KADIR HAS UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

POWER CONSUMPTION ESTIMATION USING IN-MEMORY DATABASE COMPUTATION

GRADUATE THESIS

MOHAMED ALAMIN

JUNE 2016

Mohamed Alamin M.S. Thesis 2016

POWER CONSUMPTION ESTIMATION USING IN-MEMORY DATABASE COMPUTATION

MOHAMED ALAMIN

Submitted to the Graduate School of Science and Engineering

in partial fulfillment of the requirements for the degree of

Master of Science

in

INFORMATION TECHNOLOGIES

KADIR HAS UNIVERSITY

June, 2016

KADIR HAS UNIVERSITY

GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

POWER CONSUMPTION ESTIMATION USING IN-MEMORY DATABASE COMPUTATION

MOHAMED ALAMIN

APPROVED BY:

Prof. Dr. Hasan DAĞ (Advisor)	[illegible]
Assoc. Prof. Mehmet N. AYDIN	[illegible]
Asist. Prof. Taner ARSAN	[illegible]

APPROVAL DATE: 10/06/2016

"I, Mohamed Alamin, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis."

[illegible]

MOHAMED ALAMIN

KADIR HAS University

Abstract

Graduate School of Science and Engineering
Information Technology

by MOHAMED ALAMIN

In order to efficiently predict electricity consumption, we need to improve the speed and the reliability. Concerning the speed, we use IN-Memory database, which is the best solution that allows manipulating data many times faster than HDD. For this purpose, we need to choose "the best" open-source In-Memory database using a standard benchmark, such as YCSB.

For reliability, we use machine learning algorithms. Since the model performance and accuracy may vary depending on data each time, we test many algorithms and select the best one.

In this thesis, we use SmartMeter Energy Consumption Data in London Households to predict electricity consumption using machine learning in Python and Aerospike in-memory database. The study shows that the best algorithm for our data set is Bagging. We also prove that R-squared is not always a good test to choose the best algorithm.

Finally, we propose a graphical user interface for machine learning using Python, that can be used by non-experienced users to predict the consumption at a certain time.

Contents

List of Figures

List of Tables

Symbols

IMDB	In memory data base
MMDB	Main Memory database
HD	Hard Drive
HDD	Hard Disk Drive
RAM	random access memory
DBMS	database management system
RDBMS	relational database management system
YCSB	Yahoo Cloud Serving Benchmark

Chapter 1

Introduction

The world of technology is growing every day. The development of both software and hardware along with the increase in information creation and sharing of all structures and sizes have engendered the Big Data era. The need for processing large data fast enough for decision-making has increased the need for new solutions, capable of facing this dilemma.

Today, machines create all sorts of data and share it with other machines. The development of social media has increased the size of data generated and communicated on the web. For companies, processing data is a priority. The companies that succeed in the analysis and processing of data to take the right strategic decision have fewer risks of failure. Data storage is a primary concept in big data; the capacities of storage have increased enormously. However, the usage of hard disk (HD) is not efficient enough in the era of big data. When dealing with large data set, HD or solid state drive (SDD) do not accompany the development of speed of new processors and the capacity of data, it is slow and fails to respond timely to all requests when received by millions on a daily basis. The best solution to increase the speed is the in-memory databases. The advantage of using in-memory over HD is that there will be no latency time between the hard disk and RAM.

Companies from different sectors including the energy invest in data analysis for many reasons. In fact, there are many companies specialized in electricity management offering help and assistance for electricity consumption management, based on big data analysis. Their process starts by observing and collecting information on the general consumption of a certain company, in order to understand the electricity consumption cycle and therefore propose different solutions to decrease the consumption. Actually, the need of power consumption increases due to the independence on the electrical devices and technologies in businesses. As for big companies, the high prices of power consumption impose the control and the management of the overall electricity usage. Even though multiple sources of energy are available, they are still expensive and not all of them are applicable especially for companies that use electricity in a massive way. As a result, finding solutions to decrease the power consumption prices became a new business.

Using machine learning algorithms, we can determine the peak hours and estimate the consumption in future for a better management of the energy resources. Generally, the prices of the electricity depend on time; the electricity is always more expensive in peak times. In order to avert the need of using more generators, it is essential to try decreasing the consumption in those peak hours. For example, by turning off the unnecessary lights or the heating and cooling systems during these periods.

In this thesis, we chose the use of in-memory databases since they represent the future of fast data processing.

1.1 Related Work

There are some related articles concerning the prediction of power consumption, such as: "Electric Load Identification using Machine Learning" [10], "Home Electricity Forecasting" [11], "Energy Prediction Based on Resident's Activity " [12] and "Online Machine Learning Algorithms For Currency Exchange Prediction" [13] as a report . All those articles among many others talk about power prediction and using algorithms to find best one in the prediction. But none of them has used in-memory databases or obtain the Bagging Tree Regression as the best choice to predict power consumption. Our study will be the first for prediction using in-memory databases.

1.2 Research Questions

In this thesis, the purpose is not to explain what In-memory database is and how it works, but to show how it is possible to use it in the real life and demonstrate how In-memory databases are faster than database management system. For those reasons, we need to respond to many questions, including the followings:

- What is the architecture of in-memory databases?
- Can we compare between In-memory database systems?

- What is the best model to use for the prediction of power consumption used in-memory?
- Can we use the in-memory database to store power consumption data?
- Can we use the in-memory database with some models to predict power consumption in a short time?
- Is there a way to develop a solution integrating both machine learning for prediction and in-memory databases?

1.3 Scope

The scope of this research will focus on three sides. The first is about in-memory databases and how is work; the architecture, the advantages and the disadvantages. The second side is concerned about the usage of a power consumption data set in-memory and trying to understand when people consume a lot of power and when they do not. The last part is about machine learning and its usage to predict power consumption.

1.4 Approach

We are going use some tools and softwares to accomplish our goal, as following:

- Aerospike database: this software is in memory database that stores data in memory.
- SmartMeter Energy Consumption Data in London Households as a dataset.
- Python programing language and its packages (sklearn, numpy) for prediction and plots.

1.5 Thesis outline

The Chapter 2 will explain the architecture of In-Memory database.
Chapter 3 will establish the prediction of electricity consumption using both machine learning and an In-memory database. To choose the database, comparison between some In-memory open-source systems is elaborated. At last, a graphical user interface is developed using Python for prediction.

Chapter 2

In Memory Database Architecture

2.1 Introduction

In the last few years, the In-Memory Technology has been used at a large scale in many fields. Big companies, in particular, have shown a lot of interest towards In-Memory technologies, ever since the need for fast data processing has increased dramatically. These days, with the impact of the Big Data era, there is a big need for processing data at a large scale and fast enough for a real-time or a near real-time response. The need for new technologies that can process data quickly has become a necessity. In-Memory database (IMDB) or Main Memory database (MMDB) is a database that stores data in the main memory (RAM). In conventional databases, the data goes from the HD to the RAM, and then it is processed by the Processor. However, within the IMDB, the data will reside in RAM instead of HD. This means that the data would take a short cut directly from RAM to be processed.

The IMDB contains two main parts: the Main Store and the Buffer store. It has also a backup system on the hard disk or the SSD of flash memory to restore data if the system crashes; that is called the Durability Store. In addition, it has one or more types of data structure store (key value) and indexes. Using IMDB can demonstrate many benefits, especially in speed. The stream of data when using IMDB decreases in size. The main memory can present a solution since it is by nature faster than the hard disk. The index structure of the hard disk aims to minimize both the number of disk accesses and the disk space; whereas the index structure of the main memory doesn't need to minimize any disk accesses since it is contained in main memory. 1 However, the implementation of a solution based on RAM is a challenging operation. Each memory has its own specificities. We shall describe in the upcoming part the differences between the memories.

To start with, L0 the register is the smallest one, but it is also the fastest. The latency time or the response time from first layer L1 cached data word is 0.5 nanoseconds. However, it can only hold one word. The second layer, L2 holds one line and it delays 7 nanoseconds, the delayed time from the main memory to the registry is about 100 nanoseconds (0.1 Microseconds), Solid state drive SSD takes 150 microseconds (150,000 nanoseconds)such as in figure 2.1,

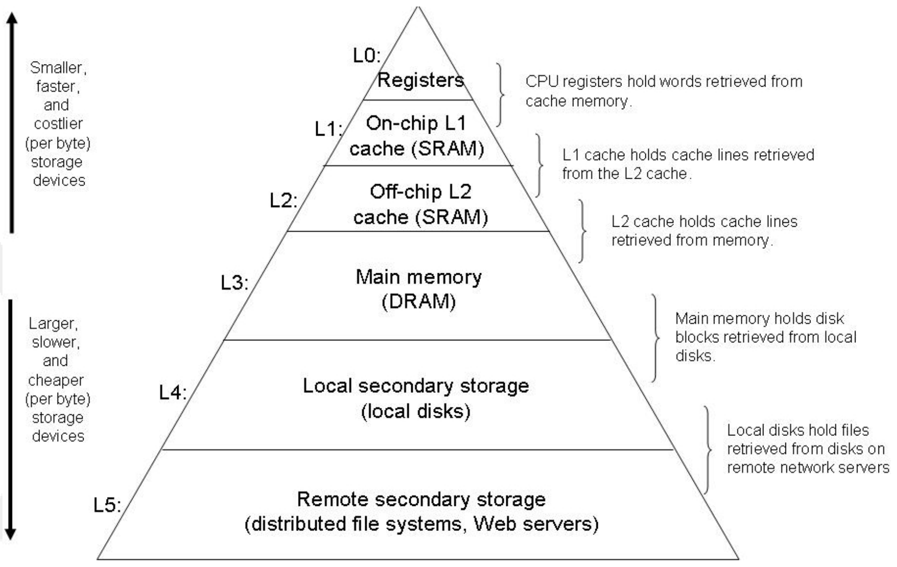
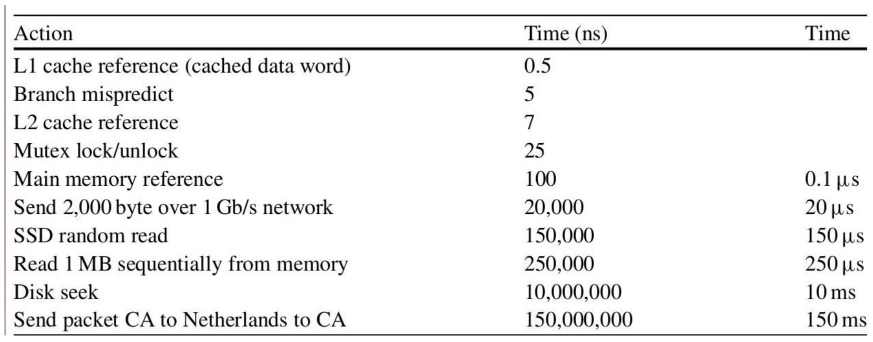

Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Registers
L1:
On-chip L1 cache (SRAM)
L2:
Off-chip L2 cache (SRAM)
L3:
Main memory (DRAM)
L4:
Local secondary storage (local disks)
L5:
Remote secondary storage (distributed file systems, Web servers)
CPU registers hold words retrieved from cache memory.
L1 cache holds cache lines retrieved from the L2 cache.
L2 cache holds cache lines retrieved from memory.
Main memory holds disk blocks retrieved from local disks.
Local disks hold files retrieved from disks on remote network servers

FIGURE 2.1: Memory Hierarchy [1]

On the other hand, the hard disk drive takes about 10 milliseconds (150,000,000). As we can see, the difference between HDD and SSD is about 9,750,000 nanoseconds. For this reason, many computer companies started to use the SSD instead of HDD. However, there is still not enough processors to process a large amount of data in a short time. Hence, there is no way to compare between the HDD and the Main memory in terms of speed. Still, the main memory is small and more expensive compared to the HDD. These technologies were used widely and developed constantly in the mid of 2000s due to the increasing capacity of data storage. The dropping prices of these technologies have encouraged the companies to obtain and invest in the in-memory technology. To illustrate this fact, in the late 1990s 1 gigabyte of main memory used to cost about a 1000 dollars. Today, you can get the same 1 gigabyte for less than 5 dollars.



Action	Time (ns)	Time
L1 cache reference (cached data word)	0.5	
Branch mispredict	5	
L2 cache reference	7	
Mutex lock/unlock	25	
Main memory reference	100	0.1 µs
Send 2,000 byte over 1 Gb/s network	20,000	20 µs
SSD random read	150,000	150 µs
Read 1 MB sequentially from memory	250,000	250 µs
Disk seek	10,000,000	10 ms
Send packet CA to Netherlands to CA	150,000,000	150 ms

FIGURE 2.2: Latency numbers[2]

2.2 Data store

A data store is used for storing and managing a set of data that is involved in a particular structure. Generally, there are two types of data store: SQL and NOSQL or Not SQL data store. SQL databases are also named Relational database since they are well structured.

2.2.1 SQL Data Store

SQL stands for the relational database management system. SQL is a type of database management system (DBMS) that stores structured data in related tables. The most commonly used DBMS are the RDBMS's.

The relational DBMS were originally created by Edgar F. Codd from IBM in 1970. The first commercial and available RDBMS was Oracle, which was accessible to the public in 1979. SQLs are used extensively and considered powerful for the reason that they require fewer assumptions about how data is related and how it will be extracted. Most of today's database applications are relational, either commercial such as Oracle Database and

Microsoft SQL or open-source like MySQL and SQLite. The structure of a relational database implies the usage of tables for data storage in addition to the creation and the management of relations between the data. In addition, a relational database can consist of a single table in this case called a flat-file database, or many tables interrelated.

When, Edgar F. Codd, a British computer scientist with IBM, published a paper back in 1970 titled "A Relational Model of Data for Large Shared Data Banks". Very few paid attention to this topic and no one could predict that his rules and vision for the RDBMS would have a huge impact on the future of data storage.

The basic rules for relational data storage can be described as the following:

- The data must be stored and presented as relations. The data must be stored in tables that have relationships with each other, using primary/foreign keys.
- The system should provide relational operators in order manipulate the data stored in tables; which means codes that enable to test the relationship between two entities. As an example, the WHERE clause of a SELECT statement, i.e., the SQL statement **SELECT * FROM CUSTOMER WHERE CUSTOMER_SURNAME = "Smith"** will query the CUSTOMER_MASTER table and return all customers with a surname of Smith.

