LOAD BALANCING BY USING MACHINE LEARNING IN CPU-GPU HETEROGENEOUS DATABASE MANAGEMENT SYSTEM

by
ANIL ELAKAŞ

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Master of Science

Sabancı University
August 2020

LOAD BALANCING BY USING MACHINE LEARNING IN CPU-GPU HETEROGENEOUS DATABASE MANAGEMENT SYSTEM

APPROVED BY:

Asst. Prof. ERDİNÇ ÖZTÜRK.. [illegible]
(Thesis Supervisor)

Asst. Prof. KAMER KAYA.. [illegible]

Asst. Prof. ALPER ÖZPINAR.. [illegible]

DATE OF APPROVAL: 31.08.2020..

ABSTRACT

LOAD BALANCING BY USING MACHINE LEARNING IN CPU-GPU HETEROGENEOUS DATABASE MANAGEMENT SYSTEM

ANIL ELAKAŞ

COMPUTER SCIENCE AND ENGINEERING M.S. THESIS, DEC 2020

Thesis Supervisor: Asst. Prof. Erdinç Öztürk

Keywords: load balancing, database management systems, machine learning, high performance computing, query optimization

Conventional OLTP systems are slow in performance for analytical queries. In the existing heterogeneous architecture OLAP database management systems, no system distributes work using machine learning. In this study, the DOLAP architecture, which is a high-performance column-based database management system developed for shared memory architectures, is explained. Also, job distribution algorithms based on heuristic and machine learning methods have been developed for computing hardware with different characters such as CPU and GPU on the server on which the database is running, and their performance has been analyzed.

ÖZET

CPU-GPU HETEROJEN VERITABANI YÖNETIM SISTEMINDE MAKINE ÖĞRENMESI KULLANARAK YÜK DENGELEME

ANIL ELAKAŞ

BİLGİSAYAR BİLİMLERİ VE MÜHENDİSLİĞİ YÜKSEK LİSANS TEZİ, ARALIK 2020

Tez Danışmanı: Asst. Prof. Erdinç Öztürk

Klasik OLTP sistemleri analiz sorguları için performans olarak yavaş kalmaktadır. Var olan heterojen mimarili OLAP veri tabanı yönetim sistemlerinde ise makine öğrenmesi kullanarak iş dağıtımı yapan bir sisteme rastlanmamıştır. Bu çalışmada paylaşımlı hafıza mimarileri için geliştirilmiş, yüksek başarımlı ve sütun tabanlı bir veri tabanı yönetim sistemi olan DOLAP mimarisi anlatılmıştır. Ayrıca, veri tabanının üzerinde çalıştığı sunucu üzerinde bulunan, CPU ve GPU gibi farklı karakterlerdeki hesaplama donanımları için, buluşsal yöntemlere ve makine öğrenmesi yöntemlerine dayanan iş dağıtım algoritmaları geliştirilmiş ve performansları analiz edilmiştir.

Anahtar Kelimeler: yük dengeleme, veritabanı yönetim sistemleri, makine öğrenmesi, yüksek performansla hesaplama, sorgu eniyilemesi

ACKNOWLEDGEMENTS

This work is supported by TÜBİTAK project number 118E044.

The successful completion of this project would not have been feasible without the help of several key individuals. First, I would like to express my deepest gratitude to my thesis advisor, Asst. Prof. Erdinç Öztürk, for his considerable encouragement, worthwhile guidance and insightful comments during the thesis process.

I would like to express my sincere gratitude and appreciation to my professors Asst. Prof. Kamer Kaya and Asst. Prof. Sinan Yıldırım, for their encouragement, creative and comprehensive advice until this work came to existence.

I would like to express my deepest thanks and sincere appreciation to Asst. Prof. Alper Özpınar of Istanbul Commerce University, for his ideas and useful comments during the thesis process.

I am thankful to my teammate, Anes, for his time and effort during the project.

I am deeply grateful to my parents Ayşegül and Aydın for their limitless love, endless support and precious trust.

I would like to express my heartfelt gratitude and sincere appreciation to my girl-friend, Esra, for her endless love, support, care and patience.

I would also like to state my special thanks to my close friends: Deniz, Eray, and Görkem for their support.

Dedicated to
the people who have supported me through my master education.
Especially my family, my love and my friends...

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CPU: Central Processing Unit
ALU: The Arithmetic Logic Unit
CU: The Control Unit
MIPS: Millions of Instructions per Second
AMD: Advanced Micro Devices
IC : Integrated Circuit
GPU: Graphic Processing Unit
MIT: Massachusetts Institute of Technology
IBM: International Business Machine Corporation
FPGA: Field Programmable Gate Array
OLAP: On-line Analytical Processing
OLTP: On-line Transactional Processing
IoT: Internet of Things
SP: Streaming Processor
API: Application Programming Interface
CUDA: Compute Unified Device Architecture
DMBS: Database Management System
IDS: Integrated Data Store
CODASYL: Conference/Committee on Data Systems Languages
COBOL: Common Business-Oriented Language
IMS: Information Management System
BOM: Bill of materials
ACID: Atomicity, Consistency, Isolation, Durability
SQL: Structured Query Language
MIT: Massachusetts Institute of Technology
AI: Artificial Intelligence
JSON: JavaScript Object Notation
REST: REpresentational State Transfer
CSV: A comma-separated values

1. INTRODUCTION

Big data is a term that describes the large volume of data that inundates a business on a day-to-day basis. Rather than the amount of data, the usage methodology of data is important. If analyzed properly, big data can give insights that lead to better decisions and strategic business moves. Industry analyst Doug Laney expressed the big data as three Vs: Volume, Velocity and Variety. Through this situation the concept of big data accelerated in the beginning of the 2000s.

Big data usage has been increasing steadily over the past years. As record counts and data quantity increases, so does the need for a database system to effectively manage this quantity of information. To make this information more useful, easy to access and protected, more efficient and tailor-made database management systems are in need. For this purpose, we have created DOLAP as a database management system. DOLAP has a column-based structure designed with OLAP (On-Line Analytical Processing) analysis queries and tasks in mind.

The developed open-source database can be used efficiently with different architectures such as CPU, GPU, and FPGA at the same time, with smart work distribution methods. DOLAP can increase performance with query optimizations and sharing methods, which work on a heterogeneous system on the server-side and it is a database management system where users can generate dynamic and static queries thanks to the interface developed on the client-side.

In this thesis, four different job distribution methods; random, algorithm 1, algorithm 2 and the multiple linear regression machine learning were designed and implemented. These methods -especially the multiple linear regression method- can efficiently distribute analysis queries between CPU and GPU devices. Also, performance analyzes and obtained test results were provided.

2. MOTIVATION

The main goal was to create a lightweight, adaptable, and scalable database that does not use an index and relational structure when writing a DOLAP database so that it can process OLAP workloads quickly. In addition, unpredictable query flow always become a significant challenge for database management systems especially ones that uses algorithm based methods; therefore my proposed load balancing model addresses this issue explicitly by learning query flow and predicting by using multiple linear regression machine learning method.

3. RELATED WORK

Conventional OLTP (On Line Analytical Processing) systems are slow in performance in dynamic and continuous analysis queries. Today's High Performance Computing hardware can perform differently in different queries. GPU-based databases with high investment such as Kinetica Kinetica (2020) and OmniSci (MapD) MAPD (2020) have become very popular in recent years. OmniSci is SQL-based, relational, columnar and specifically developed to harness the massive parallesim of modern CPU and GPU hardware. Differences between DOLAP and OmniSci are, OmniSci has multiple CPUs and GPUs where DOLAP has a single CPU and a single GPU. OmciSci is SQL-based where DOLAP is NoSQL based. DOLAP uses machine learning in load balancer module where OmniSci is not. Another similar dbms is Kinetica. Kinetica has a GPU-based architecture. They offers machine learning solution to improve accuracy of inferences but not for a load balancing between CPU and GPU as we do in DOLAP. Existing OLAP databases do not support CPU, GPU and FPGA simultaneously on heterogeneous systems. Also, according to the literature search, different load balancing modules use machine learning in cloud systems Li, Zhang, Wan, Ren, Zhou, Wu, Yang & Wang (2019), heterogeneous network systems Rajesh, Bagan, K & M (2019) , IoT systems Gomez, Shami & Wang (2018) and distributed computing systems. However, a system that distributes work using machine learning has not been encountered in database management systems with heterogeneous architecture. As a result, no database management system in the literature can provide all of the features mentioned in the section 1.

4. BACKGROUND

The background gives brief but important information about the devices I use in my project. Each section contains information about their history and how they worked.

4.1 Multicore and Manycore Architectures

4.1.1 Central Processing Unit (CPU)

We had no chance to spend time without interacting with a processor. For example, from desktop computers to cell phones, from laptops to tablets, we interact with a processor in our daily lives.

Intel launched the 4004 processor that was just 4-bit on 1971. Electronic News introduced the processor with following sentence: ”Announcing A New Era In Integrated Electronics”. In that moment the Intel 4004 became the first general-purpose programmable processor on the market. It was a "building block" that engineers could purchase and then customize with software to perform different functions in a large range of electronic devicesIntel (2020b).

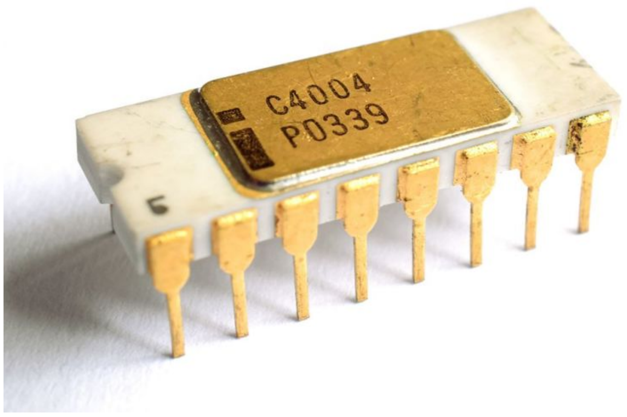


Figure 4.1 Intel 4004, First processor

This new fashioned hardware is clocked at 740 kHz rate, with the capacity of executing up to 92,000 instr/sec with a single instruction cycle of 10.8 microseconds where it is composed of 2,300 transistor. Intel (2020a)

Intel dominated the CPU market and launched various following CPU designs for the next 20 years. Then, on March 22, 1993, a new famous name in the CPU industry's history has emerged, the Intel Pentium processor. That famous device operated at a 60-66 MHz and 100 Millions of Instructions per Second (MIPS). It has a 16 KB L1 cacheIntel (Intel). The innovation movement from Intel continued for 2 more years. In 1995, AMD which is the major competitor of Intel in today's market introduced its first serious product, AMD AM5x86. The race among Intel and AMD has been going on ever since.

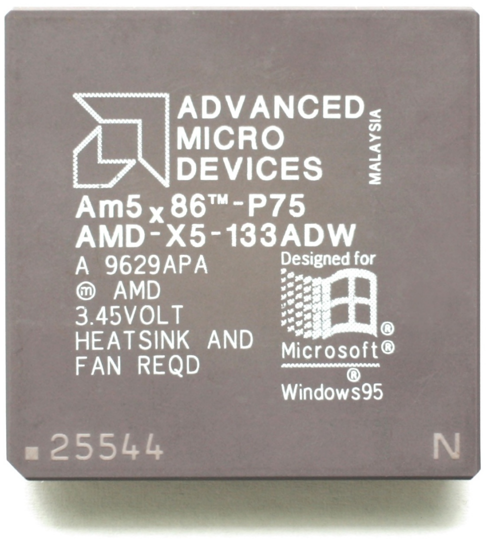


Figure 4.2 AMD AM5x86

The one of the milestones in the CPU history was reaching the achievement of the produce first 1 GHz processor. Intel launched the Intel Pentium III on February 26, 1999, and AMD launched the AMD Athlon on June 23, 1999. However, this accomplishment was reached first by the AMD Athlon K75 model which is released in November 1999. Accordingly, "Athlon" was a felicitous name for AMD's important processor since it is the Greek word for "Champion/trophy of the games".

Figure 4.3 Intel Pentium III

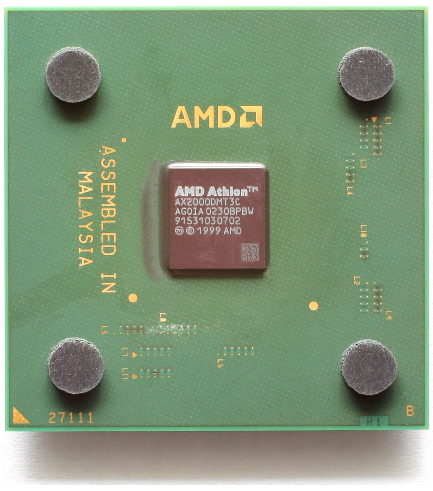


Figure 4.4 AMD Athlon

In just over almost 50 years, processor technology powered up from 740 kHz to the GHz level which means more than a 1300 % increase, and number of the transistors increased from 2,300 to more than a billion (over a 434,000 % increase).

4.1.2 CPU Multi-core

The term "multi-core" refers to a multiple-core processor which is an integrated circuit(IC) where two or more processors have been attached for increased performance, reduced power consumption, and more efficient simultaneous processing of multiple tasks via parallel processing. Parallel processing is based on the principle of decomposition. Decomposition is breaking the problem into smaller sub-problems and then solving those small sub-problems concurrently. Because of the decomposition, parallel computing has become the leading norm in computer architecture. Figure 4.5 shows the multi-core design details of an Intel I5 750 Quad-Core Processor Rouse (2013) Firesmith (2017).

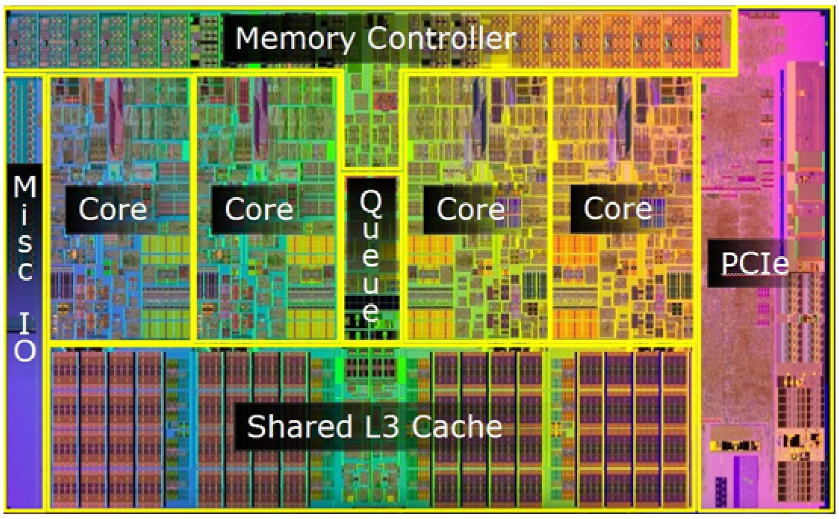


Figure 4.5 Intel I5 750 Quad Core Processor Design

Where parallel processing can complete multiple tasks using two or more processors, serial processing (also called sequential processing) will only complete one task at a time using one processor. If a computer needs to complete multiple assigned tasks, then it will complete one task at a time. Likewise, if a computer using serial processing needs to complete a complex task, then it will take longer compared to a parallel processor Rouse (2019b).

