MULTIOBJECTIVE RELATIONAL DATA WAREHOUSE DESIGN FOR THE CLOUD

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

TANSEL DÖKEROĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

SEPTEMBER 2014

Approval of the thesis:

MULTIOBJECTIVE RELATIONAL DATA WAREHOUSE DESIGN FOR THE CLOUD

submitted by **TANSEL DÖKEROĞLU** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

ABSTRACT

MULTIOBJECTIVE RELATIONAL DATA WAREHOUSE DESIGN
FOR THE CLOUD

Dökeroğlu, Tansel
Ph.D., Department of Computer Engineering
Supervisor : Prof. Dr. Ahmet Coşar

September 2014, 137 pages

Conventional distributed Data Warehouse (DW) design techniques seek to assign data tables/fragments to a given static database hardware setting optimally. However; it is now possible to use elastic virtual resources provided by the Cloud environment, thus achieve reductions in both the execution time and the monetary cost of a DW system within predefined budget and response time constraints. Finding an optimal assignment plan for database tables to machines for this design problem is NP-Hard. Therefore, robust multiobjective heuristic algorithms are needed for cost-efficient Cloud DWs in terms of query workload response time and the total ownership price of virtual resources (CPU and/or cores, RAM, hard disk storage, network bandwidth, and disk I/O bandwidth).

In this thesis we propose two algorithms for the solution of the relational Cloud DW design problem; (1) Multiobjective Design with Branch and Bound (MOD-B&B) and (2) Multiobjective Evolutionary Genetic Algorithm (MOD-GA). These algorithms make use of a novel Cloud DW single query optimizer, DPACO, that can find the best distributed query execution plan and accurately calculate its response time. By using DPACO on an input query workload we find the best query execution plans for given query workloads using the given virtual resource allocations.

The best allocation of virtual resources for a DW design is achieved by using MOD-GA. We developed a special chromosome structure, along with crossover and mutation operators, to achieve the best results from MOD-GA. We experimentally verified the accuracy of the algorithm by comparing its output designs against the optimal designs obtained by using an exhaustive MOD-B&B algorithm. Our evaluations show that the obtained designs are very close to the optimal solution set and while MOD-B&B algorithm requires hours to complete its execution, the MOD-GA is able to return almost the same results within seconds.

In order to achieve further improvement in total response time of a query workload with monetary savings from Cloud resources, we improved the Cloud DW designs by using (near-) optimal and cost-efficient materialized views. Through our experiments performed on a private Cloud server, remarkable improvements in both response times of query workloads and monetary costs of consumed Cloud resources have been achieved. The reason for these savings is that, by materializing join results on hard disk, we obtain large CPU resource savings reducing Cloud cost, offsetting the cost of extra hard disk storage by a wide margin.

Keywords: Cloud data warehouse, virtual resource, materialized view, assignment

ÖZ

BULUT İÇİN ÇOK AMAÇLI İLİŞKİSEL
VERİ AMBARI TASARIMI

Dökeroğlu, Tansel
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Ahmet Coşar

Eylül 2014 , 137 sayfa

Günümüz dağıtık veri ambarları tasarım teknikleri veri tablolarını daha önceden belirlenmiş bilgisayar donanımlarına en iyi şekilde atayan yöntemler kullanmaktadır. Bulut bilişimin kullanılmaya başlanması ile birlikte ilişkisel dağıtık veri ambarlarını alternatif sanal donanımlar ile daha maliyet etkin tasarlayabilmek mümkün olmaktadır. Bu tasarımı en iyi şekilde yapabilmek NP-Zor bir problem olduğu için tecrübeye dayalı ve güvenilir algoritmaların kullanılması kaçınılmazdır. Bu algoritmalar çok amaçlı olarak sorguların cevap zamanlarını ve bulut bilişim üzerinde en uygun maliyetli olmasını sağlamalıdır. Sanal makine tipleri, veri saklama, iletişim ağı ve I/O genişliği, maliyetleri dikkate alınması gereken noktalardır.

Bu tez ile birlikte, iki değişik algoritma önerilmektedir. Problemin çözümü için önerdiğimiz bu algoritmalar, çok amaçlı dallanma-sınırlandırma algoritması ve genetik algoritmadır. Bu algoritmalar yeni geliştirilen çok amaçlı bir sorgu eniyileme yazılımı, DPACO, kullanır ve bunun ile en iyi tasarımı elde etmeye çalışır. DPACO eniyileme yazılımı ile, sanal kaynakları belli olan dağıtık veri ambarı tasarımlarının sorgu yüklerinin cevap verme süreleri yaklaşık olarak hesaplanmaktadır.

Algoritmalardan tasarımı en iyi gerçekleştiren olarak genetik algoritma tespit edildi. Bu algoritma için yeni bir kromozon yapısı ile birlikte, çaprazlama ve mutasyon operatörleri geliştirildi. Genetik algoritmanın etkinliği en iyi sonuçları bulan çok amaçlı dallanma-sınırlandırma algoritması ile karşılaştırıldı. Sonuçların birbirlerine çok yakın olduğu gözlendi. Bunun yanında, genetik algoritma çözümleri saniyeler içerisinde bulmayı başarırken, dallanma-sınırlandırma algoritması saatlerce çalışarak eniyilemeyi gerçekleştirebildi.

Ayrıca maliyet etkin maddeleştirilmiş görünümler kullanılarak tasarımların kalitesi arttırıldı. Özel bir bulut bilişim sunumcusu üzerinde yapılan deneyler sonucunda, tasarlanan dağıtık bulut veri ambarlarının ve geliştirilen algoritmaların etkinliği doğrulandı. Harcanan bütçe ve sorgu sürelerinin iyileştirilmesinde belirgin ilerlemeler rapor edildi.

Anahtar Kelimeler: Bulut veri ambarı, sanal bilgisayarlar, maddeleştirilmiş görünüm, atama

dedicated to my beloved family

ACKNOWLEDGMENTS

First and foremost I want to thank my supervisor Prof. Dr. Ahmet Coşar for all the support and encouragement he gave me. His guidance and positive approach throughout this time period made possible to complete this study.

I also would like to thank to Prof. Dr. Adnan Yazıcı, Prof. Dr. Özgür Ulusoy, Prof. Dr. Veysi İşler, and Assist. Prof. Dr. İsmail Sengör Altıngövde for providing their valuable feedbacks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

ACID	Atomicity, Consistency, Isolation, Durability
ACO	Ant Colony Optimization
AWS	Amazon Web Service
API	Application programming interface
BASE	Basically Available, Soft state, Eventual consistent
BI	Business Intelligence
DAG	Directed Acyclic Graph
DBaaS	Database as a service
DBMS	Database Management System
DDBMS	Distributed Database Management System
DP	Dynamic Programming
DPACO	Dynamic Programming with Ant Colony Optimization
DQO	Distributed Query Optimization
DW	Data Warehouse
EPT	Extended Page Table
FC	Fibre Channel
FIPS	Fully Informed Particle Swarm
GA	Genetic Algorithm
HPC	High Performance Computing
IaaS	Infrastructure as a Service
IDP	Iterative Dynamic Programming
MOBB	Multiobjective Branch and Bound Algorithm
MOD-B&B	**M**ulti**O**bjective Data Warehouse **D**esign with **B**ranch-and-**B**ound
MOD-GA	**M**ulti**O**bjective Data Warehouse **D**esign with with **G**enetic **A**lgorithm
MOLAP	Multidimensional Online Analytical Processing
MOGA	Multiobjective Genetic Algorithm
MQO	Multiple Query Optimization
MPP	Massively Parallel Processors
MVPP	Materialized View Processing Plan
NIC	Network Interface Controller
NIST	National Institute of Standards and Technology
OLAP	Online Analytical Processing
OS	Operating System
OSI	Open Systems Interconnection
PaaS	Platform as a Service
ROLAP	Relational Online Analytical Processing

ROWA	Read-Once, Write All
RVI	Rapid Virtualization Indexing
SLA	Service Level Agreement
SMP	Symmetric Multiple Processor
SynIC	Synthetic Interrupt Controller
SaaS	Software as a Service
SAN	Storage Area Networks
SGA	Simple Genetic Algorithm
SSL	Search Space Limit
QEP	Query Execution Plan
TPC-H	Database Benchmark for Decision Support Systems
VM	Virtual Machine
VMM	Virtual Machine Monitor

CHAPTER 1

INTRODUCTION

In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox. We should not be trying for bigger computers but for more system of computers.

Grace Murray Hopper[1]

Cloud computing has emerged as a new computation paradigm that builds elastic and scalable software systems. Vendors such as Amazon, Google, Microsoft, and Salesforce offer several options for computing infrastructures, platforms, and software systems [7][67][169][129] and supply highly-scalable database services with simplified interfaces for reducing the total cost of ownership [6][9][170]. Users pay all costs associated with hosting and querying their data where database-as-a-service providers present different choices to trade-off price and performance to increase the satisfaction of the customers and optimize the overall performance [14][163]. Recently, extensive academic and commercial research is being done to construct self-tuning, efficient, and resource-economic Cloud database services that serve to the benefits of both the customers and the vendors [43][87][90][115].

Virtualization that provides the illusion of infinite resources in many respects is the main enabling technology of Cloud computing [15]. This skill is being used to simplify the management of physical machines and provide efficient systems. The perception of hardware and software resources is decoupled from the actual implementation and the virtual resources s are mapped to real physical resources. Through mapping virtual resources to physical ones as needed, the virtualization can be used by

[1] (1906-1992) The developer of the first compiler.

several databases that are located on physical servers to share and change the allocation of resources according to query workloads [144]. This capability of virtualization provides efficient Cloud Data Warehouses (DW) where each Virtual Machine (VM) has its own operating system and resources (CPU, main memory, network bandwidth, etc.) that are controlled by using a VM Monitor (VMM) [2][125][143][173].

1.1 Problem Statement

In addition to answering queries efficiently in accordance with the Service Level Agreements (SLA), contemporary relational Cloud DW systems need to optimize a multicriteria problem that the overall cost of hardware ownership price must also be minimized. This problem can be be stated more specifically as:

'*Given a budget constraint and a query workload, how can the tables and available virtual resources of the Cloud (CPU, main memory, network bandwidth, etc.) be allocated to a set of VMs, each having a part of a distributed database, that the best overall query performance can be achieved with minimum pricing*? '

In order to solve this important design problem, we develop a framework that produces cost-efficient DW designs by using alternative virtual resource configurations, assignment of data tables to VMs, and determining the selection of beneficial materialized views.

A budgetary constraint can be a more important criterion for a finance manager, whereas the response time of the queries is more crucial for a database administrator [52]. Therefore, to fully realize the potential of the Cloud, well configured virtual resources, the location of the tables, and the materialized views are explored instead of optimizing query workloads on statically designed virtual resources [28][56][118][166]. This means that instead of designing the database over standard VMs, we configure the given virtual resources and the location of the tables for a cost-efficient DW to handle query workloads successfully in terms of monetary cost and response time.

In order to explain our multiobjective problem, we give an illustrative example. When a query workload of a TPC-H decision support benchmark with different virtual resources and table locations is executed, we observe significant performance changes in terms of monetary cost and response times.

Table 1.1: Pareto-optimal solutions for 25GB TPC-H database and query workload (Elap.T.= Time Elapsed in the Cloud)

Conf.#	VMs and Assigned Tables	Netw. Mbps	Elap.T. (sec.)	Elap.T.Cost (¢)
1	XL (L,P,O,C,PS,S,R,N)	-	1,423	31.3
2	XL (L,O,C,S,R,N) XL (P,PS,S,R,N)	200	337	13.8
3	XL (L,O,S,R,N) M (PS,S,R,N) XS (P,S,R,N)	200	902	28.4
4	XL (O,PS,S,R,N) XL (L,S,R,N) L (P,S,R,N) M (C,S,R,N)	100	620	31.3
5	XL (L,O,S,R,N) M (PS,S,R,N) XS (P,S,R,N) M (C,S,R,N)	200	836	29.0

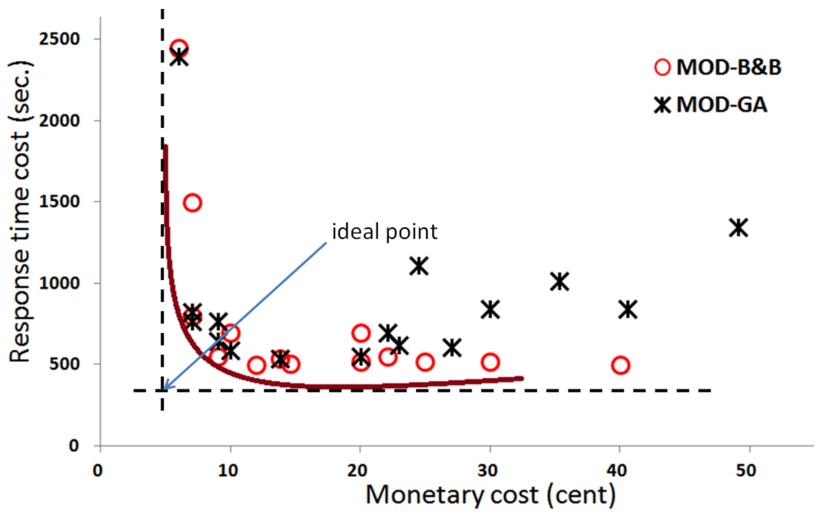


Figure 1.1: Pareto-optimal solutions for 25GB TPC-H database query workload for selected virtual resources and table locations that are obtained from our proposed algorithms.

Table 1.1 presents alternative number of VMs, types of the VMs, network bandwidth, the locations of the tables, response time of the workload, and the monetary cost for a sample Cloud DW design (details of the Cloud infrastructure are given in Chapter 4). The pareto-optimal visualization of the alternative designs that we obtain with our proposed algorithms are presented in Figure 1.1. The hypothetical ***ideal point*** is the main objective of the design problem.

Formally, our optimization problem can be stated as in Equation 1.1 (details are presented in Chapter 4).

ϕ is the configurations for the selected virtual machines on which the DW tables will be placed, τ is the assigned VM locations of the physical tables, and last Q is the set of queries in the workload. The resource consumption of the design is estimated in accordance with the best query execution plans of the queries.

$$min_{<\phi,\tau,Q>\epsilon S}(\sum_{i=1}^{\#VM}\sum_{j=1}^{\#Q} Cost(VM_i, Best\_Plan(\phi,\tau,Q_j))) \tag{1.1}$$

1.2 Main Goals and Contributions

The studies concerning the performance of the Cloud DWs are at their early ages. Most of the distributed database design and optimization concepts can be applied to this area however; multiobjective optimization on the Cloud has many new issues to research. There are many studies for tuning database system for specific workloads [167] [3] [150] [174]. On the other hand, to the best of our knowledge, there is no approach like ours that concerns both with the optimization of the total ownership price and the performance of the queries of a distributed DW by taking into account alternative virtual resource allocation, data table assignment, and selection materialized views.

Our study proposes two different algorithms for the solution of the problem, Multiobjective Branch and Bound (MOD-B&B) and Multiobjective Evolutionary Genetic Algorithm (MOD-GA). The algorithms focus on the elasticity of virtual Cloud resources and produce multiple virtual resource and table allocation plans for a set of queries in a workload, enabling the user to select the desired tradeoff with efficient cost models.

Materialized views are effective techniques for speeding up query workloads and they are increasingly being used by commercial DWs. Materialized views are specially good for DWs because of their intensive usage of common subexpressions including select-project-join operations. In our study, after finding the best configuration of VMs and table locations for the DW design, we select the most appropriate materialized views to reduce the response time, communication cost, and the ownership price of a relational DW with respect to the pricing scheme of the Cloud vendors. We observe that in addition to reducing the response time of the queries, total ownership price decreases significantly with appropriate use of the materialized views. Although storing/maintainig the selected materialized view has an additional storage cost, it is a very effective way of executing queries when indexes are constructed on the views. No resource deployment processing system deals with the concept of elasticity and cost-efficiency of relational Cloud DWs that make use of the appropriate materialized views like our system.

In addition to these contributions, we propose a novel heuristic algorithm, Dynamic Programming with Ant Colony Optimization Algorithm (DPACO), for relational Cloud DWs query optimization. The main focus of the proposed algorithm is the optimization of the multi-way chain join queries that spend most of the query execution time. The algorithm finds (near-)optimal fast response times for the queries [47, 48, 49]. The proposed design algorithms make use of DPACO algorithm while optimizing the response time of the query workloads on a given configuration of VMs and table locations.

Table 1.2 presents an overall comparison of our design method with other existing methods. Cloud (virtual resources), cost-awareness (Multiobjective), data allocation (location of Tables), materialized views, replication, fragmentation, designing at run-time, and MapReduce-based systems are the issues given in the Table. Our system is static and optimizes data allocation, virtual resource assignment, replication, cost-awareness, and optimized materialized views. These features of our system make it unique when compared with the previous design methods. Recent studies are dynamic that they balance the virtual resource consumption by migrating the data between VMs at run-time. The size of the relational databases in these studies are not big (in MB scale). However; Cloud DWs handle large amount of data up to TB

Table 1.2: Comparison of our design model with the other existing methods

	Cloud (Virtual Resources)	Cost-Awareness (Multiobjective)	Data Allocation	Materialized View	Replication	Fragmentation	Designing at run-time	MapReduce
Stonebraker et al. (1996) [149]		✓						
Sidell et al. (1996) [139]			✓		✓	✓	✓	
Tamhankar and Ram (1998) [155]			✓		✓	✓		
Ahmad et al. (2002) [5]			✓					
Zilio et al. (2004) [183]						✓		
Dash et al. (2009) [43]		✓				✓	✓	
Hauglid et al. (2010) [78]			✓		✓	✓	✓	
Soror et al. (2010) [144]	✓		✓				✓	
Kllapi et al. (2011) [90]		✓						✓
Xiong et al. (2011) [174]	✓	✓			✓		✓	
Warneke and Kao (2011) [166]	✓	✓					✓	✓
Curino et al. (2011) [41]	✓						✓	
Elmore et al. (2011) [53]			✓				✓	
Nguyen et al. (2012) [115]		✓		✓				
Koutris et al. (2013) [95]		✓					✓	✓
Our Study (2014)	✓	✓	✓	✓	✓			

level therefore, we propose a well configured static design method for rarely updated relational Cloud DWs.

1.3 Structure of the Thesis

The remainder of this thesis is organized as follows. In Chapter 2, we provide information about the virtualization of the resources and the scalability of the Cloud DWs before dealing with multiobjective design issues of relational Cloud DWs. We explain existing virtual and dynamically changing (scaling up/out and shrinking back) environment of Cloud DWs to analyze/comprehend the problem better and see the differences from the traditional database design problems. Layered architecture of

Cloud computing environment, virtualization (including the network virtualization), the existing infrastructure of relational Cloud DWs, query cost models, related query optimization techniques, and selected view materialization are explained.

In Chapter 3, we introduce our novel query optimizer (DPACO) for relational Cloud DWs and give its experimental results. The query optimization is an important issue for Cloud DWs in case you have several alternatives to execute a query. Our proposed algorithm makes use of the state-of-the-art contemporary metaheuristic, Ant Colony Optimization (ACO). It is efficient and has never been applied to this problem before. The algorithm has a single optimization objective, finding a fast response time, and it is implemented to be used both for distributed databases and shared-nothing Cloud DW architectures. The design algorithms proposed in Chapter 4 use DPACO algorithm while evaluating the response time of the query workloads with given VM configurations.

In Chapter 4, we define our multiobjective DW design formulation and present our algorithms. Infrastructure and pricing scheme parameters of the Cloud are presented. Two algorithms are proposed for the multiobjective Cloud DW design. (Near-) optimal parameter settings for MOD-GA are presented at the end of this Chapter.

In Chapter 5, we define our approach for selecting materialized views on relational Cloud DWs. We make use of a previously defined cost model developed in [177] and enhance its single objective cost model to a multiobjective cost model for materialized view selection on the Cloud. Our distributed DW integrates data from different relational databases and it is depicted as a relational OLAP (ROLAP) tool [158][76].

Chapter 6 presents the results of our experiments by using the proposed algorithms. The obtained results of the algorithms are presented and the design options are mapped to a real Cloud environment to evaluate the actual performance of our estimations and measure the efficiency of the schemes found by the proposed algorithms. We investigate the possibility of further improving the performance of DW by using selected materialized views. In the last Chapter, we present our concluding remarks and future work respectively.

CHAPTER 2

BACKGROUND

The virtualization of the resources and the scalability of the Cloud DWs are the most crucial factors of the Cloud DW design. Therefore, before dealing with multiobjective design issues of relational Cloud databases, it will be wise to explain existing virtual and dynamically changing (scaling up/out and shrinking back) environment of Cloud databases to analyze/comprehend the problem better and see the differences from the traditional database design problems. Although the architecture of the Cloud computing providers can be different from each other, we assume a shared-nothing Cloud infrastructure in our study. This Chapter first introduces the layered architecture of Cloud computing environment and Cloud DWs, explains virtualization (including the network virtualization), and later introduces the existing infrastructure of relational Cloud DWs, cost models, related query optimization studies, and view materialization.

2.1 Cloud Computing

Unites States National Institute of Standards and Technology (NIST) defined Cloud computing as "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" [106].

The emergence of this new computation paradigm has launched many fundamental modifications in information technology. It is now possible to spend computation as a utility such as electricity. The abstraction of the computation infrastructure environment as a Cloud terms the name of this new way of computing [104, 165]. With the opportunities provided by Cloud computation, today's developers no longer need to concern about wasting costly resources, or underprovisioning popular technologies, thus missing customers and revenue [13, 168]. Also, companies requiring large computation power can get solutions as soon as possible while their applications are scaling out with virtual machines in a dynamic infrastructure.

Amazon was the first provider of Cloud computing services with Amazon Web Service (AWS) on utility computing in 2006. However, Cloud computing concept dates back to 1950s [151]. Cloud computing uses and combines the ideas of previously developed computing paradigms such as client server model, autonomic computing, grid computing, utility computing, mainframe computing, and peer-to-peer computing. Cloud computing includes applications delivered over internet, the hardware, and the software running on these systems. Its main characteristics are:

Virtualization can be viewed as an autonomic computing. It is possible to build an organization's computing systems on utility computing that clients pay as they use. Virtualization also centralizes tasks while providing scalability/elasticity with hardware resources. With virtualization, it is possible to run multiple operating systems on a single server. Using virtualization, it is easier to manage installations and make rapid changes to the system and applications without disrupting the user.

Multitenancy provides sharing of resources across users therefore allows centralization of infrastructure with minimum costs, automated load balancing, adaptability, and utilization for idle waiting servers.

Scalability and Elasticity are on-demand dynamic provisioning of fine-grained resources (near real-time response times) under peak workloads.

Application programming interface (API) provides accessibility to software with Cloud software (REST-based).

Maintenance is easy and no installation is required on the client side for the applications and all the back-ups, system survival activities are being held by the provider.

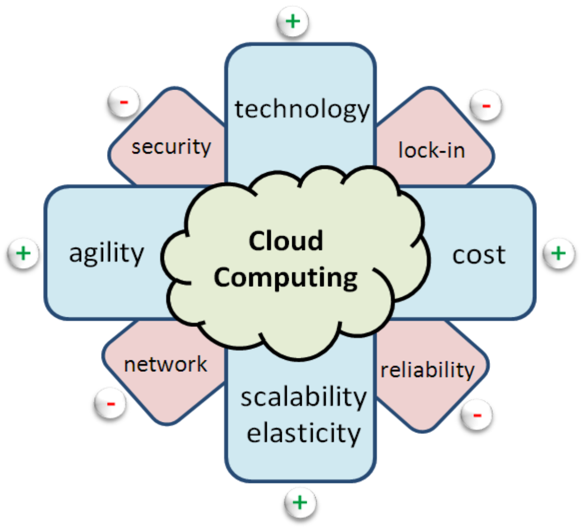


Figure 2.1: Advantages and disadvantages of Cloud computing.

Agility, Cost, and Performance Availability of new technological infrastructure, high performance and lower costs are the crucial properties of Cloud computing that make it more advantageous than other approaches.

With the advent of Cloud computing, many of the state-of-the art distributed computation approaches, techniques, and methods are combined together to provide high performance guarantees, lower costs of ownership, quality of service guarantees, zero-configuration, data privacy, scalability, and elasticity for the users even with unexpected workloads. In addition to the benefits of this new computation paradigm, some problems emerge in. These issues are shortly summarized in this part. Figure 2.1 illustrates the advantages and disadvantages of Cloud computing.

Advantages. Pay as you go model reduces the system ownership costs very much. The cost of the hardware/software maintenance is also diminished remarkably by the ubiquitous support of Cloud providers. The same computing system can be maintained and operated by fewer people. Following the technological advances in computer hardware and software becomes easier and less costly. Patches and updates are installed automatically. Any internet user can access to your platforms from anywhere in the world. Therefore, opening your ideas/products to world can be deployed rapidly with small costs. Scaling up/out computation environment on-demand is one of the most important opportunities of the Cloud. Elasticity, when the requests decrease for your software, shrinks the hardware usage of your system. This property of Cloud prevents the unnecessary deployment of virtual machines during leisure times.

Cloud gives the sense of working with infinite number of computers to the users.

Disadvantages The Cloud introduces new problems to its users. The most thought-provoking one is the security of the data in the environment. This is a very hot topic nowadays and many scientists are applying different methods to achieve a secure platform that will satisfy the users [27]. Data stored off-site have higher risks. With technology always improving, there are ways to make sure of better encryption. However, there are always people with good hacking skills and/or software tools. Lock-in is another drawback of the Cloud. The users are afraid of being dependent to a single Cloud provider, not being able to assign responsibility to anybody for the lost of the data, and not being able to recover it back. Cloud computing is dependent on the network (internet). It means that this system is sensitive to network outages at any time.

Large organizations such as government offices and financial institutions have their own services and will not move/store their data off-site. There are no widely accepted industry standards that can be applied to all systems when connecting to Cloud systems. The architecture of a Cloud computing platform can be categorized into three different service layers as it is shown in Figure 2.2 [179, 13]. These layers are Infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). Each service in this structure abstracts the layer below.

Infrastructure as a service (IaaS) The hardware layer is generally assumed to be a part of IaaS. It manages the physical resources. A data center has several hardware units. Configuration is handled by this layer. The virtualization layer generates resources for storage and computing by dividing the physical resources using virtual machine monitors such as Xen [172], VMware [175], and KVM [89]. Several crucial features are made available by this layer (such as resource assignment). IaaS layer provides these resources on-demand from their resource pools. Virtual private area networks are also one of their services (with IP support). In this model, Cloud customers use infrastructure to install operating-system and their applications. Cloud providers charge money depending on the amount of resources consumed. Windows Azure Virtual Machines, Amazon CloudFormation, Amazon EC2, DynDNS, Google Compute Engine, and HP Cloud are the main providers of IaaS.

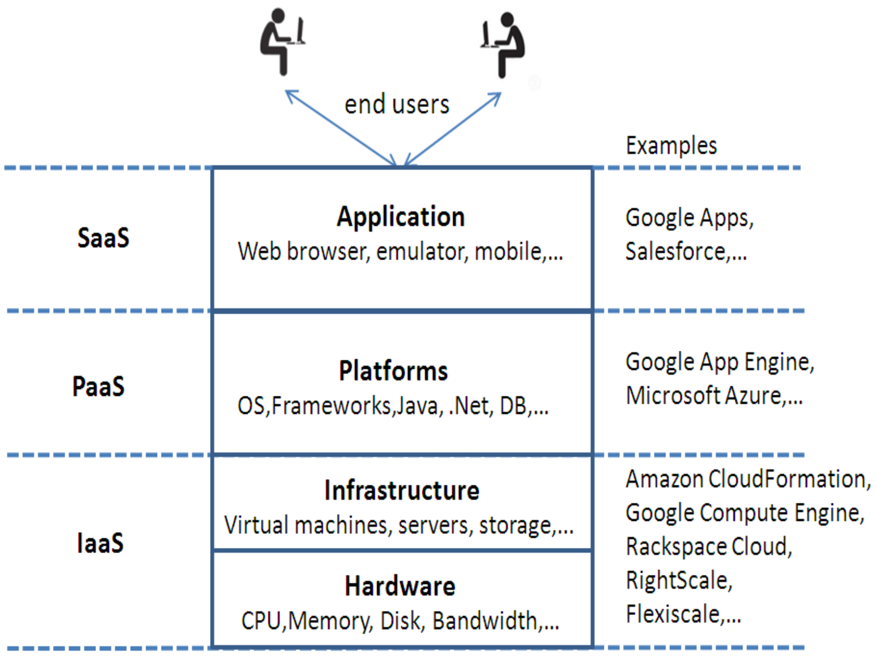

end users
Examples
SaaS
Application
Web browser, emulator, mobile,...
Google Apps, Salesforce,...
PaaS
Platforms
OS,Frameworks,Java, .Net, DB,...
Google App Engine, Microsoft Azure,...
IaaS
Infrastructure
Virtual machines, servers, storage,...
Hardware
CPU,Memory, Disk, Bandwidth,...
Amazon CloudFormation, Google Compute Engine, Rackspace Cloud, RightScale, Flexiscale,...

Figure 2.2: Cloud computing service models.

Platform as a service (PaaS) PaaS layer is the layer above the infrastructure layer. It manages operating systems, programming languages, databases, servers, and frameworks. The purpose of this layer is to reduced the workload of Virtual Machines. The data storages scale dynamically to meet the demand of the application.

Software as a Service (SaaS) locates on top of the PaaS layer. The application layer handles applications. Scaling leverages the performnce of Cloud applications and decrease operating cost. Cloud users use the software service of the Cloud. Users do not tackle with the other platforms. Cloud applications have more scalability [77]. Workload is evenly distributed to virtual machines.

The layered architecture of Cloud computing is modular whereas traditional environments are dedicated servers. Each layer is loosely coupled with the other layers. This is similar to the OSI model. The modularity make Cloud computing support requirements while reducing other overheads. Depending on the issues considered by the Cloud users (security, availability, price, etc.), these services can be deployed in different types of Cloud environments. Private Cloud infrastructure operates for a single organization. Public Cloud is a service where the resources are offered to the general public.

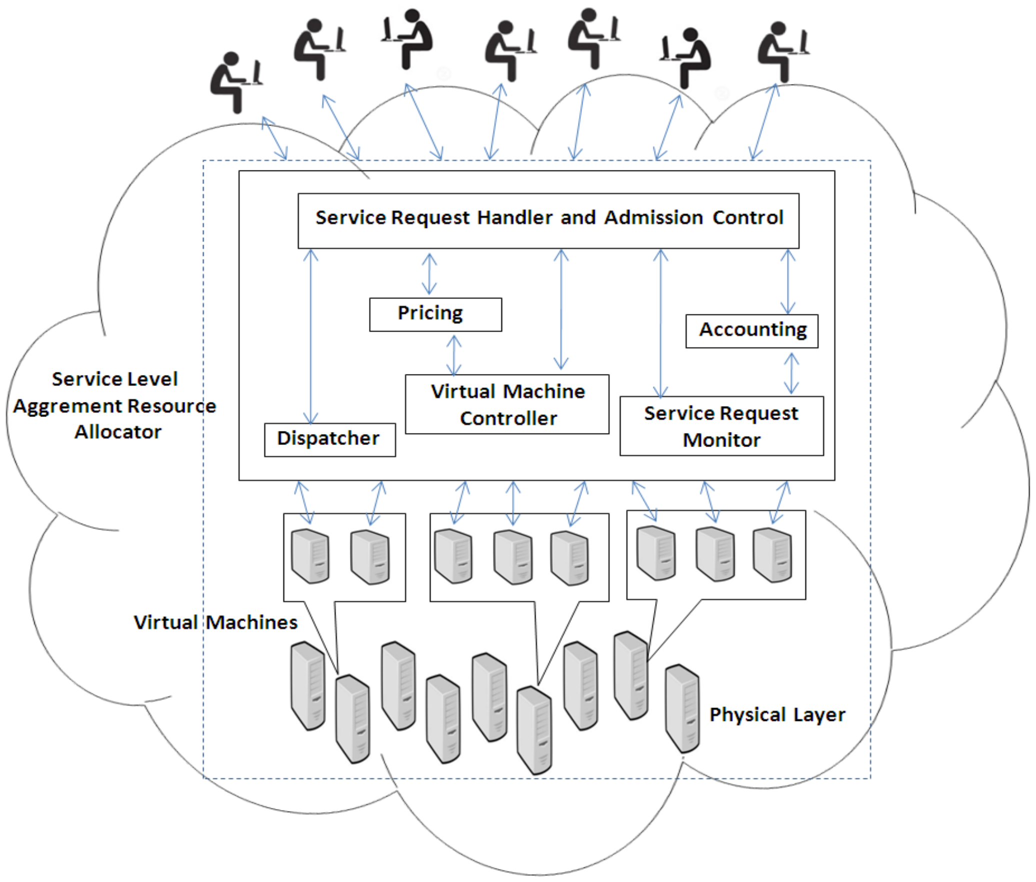

Service Request Handler and Admission Control
Pricing
Accounting
Service Level Aggrement Resource Allocator
Virtual Machine Controller
Service Request Monitor
Dispatcher
Virtual Machines
Physical Layer

Figure 2.3: High-level Cloud computing architecture.

Although there are some hot topics going on the security, privacy, compliance, and standards of the Cloud computing, we will not deal with these issues since they are not much related with the main issues of this thesis. A high-level Cloud computing architecture with its main components (physical layer, virtual machines, and service level agreement (SLA) resource allocator) can be seen in Figure 2.3. The most related opportunities of the Cloud computing for this study are the virtualization, scalability, and elasticity. Therefore, particularly examining these properties of Cloud computing more deeply will expand the vision to understand the advantages/disadvantages of Cloud and inequalities from the statically distributed computation environments for the queries.