Codd later published another paper specifying 12 rules that all databases must follow to be called relational. Even though many modern database systems do not follow all 12 rules, still they are considered to be relational because they conform to at least the two previously mentioned rules.

The popularity of the RDBMS is due to a number of advantages that it represents, which mainly are:

- Security and privacy: possibility of access limitation at different levels
- Ease of use and manipulation: creation, addition, elimination and modification
- Respect of data integrity by limiting the redundancy
- High flexibility due to the independence of the tables
- Easy data query
- Easy data recovery

The disadvantages of RDBMS can be resumed in the fact that RDBMS were created to handle structured data, meaning that they can't handle unstructured data correctly. Also, if the data tables are large the relations complex, data queries can take a lot of time to respond. In order to store large dataset in RDBMS, powerful servers are required, which are expensive and difficult to handle.

2.2.2 No SQL Data Store

NoSQL data stores became popular in the last decade for many reasons. First, the large amount of unstructured data generated every day increased the need for an appropriate data store. The development of mobile applications and social media, led to the creation of different types of data: images, videos, tweets, etc.

The traditional relational database system uses tables for data storage, with relations between the data and the possibility of complex SQL queries. However, RDBMS fail to meet the need when it comes to large databases and unstructured data. The need for managing this type of data and improving the performance implied the development of a different type DBMS.

As described previously, in a relational database, the data is stored in a way that it can be retrieved easily, since it uses tables to organize the data. In fact, one table can only handle data with the same type. The RDBMS is very advantageous and is easy to use but it cannot be a suitable solution to all the situations.

The relational databases were for a long time the first choice of companies. However, with big data, the limitation of RDBMS have surfaced. The new solution developed to overcome those limitations consists of the NoSQL databases. In fact, NoSQL is not a new solution, but it gained its popularity after the introduction of the database as a service. In reality, the NoSQL databases are used generally with the RDBMS and not as a replacement. NoSQL manages better high scale data and deals with instructed data. In addition, NoSQL can execute queries at a high speed using a volatile memory.

The choice of using either SQL or NoSQL database or both depends on the characteristics of the business and the data: type, volume and business needs. NoSQL databases were developed to increase their scalability, using a different architecture and removing some features of RDBMS. NoSQL is largely used today, notably in Google (BigTable, LevelDB), Twitter (Hadoop HBase, Cassandra), Facebook (Cassandra), Netflix (SimpleDB, Hadoop HBase, Cassandra), etc.

The NOSQL data model does not confirm to the ACID properties (Atomicity, Consistency, Isolation and Durability) which are specific characteristics of the RDBMS. Instead, NOSQL guarantees BASE properties (Basically Available, Soft state, Eventual consistency).[14].1 It means that with NoSQL, the data is highly available even in large scale, but without a guarantee of consistency.

To sum up, the relational databases are the best solution when it comes to structured data. However, for the big volume of data, especially when it is instructed, the use of NoSQL is more favorable. The aggregation of the data at larger scale is easier and consumes less time with NoSQL.

There are different types of NoSQL databases including key-value store, Column store, Document store and Graph store.

2.2.2.1 Key-value store

The key-value data store is one of the simplest NoSQL architectures, yet it is an efficient model that provides many functionalities. It stores values with indexes in order to be retrieved using key values. Key-value stores handle both structured and unstructured data. The key-value based systems provide scalability, due to the key distribution over nodes[15]. Some key-value data store can store data in RAM or on disk. For its functionalities, Key-value data store usage is common in e-commerce websites, advertisement and forums.

Examples of databases using the key-value data store model include Cassandra, DyanmoDB, Redis, Riak, Memcached, FastDB and Azure Table Storage.

The key-value data store model consists of two parts2.3. The first one is the data, stored as values; the second part is the key where they are stored as indexes. The user can request the value by calling the keys like maps. Data can only be retrieved using the keys and every key corresponds to a certain value.

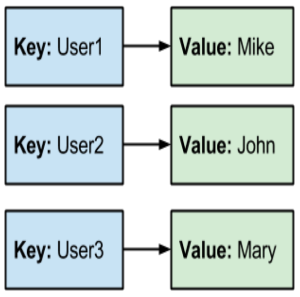
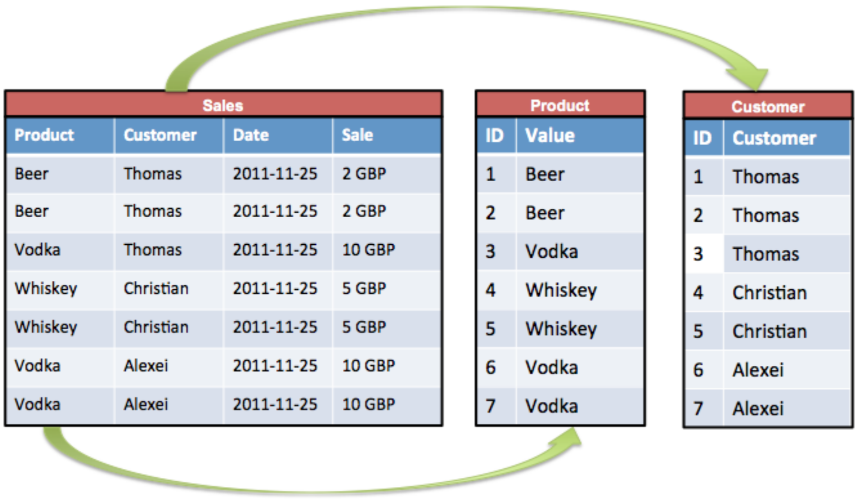


FIGURE 2.3: key valuse[3]

2.2.2.2 Column-Oriented Store

Column-Oriented stores store data in columns instead of full tables in order to save time in queries. The model is composed of columns; each column contains the keys and the correspondent values. It provides a rapid query response and thus a higher scalability than the classical RDBMS tables or a row-Oriented Store2.4.

The Column-Oriented Store model is very useful if the need of visualizing records is not often required. In fact, retrieving a single record and its modification can be a slow process since every instance data must be retrieved from the ensemble correspondent columns. Besides that, the Column-Oriented Store presents a high possibility of compression and a fast queries execution when they are column based. Examples of databases using column-oriented stores are HBase, MonetDB, LucidDB, Hekaton and eXtremeDB.



Sales

Product	Customer	Date	Sale
Beer	Thomas	2011-11-25	2 GBP
Beer	Thomas	2011-11-25	2 GBP
Vodka	Thomas	2011-11-25	10 GBP
Whiskey	Christian	2011-11-25	5 GBP
Whiskey	Christian	2011-11-25	5 GBP
Vodka	Alexei	2011-11-25	10 GBP
Vodka	Alexei	2011-11-25	10 GBP

Product

ID	Value
1	Beer
2	Beer
3	Vodka
4	Whiskey
5	Whiskey
6	Vodka
7	Vodka

Customer

ID	Customer
1	Thomas
2	Thomas
3	Thomas
4	Christian
5	Christian
6	Alexei
7	Alexei

FIGURE 2.4: Column store[4]

2.2.2.3 Document Store

The Document-Oriented Store model stores data as documents in a standard format, usually encoded in XML, YAML, JSON or other formats. They are a specific type of key-value stores. Document Stores have a schema-free organization, which allows more freedom in manipulating the data. The data is described in the documents and from there it can be modified or deleted. The data doesn't necessarily have to be in the same structure, See the figure 2.4. The advantage of the document store is that it provides the possibilities of complex indexing. Examples of databases using Document Store are Couch DB and Mongo DB2.5.

2.2.2.4 Graph data store

The Graph Data store database stores the data in a graph format. The graphical representation reminds of the classical RDBMS design, but the graph is more flexible and

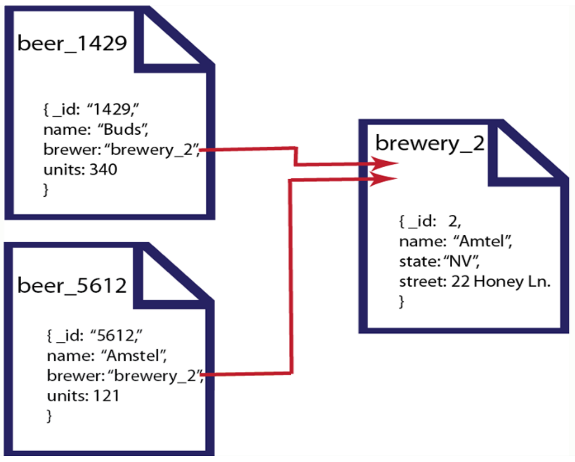


FIGURE 2.5: document store[5]

scalable. The structure consists of nodes and edges, in addition to properties describing the relations between the data. Node: is the entity or the record. Every node must point to another node. Edge: is the relationships connecting the different nodes. Properties: is the information that specifies the type of relation between the nodes2.6.

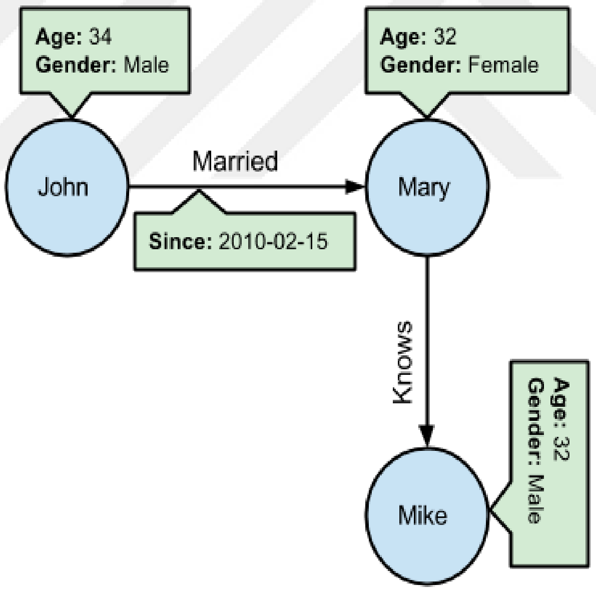


FIGURE 2.6: document store [6]

Graph data store are used when there is a need of describing the relation between the nodes, like in social media, recommendation systems and security. The databases using this type of store include Neo4j, InfoGrid and Orient DB that also uses Document store and key-value store.

2.3 Main Store

The main store or the data store is a repository and consists of the biggest part of the system that stores all the data in the main memory. In the traditional databases using the hard disk, any operation like join, find or aggregation is done in Pool buffer in in-memory and the changes will be saved in the database that is found in HDD. However, in the main memory (also knows as the Volatile memory), the main store is used to store all data. In this main store, we can read only (such as in figure 2.7). In fact, the main store is not touched by any data modifying operation, all the changes (update, delete and addition) is done in another part of main memory called the buffer store.

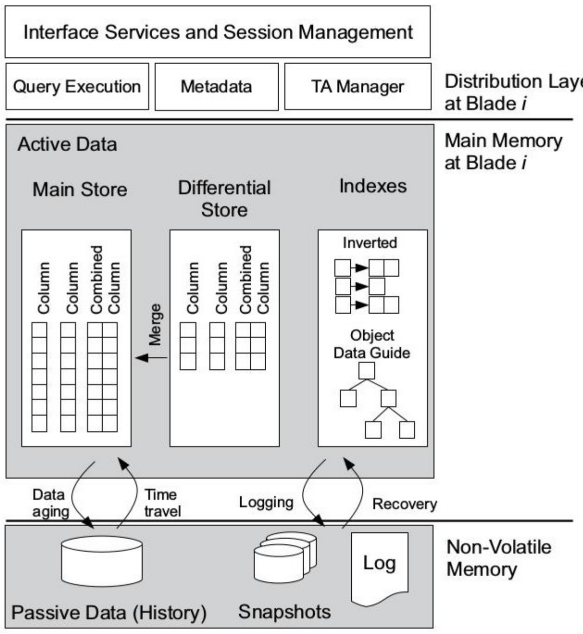


FIGURE 2.7: The memory architecture of SanssouciDB [7]

2.4 Differential Store

The differential store or the buffer store is a work place within the main memory that is used to update, delete and add data to the database. It keeps the data safe in main store and avoiding by that the overlaps when modification is happening.

The data stored in the main and the differential store may be subject to a compression, depending on the type of data store. The column, row and key-value stores accept compression. This compression is also called a dictionary. When an update occurs at the level of the buffer store, there is an operation called marge that synchronizes the data from buffer store to the main store when a lock stops the I/O data from the processor to the main store. Some relational databases management systems offer in-memory database by keeping all data in Pool buffer. Still, this happens when the memory can obtain all data to pool buffer like SQLite database, which works as RDMS but it also uses its Pool Buffer as an in-memory database.

As for my personal experience, the operation works only for one time; if we use python to call it, we can't use another API like R at the same time. In addition, if we use it from python environment, we can't call our database using another python environment.

2.5 Compression

The capacity of the main memory has increased in a fast manner within the last years. However, the main memory is still expensive especially if there is a need of many gigabytes to store large amounts of data. For that reason, some in-memory database starts using the compression in order to reduce the data size.

Yet, not all data store are suitable for compression. For the row and the column-oriented data stores, compression is possible and simple. The compression operation consists of compressing all equal values in one and storing them in a dictionary for the values. The dictionary stores the integer values instead of the real values in length string, thus saving memory and increasing the performance by improving the speed of queries. Consequently, this method is only feasible if the data contains similar values. Moore's law states that computing capacity of processors doubles every 18 months and this has shown to be valid for more than 35 years. Even though, this doesn't really solve the problem, since this law doesn't apply to the development of hard disk and the main memory access speeds. The imminent problem today concerns the delay in I/O. In order to reduce this delay, the new strategies try to develop the compression capacities, for example, to enable the retrieval of the data by accessing the values instead of decompressing the entire data.

The objective of all types of compression is to liberate space by eliminating redundancies. In the context of in-memory databases, the objective is also to reduce the data transferred between the main memory and the CPU. There are many types of compression; the most ones are the Light-Weight Compression and the Heavy Weight Compression.