Parallel computing saves time by decreasing the computation time. Also, it saves money because of lower prices of the parallel components. We can solve big problems by dividing them into the smaller sub problems through using parallel computing because it is not possible to solve big problems with a single computing resource. Also, there are some drawbacks which are transmission, speeds, limits to miniaturization, and economic in serial computing. Nowadays, computer architectures are believe increasing performance of hardware level in multiple execution units, pipelined instructions, and multi-core for escaping constrains.

4.1.3 Graphic Processing Unit (GPU)

As Prof. Jack Dongarra indicated, "GPUs have evolved to the point where many real-world applications are easily implemented on them and run significantly faster than on multi-core systems. Future computing architectures will be hybrid systems with parallel-core GPUs working in tandem with multi-core CPUs." We can understand the future place of high-performance computing from his comment Shi, Kindratenko & Yang (2013).

In the middle of the 20th century, in 1951, the Massachusetts Institute of Technology (MIT) produced the Whirlwind, a flight simulator for the Navy. Also it may be considered the first 3D graphics system, the base of today's GPUs was formed in the mid-70s with video shifters and video address generators. They carried out information from the central processor to the display. Specially designed graphics chips were extensively used in arcade system boards. In 1976, RCA built the "Pixie" video chip, which was able to output video signal at 62×128 resolution. Graphics hardware of the Namco Galaxian arcade system supported RGB color, multi-colored sprites, and tilemap backgrounds as early as 1979.

The introduction of graphics units happened in early 1980s where both Intel and International Business Machines Corporation (IBM) brought specially designed products to the market. IBM started using black and white and a color display adapter (MDA/CDA) in its computers. Still we couldn't say it is a modern GPU, it was a particular computer component designed for one purpose: to display video. In the beginning, it includes 80 columns and 25 lines that consist of text characters or symbols. ISBX 275 Video Graphics Controller Multimodule Board, introduced by Intel in 1983. It was the next radical device. At 515x512 resolution, it supported only black and white colors while at 256x256 resolution it supported 8 colors. Other companies in the market like Commodore and Texas Instruments increased their simple graphics capabilities either on-chip or using an external card. Although these cards had primitive functionality, these were comparatively high-priced. Primitive processors were supporting functions like shape drawing, filling an area and modification of basic images.

After two years, in 1985, three engineers from Hong Kong founded Array Technology Inc in Canada, today is known as ATI Technologies. The company was the leader of the market for years with its Wonder line of graphics boards and chips. At the beginning of 1991, S3 Graphics introduced its S3 86C911 card which was one of the first norms for the GPU industry. It named after the Porsche 911 to refer the performance increase. In 1995, all great companies in the market have added 2D acceleration support to their chips.

Figure 4.6 S3 S386C911 (1991)

Different “languages” like OpenGL and Direct showed up in the end of the 90s for supporting the commonality of graphics processing. Throughout the 1990s, substantial improvement was provided with the additional application programming interfaces (APIs) on the integration level of video cards. OpenGL boosted the software’s capability was usually before then Direct and it accomplished being used across cards and platforms. Before the beginning of the new century, both APIs established assistance for transform and lighting (T&L) which formed a remarkable refinement in GPU processing. T&L permitted an effortless mapping of 3D images to a 2D plane while joining the lighting all in one. At that time, there were some competitive corporations; NVIDIA, ATI, 3dfx, and S3.

2D graphic processing got into nearly every system by the middle of the 1990s. Furthermore, the competition was on the road to three dimensions processing. 3DFx’s Voodoo graphics card introduced in 1996 and dominated around %85 of the market. The next product of the company was Voodoo2 in 1998. The card had three on board chips. It was also among the first graphics cards to support the parallel operation of two cards in a single computer.

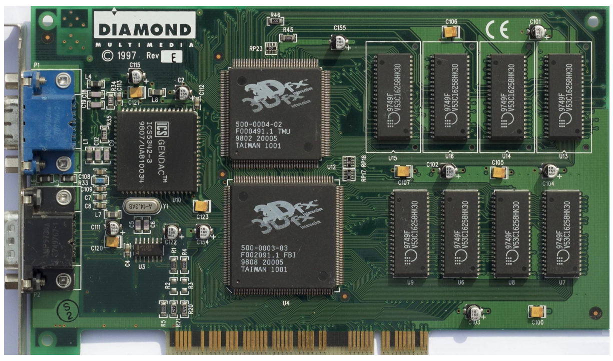


Figure 4.7 3DFx Voodoo (1996)

Then finally, on October 11, 1999, NVIDIA released NVIDIA GeForce 256 which is the world's first GPU. NVIDIA specified the graphics processing unit as "a single-chip processor with integrated modification, lighting, triangle setup/clipping, and rendering engines that process at a minimum of 10 million polygons/second."Olena (2018)

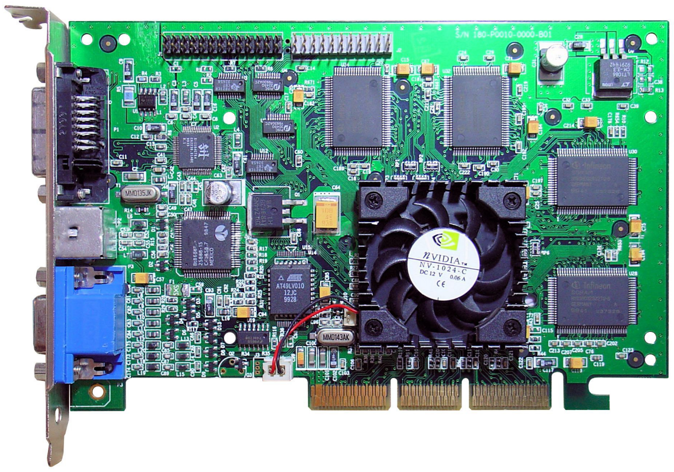


Figure 4.8 NVIDIA GeForce 256 (1999)

4.1.4 GPU Architecture and Parallelism

The graphics processing unit or GPU is a many-core multiprocessor system; the GPU chip has multiple streaming multiprocessors (SM), each of which consists of many streaming processors (SP) or cores. An SP is heavily multi-threaded with in-order execution that falls into SIMT (Single Instruction, Multiple Threads) architecture which shares its control and instruction cache to others, therefore, all threads running on SPs share the same function or program which is called a kernel. This architecture allows performing massive computations on multiple data within a few seconds. In contrast, The Central Processing Unit (CPU) is a multi-core unit; each core can achieve high performance for a single-threaded execution Linderman (2009) Sanders & Kandrot. (2010) Kirk & mei W. Hwu. (2010).

As with Intel and AMD, and ATI have been competing for many years. At the 2007, current level of the rivalry began. The age of the general purpose GPUs. Both NVIDIA and ATI had been producing their graphics cards with increasing capabilities. Despite that, these companies choose a separate paths to general purpose computing GPU (GPGPU). Same year, NVIDIA introduced CUDA

(Compute Unified Device Architecture). CUDA, is a parallel computing platform and application programming interface (API) model. Also, CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels Wikipedia (2020). Then, two years after the launch of the CUDA, OpenCL became widely supported. OpenCL specifies programming languages for programming CPUs, GPUs, FPGAs and application programming interfaces (APIs) to control the platform and execute programs on the compute devices. Also, OpenCL provides a standard interface for parallel computing using task-based and data-based parallelism Nvidia (2020). Therefore, GPUs became a more common computing hardware. The following figure shows the CUDA Software Architecture Kaur & Er.Nishi (2014).

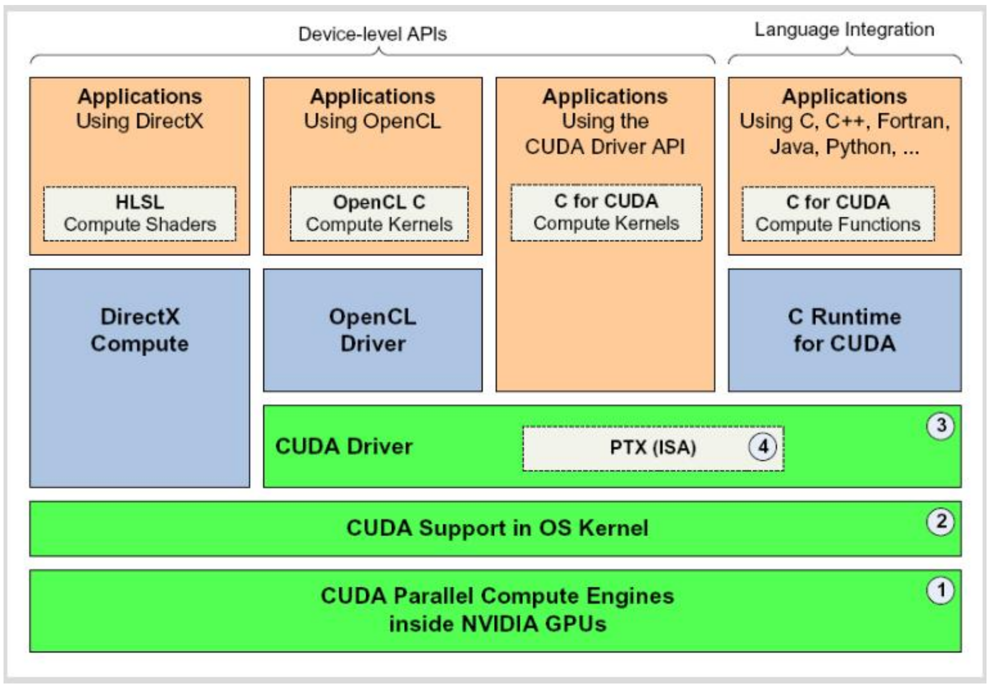

Device-level APIs
Language Integration
Applications Using DirectX
HLSL Compute Shaders
Applications Using OpenCL
OpenCL C Compute Kernels
Applications Using the CUDA Driver API
C for CUDA Compute Kernels
Applications Using C, C++, Fortran, Java, Python, ...
C for CUDA Compute Functions
DirectX Compute
OpenCL Driver
C Runtime for CUDA
CUDA Driver
PTX (ISA)
4
3
CUDA Support in OS Kernel
2
CUDA Parallel Compute Engines inside NVIDIA GPUs
1

Figure 4.9 CUDA Software Architecture

What could be the next innovation in GPU world? In January 7, 2010, NVIDIA and Audi introduced NVIDIA's Tegra GPUs. Tegra powers multimedia systems in 2010 Audi cars all over the world. This union of American engineering and German engineering establishes high technology GPUs to the automotive industry, allowing innovative visual abilities like three dimensions navigation with modern driver assistance and safety systems, and a "dual zone" multimedia system that allows two different videos to be shown at the same time on several screens.Nvidia (2010)

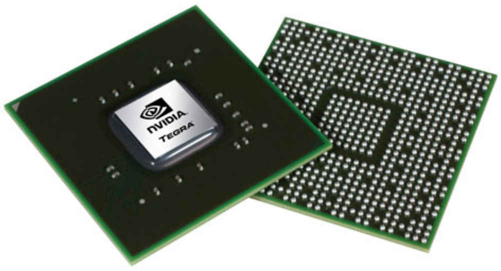


Figure 4.10 NVIDIA Tegra GPU

Figure 4.11 Audi Entertainment System With NVIDIA Tegra

"Visual computing has become important discriminator between luxury cars, whether it's delivering the necessary information to the driver or offering state-of-the-art entertainment options to the passengers," said Johan de Nysschen, Audi of America president. "We partnered with NVIDIA because it is clearly the leader in this field."

NVIDIA and Audi also worked with Google to provide Google Earth on forthcoming 3G MMI systems powered by NVIDIA GPUs. This union started with the Audi A8 in 2011. Google Earth technology has been developed by Keyhole. NVIDIA invested in the company and then Google purchased the company. The outcome of this three-way association is an industry first – a stunning 3D navigation system with detailed terrain models, snappy performance, 3D landmarks and a highly intuitive visual interface.

At present, GPUs are not just for graphics. It is used in many different areas such as artificial intelligence, machine learning, oil exploration, geoscience, scientific image processing, statistics, linear algebra, 3D reconstruction, medical research, and even stock options pricing determination.

In 2018, Audi launched new Audi A8 — which is the world's first Level 3 autonomous driving car to go into production — features a multitude of high-tech splendors, all powered by NVIDIA Nvidia (2017).

Figure 4.12 Audi A8 AI Car Powered by NVIDIA GPU

4.2 Database Management System (DBMS)

Database Management System (DBMS) is system software for creating and managing databases. A DBMS makes it possible for end-users to create, read, update, and delete data in a database. The DBMS fundamentally serves as an interface between the database and end-users or application programs, ensuring that data is consistently organized and remains easily accessible Rouse (2019a). Some of the application areas of the DBMS are; banking, airlines, universities, telecommunication, finance, sales, manufacturing, industry, education, online shopping and HR management

4.2.1 DBMS History and Database Models

4.2.1.1 Early Years and Navigational Database

The history of the DBMS started about 60 years ago. The first example of the DBMSs was a Navigational Database System. The Navigational Database is a type of database in which records or objects are found primarily by following references from other objects. One of the earliest navigational databases was the Integrated Data Store (IDS), which was developed by Charles Bachman for General Electric in the 1960s. The Oxford English Dictionary cites a 1962 report by the System Development Corporation of California as the first use of the term "database" in a specific technical sense Universiy (2020).

Since computers grew in speed and capability, multiple general-purpose database systems showed up by the middle of the 1960s for commercial use. Interest in a standard began to increase, and Charles Bechman, the founder of the Integrated Data Store, founded the "Database Task Group" within Conference/Committee on Data Systems Languages (CODASYL), the group responsible for the creation and standardization of Common Business-Oriented Language (COBOL). In 1971, the Database Task Group delivered its standard, which generally became known as the "CODASYL approach". Then, some commercial products based on this approach entered the market Foote (2017).

In 1966, IBM launched the Information Management System (IMS) with Rockwell and Caterpillar for the Apollo program, where it was used to inventory the very large bill of materials (BOM) for the Saturn V moon rocket and Apollo space vehicle. The IMS Database component stores data using a hierarchical model, which is quite different from IBM's later released Relational Database. In IMS, the hierarchical model is implemented using blocks of data known as segments. Each segment can contain several pieces of data, which are called fields Foote (2017).