Virtualization is the abstraction or decoupling of the application from the underlying physical resources [130, 27]. The physical resource can be collected together as logical/virtual resources are needed. This is called as provisioning. With Cloud computing, the provisioning of the logical resources becomes dynamic. Logical resource can scale up, out, and down in accordance with the demand. This process is called dynamic provisioning. In an efficient Cloud computing environment, every computing element must be capable of being dynamically provisioned and managed. Virtualization can mainly be applied to servers (partitioning a physical server into smaller virtual servers), operating systems, data storages, networks, and applications. Virtualization of the servers is transforming the conventional infrastructures from server-centric computing to network-centric Cloud computing architectures. It is now possible to create virtual servers. All of these opportunities transform the efficiency of the data stores. Resources (CPU, main memory, data storages, network capacities, etc.) can be increased and decreased as if working in an environment with infinite resources. Advances in network services management have been virtualized and it is possible to have the basic settings for Cloud computing.

The Cloud computing shares a set of distributed virtual computing resources optimally, whereas the server-centric computing paradigm dedicates resources to an application. The conventional model of servers is not designed to shrink up and down to response to changing workload demands [61]. The virtual networks provide an alternative to the physical network infrastructures. Resulting simplicity that is provided by virtualization significantly reduces the management costs and hypervisors are the tools that monitor this virtualized environment of Cloud [164].

Scalability can be described as an ability to adapt a system to a growing amount of work by increasing its resources to accommodate the growth. This can be typically achieved by providing additional resources to the existing system. Scaling can be done by vertically *scaling up* or by horizontally *scaling out*. Vertical scale up means to improve the overall capacity by increasing the resources in an existing node. Horizontal scale out is to increase capacity by introducing additional nodes. One of these scaling techniques or both can be employed at the same time [168].

Scaling up can be applied to stand-alone and server-based applications. It is limited by the growing capacity (e.g. available expansion slots) of available hardware. On the other hand, horizontally scaling out improves the overall system capacity by adding new nodes. The architectural structure of vertical scaling is different from the structure of horizontal scaling. The focus becomes the maximization of the overall

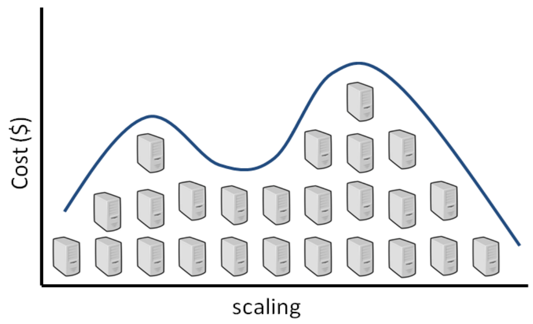


Figure 2.4: Costs vary in accordance with scales over time.

system by combining the power of many nodes, therefore; horizontal scaling is more complicated than vertical scaling. Scalability of a system becomes a necessity when resource bottlenecks occur. The contention of a system can be prevented by using fewer resources (which is not possible most of the time) or adding more resources. Therefore, Cloud computing environments must be horizontally scalable with the illusion of having infinite resources. There must be no system downtime during scaling and these settings must be done automatically.

Cloud resources are available on-demand as virtual machines and services. This model makes reversible scaling practical and important as a tool for resource utilization and cost estimation as shown in Figure 2.4. This property of Cloud environment is named as **elasticity**, i.e. reversible scaling.

2.2 Data Warehousing on the Cloud

Data warehousing emerged from the demand of using large data to achieve goals to improve the success of the organizations. A large company has many sub-organizations and managers need to evaluate how each on of these branches contributes to the profits of business performance. The integrated database of large companies stores huge data on the tasks performed by branches. This query can take very long. Finally, a report is given to managers. Database designers observed that such an approach is not feasible, because of the time and resources, and it does not always obtain the expected results. Moreover, analytical queries can slow down the system when combined with transactional queries. Contemporary DW tools process different (OLAP) not the (OLTP) [88].

In a DW, the whole data is brought together and organizations depend on the metadata obtained from the integration of these data sources [31]. The data can be queried by means of a query language or data mining tools to give information to decision makers. DWs can support on-line analytical processing (OLAP) tools that allow the navigation of the data in a warehouse. Data warehousing systems can compute statistics and predict trends within the data. Although DWs are highly demanding systems, long periods of setting up the data, high monetary costs, and system expertise needs are the limitations that restrict the usage of these systems. Recently, companies are trying to make use of Cloud computing with software as service (SaaS) to customers. The customers access the application via the internet. Google Apps, SkyInsight [142], GoodData [66] are some of the examples of these systems. By using these systems, there is no need for expertises and over-provisioning can be avoided.

Data marts are subsets of a DW that is usually oriented to a specific business. Data marts are small parts of DWs. The information in data marts is related to a single department, whereas a DW serves to whole company. Each department is considered the owner of its data mart including all the hardware, software and data. Each department uses, manipulates and develops its data without altering information inside other data marts or the DW. Business intelligence (BI) applications are the first use of data marts. Data marts can be used by smaller businesses to make use of the data they have. A data mart can be a cheaper solution than using a DW. A data mart can also be installed faster than a DW. With low costs, data marts can be a suitable method for storing data in small scale companies.

DW systems require different talents than conventional database systems. Conventional database systems are mostly used for transaction intensive operations, DWs are used for analytics (select-project-join operations) on large data. Although there are many different solutions to data warehousing, main components are the same for all of the systems. The information comes from typical OLTP systems. The DW component keeps all of the incoming data and the data can be cleaned before being processed to prevent the anomalies. After loading the data into the DW, it can be used by OLAP and data mining tools. The performance increase of OLAP and data mining tools are not the main goal of our study; however, they are heavily being used by these systems. We more focus on the high-performance issues of the SQL queries.

In OLAP, data is presented in data hypercube forms and each one of the dimensions represents a data dimension.

There are two main approaches for storing data in a warehouse. The Multidimensional OLAP (MOLAP) stores data into multidimensional data cubes. Its advantage is that it can be used by OLAP tools without much need for transformations. But it is hard to integrate them to original data sources and SQL statements. And also it does not scale well. The other approach is Relational OLAP (ROLAP). With this approach, the DW can be used with conventional SQL statements. ROLAP is also more scalable. Our main focus is the speed-up of the queries of ROLAP systems.

Cloud environments are new emerging efficient platforms for DWs. However; there are some challenges when deploying data to a Cloud DW. Shipping the data to Cloud can take long periods. Also taking back your data from Cloud provider can be another important issue. The performance expectations may not be met by the Cloud providers because of the low-performance planning of the end Virtual Machines (CPU, memory, and disk bandwidth). WAN latency and security of the data are other issues that concern the users of Cloud DWs. Cloud computing can handle the usage of a DW in an efficient financial way. For example, a DW might not be used at night and another DW may be in heavy use during daytime. An elastic Cloud DW can adjust the number of virtual machines to minimize the budget.

In DWs where terabytes of data are being processed query optimization becomes an important issue. Materialized views are important components of such systems [31]. A materialized view is a previously completed result of a generic query that may be used by other queries. Determining the selection of the mostly used materialized views is an important research area. Parallelism is another important issue that can be used to speed up the queries of DWs. Using multi-core processors and partitioning are some of the techniques that we have explained. Shared-nothing architectures are promising solutions methods for scalable DWs because virtual machines can be used in a fully parallel manner to provide linear speed-ups. Workload management of the queries is another aspect of the DWs that must be monitored carefully. Elasticity of the Cloud can provide a good infrastructure for these issues with its virtual resource opportunities.

The idea of using the relational databases for warehouses is a very efficient way. The relational model does not have measure and dimension. Therefore, specific types of schema must be created to represent the multidimensional models. Star schema is one of these approaches. Large joins are the main problems of ROLAP systems. Redundancy (materialized views) becomes a key concept to improve the performance. ROLAP requires a specialized middle layer between back-end servers and front-end components.

Multidimensional engine is the main component of DWs and it can be connected to any relational server. The performance of MOLAP system is excellent. There is no standard model set for MOLAP. A standard is missing but it is being solved. Microsoft (Analysis Services) and Oracle (Hyperion) are successful projects in this field. The architecture, hybrid OLAP (HOLAP) aims to mix the advantages of ROLAP ad MOLAP.

If we look at some of the leading commercial DW tools. IBM DB2 is one of the important tools that use shared nothing architectures [83], which means that the different databases do not share any information, making it more scalable and modular. It has a cost-based optimizer. Materialized views is another crucial way to speed up DW workloads. DB2 uses 'shared scan' to share results between data scans for the same data. Oracle also offers DW products. Their most prominent product is 'Database 11g' [117]. It has OLAP capabilities and uses data cubes. Materialized views are provided. Oracle also offers clustering capabilities, which is very useful for elasticity. The information about the internal structures of these commercial systems are not explained in detail because they are considered as commercial secrets. However, we have not come across any information that both designs/optimizes the monetary cost of the virtual resources and the response time of the query workloads at the same time as it is proposed in our study.

Microsoft Parallel DW (PDW) is a parallel processing DW tool built for large volumes of relational data (with up to hundred times performance gains) and integrates to Hadoop [107, 74]. PDW reduces ongoing costs resulting in a solution that has the lowest price/terabyte in the market.

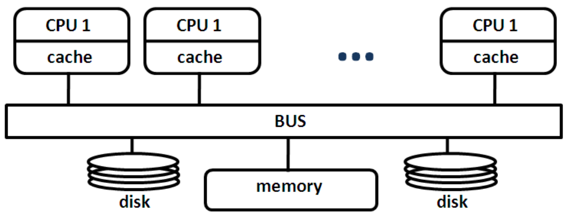


Figure 2.5: Shared-memory and shared-disk architecture.

2.3 Hardware Architectures for Relational Cloud DWs

Relational Cloud DWs run on Cloud computing platforms. Users can run DWs on the Cloud independently by themselves but automated management of the virtual resources is more efficient and wanted way of managing of the Cloud DWs. This study focuses on the performance and monetary cost of relational Cloud DWs which introduces many interesting problems to provision a successful Cloud DW management with changing requests.

Relational Cloud DWs architectures are designed mainly on the principles of distributed and parallel databases [58, 39, 118, 156, 93]. In this Section, the underlying hardware architectures will be summarized.

It is now possible to have a multiple processor computer in your home which was only a dream a decade ago. Symmetric Multiple Processor (SMP) machines, clusters of workstations, massively parallel processors (MPP), and clusters of SMP machines are easily accessible and utilized hardware by the Cloud providers. The underlying hardware structure of Cloud databases are constituted by making use of these architectures. Commonly, these distributed/parallel hardware architecture are classified according to three categories.

(1) Shared-memory and shared-disk architecture.
(2) Shared-nothing architecture.
(3) Hybrid architecture.

Shared-memory and shared-disk architecture shares a common main memory and a secondary memory. Figure 2.5 gives an example of this architecture. In this architecture, tasks are divided into sub-tasks and assigned to slave processors. Since the

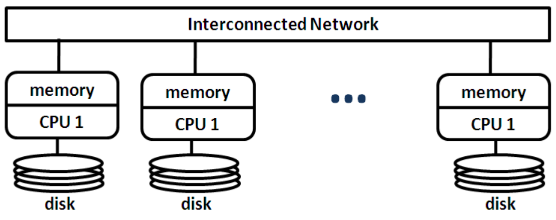


Figure 2.6: Shared-nothing architecture.

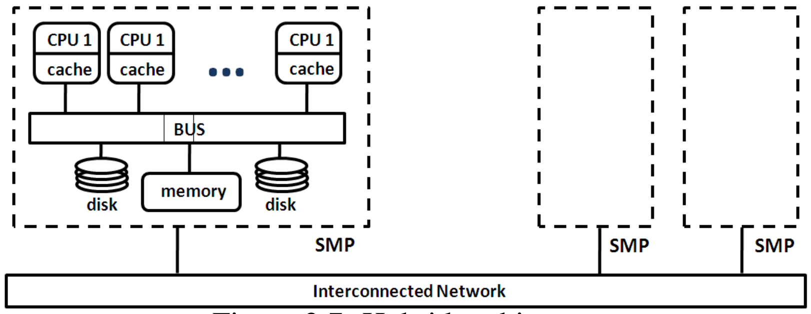


Figure 2.7: Hybrid architecture.

memory is common to all processors, most of the coordination among the processors is done easily. This architecture uses a bus interconnection network and when many processors may compete for access to the shared data it suffers from memory and bus contention. Therefore, a shared-memory computer is generally equipped with no more than 64 processors. In shared-disk architecture, the secondary memory is shared by the processors as in the shared-memory architecture. This architecture is mostly used in SMP computers. The operating system allocates tasks processors.

Shared-nothing architecture In this architecture, each processor has its own local main memory and disk and there is no contention due to the accesses to the shared data. However; load balancing is difficult to achieve. Each processor uses its own memory and disk during the processes. Therefore scaling out is known to be easier with this architecture which is one of the main concern of Cloud computing environments. The data/computation skewness is another major problem for shared-nothing architectures while processing data. Workstation farms and Massively Parallel Processing (MPP) machines are examples of shared-nothing architectures. Since the processing units do not share common data storage, they communicate among each other via a network. Figure 2.6 shows a shared-nothing architecture.

Hybrid architecture. This architecture settles the limitations of shared-memory and shared-nothing architectures. There can be different versions. The basic model can be seen in Figure 2.7 that each node (a cluster of SMPs) has a shared-memory connected to an interconnection network to form a shared-nothing architecture. Lower network communication demand and flexibility are the most important features of hybrid architecture. With this architecture, it is possible to plug in new elements to the current system and take out the old ones. It is easier to add SMP machines into an existing architecture.

Interconnection Networks In all of the aforementioned architectures, processors need to communicate via a network. The main types of interconnection networks are bus, mesh, and hypercube. Bus is a single communication line connecting processors usually in SMP architectures. CPUs components can send and receive messages from a single line. Bus architecture does not scale well with increasing number of processors. In Mesh architecture, each processor is connected directly to its adjacent components in the Grid and can communicate with others by routing messages via nodes that are connected to another. In Hypercube architecture processors are numbered in binary and connected to another if the representations of the numbers differ in exactly one. Each component is connected to log(n) other components, where *n* is the number processors in the system. This model is also scalable but the number of links increases excessively as new processors are added.

2.4 Relational Cloud DW Design Techniques

Relational Cloud DWs rely on the basic hardware architectures of conventional distributed databases and their design techniques. The classic hardware architecture used for many distributed databases is shown in Figure 2.8. Requests are processed by a load balancer by sending them to available application servers. The SQL statements are sent to the database servers to process the requests. Later, the storage system ships physical blocks of data to and from database server. SAN or single disks are being used by the traditional storage layers. Solid-state disks, large main memories (in-memory databases), or a hybrid design of both will be the media of the data storages in the near future [159] [40][112].

The classic architecture allows using the best designed components at all layers and

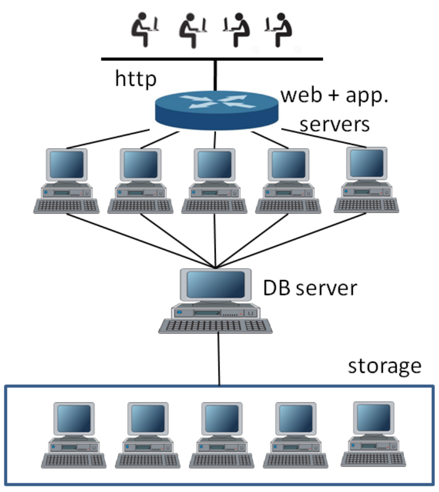


Figure 2.8: Conventional database architecture.

provides scalability and elasticity at the application and storage layers. If the workload of application servers increases due to the excessive number of clients, additional number of the application servers can be provided easily. It is also possible to decrease the number of the machines when the workload decreases. The same technique can be applied to the storage layer as well. The database server layer has some limitations in this architecture that it is not much possible to scale out its capacity easily. Therefore, Cloud service providers have designed scalable computation environments with shared-nothing architectures to answer the changing requests of the Cloud users under peak workloads. Partitioning, replication, caching, and distributed control are the main techniques that are being used by the Cloud providers to design efficient relational Cloud DWs.

Partitioning : This is the most widely used method of distributed database design techniques and is being used efficiently by Cloud DWs. Partitioning is generally used to divide a big data into fragments when it is not possible to process it in reasonable times with a single processor or there is not enough storage to maintain it. The relations are separated into fragments, by applying horizontal, vertical, round-robin, hashing, range partitioning or a hybrid method. Each fragment is handled by separate database server in this design [161, 78, 28]. With partitioning, the database layer is put on top of the storages and therefore a more scalable environment is provided. Transparency of the partitioning is a desired property for this design. On the other hand, partitioning has limitations that while scaling out, huge amount of data needs to be shipped across the machines. Figure 2.9 illustrates an architecture where database

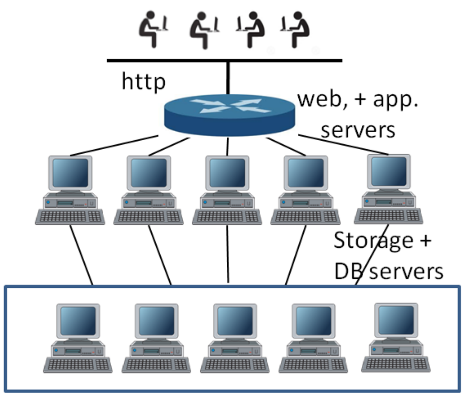


Figure 2.9: Database architecture where database servers and storages are attached together to provide more scalability.

servers and storages are attached together to provide more scalability. **Sharding** has emerged as a new way of partitioning large databases for a few years. It is commonly available for shared-nothing environments. Sharding provides a method for scalability across independent servers, each with their own CPU, memory and disk. The basic concept of sharding is very straightforward. Break a large database into a number of smaller databases across servers.

Replication : In this model, each database server controls a copy of a relation, a fragment, or whole database. Consistency of the replicas is important during the design. ROWA (read-once, write all) is the most prominent protocol that uses a master copy to keep the system consistent [120]. Application servers issue queries of read-only transactions to any database server. If the workload has read-only operations mostly, the architecture in Figure 2.9 scales-out and shrinks back nicely with replications. For workloads with intensive updates, master copy becomes a bottleneck that prevents the scalability. There are many successful studies to remove this drawback and provide efficient and reliable Cloud databases with high consistency [159].

Caching : has been an efficient tool to increase the performance of the database servers. It can be combined with other design techniques such as partitioning, replication, and distributed control. It can be maintained by the dedicated servers as well as database servers. Servers can keep intermediate results of queries/views in their main memory. There are many different schemes in order to keep the cache consistent with regard to updates to the database. Google AppEngine operates a farm of dedicated MemCache servers to provide this service.

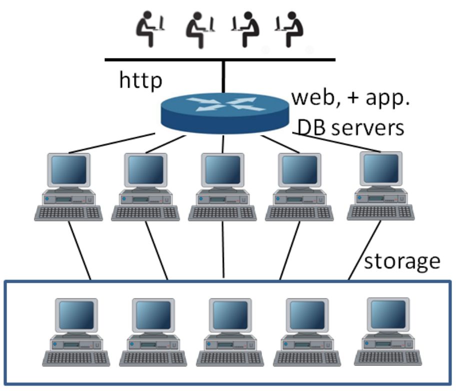


Figure 2.10: Distributed control architecture.

Distributed Storage Control : This approach uses a similar architecture of partitioning and replication and it has major impact on the implementation and performance of a system (Figure 2.10). The Distributed Control architecture can also act like a shared-disk model. It uses loosely coupled nodes to provide scalability. The storage system and the database servers are separated from each other. The database servers access the shared data from the storage system and different distributed protocols can be applied. The distributed control architecture is probably the best model for Cloud computing. Scalability and elasticity can be provided at each layer by directing each request to any application/database server. Replication and partitioning can be achieved in a scalable manner. Commodity computers can be used at each layer of this architecture. But as stated in CAP theorem [23, 1], it is not possible to provide consistency, availability, and resilience to network partitioning at the same time.

Consistency in Cloud Databases (ACID vs. BASE). Cloud DWs must make trade-offs depending on the expectations of the users. Consistency and the availability are the most two important ones due to the CAP Theory [23]. Therefore, new and efficient consistency issues are being seriously developed by many researchers. Strong consistency in database systems is defined by means of ACID properties of transactions [62]. ACID requires that for every transaction atomicity, consistency, isolation, and durability attributes hold. If ACID is chosen for consistency, it emphasizes consistency and diminishes the importance of availability and implies a pessimistic view where inconsistencies should be avoided at any cost. Complex protocols such as 2-phase-commit or consensus protocols like Paxos are required to achieve ACID properties. On the other extreme, where availability is more important, BASE [23] is proposed as the counter-part for ACID. BASE stands for: **B**asically **A**vailable, **S**oft state, **E**ventual consistent. BASE is optimistic and accepts temporary inconsistency. Eventual consistency only guarantees that updates will eventually become visible to all clients and that the changes will persist if the system comes to a quiet state [128]. Eventual consistency is easier to achieve and makes the system highly available. Between the two extremes, BASE and ACID, there is a range of consistency models from the database community [167] as well as from the distributed computing community [12].

2.5 Query Optimization in Relational Cloud Data Warehouses

Relational Cloud DWs make use of the query optimization techniques of distributed/-parallel databases [93, 92, 156]. Therefore, this chapter surveys the state-of-the-art key concepts of distributed query optimization (DQO) techniques that are applicable to relational Cloud databases.

Query processing rewrites a query sentence as a physical query plan that has a set of tasks. Query optimization chooses the best way of executing a given query. The main stages of query processing can be seen in Figure 2.11. DQO determines which resources will be used to execute a query and its order. The optimizer generates alternative plans, estimates their costs and chooses the best one [84]. Finding the best way to solve this problem is known to be an NP-Hard problem [82]. Distributed database query optimization is much harder than the centralized database version of the same problem and introduces issues of the cost of communication and an expanded search space due to the data shipping and site selection for intermediate operations during the optimization of a query.

The problem of determining a good query execution plan (QEP) for join expressions has been addressed since 1979, the development of System R [133, 79, 70, 32, 37, 68]. The work has two steps. The development of efficient algorithms and second, the algorithms that determine the order of joins. The search space is the set of all possible QEPs that can be constructed in many different ways. A solution is defined as a processing tree. The QEP tree is a simple binary tree. Its leaves correspond to base relations and inner nodes correspond to join operations. Edges indicate the flow of

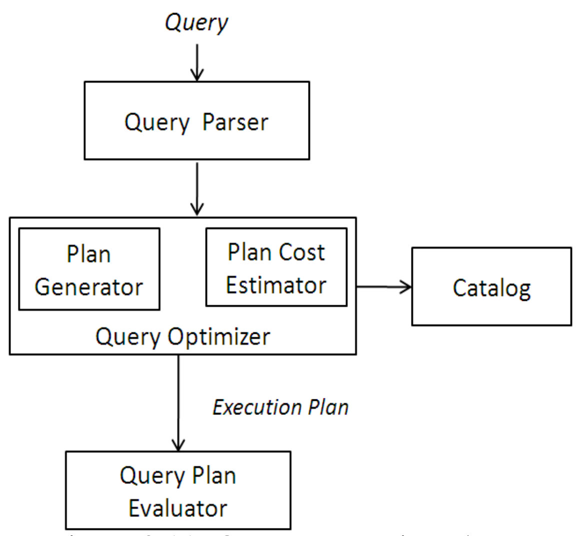


Figure 2.11: Query processing phases.

partial results from the leaves to the root of the tree. Each evaluation plan (point of the solution space) has a cost. The aim of the optimization is to find a QEP with the lowest possible cost. The solution space is defined as the set of all processing trees that compute the result of the join expression. The leaves of the processing trees handle the relations, inner nodes handle joins [116]. Many algorithms for DDB query optimization have been proposed and studied in the literature. SDD-1 was the first developed distributed query processing algorithm and it was based on Hill-Climbing strategy [147]. Dynamic Programming (DP) was also extensively studied for solving this problem by storing the sub-solutions in the main memory [94]. Distributed INGRES algorithm is derived from the centralized INGRES algorithm [149]. The Iterative Improvement algorithm makes small changes on an existing solution until no more improvement can be achieved [84, 153]. Simulated Annealing [85], Two-Phase optimization [84], and Random Sampling [59] are the members of randomized algorithms which have also been used for solving the problem. There are also efforts to design new algorithms with hybrid approaches that make use of deterministic and randomized algorithms. Mostly, deterministic algorithms provide a good starting point and the following decisions are made either by randomization or Genetic Algorithms [64]. The Iterative-DP (IDP) is a member of this class of algorithms. IDP starts exploring the search space by DP, in the later steps it prunes the search space according to some heuristics [94].

There are various algorithms developed to find a good join order for a DDB query.

Most of these algorithms are derived from centralized database query optimizers. Algorithms developed for this problem can be categorized as deterministic (exhaustive), metaheuristic [25] randomized, genetic, and hybrid algorithms according to Steinbrunn [147].

Deterministic Algorithms. Members of the deterministic algorithms construct solutions either in a top-down or a bottom-up manner and use a heuristic or an exhaustive-based strategy. The amount of the search space for large DDB join queries is so big that exhaustive deterministic algorithms are not always feasible. Dynamic Programming is a well-known deterministic algorithm which is widely applied in the field of database query optimization [147].

Some of the well-known deterministic algorithms are; Minimum Selectivity [171], Krishnamurty-Boral-Zaniola (KBZ) algorithm [82][96], AB algorithm [154], and R* algorithm [133][118].

Randomized Algorithms. *SDD-1 Algorithm* is the first developed distributed query processing algorithm and based on a hill-climbing algorithm [21]. The initial feasible solution is repeatedly improved until there is no possible other solution. The algorithm does not use semijoins, replications or fragmentation and it is good for point to point networks [118]. Distributed INGRES is derived from the centralized INGRES algorithm [54] which recursively breaks up a relational calculus expression into smaller pieces. Iterative Improvement [153, 84, 85], Two-phase optimization [84], Toured simulated annealing [97], and Random Sampling [59] are some examples.

Genetic Algorithms (GAs) try to simulate the evolution process [64]. GAs also propagate solutions for a given problem between generations. A single solution is called an individual, and the set of solutions is called population. Solutions are chromosomes of genes. Encoding is the representation of a chromosome and selection is used to separate the good and bad solutions of the population. Crossover is a way of combining good partial solutions provided by two parents in order to obtain a better result. Mutation introduces a new feature to an offspring that is not present in either of its parents. A population of randomly selected chromosomes is created, and a fraction of the fittest members of this population is selected and using crossover and mutation operators a new generation is evolved. The termination condition of the algorithm can be the achievement of a desired quality level, a limit on the number of the generations, or achievement of no improvement for a certain number of generations [136, 17, 50, 71].

Hybrid algorithms make use of the strategies of deterministic and randomized algorithms. Mostly, deterministic algorithms provide a good starting point then the following decisions are made either by randomized algorithms or GAs. The Iterative-DP (IDP) algorithm is a member of this class. IDP starts exploring the search space by DP, in later steps it prunes the search space according to heuristics [94].

Dynamic Programming (DP) works in a bottom-up manner constructing all the sub-plans until an optimal solution is obtained. Any given sub-problem is constructed only once and saved in a table, therefore it is not computed again. The algorithm first builds access plans for every relation in the query and there may be several different access plans. If relation R_1 is replicated in different sites, say site 1 (S_1) and S_2, the algorithm enumerates *table-scan* (R_1,S_1) and *table-scan* (R_1,S_2) as alternative plans. DP enumerates all two-way join plans in the second phase. Alternative join plans are enumerated, including the data shipping cost of joins for all sites. All multi-way join plans are built using individual relation access-plans and previously discovered optimal multi-way join plans as building blocks. DP continues until it has enumerated all the n-way join plans. The *joinPlans* function produces more and more complex plans using the smaller plans as building blocks and returns a new plan for every alternative join method. DP discards all of the *inferior* partial-plans (i.e. a better plan exists for calculating the same output as that of the partial-plan). This is called pruning and carried out by the *prunePlan* function. An efficient plan can prune another plan with the same output. DP enumerates ($R_1 \bowtie R_2$) and ($R_2 \bowtie R_1$) as two alternative plans, but only the better plan is kept in the *optPlan*(R_1,R_2) structure after pruning which reduces the search space. In a DDBMS, *table-scan* (R_1,S_1) and *table-scan*(R_1,S_2) do the same job, but they produce results at different sites. So, they cannot be pruned in order to ensure that the optimizer is guaranteed to find the best plan. For example, if the cost of *table-scan*(R_1,S_1) and the cost of shipping R_1 from S_1 to S_2 is lower than the cost of *table-scan*(R_1,S_2), then *table-scan*(R_1,S_2) is pruned. DP algorithm is given in Algorithm 2.1 [92].

The running time complexity of DP for a centralized database is $O(3^n)$ and the space complexity is $O(2^n)$ where n represents the number of relations [116]. The distributed query optimization time complexity of DP is $O(s^3 3^n)$ and its space complexity is $O(s2^n+s^3)$ where s is the number of sites at which a copy of at least one of the tables involved in the query is stored and the site issuing the query to which the results are returned [94].

Algorithm 2.1: Dynamic Programming Algorithm

```
Input: Select Project Join query $q$ on relations $R_1$, ... , $R_n$
Output: a query plan for $q$

for $i \leftarrow 1$ to $n$ do
    optPlan ($\{R_i\}$)= accessPlans ($R_i$)
    prunePlans(optPlan ($\{R_i\}$))

for $i \leftarrow 2$ to $n$ do
    for all $S \subset \{R_1, ..., R_n\}$ such that $| S |$= i do
        optPlan ($S$)= $\emptyset$
        for $O \subset S$ do
            optPlan ($S$)= optPlan ($S$) $\cup$ joinPlans(optPlan($O$), optPlans($S \setminus O$))
            prunePlans (optPlan ($S$))

finalizePlans(optPlan($\{R_1, ..., R_n\}$))
prunePlans(optPlan($\{R_1, ..., R_n\}$))
return optPlan($\{R_1, ..., R_n\}$)
```

Recent Relational Cloud DW Query Optimizers. There are studies to adapt traditional query optimizers to Cloud computing. In [20], a classical query optimizer is adapted to Cloud computing workloads where it uses a partitioned database on a shared-nothing architecture. In [137], a parallel DW system optimizer is developed for single queries by considering a rich space of execution alternatives with bushy-tree plans instead of simply parallelizing the best serial plans. Query optimizations in Cloud environments can have different goals unlike the traditional query optimizers, and the search space is much larger because of the number of data processing nodes provided by the Cloud environment. In an interesting study, the scheduling of data processing workflows on the Cloud is considered from the perspective of minimizing the completion time given a fixed budget [90]. Kossmann et al. developed a new database architecture that is based on batching queries and shared computation across many concurrent queries in a shared multi-processor environment [63]. A framework is developed for a Cascade-style Cloud query optimizer to enhance the performance by using MQO techniques for massive data analysis scripts that contain common subexpressions [140]. In [65] a multi-colony ant algorithm is developed to optimize the join queries of a distributed database where relations are replicated but not fragmented. Four types of ants collaborate with each other to provide an efficient execution plan and each colony makes a decision to find the optimal plan.

In order to estimate a query, it is important to provide an accurate cost model. Cost models describe the process involved and compare the solution quality of a QEP with others. There are two main cost models that are used to evaluate the efficiency of QEPs [97, 65, 118, 156]. Total execution time cost models and response time cost models. A cost model equation consists of data parameters, systems, query parameters [156].

The number of processors (sites), page size, the main memory size, and network bandwidth are the most important system parameters that a cost model must use. Projectivity ratio (π), Selectivity ratio (σ), and the semi-join are the important query parameters. Projectivity ratio, π, is the ratio between the selected attribute size and the original record size. Selectivity ratio, σ, is a ratio between the total output records and the original total number of records. Projectivity and selectivity parameters are associated with the number of records before and after the query processing, which gives the processing time of the executions in the main memory. Communication cost is calculated depending on the number of pages to be sent and message preparation cost. The parameters are summarized in Table 2.1.

Statistics Used for the Optimization of Query Expressions (Histograms). Given a QEP, it is crucial to evaluate the cost of the execution accurately. Query optimizers rely on these estimations to provide efficient QEPs. In order to make a good estimation, it is vital to know the resources consumed by a QEP. The resources of a QEP can be CPU usage, I/O cost, main memory limitations, network bandwidth, or com-

Table 2.1: Parameters used in the cost model.

Parameter Type	Name of the parameter
data	size of the relation (fragment) (Byte)
data	number of records in relation (Fragment)
system	number of processors
system	page size
system	main memory size
query	projectivity ratio
query	selectivity ratio
time unit cost	time to read a page from disk
time unit cost	time to write a record to the main memory
time unit cost	time to read a record in the main memory
communication	message protocol cost per page
communication	message latency for one page

bination of these. The estimation depends on the statistical summaries (cardinality estimates) of the data. For every relation, the cost of scans, joins, and their memory requirements are kept as important statistical information [26]. The number of physical pages used by the relation and the statistical information on columns are the parameters used to estimate the selectivity of predicates. Cardinality estimation uses this statistical information to provide accurate results.

For example, consider two different QEPs for the same query given in Figure 2.12. If the query optimizer has the information that predicate (A.a < 15) produces less tuples than predicate (C.a > 100) then QEP$_1$ is chosen by the query optimizer.

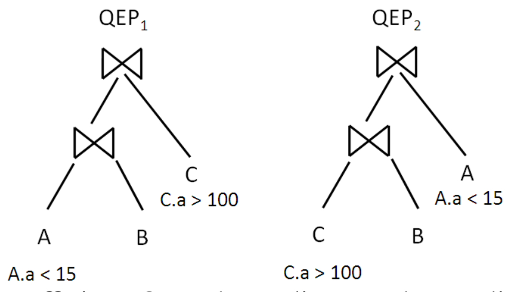


Figure 2.12: Choosing efficient QEP depending on the cardinality estimation of intermediate results.

In this part, we show how histograms are being used to estimate the cardinalities of

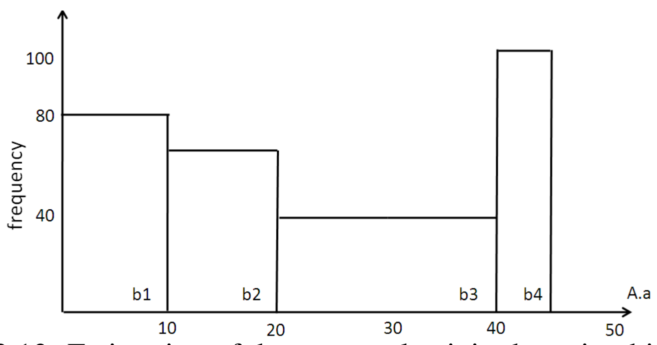


Figure 2.13: Estimation of the range selectivity by using histograms.

the tasks executed by QEPs. Histograms are the most common statistical tools used by DBMSs [91, 24]. A histogram consists of a set of buckets that hold the information on an attribute. Each bucket represents a sub-range of the values of the column. There are two associated values in a bucket, the frequency of bucket (the number of tuples in the data set) and the number of distinct values of all the tuples. The distribution of tuples in each histogram bucket is assumed to be uniform. Other techniques such as continuous and randomized models can also be used.

