named PERFMET is created on source database with a temporary table. The schema of this table is shown in Fig. 6.2. It consists of two only fields, i. e., ID type integer containing an identifier for each row and IMAGEDATA type longtext to store data URI of images. As already explained, with the help of database dump the complete source data is dumped to a SQL file which is then transferred to the target system via client/server socket file transfer mechanism.

[1] https://www.docker.com/

IMAGES	
ID	integer
IMAGEDATA	longtext

Figure 6.2: Test DB Schema

The two machines are connected via same network and the network bandwidth is measured using lperf[2]. This connection is not dedicated and hence total instantaneous speed depend on total usage by other users. A network bandwidth is measured varying between 7 - 10 Mbps. In order to neutralize this variation in network bandwidth to some extent, an average of maximum and minimum bandwidth is used, i. e., 8.5 Mbps or 1.0625 MBPS in the measurements. More precision in the evaluation results presented next could be achieved by using a dedicated internet connection between source and target machines.

6.1.2 Evaluation using Full Refresh

This section presents the results of the test system leveraging a replication of data from source database to target database. System setup and related configurations have already be discussed above.

6.1.2.1 Important Parameters

An image of size about 616.65 KB is stored in the only table of source database for 1861 times yielding a total size of the transactional store database to be $616.65KB \times 1861 = 1147.6MB = 1.147GB$. Therefore, the following important parameters are found as per the conceptual model:

C_s = 0.616 MB

C_n = 1861

S = 0.616 MB * 1861 = 1147.6 MB

W = 1.0625 MBPS

RT = 0.014 s

LT = 0.267 s

$offset$ = 0, (for full replication)

O_h = 0, (OSI overhead is ignored)

It should be noted here that reading time RT of one change from MySQL source database and loading time of one change LT to MySQL target database are measured using the Java interface of JAVA.SQL package[3] on each respective system.

[2] https://iperf.fr/

[3] https://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html

6.1.2.2 Latency

The latency of full replication in our test system for data reader is measured as per equation 4.4. So the total theoretical latency is:

$$L_{rdr} = 0 + (1861 * 0.014) = 26.05s$$

whereas the measured latency on the actual system was about 32s. The latency of data loader is also measured in a similar fashion using equation 4.37 and is:

$$L_{ldr} = 0 + (1861 * 0.267) = 496.9s$$

On the other hand measured latency of data loader was 420s. Finally, for the transmission service, the latency is calculated as per equation 4.19 considering propagation, processing, serialization and queuing delay to be zero. So the total theoretical value is:

$$L_{tms} = 0 + \frac{1147.6MB}{1.0625MBPS} = 1080.1s$$

and measured latency was around 1202 seconds. Fig. 6.3 shows these theoretical and measured values of latency in a graphical way to make a better comparison. The major difference in latencies is found in transmission service and that is because of unpredictable network connection. The result could be much better with more accurate network connection. The comparison concludes that the cost model is good enough to use in real IDAA-like systems to pre-assess the performance. Finally the overall theoretical latency of our complete test system can be measured using equation 4.40 which is:

$$L = 26.05 + 1080.1 + 496.9 = 1603.04s$$

while total measured latency was 1654s.

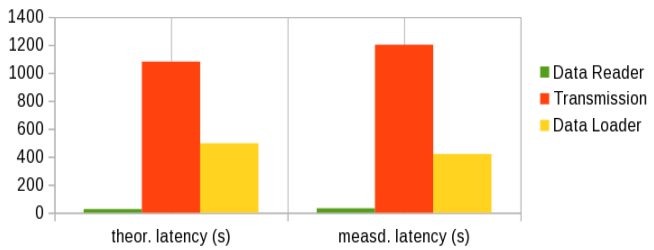

Figure 6.3: Latency Comparison

6.1.2.3 Throughput

The theoretical throughput of data reader is found using eq. 4.6 and the total value is:

$$R_{rdr} = \frac{1861}{26.05} = 71.44 changes/sec$$

and the measured value is 58 changes/sec. The theoretical throughput of data loader is found using eq. 4.39 and the total value is:

$$R_{ldr} = \frac{1861}{496.9} = 3.75 changes/sec$$

and the measured value is 4.43 changes/sec. Similarly the throughput of transmission service can be found using eq. 4.22 and the total value is:

$$R_{tms} = \frac{1.0625}{0.616MB} = 1.72 changes/sec$$

and the measured value is 1.54 changes/sec. Fig. 6.4 shows these theoretical and measured values of throughput in a graphical way to make a better comparison. As it is known, that latency directly impacts throughtput so the considerable difference in throughput is found in again transmission service and that is also because of unpredictable network connection. More accurate results could be achieved by using a fixed network connection. But still the results presented here are enough to verify the accuracy of cost model. Finally the overall theoretical throughput of our complete test system can be measured using equation 4.42 which is:

$$R = \min(71.44, 3.75, 1.72) = 1.72 changes/sec$$

while total measured throughput was 1.54 changes/sec.

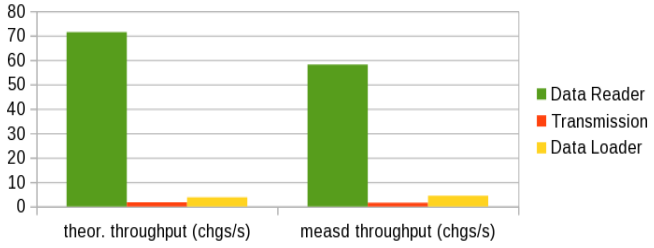

Figure 6.4: Throughput Comparison

6.1.2.4 CPU Utilization

In this section, only the measured values of CPU utilization are calculated. However, the corresponding theoretical values can be calculated using the cost model as specified in equations 4.5, 4.21, and 4.38.

The measured values of CPU utilization of each component in our test system are calculated using Linux profiling tools, i. e., perf[4]. The values are shown in Fig. 6.5, Fig. 6.6, Fig. 6.7, and Fig. 6.8. So the total CPU utilization as per equation4.41 is:

$$U = 0.046 + 0.045 + 0.006 + 0.030 = 0.127 CPUs$$

```
[tppool84@oc5041241167 bin]$ sudo perf stat -p 12802
^C
 Performance counter stats for process id '12802':

      3901.072146      task-clock (msec)         #    0.046 CPUs utilized
            5,953      context-switches          #    0.002 M/sec
              127      cpu-migrations            #    0.033 K/sec
              971      page-faults               #    0.249 K/sec
    8,650,838,417      cycles                    #    2.218 GHz                      (50.65%)
    5,652,110,007      stalled-cycles-frontend   #   65.34% frontend cycles idle     (50.79%)
   <not supported>     stalled-cycles-backend
   10,511,536,375      instructions              #    1.22  insns per cycle
                                                 #    0.54  stalled cycles per insn  (50.83%)
    2,433,005,562      branches                  #  623.676 M/sec                    (51.14%)
        2,570,948      branch-misses             #    0.11% of all branches          (51.15%)

     85.607845121 seconds time elapsed
```

Figure 6.5: Data Reader CPU Usage

```
Performance counter stats for process id '5462':

     29971.646227      task-clock (msec)         #    0.045 CPUs utilized
           38,771      context-switches          #    0.001 M/sec
            1,693      cpu-migrations            #    0.056 K/sec
              754      page-faults               #    0.025 K/sec
   52,321,729,565      cycles                    #    1.746 GHz
     (24.96%)
   96,563,765,530      instructions              #    1.85  insn per cycle
     (24.93%)
   24,433,612,879      branches                  #  815.224 M/sec
     (25.08%)
       12,430,051      branch-misses             #    0.05% of all branches
```

Figure 6.6: Data Loader CPU Usage

6.2 Summary

To wrap it all into a single paragraph, we have setup a test system to evaluate our cost model. This system was consisting of source, target and communication components. About 1.1 Gbytes of data was stored in the source database. This data was read and then transferred using TCP/IP