2.5.1 Light-Weight Compression

There are many types of Light-Weight Compression; the dictionary is the most famous one. It is called light because it doesn't require extreme computations. It consists of assigning a symbolic value to represent the most frequent values. This method makes also the queries faster. Decompressing is also simple and only requires looking up the dictionary to retrieve the original values. For in-memory databases, the Light-Weight is the more appropriate than the Heavy-Weight Compression.

2.5.2 Heavy-Weight Compression

The Heavy-Weight Compression include many techniques like Huffman encoding and arithmetic encoding. It consists of reducing the value to its entropy. Even if it compresses the values perfectly, this type of compression demands large computations, which makes it more suitable for the classical DBMS[16].

The choice of a compression method depends on the data store type. In all cases, the decompression of the entire data should be avoided as much as possible.

2.6 Index

The concept of the index for in-memory databases differs from the disk-based systems. In the classical case of the disk-based systems, the indexes are usually necessary and used to minimize the disk space. Indexes actually use the memory. In contrast, the in-memory index structure is within the main-memory. The objective, in this case, is to decrease the time consumption with less memory usage. Instead of storing the values in the indexes, the use of pointers can show more efficiency and flexibility. There are many types of index structures for the main memory such as AVL-trees, T tree, B-tree, etc.

2.6.1 AVL-trees

The AVL-tree uses a binary search for internal memory; it adjusts itself to maintain a (log n) height and limits the branching for the perfect balance. The AVL tree usually uses pointers to point to the nodes and performs fast. It is more suitable for the in-memory structure.

The drawback of this method is that the deletion may be complicated due to the possible rotations.
AVL-trees are used as indices in the AT&T Bell Laboratories Silicon Database Machine (such as in figure 2.8).

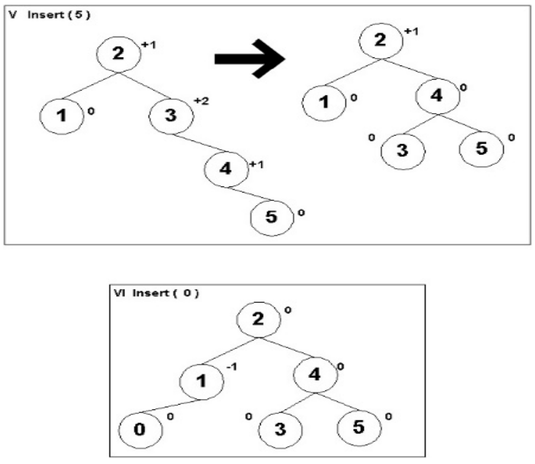


FIGURE 2.8: An example of an AVL tree [8]

2.6.2 T tree

T-tree was first introduced by Tobin J.Lehman and Michael J.Carey in 1986[17].1 is a solution that somehow combine the advantages of the AVL tree and the B-tree. It is has a binary structure and it is designed for the main memory. Similar to the AVL tree in the structure, T-tree has the same balance characteristics2.9. Among many other databases, MySQL Cluster, EXtremeDB and Oracle use the T-tree structure.

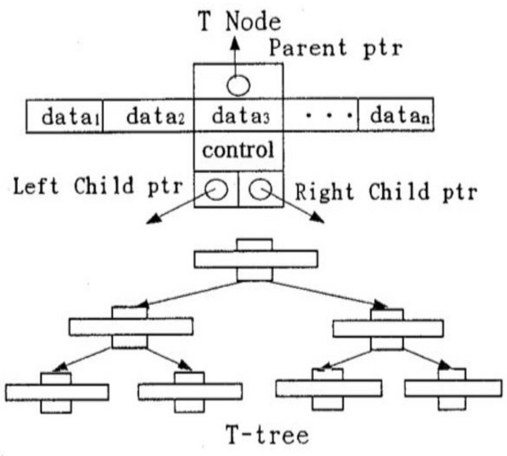


FIGURE 2.9: T tree [9]

2.6.3 B tree

B-trees were first introduced by R. Bayer and E. M. McCreigh in 1972. B-tree is mostly used for storing huge amount of data, and they are common in the external disk storage since it can store a large number of keys and has a deeper tree structure in terms of branching compared to the AVL tree. It is a better solution than the AVL tree when it comes to the hard disk, but not to the main memory. CouchDB databases use the B-tree structure.

2.6.4 T*-tree Index Structure

The t*-tree index structure is a variant of T -tree. It is a binary tree with the same design, except with more pointers for a fast access to data. This type of tree structure solve the issues encountered in the T-tree and improves the queries operations [18]. It also saves memory.

2.7 Partitioning

Partitioning is an essential process for parallelism. The process consists of dividing data into distinct datasets. Each partition has its own properties and can be managed independently. Partitioning improves performance and allows an easy control and maintenance. In fact, partitioning also enables saving time for queries. There are many ways to achieve partitioning. The most used methods for partitioning are the vertical, the horizontal and the hybrid portioning.

2.7.1 Vertical Partitioning

This type of portioning involves dividing the data while keeping the same number of rows for each dataset. The division is by columns. It implies that the splitting of rows causes the loss of the primary key. For this reason, a normalization is necessary. Normalization suggests the linking of the tables with a foreign key. The vertical partitioning is very practical if some columns of the data are used more than others. In addition, it is useful for limiting access to some columns in the data to some users. For example, in a large dataset of hospital patients, the repartition may suggest a division of the data in two datasets, the general and medical information about the patient in one dataset and his personal information in the other one.

2.7.2 Horizontal Partitioning

Horizontal portioning also known as row partitioning. It consists of splitting the data into a number of distinct datasets with respect to the columns. Which means that all the datasets must have the same number of columns. The horizontal portioning is a classical method and a simple one. However, the splitting criteria must be significant to ensure a quick access and maintain a logic of partitioning. For example, if we have weather data by hours for one year, we can divide this data by months, so we have twelve distinct datasets.

2.7.3 Hybrid Partitioning

The hybrid partitioning combines both vertical and horizontal partitioning. This type of partitioning is popular among very large datasets. Usually, the partitioning starts with vertical partitioning [19], followed by the vertical partitioning. Hence combines the advantages of both partitionings.

2.8 Replication

Replication is the main concept in managing datasets: it ensures security, consistency and integrity of data. It is essential as well for backup strategies. Mostly, Replication allows a better management of data. Managing conflicts, availability and recovery are parts of data replication process. It is important to manage a copy of the data for security matters, but managing the synchronization is necessary when the data goes through growth and modification. The updates should be performed in an automatic way to save time and protect the integrity of the data.

The user can choose from different types of replication, to find the optimum solution to increase performance.

2.8.1 Master and slave replication

This type of replication considers managing data from a master server to slave servers. The replication agent is responsible for managing the communication between the two servers using a log. The master maintains the data of all slave sites. If any change is detected from the master side, the replication agent communicates those changes to the slave's side. This method is useful for backups if the master server faces a failure or a loss of the data.

2.8.2 Replication configurations

The replication configuration must be achieved in a way it can improve performance and guarantee data availability. The configuration ca be made using SQL statements from the master to one or many slaves. in this case, it is called unidirectional replication. The other configuration is called bidirectional, it links two or more master servers to a slave. When data stores are configured for bidirectional replication it is called “N-way” or “update anywhere”.

2.8.3 Synchronous replication

Synchronous replication updates the information at the same time it is received in both sides. By that, the data is always on point and updated. This method is expensive and it is rarely used in practice. Even if it guaranties zero loss of data at any time, both datasets are identical. Which means that if the first dataset is under attack, the corruption will be instantly copied to the second one. This type of replication is only practicable if the need of instant synchronization of both datasets is more important than security and cost.

2.8.4 Asynchronous and return service replication

Asynchronous replication is the default method and the most used one. The drawback of the asynchronous master-slave architecture is that when the master server is updating the data, the slave is waiting. However, this method is safer and less costly. The asynchronous replication is done in steps. The changes are copied after the updates in the master side is done. The slave side confirms the reception and application of the changes.

2.9 Durability store

The data resident in-memory can easily be lost if there is a lack of a backup strategy. The data must be stored in a disk or other stable storage. The components of recovery include the procedure used during normal database operation to keep the backup up-to-date and the one used to recover from a failure. The system that uses a log for commit processing generally also perform backups or checkpoints to limit the amount of log data that must be processed to recover from a failure. There are many tools to achieve backups. Durability can be realized with logging and checkpoints.

2.9.1 Logging

Any system can be in a situation of failure, due to the dependence on hardware. The availability and integrity of data must be secured. Many mechanisms have been developed to achieve recovery including logging. The logging can recover the best stable state of the data. The data is written in log files then it restored on Disk. The transaction log backup allows the recovery of data at a specific time before the failure. The logging is used in enterprise applications and can work in both in-memory or for RDBMS.

2.9.2 Checkpoint

The different modifications of the data are performed in the in-memory. The checkpoint allows the system to register this modification in disk periodically. Checkpoints create a snapshot of the database at a certain point in time when the data is in the stable state. The purpose of check pointing is “to speed up the recovery process, since only log entries after the snapshot have to be replayed, while the main store can be loaded from the snapshot directly.

Chapter 3

Predicting Power Consumption Using Machine Learning And An In-memory Database

3.1 Introduction

Large data is generated every day from different sources, especially with the development of social media and the new technologies. Machine learning is the key to analyzing data and extract knowledge for future use. The business world today wants to access the valuable data and process it fast so it can extract the useful information to make decisions in the right time, and stands out in the market. For this reason, in-memory databases became popular amongst companies trying to find the best solution for speed. In this chapter, we use a real data set to predict the power consumption using machine learning in Python and an in-memory database. The choice of the in-memory database is based on a comparison of three well-known open source in-memory databases. The method used for the comparison is Yahoo Cloud Serving Benchmark. At the end, we propose a graphical user interface in Python for machine learning, to select the best algorithms based on a given dataset and also to perform predictive based on the electrical dataset.

3.1.1 Electricity Demand Side Management

Nowadays, utilities have a tendency to invest in Demand Side Management (DSM) for many reasons. In fact, it became important to manage and control the energy usage in order to reduce costs related to the power consumption and the power generation resources. The growth in population and the extensive usage of electronic devices led to an increase in the power's usage and consequently the need for further resources. The utilities today seek a way to achieve their goals with lower costs. The Electricity Demand Side Management (EDSM) offers solutions for a better management of electricity consumption based on the future prediction of the electricity usage. According to a report from Navigant Research, the global spending on DSM will increase from $214.7 million in 2015 to $2.5 billion in 2024 [5]. The implementation of smart meters is a widely used technique to collect electrical data from the devices on a big scale. This data is analyzed to determine the pattern of electrical loads which allows a future prediction of the electricity needs and therefore be prepared for the peak times. DMS proposes different solutions for the electricity management in order to avoid the waste of energy and the need for new power installations. Companies that use electricity on a large scale are the most interested in DSM since it offers solutions enabling one to decrease the costs of electricity usage and reduce the risks related to the critical needs of extra power generators.

3.1.2 Machine Learning

In order to define what machine learning is; it is important to understand to what each word refers. Machine learning is composed of two notions: machine and learning. The machine is the subject of the process, it must learn and develop itself, and that is from where learning comes.

Actually, learning means changing the behavior with experience. In machines, learning involves the amelioration of prediction based on changes in data or the program. Tom Mitchell gave the following definition when asked about machine learning: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E [20].

The machine observes the first data and makes a prediction, but when exposed to new data, it gives a better prediction. That is the process of learning.

Machine learning involves developing programs to learn in order to improve then performance every time exposed to new data. It has a goal of solving a problem based on past knowledge.

Machine learning is a subfield and an essential part of Artificial Intelligence for the simple fact that there could not be any artificial intelligence machine without machine learning.
There are many reasons that made machine learning important today:

* The development of new technologies and the interest of artificial intelligence
* Big data and the large sizes of data are created every day from different sources
* The need for speed in analyses processing and less human involvements.
* The machine can extract and remember information that is hidden or non-logical to human thinking.

Machine learning is based on many disciplines, which are statistics, psychology (copy the human brain), biology (neural network) and computer science [21].

There are some important concepts in machine learning, such as the generalization. It means that the machine learns something for data, and develops a general idea based on the examples. This generalization allows prediction of the new examples. The other important concepts are the training and testing. The algorithms learn first form the training data and develop a model that fits this data. To evaluate the algorithm, we use the test set.

Different algorithms can be used to solve a machine-learning problem. In general, the choice of the algorithms depends on the type of the data and the type of the problem that needs to be treated. There are algorithms for regression, clustering, and classification.

3.2 Objectives

Keeping cost of energy in the range of budget and save the overflow of the production is a big challenge that is becoming an important requirement for electricity companies. We witness the increase of world's population and we use the electrical devices more and more in all aspect of life. Humans became dependent on the electrical goods like mobiles, TVs, electric cars and more. This has pushed us to find all possible solutions to save energy and in the same time provide the power for each single device. To do this, we want a system that can predict the power consumption in the few next hours and tell us if we need to generate more power or not. This system can be done with the houses power consumption data and predict the values that can be generated in the next few hours. However, with all houses in the country or even in the city our dataset will be very large if we want to process all the houses' history dataset. If we want to calculate and evaluate the value of power in the next hours, using the classical databases will be impossible due to speed limitation for storing and accessing data in time. The solution is to store the data in In-Memory database like Redis.

3.3 In-memory databases comparison

3.3.1 Introduction

Databases are stored and processed in different ways, some of them in-memory and others in disk. These systems defer in key store, durability, replication, capacity, and speed, which is main feature. In order to distinguish which system is the best, we should proceed with a comparative approach. NoSQL databases and disk-based databases present enormous advantages and facilities and many papers and articles were published in relation to the subject. However, with the in-memory databases, there are fewer publications done, in addition to some comparisons that were made by in-memory companies, but none from neutral researchers. In this Chapter, we will compare three in-memory databases key-values store systems. We chose databases that were originally built as in-memory databases, and not on disk like Aerospike , Redis and Tarantool.

For the comparison, we used YCSB tool from Yahoo. We also propose a general table for comparison between Redis, Aerospike and Tarantool.