Figure 2.13 gives a histogram on relation A (attribute a). Buckets b1, b2, b3, and b4 represent 80, 70, 40, and 100 tuples respectively. For predicate (A.a > 15), b4, b3, and 50 % of the tuples in b3 are added and the total sum of tuples is found as 175.

For example, in case of multiple predicates such as in the query below :

```
SELECT *
FROM A
WHERE (A.a > 15) AND (A.b < 50)
```

The selectivity for the whole predicate is estimated as (A.a).(A.b) where A.a is the selectivity of (A.a > 15) and A.b is the selectivity of (A.b < 50).

Estimation of cardinality for the join predicates is composed of three steps. In the first step, the buckets are aligned to agree their boundaries. In the second step, each pair of aligned buckets is analyzed and estimation per bucket of join sizes is done. At the last step, the partial frequency aggregation from each resulting bucket is done to get the cardinality estimation for the join. When arbitrary select-project-join (SPJ) operations are considered, the cardinality estimation requires propagation of the statistics.

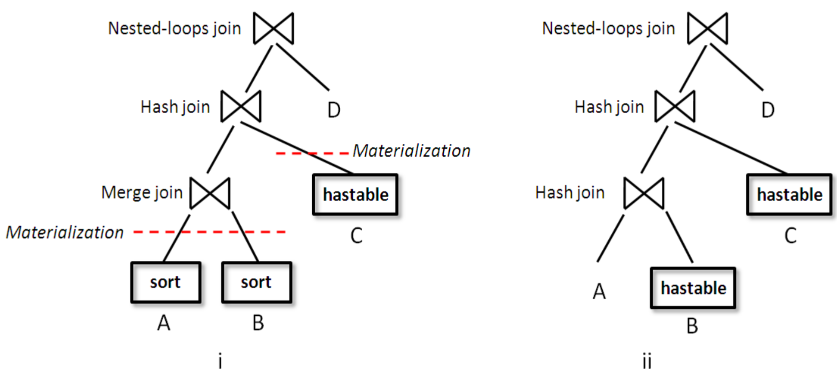


Figure 2.14: (i) a non-pipelined plan; (ii) a pipelined plan

Pipelining and Materializing in Query Execution Plans. The result of an operator is sometimes passed to another operator without saving the intermediate result. This is called pipelining and it saves the cost writing/reading the results back in. If the output is saved in main memory or on a disk it is called materialization. Pipelining has lower overhead than materialization. However it is not always possible to pipeline the results [62]. During the execution of the QEPs, blocking operators can be faced. Blocking operator need to keep sub-solutions. Sorting operator or hash join operator that the build relation is stored in a hash table and probed many times are the blocking operators while processing a QEP [42]. Whether the QEP is executed with pipelining or materialization constraints the processing schedule of the QEP [45]. Non-pipelined QEPs have at least a blocking operator during their execution. The materialized results are kept in the memory or sent to another processor to complete the QEP. Figure 2.14 shows non-pipelined and pipelined plans. Pipelined plans execute in parallel, by sending the resulting tuples of an operator to the next operator. Figure 2.14 (ii) shows a pipelined plan using hash join operators.

Query Processing Trees. A QEP is a tree with each node representing a sub-query and an edge between two nodes specifying the execution order (network) of the sub-queries. Each node depends on the incoming edges and cannot be executed until its children node(s) finish. An execution plan is performed by several phases. The first phase involves only base tables. The next process comes after the completion of the previous phase. They have dependencies. The last phase produces the result. In a multi-processor environment, each operation is allocated to one or more processors in parallel and expected to finish in harmony. Left/right-deep tree, bushy-tree, and zig-zag tree are the main execution plan types used by the optimizers. They are given in Figure 2.15. The purpose of parallelization is to reduce the height of the execution tree. When each evaluation is independent, bushy-tree is the best performing option. However, when each task depends to another, then bushy-tree is not possible. Left and right-deep trees are similar to sequential processing. The output of a task becomes an input to the next tasks. These execution trees are good when implemented together with pipelining. The main objective of a QEP is to complete all sub-tasks as early as possible.

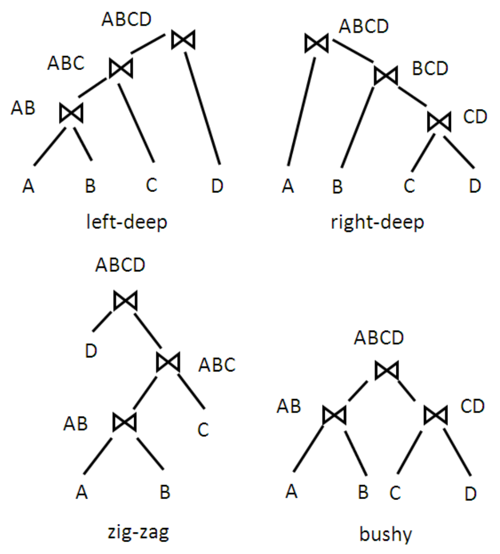


Figure 2.15: Query execution plans.

Access Path Selection. After parsing the SQL statement, optimizer use the names of the relations and columns in the query and verifies statistics about them. The database catalog holds the statistics. The optimizer selects the best access paths from this catalog to obtain the most efficient QEP. The access paths that reduce the total cost for the query blocks are selected from a tree of different choices. The primary way of accessing data is through data storage. Scan is the basic operation to reach data. There are currently two types of scans, namely, segment scan and index scan. The first type of scan searches all the tuples in a relation whereas an index cares only with the related tuples. Indexes are maintained on separate pages from those that contain the tuples of relations. Index scan does not scan all the tuples in a relation. Both segment and index scans can take a set of predicates. The optimizer examines the predicates and the access paths on the relations and uses a formula to predict the cost of the query. During the optimization of a query, WHERE tree of predicates is examined. If it is possible to use an index on a column then it is exploited to obtain a better QEP.

The optimizer gets the statistics in the database catalog. The most important statistics are the cardinality of the relations, the number of pages, number of distinct keys in an index, and the number of pages in an index [133]. A selectivity factor is assigned for each predicate in the query list. A detailed explanation of selectivity factors can be found in [133]. For single queries, best access path selection is obtained by calculating the cost of each path. The ordering of the tuples is an important criterion if there are GROUP BY and ORDER BY clauses. Then, the cost of sorting the tuples must be considered to find the optimal solution. The join is another frequent operator that must be carefully estimated to find the optimal QEP [108]. Joins can be implemented in many ways. Certain techniques can be better than the others depending on the design of the relations (indexes, sorted columns, etc.). In a join operation, there are two relations that are called outer and inner relations. Outer one is the relation that a tuple will be retrieved first and the inner one is the relation from which the tuples will be retrieved. Join column is the matching criterion for both relations.

Nested-Loops, **Sort-Merge**, and **Hash-Join** are some of the main techniques applied during the join operation. In a nested loop join, after fetching the first tuple of the outer relation, a scan is performed on the inner relation to retrieve the tuples that satisfy the join predicate. If the outer relation has m tuples and inner relation has n tuples. Then the performance of this join is calculate as $O(m \text{ x } n)$. In sort-merge join, both relations (or only one of them) are sorted on the join predicate. Later, both relations are matched and resulting tuples that satisfy the join condition form a relation. The performance of the algorithm is $O(m+ n\log n)$. The sort-merge join reduces the number of comparisons between the tuples of the relations. A tuple of the outer relation is not compared with the tuples of inner relation in which it will not join. Hash join methods provide the same solution. They isolate the tuples of the outer relation that can join with a tuple from the inner relation. The tuples of the inner relation are compared with a limited set of tuples from the outer relation. The performance of hash-joins is $O(m+n)$. During the evaluation of the join operators in a distributed environment, communication cost of data shipping among sites must be also taken into consideration. Shipping cost of the data mostly overshadows the CPU cost of the join. Query optimizers must select faster networks to send/receive data during these operations to obtain the optimal QEP. Partitioned data is another factor. The type of the partitioning, horizontal, vertical or hybrid must be taken into account during the selection. Replication, site of the join operation, semi-join are other important issues. As it can be seen, selecting the best ways for better QEPs is more complicated in a distributed environment [92, 94].

Semi-join algebraic operation is an efficient and commonly used way of joining two relations in a distributed environment [19, 108, 123]. The semi-join operator can reduce the amount of work required to do a later expensive join. In a conventional join operation, the resulting relation has all the attributes/columns of both input relations. However, this is not a desired way of executing a join operation in a distributed database system. The main goal of semi-join is to reduce the cost of communication by sending the join attribute(s) of one relation to the site of the other and later sending all matching tuples from the second relation back to the site of the first relation. SDD-1 [19] was one the first distributed database management systems that make use of semi-joins and it is still an efficient way of executing joins in the recent distributed database query optimizers [65].

Multiobjective Query Optimization : In this part, we summarize some of the multiobjective query optimization studies related to our work. There has been a lot of research related to the Cloud, but relatively there is no approach like ours that concerns both with the optimization of the total ownership price and the performance of the queries by taking into account alternative virtual resource allocation and materialized view supported query plans.

Grid/Distributed databases can be considered as the first representatives of the Cloud DWs. Therefore, we first analyzed Mariposa, an early distributed database system that implements an economic paradigm to solve many drawbacks of wide-area network cost-based optimizers [149]. In Mariposa, clients and servers have an account in a network bank and users allocate a budget to each of their queries. The processing mechanism aims to service the query in the limited budget by executing portions of it on various sites. The latter place bids for the execution of query parts and the bids are collected in query brokers. The decision of selecting the most appropriate bids is delegated to the user. A series of similar works have been proposed for the solution of the problem [103][111]. Papadimitriou et al. [119] showed that Mariposa's greedy heuristic can be far from the optimum solution and proposed that the optimum cost-delay tradeoff (Pareto) curve in Mariposa's framework can be approximated fast within any desired accuracy. They also present a polynomial algorithm for the general multiobjective query optimization problem, which approximates the optimum cost-delay tradeoff.

An advisor automatically configures a set of VMs for database workloads where the advisor requests domain knowledge in [144]. Although Soror's approach concerns with the efficient allocation of the VMs, it does not optimize the total ownership price of the system.

Recently, efficient cost models have been proposed in the Cloud for scheduling of data flows with regard to monetary cost and/or completion time [90] and cost amortization of data structures to ensure the economic viability of the provider [87], particularly for self-tuned caching [43] and for a real-life astronomy application using the Amazon Cloud [18]. New cost models that fit into the pay-as-you-go paradigm of Cloud computing are introduced in [115]. These cost models achieve a multiobjective optimization of the view materialization vs. CPU power consumption problem under budget constraints. It is shown that Cloud view materialization is always desirable. Koutris et al. [95] built a theoretical foundation, the first one towards a practical design and implementation of a pricing system. They present a formal framework for query-based pricing. Central to this framework are the notions of arbitrage-free and discount-free pricing.

In [44], the cost performance tradeoffs of different execution and resource provisioning plans have been simulated, showing that by provisioning the right amount of storage and computing resources, cost can be reduced significantly. The performance of three workflow applications with different I/O, memory, and CPU requirements has also been compared on Amazon EC2 and a typical high-performance cluster (HPC) to identify what applications achieve the best performance in the Cloud at the lowest cost [18].

Recent research takes interest in various aspects of database and decision support technologies in the Cloud. Different studies investigate the storage and processing of structured data [33], the optimization of join queries, and how to support analysis operations such as aggregation [38]. Cloud data warehousing and OLAP systems also raise various problems related to storage and query performance [101]. Adaptations of these technologies to the Cloud are addressed in [13], or the calculation of OLAP cuboids using the MapReduce runtime environment [180]. In [162], a virtual-machine provision policy based on marginal cost and revenue functions is proposed.

In [138], a cost-aware provisioning system for Cloud applications that can optimize either the rental cost for provisioning a certain capacity or the transition cost of reconfiguring an application's current capacity is proposed. The system exploits both replication and migration to dynamically provision capacity and uses an integer linear program formulation to optimize cost.

There has been a great amount of work for tuning databases for workloads and execution environments [167] [3] [150]. Our study optimizes the objectives of minimum money consumption and maximum benefit from the virtual resources and optimizes this with an efficient multiobjective genetic algorithm. In summary, our study focuses on the elasticity of Cloud resources and produces multiple resource deployment plans with alternative query plans for a set of queries, enabling the user to select the desired tradeoff with efficient cost models. To the best of our knowledge, no resource deployment processing system deals with the concept of elasticity and cost-efficiency on relational Cloud databases like our system.

The MQO problem was first defined in 1980s and finding a global optimal QP by using MQO was shown to be an NP-Hard problem [134] [55] [135]. A detailed theoretical study of query scheduling, caching, and pipelining in MQO can be found in [46]. Considerable amount of MQO work has been done on relational databases [113] [127] [11]. The idea of using joint subexpressions has been applied to batch execution of multiple related queries and efficient maintenance of materialized views [122] [126]. The studies in [55] [152] considered these optimizations and used only the best plans of queries, thus achieving less sharing (i.e. higher total cost) than that could be obtained by using suboptimal query plans. Cosar et al. provide heuristics and methods for generating alternative query plans that will improve the performance of MQO in [121]. The execution time of a batch of queries is improved by evaluating a common subexpression once obtained by using a light-weight and effective mechanism for detecting potential sharing opportunities among subexpressions [180].

Detecting common expressions. It is mentioned that decomposition of the queries at the early stages of MQP diminishes the complexity of the combinations for uncorrelated queries[81]. Wong proposed a matrix and graph connectivity algorithm to detect connected components for processing single queries [171]. This approach is extended for connectivity detection of MQO in [36]. The problem of identifying common subexpressions is proved to be NP-hard in [86]. Therefore, Jarke states that multi-relation subexpressions can be addressed heuristically. Finkelstein shows how a query can be improved by comparing an incoming query with materialized results [55]. Jarke analyzes common subexpression isolation in relational algebra. Chakravarthy and Minker define the equivalence and subsumption of two expressions at the logical level, using heuristics [29]. Rosenthal and Chakravarthy use an AND/OR graph to represent queries and detect the subsumption by comparing each pair of operator nodes from distinct queries [134]. The representation and the processing of multiple queries is another issue. Chakravarthy proposed the multigraph for representing multiple Select-project-join queries [29]. The multigraph facilitates query processing by using Ingres' instantiation and substitution [29, 171]. In [30], the multigraph was modified to represent the initial state of multiple queries. The approach in [36] differs from these. They provided a simple technique to decompose a set of queries into unrelated sets. After the creation of the subsets, the queries in each subset are executed separately. This decomposition is performed before the determination of common subexpressions to reduce the size of query sets which the multiple query processing algorithm needs to consider, which is similar to Chakravarthy and Minker's [29].

When we survey MQO on distributed/parallel databases, we can see early studies such as:

- Increasing inter-query locality by decomposing a query into parallel sub-tasks so that a scheduler rearranges the execution order to maximize the reuse of cached-data [141],
- Resource usage models to perform multiple query scheduling on parallel query

processing systems in order to reduce the response times of queries [60],

- Dividing a query into sub-queries that can be executed in parallel on many processors and enabling already computed sub-query results to be re-used for improving processing speeds of new queries [11].

Mehta and DeWitt developed algorithms to take advantage of intra-operator parallelism, used CPU loads and tuple production rates of select and hash-join database operations for deciding on the number of allocated processors and the assignment of database operations to these processors [105]. Distributed query processing middleware systems have also been extensively studied as a solution for data intensive scientific applications. MOCHA [124] was one of the first sample database middlewares developed to execute database queries over distributed data sources. MOCHA can move the code required to process the query to the location where the data resides. In Beynon [22], user-defined functions can be executed at data storage sites to perform subsetting operations and many *filter* (e.g. aggregation) operators can be run in parallel on a large number of computers.

Indexing the data at each server is an efficient method for distributed query optimization. R-trees are widely used to index and integrate the back-end servers as a single query server. Parallel R-trees, Master R-trees, and Master-Client R-trees are mechanisms used for improving the performance of shared-nothing environments [131]. Mondal et al. used data migration to shift the workload from intensely working servers to idle servers in shared-nothing environments [110]. Chen et al. considered the network layer of the problem and reduced the communication costs with a query reconstruction algorithm to enable sharing of overlapped data through micro-engines that collaborate for evaluating query batches [35].

IGNITE [99] , OGSA-DQP [148] [57], CoDIMS-G [114], and GridDB-Lite are some of the important projects that focus on Cloud/Grid data integration [75] [10] [92]. Except IGNITE, none of these systems has MQO support and they focus on improving the performance of only single queries.

Recently, there are studies to adapt traditional query optimizers to Cloud computing. In [20], a classical query optimizer is adapted to Cloud computing workloads where it uses a partitioned database on a shared-nothing architecture. In [137], a parallel DW system optimizer is developed for single queries by considering a rich space of execution alternatives with bushy-tree plans instead of simply parallelizing the best serial plans. Query optimizations in Cloud environments can have different goals unlike the traditional query optimizers, and the search space is much larger because of the hardware flexibilities (e.g. number of data processing nodes) provided by the Cloud environment. In an interesting study, the scheduling of data processing workflows on the Cloud is considered from the perspective of minimizing the completion time given a fixed budget [90].

Although there are some studies to integrate MQO techniques into existing relational Cloud database query engines, to our knowledge, there is no approach that optimizes a batch of queries into a relational Cloud database query optimizer by exploiting alternative query execution plans. Recently, there were two remarkable projects. Kossmann et al. developed a new database architecture that is based on batching queries and shared computation across many concurrent queries in a shared multi-processor environment [63]. Their model does not generate any new alternative plans for input queries. A framework is developed for a Cascade-style Cloud query optimizer to enhance the performance by using Multiple Query Optimization (MQO) techniques for massive data analysis scripts that contain common subexpressions [140] but this approach differs from our technique because new alternative plans are not generated and subexpression costs are used for making optimization decisions only.

2.6 Summary

In this chapter, we gave brief information about Cloud computing, Cloud DWs, Cloud DW hardware/software architectures, relational Cloud DW design techniques, and related query optimization techniques (in terms of single queries and multiple queries). The information given here lays the ground for the next Chapters to comprehend the details of the underlying architecture during the application of multiobjective design of relational Cloud databases. Chosen architectures affect the efficiency of the Cloud DWs. The model we will work throughout the study will be a shared-nothing architecture.

CHAPTER 3

A NOVEL CLOUD DATA WAREHOUSE QUERY OPTIMIZER

In this Chapter, we propose a novel heuristic algorithm for the query optimization of Cloud DWs, Dynamic Programming with Ant Colony Optimization Algorithm (DPACO). The proposed algorithm makes use of Dynamic Programming (DP) (see Chapter 2 for details of DP) and the state-of-the-art contemporary metaheuristic, Ant Colony Optimization (ACO) while pruning the search space of alternative query plans [47, 49, 48]. The algorithm optimizes the response time of a Cloud DW query very efficiently and it is used by our design algorithms, MOD-B&B and MOD-GA, while estimating the workload response times of the alternative Cloud DW designs. The query optimization is an important issue for Cloud DWs in case you have several alternatives to execute a query. The alternative ways of finding optimal plans increase exponentially in accordance with the order of joins, the number of virtual machines (VM), and the number of tables. DPACO finds optimal results for queries up to 6-way joins. For larger multi-way chain join queries of the Cloud DWs, DPACO finds (near-)optimal response times. With DPACO, we enhance the capabilities of DP, a widely used algorithm in existing commercial query optimizers, by adding an efficient pruning technique, Ant Colony Optimization (ACO) to reduce its exponentially increasing search space.

The architecture of the Cloud DW that we study on is a shared-nothing environment (see Figure 3.1), which is known as the most scalable architecture. The queries are submitted to the control node. After decomposing the query into sub-tasks, they are processed by VMs.

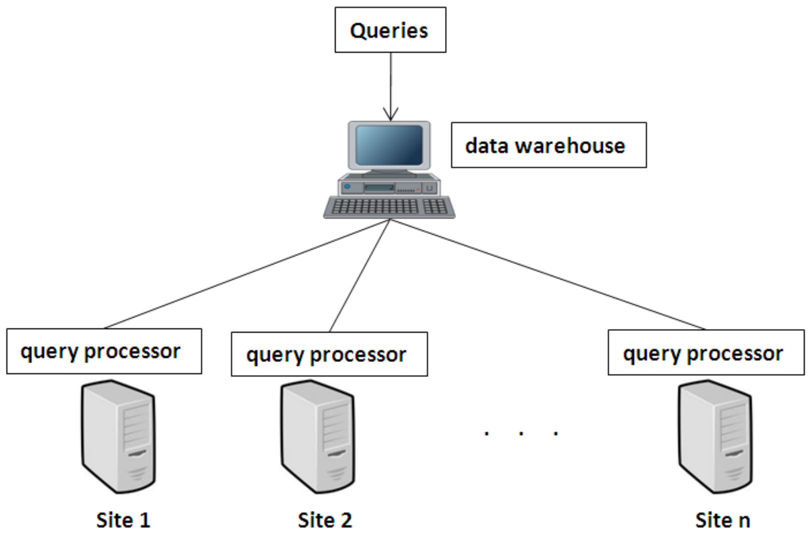


Figure 3.1: Distributed shared-nothing data warehouse architecture.

3.1 Ant Colony Optimization (ACO) Metaheuristics

Ant Colony Optimization (ACO) Metaheuristic is inspired by the behavior of real ants where individuals cooperate through self-organization. Dorigo and colleagues proposed this property of ants as a metaheuristic for solving difficult combinatorial problems [51]. French entomologist Pierre-Paul Grasse was the first researcher to investigate the social behavior of insects. He used the term stigmery to define the indirect communication among insects. A substance called *pheromone* is deposited on the ground while the ants are foraging and thus pheromone trails are formed on the ground. Stochastic fluctuations at the initial phases are reduced by this mechanism and the shorter trails are used more frequently as they gain more pheromone. Goss et al. developed a formula to explain the behavior of the ants [69]. Assuming m_1 ants had passed the first bridge and m_2 the second one, the probability of (m_1+1)-th ant to choose the first bridge is given in Equation 3.1. The values of k and h are chosen to fit the experimental data.

$$P_1(m+1) = \frac{(m_1+k)^h}{(m_1+k)^h + (m_2+k)^h} \tag{3.1}$$

Analogously, pheromone laying of ants can be simulated by artificial agents that modify the appropriate pheromone values associated with the edges of a graph. For example better solutions of the travelling salesman deposit more pheromones on the paths and by the evaporation mechanism artificial ants may forget the history of the solutions and search for new directions. After the initialization of the algorithm with zero pheromones, a set of artificial ants constructs solutions using a domain specific partial solution construction algorithm. The actions that single ants cannot do are implemented at the phase of *ApplyLocalSearch* procedure. By evaporating and laying, the pheromone values for partial solutions are updated at the phase of *UpdatePheromones*. ACO metaheuristic is presented in Algorithm 3.1.

Algorithm 3.1: Ant Colony Optimization (ACO) Metaheuristics

```
while Termination conditions not met do
    Construct Ant Solutions
    Apply Local Search
    Update Pheromones
```

3.2 Dynamic Programming with ACO Algorithm (DPACO)

DP works in a bottom-up manner constructing all the sub-plans untilan optimal solution is obtained (See Chapter 2 for details of DP). Any given sub-problem is constructed only once and saved in a table, therefore it is not computed again. The algorithm first builds access plans for every relation in the query and there may be several different access plans. If relation R_1 is replicated in different sites, say Site1 and Site2, the algorithm enumerates *table-scan (R_1,Site1)* and *table-scan (R_1,Site2)* as alternative plans. DP enumerates all two-way join plans in the second phase. Alternative join plans are enumerated, including the data shipping cost of joins for all sites. All multi-way join plans are built using individual relation access-plans and previously discovered optimal multi-way join plans as building blocks. DP continues until it has enumerated all the n-way join plans. The *joinPlans* function produces more and more complex plans using the smaller plans as building blocks and returns a new plan for every alternative join method. DP discards all of the “inferior” partial-plans (i.e. a better plan exists for calculating the same output as that of the partial-plan). This is called pruning and carried out by the *prunePlan* function. An efficient plan can prune another plan with the same output. DP enumerates (R_1 $\bowtie$

R_2) and ($R_2 \bowtie R_1$) as two alternative plans, but only the better plan is kept in the *optPlan(R_1,R_2)* structure after pruning which reduces the search space. *table-scan (R_1,Site1)* and *table-scan(R_1,Site2)* do the same job, but they produce results at different sites. So, they cannot be pruned in order to ensure that the optimizer is guaranteed to find the best plan. For example, if the cost of *table-scan(R_1,Site1)* and the cost of shipping R_1 from Site1 to Site2 is lower than the cost of *table-scan(R_1,Site2)*, then *table-scan(R_1,Site2)* is pruned.

Dynamic Programming with ACO Solution Model

DPACO simulates the actions of ants on the directed acyclic graph (DAG) of QEPs. Each process attempting to discover the pheromone of the next execution plan simulates ants and the solutions represent the food. Ants look for better execution plans where the paths correspond to the edges of the communication infrastructure of the Cloud. DPACO starts with the initial stage of the DP.

For example, in order to construct (A$\bowtie$ B $\bowtie$ C $\bowtie$ D) join in a bottom-up manner, at first stage of the algorithm 2-way sub-joins (A $\bowtie$ B), (B $\bowtie$ C), (C $\bowtie$ D) are generated at each site S_1 ,..., S_n, where n is the number of the sites. Therefore, we have n times (A $\bowtie$ B) sub-joins, n times (B $\bowtie$ C) sub-joins, and n times (C $\bowtie$ D) sub-joins in the list. The solution list holds the costs of all the sub-join of (A $\bowtie$ B) at every site. Next, we need to construct (A $\bowtie$ B $\bowtie$ C) and (B $\bowtie$ C $\bowtie$ D) 3-way joins to find the best plan for the (A $\bowtie$ B $\bowtie$ C $\bowtie$ D) multi-way join. Equation 3.2 shows the probability of selecting the site of the new sub-join. For (A $\bowtie$ B $\bowtie$ C) sub-join, P^k_{ij} represents the probability of ant number k making the choice of performing the join between relation C (residing at any site) and (A $\bowtie$ B) (residing at S_i) to obtain (A $\bowtie$ B $\bowtie$ C) result at S_j. There are as many possible paths as the number of sites. The symbol l represents all the allowed paths to construct (A $\bowtie$ B $\bowtie$ C). τ_{ij} is the pheromone value of (A $\bowtie$ B) evaluated at S_i that is based on the response time of (A $\bowtie$ B) at different sites. The faster the response time of (A $\bowtie$ B), the higher its pheromone level is.

$$P^k_{ij} = \frac{\tau^{\alpha}_{ij}.\eta^{\beta}_{ij}}{\sum\limits_{l \in allowed_k} (\tau^{\alpha}_{il}.\eta^{\beta}_{il})} \tag{3.2}$$

Algorithm 3.2: Dynamic Programming Algorithm with Ant Colony (DPACO)

```
Input: Select Project Join query $q$ on relations $R_1, \dots , R_n$
Output: a query plan for $q$
Set Search Space Limit ($SSL$)=k
for $i \leftarrow 1$ to $n$ do
    optPlan ($\{R_i\}$)= accessPlans ($R_i$)
    prunePlans(optPlan ($\{R_i\}$))
for $i \leftarrow 2$ to $n$ do
    for all $S \subset \{R_1, ..., R_n\}$ such that $| S |=$ i do
        optPlan ($S$)= $\emptyset$
        for $O \subset S$ do
            updatePheromones ($O$)
            calculateProbability($O$)
            prunePlans($S$,$SSL$)
            calculateCostsOfRemainingPlans($O$)
finalizePlans(optPlan($\{R_1, ..., R_n\}$))
prunePlans(optPlan($\{R_1, ..., R_n\}$))
return optPlan($\{R_1, ..., R_n\}$)
```

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{3.3}$$

η_{ij} is the heuristic information (Equation 3.3) where d_{ij} represents the distance of (A ⋈ B) at S_i from relation C at S_j. (A ⋈ B) and relation C can be at the same site or different sites. The site of the join determines the distance. If both are at the site of the join, the distance is taken as 1, if one is at the join site the distance is taken as 2, and if both are at different sites from the join site, then the distance is taken to be 4. These values are decided after exhaustive tests with different values. α and β control the relative importance of pheromone (τ_{ij}) versus the heuristic information and are decided as 1 (meaning that the heuristic information and pheromone level have the same importance).

After finding the probabilities of all the (A ⋈ B ⋈ C) joins, plans are pruned by a limitation value called, SSL. If the SSL is 2 then there are two plans left to prevent the exponential expanding of the search space. The costs of these two plans are calculated and prepared for the next level of the joins (see Algorithm 3.2). The total value of all pheromones out of a partial solution is decided to be as 1. The costs of the sub-solutions that are generated by joining a new base-relation to the current partial-solution (or base relation) are calculated and pheromone levels for each possible path from that sub-solution are assigned accordingly. After finding all possible sub-solutions achievable at that level, pruning is performed by selecting a fixed number (SSL) of the lowest cost sub-plans to obtain each distinct sub-solution.

In Figure 3.2, how DPACO explores the search space of left-deep QEP (A ⋈ B ⋈ C) can be seen. All of the relations are located at different sites. Relation A is at S_1, relation B is at S_2, and etc. DPACO searches for the best order to evaluate the join of (A ⋈ B ⋈ C).

At the initial step, the ants are located at the first level (access plan level) and they can see the possible available open paths to construct a sub-join. The only possible join for the leftmost ant without any cross-product is (A ⋈ B). Looking at the four paths, each ant makes a decision. In the second step, the leftmost ant is at S_1 and can perform the (A ⋈ B) sub-join. In the next level, this ant makes a decision and chooses to join with relation C at S_1 depending on the most promising pheromone level. Now,

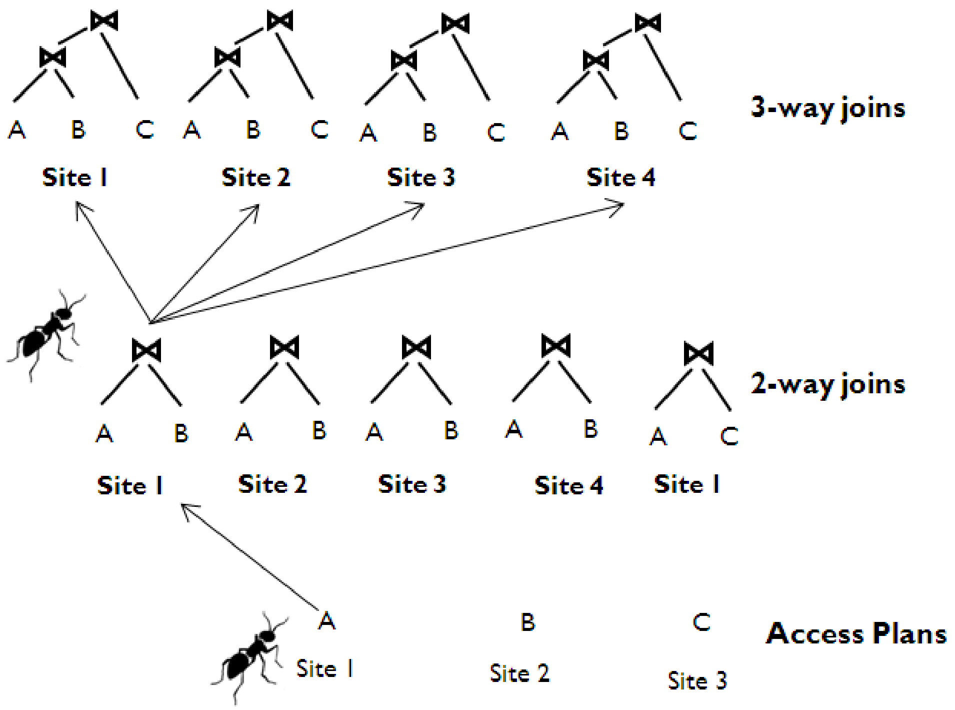


Figure 3.2: Illustration of the ants during DPACO algorithm

the leftmost ant has a (A ⋈ B ⋈ C) sub-join with itself. Meanwhile, the other ants also keep searching for a complete solution. The number of ants searching for a solution to problem depends on the initially fixed SSL value. In the final step, we can evaluate all the solutions discovered by ants and choose the best one. Although the paths have not been traversed by any other ant(s) before and there is no pheromone present, there is sufficient information for the ants to use to choose the way ahead. The calculated costs of the sub-solutions, bandwidth of the network, cardinalities of the relations to be joined are some of the available information to guide the ants towards a good solution. This decision mechanism can be extended by adding new parameters such as available indexes and join methods.

Search Space Limit (SSL) :

SSL is a pruning mechanism for uncontrollably large search space of the join queries. For example if the sub-join of (A ⋈ B) is constructed at 10 different sites while searching for bigger query plans and the SSL value is 2, then the most promising two (A ⋈ B) sub-joins are kept and the other eight sub-joins are pruned. Therefore, memory demand for the optimization is reduced. Without any pruning, DPACO acts like DP. The SSL can be set depending on the tradeoffs of the algorithm. Greater SSL values increase the time and the search space demanded by DP while they can generate more qualified query execution plans.

3.3 Performance Evaluation of DPACO Algorithm

We compared the performance of DPACO with DP, IDP_1 [94], and Simple Genetic Algorithm (SGA) on left-deep trees of QEPs. The details of the algorithms are given below.