[4]https://perf.wiki.kernel.org/index.php/Main˙Page

```
[tppool84@oc5041241167 bin]$ perf stat -p 7912
^C
 Performance counter stats for process id '7912':

       8584.063609      task-clock (msec)         #    0.006 CPUs utilized
            51,051      context-switches          #    0.006 M/sec
             1,088      cpu-migrations            #    0.127 K/sec
             5,176      page-faults               #    0.603 K/sec
    12,581,817,654      cycles                    #    1.466 GHz                      (74.79%)
     8,505,054,967      stalled-cycles-frontend   #   67.60% frontend cycles idle     (77.89%)
   <not supported>      stalled-cycles-backend
     7,858,200,994      instructions              #    0.62  insns per cycle
                                                  #    1.08  stalled cycles per insn  (78.31%)
     1,418,814,337      branches                  #  165.285 M/sec                    (79.62%)
        47,881,606      branch-misses             #    3.37% of all branches          (80.90%)

    1559.778140848 seconds time elapsed
```

Figure 6.7: Source TMS Process CPU Usage

```
[usman@PC192-168-2-108 bin]$ perf stat -p 3609
^C
 Performance counter stats for process id '3609':

      46081.134248      task-clock (msec)         #    0.030 CPUs utilized
           852,160      context-switches          #    0.018 M/sec
            23,011      cpu-migrations            #    0.499 K/sec
               349      page-faults               #    0.008 K/sec
    58,980,903,793      cycles                    #    1.280 GHz                      (24.76%)
    20,168,829,744      instructions              #    0.34  insn per cycle           (25.12%)
     3,144,746,438      branches                  #   68.244 M/sec                    (25.05%)
```

Figure 6.8: Target TMS Process CPU Usage

from the source database to target side where it was loaded into the target database. During this process, we calculated theoretical and actual values of latency and throughput, whereas only actual values of CPU usage were determined. We also mentioned which equation from our cost model we used to measure theoretical values. This test system can be used as an example to use our cost model in real setup.

Chapter 7

Conclusion and Outlook

7.1 Conclusion

It is a massive challenge to synchronize source and target databases when the data in data sources is continuously updated due day-to-day transactions. Given the certain workload, any one of the available data replication strategies can be used. Before hiring an online analytical service that is supposed to provide analytics on the current customer's data, it is very helpful to pre-assess whether the actual data can be duplicated from transactional data sources to the target databases in a right time using a certain replication methodology. And which certain component in the system should be optimized to get a fair performance on given workload.

In the course of this thesis, a framework to model cost and monetary metrics for different replication strategies was developed. First of all, two different but mostly used replication techniques were combined and a unified pipeline model was crafted out which equally fits on any of two methodologies. This pipeline model consists of different routes, and each route involves multiple components, and they carry a unique responsibility in data replication. For each component in that pipeline model, we have developed a cost model which determines important performance metrics, i. e., latency, CPU usage, and throughput. Using this cost model, a pre-evaluation of data synchronization can be achieved. A monetary model of the overall system was also presented which can be used to estimate financial aspects of the system.

The user should select a particular route in our generalized unified pipeline model which he wants to leverage in his system to replicate data from transactional source databases to the target databases. For each component in that route, he needs to find out important parameters that are discussed in our cost model. Once they are determined, use our framework and apply formulas to measure probable values of performance metrics in the real system. In the course of

this thesis, a web application was developed that implements a basic use case of our cost model and can be used to measure the known performance metrics directly.

Finally, for the evaluation of our framework, we created a real sample system which transfers a certain set of data changes from a transient data source to a target database. The suitable route of the available routes in unified pipeline model was determined that fits into the test system and find out individual component and applied our cost model on them to find out theoretical values of performance metrics, which are then compared with real values as well.

The cost model along with implemented web application makes it really simple to measure cost and monetary metrics required to synchronize source and target databases. It assists in estimating the worth of the overall system before investing a lot of money to buy any off-premise analytical system whether it fits one's requirements.