3.3.2 Aerospike

Aerospike is an open source data base key-value store first release was 2012. It is written in C, and works with traditional disks, RAM and flashes. The architecture of Aerospike has three layers. The first one is called the distribution layer or the cluster, it has a replication distribution to set the consistency with a cross data-center synchronization. The second layer is the smart client layer, which is used to control the communications in the server node. The last layer is the data layer, and it is used to store the data in the RAM or the disk.

Aerospike works on Linux ,Windows and Mac. It works as well with the cloud like amazon and Google Compute Engine. Aerospike can support different data structures: integers, string, bytes, double, lists, maps and GeoJSON. It can be used from shell as console mode or by using client like java, pyhon, node.js, go, ruby, C, C sharp, Go, PHP.

3.3.3 Redis

Redis is a database based on the main memory which stores all the data in-memory without the need of a hard disk (except for durability).

Redis is open source and a key-value data store. Its first release was in 2009 by Salvatore Sanfilippo. Redis is written in ANSI, It works on Linux and OS X. it is very easy and light to install and run. It has also a lot of APIs that allow the access between Redis and other applications. Redis can work with a lot of clients like ActionScript, Bash, C, C++, Clojure, Common Lisp, Crystal, D, Dart, Elixir, emacs lisp, Erlang, Fancy, gawk, GNU rolog, Go, Haskell, Haxe, Io, Java, Javascript Julia, Lua, Matlab, mruby, Nim, Node.js, Objective-C, Ocaml, Pascal, Perl, PHP, Pure Data, Python, R, Racket, Rebol, Ruby, Rust, Scala, Scheme, Smalltalk, Swift, Tcl, VB and VCL.

3.3.4 Tarantool

Tarantool is an open source distributed under a BSD license, as an integral part of the Mail.Ru, an internet company in Russia. It is a key value store, and written in c. Its first release was in 2008. It runs on Linux and Mac and it was developed to be in-memory database. It has a fiber model that makes it possible for the applications run simultaneously on a single thread. It is a Lua application server and has its own console and clients. The data types that Tarantool can support are number, String, Boolean, Nill, Lua table or Tuble. Tarantool connectors are PHP, Java, Python, C, Ruby, Node.js, Erlang and Go.

3.3.5 Table of comparison

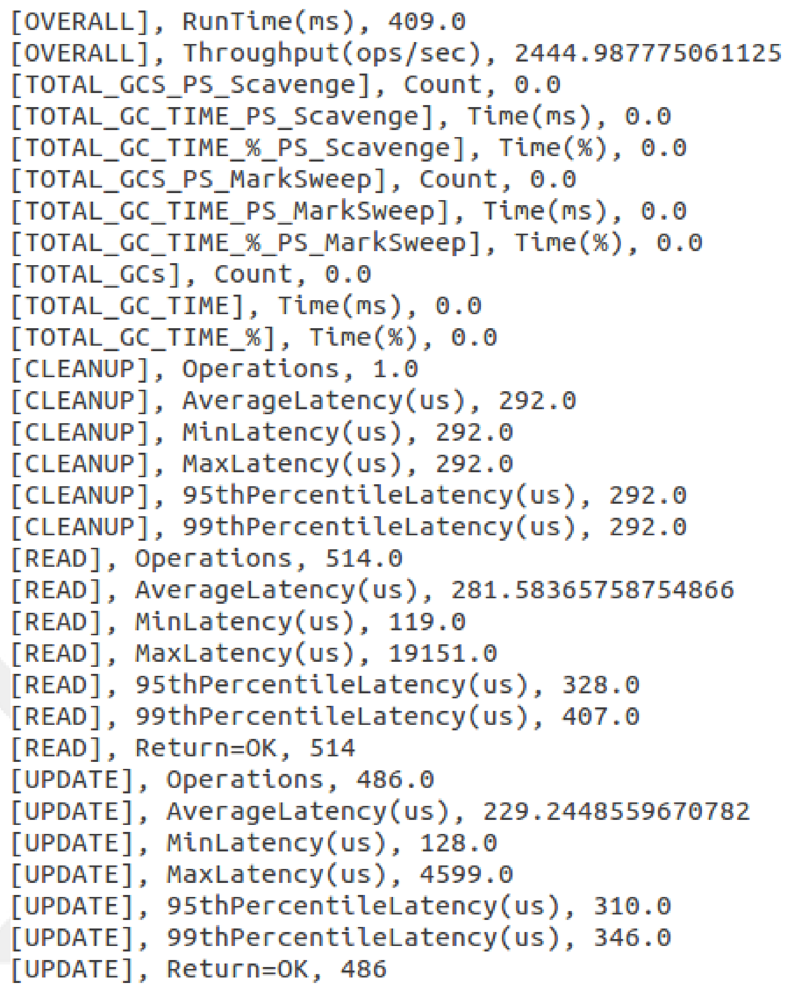
We looked for the features for every database and we go the result as in table 3.1

3.3.6 Yahoo Cloud Serving Benchmark(YCSB)

YCSB is an open source project released in 2010 by Yahoo to compare NoSQL database systems. It uses some tests called workload. The goal of (YCSB) project is to develop a framework and common set of workloads for evaluating the performance of different key-value and cloud serving stores. YCSB has a set of predefined workloads to define a basic benchmark, with the possibility to add new workloads. Each workload represents a different load that the application will put on the database system. YCSB is a commonly used method to benchmark the NoSQL database management systems. For comparison, we use five tests from workload A, B, C, D ,E and F. A Workload is type of test for the database. It may can be reading or writing or mixed with more than one operation. Each workload represents a particular mix of read/write operations. The YCSB shows us the latency of every operation like read and insert in microseconds, in addition, every workload gives us the throughput (the number of operations in one second). The data set used for the comparison is a random data that generates itself automatically with a size of 1 GB.

TABLE 3.1: In-memory Databases General Comparison

	Aerospike	**Redis**	**Tarantool**
Model	Key-value	Key-value	Key-value
Developer	Salvatore Sanfilippo	Aerospike Inc	Tarantool
License	BSD	AGPL	BSD
Implementation Language	C	C	C
First public Release year	2009	2012	2008
Latest stable version	3.2.0	3.7.3	1.6.7
Embedded	No	No	Yes
Cross-platform (Windows, OS X, Linux)	Yes	Yes	No no support for Windows
Supported programming languages	$C, C\#, C++, Clojure, Dart, Erlang, Go, Haskell, Java, JavaScript, Lisp, Lua, Objective-C, Perl, PHP, Python, Ruby, Scala, Smalltalk, Tcl$	$C, C++, Java, C\# Node.js, Go, PHP, HHVM, Python, Ruby, Erlang, libevent2(C), Perl$	$C, php, Perl, Python, Java, Ruby, JavaScript, Go$
Server Side scripts	Lua	Lua	Lua
Locking model	Optimistic locking	Lock Free Model	Lock Free Model
Query language	API calls	UDF and Lua	Lua, HTTP, JSON
Interface	CLI	AMC and CLI	CLI
Compression	Yes	No	No
Minimum memory	3 GB	2 GB	8 M
Replication mode	Master-Slave Replication, Synchronous replication	Master-Master Replication	Asynchronous replication
Indexing	Hash, Bitmap	Secondary Indexes, B tree, Hash	Secondary Indexes Composite keys Geospatial Indexes R-tree, Hash, Bitmap
Cloud platform support	Heroku Cloud Foundry	Google App Engine Amazon	No



```
[OVERALL], RunTime(ms), 409.0
[OVERALL], Throughput(ops/sec), 2444.987775061125
[TOTAL_GCS_PS_Scavenge], Count, 0.0
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 0.0
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.0
[TOTAL_GCS_PS_MarkSweep], Count, 0.0
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 0.0
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 0.0
[TOTAL_GCs], Count, 0.0
[TOTAL_GC_TIME], Time(ms), 0.0
[TOTAL_GC_TIME_%], Time(%), 0.0
[CLEANUP], Operations, 1.0
[CLEANUP], AverageLatency(us), 292.0
[CLEANUP], MinLatency(us), 292.0
[CLEANUP], MaxLatency(us), 292.0
[CLEANUP], 95thPercentileLatency(us), 292.0
[CLEANUP], 99thPercentileLatency(us), 292.0
[READ], Operations, 514.0
[READ], AverageLatency(us), 281.58365758754866
[READ], MinLatency(us), 119.0
[READ], MaxLatency(us), 19151.0
[READ], 95thPercentileLatency(us), 328.0
[READ], 99thPercentileLatency(us), 407.0
[READ], Return=OK, 514
[UPDATE], Operations, 486.0
[UPDATE], AverageLatency(us), 229.2448559670782
[UPDATE], MinLatency(us), 128.0
[UPDATE], MaxLatency(us), 4599.0
[UPDATE], 95thPercentileLatency(us), 310.0
[UPDATE], 99thPercentileLatency(us), 346.0
[UPDATE], Return=OK, 486
```

FIGURE 3.1: The result of Workload A

3.3.6.1 Workload A

This workload has a mix of 50/50 reads and writes. Figure 3.1 shows the result of workload A.

The result of this test for reading shows that in reading, Aerospike was the fastest with 281 us. While for the update, Redis was the best with 222 us, as shown in figure 3.2.

3.3.6.2 Workload B

This workload has a 95/5 reads/write mix. This test shows the partial update and most of operations are read. For reading, Aerospike was the fastest. However, in updating it was advanced by Redis. Tarantool had the slowest times in both operations 3.3.

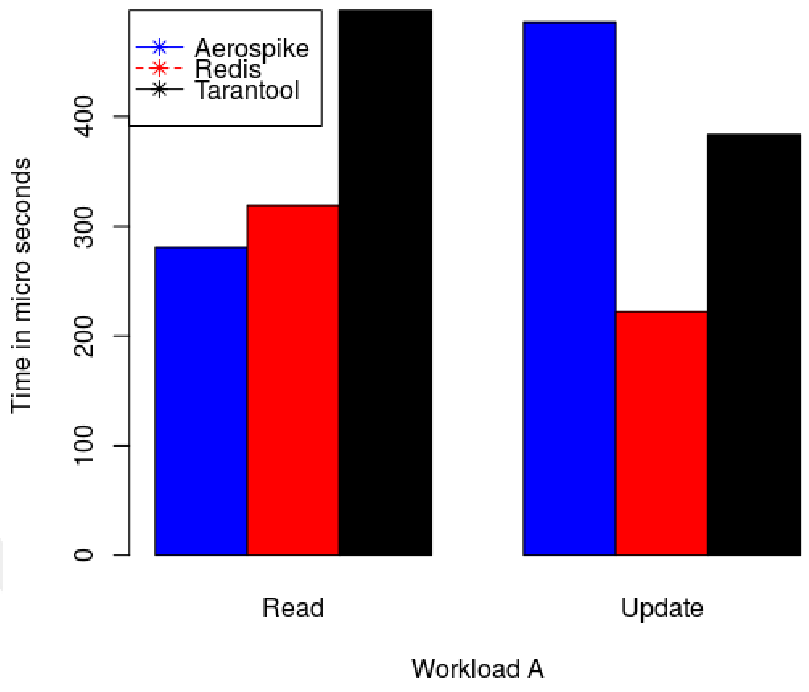


FIGURE 3.2: Workload A

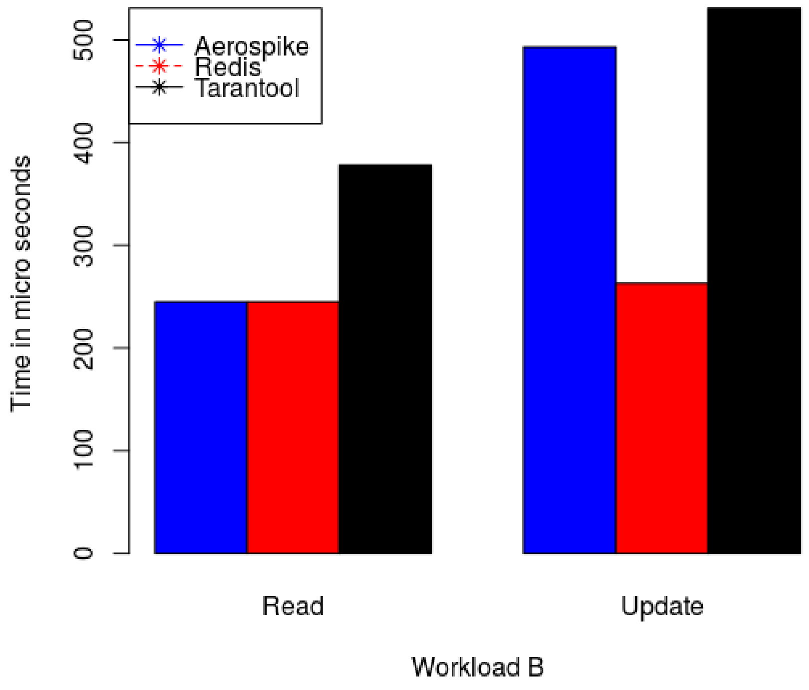


FIGURE 3.3: Workload B

3.3.6.3 Workload C

This workload is 100/100 read. There is no writing. From previous tests, we expect Aerospike to be the fastest in reading, which was true. It is quickly followed by Redis, then Tarantool with 378 us. See figure3.4.