IDP_1 Algorithm

IDP_1 is one of the most elegant hybrid algorithms in query optimization and produces as good plans as DP if there are sufficient resources available. It is also very appropriate for the architecture of shared-nothing Cloud DWs. IDP_1 has been proven to be better than any previously developed randomized algorithm [94]. This is the main reason why we choose to compare IDP_1 with DPACO. IDP_1 introduces *block size* strategy taking sub-optimal plans and pruning the search space when it reaches block size. After pruning the sub-solutions, IDP_1 restarts applying DP until it reaches the next block size. During the experiments, the block size of IDP_1 is selected as 1 to be able to reach upper levels of joins. IDP_1 with larger block sizes give better execution plans but this increases the running time complexity exponentially. At every block size the best sub-plan is chosen and remaining plans are pruned. This sub-plan can be any combination of the existing relations.

Increasing the block size parameter of IDP_1 and the SSL value of the DPACO causes an exponential rise in its optimization time.

Simple Genetic Algorithm (SGA)

SGA is an efficient heuristic algorithm that works with the principles of evolutionary computation. It is developed for comparison tests of DPACO algorithm. GAs are frequently being used to solve optimization problems since they were first introduced by Holland [80]. They have gained this prominence due to their robustness and simplicity they offer. Individuals with higher aptitude have more probability to survive, to reproduce, and to transmit their genetic characteristics to next generations. GAs can perform efficient search operations in search spaces where it is not easy to understand the environment. Each potential solution existing in the search space is considered as an individual and individuals are represented by using *bit-strings* that are called *chromosomes*. Genes are the atomic parts of chromosomes and they codify a specific characteristic of a chromosome.

GAs have been previously used for the solution of database query optimization problem [147] but SGA is different in its approach that it makes use of the commutativity property of the join operator and prunes the search space with a big ratio. SGA provides optimal solutions for the queries with a small number of joins (up to 5-6 joins) and near optimal solutions for larger problems.

Selection mechanism of SGA is *tournament* and the size of the population is 100. At every generation of the algorithm, as many as half of the population size mutations and crossovers are applied and the worst query execution plans (QEP) are replaced with the new and better QEPs. Mutations are randomly set with the defined probability ratios and the crossovers are *one-point*. SGA terminates the optimization process when the QEPs do not improve more than the previously defined refinement threshold (5% less execution time than the best QEP in the population). The SGA algorithm and the details of the parameters used by the SGA are given in Algorithm 3.3 and Table 3.1 respectively. The chromosome structure of SGA can be seen in Figure 3.3. The numbers assigned to genes indicate the site where the corresponding join operation will be executed. The execution order of the join (in accordance with the structure of left-deep trees) is that relation A and B are joined at site 2 (S_2) to build (A ⋈ B), the obtained result and relation C are shipped to S_1 to build ((A ⋈ B) ⋈ C), Relation D is shipped to S_1 for (A ⋈ B ⋈ C ⋈ D) with ((A ⋈ B) ⋈ C) and so on. The intermediate results are materialized in the main memory.

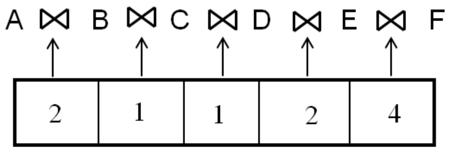


Figure 3.3: Chromosome structure of SGA.

The order of the joins are evaluated without any Cross-Product operator. An example of join orders with respect to the names of the relations can be seen in Table 3.2.

Algorithm 3.3: Simple Genetic Algorithm (SGA)

```
$p$: population
$par$: individual selected from population
$s$: generated individual
$p \leftarrow$ generate random individuals ($Q$)
evaluate fitness of individuals ($p$)
$p \leftarrow$ truncate ($p$)
for $k \leftarrow 1$ to generations do
    ($par_1$,$par_2$)$\leftarrow$ select pair of parents ($p$)
    $s \leftarrow$ evaluate (crossover ($par_1$,$par_2$))
    replace with least-fit in the population ($p$,$s$)
    $s \leftarrow$ evaluate (mutation ($p$,$s$))
    replace with least-fit in the population ($p$,$s$)
return the best individual in the population
```

Commutativity property (A ⋈ B = B ⋈ A) of a join operation will also help in reducing the search space of the multi-way chain joins. All possible permutations of a 4-way chain-join left-deep tree of (A ⋈ B ⋈ C ⋈ D) without any Cross-Product operators can be seen in Table 3.2. For a 10-way join operation the number of join orders is 256 but with the property of commutativity the size of this search space is reduced to 50 (the selection of the sites during the execution is not included).

Table 3.1: Parameter settings for Simple Genetic Algorithm (SGA)

Parameter	**Value**
population size	100
number of generations	100
population initialization	random
crossover	one-point
parent selection	tournament
tournament size	10
mutation ratio	1%

The SGA generates a random population at the first step of the algorithm and then

Table 3.2: All possible left-deep tree orders of (A ⋈ B ⋈ C ⋈ D) 4-way join. search space that is reduced to 4 instead of 4!

Number	Join Order
1	A ⋈ B ⋈ C ⋈ D
2	B ⋈ C ⋈ A ⋈ D
3	B ⋈ C ⋈ D ⋈ A
4	C ⋈ D ⋈ B ⋈ A

selection, crossover, and mutation operations are applied repetitively, creating new generations of individuals [64]. The individual with the best fitness value in the population will be the proposed solution of the problem. For each pair individuals chosen for mating using a selection mechanism, the parent chromosomes are split into two parts and the parts are exchanged (crossed) to generate two new chromosomes (see Figure 3.4). The original solution pool is combined with the newly produced individuals and becomes the candidate set for the next generation. Another mechanism for searching the solution space is to allow random "mutations" to occur in chromosomes. Thus, the newly formed chromosomes also become part of the pool and are used in the evaluation of next generation. Mutations prevent stagnation of the search; enable the exploration of the search space that could not be reached with the crossover operators alone [136] (see Figure 3.5). After executing genetic operators, a new population is generated and another iteration begins. To measure the statistical confidence of the SGA, standard deviation of each solution is calculated by executing SGA 20 times and validated that all measurements are within 2% of averages with 95% confidence.

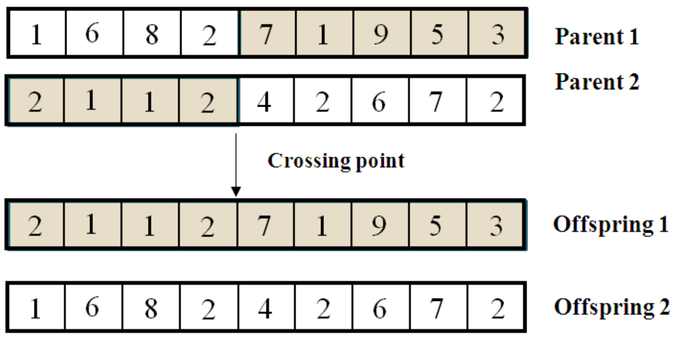


Figure 3.4: Crossover operator.

We use synthetic relational DWs that are derived from the relations of TPC-H bench-

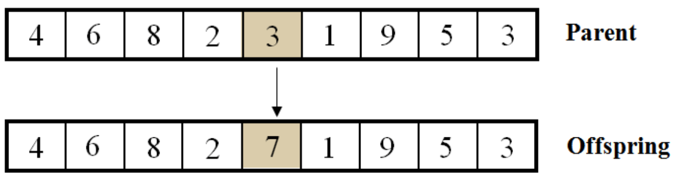


Figure 3.5: Mutation operator.

mark during our experiments. Relations are sorted by their primary keys and stored in text files. Type of each data is a string with average length of five to ten characters. Relations with increasing cardinalities are generated to test the performance scalability of the developed algorithms with various queries and combinations of relations. Joins are assumed to be implemented by using merge-sort. Database schemes allow 15-way chain queries. All the relations are assumed to be stored in such a way that each relation resides at a different site. The cardinality of the relations and the selectivity of the join predicates are varied in the range of 10% to 50% for all the experiments so that we were able to evaluate performance of our algorithms on different databases. The selectivity of our queries are also varied so that the query result sizes can be large or small. The referential integrity is guaranteed to be satisfied by all relations. Fragmentation and replication of relations are not modelled in our experiments but their effect on our algorithms can be easily calculated by modelling fragments as separate relations and re-writing our queries. We do not attempt to locally optimize a given access path of a query therefore, all queries are assumed to be irreducible.

SQL statements are multi-way chain joins where relations are unique, as in (A ⋈ B ⋈ C ⋈ D ⋈ E ⋈ F). We limited our SQL statements to chain equijoin queries with a single selection predicate because complex queries might mislead the comparison of algorithms. Semi-joins are also considered during plan generation in the experiments. We added a new site for every new relation in the join operation. For ten relations, we have ten sites. Relation A is at S_1, Relation B is at S_2, and so on. The response time of QEPs includes; Scanning relations from disk, shipping (if join site is different) the relations over a network (with homogeneous bi-directional links), and equijoining the relations. A sample SQL statement with 6-way joins is:

```
SELECT A.Name
FROM A, B, C, D, E, F
WHERE A ⋈ B ⋈ C ⋈ D ⋈ E ⋈ F
AND F.shippingdate > '2.3.2014'
```

With three different DW schema and different SQL statements, experiments are performed using the DPACO, DP, IDP$_1$, and SGA. SQL statements are extended up to 15-way joins. Running DP with more than 6-way joins requires a large amount of time and memory resources. The optimization times of the algorithms are shown in Figure 3.6.

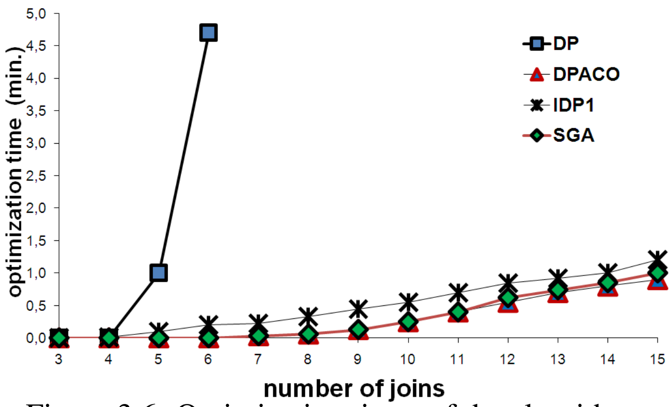


Figure 3.6: Optimization times of the algorithms.

The DPACO, DP, IDP$_1$, and SGA algorithms produce the same solutions up to 6-way joins. The running time of SGA algorithm is observed to increase as a polynomial function with the corresponding number of joins and the termination condition. When we use the experimentally determined proper values for the population size and termination condition, the quality of SGA execution plans are similar to those of the DPACO, DP, and IDP$_1$. It is clear that optimization times of DPACO, IDP$_1$ (with block size 1), and SGA are polynomial and the search space sizes of IDP$_1$ and DPACO increased linearly with the number of sites, whereas the running time of the SGA is not affected but its generated plan quality degraded as the number of sites becomes larger. DPACO has an almost linearly increasing search space size (if SSL value is kept as 1 or 2). The search space of IDP$_1$ is almost the same with DPACO. However, for block sizes bigger than 1, DPACO outperforms IDP$_1$. Calculating the costs of the plans is the major factor that increases the running times of all the algorithms. Because DPACO pre-prunes plans with lower pheromone values before calculation and decreases the optimization time of the algorithm significantly.

DPACO algorithm produces QEPs in slightly shorter times than IDP$_1$ up to 15 joins, and as the problem size grows beyond 10 joins the DPACO performs better than IDP$_1$.

The QEPs produced by the algorithms can be seen in Figures 3.7, 3.8, and 3.9 for database1, database2, and database3 respectively. All the algorithms provide the same quality solutions up to 6 way joins. Except DP, the optimization times of the algorithms are reasonable with the given settings (black size is 1 for IDP_1, and SSL is 1 for DPACO). DPACO, IDP_1 and SGA are able to produce QEPs up to 15 way chain joins where it is not possible to produce QEPs with DP. The QEPs provided by IDP_1 are slightly better than the plans of SGA. On the other hand, DPACO produces the best performing QEPs when compared with the other algorithms.

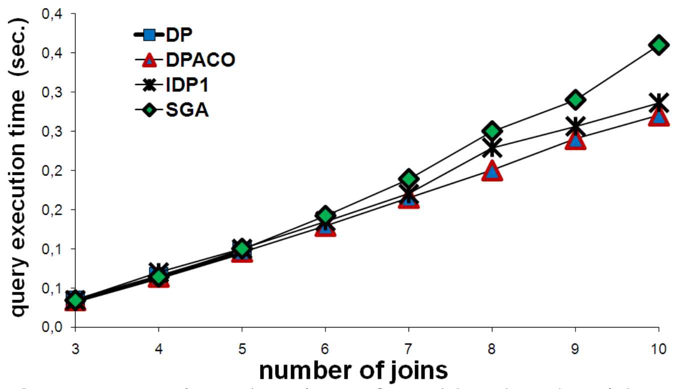


Figure 3.7: Query execution plan times found by the algorithms for DW 1.

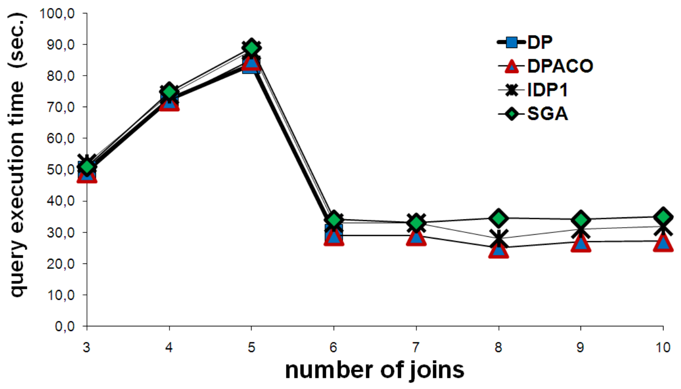


Figure 3.8: Query execution plan times found by the algorithms for DW 2.

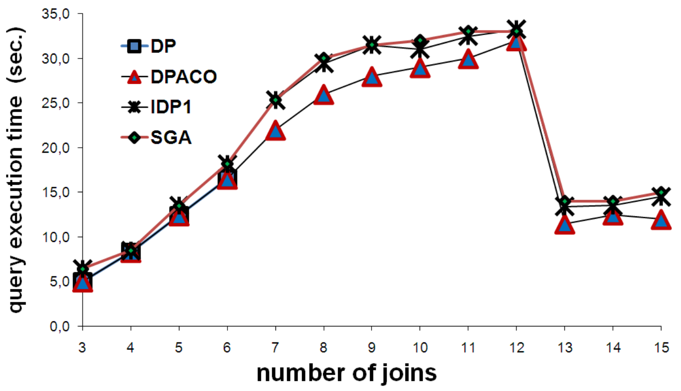


Figure 3.9: Query execution plan times found by the algorithms for DW 3.

CHAPTER 4

MULTIOBJECTIVE CLOUD DATA WAREHOUSE DESIGN

In this Chapter, we define our multiobjective method for designing Cloud DWs. Two objectives we aim to optimize are the monetary cost of the designed DW and the response time of the query workloads on that design. The response time of the query workloads improves as expensive DWs are designed and cheaper DW designs will cause the monetary costs to be cheaper while also increasing the response time of the query workloads. Therefore, these two objectives work in opposite directions and make the design problem multiobjective. DPACO query optimization algorithm we proposed in Chapter 3 is used to estimate the overall response time of a query workload with the given VM configuration and location of Tables. The total time spent for the query workload response time is calculated in terms of the virtual resources consumed by the design.

In the following sections, multiobjective relational Cloud DW design formulation, Cloud infrastructure and pricing scheme parameters of the Cloud, two algorithms for the multiobjective Cloud DW design, Branch-and-Bound Algorithm (MOD-B&B) and Genetic Algorithm (MOD-GA), and (near-) optimal parameter settings for MOD-GA are presented.

4.1 Multiobjective Relational Cloud DW Design Formulation

In this section, we explain our multiobjective design method and give the problem formulation. Our design method can be used for any relational Cloud DW that has select-project-join intensive queries. In the proposed model, the inputs of the system are the tables of relational Cloud DW (including the histograms and statistics), Directed Acyclic Graphs (DAGs) of the queries (most frequently used queries), and the pricing schema of the virtual resources. After obtaining this information we choose the most efficient virtual resources, optimal query execution plans to find the pareto-optimal set of solutions with fast response times and minimum monetary costs. Tables 4.1 and 4.2 give a sample scenario of two alternative configuration for a TPC-H decision support relational DW. Tables are located on 3 and 5 VM configurations respectively. Small tables, Supplier, Region, and Nation are replicated at all VMs (we use this heuristic for all designs throughout the study). After this assignment, they are evaluated with the given queries to find the best performing configuration. The configuration at Table 4.1 has 3 XS VMs that use (1 x 2Ghz CPU, 768MB RAM, 100 Mbps network bandwidth) and the configuration at Table 4.2 uses 5 XL VMs (4 x 2Ghz CPU, 8GB RAM, 300Mbps network bandwidth). The location of the table on the VM is indicated with number 1. The configurations give different response time performances and monetary costs. The main goal of the design is to minimize the cost given in Equation 4.7.

Table 4.1: TPC-H Database configuration of 3 Extra Small (XS) Virtual Machines

Table Name	VM0 (XS)	VM1 (XS)	VM2 (XS)
LINEITEM	1	0	0
CUSTOMER	0	1	0
ORDERS	0	0	1
PART	0	1	0
PARTSUPP	0	0	1
SUPPLIER (replicated)	1	1	1
REGION (replicated)	1	1	1
NATION (replicated)	1	1	1

In our approach, we have a query optimizer, DPACO, for estimating the response of the query workload of a given distributed DW design. The monetary cost is calculated depending on the virtual resources spent by the design. Below, we first give the definitions of some terms that are frequently used to provide the reader a better understanding and later, we present the mathematical representation of our multiobjective design of a Cloud DW.

Table 4.2: TPC-H Database configuration of 5 Extra Large (XL) Virtual Machines

Table Name	VM0 (XL)	VM1 (XL)	VM2 (XL)	VM3 (XL)	VM4 (XL)
LINEITEM	1	0	0	0	0
CUSTOMER	0	1	0	0	0
ORDERS	0	0	1	0	0
PART	0	0	0	1	0
PARTSUPP	0	0	0	0	1
SUPPLIER (replicated)	1	1	1	1	1
REGION (replicated)	1	1	1	1	1
NATION (replicated)	1	1	1	1	1

Cloud DW : A DW that is located over a set of VMs on a network. Each table of the relation DW is located on VMs. VMs communicate with each other by paying a cost required by Cloud vendors (see Section 2.2 for detailed information).

Virtual machine (VM) : is a software that emulates the architecture and functions of a real computer. A VM consists of CPUs, RAM, disk, and network parameters. The number of its processors and main memory can be changed through virtualization.

Workload : A set of queries that are submitted to a distributed DW. Our relational DW is a distributed database so that the tasks of the queries are sent to the related VM and executed to produce a result. The workload consists of the standard queries of well-known TPC-H decision support benchmark.

Response time (Resp_time) : The time that has elapsed between the submission of the query workload and obtaining of the results.

Table Location (TL) : defines the number of the VM where the table of relational Cloud DW is located.

Monetary cost (Money_cost) : of a Cloud DW workload includes renting the resources to execute the workload on the relational Cloud DW. These resources are mainly, data storage, processing time of the VMs, and the sum of data transfer cost [115].

The formulation of the multiobjective Cloud DW design problem consists of two parts. Finding the best response time of the query workload and executing these tasks with minimum monetary cost that will work on the selected virtual resource configuration with the optimized QPs of the queries. If we give a formal definition of the problem. ϕ is the set of all VM configurations such as Extra Small (XS), Large (L), etc.

$$\phi = \{VMC_1, ..., VMC_n\} \tag{4.1}$$

where n is number of virtual machines, τ is the set of table locations and m is the number of Tables in DW.

$$\tau = \{TL_1, ..., TL_m\} \tag{4.2}$$

The solution vector x is :

$$x =< \tau, \phi, Q_{best-plan} > \tag{4.3}$$

The monetary cost of the computation includes, data storage, CPU usage (VM type), and communication cost. Data storage cost depends on the size of the data (including the structures such as indexes, and replications) and its storage time. Processing time of the VMs is the total price for CPU usage. During the execution of the queries, different VM configurations can be used and the configuration of a VM (RAM, number of CPUs, and etc.) is flexible in accordance with the resources used. Micro, small, large, and extra large are some of the configurations provided by the Cloud vendors at various prices [169]. Data transfer cost is related with the amount of data migrated between sites and the pricing model applied by the Cloud provider.

Response time cost model : In order to measure the response time of a workload with a configuration of VMs, we developed a response time based cost model [118][97] that uses left-deep QP trees [65]. The cost model depends on the statistics of the database catalog. The main parameters used in the cost model are shown in Table 4.3

[100]. The response time of the query workload is calculated with the parameters and statistics used by query optimizer, DPACO.

Table 4.3: Parameters used in the cost model

Symbol	**Definition**
$T_{I/O}$	I/O time for a page
#I/O	number of page I/O operations
seq_#I/O	max. number of sequential page I/O
T_{CPU}	time for a CPU instruction
seq_#insts	max. number of sequential instructions
T_{MSG}	time to initiate and receive a message
seq_#msgs	max. number of sequential messages
T_{TR}	time to transmit a page
seq_#pages	max. number of sequential pages
#insts	number of instructions

Using the number of pages as a parameter, the response time taken by a task is calculated as:

$$\begin{aligned} Resp\_time = (T_{CPU} * seq\_\#insts) + (T_{I/O} * seq\_\#I/Os) \\ +(T_{MSG} * seq\_\#msgs) + (T_{TR} * seq\_\#bytes) \end{aligned} \tag{4.4}$$

The network communication time of transferring an intermediate query result from one site to another is calculated as:

$$CT(\#pages) = T_{MSG} + (T_{TR} * \#pages) \tag{4.5}$$

The main goal is to obtain the pareto-optimal set of solutions such that the overall costs of workload and money are minimized. Equation 4.6 is the formulation of objective functions.

$$min_{<\phi,\tau,Q>\epsilon S}(\sum_{i=1}^{\#VM}\sum_{j=1}^{\#Q} Cost(VM_i, Best\_Plan(\phi, \tau, Q_j))) \tag{4.6}$$

where S is the set of all VM configurations from 1,..., $CF_{alternatives}$ and the alternative assignments of DW Tables with the best execution plans of the queries.

The overall multiobjective function can be represented as in Equation 4.7 (finding solutions closer to the hypothetical ideal point given in Equation 4.8). Equation 4.9 is the response time distance of the given solution vector to best response time, Equation 4.10 is the monetary distance of the given solution vector to best monetary cost. The best response time is obtained by using the most expensive VM configuration and the cheapest monetary cost is obtained by using the cheapest VM configuration (single extra small VM). These values are obtained heuristically. If any better value is obtained during the optimization, they are updated.

$$min(\sqrt{Objective\_1 + Objective\_2}) \tag{4.7}$$

$$Ideal\_Point(x) = (Min\{Resp\_Time(x)\}, Min\{Money\_Cost(x)\}) \tag{4.8}$$

$$Objective\_1 = (Resp\_time - Best\_Resp\_time(x))^2 \tag{4.9}$$

$$Objective\_2 = (Money\_cost - Best\_Money\_cost(x))^2 \tag{4.10}$$

Best_Resp_time(x) is the best response time and Best_Money_cost(x) is the minimal monetary cost for the whole search space of vector x on the given VM configurations and with table assignments.

4.2 Infrastructure and Pricing Scheme Parameters of the Cloud

In this section, we describe the infrastructure details of the Cloud DW and the pricing scheme that we use during the design optimizations. Each customer requests queries from the Cloud by using Internet and contacts with the aggregate node. The aggregate node distributes the query to the appropriate VM. The cloud infrastructure provides unlimited amount of storage space, CPU nodes, RAM, and very high speed intra-cloud networking. All the resources of the Cloud are assumed to be on a network. The CPU nodes, RAM, and I/O bandwidth of each VM are different from and can be deployed by using VM Monitors in milliseconds [15]. The storage system is based on a clustered file system where the disk blocks are stored close the CPU nodes accessing them. I/O bandwidth of the storage is divided evenly to the VMs (that may have multiple cores up to 8).

There are many Cloud Service Providers (CSP) in the market and they offer different pricing schemes for the services they provide. Every different pricing schema of Cloud server providers can be an opportunity for customers in accordance with the tasks they want to complete. In our study, we will use a pricing scheme that is similar to Window's Azure [169]. Configurations such as Extra Small, Small, and Medium VMs are provided by the Cloud service providers. The detailed information of VM configurations can be seen in Table 4.4. The cost for a small VM (1GHz CPU, 768MB RAM) is \$0.02/hr, whereas A7 (8 x 1.6GHz CPU, 56GB RAM) is \$1.64/hr.

Table 4.4: Virtual Machine prices

Symbol	Virtual Machine Configuration	**Price**
XS	1GHz CPU, 768MB RAM	\$0.02/hr
S	1.6GHz CPU, 1.75GB RAM	\$0.06/hr
M	2 x 1.6GHz CPU, 3.5GB RAM	\$0.12/hr
L	4 x 1.6GHz CPU, 7GB RAM	\$0.24/hr
XL	8 x 1.6GHz CPU, 14GB RAM	\$0.48/hr
A6	4 x 1.6GHz CPU, 28GB RAM	\$0.82/hr
A7	8 x 1.6GHz CPU, 56GB RAM	\$1.64/hr

Data storage is also billed by the Cloud service providers. In our model, monthly storage price is used. During our experiments, the data storage price was constant for all the queries. But for design that use materialized views an additional storage cost needs to be added. The detailed information of Database storage prices can be seen in Table 4.5.

Most of the Cloud providers do not charge for the data transfers in a private Cloud but the data that leaves the Cloud. In order to make our problem more interesting and

Table 4.5: Cloud database storage prices

Database Size	**Price**
100MB	\$5.00/mo
1GB	\$9.99/mo
2GB	\$13.99/mo
5GB	\$25.98/mo
10GB	\$45.96/mo
50GB	\$125.88/mo
150GB	\$225.78/mo

Table 4.6: Network bandwidth prices

Bandwidth	**Price**
10Mbps	\$0.02/hr
20Mbps	\$0.04/hr
50Mbps	\$0.08/hr
100Mbps	\$0.1/hr
200Mbps	\$0.2/hr

handle this dimension of the optimization, we locate our VMs on a virtual switch. Different bandwidth networks can be chosen. The pricing we use for the network layer is given in Table 4.6. The bandwidth of the network is increased from 10 Mbps up to 200 Mbps during the experiments.

4.3 Multiobjective Cloud Data warehouse Design with Branch-and-Bound Algorithm (MOD-B&B)

Multiobjective Cloud DW Design Branch-and-bound Algorithm (MOD-B&B) is an exhaustive optimization algorithm. It enumerates all candidate solutions, where fruitless subsets of candidates are discarded, by using upper and lower estimated bounds of the problem instance being optimized [145]. MOD-B&B starts searching with *null* initial values indicating that no query has yet been executed with the current VM and location of Tables configuration. Later, queries are added to current selected configuration. At each level of the tree, one additional query is assigned to the query workload [16]. This procedure is repeated for every VM and Tables configuration.

We define two initial upper bounds for MOD-B&B algorithm. The minimum monetary cost is the running time of the cheapest VM configurations that execute the queries in a workload. The response time is the finishing time of the workload with the given VM configuration. In order to estimate a lower bound different heuristic functions can be used. The heuristic we proposed here is reasonable and performs well during the optimization process. We will explain the heuristic with a scenario. In Figure 4.1 we can see the results of a sample multiobjective query workload optimization. The best response time and the minimum monetary cost values are defined and marked on the Figure. We can obtain these values with the most expensive and the cheapest VM configurations easily. Hereby, we propose a heuristic value (marked as *Heuristic point* on the Figure) that is the center of the square constructed by the response time and monetary costs of the most expensive and the cheapest VM configurations. If the response time of a workload falls above heuristic point or if the monetary cost is at the right-hand side of heuristic point on the Figure then it is pruned according to our heuristic.

Pseudocode of our MOD-B&B algorithm is given in Algorithm 4.1.

Algorithm 4.1: Multiobjective Branch-and-Bound (MOD-B&B) Algorithm for designing a Cloud DW

```
Input: Set of Virtual Resource Configurations with location of Tables ($VMC_1$ ,...,
       $VMC_n$), Queries ($Q_1$ ,..., $Q_m$), Set of Tables in the DW ($T$)
Output: Set of pareto-optimal solutions ($S$)

$Q$: Query workload;
$Q_{cur}$: Current query workload;
Calculated_value : Multiobjective cost of a database design

$S$={}; // Solution set is empty at the beginning
Heuristic_point ← Calculate_value (MostExpensiveVM_Point, CheapestVM_Point)

/* for all VMC and location of Tables configurations */
for (i=1 to n) do
    $Q_{cur}$=null;
    /* for all the queries in the workload ($Q$) */
    for ( j=1 to m) do
        $Q_{cur} = Q_{cur} \cup Q_j$; // $Q_j$ is the best plan of the query.
        Calculated_value= Calculate_the_Cost_of_Design_with ($VMC_i$, $Q_{cur}$, $T$);
        /* Bounding */
        if (Calculated_value is worse than Heuristic_point) then
            Break and start with the next virtual resource configuration $VMC_{i+1}$ ;
        /* Adding solution to the pareto-optimal set after consuming all queries */
        if (j == m and Calculated_value is better than Heuristic_point ) then
            $S := S \cup (VMC_i, Q_{cur}, T)$;

return $S$;
```

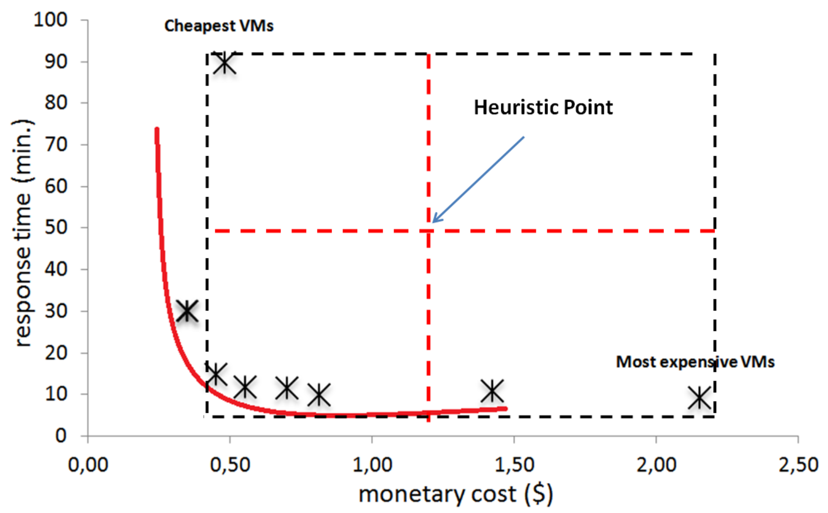


Figure 4.1: Proposed heuristic point for MOD-B&B algorithm (total monetary cost vs query workload response time).

4.4 Multiobjective Cloud Data Warehouse Design with Genetic Algorithm (MOD-GA)

The principles of applying natural evolution to optimization problems were first described by Holland [80]. The GA theory has been further developed and GAs have become very powerful tools for solving search and optimization problems [160]. GAs are based on the principle of genetics and evolution and have been frequently used to solve many NP-Complete problems. GAs use a computational model that simulates the natural processes of selection and evolution. Individuals with better quality have more chance to survive, to reproduce, and to pass their genetic characteristics to future generations. Each potential solution in the search space is considered as an individual and is represented by strings called chromosomes. Genes are the atomic parts of chromosomes and codify a specific characteristic. Chromosomes are encoded in different ways for each application. A random population is generated in the first step of the algorithm and by applying selection, crossover, and mutation operations iteratively, new generations are created [64]. The individual having the best fitness value in the population is returned as the solution of the problem. Algorithm 4.2 gives the details of MOD-GA.

Our multiobjective data Cloud warehouse design problem can be modeled by using evolutionary methods. A chromosome corresponds to a solution instance including

Algorithm 4.2: Multiobjective Genetic Algorithm (MOD-GA) for designing a Cloud DW

```
Input: Set of Virtual Resource Configurations (VMC), Set of Queries Q,
Set of Tables in the DW (T)
Output: Set of pareto-optimal solutions (S)

Best_response_time : The fastest response time of queries
Cheapest_cost : The cheapest virtual resource configuration
p: Population
$par_1$, $par_2$: Individuals (parents) selected for crossover or mutation processes

p ← Generate random individuals (VMC, Q, T)
p ← Truncate (p)
Best_response_time ← Find_best_response_time (VMC, Q, T)
Cheapest_cost ← Find_cheapest_cost (VMC, Q, T)

/* Generations start here */
for (i:=1 to generations) do
    for (j:=1 to size of population (p)) do
        ($par_1$, $par_2$)← Select parents (p)
        p ← Crossover ($par_1$, $par_2$)
        $par_1$← Select an individual (p)
        p ← Mutation ($par_1$)
    Replace_duplicate_chromosomes (p)
    Update Best_response_time
    Update Cheapest_cost
/* population is the pareto-optimal set of solutions */
S=p;
return S;
```

a set of VMs (having different CPU, RAM, and etc.) with tables/replications located on their databases, alternative query plans (QPs) of queries in the workload, and a network gene. Figure 4.2 shows the chromosome structure of a solution. The left-most segment represents the configuration of the VMs that the tables locate. Middle

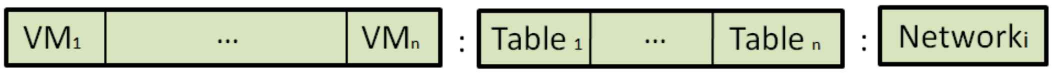
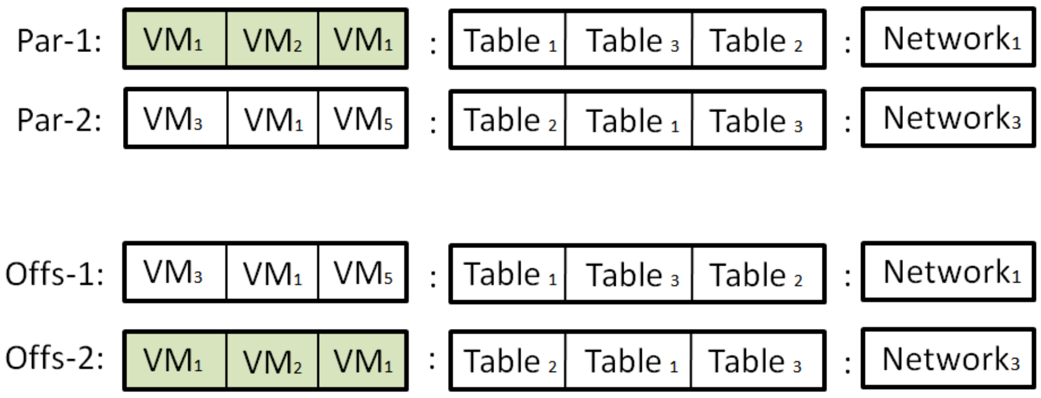


Figure 4.2: Chromosome structure for the proposed multiobjective genetic algorithm that consists of the Virtual Machines, Tables, and a network layer.

segment is the location of the tables on the VMs given in the first segment of the chromosome. Rightmost part gene represents the selected network layer of the solution vector. Because we do not make use of partitioned fragments of the relations, the size of the VM segment of the chromosome can be at most as many as the number of tables in the database. Using fragments of larger tables can even further improve the performance as it is used by most of the commercial data warehouses [83, 117]. Our proposed genetic model can be enhanced to include the fragments of the tables.