7.2 Outlook

This thesis covers the actual problem in all possible angles to make our cost model very robust so that it can be applied to any replication strategy with any sort of workload. The solution presented is very generalized which can be more optimized for a certain product. The basics of this framework have already been laid down with some advanced topics taken into account. But depending upon a specific product or algorithm used in each component, the cost model can be made more specialized. If new metrics are to be identified as the part of cost optimization future, this cost model can be easily applied as it was used in finding CE above..

Similarly, the web application that we developed in the course of this thesis covers just a very basic use case, which can be extended further to cover all options including all routes in unified pipeline model and other transmission service choices like P2P and Pub/Sub messaging as discussed in earlier chapters.

The cost model of transformation model can be further extended by determining a specific set of operations which are not identified yet.

In order to optimize the performance of this cost model, a further very deep study (although not needed) can be performed on each individual component that should take hardware and environmental affects into account as well. However, it should be kept in mind that the performance is a never-ending tale which can only be improved with further regressions without a final full stop.

Bibliography

[1] K. Bohn, "Understanding capex vs. opex in a cloud computing world." http://mspmentor.net/site-files/mspmentor.net/files/uploads/2016/04/Arrow01.pdf.

[2] J. Myers, "Analytics in the cloud," 2015.

[3] S. Chaudhuri and U. Dayal, "An overview of data warehousing and olap technology," *SIGMOD Rec.*, vol. 26, pp. 65–74, Mar. 1997.

[4] H. G. Andreas Bauer, *Data-Warehouse-Systeme*. dpunkt, 2004.

[5] Oracle, "What is an oltp system?,"

[6] K. Prajapati, "Fasmi test definition," 2012. http://www.bigator.com/2012/09/07/what-is-olap-database-what-is-fasmi/.

[7] C. S. E.F. Codd, S.B. Codd, "Providing olap to user-analysts: An it mandate," 1993.

[8] T. G. Peter Mell, "The nist definition of cloud computing," 2011. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[9] B. Kepes, "Understanding the cloud computing stack: Saas, paas, iaas,"

[10] A. Rajkumar, James, *Cloud Computing Principles and Paradigms*.

[11] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer, 2014.

[12] R. S. J. F. K. H. Mark Hapner, Rich Burridge, *Java Messaging Service API Tutorial and Reference*. 2002.

[13] J. Jenkov, "Software architecture," 2014. http://tutorials.jenkov.com/software-architecture/index.html.

[14] S. Krakowiak, *Middleware Architecture with Patterns and Frameworks*. 2009.

[15] A. Vavouras, S. Gatziu, and K. R. Dittrich, *The SIRIUS Approach for Refreshing Data Warehouses Incrementally*, pp. 80–96. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.

[16] IBMCDC, "Ibm infosphere change data capture." http://www-03.ibm.com/software/products/en/ibminfochandatacapt.

[17] "Oracle change data capture help center."

[18] J. Arnold, *Implementing IBM InfoSphere Change Data Capture for DB2 z/OS V6.5*. 2011.

[19] H. Garcia-Molina and W. J. Labio, "Efficient snapshot differential algorithms for data warehousing," tech. rep., Stanford, CA, USA, 1996.

[20] M. R. R. Kimball, L. Reeves, *The Data Warehouse Lifecycle Toolkit*. 1998.

[21] V. Beal, "Latency." http://www.webopedia.com/TERM/L/latency.html.

[22] IBM, "Sychronizing data in ibm db2 analytics accelerator for z/os," tech. rep., IBM, 2013. http://www-01.ibm.com/support/docview.wss?uid=swg27038501.

[23] T. A. Majchrzak, T. Jansen, and H. Kuchen, "Efficiency evaluation of open source etl tools," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, (New York, NY, USA), pp. 287–294, ACM, 2011.

[24] A. M. Tho, M. Nguyen; Tjoa, "Zero-latency data warehousing for heterogeneous data sources and continuous data streams," in *Conference: iiWAS'2003 - The Fifth International Conference on Information Integrationand Web-based Applications Services, 15-17 September 2003, Jakarta, Indonesia*, 2003.

[25] M. J. Eccles, *Pragmatic Development of Service Based Real-Time Change Data Capture*. PhD thesis, 2012.