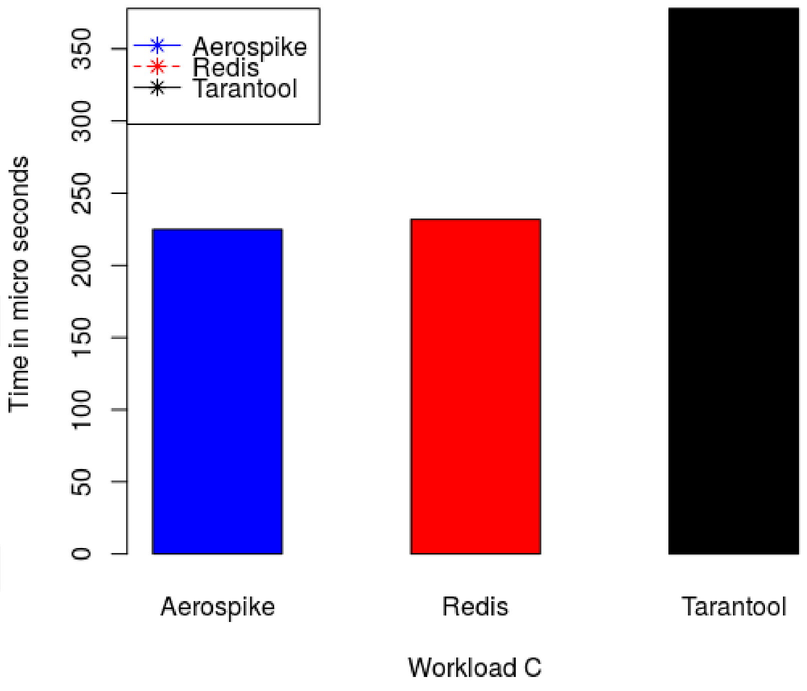


FIGURE 3.4: Workload C

3.3.6.4 Workload D

In this workload, new records are inserted, and the most recently inserted records are the most popular. As we see in 3.5, Aerospike was the best with time of insertion equal to 665 us and 203 us for reading. Redis was second with 1051 us for insert and 231 us for read. Tarantool came in the last place with 1059 us for insert 368 us for read.

3.3.6.5 Workload E

This test uses short ranges of the records. In this test as we see in the figure 3.6, Aerospike has failed in scanning Aerospike scan but was the fastest in insert with 888 us. Redis was the slowest in the scan test with 3674 us. Tarantool'd results were 1151 us for scan and 1070 us for insert.

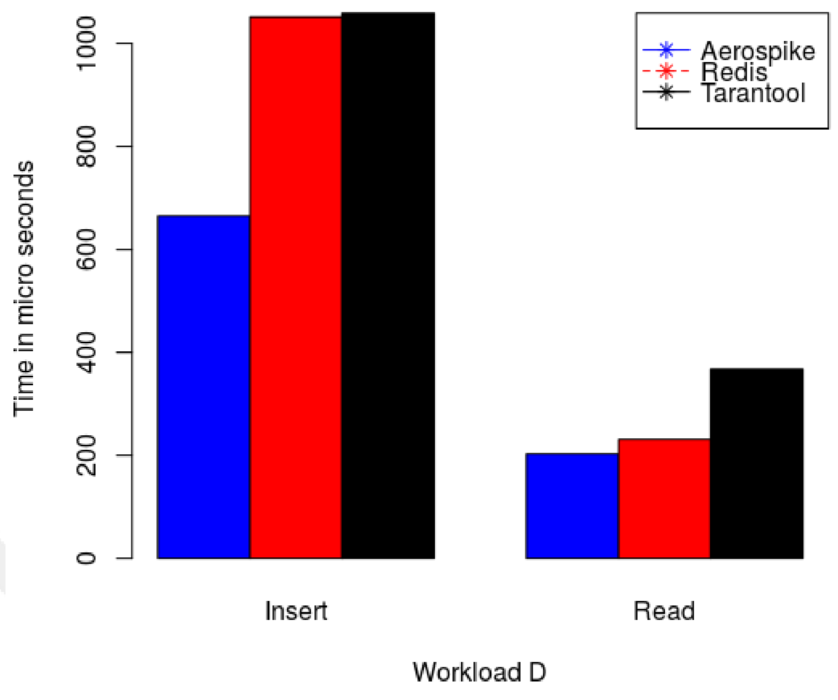


FIGURE 3.5: Workload D

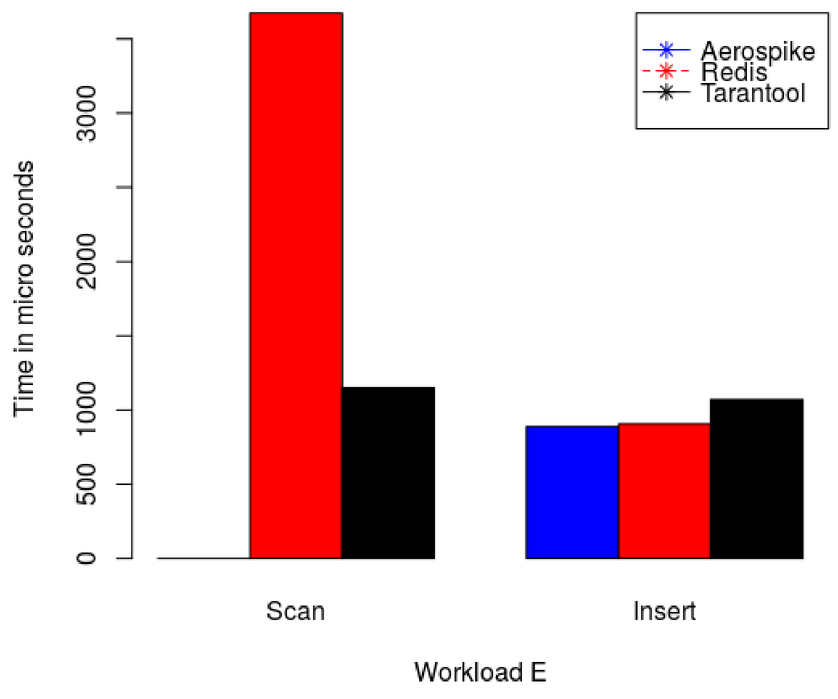


FIGURE 3.6: Workload E

3.3.6.6 Workload F

In last workload, the client will read a record, modify it, and write back the changes. The figure 3.7 shows the following results.

Aerospike: read 197, READ-MODIFY-WRITE 507, update192
Redis: read 245, READ-MODIFY-WRITE 524, update 227
Tarantool: read 375, READ-MODIFY-WRITE 512, update 398

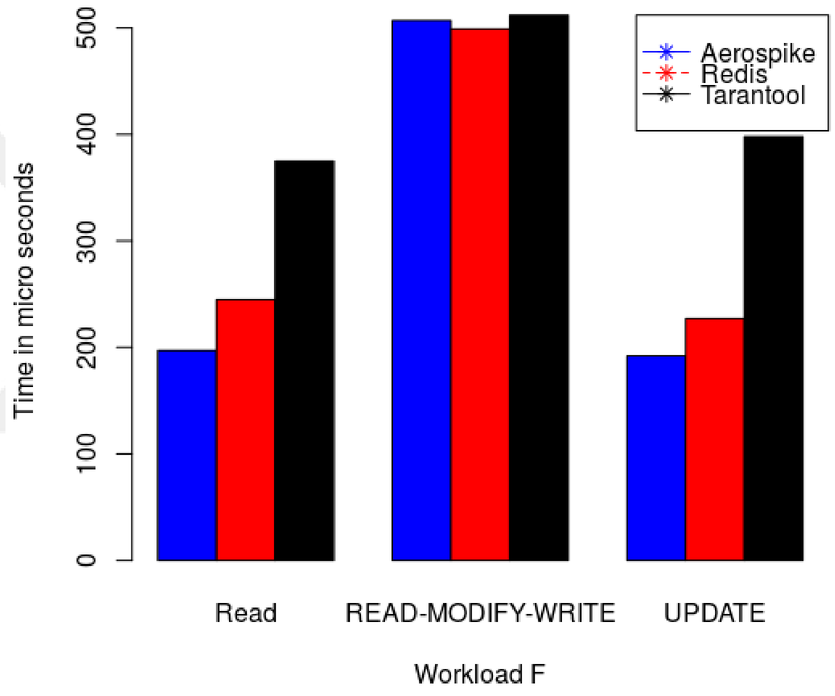


FIGURE 3.7: Workload F

3.3.6.7 Throughput

Throughput refers to how many operations can do the system in one second. In the next figure 3.8, we can see Redis was the best system in the workloads B and C, however, in workloads A, D, E and F Aerospike was the best.

FFrom the different workloads, it seems that Aerospike is the best system in reading while Redis is good in updating. With read-modify-write, both systems had the same

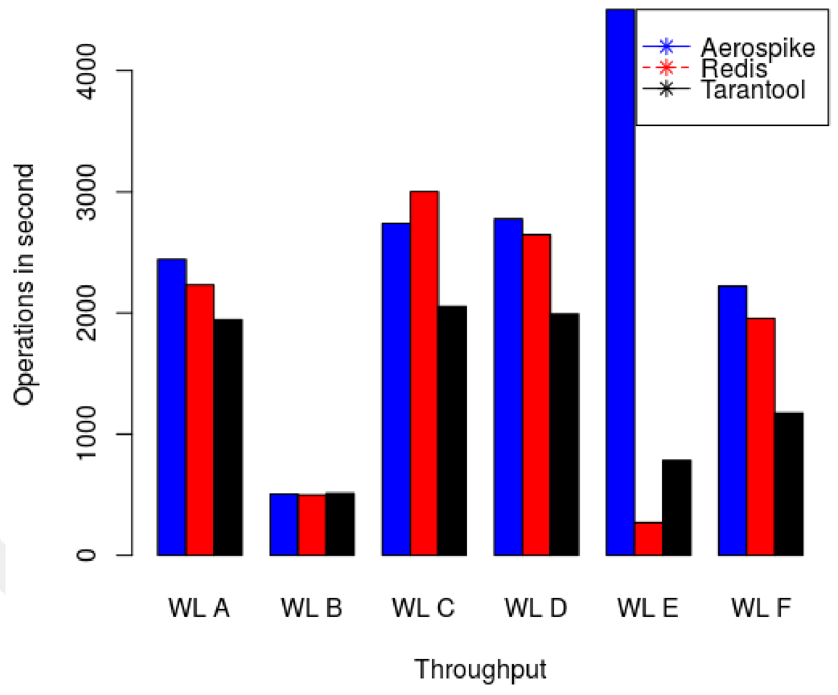


FIGURE 3.8: Throughput

results. With inserting, Aerospike was the best one. Meanwhile, Redis was good in throughput.

These observations lead us to the following conclusion:

Aerospike is good for databases that do not need a lot of modifications or writing. However, Redis can be good for online systems and frequent modifications.

Comparison for memory usage:
We added another comparison based on memory usage. We tried to calculate the space that in-memory database takes in main memory when it uploads one gigabyte of data. As shown in figure 3.9, we can see Tarantool didn't take any more space over real value (Gigabyte). Aerospike took 200 MB over one gigabyte. However, surprisingly Redis took double of the value size of data. It took exactly one more gigabyte.

To sum up this comparison, Redis is easy to use, you can insert and get data with just one command, also you don't need to set the connection from another place. Another

Advantage is the backup. If the machine crashes, you will not lose your data. Aerospike is fast in response and has good supporting. In addition, Aerospike has a continuous development in its capacities. When I did a comparison between Aerospike and Redis in December 2015, Redis was by far better than Aerospike in the majority of the tests. However, today Aerospike is the best. Tarantool has a major advantage because it doesn't take a lot of memory. However, it has problems with the configuration and making the connection. You need to open Tarantook server itself before establishing the connection from python.

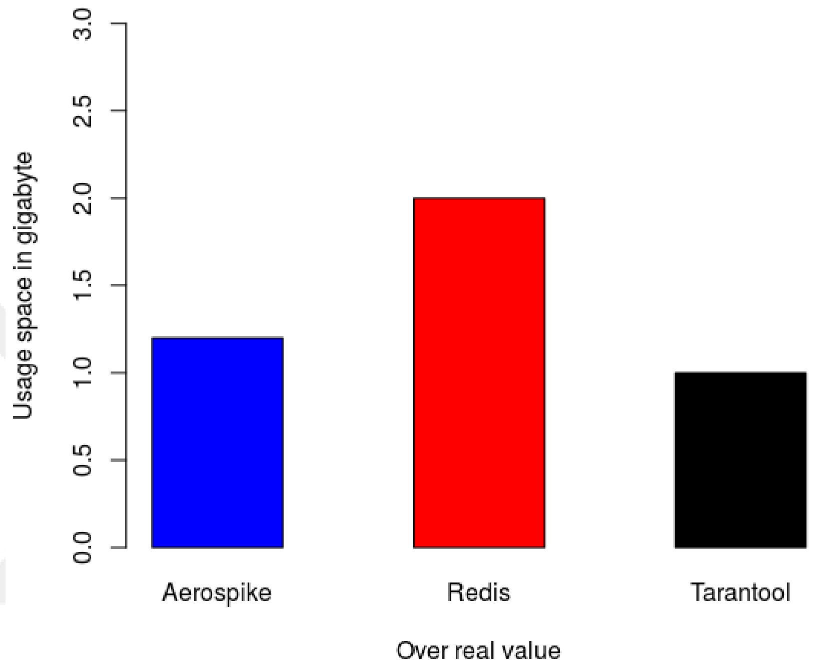
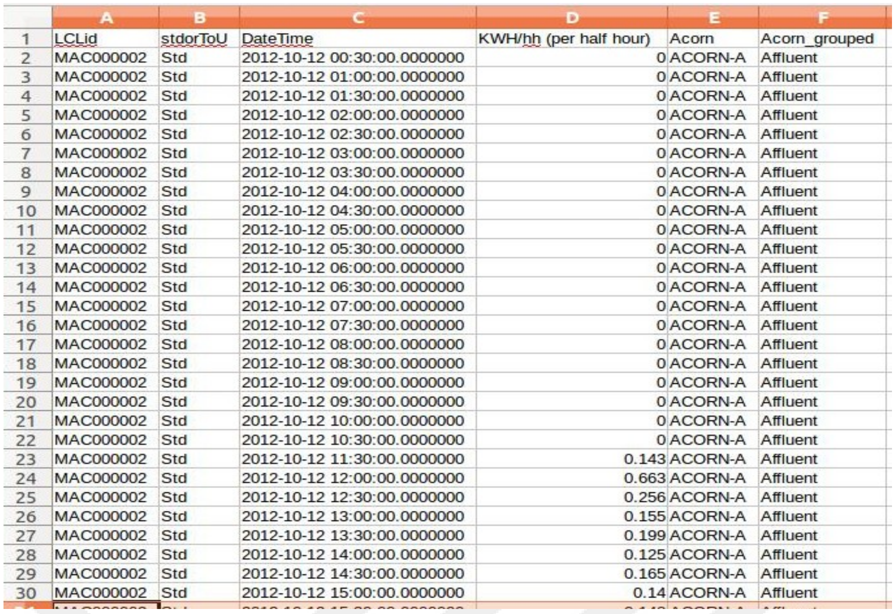


FIGURE 3.9: The data on main memory

3.4 SmartMeter Energy Consumption Data in London Households Dataset

In this study, we use a sample of the energy consumption readings that contain 4,376 London Households in the UK, within the period of November 2011 and February 2014 [22].