We define two operators, crossover and mutation, for the solution of MOD-GA model.

Crossover operator : uses two parents that are selected from the population by tournament selection method. Crossover operator swaps VM, Tables, or network part of two selected chromosomes with the same part of the other chromosome. In Figure 4.3, we can see two parents and their VM parts are exchanged to provide two new chromosomes.


Par-1: VM1 VM2 VM1 : Table 1 Table 3 Table 2 : Network1
Par-2: VM3 VM1 VM5 : Table 2 Table 1 Table 3 : Network3
Offs-1: VM3 VM1 VM5 : Table 1 Table 3 Table 2 : Network1
Offs-2: VM1 VM2 VM1 : Table 2 Table 1 Table 3 : Network3
Figure 4.3: Crossover operator for the multiobjective optimization of query workloads

Mutation operator : acts on VM and the location of Tables segments. In Figure 3.5, we can see how the second VM is replaced with a new type of VM.

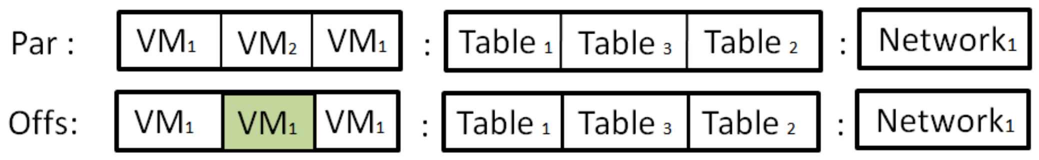


Figure 4.4: Mutation operator for virtual machines

In Figure 4.5, the mutation operator swaps the location of the tables on VM_1 and VM_2.

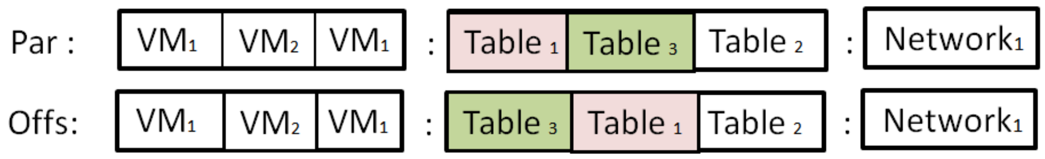


Figure 4.5: Mutation operator for location of tables

Fitness Calculation : Multiobjective fitness evaluation produces a set of solutions as many as the number of individuals in the population. The fitness of the individuals are evaluated in accordance with the Equation 4.7 and if the same values are found they are not included in the population. The response time of the query workload is evaluated by the DPACO query optimizer and the monetary cost is evaluated in terms of the consumed virtual resources.

Parameters of MOD-GA : The parameters used in the implementation of MOD-GA are:

- *Population Size:* Total number of chromosomes (individuals) in each generation.
- *Number of generations:* Each iteration of a GA that a number of crossovers and mutations are applied.
- *Selection Type (Tournament):* r chromosomes (r is the tournament size) are selected from the population, and the chromosome with the best fitness value is chosen for the next generation from the r-element group. This process is repeated as many times as the population size of the next generation.
- *Tournament Size:* Number of individuals entering a selection in tournament selection technique.

- *Truncate Ratio:* Ratio of the best individuals, which are sorted according to their fitness values, used for producing the next generation.
- *Mutation Ratio:* Probability of mutations in a single gene.

Selection Technique

Three different selection techniques can be used to determine the chromosomes to survive to the next generation. The simplest way to select a chromosome is giving a higher chance to better ones as it happens in nature. On the other hand, genetic operators may sometimes produce chromosomes with better fitness values by using unfit chromosomes. This provides a diversification for the generation process. We prefer tournament as our selection method from our previous experiences.

Tournament. A number of chromosomes are selected from the population. This number must be at least two and at most the same as the population size. Among these selected chromosomes, two best chromosomes are selected. With tournament selection technique, it is possible to choose the same chromosome several times.

Details of the experiments for parameter settings and performance evaluation of the MOD-GAs are presented below.

4.5 Parameter Settings for Multiobjective Genetic Algorithm

In this part, we present our result concerning the parameter settings of the MOD-GA. Population size and the number of generations of a genetic algorithm are the most important parameters that must be well tuned to obtain (near-)optimal solutions during the optimizations. Larger number of individuals and generations explore the search space more effectively. On the other hand, this may bring very long optimization times. In order to diminish the effect of this drawback, we perform some experiments with changing number of population sizes and generations. In Figure 5.5, we give the performance details of MOD-GA with different population sizes (10 to 100) and the number of generations (10 to 100) for workload-2 queries (10GB database). The Figure gives the average fitness value of populations during the generations up to 100. As it can be seen, populations that have small number of individuals converge earlier (stuck into local optima) and cannot improve the overall fitness value of the population. MOD-GA almost converges after 100 generations and continues to improve itself slightly after this point.

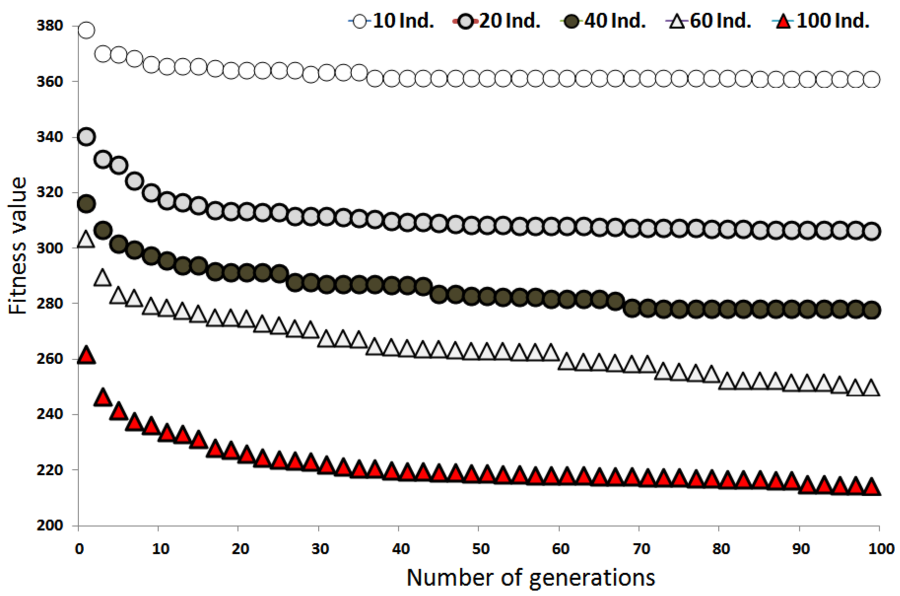


Figure 4.6: Average fitness value of populations on 10GB database and query workload-2.

Figure 4.7 gives the optimization times of MOD-GA with increasing number of populations. The optimization time of MOD-GA increases in accordance with the number of individuals in the population. For 10 individuals optimization time is 2 seconds and for 100 individuals it is 16 seconds. We select 100 individuals and 100 generations as our (near-)optimal parameters for the optimizations. These values provide good solutions for moderate size problems such as ours.

Table 4.7: Parameter settings for Multiobjective Genetic Algorithm (MOD-GA)

parameter	**value**
Population Size	100
Number of Generations	100
Selection Method	Tournament
Tournament Size	10
Mutation Rate	1%

Table 4.7 shows our parameter settings used for MOD-GA. We tested three different selection mechanisms (roulette wheel, truncation, and tournament) during the exper-

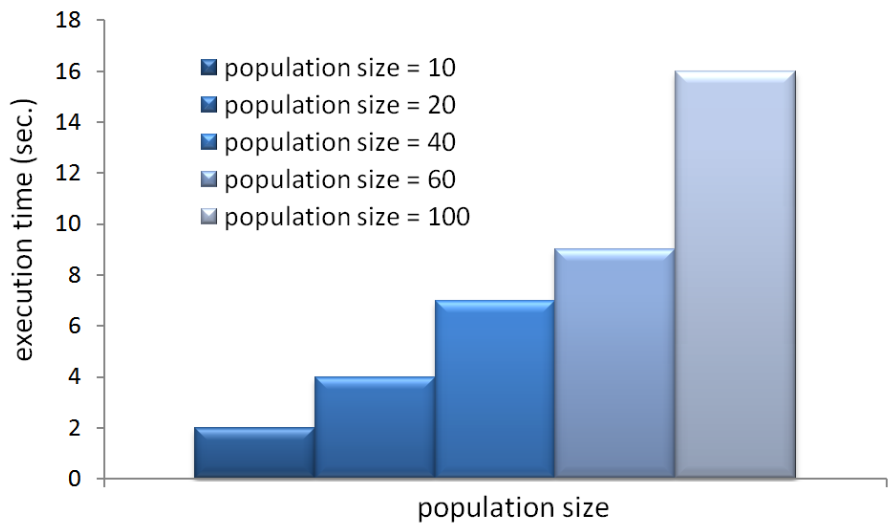


Figure 4.7: Multiobjective optimization times for increasing population sizes.

iments. Tournament is the selection mechanism that has performed well in our previous studies [47, 49]. We observe that roulette wheel selection technique needs more time than the other selection techniques because it requires sorting of the individuals and construction of the wheel from the sorted individuals. Truncation selection technique also needs sorting. On the other hand, tournament selection does not sort or construct any value during the selection process therefore, it is the fastest selection technique among the three selection techniques. There are not big differences between the solutions produced by the selection mechanisms. We select tournament selection as our main selection mechanism because of its simplicity and execution time during our experiments.

Figures 4.8 and 4.9 show how the solution quality of the individuals in a population improve with respect to the increasing number of generations. Initial population gives a picture of disorganized individuals but as the number of generations increases, the population tends to get closer to the hypothetical ideal point.

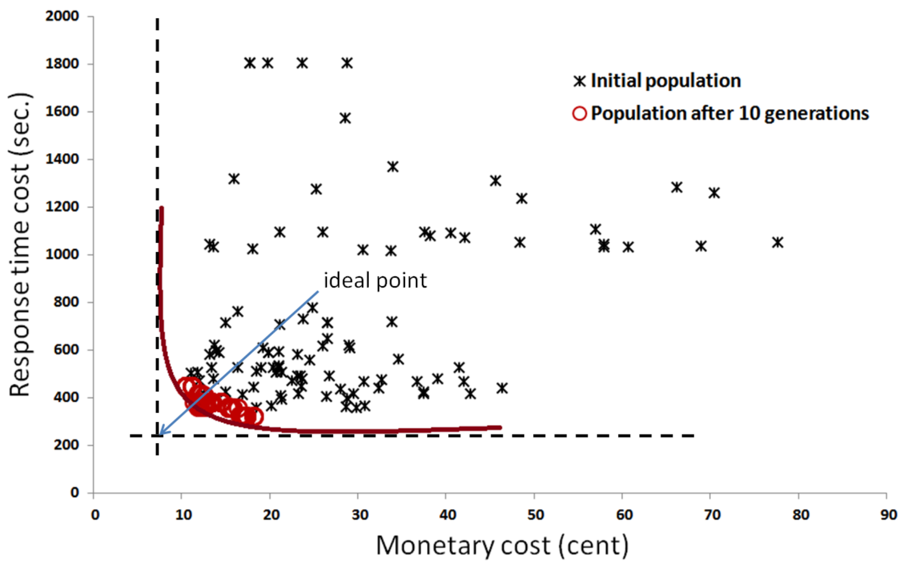


Figure 4.8: Distribution of solutions for initial population and after 10 generations.

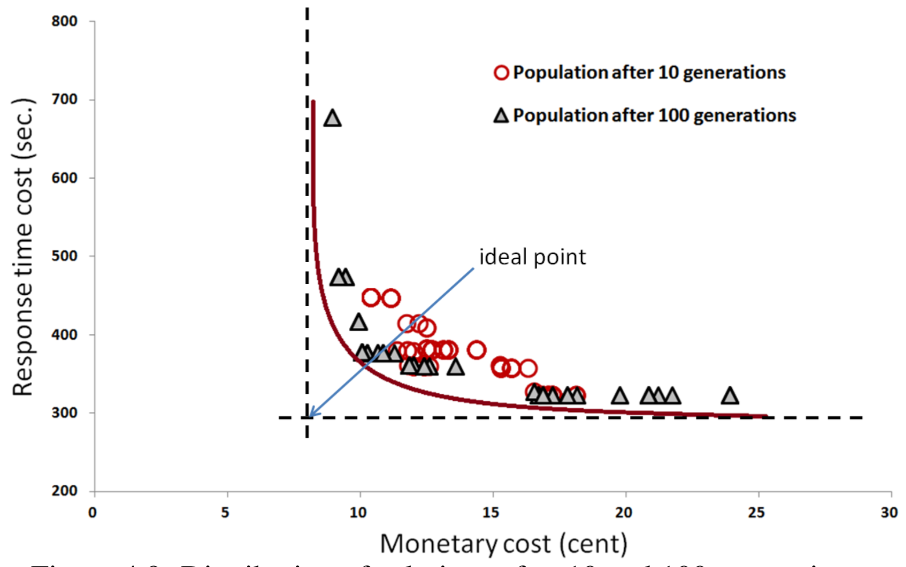


Figure 4.9: Distribution of solutions after 10 and 100 generations.

CHAPTER 5

MATERIALIZED VIEW SELECTION FOR THE MULTIOBJECTIVE OPTIMIZATION OF DATA WAREHOUSE QUERY WORKLOADS

Some complicated game had been playing up and down the hillside all the afternoon.

E. M. Forster[1]

In this Chapter, we define our approach for selecting materialized views on Cloud DW. We make use of a previously defined cost model developed in [177] and enhance its single objective cost model to a multiobjective cost model for materialized view selection on the Cloud. Materialized views are effective techniques for speeding up query workloads and they are increasingly being used by many commercial database systems. Materialized views are specially good for DWs because of the intensive usage of common subexpressions such as select-project-join operations. Our distributed DW integrates data from different relational databases and it is depicted as a relational OLAP (ROLAP) tool [158][76]. We show that selection of the appropriate materialized views remarkably reduces the communication cost, response time, and the ownership price of a relational Cloud DW.

The materialized view selection design problem is NP-hard [72] and there can be many alternatives to select a materialized view for a DW. For a distributed DW, the complexity of selecting the most appropriate materialized views gets even harder. Its complexity is due to selecting the most appropriate query subset, deciding the right VMs to keep the materialized view, and reducing the cost of communication between

[1] From his novel "A Room with a View", 1908

virtual machines.

Materialized view selection seems to be the same problem with the multiple query optimization (MQO). Both try to find the commonalities of the queries. However, MQO is the process of finding an optimal plan for a set of queries executing at the same time, whereas materialized view selection can serve for single queries executed at different times (if they have any commonalities). In MQO, total execution time is compared with the serial execution of queries. In materialized views, if a result is materialized an index can be created on it and remarkable performance gains can be obtained. If this subexpression can serve to more than a query then there is also a performance gain in view maintenance. MQO tries to achieve the best performance of a set of queries. Materialized views consider two objectives, query optimization and view maintenance.

5.1 Motivating Example

A query can be executed with several query plans (QPs) on a relational Cloud DW. The search space complexity of QPs gets larger with the number of sites, relations, fragments of the relations, replications, and with the type of the processing trees (left-deep, right-deep, zig-zag, and bushy tree). Existing query optimizers either use the best QPs of the queries or if they have support for MQO they try to merge the best QPs of the incoming queries. The best QPs are good for processing single queries however; alternative QPs can provide more sharable sub-expressions, which is an efficient way of executing query batches.

Query transformations : Join is the most expensive operation during the execution of the queries therefore, finding sharable join operations increases the performance of the system remarkably. In order to provide more sharable join operations, we pull up the selection, projection, and aggregate operations of the queries when materialized view processing plan (MVPP) is being generated. Later, we push down these operations back to their original locations on the query processing trees. If two queries share the same join operation, the select conditions of these two queries are the disjunction of the select conditions of the queries. Like selection condition, projection conditions (attributes) are unioned if they use the same join operation. If the group by conditions are the same for the queries, then all the aggregation functions are included in the global plan. If the group by operations are not the same, we use combinations of the group by attribute. Two alternative QPs whose selection predicates are pushed down and pulled up for TPC-H query Q3 can be seen in Figure 5.1.

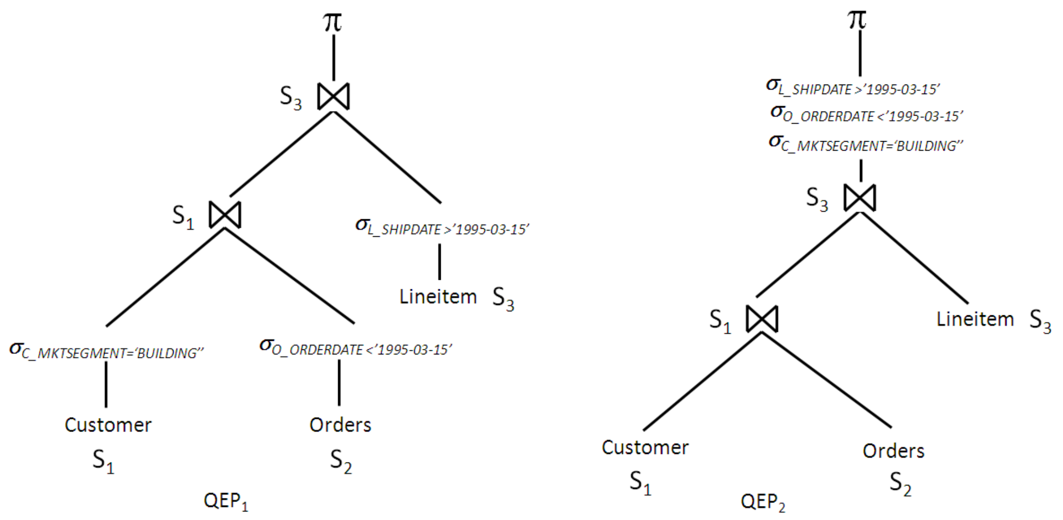


Figure 5.1: Alternative QPs for TPC-H Query3 (i) selections are pushed down, (ii)selections are pulled up.

If we give an example of how the QPs can exploit more of their subexpression, we can show the global plan of queries TPC-H Q3 and Q4 (see Figure 5.2). The chosen QP of Q3 shares more of its tasks with Q4. With this configuration of the TPC-H Q3 and Q4, the Q4 becomes the sub-expression of the Q3.

```
/*TPC-H Query 3*/

SELECT TOP 10 L_ORDERKEY, ,..., O_SHIPPRIORITY
FROM CUSTOMER, ORDERS, LINEITEM
WHERE C_MKTSEGMENT = 'BUILDING'
AND C_CUSTKEY = O_CUSTKEY
AND L_ORDERKEY = O_ORDERKEY
AND O_ORDERDATE < '1995-03-15'
AND L_SHIPDATE > '1995-03-15'
GROUP BY L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
ORDER BY REVENUE DESC, O_ORDERDATE;
```

Multiple view processing plan (MVPP) is one of the most important components of the materialized view selection problem. It can be depicted as a global query processing directed acyclic graph that combines the possible sub-queries together

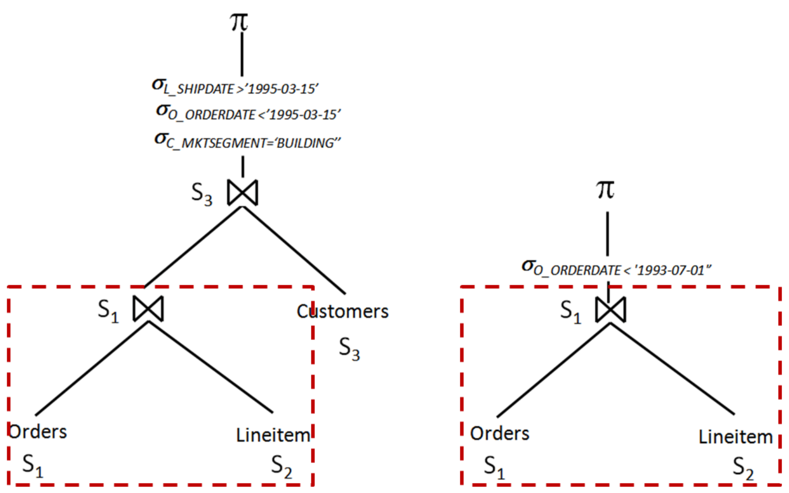


Figure 5.2: Query plans of TPC-Q3 and Q4 where they can share more of their subexpressions with pulled up selection predicates.

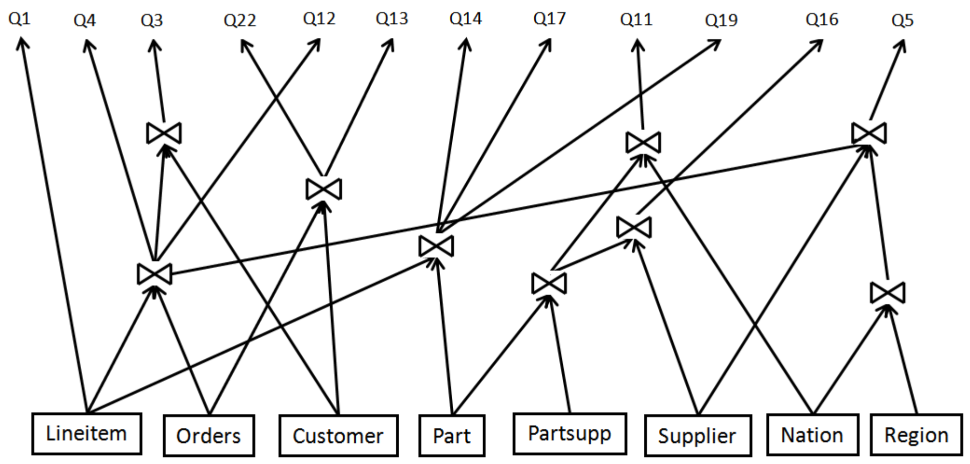


Figure 5.3: An example for materialized view processing plan that merges TPC-H queries.

[177]. It contains the query execution plan of the queries. For example Figure 5.3 can be considered as a simplified MVPP for a set of TPC-H queries. There can be several MVPPs in accordance with the alternative query execution plans of the queries. The challenge of materialized view selection problem is to construct an optimal MVPP and select the most efficient set of views to be materialized from this MVPP, which is an NP-hard problem [72].

Materializing of all the views in an MVPP can provide the best performance but it has the highest cost of view maintenance. Leaving all the views virtual gives the lowest view maintenance cost but the worst performance. We can select a set of views to be materialized and leave others as virtual views. In this way we may achieve a (near-)optimal trade-off between the performance gain and maintenance cost.

5.2 Materialized View Selection Formulation

Traditionally, the criteria to consider in the materialized view selection process mainly include view storage and maintenance costs. For Cloud databases, the optimization problem is multiobjective that the monetary cost must also be considered. The traditional materialized view selection problem can be formulated as :

Given a set of queries (Q) and a set of alternative query execution plans for each query (Q_i)=$\{p_{i1}, ..., p_{ik}\}$ where k is the number of alternative plans for (Q_i), select a set of views to be materialized with a global processing plan such that the total query and maintenance cost is minimized.

Although storing the selected materialized view has an additional cost, it is a very effective way of executing queries on the Cloud. Scan, computation, and communication sharing among concurrent queries in a database system always improve performance by eliminating redundant data accesses and computation.

Hereby, we formulate the generation of a multiobjective materialized view processing plan while optimizing the global monetary cost of storing and querying a database in the Cloud. Our problem formulation is an enhanced variant of the single objective formulation of the problem that is presented in [177][178].

A multiobjective multiple view processing plan (MVPP), M, can be generated depending on the formulation below:

$$M{=}(V, A, C^q_a, C^r_m, f_q, f_u)$$

where V is the set of vertices and A is the set of arcs over V.

- Create a vertex (v) for every base relation, every relational algebra operator (select and join) in a query tree, and every distinct query.
- For (v) $\in V$ $T(v)$ is the relation produced by the corresponding vertex v. $T(v)$ can be a relation at the leaf level or an intermediate result that is produced during the processing of a query.
- For any leaf v, $T(v)$ represents a base table. L is the set of leaf nodes.
- If a base relation or an intermediate result relation $T(u)$ corresponds to vertex u is needed for a process at node v, an arc $u \rightarrow v$ is introduced.
- $S(v)$ denotes the sources nodes that have edges pointed to vertex v. $S(v)$={} if v is a member of leaf nodes L. $S^*\{v\}$ is the set of descendants of v.
- $D(v)$ denotes the destination nodes to which v is pointed. For any $v \in R$, $D(v)$={}.
- $C^q_p(v)$ is the cost of query q that accesses to $T(v)$, $C^r_m(v)$ is the cost of maintaining $T(v)$ based on the changes to the base relation $S^*(v) \cap R$, if $T(v)$ is materialized.
- f_q and f_u the frequencies of query and maintenance respectively.

If M is the set of materialized views, $C_{q_i}(M)$ is the cost of computing q_i from the set of materialized views, M, and $C_m(v)$ is the cost of maintenance when v is materialized, then the total query execution cost is $\sum_{q_i \in Q} f_{q_i} C_{q_i}(M)$ and the total maintenance cost is $\sum_{v \in M} f_u C_M(v)$.

The cost model for distributed computation environments needs to take into account the communication cost of data transferring. If a query is submitted by node N_j and a view V_k is used to answer the query, the communication cost is zero if V_k is at the same node. Otherwise, the cost for transferring V_k from N_l to N_j is:

$CommunCost_{(V_k, N_l \rightarrow N_j)} = C_{N_j, N_l} \times \text{size}(V_k)$

where C_{N_j, N_l} is the network transmission cost per unit of data transferred between N_j and N_l and size(V_k) is the size of the view V_k [102] [34].

The total response time cost of the materialized view, M, is :

$$\text{Resp\_time}(M) = \sum_{q_i \in Q} f_{q_i} C_{q_i}(M) + \sum_{v \in M} f_u C_m(v)$$

Our objective is to minimize the multiobjective cost of the function :

$$F(M) = \{Resp\_time(M), Monetary\_cost(M)\}$$

where Monetary_cost(M) is the total monetary cost of materialized view processing plan M including the storage costs of the relations, intermediate join storages, network usage, and virtual machine consumptions.

During the generation of a materialized view processing plan (MVPP), we select a set of query plans whose selection and projection predicates are pulled up. The selected query plans are searched for the most expensive and the most common join operations that exist in the query set. Since this operation is a very expensive operation, we limit the number of the query plans of the queries (including the best query execution plans of the queries). Therefore finding a solution is realized in more reasonable times. Algorithm 5.1 shows the details of generating MVPPs from a set of queries. In our common subexpression detection method, we use a bottom-up model. The bushy-tree query execution plans are inspected whether they have similar join operations or not. Our main focus is on the join operations because they are the most time-consuming part of a query. Moreover, for a distributed database the costs increase in accordance with the bandwidth of the network. The detected common parts of the queries are merged into the MVPP. Individual queries mostly communicate the same set of tuples between each other. Efficient MVPPs prevent the redundant messages and packaging the tuples in the same bundle (preparing packages for a set tuples instead of a single tuple) provides an efficient way of executing query batches.

Algorithm 5.1: Generating a materialized view processing plan from a set of queries

```
Input: Queries ($Q_1$ ,..., $Q_m$)
Output: Materialized view processing plan ($M$)

$M$={};
for (i=1 to m) do
    Pull up select and project operations of query ($Q_i$);
for (i=1 to (m-1)) do
    for (j=i+1 to m) do
        if Queries have the same join operations ($Q_i$, $Q_j$) then
            Merge queries ($Q_i$, $Q_j$, $M$) ;
for (i=1 to m) do
    Push down select and project operations of query ($Q_i$);
return $M$;
```

5.3 Efficient Materialized View Maintenance on the Cloud

Materialized views provide fast access to the queried data. When views are complex, and the query rate is high, they can gain big advantages. But when the data is updated (gets dirty) at the source tables, the materialized views also need to be updated, which is called as materialized view maintenance [73]. Under heavy update workloads, materialized view maintenance can be a very expensive and time consuming process. The communication overhead in a Cloud data warehouse can even worsen this problem. Keeping the consistency with the resource tables can become very hard.

There are a number of studies concerning the materialized view maintenance in distributed (Cloud) databases [132]. The studies propose solutions ranging from a fully virtual approach where there is no materialized data and queries are responded with by using the resource tables to a full replication of data resources at the DW where the updates can be handled locally. These solutions are not efficient in terms of communication and data storage respectively. An efficient solution is to materialize the updated subsets of rows that are at the resource tables. The updates of the resource tables are propagated to the materialized views and the views are maintained incrementally.

Most of the time, materialized views are not computed from scratch. This way of updating views with only the dirty rows is called incremental materialized view. Incremental view materialization only computes updates to a view in response to updates to source tables [4]. If the intensity of the updates is now so high and the recomputation of the materialized views is not interfered, the views maintenance can be executed in a straightforward way. With this method, if an update is applied to the base relation, the incremental changes are computed by selecting the rows from the data sources.

If a view (V) is given over two relations R_1 and R_2 as $V = R_1 \bowtie R_2$ and ΔR_1 is the update of relation R_1 then the new materialized view can written as :

$(R_1 + \Delta R_2) \bowtie R_2 = (R_1 \bowtie R_2) + (\Delta R_1 \bowtie R_2)$

For the incremental maintenance only the $(\Delta R_1 \bowtie R_2)$ part needs to be computed and incorporate the resulting tuples into the materialized view. If ΔR is an insert update, then it will have the effect of adding the new tuples to the materialized view. If ΔR is a delete operation, it will remove the related tuples from the materialized view. With this approach, the views can be maintained incrementally. The updates must be separated from each other far enough to complete the computation of the views. This separation is not always possible between updates. Therefore, maintenance with incremental updates can be a problem for DWs.

ECA [181] and Strobe [182] algorithms are proposed for the solution of the problem. In ECA, the number of data sources is limited to a single source data. Actually, the data source may store several base tables. If we give an illustrative example of a three-way join operation ($R_1 \bowtie R_2 \bowtie R_3$), when an update ΔR_1 comes, then query ($\Delta R_1 \bowtie R_2 \bowtie R_3$) needs to be computed. During this update, another update such as ΔR_2 may occur. Thus, as a result of theses two updates, the view should be like:

$(R_1 \bowtie \Delta R_1) \bowtie (R_2 \bowtie \Delta R_2) \bowtie R_3$

The answer to the incremental query includes the effects of ΔR_1 and ΔR_2 and the answer A_1 will be $(\Delta R_1 \bowtie R_2 \bowtie R_3) + (\Delta R_1 \bowtie \Delta R_2 \bowtie R_3)$. In the incremental answer to the concurrent update ΔR_2, $(\Delta R_1 \bowtie \Delta R_2 \bowtie R_3)$ is refereed as error term [4]. Answer

A_1 will not provide the consistency of the data after two updates. $(R_1 \bowtie \Delta R_2 \bowtie R_3)$ will be missing. The formulation of the incremental query as $(R_1 \bowtie \Delta R_2 \bowtie R_3)$ will be incorrect because the first update has partially incorporated the effects of update ΔR_2. ECA algorithm uses the 'compensation' notion to solve this problem. The solution can be formulated as :

$(R_1 \bowtie \Delta R_2 \bowtie R_3) - (\Delta R_1 \bowtie \Delta R_2 \bowtie R_3)$

With the above formula, an optimization for ECA is easy to identify. If there is a sequence of concurrent updates, $\Delta_1 \bowtie R_1, \Delta_2 \bowtie R_2, ..., \Delta_k \bowtie R_2$, occurring while executing the $(\Delta R_1 \bowtie R_2 \bowtie R_3)$, only an incremental update $\Delta_i \bowtie R_2$ is needed.

In the Strobe algorithm, a new method is proposed for maintaining the materialized views incrementally in an environment with several data sources. There are some assumptions made by the algorithm. The materialized views are assumed to include key attributes of each relation. If update is a delete operation, it is handled by inserting a delete marker as well as in an accumulated answer list for following incorporation in materialized views. If the update is insert, a new query is started in accordance with the definition of the view and executed at the sources. The Strobe also handles the duplicate data.

If we give an example of how Strobe works with the same problem given above. In $(R_1 \bowtie R_2 \bowtie R_3)$, consider both updates ΔR_1 and ΔR_2 concurrently that insert data. If ΔR_2 overlaps the $(\Delta R_1 \bowtie R_2 \bowtie R_3)$, unlike ECA, the second query does not satisfy offsetting the error in query 1. It is formulated as : $(R_1 \bowtie \Delta R_2 \bowtie R_3)$. The answers to the queries are:

$(\Delta R_1 \bowtie R_2 \bowtie R_3) + (\Delta R_1 \bowtie \Delta R_2 \bowtie R_3)$

$(R_1 \bowtie \Delta R_2 \bowtie R_3) + (\Delta R_1 \bowtie \Delta R_2 \bowtie R_3)$

respectively.