4.2.1.2 Relational Database

In 1970, Edgar F. Codd who was working for IBM in Silicon Valley wrote several papers that described a new method to database structure that finally resulted in the revolutionary A Relational Model of Data for Large Shared Data Banks. In the paper, he expressed a new method for storing and working with large databases. Rather than storing records in some kind of linked list of free-form records as in CODASYL, Codd proposed a "table" method with a fixed-length records and every single table used for a different type of entity. A linked-list system would be very inefficient when storing "sparse" databases where some of the data for anyone record could be left empty. The relational model solved this by splitting the data into a series of normalized tables (or relations), with optional elements being moved out of the main table to where they would take up room only if needed. Data may be freely inserted, deleted and edited in these tables, with the DBMS doing whatever maintenance needed to offer a table view to the application/user Praveen, Chandra & Wani (2017).

IBM started working on a prototype system loosely based on Codd's concepts as System R in the early 1970s. System R was the first implementation of SQL, which has since become the standard relational data query language. In 1975, the initial edition was ready, afterwards work started on multi table systems. Through this new system, the data could be split in such a way that all data for a record need not be stored in one large "chunk". In 1978-1979, multi-user versions were released. Also, that version includes query language, SQL Praveen et al. (2017).

In 1978 Larry Ellison and his coworkers Bob Miner and Ed Oates wrote the Oracle Version 1 in assembly language. However, Oracle V1 is never officially released. In 1979, Oracle Version 2, the first commercial SQL relational database management system, is released. The company changes its name to Relational Software Inc. (RSI) Oracle (2007).

The 1980s are a precursor of the age of desktop computing. The new computers authorized their users with spreadsheets like Lotus 1-2-3 and database software like dBASE. The dBASE was one of the first DBMS for microcomputers and the most successful in these days Lazzareschi (1990). The dBASE system includes the core database engine, a query system, a forms engine, and a programming language that ties all of these components together. dBase's file format is .dbf, it is widely used in applications needing a simple format to store structured data Praveen et al. (2017).

4.2.1.3 Object Oriented Database

Object-oriented programming started to rise in the early 1990s. A lot of research has done at that time. One of these works expended existing relational database concepts by adding object concepts. The researchers intended to possess a declarative query-language based on predicate calculus. One of the most distinguished research projects, Postgres (UC Berkeley), came up with two products having relation with that research: Illustra and PostgreSQL. In the mid-1990s, the very first commercial products stared emerging, including Omniscience (Omniscience Corporation, acquired by original Oracle Lite), Illustra (Illustra information Systems, acquired by Informix Software, then owned later by IBM), and UniSQL. Moreover, the founder of Paradigma Software Incorporation, the Ukrainian software developer Ruslan Zasukhin around the mid-1990s, introduced and distributed the first edition of Valentina Database as a C++ Software Development Kit (SDK). In the next decade, PostgeSQL had become a market-base feasible database. Also PostgeSQL is the starting point for various existing products that maintain its Object-Relational Database Management System (ORDBMS) features Praveen et al. (2017).

4.2.1.4 Multi Dimensional Database

In such databases, multidimensional structures are used to edit and express data. These structures are expressed as fragmented cubes and the user accesses the data through these structures. The idea behind these databases is that a relational model can be expressed in terms of rows and columns Praveen et al. (2017).

4.2.1.5 NoSQL Database

Despite intensive research on the relational databases and the variations produced, the fact that these databases are not in the tabular form has created scalability problems in terms of computation costs. NoSQL, word meaning Not Only SQL, but database model has also opened a new page in search of a database with flexibility and scalability capabilities. More than 150 products using this database are currently used by leading companies. This database model can be examined in four main sub-headings: Key-Value, Column-Based, File-Based and Graph-Based Grolinger, Higashino, Tiwari & Capretz (2013).

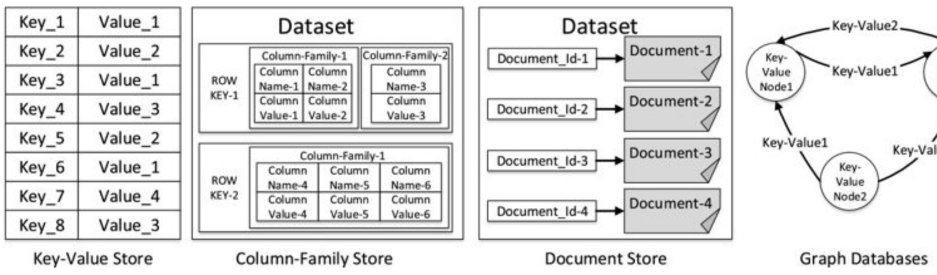


Figure 4.13 NoSQL database typesGrolinger et al. (2013)

The key-value database is a simple data model. The keys in this database model are used to express a unique value. With this feature, this database model can be considered as a large dictionary. In this database model value can be used for any data type (string, integer, array, object) and is invisible to the database. The key is used to access these values. Also, to be a schema-free database model, this database is extremely suitable for distributed data systems but is not suitable for keeping structured and relational data. For such applications to be executed in this database model, the application interacting with the database must be coded accordingly. Also, because the data is invisible to the database, query and indexing operations can only be performed on keys Grolinger et al. (2013).

The second database model is a Column Family Database. Most of the column-based NoSQL databases are derived from the database model named Bigtable, which is Google's column-based database. The data set in this database consists of rows with different column groups. Like key-value store database, in this model column key (row key) forms the key. Column-based databases can also be considered collections of key-value binaries. Or, more extensively, they may also be considered as expandable recordings that are owned by semi- format indexes. Column-based databases can keep each row in a different column family. Column-based databases occupy a large place within the NoSQL databases. Examples of databases using this model are Cassandra, SQream, Kinetica, Hadoop HBase.

The third model is a Document Oriented Database. Document-based NoSQL is a key-value database derivative that uses keys to locate documents within the database. Most document-based database expresses documents using JSON and derivative notations. Document-based databases are useful in applications where the input is in the form of documents or complex data structures Grolinger et al. (2013).

The last model is a Graph Database. Graph-based databases use graphs as a data model. This database model can be used to efficiently store relationships between nodes of the graph. The edges and nodes also have the properties of key-value pairs. This database model is suitable for use in applications such as social networking applications, pattern recognition, pathfinding Grolinger et al. (2013).

4.2.1.6 New-SQL Database

NewSQL was created to increase the scalability of relationship-oriented SQL databases. It aims to increase scalability by compromising relational features. An interesting feature of these databases is that, although users interact with tables and relationships, they use different data structures as internal data structures. For example, NuoDB keeps its data according to the key-value model. Another important feature of databases using the NewSQL database model is that they maintain the ACID (Atomicity, Consistency, Isolation, Durability) properties of the relational databases, although they are scalable Praveen et al. (2017).

4.2.2 Workload Types

The frequently encountered workloads of today's databases can be analyzed in two main sections as transferring information and data analytic.

4.2.2.1 Online Transaction Processing (OLTP)

OLTP is the workload that contains transmission records, banking transactions, historical data, in other words, OLTP contains data where the storage and consistency are essentials. OLTP workloads consist of formatted data and these data must be compatible with the ACID semantic structure. Fast processing, number of operations per second are the most wanted features in these workloads, in addition to that quick answers should be given to quick, simple queries. An example of an OLTP workload is banking transactions Appuswamy, Karpathiotakis, Porobic & Ailamaki (2017) Warehouse (2020).

4.2.2.2 Online Analytical Processing (OLAP)

OLAP workloads are usually versions of OLTP workloads that will be applied to analytical procedures for decision making, planning, and learning purposes. OLAP workloads are usually summarized and complex queries are executed on them. In OLAP workload queries, real-time or near-real-time response times are wanted Appuswamy et al. (2017) Warehouse (2020) Conn (2005).

4.2.2.3 Hybrid Transaction Analytical Processing (HTAP)

HTAP workloads are defined by performing OLTP and OLAP workloads on the same database and shared data on the transfer processing and data analysis. It has emerged due to the recent need for examination of real word business data which has been just recorded. High-performance databases that are optimized for working on HTAP workloads, such as HyPer, are available in the literature Appuswamy et al. (2017).

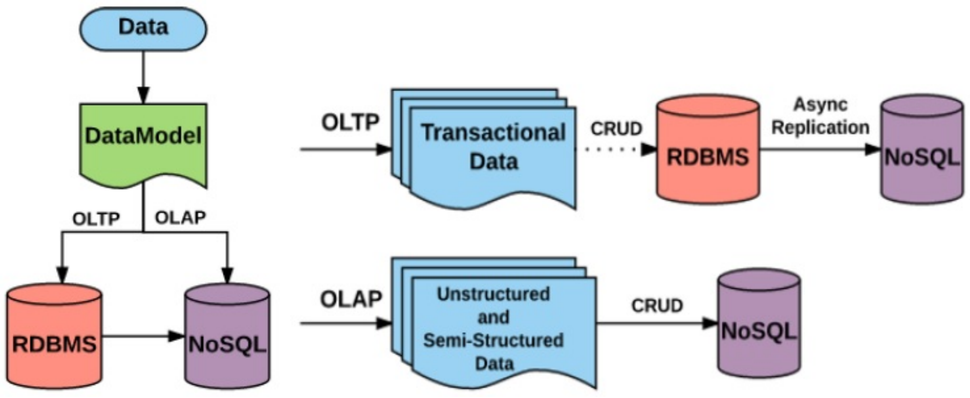


Figure 4.14 OLTP and OLAP workload typesZheng (2018)

4.3 Machine Learning

Machine learning emphasizes on the development of programs that accesses and uses data to learn and automate the improvement of its performance from the gained experience. The major point of machine learning models is to avoid explicit programming and provide the ability to predict, label and classify automatically. Figure 4.15 demonstrates the Machine Learning Process.

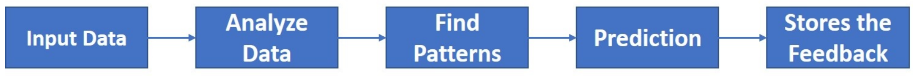


Figure 4.15 Machine Learning Process

4.3.1 Machine Learning History

Nowadays, machine learning builds tools for instance self-driving cars, unmanned drones, voice-activated assistants, robots, social media feed, and recent database management systems. Besides, the concepts behind machine learning have a long history and depend on maths from ages ago and huge technological development in computing in the last 70 years.

In the 18th Century, Thomas Bayes released "An Essay towards solving a Problem in the Doctrine of Chances". The essay shows work that supports the basis of the Bayes Theorem. Adrien-Marie Legendre introduced the Least Squares Method in 1805, which is used widely in data fitting. Followed by "Théorie Analytique des Probabilités" theorem from Pierre-Simon Laplace published in 1812, in which he extends Bayes's work and outlined the term known as Bayes' Theorem O'Connor & Robertson (2016). One hundred years later, Andrey Markov described a poem analyzing technique which became later the Markov Chain.

In the middle of the 20th Century, Alan Turing published Computing Machinery and Intelligence, where his -long living- famous question emerged: "Can machines think?". On the basis of the increasing knowledge of the power of computers, the paper stated the fundamentals of "Artificial Intelligence" through several problematic questions and experiments which were the first attempts to develop artificial intelligence models. It famously proposed the test known as the 'imitation game' to determine the degree of dependency from a human's decision in distinguishing "Intelligently" between a human and a computer through communication with them both through typed messages bbc (2020).

In 1951, the first artificial neural network was simulation introduced by Marvin Minsky and Dean Edmonds, they built it as a computer program to simulate the organic brain functionalities. The Stochastic Neural Analog Reinforcement Computer (SNARC) had several experiments to learn from then it was used to search a maze, similar to what a rat does in a psychology experiment. Marvin Minsky started to work at the MIT Artificial Intelligence Laboratory and continued to produce substantial works for the Artificial Intelligence database bbc (2020).

Then, in 1952, Arthur Samuel joins IBM's Poughkeepsie Laboratory and begins working on some of the very first machine learning programs. He made the first checkers program on IBM's first commercial computer, the IBM 701 bbc (2020).

In 1967, Donald Michie created a machine that included 304 matchboxes and beads. The machine was using reinforcement learning to play Tic-Tac-Toe. The nearest neighbor algorithm was first introduced in the same year. It was the basis of pattern recognition. In 1995, the Random Forest Algorithm and Support Vector Machines model has found. In 1997, after the IBM computer "Deep Blue" defeated the world chess champion, Garry Kasparov, in a game of a match, AI started attracting the public and enthused researchers to excavate for further improvements bbc (2020).

The successor of Deep Blue in matter of its big impact in the AI field was the great achievement of AlphaGo at Go game. It was developed by DeepMind researchers. AlphaGo started winning against professionals in 2015, then overcame the number two and one players of the game, Lee Sedol and Ke Jie in March 2016 and 2017 respectively. Playing computers and humans, and implementing the Monte Carlo tree search algorithm to figure next moves was the method of training the AlphaGo's neural network.

4.3.2 Three Paradigms in Machine Learning

We can categorized machine learning algorithms in three major topics; supervised learning, reinforcement learning, and unsupervised learning.

4.3.2.1 Supervised Learning

Supervised Learning is the machine learning task of deriving a learning algorithm from labeled training data that includes a collection of training examples. Supervised learning is divided into two parts; Classification and Regression. A classification problem is when the output variable is a category, such as "red" or "blue". On the other hand, a regression problem is when the output variable is a real value (number), such as "dollars" or "weight". In a supervised learning model, input and output variables will be given. Algorithms are trained using labeled data. The model uses training data to learn a link between the input and the outputs. Also, the number of classes is known in this method. Supervised learning is a simple, and highly accurate method when we compare with unsupervised learning. Support vector machine, neural network, linear regression, multiple linear regression, logistics regression, random forest, and classification trees are the most popular supervised machine learning algorithms.

Linear regression analysis is a technique used in statistics for investigating and modeling the relationship between variables. For predictive analysis, linear regression is commonly used. Since the goal is prediction, the main task of linear regression is to fit a predictive model to a training data set of values of the dependent and independent variables. After building the model, the process of using the model is to feed it with unlabeled test data, the fitted model is used to predict the label (dependent variable).

Simple linear regression is a model with a single independent variable x that has a relationship with a response y. This relation is a straight line. This simple linear regression model can be shown as,

$$y = b_0 + (b_1 * x) + \epsilon \tag{4.1}$$

where it has a constant (intercept) value, single coefficient, single independent variable and error component.

For more than one independent variable, the process is called multiple linear regression. This multiple linear regression model can be expressed as,

$$y = b_0 + (b_1 * x_1) + (b_2 * x_2) + (b_3 * x_3) + ... + (b_n * x_n) + \epsilon \tag{4.2}$$

where it has a constant (intercept) value, multiple coefficients, multiple independent variables and error component.

The regression analysis has the power to predict what is likely to happen in the future. It mainly depends on historical data to derive equations used to make future estimation which helps in the decision making process and usually leads to better decisions.

4.3.2.2 Unsupervised Learning

Unsupervised learning is different from other methods because in this method the machine receives only the inputs without labels, and the goal of the model is to extract patterns and draw inferences from the inputs in order to group or classify into clusters to make decisions, predict future inputs, or perform any other desired tasks. The number of classes is not known in this method. Also, unsupervised learning is a computationally complex method when we compare with supervised learning. Cluster, k-means, and hierarchical clustering are the most popular unsupervised machine learning algorithms.