The consumption is measured in kWh for every 30 minutes. The features include an ID of the households, the date and time, and CACI Acorn group. The total size of the data set is 11 gigabytes, which is around 167 million rows.



	A	B	C	D	E	F
1	LCLid	stdorToU	DateTime	KWH/hh (per half hour)	Acorn	Acorn_grouped
2	MAC000002	Std	2012-10-12 00:30:00.0000000	0	ACORN-A	Affluent
3	MAC000002	Std	2012-10-12 01:00:00.0000000	0	ACORN-A	Affluent
4	MAC000002	Std	2012-10-12 01:30:00.0000000	0	ACORN-A	Affluent
5	MAC000002	Std	2012-10-12 02:00:00.0000000	0	ACORN-A	Affluent
6	MAC000002	Std	2012-10-12 02:30:00.0000000	0	ACORN-A	Affluent
7	MAC000002	Std	2012-10-12 03:00:00.0000000	0	ACORN-A	Affluent
8	MAC000002	Std	2012-10-12 03:30:00.0000000	0	ACORN-A	Affluent
9	MAC000002	Std	2012-10-12 04:00:00.0000000	0	ACORN-A	Affluent
10	MAC000002	Std	2012-10-12 04:30:00.0000000	0	ACORN-A	Affluent
11	MAC000002	Std	2012-10-12 05:00:00.0000000	0	ACORN-A	Affluent
12	MAC000002	Std	2012-10-12 05:30:00.0000000	0	ACORN-A	Affluent
13	MAC000002	Std	2012-10-12 06:00:00.0000000	0	ACORN-A	Affluent
14	MAC000002	Std	2012-10-12 06:30:00.0000000	0	ACORN-A	Affluent
15	MAC000002	Std	2012-10-12 07:00:00.0000000	0	ACORN-A	Affluent
16	MAC000002	Std	2012-10-12 07:30:00.0000000	0	ACORN-A	Affluent
17	MAC000002	Std	2012-10-12 08:00:00.0000000	0	ACORN-A	Affluent
18	MAC000002	Std	2012-10-12 08:30:00.0000000	0	ACORN-A	Affluent
19	MAC000002	Std	2012-10-12 09:00:00.0000000	0	ACORN-A	Affluent
20	MAC000002	Std	2012-10-12 09:30:00.0000000	0	ACORN-A	Affluent
21	MAC000002	Std	2012-10-12 10:00:00.0000000	0	ACORN-A	Affluent
22	MAC000002	Std	2012-10-12 10:30:00.0000000	0	ACORN-A	Affluent
23	MAC000002	Std	2012-10-12 11:30:00.0000000	0.143	ACORN-A	Affluent
24	MAC000002	Std	2012-10-12 12:00:00.0000000	0.663	ACORN-A	Affluent
25	MAC000002	Std	2012-10-12 12:30:00.0000000	0.256	ACORN-A	Affluent
26	MAC000002	Std	2012-10-12 13:00:00.0000000	0.155	ACORN-A	Affluent
27	MAC000002	Std	2012-10-12 13:30:00.0000000	0.199	ACORN-A	Affluent
28	MAC000002	Std	2012-10-12 14:00:00.0000000	0.125	ACORN-A	Affluent
29	MAC000002	Std	2012-10-12 14:30:00.0000000	0.165	ACORN-A	Affluent
30	MAC000002	Std	2012-10-12 15:00:00.0000000	0.14	ACORN-A	Affluent

FIGURE 3.10: Sample of SmartMeter energy consumption dataset.

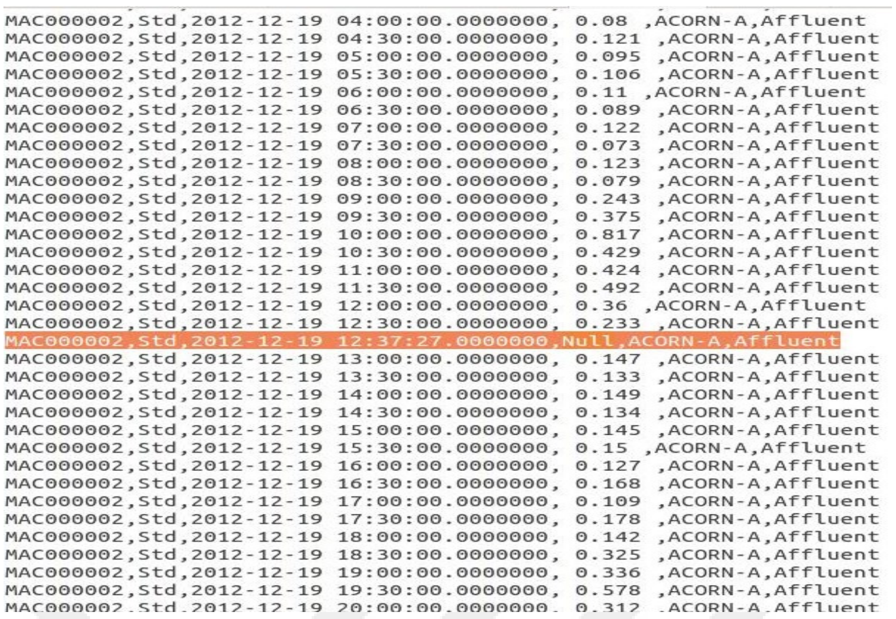
3.4.1 Why London Households Data set?

Our choice is based on the fact that the data set represents the real data of electricity consumed by real people in their houses. Unlike the electricity consumption data of factories or malls - which have approximately the same value every day- this data represents the inconsistency and diversity in the electricity consumption of people depending on their life style and habits, such as the time spent inside the house, late parties, usage of electrical gadgets, etc. In addition, some people use their house only during the summer holidays or weekends.

3.4.2 Preprocessing Data set

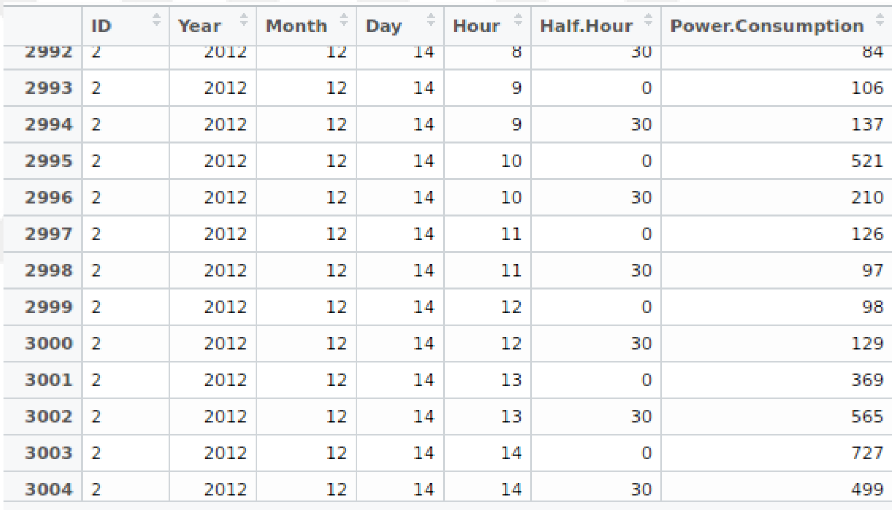
We need to preprocess our data to eliminate the unwanted or redundant data [23], make corrections, deal with missing values, and prepare it for the analysis. To this end, we apply the following preprocessing steps:

- *Feature reduction:* We deleted some columns that have the same values and never change like, Acorn and Acorn grouped.
- *Delete Null Values:* Every house that has one or more null values of power and non-hourly time, is eliminated. as in Fig. 3.11.
- *Separate DateTime column:* We divided the DataTime column to six columns in order to use them in the comparison and make every column as a variable that can affect the result of prediction as seen in Fig. 3.12.



```
MAC000002,Std,2012-12-19 04:00:00.0000000, 0.08 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 04:30:00.0000000, 0.121 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 05:00:00.0000000, 0.095 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 05:30:00.0000000, 0.106 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 06:00:00.0000000, 0.11 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 06:30:00.0000000, 0.089 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 07:00:00.0000000, 0.122 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 07:30:00.0000000, 0.073 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 08:00:00.0000000, 0.123 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 08:30:00.0000000, 0.079 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 09:00:00.0000000, 0.243 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 09:30:00.0000000, 0.375 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 10:00:00.0000000, 0.817 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 10:30:00.0000000, 0.429 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 11:00:00.0000000, 0.424 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 11:30:00.0000000, 0.492 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 12:00:00.0000000, 0.36 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 12:30:00.0000000, 0.233 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 12:37:27.0000000,Null,ACORN-A,Affluent
MAC000002,Std,2012-12-19 13:00:00.0000000, 0.147 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 13:30:00.0000000, 0.133 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 14:00:00.0000000, 0.149 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 14:30:00.0000000, 0.134 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 15:00:00.0000000, 0.145 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 15:30:00.0000000, 0.15 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 16:00:00.0000000, 0.127 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 16:30:00.0000000, 0.168 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 17:00:00.0000000, 0.109 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 17:30:00.0000000, 0.178 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 18:00:00.0000000, 0.142 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 18:30:00.0000000, 0.325 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 19:00:00.0000000, 0.336 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 19:30:00.0000000, 0.578 ,ACORN-A,Affluent
MAC000002,Std,2012-12-19 20:00:00.0000000, 0.312 ,ACORN-A,Affluent
```

FIGURE 3.11: Delete null values



	ID	Year	Month	Day	Hour	Half.Hour	Power.Consumption
2992	2	2012	12	14	8	30	84
2993	2	2012	12	14	9	0	106
2994	2	2012	12	14	9	30	137
2995	2	2012	12	14	10	0	521
2996	2	2012	12	14	10	30	210
2997	2	2012	12	14	11	0	126
2998	2	2012	12	14	11	30	97
2999	2	2012	12	14	12	0	98
3000	2	2012	12	14	12	30	129
3001	2	2012	12	14	13	0	369
3002	2	2012	12	14	13	30	565
3003	2	2012	12	14	14	0	727
3004	2	2012	12	14	14	30	499

FIGURE 3.12: Separate DateTime column

- *Id of House:* We changed the ID of houses from string values to integer values, in order to use the ID in most of the machine learning models. In addition, we changed the power consumption from KW to Watts.

- *Delete incomplete data:* Not all the houses have full region data covering the entire period from November 2011 until February 2014. For example, house number 1957 contains only data from 11-06-2012 till 26-6-2012, which is less than one month. We delete this type of houses. As a result, we have a total of 4,130 houses to work with.

- *Add days:* We add the days' feature to make our prediction more accurate. As we know, by adding weekdays and weekends we can determine the reasons behind the changes in power consumption based on the days. For example, the difference between the consumption in Sundays and Mondays is noticeable. This operation has increased the prediction value by approximately 10% in some houses.

- *Half-hourly to Hourly:* We change the data from half hourly to hourly power consumption as in Fig. 3.15. This would reduce our data to half and improve the prediction. The total size of the data is now 4150 houses with 1.6 Gigabyte.
- *Shape the data:* Figure 3.13 show the consumption for an house just for one day, 28-4-2012. we can determine in which hours the power consumption is higher. As we see, the costumer consumes more power between 11 am and 15 pm, and the highest period is between 18pm and 22 pm in the evening. Furthermore, the period of time when the costumer doesn't consume much power is between 2 and 7 am.

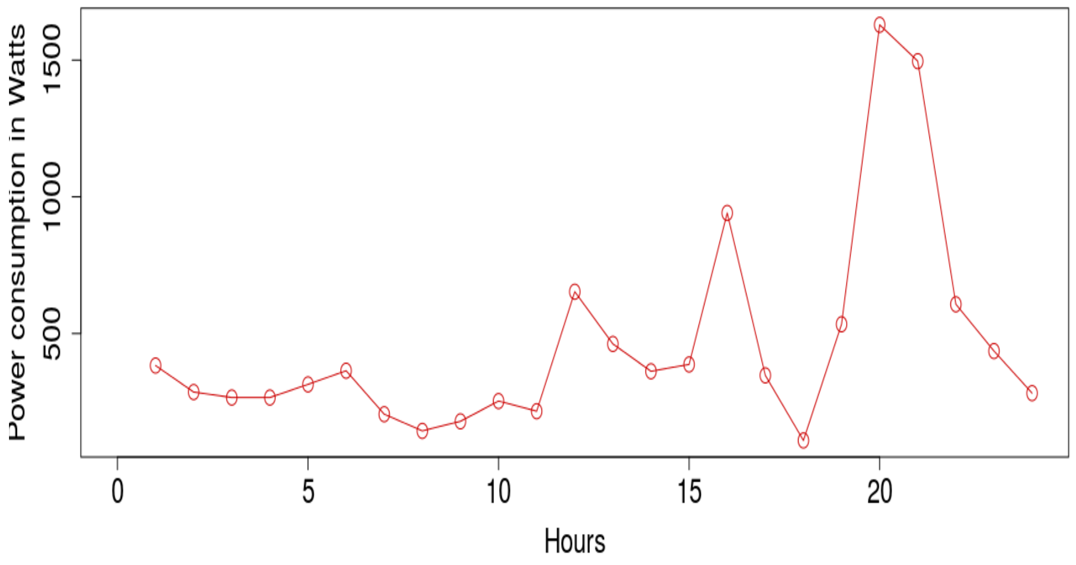


FIGURE 3.13: Electricity consumption for one day for one house.

Figure 3.14 depicts monthly power consumption data of house number 145. We can understand the months where the users consume more power, in this case it is in December, January, and February, which can be explained by the weather condition. Also there is small increase in June, which may be caused by the usage of the air conditioner.