The result of $(\Delta R_1 \bowtie \Delta R_2 \bowtie R_3)$ is included twice in the materialized views. Therefore, suppressing the duplicates can be handled easily with Strobe algorithm. The problem of the Strobe is that it must wait for quiescence in the system, which means that all updates subside and the resulting answers are merged in the views. The materialized views are not updated until there is no period of quiescence in the system. Strobe provides strong consistency but not complete because it incorporates the effects of the updates collectively.

In order to get rid of the quiescence effect of Strobe algorithm, C-strobe is proposed [182]. In C-strobe, updates are handled at the DW before subsequent updates. An answer is to a given update is computed by dealing with all the sources. This answer may contain errors due to the concurrent updates.

For example, given an update ΔR_1 and multiple relation, the DW dispatches a query ($\Delta R_1 \bowtie R_2 \bowtie R_3 \bowtie ...$) to the data sources containing the relations $R_1, R_2, ...$ If there is no concurrent update, then the answer can be merged into the materialized view. If a concurrent update ΔR_2 happens, then the answer will have an error of ($\Delta R_1 \bowtie R_2 \bowtie R_3 \bowtie ...$). The DW send a query ($\Delta R_1 \bowtie \Delta R_2 \bowtie R_3 \bowtie ...$) to R_3 to subtract out the errors. If a current update occurs then a second order error term ($\Delta R_1 \bowtie \Delta R_2 \bowtie \Delta R_3 \bowtie ...$) happens. Another compensation query needs to be sent to the base relations.

SWEEP is an efficient way of maintaining the materialized views [4] that works with update at a time on the ware houses and constructs the changes in the view for that update.

In our Cloud DW problem definition, we assume a rarely updated DW. Therefore, the maintenance of the materialized views is done incrementally as the base relations are updated.

5.4 Evolutionary Algorithm for Materialized View Selection

In this part we explain our solution model for the materialized view selection for relational Cloud DWs. This model is proposed as an additional segment to MOD-GA algorithm we present in Chapter 4. The chromosome structure of the evolutionary model can be seen in Figure 5.4. The chromosome holds the views that are derived from the MVPP of the alternative query plans. The colored gene of the views part of the chromosome shows the materialized view, whereas the white one shows a virtual view.

The solution segment is added to the algorithm presented in Algorithm 4.2. At every generation of the optimization, the selection of materialized views are evaluated in accordance with the virtual resources. Every MVPP has a special set of views to consider therefore, if a crossover operator is applied to the views segment it is most likely that the newly generated views segment becomes an invalid solution. In order to prevent this drawback, only mutation operator is applied to the views segment of the chromosome. Mutation can provide efficient results when it is applied without using crossover [146]. The mutation operator changes the status of the view from a materialized view to a virtual view. If MVVP is replaced with another MVVP (in another meaning, if the query plans of the queries are changed), then view segment of the chromosome is reconstructed to produce a valid view segment. In this way we ensure that the solutions produced by the crossover/mutation operators are valid.

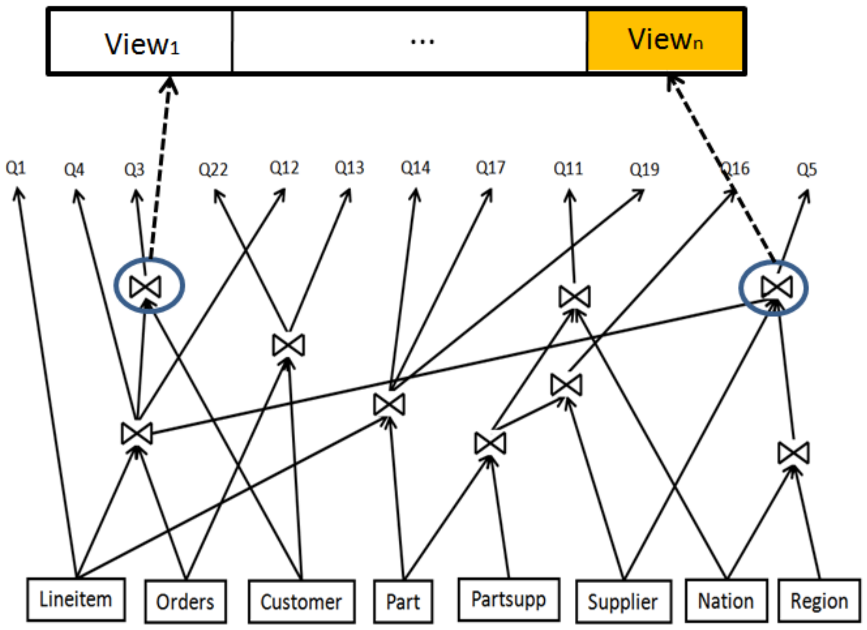


Figure 5.4: Chromosome segment for materialized views.

Fitness calculation : The fitness of a chromosome gives us the performance of a solution that is located a set of virtual resources and with selected materialized views. The fitness value has two dimensions as explained in our previous sections. The monetary cost and the response times of the query workload. The solutions that are closer to the hypothetical point (having the best response time and cheapest resources together) are kept in the population as proposed best solutions.

The solution space of the problem is huge. The varying elements in the solution space can be listed as: number of alternative virtual machines, selected network, number of data tables, alternative queries, number of materialized view processing plans, and views that can be materialized in MVPPs. Therefore, finding an exhaustive solution for this problem takes very long times (even not possible to obtain in months). With the robustness of the MOD-GA algorithm, we intend to obtain solutions that are very close (or the same) to the optimal solution set and improve the solutions in the pareto-optimal set by introducing materialized views.

5.5 Performance Evaluation of the Proposed Genetic Algorithm

In this section, we evaluate the performance of our GA that is proposed for selection of materialized views. Well-known TPC-H decision support benchmark database (10GB) is used in the experiments. Three different types of relational Cloud DW environments are prepared for the experiments. In the environments 1, 2, and 3 clusters have 6, 10, and 15 virtual machines respectively. Virtual machines have equal computing capabilities and have a DBMS that can access local data of other machines and execute distributed queries. It is possible to store the intermediate results in the main memory of the virtual machines and remove the least recently used cached query results [35].

In environment 1, each relation is located on a single virtual machine (except the replicated N and R). In the environment 2, relation L is horizontally fragmented to 5 additional virtual machines. In environment 3, the relations O and PS are horizontally fragmented and allocated to 4, 3 additional virtual machines respectively. Partitioned relations are horizontally divided into equal size fragments. Small size relations R and N are replicated at every virtual machine. Table 5.1 gives the details of the relational Cloud DW environments.

Queries that are executed on the environments given above are randomly selected from standard 22 TPC-H queries. The average number of alternative query plan is 5, 20, and 50 for environments 1, 2, and 3 respectively. The average number of tasks for each alternative query plan is 10. Initially, all of the virtual machines are assumed to

Table 5.1: Environment settings for the relational Cloud DWs used in the experiments

type	# Virtual machines	Network bandwidth	# Altr. QPs
1	6	100Mbps	5
2	10	1Gbps	20
3	15	10Gbps	50

be idle and the assigned set of queries is processed from scratch (meaning that neither any task is assigned to any site previously nor there is any cached result in the main memories).

The algorithms we compare are Hill Climbing Algorithm (HCA) and Hybrid Genetic - Hill Climbing Algorithm (Hybrid-GHCA).

Hill Climbing Algorithm (HCA)

Hill climbing is an optimization algorithm that belongs to local search family [109]. It works iteratively on an arbitrary solution and attempts to find a better solution by visiting the neighbors of the current solution. If a better solution is found, it becomes the current solution and this process continues until the termination condition is met. Hill climbing does not guarantee to find the best possible solution of all possible solutions. Instead, it gives a local optimum solution. The simplicity of hill climbing makes it a popular algorithm. Hill climbing is observed to produce better results than most of the algorithms when there is a time constraint. A sample hill climbing algorithm is given in Algorithm 5.2.

Algorithm 5.2: Simple Hill Climbing Algorithm, HCA

```
curr = initial solution;
while termination condition is not satisfied do
    new_solution = neigbour_of_(curr);
    if new_solution is better than curr then
        curr:= new_solution;
return curr;
```

In our study, we propose a simple Hill Climbing Algorithm, HCA, for the solution of the problem. HCA starts with a random initial solution and improves its quality by visiting its neigbours. There can be defined many different neigbourhood relations between solutions. We generate a new solution by replacing the query execution plan of each query with every possible plan.

For example, if a solution for a set of four queries (each having three alternative plans) is represented with the vector below.

$curr = \{p_{1,1}, p_{2,1}, p_{3,1}, p_{4,1}\}$

then, the neighboring solutions are produced as :

$neigb_1 = \{\mathbf{p_{1,2}}, p_{2,1}, p_{3,1}, p_{4,1}\}$

$neigb_2 = \{\mathbf{p_{1,3}}, p_{2,1}, p_{3,1}, p_{4,1}\}$

$neigb_3 = \{p_{1,1}, \mathbf{p_{2,2}}, p_{3,1}, p_{4,1}\}$

.

.

$neigb_n = \{p_{1,1}, p_{2,1}, p_{3,1}, \mathbf{p_{4,3}}\}$

As it can be seen, every neighboring solution has only a single different alternative plan than the original solution. When a better solution is found, it is replaced with the current solution and the search process is restarted from scratch.

Hybrid Genetic - Hill Climbing Algorithm (Hybrid-GHCA)

Hybrid algorithms are known to benefit from the advantages of different heuristics. They can further improve the solution quality of NP-Hard problems that have a large search space. Therefore, we propose a hybrid algorithm, an extension of genetic algorithm that combines the advantages of evolutionary with Hill Climbing (called Hybrid-GHCA) [98] [157]. It prevents the premature convergence and improves the performance of both GA and HCA.

Hybrid-GHCA has two stages. At the first stage, GA is applied to the population later, hill climbing algorithm, HCA, is applied to each one of the individual in the population. The best performing individual is selected as the solution of the problem.

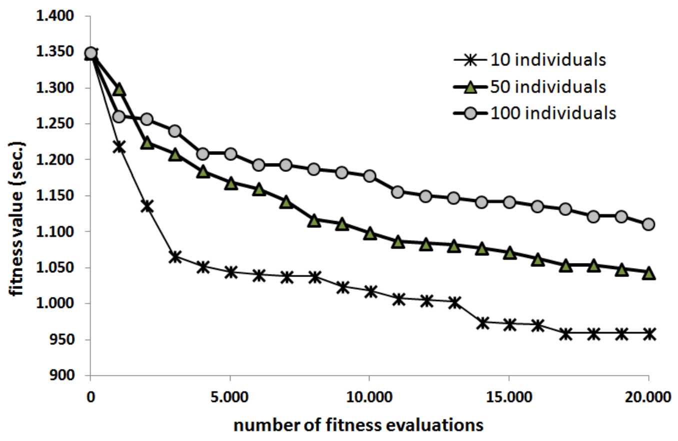


Figure 5.5: Population performance tests with 10, 50, and 100 individuals in accordance with the number of fitness evaluations (environment-2 with 50 queries).

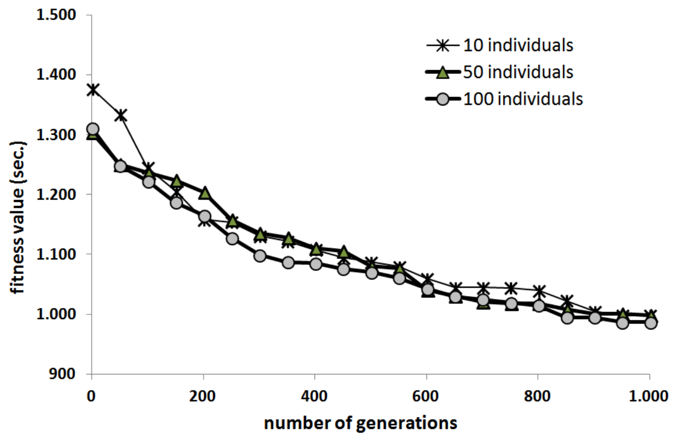


Figure 5.6: Population performance tests with 10, 50, and 100 individuals in accordance with the number of generations (environment-2 with 50 queries).

Table 5.2: Parameter settings for Genetic Algorithm (GA)

Parameter	**Value**
Population size	10
Number of generations (20,000 evaluations)	1,000
Maximum number of genes to transfer	50%
Minimum number of genes to transfer	10%
Mutation Ratio	1%

The population size is fixed at 10 and the optimization process of GA terminates after

1,000 generations (20,000 evaluations).

Comparison of the proposed algorithms

Before comparing the performance of the proposed algorithms, we optimized the number of individuals and the number of evaluations for GA. Figures 5.5 and 5.6 present the results of the experiments with 10, 50, 100 individuals in accordance with the number of chromosome fitness evaluations and generations respectively. In our GA, the number of evaluations increases in accordance with the number of individuals therefore, we present both of the Figures. The experiment is executed with a moderate size relational Cloud database environment, environment-2 (with 50 queries). Small size population with 10 individuals is observed to be the best performing population size. It has better performance than larger population sizes and terminates much earlier. Throughout the experiments, these parameter settings are used for GA (see Table 5.2).

Two criteria are used to evaluate the performance of the proposed algorithms, optimization times and the quality of the solutions.

Optimization times of the proposed algorithms

A heuristic search space analysis parameter can explain the complexity of the problem more clearly. If $|Q|$ is the number of queries, A is the average number of alternative query execution plans for queries, then the search space size, SS, can be defined as :

$$SS = A^{|Q|}$$

In our experimental environment, there are 7 queries that have 20 alternative query execution plans for each and the search space number is $SS = 20^7$ (nearly 1,5 billion).

Optimization time of the genetic algorithms depends on its termination condition. 1,000 generations (20,000 evaluations) is defined as the (near-)optimal solution for the proposed GA. HCA continues its optimization process until it cannot find any better solution (or gets stuck in a local optimum value). Hybrid-GHCA includes both of the optimization times of GA and HCA and because the starting solutions (individuals in the population) are improved with GA, the optimization time of Hybrid-GHCA takes less time than starting with random solutions.

In Figure 5.7, optimization times of GA, HCA, and Hybrid-GHCA can be seen for problem sets in environment-2 with increasing number of queries.

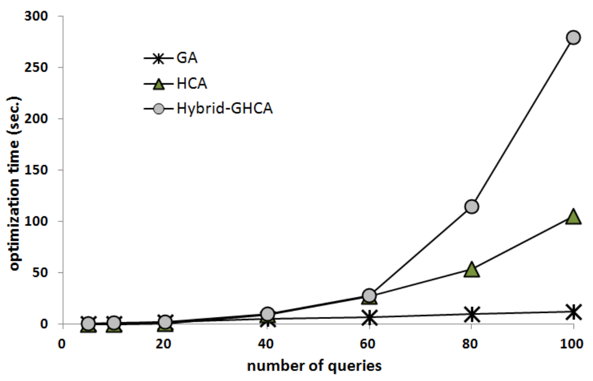


Figure 5.7: Optimization times of GA, HCA, and Hybrid GHCA algorithms (with increasing number of queries in environment-2).

Solution quality of the proposed algorithms

In order to observe the solution quality of the results that are obtained by the proposed algorithms, randomly generated TPC-H query batches are run on the relational Cloud database environments given in Table 5.1. The search space complexity of environment-1 is the smallest, the environment-3 has the largest complexity in accordance with the increasing number of queries in the query batch. Executed query batches include [5-100] queries.

Figures 5.8, 5.9, and 5.10 give the optimization results found by the proposed algorithms and a pure random research (that gives the best of 20,000 pure random solutions). For all the environments, Hybrid-GHCA algorithm is observed to find the best results. GA finds solutions that are mostly the same or slightly worse (0-5%) than the solutions of Hybrid-GHCA for problem sets with very large complexity. HCA is good and fast for small problems however; results found by HCA get worse when the search space is very complex. With respect to the optimization time and the quality of the solutions provided, GA algorithm is evaluated to be the best performing algorithm among the others.

Conventional relational databases use the best query execution plans of the queries

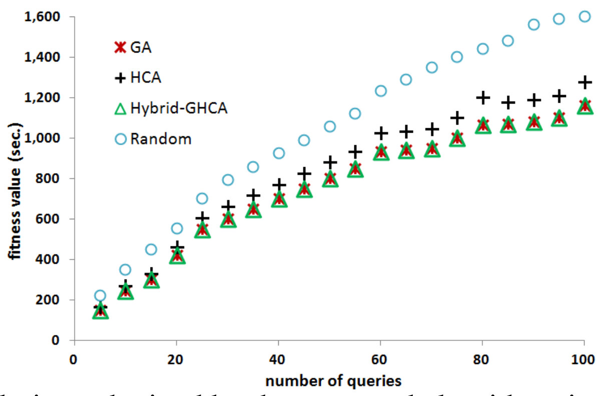


Figure 5.8: Solutions obtained by the proposed algorithms in environment-1.

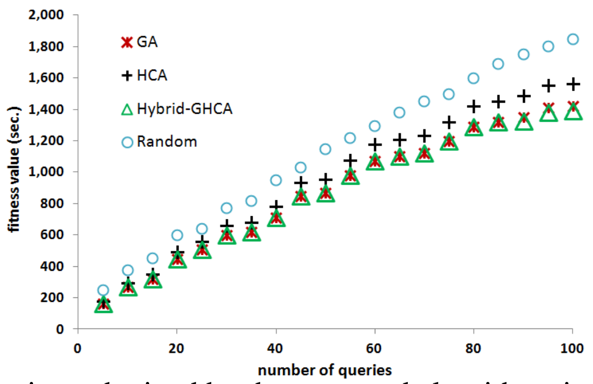


Figure 5.9: Solutions obtained by the proposed algorithms in environment-2.

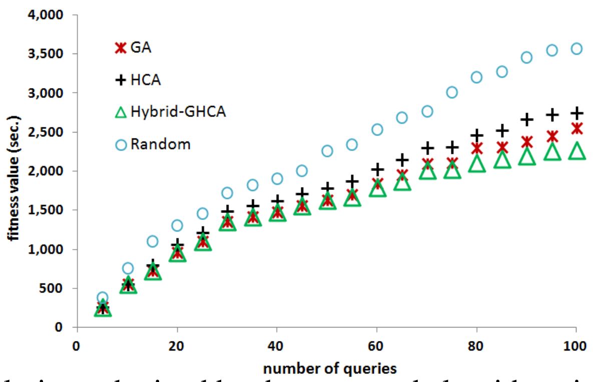


Figure 5.10: Solutions obtained by the proposed algorithms in environment-3.

when executing query batches. The architecture we propose searches for query execution plan sets that share more of their tasks. With the experiments that are performed on randomly selected set of [5-100] queries, 23%-55% improvements are observed.

CHAPTER 6

EXPERIMENTAL SETUP AND RESULTS

I've tried everything. I have not failed. I have just found 10,000 ways that won't work!

Thomas Edison

This Chapter presents the results of our experiments by using the proposed algorithms. The algorithms use the multiobjective query optimizer during the design of the relational Cloud DW. The obtained results of the algorithms are presented. The design options are mapped to a real Cloud environment to evaluate the actual performance of our algorithms and measure the efficiency of the schemes found by the proposed algorithms. We investigate the possibility of further improving the DW performance by using the selected materialized views.

Before giving the results, we describe our experimental environment, VMM Hyper-V, the TPC-H database, selected query workloads, and present the results of the experiments we obtain with Multiobjective Branch and Bound (MOD-B&B) and Multiobjective Genetic Algorithm (MOD-GA).

6.1 Experimental Environment

We perform our experiments on a private Cloud server, 4U DELL PowerEdge R910 having 32 (64 with Hyper Threading) cores and each core is Intel Xeon E7-4820 with a total of 2.00 Ghz processing power. Server has 128GB DDR3 1600 Mhz virtualized memory and Broadcom Nextreme II 5709 1Gbps NICs. Operating system is Windows Server 2012 Standard Edition and as guest operating systems Windows Server 2008 R2 SP2 Enterprise Edition are used and on top of guest operating system, SQL Server 2012 Enterprise Edition Service Pack 1 is implemented as the database server. Windows Hyper-V 3.0 is used as virtualization platform. Network page size is 4 KByte during the experiments. One of the the configuration DW infrastructure we use during the experiments is given Figure 6.1. The number of the VMs and the configurations (CPUs, main memory, and network bandwidth) are changed according to the proposed solutions.

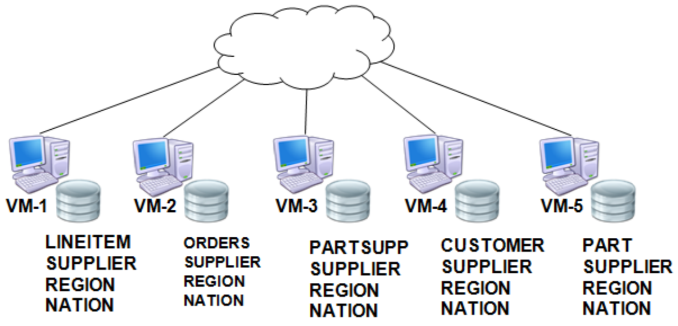
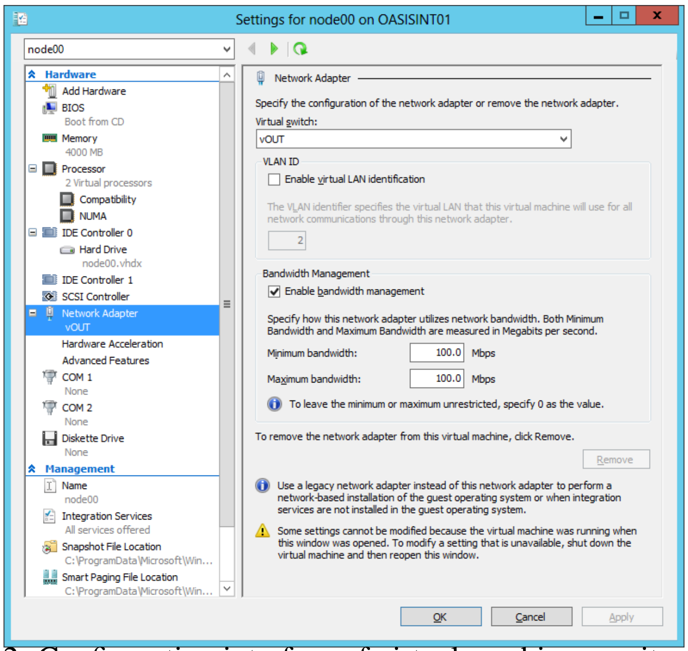


Figure 6.1: A sample Cloud DW architecture.

Windows Server Virtualization, Hyper-V, is a hypervisor that enables virtualization of x86-64 systems [164]. Figure 6.2 shows the management console of Hyper-V.


Settings for node00 on OASISINT01

node00

Hardware
- Add Hardware
- BIOS
  Boot from CD
- Memory
  4000 MB
- Processor
  2 Virtual processors
  - Compatibility
  - NUMA
- IDE Controller 0
  - Hard Drive
    node00.vhdx
- IDE Controller 1
- SCSI Controller
- Network Adapter
  vOUT
  - Hardware Acceleration
  - Advanced Features
- COM 1
  None
- COM 2
  None
- Diskette Drive
  None

Management
- Name
  node00
- Integration Services
  All services offered
- Snapshot File Location
  C:\ProgramData\Microsoft\Win...
- Smart Paging File Location
  C:\ProgramData\Microsoft\Win...

Network Adapter

Specify the configuration of the network adapter or remove the network adapter.

Virtual switch: vOUT

VLAN ID

☐ Enable virtual LAN identification

The VLAN identifier specifies the virtual LAN that this virtual machine will use for all network communications through this network adapter.

2

Bandwidth Management

☑ Enable bandwidth management

Specify how this network adapter utilizes network bandwidth. Both Minimum Bandwidth and Maximum Bandwidth are measured in Megabits per second.

Minimum bandwidth: 100.0 Mbps

Maximum bandwidth: 100.0 Mbps

To leave the minimum or maximum unrestricted, specify 0 as the value.

To remove the network adapter from this virtual machine, click Remove.

Remove

Use a legacy network adapter instead of this network adapter to perform a network-based installation of the guest operating system or when integration services are not installed in the guest operating system.

Some settings cannot be modified because the virtual machine was running when this window was opened. To modify a setting that is unavailable, shut down the virtual machine and then reopen this window.

OK Cancel Apply

Figure 6.2: Configuration interface of virtual machine monitor, Hyper-V.

6.2 TPC-H Data Warehouse and Query Workloads

TPC-H databases with 1GB, 10GB, and 25GB sizes are used in the experiments. The TPC-H database has 8 tables. The tables are **L**INEITEM (8,145MB;60 million rows), **O**RDERS (1,757MB;15 million rows), **P**ART**S**UPP (1,236MB;8 million rows), **P**ART (290MB;2 million rows), **C**USTOMER (256MB;1,5 million rows), **S**UPPLIER (15.5 MB;100,000 rows), **R**EGION (0,008MB;5 rows), and **N**ATION (0,008 MB;25 rows). The parameters inside the parentheses show the size and number of rows of a 10GB TPC-H database. We used replications of small tables, NATION, SUPPLIER, and REGION to obtain better performances from the proposed databases and decrease the complexity of the query plans. Tables of TPC-H database can be seen in Figure 6.3. Although TPC-H database is used in our experiments, our framework is generic that it can develop Cloud database design for any relational database.

We use three different workloads, workload-1, workload-2, and workload-3 for 1GB, 10GB, and 25GB databases respectively. Each workload consists of 10 to 14 different TPC-H queries that are given in Table 6.1. The selected queries (workloads) represent the most frequent set of queries of a database. We setup such an environment to test the proposed algorithms under diverse set of queries and database sizes in terms of the response time of query execution times and total cost of ownership. SQL statements are input as Directed Acyclic Graphs (DAG) into the system [176].

Table 6.1: Query workloads used in the experiments

Workload	**TPC-H Queries**
Workload-1 (10 queries)	1,2,3,4,5,6,7,8,9,10
Workload-2 (14 queries)	1,3,5,6,7,8,9,10,11,12,13,18,20,22
Workload-3 (10 queries)	2,3,4,9,10,11,12,13,18,22

In Figure 6.4, we present the response time of the selected TPC-H queries we use during the experiments. These response times are obtained with the most expensive configuration of VMs (XL) and the network bandwidths (200Mbps).

In order to show the changing needs of virtual machines in terms of CPU, RAM,

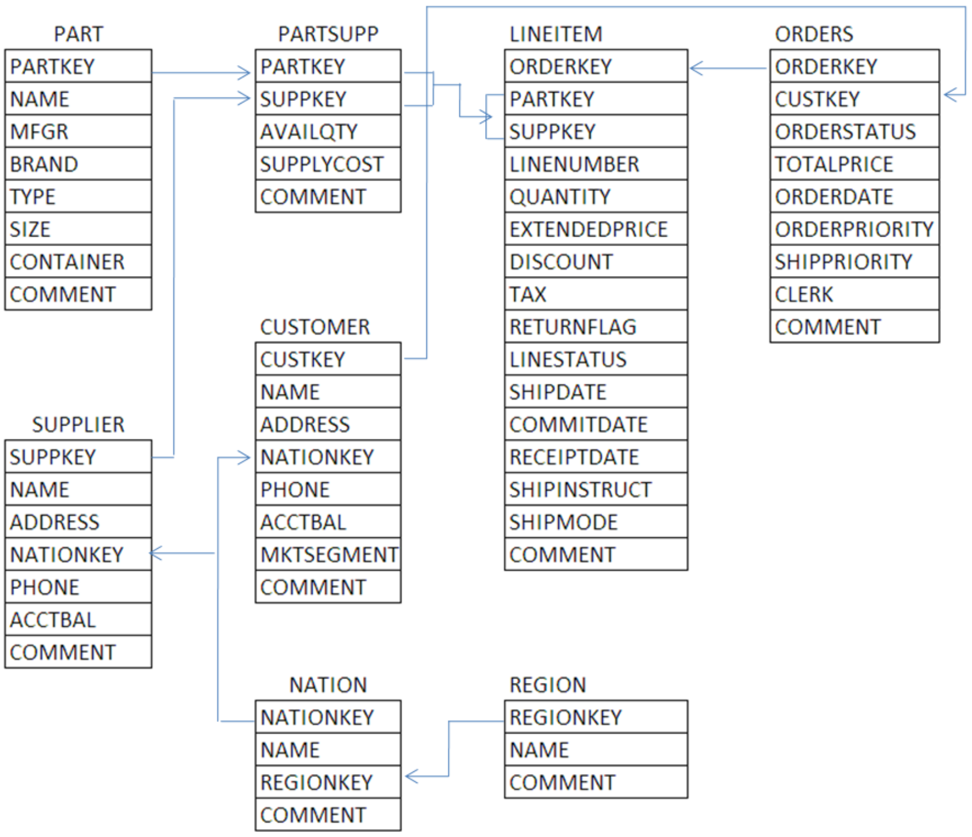


Figure 6.3: Tables of TPC-H database.

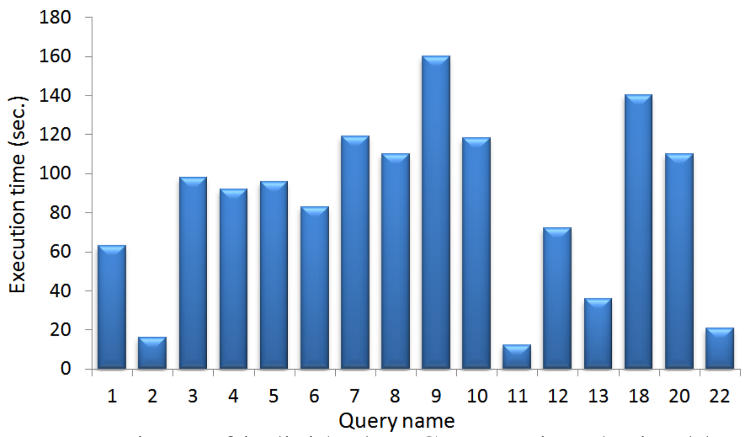


Figure 6.4: Response times of individual TPC-H queries obtained by query generator with 10GB database executed on 5 XL virtual machines and 200Mbps network bandwidth.

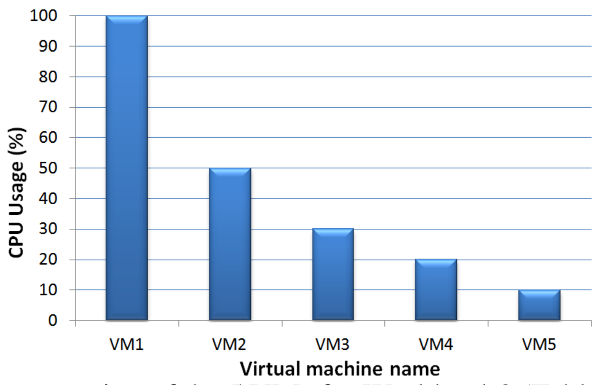


Figure 6.5: CPU consumption of the 5 VMs for Workload-2 (Tables Lineitem, Orders, Customer, Part, and PartSupp are located at virtual machines 1,2,3,4, and 5 respectively. Tables Supplier, Region, and Nation are replicated at each virtual machine).

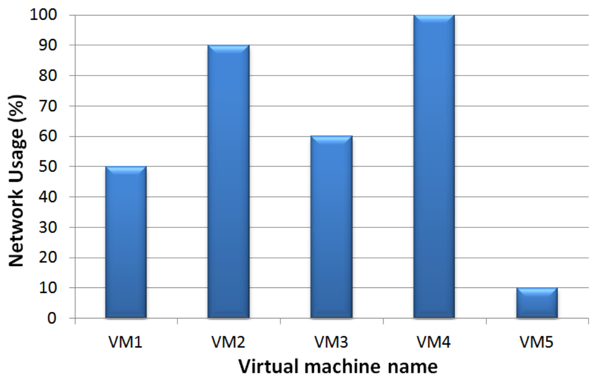


Figure 6.6: Network consumption of query workload in Figure 6.5

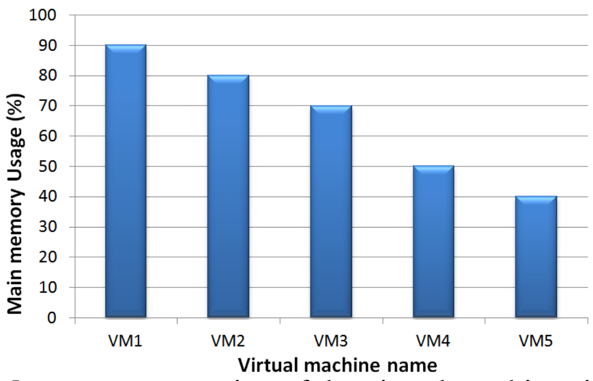


Figure 6.7: Memory consumption of the virtual machines in Figure 6.5.

and network bandwidth, we give the snapshots of the CPU, network, and memory consumptions of WMs during the execution of workload-1 with the most expensive virtual resources (5 XL virtual machines and 200Mbps network) (see Figures 6.5, 6.6, and 6.7). As it can be seen WM1 demands the largest CPU resource and memory usage. VM2 and VM4 ship the largest amount of data to the other VMs to execute the queries.

6.3 Comparison of the Proposed Data Warehouse Design Algorithms: MOD-B&B and MOD-GA Algorithms

In this part, we give results of our tests with MOD-GA and MOD-B&B algorithms on the selected TPC-H databases. MOD-B&B, a heuristic-based exhaustive algorithm and produces exact solutions. MOD-GA is a robust metaheuristic algorithm that finds (near-)optimal solutions in shorter optimization times. During the experiments, the optimization times of the MOD-GA and MOD-B&B algorithms are observed to be 16 and 3,640 seconds (nearly 1 hour) respectively for workload-2. The optimization time of MOD-B&B algorithm is reasonable for our problem that has moderate number of queries, tables, and VMs. However, it can be prohibitive for problems with larger search spaces. The optimization time of MOD-GA can be tuned depending on the users requirements. If the user wants high-quality solutions or if the size of the search space is very large then the optimization may take longer times.