4.3.2.3 Reinforcement Learning

Reinforcement learning is different from the supervised learning problem because in this method the machine not receives the correct input/output pairs. In addition to that, on-line performance is the target, which includes finding a balance between exploration (of uncharted territory) and development (of current knowledge). Machine produces actions (a_1, a_2 , a_3, ... a_n) which is a method of interacting with its environment. By the actions' effect exercised on the environment's state, it leads the machine to obtain scalar rewards (or punishments) (r_1, r_2, ..., r_n). Therefore, the aim of the machine narrows down to focus on maximizing the rewards and minimizing the punishments.

4.4 Load Balancing

4.4.1 Background

Load Balancing is one important concern that can affect the overall performance of the system. Depending on a system, load balancing is a technique to distribute workload throughout the servers, devices, computers to increase resources, utilizing parallelism, improve throughput, increase availability, reduce power consumption and reduce response time. With proper load balancing waiting time can be kept to a minimum which will further maximize the response time. Minimizing the decision time is the most important part of the load balancing. Load Balancing algorithms can be categorized in two different topics: Static Load Balancing, Dynamic Load Balancing.

4.4.2 Static Load Balancing

The decisions related to load balance are made at compile time when resource requirements are estimated. The advantage of Static Load Balancing is the simplicity according to both implementation and overhead since there is no need to constantly monitor the servers, devices, computers for performance statistics.

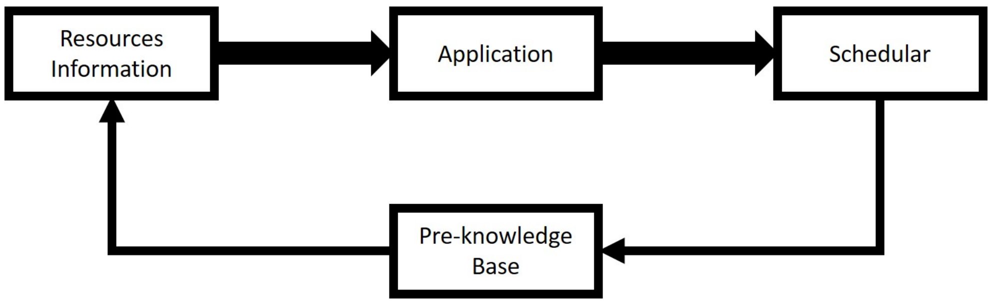


Figure 4.16 Static Load Balancing

Static Load Balancing works correctly only when servers, devices, computers are having low variety in the load. Therefore this algorithm is not well suited for our CPU-GPU Heterogeneous Database Management System.

4.4.3 Dynamic Load Balancing

Dynamic Load Balancing makes changes to the distribution of work among servers, devices, computers at run-time; they use current or load information when making distribution decisions. Dynamic load balancing algorithms are advantageous over static algorithms. But to gain this advantage, the cost to collect and maintain the load information should be considered.

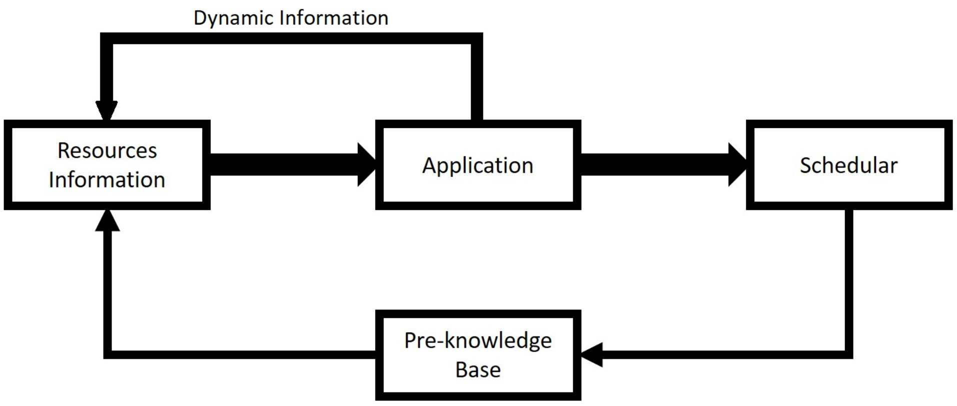


Figure 4.17 Dynamic Load Balancing

5. DOLAP (DATABASE MANAGEMENT SYSTEM)

5.1 Background and General Features

DOLAP has a column-based structure designed with OLAP (On-Line Analytical Processing) analysis queries and tasks in mind. It is aimed that the developed open-source database can be used efficiently with different computing hardware such as CPU, GPU, and FPGA at the same time, with smart work distribution methods. DOLAP can increase performance with query optimization and sharing methods that work in a heterogeneous system on the server-side. It is a database management system where users can generate dynamic and static queries thanks to the interface developed on the client-side. While testing the performance of DOLAP, the computer with the following features was used.

Operating System: Ubuntu 18.04.3 LTS x86_64
Kernel: 4.15.0-58-generic
CPU: Dual-socket Intel Xeon Gold 6152 @ 3.70GHz (44 cores, 88 threads)
Memory: 1030725MiB DDR4@2666MHz
GPU: Nvidia Quadro GV100, 80 SMs, 2048 threads per block, 2 blocks per SM, 32 GB memory
Compilers and Environment: OpenMP for the CPU and CUDA for programming the GPU, gcc 7.5, Python 3, Grafana v6.0

5.2 DOLAP Architecture

DOLAP is a high-performance database written in C ++ language. The developed architecture uses the insertion order as the main index to search on all tables. A record and the fields of this record in the DOLAP are provided by logical links in block numbers. In this method, records with the same index of adjacent blocks as many as the number of columns of the data hold different columns of a record. This scheme allows the data to be passed over one after another to stay close to each other in memory and to be found in the fastest memory (first cache, then memory, then hard disk) when it comes to processing. The following figure shows the data storage and index schema of the DOLAP.

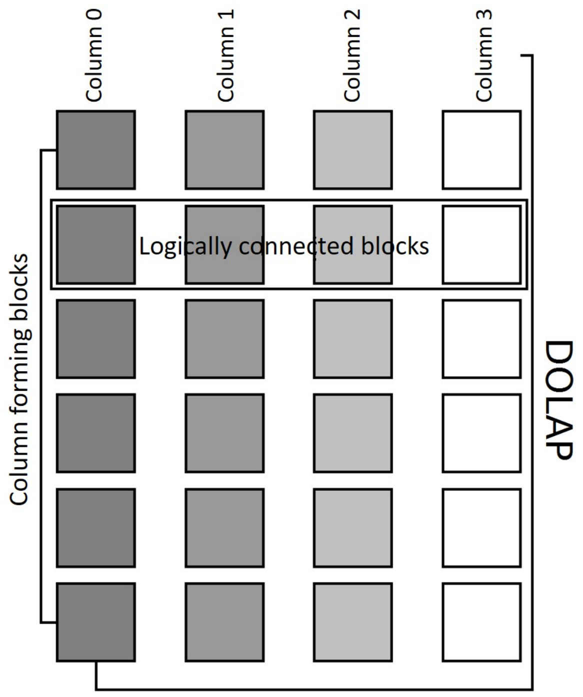


Figure 5.1 The data storage and index scheme of the DOLAP

The DOLAP has an object-based connector in Python to connect to visualization tools and resolve incoming Lucene Queries, and a core engine that holds the database and all its functions, written in high-performance, parallel C++, C and CUDA languages called from that connector. In Python, which is easy to communicate and follow standards, Lucene queries received in JSON format are resolved and sent to the C++ core engine compiled as a shared object by wrappers. In there, queries will be answered on the CPU, GPU, FPGA. The received response is converted to JSON format on the adapter and sent to the visualization tool via the REST interface. In this part of the project, REST interface implementation is not used in the Core Engine as it is not practical and standard. All objects created in parallel and high performance on C ++ are kept as addresses by the adapter and all functions of these objects are called in C ++. Figure 5.2 shows DOLAP architecture.

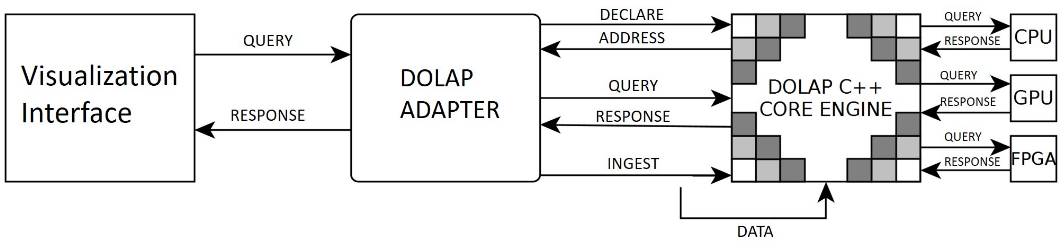


Figure 5.2 DOLAP architecture; Interface, adapter and core engine

DOLAP has been designed and developed to meet the need for real-time or nearly real-time analytical queries on increasing big data. DOLAP, is a column-based NoSQL database that can work on the graphical accelerator, processor, and FPGA, and can distribute the job (load balancing) according to processor capacity for incoming queries. DOLAP, applies Count-Min sketch, Bloom filter, and on-query learning techniques that are not available in other database management systems. The main data structure of the DOLAP is the sketch and bloom filters of the type of a column (int, string, date, etc.), and the blocks with a certain number of records of the corresponding column. This number, BLOCK_SIZE, varies according to the properties of the data and column, especially the data type and data size.

5.3 DOLAP Data Upload

The DOLAP can upload files by ingesting CSV formatted files or adding them to the database in real-time. An algorithmic representation of the process is given in Algorithm 1 in Appendix . The general process has been shown in Figure 5.3. DOLAP, parallelized the rows as column-based and adds blocks with block size elements to the related columns while uploading the file. Each core is responsible for one block at a time, and the number of core blocks are added to the database management system at the same time.

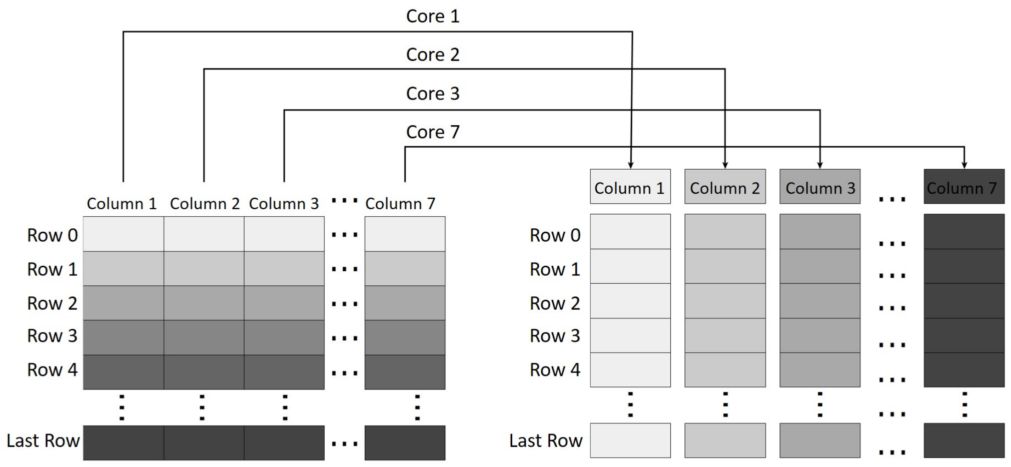


Figure 5.3 DOLAP data loading in column format

5.4 DOLAP Queries

DOLAP is used Lucene query language to connect to the graphical interface. According to the previous queries, DOLAP stores the blocks that are more likely to be crossed in its faster parts. Most probable ones are stored in graphical accelerator memory, less probable ones are stored in system memory and, the least probable ones are stored in storage memory. It avoids unnecessary data overrides by asking the Bloom filter of the relevant block before entering each block after the query arrives. The simple representation of the query process is given at Figure 5.4.

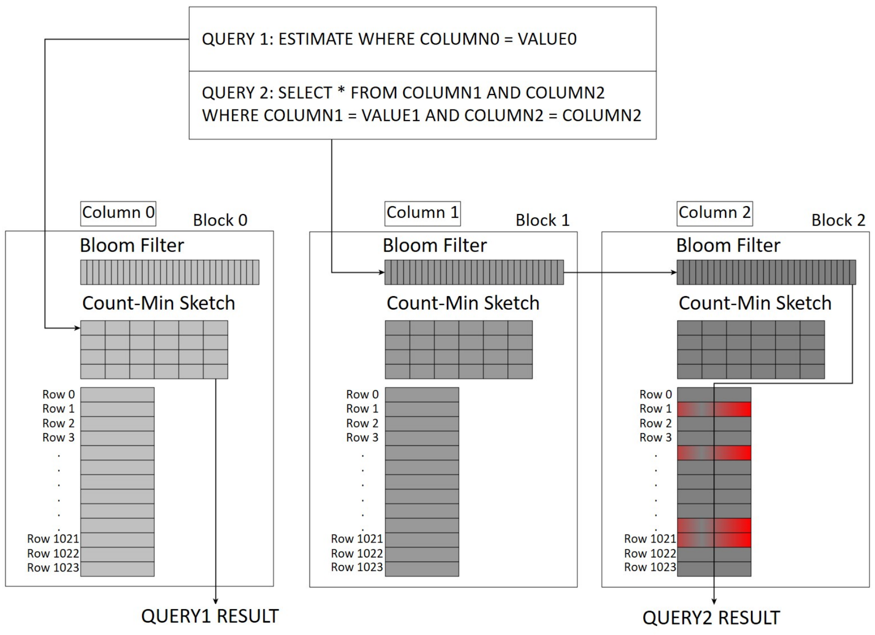


Figure 5.4 Example DOLAP query process

Query types are categorized in order to obtain sharper and more realistic results. Accordingly, there are three query types.

- *Query Type 1* The type of query performed on the entire column without targeting a specific value. (ex., [fare]).
- *Query Type 2* It is a type of query performed on a single specific value in a single column. (ex., [fare, 11] Returns all data with a value of 11 in this column.).
- *Query Type 3* It is a complex query type that consists of a combination of queries with different columns and values. (ex., [fare,11,tips,3]).

5.4.1 DOLAP Bloom Filter

DOLAP uses Bloom Filters to avoid the time it takes to process blocks that do not affect the result of the query in the data it holds in blocks.

The bloom filter is a space-efficient data structure that interrogates the membership of an element to a certain S set Bloom (1970). Figure 5.5 shows the addition and query operations on the bloom filter. The filter uses a bit-array of size m. To add an element x, it uses k independent hash functions that summarize that element in m bits. If the outputs of these functions are shown as $h_1(x), \ldots, h_k(x)$, the positions that appear as 1 in each $h_i(x)$ output are set to 1 in the corresponding row of the filter table. When querying the existence of the element in the set, again, the queried element is summarized with k hash functions. If 1 is seen in all the positions indicated by the hash functions in the filter table, then the response is returned that $x \in S$, ie the element x is in the set S. Otherwise, x is not an element of the S set.