- *Delete non-complete data* Not all the houses have full region data converıng the entıre period from November 2011 until February 2014. For example, house number 1957 has only data from 11-06-2012 till 26-6-2012, which is less than one month.

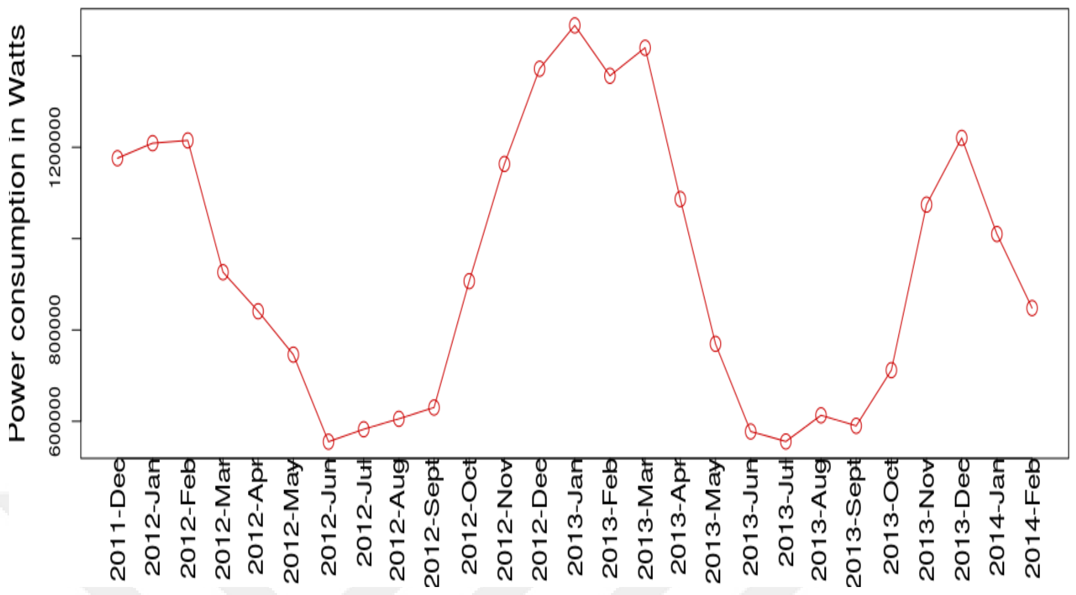
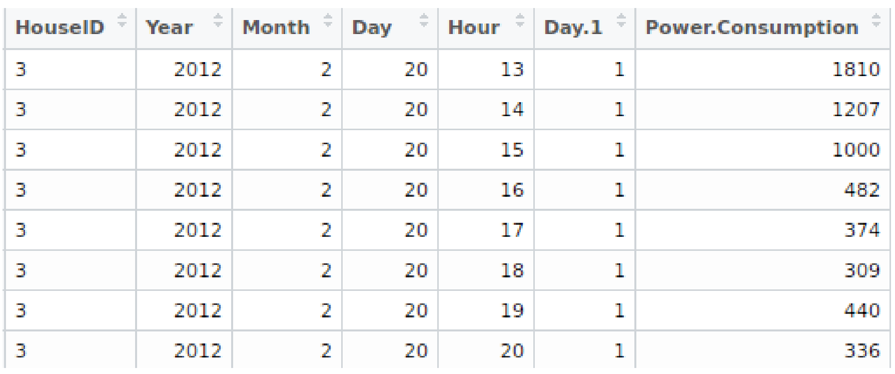


FIGURE 3.14: Total electricity consumption based on months.

We delete this type of houses. As a result, we have a total of 4,130 houses to work with.

- *Add days* We add the days' feature to make our prediction more accurate. As we know, by adding weekdays and weekends we can determine the reasons behind the changes in power consumption based on the days. For example, the difference between the consumption in Sundays and Mondays is noticeable. This operation has increased the prediction value by approximatively 10
- *Half-hourly to Hourly* We change the data from half hourly to hourly power consumption as in figure 3.15. This would reduce our data to half and improve the prediction. The total size of the data is now 4150 houses with 1.6 Gigabyte.



HouseID	Year	Month	Day	Hour	Day.1	Power.Consumption
3	2012	2	20	13	1	1810
3	2012	2	20	14	1	1207
3	2012	2	20	15	1	1000
3	2012	2	20	16	1	482
3	2012	2	20	17	1	374
3	2012	2	20	18	1	309
3	2012	2	20	19	1	440
3	2012	2	20	20	1	336

FIGURE 3.15: Sample of dataset after preprocessing.

3.5 Choice of the best model of machine learning to fit our data

First, we want to choose some models, to apply them and determine which model can fit all or most of our data and give good results. We did a comparison of some models using different machine learning tools: First we split data to training and testing sets.

The split is done using a library called (train test split) to split the data to a training data and a testing data. The value that was given for data training is 80 percent and for testing 20 percent. First, we chose five regression models: Random Forest, Decision Tree regression, Extra Tree regression, Bagging regression and Linear regression.

3.5.1 First Test (R-squared, R^2)

We used R^2 to test the different models and chose the best one.
The R^2, also called the coefficient of determination is a statistical method used for many regression and statistical analysis and is most commonly used to determine the quality of a statistical model's fit [4]. The R^2 error value generally varies from 0 to 1. The model is considered to be a good fit if its R^2 value is close to 1. The formula of the R^2 error is:
$R^2 = 1 - \frac{SS_E}{SS_T}$,
where the SS_E represents the sum of squared errors and the SS_T stands for the total sum of squares[3].

First we applied it to house number 3. We can see the results in Table 3.2. We obtained the best results from the Bagging regression (0.896) and the Random forest (0.895) methods for the house number 3 as shown in Table 3.2. While replicating the same

TABLE 3.2: R^2 Results for various algorithms using House 3 data.

Algorithm	R^2
Bagging regression	0.896080196574
Random regression	0.895494830121
Extra regression	0.87848854631
Decision regression	0.376203098132
Linear regression	0.209074906098
k nearest neighbors	0.691809565284

calculation on the house number 1199 as seen in table 3.3, the R results were not good. For this house, R2 calculation for Bagging was -0.005, R2 for Random Forest was -0.053 but with Decision tree it was 0.10.

TABLE 3.3: R^2 Results for various algorithms using House 1199 data.

Algorithm	**R-squared**
Bagging regression	-0.00505593237186
Random regression	0.0533046674133
Extra regression	-0.0604631321477
Decision regression	0.108030788443
Linear regression	0.0350849871945
k nearest neighbors	0.0425716922033

3.5.2 Second Test (Summation of months)

In this test we calculated the summation of every month independently for testing purpose. Then we divided this sum by the summation of the predicted data.

As presented in the house 3 as in Fig. 3.16. We can see the line fits perfectly with Bagging and Extra regression and lost in some points with Random. But Decision and Knn didn't fit exactly, yes they got the same curve put no on the points.

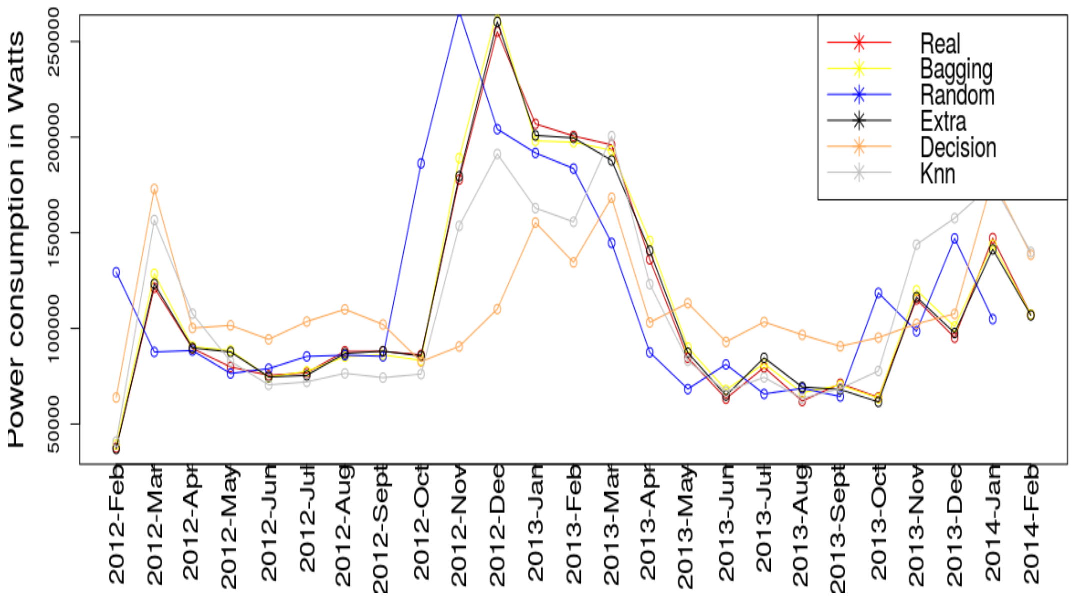


FIGURE 3.16: Comparison of ML algorithms for monthly electicity consumption for house number 3.

In house number 1199 as shown in Fig. 3.17 , the R2 was negative. However, we can see that there is no one fitted. But we can see some points near fitted with Bagging and Random. Like in June and December 2012 and Mars, April and September in 2013. this result led us to elaborate another test.

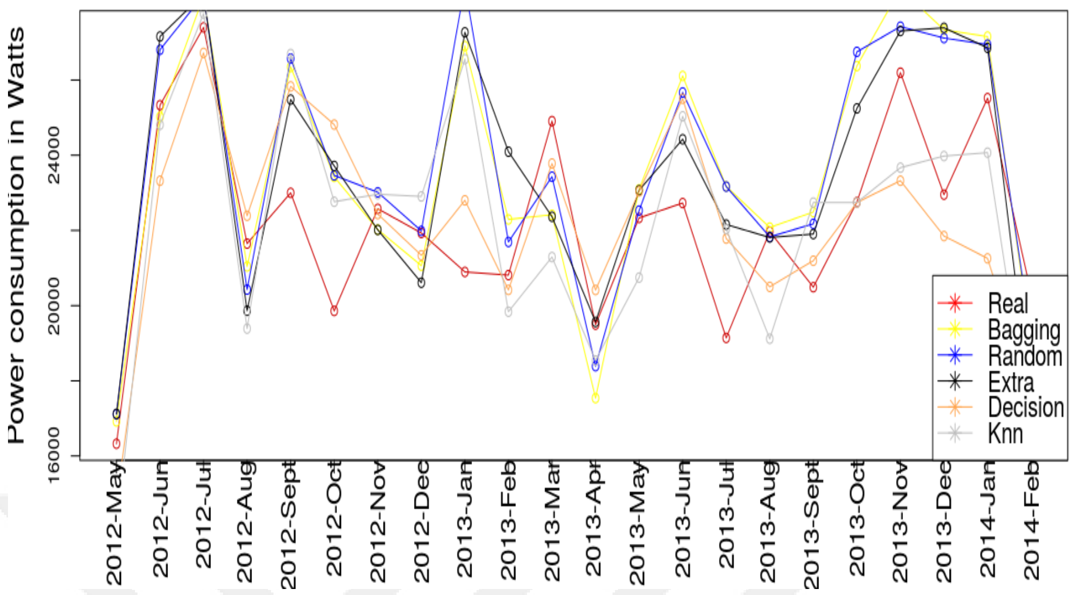


FIGURE 3.17: Comparison of ML algorithms for monthly electicity consumption for house number 1199.

3.5.3 Third Test (By differences between real and predicted individual data)

We conducted another test by calculating the differences between the real data of the test set and the predicted data. This test includes the calculation of the difference between the first value of the test set and the first value of the prediction, then the difference between the second value of the test set and the second value of the prediction, and so on. At the end, we sum up the differences and divide it by the summation of the real values of the test set.

$$Result = \frac{\sum_{n=1}^{m} |\tilde{P} - P|}{\sum_{n=1}^{m} P}$$

where:

$\tilde{P}$ is the predicted data

P is the real data

Python Code:

$$if sum(abs(\tilde{P} - P)) < sum(P):$$

$$result = abs(sum(abs(\tilde{P} - P))/sum(P) - 1)$$

$$else:$$
$$result = abs(sum(abs(\tilde{P} - P))/sum(P))$$

The results of this test were different than the result of the R-squared. For house number 3 as seen in table 3.4, using Bagging, the results of (differences between data test and predicted data) are 0.73 for Bagging and 0.72 for Random Forest. We notice that the results are less than the R-squared calculated values. However, the house 1199 as seen in table 3.5 showed a completely different and interesting results. The Bagging and Random got 0.38 although R2 was equal to -0.005. But for the Decision Tree the result was 0.4.

In fact, house 3 and house 1199 are different. House 3 represents a decent house with a regular and ordinary power consumption. On the other hand, house 1199 is characterized by the irregularity in the power consumption, which may have confused the R^2 calculation to give a negative output.

Since R^2 may in some cases give inconclusive values, we decided to perform more tests.

Table 3.4: Comparison of algorithms for House 3.

Algorithm	**R-squared**	**Differences**
Bagging regression	0.896080196574	0.730565101844
Random regression	0.895494830121	0.721535570183
Extra regression	0.87848854631	0.715903642935
Decision regression	0.376203098132	0.333302025741
Linear regression	0.209074906098	0.139211611706
k nearest neighbors	0.691809565284	0.551830094765

Table 3.5: Comparison of algorithms for House 1199.

Algorithm	**R-squared**	**Differences**
Bagging regression	-0.00505593237186	0.388482137428
Random regression	0.0533046674133	0.381369335206
Extra regression	-0.0604631321477	0.370511945392
Decision regression	0.108030788443	0.401667363226
Linear regression	0.0350849871945	0.353074466826
k nearest neighbors	0.0425716922033	0.383886275354

3.5.4 Fourth Test (Choose 500 houses randomly)

We calculated the average of R^2 and the differences for 500 houses selected randomly as seen in Table 3.6. We can see that all results of R^2 are less than 0.43. But using the differences formula, the Random forest and Bagging result in a higher value of 0.61. Extra regression got 0.60 and K Nearest Neighbors with 0.58. Linear regression value equals 0.44 (although Linear got 0.12 using R-squared).