The results of the solutions produced by the algorithms are presented in Figures 6.8, 6.9, and 6.10 for workload-1 (1GB), workload-2 (10GB), and workload-3 (25GB) respectively. After the optimization, we experienced some of the interesting solutions on our private Cloud environment. Tables 6.2, 6.3, and 6.4 present the real execution of the workloads. Since our optimization algorithms produce solutions depending on query-optimizers, the mapping of to the solutions to the Cloud may display some variations. During the experiments, we tried to minimize this variance by carefully handling the statistics of each query in the workloads. The selectivity of the queries, the execution time of the join operations, and the number of resulting tuples that will be sent to another virtual machine are meticulously calculated during the optimizations.

The solutions shown in the Tables are selected from the results that are produced by both of the algorithms. The column (VMs and Assigned Tables) in the Tables 6.2, 6.3, and 6.4 gives the types of the virtual machines and the tables that are located on them. For example, (M(O,C,PS,S,R,N); L(L,P,S,R,N)) shows two VMs, M(2 x 1.6GHz CPU, 3.5GB RAM) and L(4 x 1.6GHz CPU, 7GB RAM). Capital letter inside the parentheses give the first letters of the tables (L=LINEITEM; O=ORDERS; C=CUSTOMER; P=PART; PS=PARTSUPP; N=NATION; R=REGION; S: SUPPLIER). The abbreviations used in the Table are : Est.Resp.T.= Estimated Response Time; Est.Cost= Monetary Cost of Estimated Response Time; Fit.= Fitness value; Elap.T.= Time Elapsed in the Cloud; Elap.T.Cost= Monetary Cost of Elapsed Time on the Cloud

It was possible to obtain a pareto-optimal convex curve with the solutions produced by the algorithms. For small DW instances the usage of several virtual machines does not show big performance gains. However, as the size of the DW gets larger (it was possible for us to experience with TPC-H database sizes up to 25GB), additional VMs and broader bandwidth networks highly improve the performance of the system. 25GB DW response time was 1,423 seconds with a single XL virtual machine, whereas this response time is improved to 337 seconds (76.3% decrease in the execution) with an additional XL virtual machine, 200Mbps network, and well located tables. The monetary cost is also reduced by 55.9%.

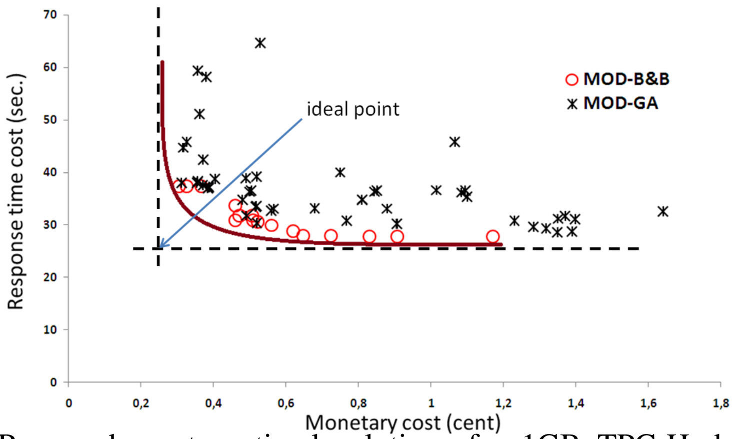


Figure 6.8: Proposed pareto-optimal solutions for 1GB TPC-H database query workload-1 by MOD-GA and MOD-B&B algorithms.

Table 6.2: Selected pareto-optimal solutions that are produced by MOD-GA and MOD-B&B Algorithms for 1GB TPC-H Database and Workload-1 (Est.Resp.T.= Estimated Response Time; Est.Cost= Monetary Cost ; Elap.T.= Time Elapsed in the Cloud)

Conf.#	VMs and Assigned Tables	Netw. Mbps	Est.Resp.T. (sec.)	Est.Cost (¢)	Fit.	Elap.T. (sec.)	Elap.T.Cost (¢)
1	XL (L,C,O,P,PS,S,R,N)	-	33.7	**0.46**	22.7	25	0.34
2	M(O,C,PS,S,R,N) L (L,P,S,R,N)	100	39.28	0.52	41.4	34	0.45
3	XS(O,L,C,S,R,N) XL (PS,P,S,R,N)	10	30.87	**0.46**	**15.3**	**21**	**0.3**1

4	XS(C,P,S,R,N)
	XL (L,PS,S,R,N)
	S(O,S,R,N)
		100	40.1	0.75	52.1	26	0.47

5	S(PS,P,S,R,N)
	M (O,L,S,R,N)
	S(C,S,R,N)
		20	64.7	0.53	124.1	42	0.34

6	L (L,P,O,S,R,N) XL (PS,S,R,N) XS (C,S,R,N) L (O,S,R,N)	100	35.4	1.10	62.5	47	1.46
7	L (L,S,R,N) XS (C,S,R,N) XS (O,S,R,N) S (P,S,R,N) XS (PS,S,R,N)	50	39.02	0.49	39.8	44	0.55
8	XL (L,S,R,N) XL (C,S,R,N) XS (O,S,R,N) XL (P,S,R,N) L (PS,S,R,N)	100	**32.64**	1.64	97.1	37	1.85

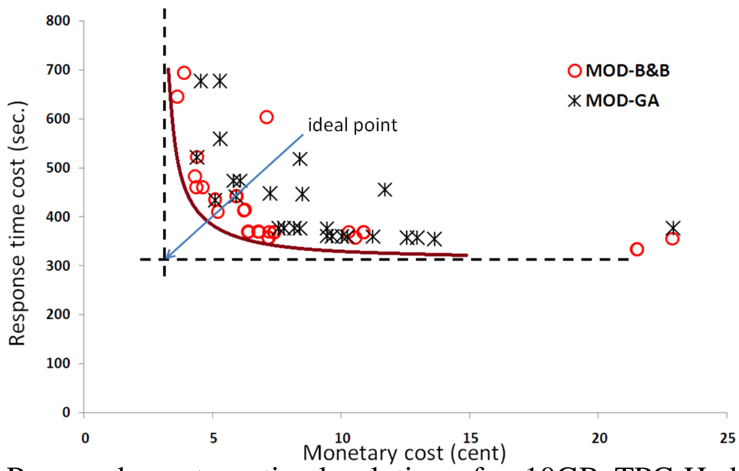


Figure 6.9: Proposed pareto-optimal solutions for 10GB TPC-H database query workload-2 by MOD-GA and MOD-B&B algorithms.

Table 6.3: Selected pareto-optimal solutions that are produced by MOD-GA and MOD-B&B algorithms for 10GB TPC-H database and query workload-2 (Est.Resp.T.= Estimated Response Time; Est.Cost= Monetary Cost ; Elap.T.= Time Elapsed in the Cloud)

Conf.#	VMs and Assigned Tables	Netw. Mbps	Est.Resp.T. (sec.)	Est.Cost (¢)	Fit.	Elap.T. (sec.)	Elap.T.Cost (¢)
1	XL (L,P,O,C,PS,S,R,N)	-	410	**6.2**	**38.3**	252	**3.8**
2	L (L,C,S,R,N) XS (PS,P,O,S,R,N)	100	604	7.1	83.3	430	5.1
3	XL (L,PS,P,S,R,N) S (C,O,S,R,N)	50	447	8.5	58.7	605	11.5
4	XL (L,P,S,R,N) XL (PS,S,R,N) XL (O,S,R,N) XL (C,S,R,N)	200	**378**	22.91	167.8	**245**	14.9
5	XL (L,S,R,N) XS (C,S,R,N) XS (O,S,R,N) XS (P,S,R,N) M (PS,S,R,N)	200	456	11.7	82.3	332	8.5
6	L (L,S,R,N) XS (C,S,R,N) XS (O,S,R,N) XS (P,S,R,N) M (PS,S,R,N)	100	519	8.4	70.9	360	5.8

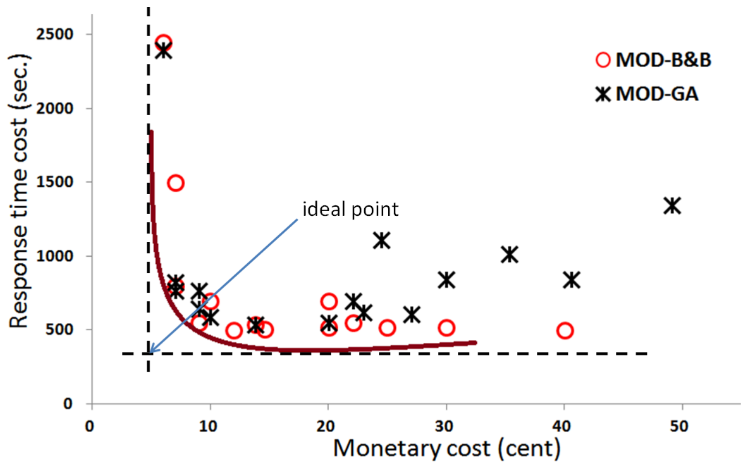


Figure 6.10: Proposed pareto-optimal solutions for 25GB TPC-H database query workload-3 by MOD-GA and MOD-B&B Algorithms.

Table 6.4: Selected pareto-optimal solutions that are produced by MOD-GA and MOD-B&B Algorithms for 25GB TPC-H database and query workload-3 (Est.Resp.T.= Estimated Response Time; Est.Cost= Monetary Cost ; Elap.T.= Time Elapsed in the Cloud)

Conf.#	VMs and Assigned Tables	Netw. Mbps	Est.Resp.T. (sec.)	Est.Cost (¢)	Fit.	Elap.T. (sec.)	Elap.T.Cost (¢)
1	XL (L,P,O,C,PS,S,R,N)	-	1,114	24.5	49.5	1,423	31.3
2	XL (L,O,C,S,R,N) XL (P,PS,S,R,N)	200	**552**	22.6	36.7	**337**	**13.8**
3	XL (L,O,S,R,N) M (PS,S,R,N) XS (P,S,R,N)	200	702	**22.1**	**36.4**	902	28.4
4	XL (O,PS,S,R,N) XL (L,S,R,N) L (P,S,R,N) M (C,S,R,N)	100	844	40.6	75.6	620	31.3
5	XL (L,O,S,R,N) M (PS,S,R,N) XS (P,S,R,N) M (C,S,R,N)	200	1,016	35.3	67.3	836	29.0
6	XL (L,S,R,N) L (C,S,R,N) M (O,S,R,N) XS (P,S,R,N) M (PS,S,R,N)	100	845	29.9	53.9	983	34.8
7	XL (L,S,R,N) L (C,S,R,N) M (O,S,R,N) S (P,S,R,N) XS (PS,S,R,N)	50	1,347	49.1	100.1	1,055	38.5

Figures 6.11, 6.12, and 6.13 show the percentage deviations of the evaluated designs with respect to the real Cloud DW designs. *x*-axis represents the execution times of real Cloud DW designs. The region above the *x*-axis is the over-estimation of the designed DW. The deviation of the estimations is observed to be between % 40 and % -40. The overestimation is due to the higher capabilities of SQL-Server database. It implements multiple query optimization and scan sharing during the execution of submitted queries. Underestimation of the performance is due our errors in estimating the join operations and data distributions.

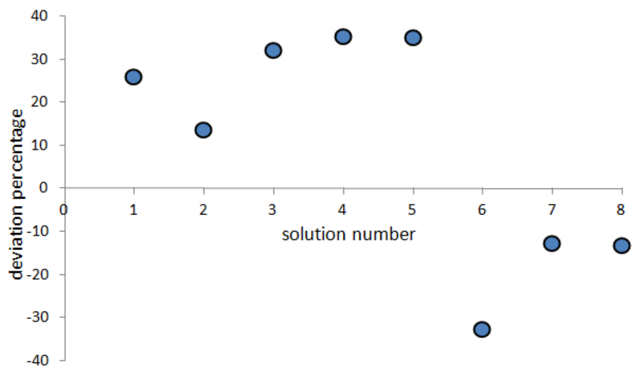


Figure 6.11: Deviations of the estimated designs with 1GB tph-h DW.

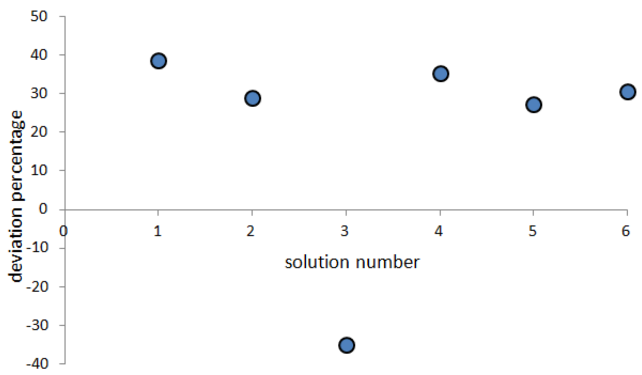


Figure 6.12: Deviations of the estimated designs with 10GB tph-h DW.

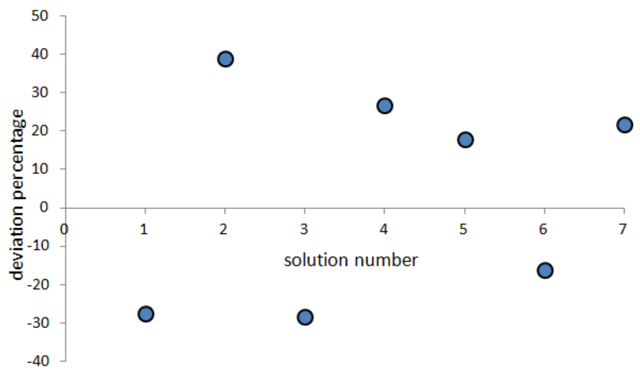


Figure 6.13: Deviations of the estimated designs with 25GB tph-h DW.

Table 6.5 gives the frequencies of the table and join usages of the query workloads. It can be observed that the VM with Lineitem table should be a powerful machine (such as XL). **L** and **O** tables should be located on the same machine or they have a broad bandwidth between both. If the tables of the mostly executed joins are located at separate VMs, then the network bandwidth between should be higher. These patterns can be seen in the proposed designs we obtained with our proposed algorithms.

Table 6.5: Table and join usage frequencies of the query workloads

Table/Join name (scale factor)	$W_1 - 1GB$	$W_2 - 10GB$	$W_3 - 25GB$
L (800)	**8**	**10**	**6**
O (175)	**7**	**10**	**8**
PS (125)	2	3	4
C (25)	5	**8**	3
P (29)	3	2	3
S (2)	5	6	4
N (1)	6	7	4
R (1)	2	1	1
L ⋈ O	**7**	**7**	**7**
C ⋈ O	2	**5**	**4**
L ⋈ P	2	2	2
L ⋈ PS	1	2	2
L ⋈ S	1	1	0
S ⋈ N	3	4	3
C ⋈ N	2	2	0
PS ⋈ P	1	0	2
N ⋈ R	3	2	2

6.4 Performance Improvements Using the Selected Optimal Materialized Views

In this part of the experiments, we analyze the efficiency of our materialized view selection algorithm in terms of quality of the set of proposed solutions. A single solution produced by the algorithm gives a Cloud DW that is designed by using virtual resources and materialized views. With the results we obtain, we intend to improve the quality of our solutions proposed for the design of the Cloud DWs.

First experiment we conducted was to observe the performance increase of a query Q5 for 10GB TPC-H DW. The query is executed on one of the pareto-optimal solutions proposed by MOD-GA algorithm. There are two virtual machines in the solution L(L,C,S,R,N) XS(PS,P,O,S,R,N). The location of the tables are given between the parentheses. The network bandwidth is 100Mbps.

Figure 6.14 shows a materialized view processing plan (MVPP) for TPC-H workload-2 that we have used during our simple experiment. In the Figure, some of the queries of workload-2 are merged together. As we can see tables LINEITEM and ORDERS can be joined together and serve many of the queries in this workload.

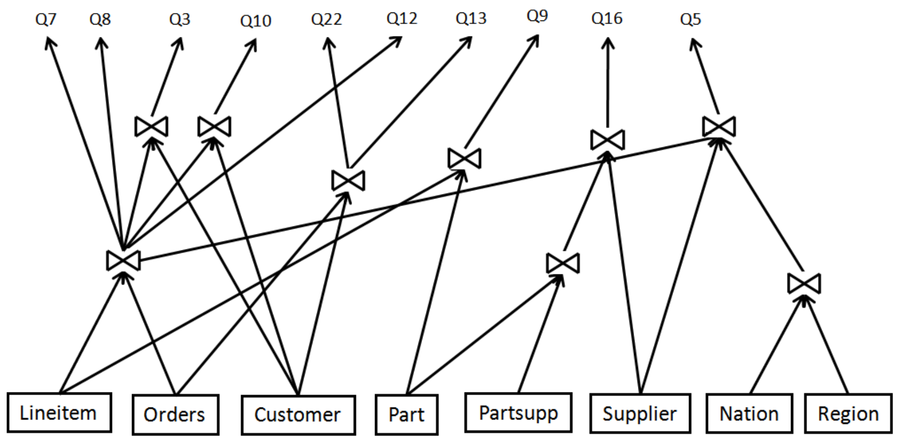


Figure 6.14: A selected Materialized View Processing Plan for query workload-2.

Materialized view for TPC-H Queries Q3, Q5, Q7, Q8, Q10, and Q12 is designed as below. All of the predicates of the queries are combined together with OR statements. The view is located on virtual machine 1. It does not need to communicate with the other virtual machines during the execution of the queries. Its size is 5.1GB. It takes about 300 seconds to build this materialized view. This causes an additional monetary cost that comes from the storage of the data however the performance increase of the queries is so remarkable that it pays back with its high-performance.

Materialized view, L_JOIN_O, for TPC-H Queries Q3, Q5, Q7, Q8, Q10, and Q12.

```
CREATE VIEW L_JOIN_O WITH SCHEMABINDING
AS
SELECT O_ORDERPRIORITY, L_ORDERKEY, L_LINENUMBER,
O_ORDERDATE, L_SHIPMODE, O_SHIPPRIORITY,
O_CUSTKEY, L_EXTENDEDNPRICE, L_DISCOUNT,
L_COMMITDATE, L_RECEIPTDATE, L_SHIPDATE,
L_SUPPKEY, L_RETURNFLAG
FROM LINEITEM, ORDERS
WHERE L_ORDERKEY=O_ORDERKEY
AND
( O_ORDERDATE ≥ ’1993-07-01’
OR O_ORDERDATE < ’1993-03-15’
OR L_SHIPDATE > ’1993-03-15’
OR O_ORDERDATE ≥ ’1993-10-01’
OR L_COMMITDATE < L_RECEIPTDATE
OR L_SHIPMODE IN (’MAIL’, ’SHIP’)
OR L_RETURNFLAG =’R’
OR L_SHIPDATE < L_COMMITDATE
OR L_RECEIPTDATE ≥ ’1994-10-01’)


CREATE UNIQUE CLUSTERED INDEX Idx_Myview_ClusteredIndex
ON DBO.O_JOIN_C(L_ORDERKEY, L_LINENUMBER)
```

Rewritten TPC-H Q5 query with the proposed L_JOIN_O materialized view as below.

```
SELECT N_NAME, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))
AS REVENUE
FROM L_JOIN_O, CUSTOMER, SUPPLIER, NATION, REGION
WHERE C_CUSTKEY = L_JOIN_O.O_CUSTKEY
AND L_JOIN_O.L_SUPPKEY = S_SUPPKEY
AND C_NATIONKEY = S_NATIONKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND R_NAME = ’ASIA’
AND L_JOIN_O.O_ORDERDATE >= ’1994-01-01’
AND L_JOIN_O.O_ORDERDATE < DATEADD(YY, 1, cast(’1994-01-
01’ as datetime))
GROUP BY N_NAME
ORDER BY REVENUE DESC;
```

When query Q5 is run without a materialized view, it takes about 111 seconds to complete the query with the given DW design. On the other hand, the same query is executed within 17 seconds by using the L_JOIN_O materialized view. This shows 84.6% improvement in the response time of the query.

When the queries Q3, Q5, and Q10 are executed together with materialized views the total response time is 28 seconds. The parallel query execution time of the queries is 139 seconds when executed without using the L_JOIN_O materialized view. This is a 79.8% improvement in the response time of the queries of Q3, Q5, and Q10.

In Table 6.6, we present the time spent for the creation of the materialized views, the size of the views, and the monetary costs. XL virtual machines are used while creating the materialized views.

Table 6.6: Time spent for the creation of selected materialized views and their monetary costs

Name of the mater. view	Time spent for creation(sec.)	Monetary cost (¢)	Size(MB)	Index size (MB)
L⋈O	345	4.6	5,116	10.7
O⋈C	34	0.45	193	0.32
L⋈P	215	2.86	1,246	2.6
P⋈PS	40	0.53	361	0.2
S⋈N⋈R	2	0.026	8	0.064
Total	636	8.46	6,924	15.68

At this phase of our experiments, we compare the results of the selected materialized views with the previous design solutions for 10GB TPC-H test data given in Table 6.3. The virtual machine settings given in the Table are taken as they are presented to show the performance/cost increase in workload-2 when materialized views are used. The selected materialized views and the location of these views are presented in Table 6.7. Location of the materialized views are given in the 'VMs, Tables, and Views' column.

From the results we obtained by using the materialized views, we observe that the response time of the workload-2 is improved 71.6% on the average for the designs presented in Table 6.7 and Figure 6.15. The monetary costs are improved 67.5%. These are remarkable results in accordance with both objectives that we aim to optimize.

The initialization and maintenance cost of the views can be seen as additional weights that need to be handled. This is true under heavy update workloads. If you have a not frequently updated data warehouse (write once, read many times) this will not be a big burden. Even with our small data warehouse we conclude that the materialized views pay their extra storage costs when the queries are submitted for the second time. As the storage costs continue to decrease for the Cloud, materialized views will become more efficient tools with their capability to decrease the communication between virtual machines.

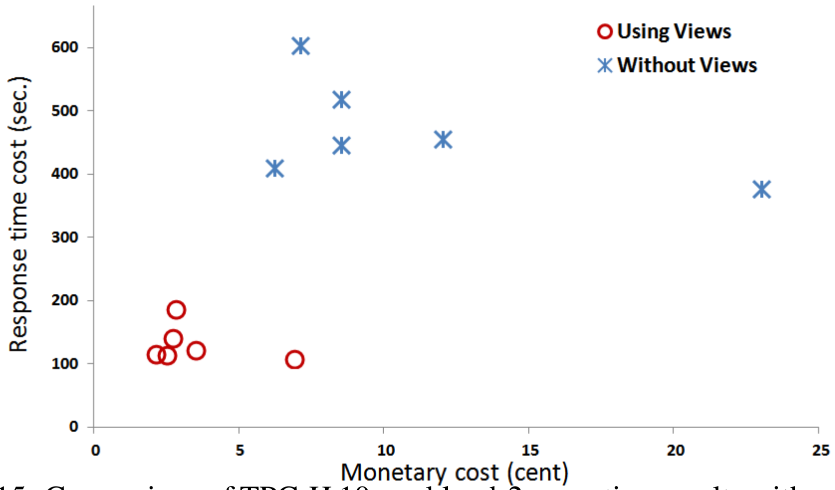


Figure 6.15: Comparison of TPC-H 10 workload-2 execution results with and without using materialized views.

Figure 6.16 presents the comparison of our results with randomly selected set of solutions. The randomly selected solutions consist of the largest number of best VMs that are as many as the number of tables, single VM with the cheapest and the most expensive configurations, and such solutions that are obtained with heuristic approaches. Workload 1, workload 2, and workload 3 can be executed with 35.5 %, 40.7 %, and 32.5 % improvements respectively.

Table 6.7: Comparison of the 10GB design results given in Table 6.3 with new DW designs implemented by using materialized views.

Conf.#	VMs, Tables, and Views	Netw. Mbps	Elap.T. (sec.)	Elap.T. Cost(¢)	V.Elap.T. (sec.)	V.Elap.T. Cost (¢)	Elap.T. Impr.(%)	Elap.T.Cost Impr.(%)
1	**XL**=(L,P,O,C,PS,S,R,N, L⋈O,O⋈C,L⋈P,P⋈PS,S⋈N⋈R)	-	410	**6.2**	117	**2.1**	71.4	66.1
2	**L**=(L,C,S,R,N, L⋈O,L⋈P,S⋈N⋈R) **XS**=(PS,P,O,S,R,N, O⋈C,P⋈PS)	100	604	7.1	188	2.8	66.9	60.5
3	**XL**=(L,PS,P,S,R,N, L⋈O,L⋈P,S⋈N⋈R) **S**=(C,O,S,R,N, O⋈C,P⋈PS)	50	447	8.5	115	2.5	**74.2**	**70.5**
4	**XL**=(L,P,S,R,N,L⋈P) **XL**=(PS,S,R,N,P⋈PS) **XL**=(O,S,R,N,O⋈C) **XL**=(C,S,R,N,S⋈N⋈R,L⋈O)	200	**378**	22.9	**109**	6.9	71.1	69.9
5	**XL**=(L,S,R,N,L⋈O) **XS**=(C,S,R,N,O⋈C) **XS**=(O,S,R,N,S⋈N⋈R) **XS**=(P,S,R,N,L⋈P) **M**=(PS,S,R,N,P⋈PS)	200	456	11.7	123	3.5	73.0	70.1
6	**L**=(L,S,R,N,L⋈O) **XS**=(C,S,R,N,O⋈C) **XS**=(O,S,R,N,S⋈N⋈R) **XS**=(P,S,R,N,L⋈P) **M**=(PS,S,R,N,P⋈PS)	100	519	8.4	141	2.7	72.8	67.9
Average			469	10.8	132.2	3.3	71.6	67.5

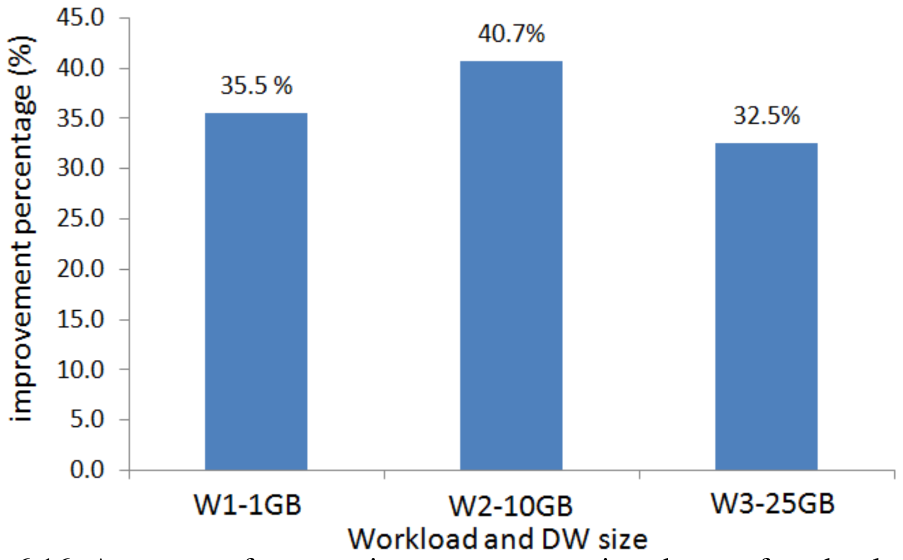


Figure 6.16: Average performance improvements against the set of randomly selected solutions.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

Today, we are living in a world that computer scientists have to deal with very large amounts of data. Users want to take advantage of these big data sources however; relational databases stored on single servers cannot meet the requirements of big data applications and must be prepared to work in distributed computation (Cloud) environments. The newly emerging database as a service paradigm is one of the best promising areas to meet this demand of the users. The Cloud database systems offer highly-scalable services with simplified interfaces and the goal of reducing the total cost of ownership. Users pay all costs associated with hosting and querying their data where database-as-a-service providers present different choices to trade-off price and performance to increase the satisfaction of the customers and optimize the overall performance.

Recently, extensive academic and commercial research is being done to construct self-tuned, efficient, and resource-economic Cloud database services that protect the benefits of both the customers and the vendors. Scalable, elastic, and fault-resilient shared-nothing relational databases with cheaper commodity hardware seem to be the most cost-efficient and dominating approach for such systems. Instead of using expensive and powerful servers, databases that are located on numerous commodity computers can be used to provide more cost-efficient systems.

7.1 Overview

The aim of the thesis was to design multiobjective (cost-efficient and high performance) efficient and scalable Cloud DWs. The classical database design techniques are intensively being used by the DBaaS providers. But this is not enough for the Cloud database customers that also demand for cost-efficient solutions, which means that they want fast response times in addition to the reduced monetary costs. Therefore, classical optimization goal (response time) must be combined with the monetary usage dimension of the Cloud resources. A design for a database that has the best response time is not an only objective for the Cloud DW. Instead, the data warehouses must be well designed to consume minimum amount of virtual resources in order to provide smaller monetary costs. This demand of the consumers introduces the problem of multiobjective design of relational Cloud databases. Within these expectations, optimization of the virtual resource use of the Cloud becomes a very important issue. It is not possible to provide these facilities on a single server when the frequency of the queries reach to a limit. The distributed computation environments need to provide scalability and elasticity without the notice of the users. The same performance of the database must be maintained in accordance with the expectations of the users no matter how large the data becomes. This is not an easy task. This thesis that approaches the problem from the perspective of the users provides a nice way that balances the resource consumption and the monetary issues of the Cloud while keeping the high performance of the database system.

7.2 Summary of the Results and Contributions

Our first contribution in this thesis was to design novel Cloud DW query optimization algorithms that provide alternative options to the needs of the query optimizers. We noticed that new state-of-the-art heuristic, Ant Colony Optimization has not been applied to this area until now and DPACO can provide efficient ways for the intractable Cloud DW query optimization problem.

We proposed a novel and efficient heuristic algorithm for the query optimization of relational Cloud DWs. The Dynamic Programming with Ant Colony Optimization

Algorithm (DPACO) optimizes multi-way chain join queries of the Cloud that operate on shared-nothing architectures [47, 49, 48]. DPACO is an extension of well-known commercial optimizer's DP algorithm and can be adapted to the existing systems easily.

In order to analyze this multiobjective problem, we selected the most used large database systems, DWs, as our test environment and developed a framework that makes use of the virtual Cloud resources and materialized views to design efficient multiobjective Cloud DWs. Our framework designs scalable relational OLAP (ROLAP) systems. Given the tables and the query workload of a DW, we allocate the data on (near-) optimal number of machines while considering the monetary cost and efficient response time of the queries. We proposed two algorithms for the design of the Cloud DW systems, Branch-and-Bound Algorithm (MOD-B&B) and Multiobjective Data Warehouse Design with Genetic Algorithm (MOD-GA). MOD-B&B is an exhaustive solution method with longer optimization times, whereas MOD-GA produces (near-) optimal solutions with reasonable optimization times. After having experienced with the algorithms we were able to verify the quality of our results on a private Cloud environment. The proposed Cloud DW systems have fast responding times and low-cost prices.

At the end of our study, we conclude that virtualization provides a scalable and elastic means to design multiobjective Cloud DWs. Traditional way of designing DWs on randomly selected virtual resources or most expensive hardware is not a good way. It may cause the customer to lose money or time while waiting longer times to take the results of the queries. With our proposed framework, we minimize the monetary cost as well as providing fast response times. We formulated the DW design problem and to the best of our knowledge, the multiobjective Cloud data warehouse design problem is being solved for the first time with such a method. There are studies that concern with the best virtual resource deployment or with the minimal monetary cost of workloads in a static hardware environment individually. However we combined both of these optimization techniques together with materialized views and obtained remarkable results as they are presented.

Materialized views are effective techniques for speeding up query workloads and reducing the prices. They are increasingly being used by many commercial DW. From the perspective of a Cloud customer, materialized views can provide cost-effective DWs. Because of the intensive usage of common subexpressions such as select-project-join operations, materialized views are specially good for DWs. Our distributed DW integrates data from different relational database resources and it can be depicted as a relational OLAP (ROLAP) tool. We show that selection of the appropriate materialized views remarkably reduces the communication cost, response time, and the ownership price of a relational Cloud database. From the results we obtained by using the materialized views, we observe that the response time of the workloads and the monetary costs are improved 71.6% and 67.5% on the average respectively. These are remarkable results in accordance with both of the objectives that we aim to optimize. Although the maintenance cost of the materialized views is an issue to control, this becomes trivial when the intensity of the queries is more than the updates.

Most of the recent studies are dynamic that they balance the virtual resource consumption by migrating the data between virtual machines. The size of the databases in these studies are not big (in MB scale). However, in our study, DWs handle large amount of data up to TB level. Therefore it is not wise and efficient to migrate this data at runtime. Instead, a well configured static design such as ours can be a more appropriate approach.

7.3 Future Work

Certainly, this thesis does not present a complete solution to all of the problems associated with efficient and multiobjective relational Cloud DWs. There are still a number of important issues to deal with. We can summarize some of these subjects as:

- Partitioning the big tables as horizontal/vertical fragments significantly improves the performance of the Cloud DWs. Finding scalable fragmentation algorithms for Cloud DWs can be a research area.

- Replicating small data tables at different virtual machines improves the performance of the Cloud DWs. In our study, we replicated small size tables with a heuristic approach. Designing a self-adaptive system that selects the most appropriate replication policy is a good area for research. The use of replication can be decided automatically.

- Designing adaptive query optimization algorithms that make use of the dynamic resources of the Cloud can be an interesting issue. Fault-resilient systems are always desired to support the recovery issues.

- A self-monitoring approach can be designed for Cloud DWs. The system can decide some of the optimizations in the run-time of the database. Meaning that the virtual resources can be adjusted according to the current workload of the database. Number of virtual machines, materialized selection, replication and other performance increasing issues can be designed at run-time. Tracking the nature of the queries (their frequency and resource demands) can be followed and intelligent decision can be made by using machine-learning techniques, such as reinforcement learning.

- This thesis focused on the cost-effectiveness and the response time issues of the Cloud DWs. Energy consumption (green computing) can be another objective for the research.

REFERENCES

[1] Abadi, D. (2012). Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. Computer, 45(2), 37-42.

[2] Aboulnaga, A., Amza, C., and Salem, K. (2008). Virtualization and databases: State of the art and research challenges. In Proceedings of the 11th international conference on Extending database technology: Advances in database technology (746-747).

[3] Agrawal, S., Chaudhuri, S., Das, A., and Narasayya, V. (2003). Automating layout of relational databases. ICDE (607-618).