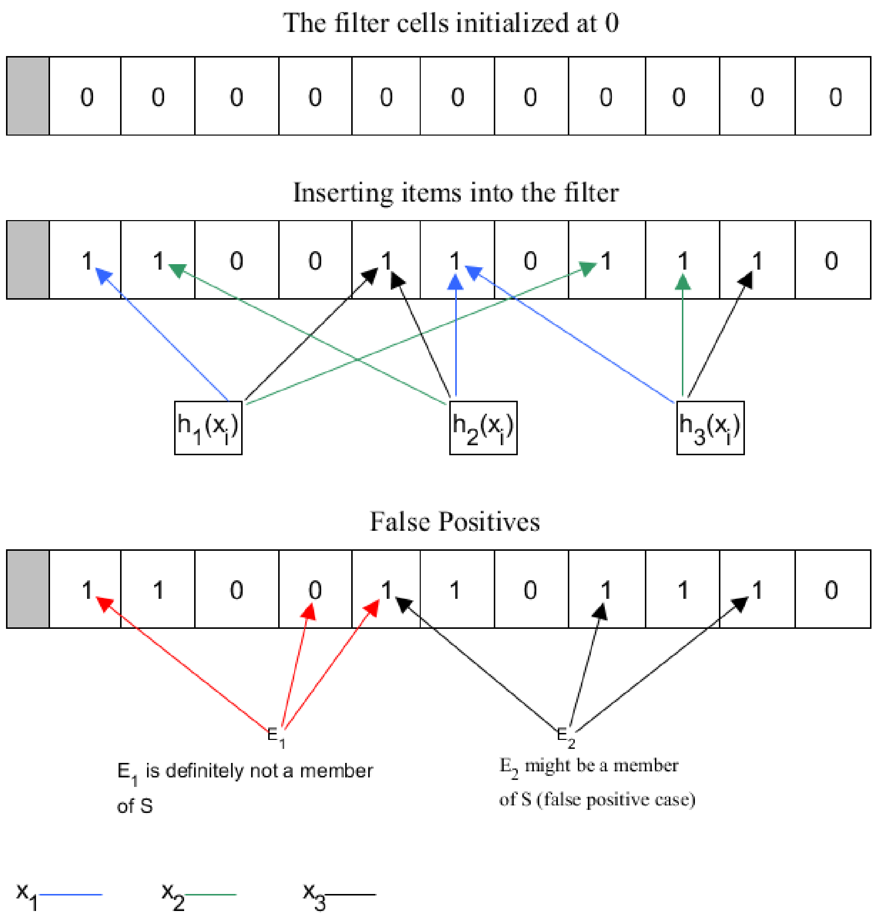

The filter cells initialized at 0
0 0 0 0 0 0 0 0 0 0 0
Inserting items into the filter
1 1 0 0 1 1 0 1 1 1 0
h1(xi)
h2(xi)
h3(xi)
False Positives
1 1 0 0 1 1 0 1 1 1 0
E1
E2
E1 is definitely not a member of S
E2 might be a member of S (false positive case)
x1
x2
x3

Figure 5.5 Bloom Filter

Taking advantage of the concept of polymorphism, a class called "Sketch" is used in DOLAP to create the bloom filter object when a block is created. As mentioned earlier, each block in the DOLAP will have its Bloom filter responsible for answering membership queries to minimize access to the database. Once the structure of a block is created, a Bloom filter object will also be created. As a result, each block will be associated with its bloom filter. Figure 5.6 shows the structure of a block of the DOLAP.

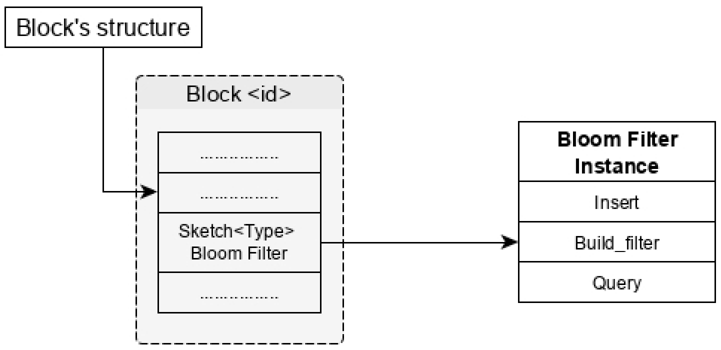


Figure 5.6 Bloom Filter for each block

As expected in theory, Bloom filter query response performance increases, especially as the number of results returned from the database query decreases. When the more specific value is targeted in the database, the effect of searching with Bloom filter on performance increases. This performance increase is completely related to skipping blocks without accessing them. However, it can be seen that if there are a great number of the query-requested value in the database, the effect of the Bloom filter decreases since most of the blocks will contain this answer. Thanks to the bloom filter, an improvement between %20 and %35 (depends on τ value) have been achieved in total response time.

5.4.1.1 Bloom Filter Usage in CPU and GPU Queries

Since the Query Type 1 query retrieves all the data of the specified column, the bloom filter is not used with this query type. In other query types other than Query Type 1, the bloom filter is used to check the query and the desired element in a specific block. In Query Type 2, before the threads retrieve the results from the data blocks, each thread will check the i bloom filter corresponding to the block i. If the filter value entered by the user is not found in this controlled block, that block will be skipped without having to check all 1024 lines.
In answering Query Type 3, DOLAP works in a similar way to the work it does when answering other query types. However, in such queries, because the columns are combined thanks to the **AND** operation, if the k thread that processes the j block corresponding to the i column of the query gets a negative result about the existence of the query value in one of the blocks since at least one of the filtering values is not a member of the queried block, the entire block is ignored and the process continues with the next block to prevent time loss as shown in Figure 5.7.

Scan with dependency and scan with a dependency with bloom filter CPU query answering algorithms are given in Appendix.

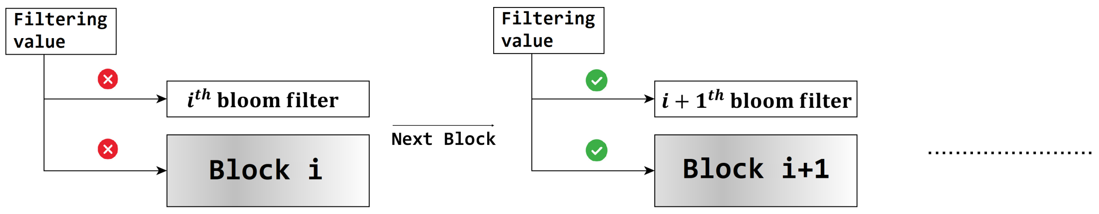


Figure 5.7 Bloom Filter usage for query type 3

5.4.2 DOLAP Query Answering

5.4.2.1 Query Answering on CPU

The DOLAP database has been designed in such a way that the CPU query response algorithm can work with multiple threads in parallel in order to provide better performance when answering queries, especially when answering queries of multiple users. In keeping data in the DOLAP database, each column contains many blocks and each block contains 1024 row records. For the tests performed, a 900 block with 1024 row records each was used.

CPU query type 1 algorithm which is not using the bloom filter is given in 2 in Appendix. For query type 2, scan with dependency and scan with dependency by using bloom filter algorithms are given in Appendix. For query type 3, the process proceeds as described in Section 5.4.1.1.

5.4.2.2 Query Answering on GPU

In the query type 1 queries, as in CPU, the bloom filter is not used in GPU. In the query type 2 queries performed on the GPU, when the Grafana user enters a value to find in the selected column, the GPU kernel begins to process the query and only the column where the query is made is copied to the GPU memory as it is shown in Figure 5.8. The query type 3 query contains all combinations of columns

and searched filter value. In such queries answered on the GPU, the kernel assigns each thread to a row that groups all the columns that meet the values given from a block.
Different cores are designed according to the different column numbers required in the queries. For instance, when a query on 2 columns is processed, more threads and thread blocks are assigned than when a query on 5 columns is processed. This is because when the number of columns queried increases, the number of rows to check decreases. Therefore, less number of threads will be sufficient. Consequently, the protected thread and thread blocks will be simultaneously assigned to answer another user's query.

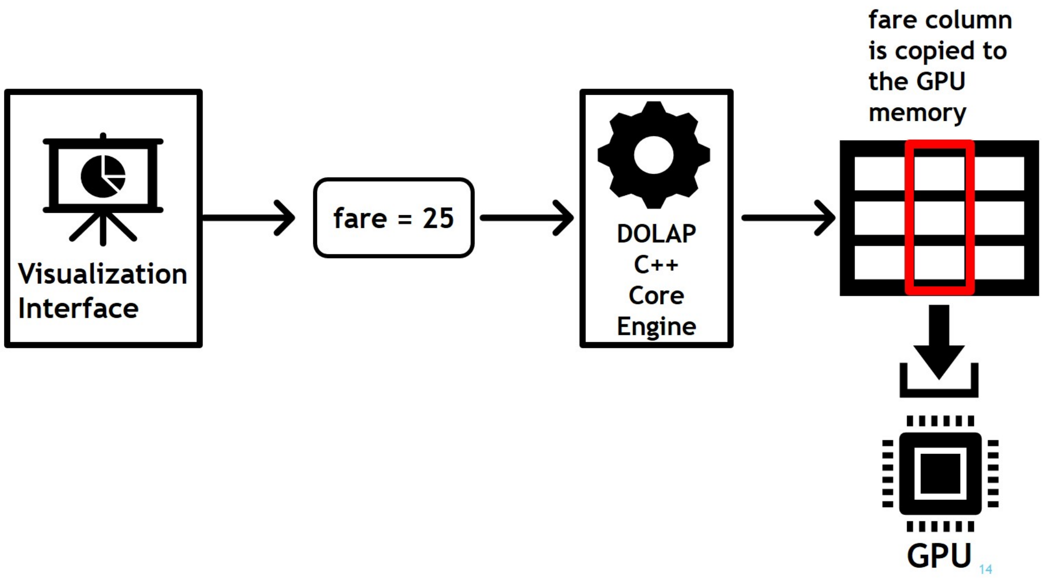
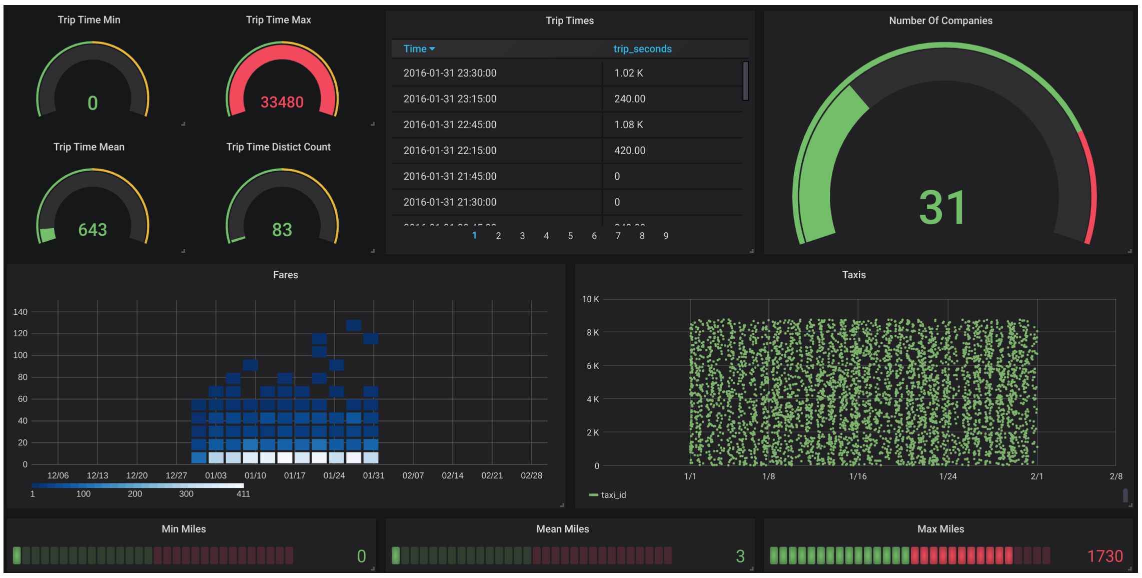


Figure 5.8 Answering Query Type 2 on GPU

5.5 DOLAP Data Visualization

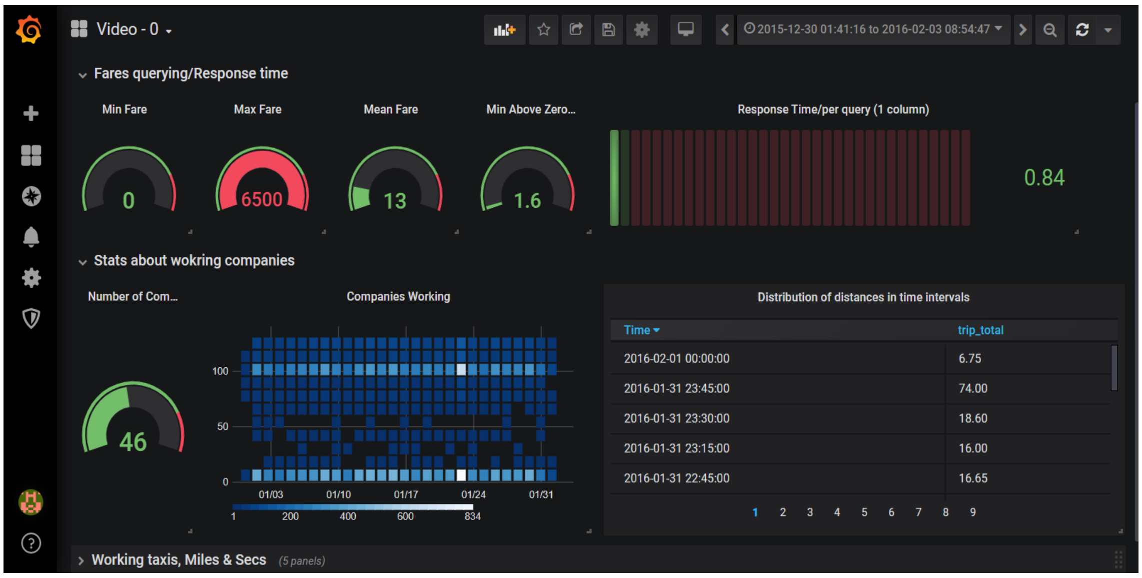
DOLAP uses Grafana for data visualization. Grafana can create visual panels by using the names of the columns reported to it. These panels form the dashboard. DOLAP can send the desired value and the associated time value to Grafana in a single search by using the logical connected block structure to create a time-axis graph of any desired query. It can execute queries for all panels in a dashboard simultaneously and distribute these queries to different hardware devices (CPU, GPU, FPGA) simultaneously.

Algorithm 3 adds the ability to refresh the visualization interface to a certain value, which is very important for a visualization tool, to the DOLAP. In this way, another value of the same record requested by the interface can be sent to the interface within the same query process by using the logical links between the blocks. Example dashboard screen is given in figure 5.13.