[26] M. Rouse, "Change data capture," 2007. http://whatis.techtarget.com/definition/change-data-capture-CDC.

[27] A. Pareek, "Addressing bi transactional flows in the real-time enterprise using goldengate tdm," in *Enabling Real-Time Business Intelligence*, 2009.

[28] M. Bouzeghoub, "A framework for analysis of data freshness," in *Proceedings of the 2004 International Workshop on Information Quality in Information Systems*, IQIS '04, (New York, NY, USA), pp. 59–67, ACM, 2004. http://doi.acm.org/10.1145/1012453.1012464.

[29] J. Raitto, "Oracle change data capture," 2004. http://nyoug.org/Presentations/SIG/LI/lisigcdc.ppt.

[30] *Impact of HTTP Compression on Web Response Time in Asymmetrical Wireless Network*, 2009.

[31] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, "Potential benefits of delta encoding and data compression for http," in *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '97, (New York, NY, USA), pp. 181–194, ACM, 1997.

[32] *Analysis of Transport Optimization Techniques*, 2006.

[33] W. Dong, X. Chen, S. Xu, W. Wang, and G. Wei, "Proxy-based object packaging and compression: A web acceleration scheme for umts," in *Proceedings of the 5th International Conference on Wireless Communications, Networking and Mobile Computing*, WiCOM'09, (Piscataway, NJ, USA), pp. 4965–4969, IEEE Press, 2009.

[34] *Improving I/O Forwarding Throughput with Data Compression*.

[35] D. G. A. K. M. P. Paolo Bruni, Marcelo Antonelli, *DB2 9 for z/OS: Using the Utilities Suite*. 2010.

[36] IBM, "Configuring and monitoring the incremental update process," 2016. http://www-01.ibm.com/support/docview.wss?uid=swg27039147.

[37] M. TechNet, "Monitoring the change data capture process," 2013. https://technet.microsoft.com/en-us/library/cc627415

[38] I. Cisco Systems, "Design best practices for latency optimization."

[39] K. W. R. James F. Kurose, *Computer Networking: A Top-Down Approach*. Pearson.

[40] D. J. W. Andrew S. Tanenbaum, *Computernetzwerke*.

[41] SPEC, "Spec jms 2007," 2007. https://www.spec.org/jms2007/.

[42] J. B. A. B. Kai Sachsa, Samuel Kouneva, "Performance evaluation of message-oriented middleware using the specjms2007 benchmark," in *Performance Evaluation*, 2009.

[43] IBM, "Ibm cdc information center," http://www.ibm.com/support/knowledgecenter/en/SSX3HK.

[44] R. Götz, "Ibm db2 analytics accelerator," 2011. http://www-05.ibm.com/de/events/netezza˙webcasts/pdf/Netezza-WebCast-DB2-Analytics-Accelerator.pdf.

[45] P. Francisco, *IBM PureData System for Analytics Architecture*. 2014. http://www.redbooks.ibm.com/redpapers/pdfs/redp4725.pdf.

[46] D. Martin, O. Koeth, J. Kern, and I. Ivanova, "Near real-time analytics with ibm db2 analytics accelerator," in *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, (New York, NY, USA), pp. 579–588, ACM, 2013.

[47] IBM, "Idaa guides and manuals," 2014. http://www-01.ibm.com/support/docview.wss?uid=swg27040146.

[48] K. Stolze, G. M. Lohman, V. Raman, R. Sidle, and F. Beier, "Evolutionary integration of in-memory database technology into ibm's enterprise db2 database systems," in *GI*, 2013.

[49] L. C. . Paolo Bruni, Jason Arnold, *Reliability and Performance with IBM DB2 Analytics Accelerator*. 2014.

[50] "Ibm dashdb." http://www.ibm.com/analytics/us/en/technology/cloud-data-services/dashdb/.

[51] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Abstraction and Reuse of Object-Oriented Design*, pp. 406–431. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993.

Appendix A

Appendix Part1

Development code that is relevant to Chap. 5 and Chap. 6 is provided separately.