[4] Agrawal, D., El Abbadi, A., Singh, A., and Yurek, T. (1997). Efficient view maintenance at data warehouses. In ACM SIGMOD Record (Vol. 26, No. 2, pp. 417-427).

[5] Ahmad, I., Karlapalem, K., Kwok, Y. K., and So, S. K. (2002). Evolutionary algorithms for allocating data in distributed database systems. Distributed and Parallel Databases, 11(1), 5-32.

[6] Amazon Elastic MapReduce. aws.amazon.com/elasticmapreduce (last accessed 5 September 2014).

[7] Amazon Web Services (AWS). aws.amazon.com (last accessed 5 September 2014).

[8] Amazon Redshift http://aws.amazon.com/redshift/ (last accessed 5 September 2014).

[9] Amazon Relational Database Service. aws.amazon.com/rds/ (last accessed 5 September 2014).

[10] Amol, D., Zachary, I., and Vijayshankar, R. (2007) Adaptive query processing. Foundations and Trends in Databases. 1(1):1-140.

[11] Andrade, H., Kurc, T., Sussman, A., and Saltz, J. (2002) Multiple query optimization for data analysis applications on clusters of SMPs. Proceedings of the 2nd IEEE/ACM Inter. Symposium on Cluster Computing and the Grid.

[12] Andrew S. Tanenbaum, and M. V. Steen. (2006). Distributed Systems: Principles and Paradigms. Prentice Hall, 2 edition.

[13] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... and Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50-58.

[14] Balazinska, M., Howe, B., and Suciu, D. (2011). Data markets in the cloud: An opportunity for the database community. PVLDB, 4(12).

[15] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., ... and Warfield, A. (2003). Xen and the art of virtualization. ACM SIGOPS Operating Systems Review, 37(5), 164-177.

[16] Bayir, M.A., Toroslu, I.H., and Cosar, A. (2007). Genetic Algorithm for the Multiple-Query Optimization Problem. IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews, Vol. 37 (1):147-153.

[17] Bennett, K., Ferris, M. C., and Ioannidis, Y. E. (1991). A genetic algorithm for database query optimization (400-407). Computer Sciences Department, University of Wisconsin, Center for Parallel Optimization.

[18] Berriman, G. B., Juve, G., Deelman, E., Regelson, M., and Plavchan, P. (2010). The application of cloud computing to astronomy: A study of cost and performance. In Sixth IEEE International Conference e-Science Workshops(1-7).

[19] Bernstein, P. A., and Chiu, D. M. W. (1981). Using semi-joins to solve relational queries. Journal of the ACM (JACM), 28(1), 25-40.

[20] Bernstein, P.A., Cseri, I, D, Nishant, E, Nigel, Kalhan, A., Kakivaya, G., Lomet, D.B., Manne, R., Novik, L., and Talius, T., (2011) Adapting microsoft SQL server for cloud computing. ICDE: 1255-1263.

[21] Berstein, P.A., Goodman, N., Wong, E., Reeve, C.L., and Rothnie, J.B. (1981). Query Processing in a System for Distributed Databases (SDD-1). ACM Trans. Database Syst., 6(4): 602-625.

[22] Beynon, M., Chang, C., Catalyurek, U., Kurc, T., Sussman, A., Andrade, H., Ferreira, R., and Saltz, J. (2002) Processing large-scale multi-dimensional data in parallel and distributed environments. Parallel Computing, 28(5), 827-859.

[23] Brewer, E. A. (2000). Towards robust distributed systems. In Proceedings of the Annual ACM Symposium on Principles of Distributed Computing (Vol. 19, 7-10).

[24] Bruno, N., Jain, S., and Zhou, J. (2013). Recurring Job Optimization for Massively Distributed Query Processing.

[25] Bruno, N., Galindo-Legaria, C., and Joshi, M. (2010). Polynomial heuristics for query optimization. In Data Engineering (ICDE) (pp. 589-600).

[26] Bruno, N., and Chaudhuri, S. (2002). Exploiting statistics on query expressions for optimization. In Proceedings of the 2002 ACM SIGMOD international conference on Management of data (263-274).

[27] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems, 25(6), 599-616.

[28] Ceri, S. and Pelagatti, G. (1984). Distributed databases principles and systems. McGraw-Hill, Inc..

[29] Chakravarthy, U. S., and Minker, J. (1986). Multiple query processing in deductive databases using query graphs. In Proceedings of the 12th International Conference on Very Large Data Bases (384-391).

[30] Chakravarthy, S. (1991). Divide and conquer: A basis for augmenting a conventional query optimizer with multiple query-processing capabilities. In Data Engineering. Proceedings. Seventh International Conference on (482-490). IEEE.

[31] Chaudhuri, S., and Dayal, U. (1997). An overview of data warehousing and OLAP technology. ACM Sigmod record, 26(1), 65-74.

[32] Chaudhuri, S. (1998). An overview of query optimization in relational systems. In Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (34-43).

[33] Chatziantoniou, D., and Tzortzakakis, E. (2009). Asset queries: a declarative alternative to mapreduce. ACM SIGMOD Record, 38(2), 35-41.

[34] Chaves, L. W. F., Buchmann, E., Hueske, F., and Böhm, K. (2009, March). Towards materialized view selection for distributed databases. In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (pp. 1088-1099). ACM.

[35] Chen, G., Wu, Y., Liu, J., Yang, G., and Zheng, W. (2011) Optimization of sub-query processing in distributed data integration systems. Jour. of Network and Computer Applications 34: 1035-1042.

[36] Chen, F. C. and Dunham, M. H. (1998). Common subexpression processing in multiple-query processing. Knowledge and Data Engineering, IEEE Transactions on, 10(3), 493-499.

[37] Comito, C., Gounaris, A., Sakellariou, R., and Talia, D. (2009). A service-oriented system for distributed data querying and integration on Grids. Future Generation Computer Systems, 25(5), 511-524.

[38] Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Gerth, J., Talbot, J., ... and Sears, R. (2010). Online aggregation and continuous query support in mapreduce. In Proceedings SIGMOD (1115-1118).

[39] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., ... and Woodford, D. (2012). Spanner: Google's Globally-Distributed Database.

[40] Curino, C., Jones, E., Popa, R., Malviya, N., Wu, E., Madden, S., Balakrishnan, H., and Zeldovich, N. (2011). Relational Cloud: A Database Service for the Cloud. CIDR, pp.235-240.

[41] Curino, C., Jones, E. P., Madden, S., and Balakrishnan, H. (2011). Workload-aware database monitoring and consolidation. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (pp. 313-324).

[42] Dalvi, N. N., Sanghai, S. K., Roy, P., and Sudarshan, S. (2003). Pipelining in multi-query optimization. Journal of Computer and System Sciences, 66(4), 728-762.

[43] Dash, D., Kantere, V., and Ailamaki, A. (2009). An economic model for self-tuned cloud caching. In Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on (pp. 1687-1693).

[44] Deelman, E., Singh, G., Livny, M., Berriman, B., and Good, J. (2008). The cost of doing science on the cloud: the montage example. In Proceedings of the 2008 ACM/IEEE conference on Supercomputing (p. 50).

[45] Deshpande, A., Ives, Z., and Raman, V. (2007). Adaptive query processing. Foundations and Trends in Databases, 1(1), 1-140.

[46] Diwan, A. A., Sudarshan, S., and Thomas, D. (2006). Scheduling and caching in multi-query optimization. COMAD.

[47] Dokeroglu, T. (2012). Parallel Genetic Algorithms for the Optimization of Multi-Way Chain Join Queries of Distributed Databases 38th VLDB Ph.D. Workshop, Istanbul/TURKEY.

[48] Dokeroglu, T., and Cosar, A. (2011). Dynamic Programming with Ant Colony Optimization Metaheuristic for Optimization of Distributed Database Queries. In Computer and Information Sciences II: 26th International Symposium on Computer and Information Sciences. 107-113. Springer.

[49] Dokeroglu, T., Tosun, U., and Cosar, A. (2012). Particle Swarm Intelligence as a Novel Heuristic for the Optimization of Distributed Database Queries, The 6th International Conference on Application of Information and Communication Technologies AICT2012 Georgia, Tbilisi.

[50] Dong, H., and Liang, Y. (2007). Genetic algorithms for large join query optimization. In Genetic And Evolutionary Computation Conference: Proceedings of the 9 th annual conference on Genetic and evolutionary computation (Vol. 7, No. 11, 1211-1218).

[51] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 26(1), 29-41.

[52] D'Orazio, L., Bimonte, S., and Darmont, J. (2012). Cost Models for View Materialization in the Cloud. In Proceedings of the Workshop on Data Analytics in the Cloud (EDBT-ICDT/DanaC).

[53] Elmore, A. J., Das, S., Agrawal, D., & El Abbadi, A. (2011). Zephyr: live migration in shared nothing databases for elastic cloud platforms. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (pp. 301-312). ACM.

[54] Epstein, R., Stonebraker, M., and Wong, E. (1978). Query Processing in a Distributed Relational Database System , In Proc. ACM SIGMOD Int. Conf. On Management of Data, pp. 169-180.

[55] Finkelstein, S. (1982) Common Expression Analysis in Database Applications. In Proc. SIGMOD, 235-245.

[56] Fiore S. and Aloisio, G. (2011). Grid and Cloud Database Management. Springer-Verlag, Berlin Heidelberg.

[57] Fontes, V., Schulze, B., and Dutra, M. (2004) CoDIMS-G: a data and program integration service for the grid. Proceedings of the 2nd workshop on Middleware for grid computing. Ontario, Canada,29-34.

[58] Fox, A., and Griffith, R. (2009). Above the clouds: A Berkeley view of cloud computing. Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Tech. Rep. UCB/EECS, 28.

[59] Galindo-Legaria, C., Pellenkoft, A., and Kersten, M., (1994). Fast, randomized join-order selection - why use transformations? Proceedings of Conf. Very Large Data Bases, Santiago, Chile,85-95.

[60] Garofalakis, M. N., and Ioannidis, Y. E. (1996). Multi-dimensional resource scheduling for parallel queries. In ACM SIGMOD Record (Vol. 25, No. 2,365-376).

[61] Gartner's 2008 Data Center Conference Instant Polling Results: Virtualization Summary - March 2, 2009.

[62] Gehrke, J. and Ramakrishnan, R. (2003). Database management systems. New York.

[63] Giannikis, G., Alonso, G., and Kossmann, D. (2012). SharedDB: Killing One Thousand Queries with One Stone. In Proc.VLDB, Vol. 5, No.6.

[64] Goldberg, D. (1989). Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading, Mass.

[65] Golshanara, L., Rankoohi, S. M. T. R., and Shah-Hosseini, H. (2013). A multi-colony ant algorithm for optimizing join queries in distributed database systems. Knowledge and Information Systems, 1-32.

[66] GoodData, general website http://www.gooddata.com/ (last accessed 5 September 2014).

[67] Google App Engine. http://code.google.com/appengine/ (last accessed 5 September 2014).

[68] Gounaris , A. (2005). Resource aware query processing on the grid. PhD thesis. University of Manchester.

[69] Goss, S., Aron, S., Deneubourg, J. L., and Pasteels, J. (1989). Self-organized shortcuts in the Argentine ant, Naturwissenschaften, vol.76,no.12, 579-581.

[70] Graefe, G. (1993) Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73-170.

[71] Gregory, M. (1998). Genetic algorithm optimisation of distributed database queries. In Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on (271-276).

[72] Gupta, H., and Mumick, I. S. (2005). Selection of views to materialize in a data warehouse. Knowledge and Data Engineering, IEEE Transactions on, 17(1), 24-43.

[73] Gupta, A., and Mumick, I. S. (1995). Maintenance of materialized views: Problems, techniques, and applications. IEEE Data Eng. Bull., 18(2), 3-18.

[74] Apache Hadoop, 2014, http://hadoop.apache.org/ (last accessed 5 September 2014).

[75] Halevy, A., Rajaraman, A., and Ordille, J. (2006) Data integration: the teenage year. Proceedings of the VLDB, Seoul, Korea, 2006.9-16.

[76] Halevy, A. Y. (2001). Answering queries using views: A survey. The VLDB Journal, 10(4), 270-294.

[77] Hamdaqa, M., Livogiannis, T., and Tahvildari, L. (2011). A reference model for developing cloud applications. In proceedings of CLOSER, 11.

[78] Hauglid, J.O., Ryeng, N. H., and Nrvg, K. (2010). DYFRAM: dynamic fragmentation and replica management in distributed database systems. Distributed and Parallel Databases, 28: pp.157-185.

[79] Herodotou, H., Borisov, N., and Babu, S. (2011). Query optimization techniques for partitioned tables. In Proc. of SIGMOD, Athens, Greece.

[80] Holland, J. H. (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI, USA.

[81] Hong, W. and Wong, E. (1989). Multiple Query Optimization Through State Transition and Decomposition. Electronics Research Laboratory, College of Engineering, University of California.

[82] Ibaraki, T., and Kameda, T. (1984). On the optimal nesting order for computing N-relational joins. ACM Trans. Database Syst. 9, 3, 482-502.

[83] IBM DB2 http://www-01.ibm.com/software/data/db2/ (last accessed 5 September 2014).

[84] Ioannidis, YE., and Kang, YC. (1990). Randomized algorithms for optimizing large join queries. Proceedings of ACM SIGMOD Conf Management of Data, Atlantic City, NJ, April,312-321.

[85] Ioannidis, YE., and Wong, E. (1987). Query optimization by simulated annealing. Proceedings of ACM SIGMOD Conf Management of Data, San Francisco, Calif,9-22.

[86] Jarke, M. (1985). Common subexpression isolation in multiple query optimization. In Query Processing in Database Systems (pp. 191-205). Springer Berlin Heidelberg.

[87] Kantere, V., Dash, D., Francois, G., Kyriakopoulou, S., and Ailamaki, A. (2011). Optimal service pricing for a cloud cache. Knowledge and Data Engineering, IEEE Transactions on, 23(9), 1345-1358.

[88] Kelly, S. (1994). Data warehousing: the route to mass customisation. John Wiley & Sons, Inc..

[89] Kernal Based Virtual Machine, www.linux-kvm.org/page/MainPage (last accessed 5 September 2014).

[90] Kllapi, H., Sitaridi, E., Tsangaris, M. M., and Ioannidis, Y. E. (2011). Schedule optimization for data processing ows on the cloud. In Proceedings of the ACM

SIGMOD International Conference on Management of Data (289-300).

[91] Konig, A. C., Ding, B., Chaudhuri, S., and Narasayya, V. R., (2012). A Statistical Approach Towards Robust Progress, Proceedings of the VLDB Endowment, Vol. 5, No. 4.

[92] Kossmann, D. (2000). The state of the art in distributed query processing. ACM Computing Surveys (CSUR), 32(4), 422-469.

[93] Kossmann, D., Kraska, T., and Loesing, S. (2010). An evaluation of alternative architectures for transaction processing in the cloud. In Proceedings of international conference on Management of data (pp. 579-590).

[94] Kossmann, D., and Stocker, K. (2000). Iterative Dynamic Programming: a New Class of Query Optimization Algorithms. ACM Transactions on Database Systems, vol.25, issue 1, 43-82.

[95] Koutris, P., Upadhyaya, P., Balazinska, M., Howe, B., and Suciu, D. (2013) Toward Practical Query Pricing with QueryMarket, SIGMOD.

[96] Krishnamurthy, R., Boral, H., and Zaniolo, C. (1986). Optimization of non-recursive queries. In: Proc. Conf. Very Large Data Bases (VLDB), Kyoto, Japan, 128-137.

[97] Lanzelotte, R. S., Valduriez, P., and Zat, M. (1993). On the effectiveness of optimization search strategies for paralel execution spaces. In Proceedings of VLDB (493-493).

[98] Lozano, M., Herrera, F., Krasnogor, N., and Molina, D. (2004). Real-coded memetic algorithms with crossover hill-climbing. Evolutionary computation, 12(3), 273-302.

[99] Lee, R., Zhou, M., and Liao, H. (2007) Request window: an approach to improve throughput of RDBMS-based data integration system by utilizing data sharing across concurrent distributed queries. Proceedings of the VLDB, Vienna, Austria,1219-1230.

[100] Lohman, G., Mohan, C., Haas, L., Daniels, D., Lindsay, B., Selinger, P. and Wilms, P. (1985) Query processing in r*. In Query Processing in Database Systems. Springer.

[101] Mahboubi, H., and Darmont, J. (2009). Enhancing xml data warehouse query performance by fragmentation. In Proceedings of ACM symposium on Applied Computing (1555-1562).

[102] Mami, I. and Bellahsene, Z. (2012). A survey of view selection methods. ACM SIGMOD Record, 41(1), 20-29.

[103] Marbukh, V., and Mills, K. (2008). Demand pricing and resource allocation in market-based compute grids: A model and initial results. ICN. (752-757).

[104] Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., and Ghalsasi, A. (2011). Cloud computing-The business perspective. Decision Support Systems, 51(1), 176-189.

[105] Mehta, M. and DeWitt, D.J. (1995) Managing intra-operator parallelism in parallel database systems. Proc. of the 21st VLDB, 382-394.

[106] Mell, P. and Grance, T. (2009). NIST definition of cloud computing. National Institute of Standards and Technology.

[107] Microsoft Parallel Data Warehouse (PDW) http://www.microsoft.com/en-us/sqlserver/solutions-technologies/data-warehousing/pdw.aspx (last accessed 5 September 2014).

[108] Mishra, P., and Eich M.H. (1992). Join Processing in Relational Databases. ACM Comput. Surv. 24(1): 63-113.

[109] Mitchell, M., Holland, J. H., and Forrest, S. (1993). When will a genetic algorithm outperform hill climbing?. In NIPS (pp. 51-58).

[110] Mondal, A., Kitsuregawa, M., Ooi, B.C., and Tan, K.L. (2001) R-treebased data migration and self-tuning strategies in shared-nothing spatial databases, ACM Proceedings of the 9th international symposium on Advances in Geographic Information Systems, GIS,28-33.

[111] Moreno, R., and Alonso-Conde, A. B. (2004). Job scheduling and resource management techniques in economic grid environments. In Grid Computing (25-32). Springer Berlin Heidelberg.

[112] Mozafari, B., Curino, C., and Madden, S. (2013) Resource and Performance Prediction for Building a Next Generation Database Cloud, CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, Online Proceedings.

[113] Myong, H.K., Henry, G.D., and Bharat, K.B. (1994) MQO at algorithm-level. Data and Knowledge Engineering, 14 (1): 57-75.

[114] Narayanan, S., Kurc, T.M., and Saltz, J. (2003) Database support for data-driven scientific applications in the grid. Parallel Processing Letters 13 (2):245-271.

[115] Nguyen, T. V. A., Bimonte, S., d'Orazio, L., and Darmont, J. (2012). Cost models for view materialization in the cloud. In Proceedings of EDBT/ICDT Workshops (47-54).

[116] Ono, K., and Lohman, G. (1990). Measuring the complexity of join enumeration in query optimization. In Proceedings of the 16th Inernational Conference on Very Large Data Bases, 314-325.

[117] Oracle Database 11g http://www.oracle.com/us/products/database/index.html (last accessed 5 September 2014).

[118] Ozsu, M.T., and Valduriez, P. (2011). Principles of Distributed Database Systems. 3rd Edition, 245-293.

[119] Papadimitriou, C. H., and Yannakakis, M. (2001). Multiobjective query optimization. In Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (52-59).

[120] Plattner, C. and Alonso, G. (2004). Ganymed: Scalable Replication for Transactional Web Applications. In Proc. of Middleware, 155-174.

[121] Polat, F., Cosar, A., and Alhajj, R. (2001). Semantic information-based alternative plan generation for multiple query optimization. Information Sciences, 137(1), 103-133.

[122] Rao, J. and Ross, K.A. (1988) Reusing invariants: A new strategy for correlated queries. In Proceedings of SIGMOD Conference.

[123] Rho, S. and March, S. T. (1997). Optimizing distributed join queries: a genetic algorithm approach. Annals of Operations Research, 71, 199-228.

[124] Rodriguez-Martínez, M. and Roussopoulos, N. (2000) MOCHA: A self-extensible database middleware system for distributed data sources. in: Proceedings of ACM SIGMOD ACM Press, pp.213-224. ACM SIGMOD Record, Vol. 29, No.2.

[125] Rosenblum, M. and Garfinkel, T. (2005) Virtual machine monitors: Current technology and future trends. IEEE Computer, 38(5).

[126] Ross, K.A., Srivastava, D., and Sudarshan, S. (1996) Materialized view maintenance and integrity constraint checking: Trading space for time. In Proceedings of SIGMOD.

[127] Roy, P., Sehadri, S., Sudarshan, S., and Bhobe, S. (2000) Efficient and extensible algorithms for multi-query optimization. Proceedings of ACM SIGMOD Conf. on Management of Data, 249-260.

[128] Saito, Y. and Shapiro, M. (2005). Optimistic Replication. ACM Comput. Surv., 37(1):42-81.

[129] Salesforce. http://www.salesforce.com/.

[130] Sarathy, V., Narayan, P., and Mikkilineni, R. (2010). Next Generation Cloud Computing Architecture: Enabling Real-Time Dynamism for Shared Distributed Physical Infrastructure. In Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE) (48-53).

[131] Schnitzer, B. and Leutenegger, S.T. (1999) Master Client R-Trees: A new parallel R-tree architecture. Proc. of 11th Conference on Scientific and Statistical Database Management, 68-77.

[132] Segev, A., and Park, J. (1989). Updating distributed materialized views. Knowledge and Data Engineering, IEEE Transactions on, 1(2), 173-184.

[133] Selinger, P.G., and Adiba, M. (1980). Access Path Selection in Distributed Database Management Systems. In Proc. First Int. Conf. on Databases, 204-215.

[134] Sellis, T.K. (1988) Multiple query optimization. ACM Trans. Database Syst., vol. 13, no. 1, 23-52.

[135] Sellis, T.K. and Ghosh, S. (1990) On the multiple-query optimization problem," IEEE Transactions on Knowledge and Data Engineering. 2(2): 262-266.

[136] Sevinc, E. and Cosar, A. (2011). An Evolutionary Genetic Algorithm for Optimization of Distributed Database Queries, The Computer Journal, vol.54, issue: 5, 717-725.

[137] Shankar, S., Nehme, R., Aguilar-Saborit, J., Chung, A., Elhemali, M., Halverson, A., ... and Galindo-Legaria, C. (2012). Query optimization in microsoft SQL server PDW. In Proceedings of international conference on Management of Data (767-776).

[138] Sharma, U., Shenoy, P., Sahu, S., and Shaikh, A. (2011). A cost-aware elasticity provisioning system for the cloud. In Distributed Computing Systems (ICDCS), 2011 31st International Conference on (559-570).

[139] Sidell, J., Aoki, P. M., Sah, A., Staelin, C., Stonebraker, M., and Yu, A. (1996). Data replication in mariposa. In Data Engineering, 1996. Proceedings of the Twelfth International Conference on (pp. 485-494).

[140] Silva, Y. N., Larson, P., and Zhou, J. (2012). Exploiting Common Subexpressions for Cloud Query Processing. In Data Engineering (ICDE), 2012 IEEE 28th International Conference on (1337-1348).

[141] Sinha, A. and Chase, C. (1996) Prefetching and caching for query scheduling in a special class of distributed applications. Proceedings International Conference on Parallel Processing, 95-102.

[142] SkyInsight web page http://www.ingres.com/products/cloud/skyinsight (last accessed 5 September 2014).

[143] Smith, J. E., and Nair, R. (2005). The architecture of virtual machines. Computer, 38(5), 32-38.

[144] Soror, A. A., Minhas, U. F., Aboulnaga, A., Salem, K., Kokosielis, P., and Kamath, S. (2010). Automatic virtual machine configuration for database workloads. ACM Transactions on Database Systems (TODS), 35(1), 7.

[145] Sourd, F., and Spanjaard, O. (2008). A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. INFORMS Journal on Computing, 20(3), 472-484.

[146] Stawowy, A. 2008. Evolutionary based heuristic for bin packing problem. Computers & Industrial Engineering 55, 465-474.

[147] Steinbrunn, M., Moerkotte, G., and Kemper, A. (1997). Heuristic and randomized optimization for the join ordering problem. Very Large Data Bases Journal 6, 3, 191-208.

[148] Steven, L., Arijit, M., and Alastair, H. (2009) The design and implementation of OGSA-DQP: A service-based distributed query processor. Future Generation Computer Systems, 25 (3):224-36.

[149] Stonebraker, M., Aoki, P. M., Litwin, W., Pfeffer, A., Sah, A., Sidell, J., ... and Yu, A. (1996). Mariposa: a wide-area distributed database system. The VLDB Journal, 5(1), 48-63.

[150] Storm, A. J., Garcia-Arellano, C., Lightstone, S. S., Diao, Y., and Surendra, M. (2006). Adaptive self-tuning memory in DB2. In Proceedings of VLDB (1081-1092).

[151] Strachey, C., (1959). Time Sharing in Large Fast Computers. Proceedings of the International Conference on Information processing, UNESCO. paper B.2.19: 336-341.

[152] Subramanian, S.N. and Venkataraman, S. (1998). Cost-based optimization of decision support queries using transient-views, Proceedings of the 1998 ACM SIGMOD international conference on Management of data. New York, NY, 319-330.

[153] Swami, A., and Gupta, A. (1988). Optimization of large join queries. Proceedings of ACM SIGMOD Conf. on Management of Data, Chicago, Ill, May, 8-17.

[154] Swami, A., and Iyer, B. (1993). A Polynomial Time Algorithm for Optimizing

Join Queries. In Proc. of ICDE, pp.345-354.

[155] Tamhankar, A. M., and Ram, S. (1998). Database fragmentation and allocation: an integrated methodology and case study. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 28(3), 288-305.

[156] Taniar, D., Leung, C.H.C., Rahayu, W., and Goel, S. (2008). High-Performance Parallel Database Processing and Grid Databases. A John Wiley and Sons, Inc., Publication.

[157] Tao, F., Feng, Y., Zhang, L., and Liao, T. W. (2014). CLPS-GA: A case library and Pareto solution-based hybrid genetic algorithm for energy-aware cloud service scheduling. Applied Soft Computing, 19, 264-279.

[158] Theodoratos, D., Ligoudistianos, S., and Sellis, T. (2001). View selection for designing the global data warehouse. Data and Knowledge Engineering, 39(3), 219-240.

[159] Thomson, A., Diamond, T., Weng, S. C., Ren, K., Shao, P., and Abadi, D. J. (2012). Calvin: Fast distributed transactions for partitioned database systems. In Proceedings of the 2012 international conference on Management of Data (1-12).

[160] Tosun, U., Dokeroglu, T., and Cosar, A. (2013). A robust Island Parallel Genetic Algorithm for the Quadratic Assignment Problem. International Journal of Production Research, 1-17.

[161] Tozun, P., Pandis, I., Johnson, R., and Ailamaki, A. (2012). Scalable and dynamically balanced shared-everything OLTP with physiological partitioning. The VLDB Journal, 1-25.

[162] Tsakalozos, K., Kllapi, H., Sitaridi, E., Roussopoulos, M., Paparas, D., and Delis, A. (2011). Flexible use of cloud resources through profit maximization and price discrimination. In Data Engineering (ICDE), IEEE 27th International Conference on (75-86).

[163] Upadhyaya, P., Balazinska, M., and Suciu, D. (2012). How to price shared optimizations in the cloud. Proceedings of the VLDB Endowment, 5(6), 562-573.

[164] Velte, A. and Velte, T. (2009). Microsoft virtualization with Hyper-V. McGraw-Hill, Inc..

[165] Voorsluys, W., Broberg, J., Buyya, R. (2011). Introduction to Cloud Computing. In R. Buyya, J. Broberg, A.Goscinski. Cloud Computing: Principles and Paradigms. New York, USA: Wiley Press. (1-44).

[166] Warneke, D., and Kao, O. (2011). Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. Parallel and Distributed Systems, IEEE Transactions on, 22(6), 985-997.

[167] Weikum, G. and Vossen, G. (2002). Transactional Information Systems. Morgan Kaufmann.

[168] Wilder, B. (2012). Cloud Architecture Patterns. O'Reilly Media.

[169] Windows Azure Platform. microsoft.com/windowsazure/(last accessed 5 September 2014).

[170] Windows Azure Storage Services REST API Ref. http://msdn.microsoft.com/en-us/library/dd179355.aspx (last accessed 5 September 2014).

[171] Wong, E., and Youssefi, K. (1976). Decomposition-a strategy for query processing. ACM Transactions on Database Systems (TODS), 1(3), 223-241.

[172] XenSource Inc, Xen, www.xensource.com (last accessed 5 September 2014).

[173] http://www.xen.org/ (last accessed 5 September 2014).

[174] Xiong, P., Chi, Y., Zhu, S., Moon, H. J., Pu, C., and Hacigumus, H. (2011). Intelligent management of virtualized resources for database systems in cloud environment. In Data Engineering (ICDE), IEEE 27th International Conference on (pp. 87-98).

[175] VMWare ESX Server, www.vmware.com/products/esx (last accessed 5 September 2014).

[176] Wu, W., Chi, Y., Hacígümüş, H., and Naughton, J. F. (2013). Towards predicting query execution time for concurrent and dynamic database workloads. Proceedings of the VLDB Endowment, 6(10), 925-936.

[177] Yang, J., Karlapalem, K., and Li, Q. (1997). Algorithms for materialized view design in data warehousing environment. In VLDB (Vol. 97, pp. 136-145).

[178] Zhang, C., Yao, X., and Yang, J. (2001). An evolutionary approach to materialized views selection in a data warehouse environment. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 31(3), 282-294.

[179] Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1), 7-18.

[180] Zhou, J., Larson, P-A , Freytag, J.C., and Lehner, W., (2007).Efficient ex-

ploitation of similar subexpressions for query processing. SIGMOD Conference, 533-544.

[181] Zhuge, Y., Garcia-Molina, H., Hammer, J., and Widom, J. (1995). View maintenance in a warehousing environment. ACM SIGMOD Record, 24(2), 316-327.

[182] Zhuge, Y., Garcia-Molina, H., and Wiener, J. L. (1996). The Strobe algorithms for multi-source warehouse consistency. In Parallel and Distributed Information Systems, 1996., Fourth International Conference on (pp. 146-157).

[183] Zilio, D. C., Rao, J., Lightstone, S., Lohman, G., Storm, A., Garcia-Arellano, C., and Fadden, S. (2004). DB2 design advisor: integrated automatic physical database design. In Proceedings of the Thirtieth international conference on Very large data bases-Volume 30 (pp. 1087-1097). VLDB Endowment.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name:	Dökeroğlu, Tansel
Nationality:	Turkish (TC)
Date and Place of Birth:	24 May 1969, Luleburgaz
Marital Status:	Married
Phone:	+90 312 441 44 75
e-Mail:	tansel@ceng.metu.edu.tr

EDUCATION

Degree	Institution	Year of Graduation
MS	METU-Computer Engineering	2006
BS	Turkish Land Force Academy	1991
High School	Kuleli Military High School	1987

WORK EXPERIENCE

Year	Place	Enrollment
2000-2002	General Staff HQ CIS Department	Software Developer
2002-2004	Land Force HQ Decision Support Department	Project Manager
2004-2007	General Staff HQ CIS Department	System Administrator
2007-2008	Ministry of Defence CIS Department	Database Administrator
2008-2014	Land Force HQ Distance Learning Center	Software Developer

FOREIGN LANGUAGES

Advanced English

PUBLICATIONS

Published Journals

1. Dokeroglu T., Ozal S., Bayir M.A., Cinar M.S., Coşar A., (2014) Improving the performance of Hadoop Hive by sharing scan and computation tasks, Journal of Cloud Computing: Advances, Systems and Applications 3(1), 12. Springer.
2. Dokeroglu, T., Cosar, A. (2014). Optimization of one-dimensional Bin Packing Problem with island parallel grouping genetic algorithms. Computers & Industrial Engineering, 75, 176-186.
3. Dokeroglu, T., Sert, S.A., and Cinar, M.S. (2014) Evolutionary multiobjective query workload optimization of Cloud data warehouses, The Scientific World Journal.
4. Tosun, U., Dokeroglu, T., and Cosar, A., (2013) A Robust Island Parallel Genetic Algorithm for the Quadratic Assignment Problem, International Journal of Production Research 51(14), 4117-4133.

Published Conferences

1. Dokeroglu, T., Cosar, A. (2014). Integer Linear Programming for MQO problem, Proceedings of the 29th ISCIS, Krakow, Poland.

2. Dokeroglu, T., Sert, S.A., Cinar, M.S., and Cosar, A. (2014). Designing Cloud Data Warehouses using Multiobjective Evolutionary Algorithms, International Conference on Agents and Artificial Intelligence (ICAART) Eseo, Angers, Loire Valley, France.

3. Dokeroglu,T., Tosun, U., and Cosar, A. (2013). Evaluating the Performance of Recombination Operators with Island Parallel Genetic Algorithms, International Federation of Automatic Control (IFAC), Saint Petersburg, Russia.

4. Dokeroglu, T. (supervised by Ahmet Cosar) (2012). Parallel Genetic Algorithms for the Optimization of Multi-Way Chain Join Queries of Distributed Databases, the 38th VLDB Ph.D. Workshop, August 27-31, Istanbul/TURKEY.

5. Dokeroglu, T., Tosun, U., and Cosar, A. (2012). Particle Swarm Intelligence as a Novel Heuristic for the Optimization of Distributed Database Queries, The 6th International Conference on Application of Information and Communication Technologies AICT2012 Georgia, Tbilisi, 17-19.

6. Dokeroglu, T. Tosun, U., and Cosar, A. (2012). Parallel Optimization with Mutation Operator for the Quadratic Assignment Problem Proceedings of WIVACE, Italian Workshop on Artificial Life and Evolutionary Computation, Parma/Italy.

7. Dokeroglu, T and Cosar, A. (2011). Dynamic Programming with Ant Colony Optimization Metaheuristic for The Optimization of Distributed Database Queries, Proceedings of the 26th ISCIS, London, UK.

8. Tosun, U., Dokeroglu, T., and Cosar, A. (2012). Heuristic Algorithms for Fragment Allocation in a Distributed Database System, 27th ISCIS, October 3-5, Paris/France.