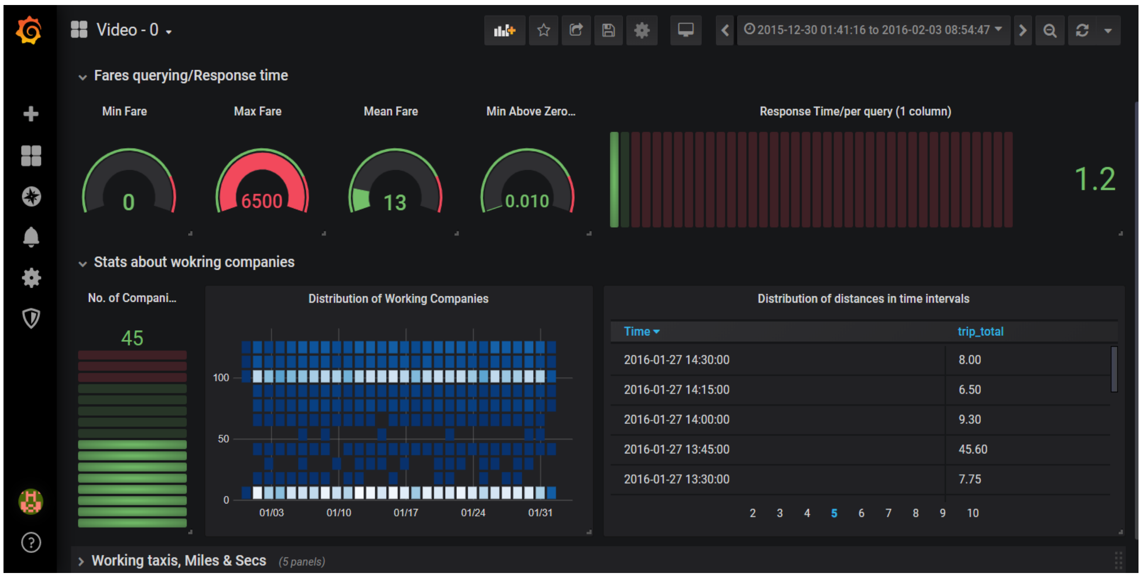
Time	trip_seconds
2016-01-31 23:30:00	1.02 K
2016-01-31 23:15:00	240.00
2016-01-31 22:45:00	1.08 K
2016-01-31 22:15:00	420.00
2016-01-31 21:45:00	0
2016-01-31 21:30:00	0

Figure 5.9 Example dashboard which is created by using DOLAP


Video - 0
2015-12-30 01:41:16 to 2016-02-03 08:54:47
Fares querying/Response time
Min Fare
0
Max Fare
6500
Mean Fare
13
Min Above Zero...
1.6
Response Time/per query (1 column)
0.84
Stats about wokring companies
Number of Com...
46
Companies Working
Distribution of distances in time intervals
Working taxis, Miles & Secs
(5 panels)

Time	trip_total
2016-02-01 00:00:00	6.75
2016-01-31 23:45:00	74.00
2016-01-31 23:30:00	18.60
2016-01-31 23:15:00	16.00
2016-01-31 22:45:00	16.65

Figure 5.10 Different queries answered on CPU (Query type 3)


Video - 0
2015-12-30 01:41:16 to 2016-02-03 08:54:47
Fares querying/Response time
Min Fare
Max Fare
Mean Fare
Min Above Zero...
Response Time/per query (1 column)
0
6500
13
0.010
1.2
Stats about wokring companies
No. of Compani...
45
Distribution of Working Companies
100
50
0
01/03
01/10
01/17
01/24
01/31
Distribution of distances in time intervals
Working taxis, Miles & Secs
(5 panels)

Time	trip_total
2016-01-27 14:30:00	8.00
2016-01-27 14:15:00	6.50
2016-01-27 14:00:00	9.30
2016-01-27 13:45:00	45.60
2016-01-27 13:30:00	7.75

2 3 4 5 6 7 8 9 10

Figure 5.11 Different queries answered on GPU (Query type 3)

As another example of the Query Type 2 query, as shown in Figure 5.12, the filter value is entered for a column. In this example, the value entered is also used for other panels added to the interface.

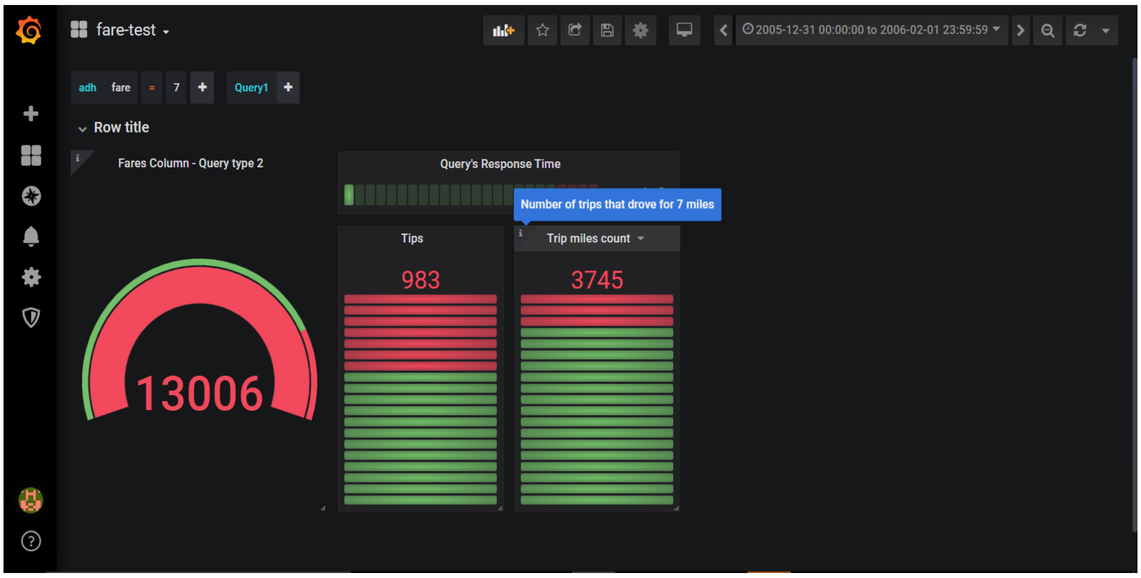
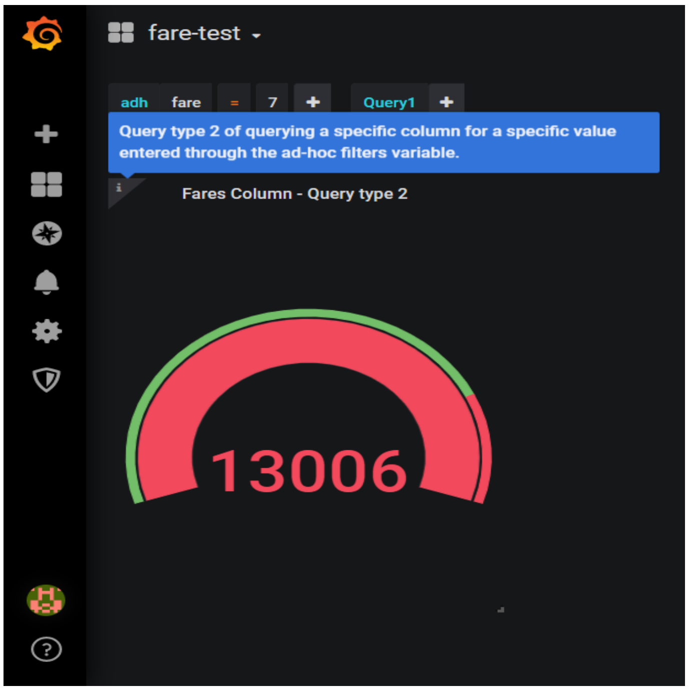

fare-test
2005-12-31 00:00:00 to 2006-02-01 23:59:59
adh
fare
=
7
Query1
Row title
Fares Column - Query type 2
Query's Response Time
Number of trips that drove for 7 miles
Tips
Trip miles count
983
3745
13006

Figure 5.12 Query type 2 queries
(fare = 7, Tips = 7, Trip miles = 7)



Query type 2 of querying a specific column for a specific value entered through the ad-hoc filters variable.

Figure 5.13 Query type 2 query dashboard on Grafana (fare = 7)

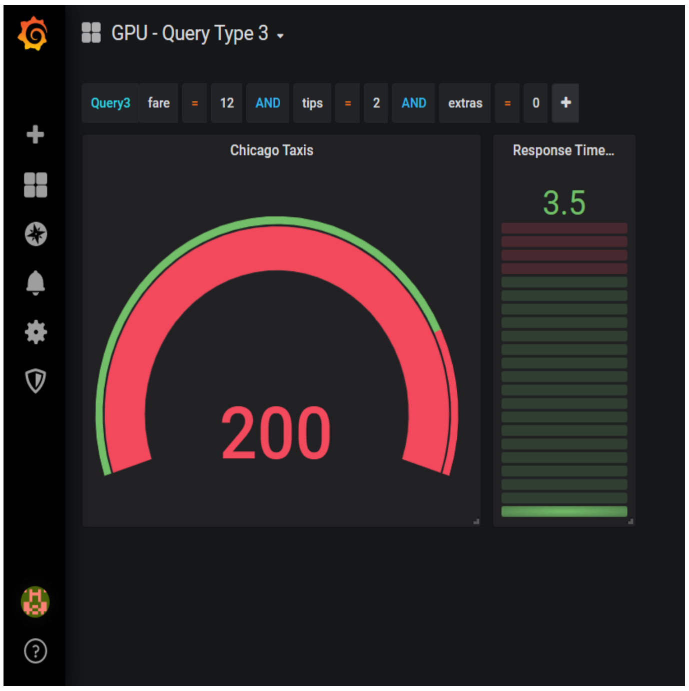


Figure 5.14 Query type 3 query dashboard on Grafana (fare = 12 AND tips = 2 AND exras = 0)

6. DOLAP LOAD BALANCER

The load balancing technique is to distribute the workload among the computing hardware on the server to provide parallelization, increase performance, speed, availability of devices (CPU, GPU), efficiency and reduce power consumption and response time. With a successful load balancing, the response time can be reduced by keeping the waiting time of the users to a minimum.

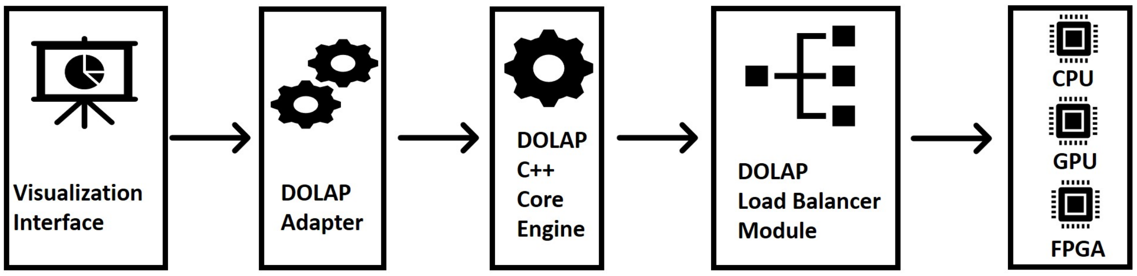


Figure 6.1 General view of the DOLAP

6.1 Methods

Four different load balancing methods have been used and tested. These methods are; random based, algorithm based, algorithm 2 based and machine learning based.

To measure the performance of DOLAP, Kaggle's Chicago Taxi Rides 2016 data set, which contains 20 columns and different types of data structures (uint8_t, uint32_t, double, string), is used Kaggle (2017). A test scenario was prepared with the columns of the data set. The scenario includes 300 mixed queries from query type 1, query type 2 and query type 3 with different columns and values.

6.1.1 Random Based Method

In the hard-coded random method, the decision to send the query to the device queues (CPU & GPU) is given statically, regardless of the query type, average query response time and instant device states. The query is sent to a randomly selected device and answered on it.

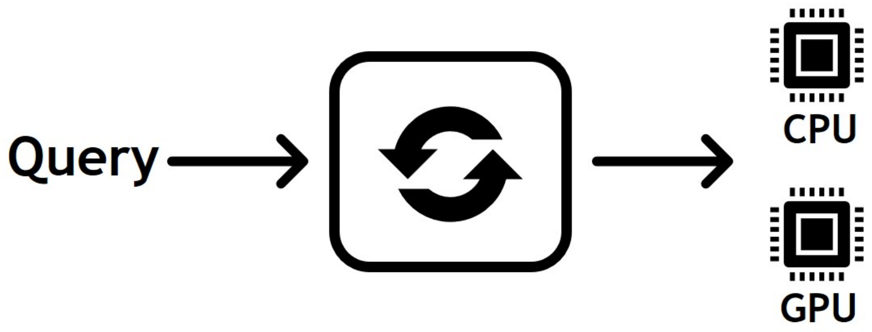


Figure 6.2 Random-Based Method

6.1.2 Algorithm Based Method

As seen in Algorithm 5 in Appendix 9, the load balancer module gets three parameters from the devices. These parameters are; CPU usage, GPU usage and the *dev* parameter that represents the device that answers the current query type faster in the previous query records. For example, for query type 2, the average of the CPU's response time to all query type 2 queries is compared with the average of the GPU's response time to all query type 2 queries, and the device with a lower time average is determined as the *dev* parameter. As the database answers new queries and records the results, the average response times of the devices are periodically updated.

In the algorithm-based load balancing model, the decision is made according to the difference in the usage percentages of the CPU and GPU. If the difference in CPU and GPU usage percentages is greater than a predetermined threshold of τ, the decision will be the device with less usage percentage. If the usage percentages of the two devices are equal, the decision will be to choose one of the two devices at random. If none of these conditions are met, i.e. if the difference between usage percentages is positive but less than τ, the average response times of the devices will be checked and the *dev* parameter calculated from these times will be determined as the decision. If the CPU and GPU usage percentages are equal in Algorithm 5, the load balancing method randomly selects a device between the CPU and the GPU to answer the query, regardless of the current status of the devices. Algorithm 6 was prepared in order to increase the performance on this deficiency detected in Algorithm 1.

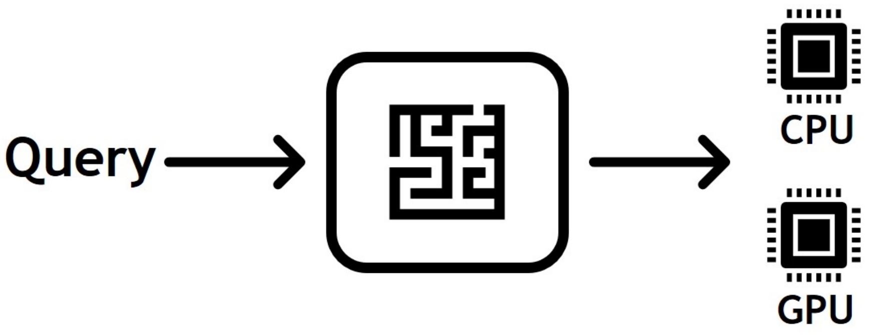


Figure 6.3 Algorithm-Based Method

6.1.3 Algorithm 2 Based Method

As seen in Algorithm 6 in Appendix 9, Algorithm 6 selects the device to be sent to answer the query according to the query type in case the usage percentages of the devices are equal. As seen in Table 7.1, the GPU gives faster results for Query Type 1 and 2, while the CPU gives faster results for Query Type 3. As a result, as seen in Figure 7.2, when Algorithm 2 is used for the same data set, a performance increase has been achieved compared to Algorithm 1.

6.1.4 Machine Learning Based Method

In this project, a dynamic load balancing model (Section 4.4.3) that can calculate by taking into account the current loads of the devices is used.

A queue structure has been created for CPU and GPU. When the query arrives, many parameters, in other words, the current state of the system is controlled by the machine learning model. The machine learning model decides to send the query to the device where it will be answered as quickly as possible.

Multiple Linear Regression (Section 4.3.2.1) method was used as a machine learning algorithm. At the beginning of the study, a single regression model was used. The prediction array includes information of both CPU and GPU devices. Data that includes state information of both CPU and GPU were collected by running the database with random based method and with algorithm 1 and algorithm 2 methods.

Secondly, two regression models have been used for each devices, CPU and GPU. Data was collected for each CPU and GPU by running the database with random based method and with algorithm 1 and algorithm 2 methods. The collected data for CPU includes cpu usage(%), cpu temperature (C) and query type (1,2 or 3). Also, the collected data for GPU includes gpu usage (%), gpu temperature (C) and query type (1,2 or 3).

The first method had a 0.55 R2 score. Total response time for the scenario measured as 450 seconds as an average of multiple runs. The second method's CPU model had a 0.99 R2 score and the GPU model had a 0.82 R2 score. Total response time for the scenario measured as 350 seconds as an average of multiple runs. The second method that has a higher R2 score and lower total response time has been chosen.

For both of the methods, the data has been split and built regression model using all the predictors, P-value has been checked for each of them to find which predictors are contributing to the model and which are not and interpret the result. The alpha value was 0.05 while checking the P-values.

In method 2, before deciding on the device to which the query will be sent, the machine learning model checks the following parameters for CPU,

- $x_1 = cpu\_util =$ CPU usage
- $x_2 = cpu\_temp =$ CPU temperature
- $x_3 = \begin{cases} 1 & \text{if the query type is equal to 2} \\ 0 & \text{otherwise} \end{cases}$
- $x_4 = \begin{cases} 1 & \text{if the query type is equal to 1} \\ 0 & \text{otherwise} \end{cases}$

If both of the x_3 and x_4 are equal to 0, it means that the query type is 3.

also the machine learning model checks the following parameters for GPU,

- $x'_1 = gpu\_util =$ GPU usage

- $x'_2 = gpu\_temp =$ GPU temperature
- $x'_3 = \begin{cases} 1 & \text{if the query type is equal to 2} \\ 0 & \text{otherwise} \end{cases}$
- $x'_4 = \begin{cases} 1 & \text{if the query type is equal to 1} \\ 0 & \text{otherwise} \end{cases}$

If both of the x'_3 and x'_4 are equal to 0, it means that the query type is 3.

These multiple regression coefficients obtained from training data are used in practice as follows: By using current CPU parameters, model's coefficients (b_1, b_2, b_3, b_4) and intercept (b_0), predicted CPU query response time is calculated as follows:

$$y_{CPU} = b_0 + (b_1 * x_1) + (b_2 * x_2) + (b_3 * x_3) + (b_4 * x_4) \tag{6.1}$$

By using current CPU parameters, model's coefficients (b'_1, b'_2, b'_3, b'_4) and intercept (b'_0), predicted GPU query response time is calculated as follows:

$$y_{GPU} = b'_0 + (b'_1 * x'_1) + (b'_2 * x'_2) + (b'_3 * x'_3) + (b'_4 * x'_4) \tag{6.2}$$

Then these times are compared and the query is sent to the device with less estimated query response time. The load balancer adds the current query that has been decided to send to the queue.

7. IMPLEMENTATION RESULTS

Average query response times for two devices and three query types are given in Table 7.1. The bloom filter is not used in query type 1 queries, because the query type 1 is a query that returns an entire column. As seen in Table 7.1, GPU does this operation almost 2 times faster. In query type 2 and 3, the bloom filter is used. In query type 2, still, GPU is faster than CPU but this time difference is not as much as the query type 1. In query type 3, which is a complex query, CPU does the operation faster than GPU.

	CPU	**GPU**
Query Type 1	3.45	1.61
Query Type 2	1.71	1.29
Query Type 3	0.80	1.33

Table 7.1 Average query response times for each query types and both devices

Bloom filter effect

	%5	**%10**	**%15**	**%20**	**%25**
Random	452.54	452.54	452.54	452.54	452.54
Algorithm 1	454.12	457.80	454.91	456.00	454.70
Algorithm 2	437.15	442.37	443.99	442.51	440.09
Machine Learning	350.4	350.4	350.4	350.4	350.4

Table 7.2 Scenario results for all load balancing models with 1 user

In Table 7.2, the results of running the scenario 5 times with different τ threshold values of two algorithm methods are shown. The results shown in the figure are the average values of 5 results. In addition, the random method and machine learning method are also included in the figure. Since the τ value does not affect the operation of these methods, the performance of these methods is constant at each τ value.

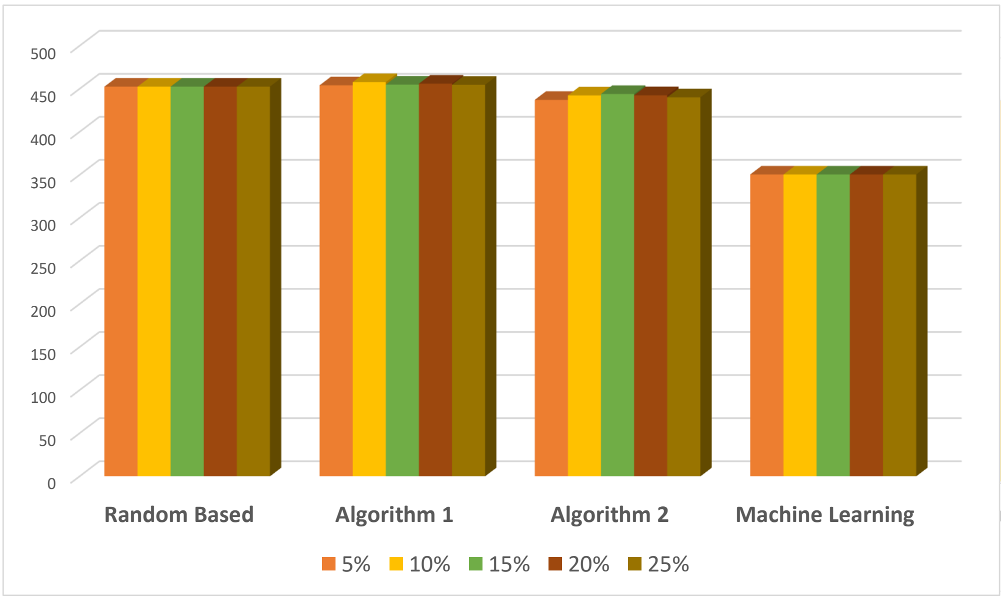


Figure 7.1 Scenario results for all load balancing models with 1 user

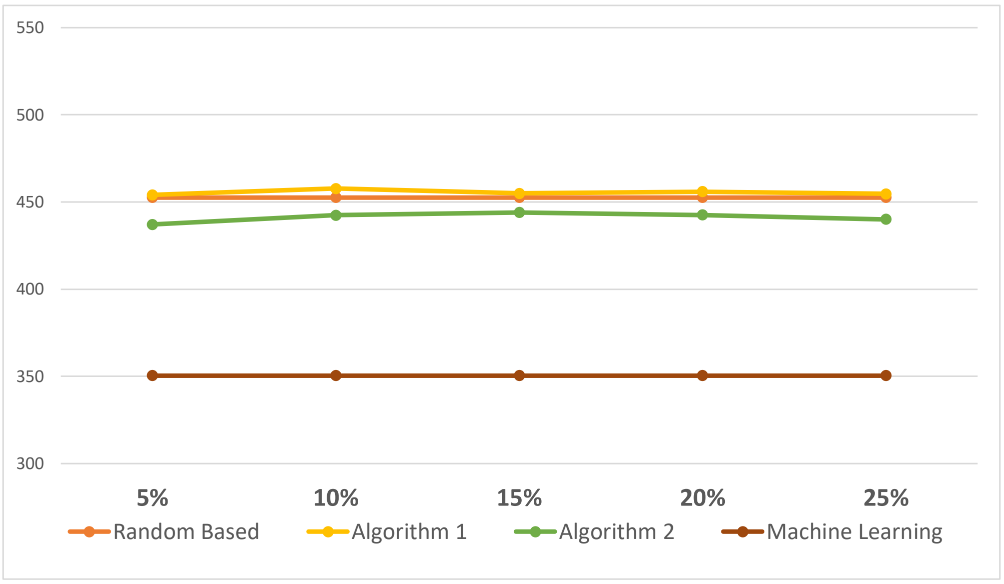


Figure 7.2 Scenario results for all load balancing models with 1 user

As it is shown in Table 7.2, Figure 7.1 and Figure 7.2, Algorithm 2 performed better than Algorithm 1 at all tau values. This is because when CPU & GPU usage percentages are equal, query type 1 queries are assigned to the GPU and query type 3 queries are assigned to the CPU. On the other hand, because Algorithm 5 does not use the preliminary information according to the devices and makes random assignments in case of an equality of usage percentages, in some cases there have been worse results than the random method. Table 7.3 shows the numbers of decisions of Algorithm 1. These are the average numbers of multiple runs.

	%5	**%10**	**%15**	**%20**	**%25**
GPU	88.3	92.5	102.3	94	97.6
CPU	113	111.5	114	115.3	115
Random	98	100	83.6	90.6	87.3
Dev	0	0	0	0	0

Table 7.3 The numbers of the decisions of Algorithm 1

	%5	**%10**	**%15**	**%20**	**%25**
Random	449.71	449.71	449.71	449.71	449.71
Algorithm	427.27	423.23	420.31	419.92	411.25
Algorithm 2	409.76	424.47	406.31	403.14	400.26
Machine Learning	372.37	372.37	372.37	372.37	372.37

Table 7.4 Scenario results for all load balancing models with 2 users at the same time

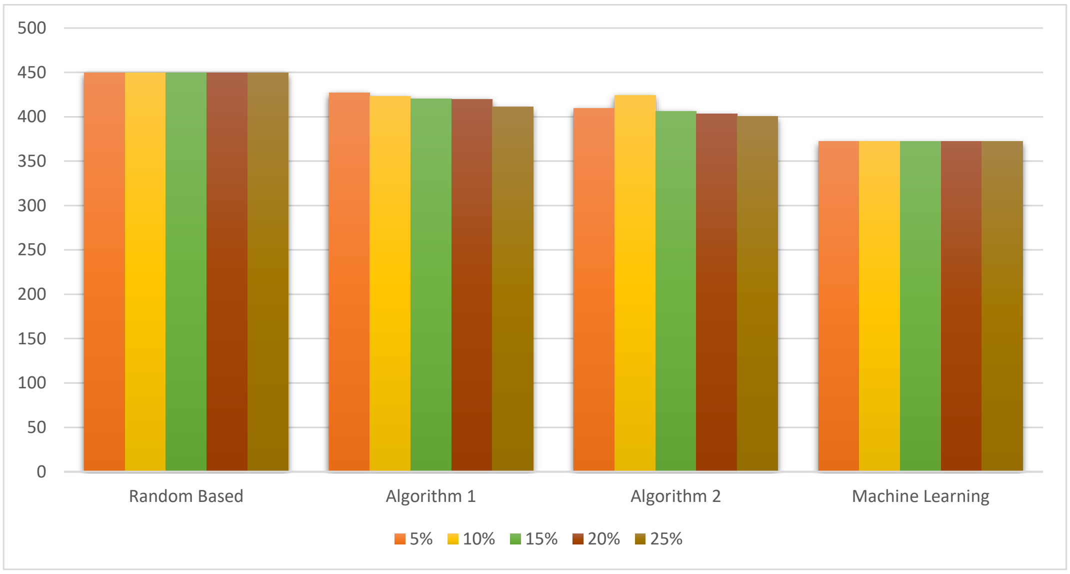


Figure 7.3 Scenario results for all load balancing models with 2 users at the same time

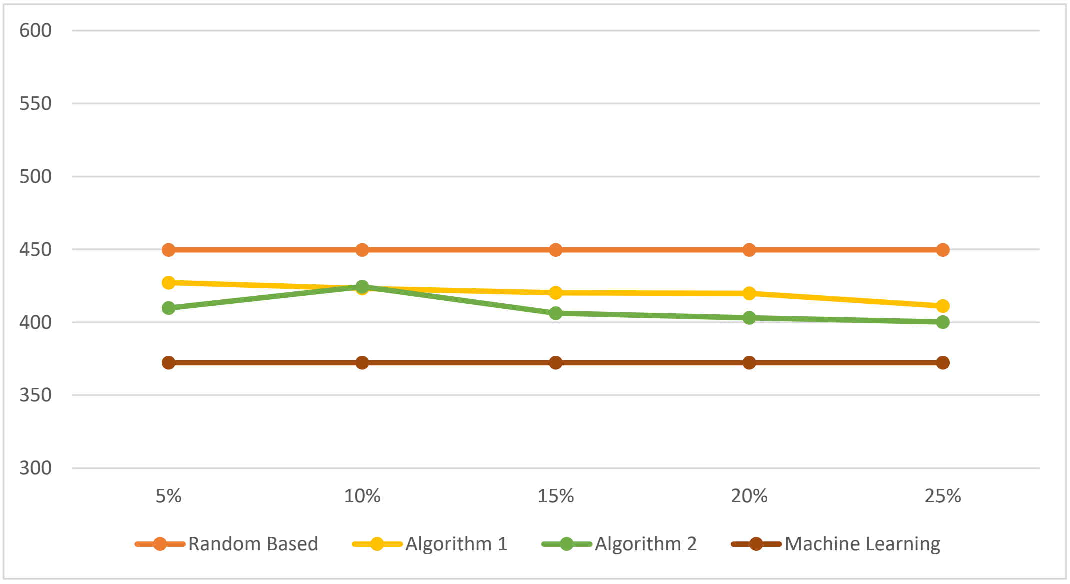


Figure 7.4 Scenario results for all load balancing models with 2 users at the same time

The Algorithm 2 method that is the second faster after the machine learning method, sends an average of 19 of query type 1 queries to the CPU and 24 to the GPU, where machine learning method sends an 0 of query type 1 queries to the CPU and 43 to GPU. Also, it sends an average of 53 of query type 2 queries to the CPU and 46 to the GPU, where the machine learning method sends an 0 of query type 2 queries to the CPU and 99 to GPU. Last, it sends an average of 93.2 of query type 3 queries to the CPU and 64.8 to the GPU, where the machine learning method sends a 158 of query type 3 queries to the CPU and 0 to GPU. Following Figure 7.5 shows the results for each *tau* threshold values.

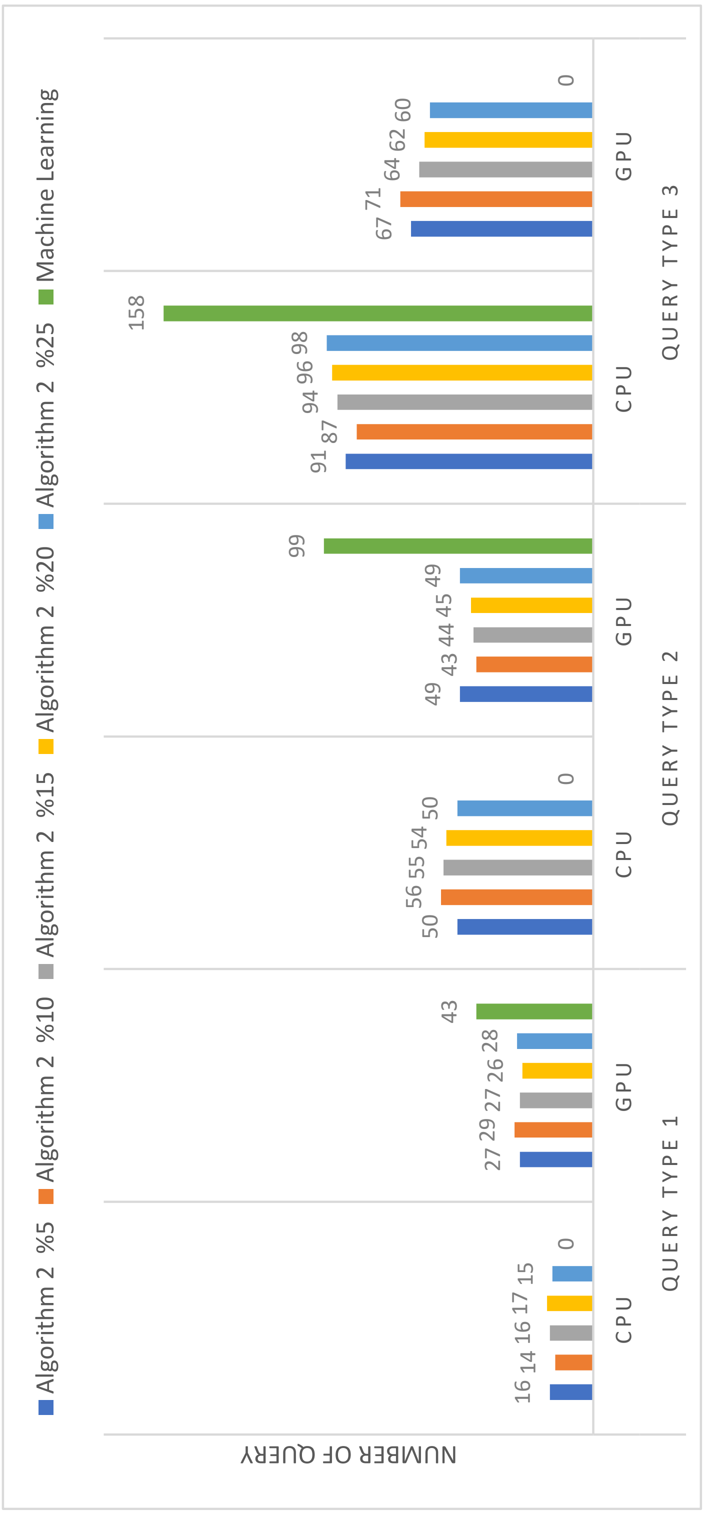


Figure 7.5 Number of query for each device and each query type comparison between Algorithm 2 and Machine Learning methods

It has been shown that the proposed Multiple Linear Regression Machine Learning method has superior performance compared to the other three methods in terms of total scenario response time.

Multiple Linear Regression Machine Learning method is %22 faster than the Random method, %24 faster than the Algorithm 1 method and %20 faster than the Algorithm 2 method in the case of 1 user tests. In case of 2 users at the same time, the proposed method is %17 faster than the random method, %12 faster than the Algorithm 1 method and %9 faster than the Algorithm 2 method.

8. CONCLUSION

In this thesis, it has been shown that the proposed Multiple Linear Regression Machine Learning method is able to learn heterogeneous database management systems dynamics, query and user profiles. Performance has been improved by progressing learning from multiple runs with different algorithms in order to deal with dynamic query and user profiles. It has been shown that the proposed method has superior performance compared to the other described methods in terms of total scenario response time. Multiple Linear Regression Machine Learning method is %24 faster than the Random method, %28 faster than the Algorithm 1 method and %20 faster than the Algorithm 2 method in case of 1 user tests. In case of 2 users at the same time, proposed method is %17 faster than the random method, %12 faster than the Algorithm 1 method and %9 faster than the Algorithm 2 method.

9. FUTURE WORK

The work in this thesis is a part of a project. FPGA hardware and it's query answering algorithms and also a probabilistic differential privacy for the analysis results are planned to be added to this work. When this addition is completed, DOLAP database system will be answer queries on CPU, GPU or FPGA with providing a probabilistic differential privacy guarantee of the analysis results when necessary. These features will make DOLAP unique in the literature.

BIBLIOGRAPHY

Appuswamy, R., Karpathiotakis, M., Porobic, D., & Ailamaki, A. (2017). The case for heterogeneous htap.

bbc (2020). The history of machine learning. `https://www.bbc.com/timelines/zypd97h`. Last Accessed: November 2019.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors.Commun.ACM, 13(7):422–426.

Conn, S. S. (2005). Oltp and olap data integration: a review of feasible implementation methods and architectures for real time data analysis. In *Proceedings. IEEE SoutheastCon, 2005.*, (pp. 515–520).

Firesmith, D. (2017). Multi-core processor. software engineering institute. carnegie mellon university. `https://insights.sei.cmu.edu/sei_blog/2017/08/multicore-processing.html`. Last Accessed: October 2019.

Foote, K. D. (2017). A brief history of database management. `https://www.dataversity.net/brief-history-database-management/`. Last Accessed: November 2019.

Gomez, C., Shami, A., & Wang, X. (2018). Machine learning aided scheme for load balancing in dense iot networks. *Sensors*, *18*.

Grolinger, K., Higashino, W. A., Tiwari, A., & Capretz, M. A. (2013). Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, *2*(1), 22.

Intel. Intel pentium processor. britannica. `https://www.britannica.com/technology/Pentium`. Last Accessed: October 2019.

Intel (2020a). Intel 4004 website. `http://www.intel4004.com/`. Last Accessed: October 2019.

Intel (2020b). The story of the intel® 4004. `https://www.intel.com.tr/content/www/tr/tr/history/museum-story-of-intel-4004.html`. Last Accessed: October 2019.

Kaggle (2017). Chicago taxi rides. `https://www.kaggle.com/chicago/chicago-taxi-rides-2016`. Last Accessed: September 2019.

Kaur, E. & Er.Nishi (2014). A survey on cuda.

Kinetica (2020). Kinetica web page. `https://www.kinetica.com/`. Last Accessed: May 2020.

Kirk, D. B. & mei W. Hwu., W. (2010). Programming Massively Parallel Processors. A Hands-on Approach. 1st ed.

Lazzareschi, C. (1990). Database best seller. `https://www.latimes.com/archives/la-xpm-1990-12-15-fi-5823-story.html`. Last Accessed: October 2019.

Li, M., Zhang, J., Wan, J., Ren, Y., Zhou, L., Wu, B., Yang, R., & Wang, J. (2019). "Distributed machine learning load balancing strategy in cloud computing services," Wireless Networks.

Linderman, M. D. (2009). A Programming Model and Processor Architecture for Heterogeneous Multi-core Computers.

MAPD, O. (2020). "accelerated analytics platform," omnisci. `https://www.omnisci.com/`. Last Accessed: May 2020.

Nvidia (2010). Audi and nvidia. https://nvidianews.nvidia.com/news/nvidia-and-audi-marry-silicon-valley-technology-with-german-engineering. Last Accessed: October 2019.

Nvidia (2017). Audi a8 and nvidia. https://blogs.nvidia.com/blog/2017/07/11/audi-2018-a8-nvidia-barcelona/. Last Accessed: October 2019.

Nvidia (2020). Opencl. nvidia. https://developer.nvidia.com/opencl. Last Accessed: October 2019.

O'Connor, J. J. & Robertson, E. F. (2016). "Pierre-Simon Laplace". School of Mathematics and Statistics, University of St Andrews, Scotland.

Olena (2018). The gpu evolution. https://medium.com/altumea/a-brief-history-of-gpu-47d98d6a0f8a. Last Accessed: October 2019.

Oracle (2007). Oracle. http://www.oracle.com/us/corporate/profit/p27anniv-timeline-151918.pdf. Last Accessed: October 2019.

Praveen, S., Chandra, U., & Wani, A. A. (2017). A literature review on evolving database. *International Journal of Computer Applications*, *162*, 35–41.

Rajesh, L., Bagan, K. B., K, T., & M, M. (2019). Load balancing in heterogeneous network using machine learning technique. International Journal of Innovative Technology and Exploring Engineering(IJITEE).

Rouse, M. (2013). Multi-core processor. https://searchdatacenter.techtarget.com/definition/multi-core-processor. Last Accessed: October 2019.

Rouse, M. (2019a). Dbms. https://searchsqlserver.techtarget.com/definition/database-management-system. Last Accessed: October 2019.

Rouse, M. (2019b). Parallel processing. https://searchdatacenter.techtarget.com/definition/parallel-processing. Last Accessed: October 2019.

Sanders, J. & Kandrot., E. (2010). Cuda by Example. An Introduction To General-Purpose Programming. 1st ed. Boston: Addison-Wesley Professional.

Shi, X., Kindratenko, V., & Yang, C. (2013). *Modern Accelerator Technologies for Geographic Information Science*. Springer US.

Universiy, O. (2020). Dbms history, oxford. "database, n". OED Online. Oxford University Press. (Subscription required). Last Accessed: October 2019.

Warehouse, D. (2020). Oltp vs. olap. http://www.datawarehouse4u.info/OLTP-vs-OLAP. Last Accessed: October 2019.

Wikipedia (2020). Gpu cuda history. https://en.wikipedia.org/wiki/CUDA. Last Accessed: October 2019.

Zheng, X. (2018). Database as a service - current issues and its future. *CoRR*, *abs/1804.00465*.

Data Uploading Algorithm

Algorithm 1: DOLAP: Data Upload from File

```
number_of_row = total_row_number;
number_of_column = file_column_number;
bloom_filter = new Bloom_Filter[number_of_row/BLOCK_SIZE];
CMSs = new CMS[number_of_row/BLOCK_SIZE];
for (d = 0; d <number_of_row) do
    for (l = 0; l <number_of_column) in parallel do
        bloom_filter[d/BLOCK_SIZE].add(value);
        CMSs[d/BLOCK_SIZE].add(value);
        block[d/BLOCK_SIZE][d%BLOCK_SIZE] = value;
```

CPU Algorithms

Algorithm 2: DOLAP: scan(Reporting a column)

```
parameters: Database object address(db), target_column_num,
 time_column_num, block_number, column_number
char** result;
total_element = column_number*BLOCK_SIZE*2;
per_thread = total_element/max_threads;
group_per_thread = block_number/max_threads;

parallel_part{
tid = get_thread_num;
first_group = group_per_thread*tid
last_group = first_group+group_per_thread
result_index = first_group * BLOCK_SIZE *2
for b = first_group; b < last_group; b++ do
    target_block_index = b*column_number + target_column_num;
    time_block_index = b*column_number + time_column_num;
    strcpy(result[result_index], db->[target_block_index]->bring_value());
    strcpy(result[result_index+1], db->[time_block_index]->bring_value());
} return result
```

Algorithm 3: DOLAP: scan_with_dependency

```
parameters: Database object address(db), target_column_num,
  time_column_num, block_number, column_number, query_value,
  query_column_num
char** result;
total_element = column_number$*$BLOCK_SIZE*2;
per_thread = total_element/max_threads;
group_per_thread = block_number/max_threads;

parallel_part{
tid = get_thread_num;
first_group = group_per_thread$*$tid
last_group = first_group+group_per_thread
result_index = first_group $*$ BLOCK_SIZE $*$2
for b = first_group; b < last_group; b++ do
    target_block_index = b$*$column_number + target_column_num;
    time_block_index = b$*$column_number + time_column_num;
    query_block_index = b$*$column_number + query_column_number;
    answer = db->[query_block_index]->bring_value();
    if answer = query_value then
        strcpy(result[result_index], db->[target_block_index]->bring_value());
        strcpy(result[result_index+1],
          db->[time_block_index]->bring_value());
} return result
```

Algorithm 4: DOLAP: scan_with_dependency_with_bloom_filter

```
Bloom Filter
parameters: Database object address(db), target_column_num,
  time_column_num, block_number, column_number, query_value,
  query_column_num
char** result;
total_element = column_number*BLOCK_SIZE*2;
per_thread = total_element/max_threads;
group_per_thread = block_number/max_threads;

parallel_part{
tid = get_thread_num;
first_group = group_per_thread*tid
last_group = first_group+group_per_thread
result_index = first_group * BLOCK_SIZE *2
if db->[query_block_index]->filter_query(query_value) then
    for b = first_group; b < last_group; b++ do
        target_block_index = b*column_number + target_column_num;
        time_block_index = b*column_number + time_column_num;
        query_block_index = b*column_number + query_column_number;
        answer = db->[query_block_index]->bring_value();
        if answer = query_value then
            strcpy(result[result_index],
              db->[target_block_index]->bring_value());
            strcpy(result[result_index+1],
              db->[time_block_index]->bring_value());
} return result
```

Algorithm 5: Algorithm Based Load Balancing

```
Result: GPU veya CPU
dev = deviceWithLeastAvgExecTime()
cpu_util = getCPUUsage()
gpu_util = getGPUUsage()
$\tau$ (Device Utilization difference threshold)
if cpu_util − gpu_util > $\tau$ then
    decision = GPU
else if gpu_util − cpu_util > $\tau$ then
    decision = CPU.
else if gpu_util = cpu_util then
    decision = random(CPU, GPU)
else
    decision = dev
end
```

Algorithm 6: Algorithm 2 Based Load Balancing

```
Result: GPU veya CPU
dev = deviceWithLeastAvgExecTime()
cpu_util = getCPUUsage()
gpu_util = getGPUUsage()
$\tau$ (Device Utilization difference threshold)
if cpu_util − gpu_util > $\tau$ then
    decision = GPU
else if gpu_util − cpu_util > $\tau$ then
    decision = CPU.
else if gpu_util = cpu_util then
    if query_type = 1 then
        decision = GPU
    else
        decision = CPU
    end
else
    decision = dev
end
```