Table 3.6: R^2 and Difference of 500 Houses randomly

Algorithm	**R-squared**	**Differences**
Bagging regression	0.421451144778	0.614139562817
Random regression	0.421121406344	0.614143242967
Extra regression	0.388489691547	0.605980602049
Decision regression	0.212188577625	0.495803420023
Linear regression	0.125578016697	0.447913538955
k nearest neighbors	0.382293658572	0.580228344575

3.5.5 Test Five (Predict determined hour for All Houses)

In this test, we chose some hours in different days. We chose 8 hours randomly of the the data as in table 3.7 . We have Real data is the result of summation all houses. and the predicted data of every model. By divid the predicted data on Real data we can understand the result clearly like in table 3.8.As we see the there is no difference between Bagging and Random. But the result came perfectly from Extra tree regression.and the worst is Linear Regression. As we see small difference but with thousands of houses can make big effict.

Table 3.7: Comprison Algorithms of 500 Houses Randomly

Time	Real Data	Bagging	Random	Decision	Extra	KNN	Linear
2012-2-27-23-Mon	298415	301952.4	304031.6	380585	294216	293314	381947
2012-7-3-20-Tue	1645631	1658973	1663170	1626277	1649456	1604907	1618319
2012-11-10-12-Sat	2133610	2155970	2146603	1934544	2137980	2141772	1763494
2013-1-25-6-Fri	1689121	1689446	1687991	1533603	1682233	1612100	1559988
2013-4-15-5-Mon	1247781	1260324	1262069	1184304	1250630	1259841	1400693
2013-9-21-1-Sat	919730	926730	926426	1088458	923555	940225	1134373
2013-12-18-14-Wed	1758775	1795363	1797447	1854697	1778146	1808561	1771957
2014-1-19-14-Sun	1921865	2013432	2021547	1758047	1963316	2117988	1759506

TABLE 3.8: Percentage Prediction Comprison Algorithms of 500 Houses Randomly

Time	Bagging %	Random %	Decision %	Extra %	KNN %	Linear %
2012-2-27-23-Mon	98.8	98.1	78.4	98.5	98.2	78.1
2012-7-3-20-Tue	99.1	98.9	98.8	99.7	97.5	98.3
2012-11-10-12-Sat	98.9	99.3	90.6	99.7	99.6	82.6
2013-1-25-6-Fri	99.98	99.9	90.7	99.5	95.4	92.3
2013-4-15-5-Mon	99	98.8	94.9	99.7	99	89
2013-9-21-1-Sat	99.2	99.2	84.4	99.5	97.8	81
2013-12-18-14-Wed	97.9	97.8	94.8	98.9	97.2	99.2
2014-1-19-14-Sun	95.4	95	91.4	97.8	90.7	91.5
Average	98.4	98.4	90.5	99.2	96.9	89

3.6 Create grafical interface software to use on the power dataset

With this GUI, it is possible for a user to use the graphical interface to predict power consumption without being experienced in machine learning or any programming languages. This tool performs automatic prediction using the SmartMeter Energy Consumption Data in London Households.

he first page is **Login page** as in Fig.3.18, which allows the user to enter the User name and Password to onto to the system. This would take us to **Algorithms Page**.

3.6.1 Algorithm page

In this page we have four button boxes and two menus:

3.6.1.1 Add Data Button:

It is used to upload the data to Aerospike (In-memory). By picking up the file from the drag dialog. We can also insert a folder (more than one file).

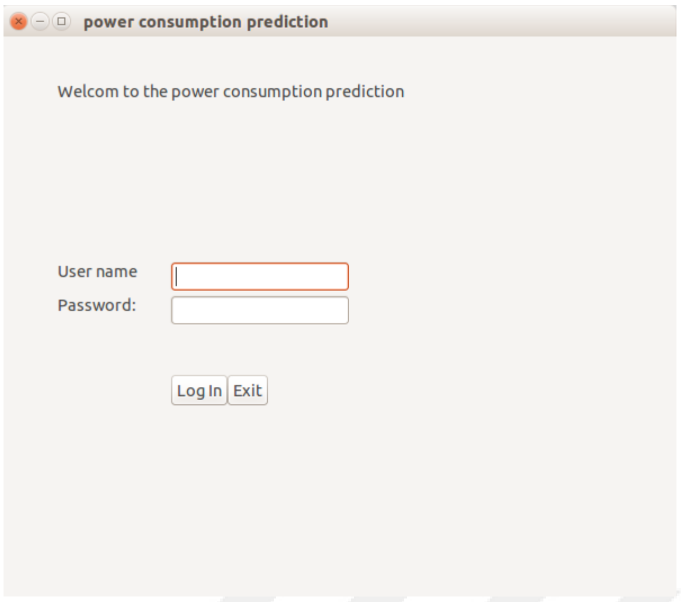

power consumption prediction
Welcom to the power consumption prediction
User name
Password:
Log In
Exit

FIGURE 3.18: Login Page

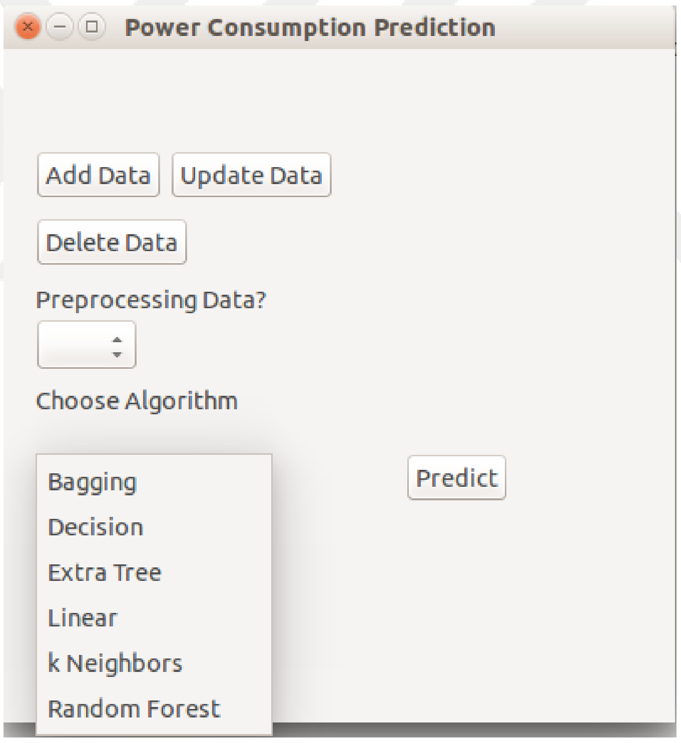


FIGURE 3.19: Main page

3.6.2 Upload the data to Aerospike (In-memory)

we put data on the main memory using Aerospike. It takes the space of 2 GB from RAM after preprocessing the data.. In next two figures, we can see the difference before uploading data in Aerospike as in Fig. 3.20. We have 3.7GB of RAM and 600MB used by the system, after uploading data to Aerospike as in the Fig. 3.21, we can see that the new size of RAM is about 2.9GB. This means that our data size is now 2.4GB, when before it was 6.1GB. This is due to deleting some columns and Null values. We are left with about 800MB for computations and process's work. Of course, we can't put all data in the memory due to the space needed for processing. Finally, we have 3043 houses in our final database.

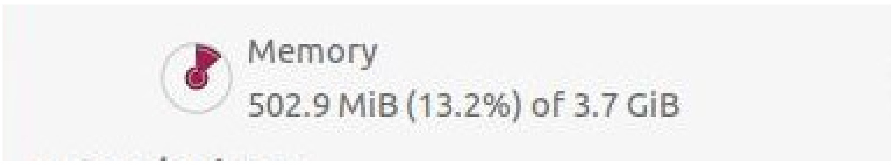

Memory
502.9 MiB (13.2%) of 3.7 GiB

FIGURE 3.20: Before Uplading To Aerospike

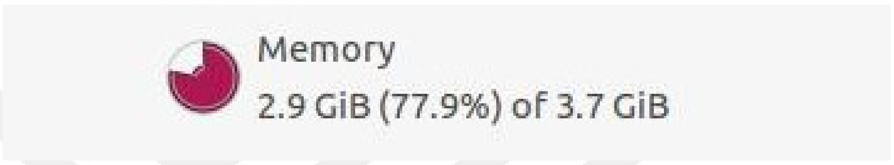

Memory
2.9 GiB (77.9%) of 3.7 GiB

FIGURE 3.21: After Uplading To Aerospike

3.6.2.1 Update Data:

In real life, electrical data of houses has gigabytes and we may want for example to add daily data that comes from meters every few hours. it would be difficult to delete all data from Aerospike and update it then re-upload it to main memory. This would take a lot of time. The solution consists of keeping our data on the RAM and every new data that comes from the houses or the meters will be appended to Aerospike on the Memory to reduce the time.

3.6.2.2 Delete Data:

It is used to delete data from the RAM in order to keep our RAM free

3.6.2.3 Preprocessing Data

In this menu, we have two options YES and NO. It asks if your data needs preprocessing or not. if not, that means that the software will skip the preprocessing part and move to the processing or the prediction phase. If you chose yes, this means that the software will do preprocessing at background.

First, the software deletes Null values, change the ID of every house from string value to integer values, and separates every house alone. It also deletes the columns we don't need for prediction. And finally, separates the time to individual columns and adds weekdays.

3.6.2.4 Choose Algorithm

Here we have a menu that allow us to choose one of six algorithms to apply on the data set: Extra tree regressior,Bagging regressor, random forest regressor, Decision Tree regressor, K nearest neighbor and Linear regressor.

3.6.2.5 Predict:

The predict option redirects us to another page to determine the period we want for prediction.

3.6.3 Period Page

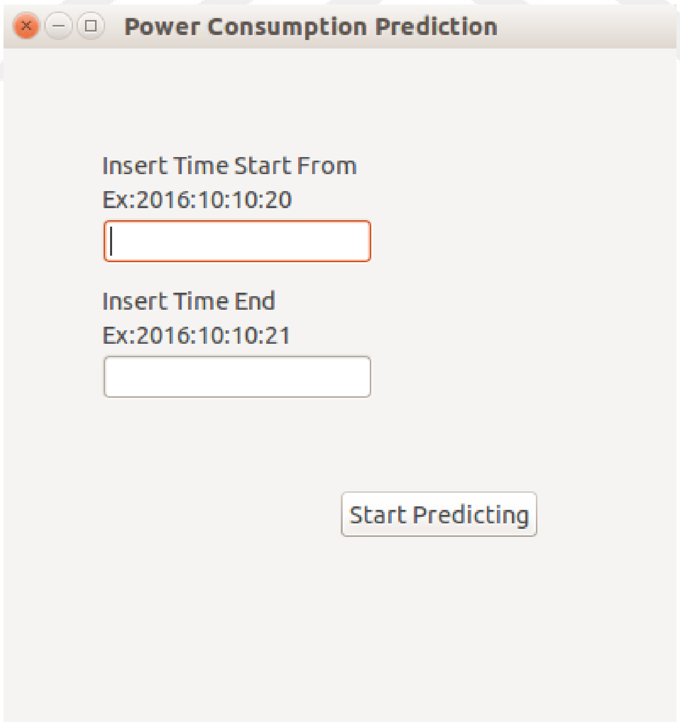


FIGURE 3.22: Prediction Period Page

This page gives us the following commands:
Insert Time from: It is used to insert the beginning of the time period. We added a descriptive note to show that there is only one way to insert the time, and it is by

writing the Year then ":" then the Day. Etc . In this form:
Year:Month:Day:Hour

Insert Time to: the end of the time period must be inserted in the same format. Here, the user must insert the end of the time that he wants to predict.

Start Predicting: it gives the command to start the prediction and redirects the user to another page.

In the background, the data will be divided into a training and a test set. 80 percent of the data will be for training, and the rest for testing.

In normal cases, the code of comparison workss in sequential manner, operation after operation. Our machine has 4 cores and to benefit from those cores we forced our code to work in parallel. We can see in there in Fig. 3.23 that every core has line with special

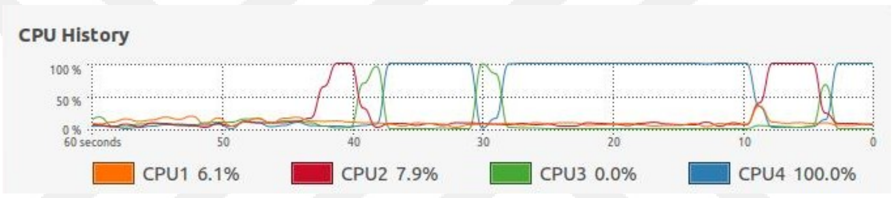


FIGURE 3.23: One core

Color. Traditionally, one core works at the time, and wait for the others. When the first core finishes its work, another one starts. However, with parallel programming we can keep all 4 cores busy working together as in Fig. 3.24, which makes our code 4 times faster than normal.

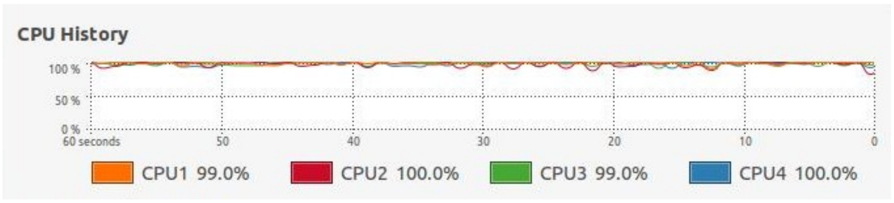


FIGURE 3.24: Work in parallel with four cores

3.6.4 Result Page

In this page as in Fig. 3.25. We have the results of our dataset prediction as following:
Total OF Houses: It shows how many houses we have predicted.
Time In Seconds : It retrieves the total of time in seconds
Results in K/W: It gives the total of power consumption that we predicted within the time interval.

R-squared: Will calculate R-squared for all data

Differences: Will show us the summation of differences between data test and predicted data

Plot: Clicking on this button will generate a plot as a line of the power consumption of real testing data with a line of predicted data.

The final image is saved on the hard disk.

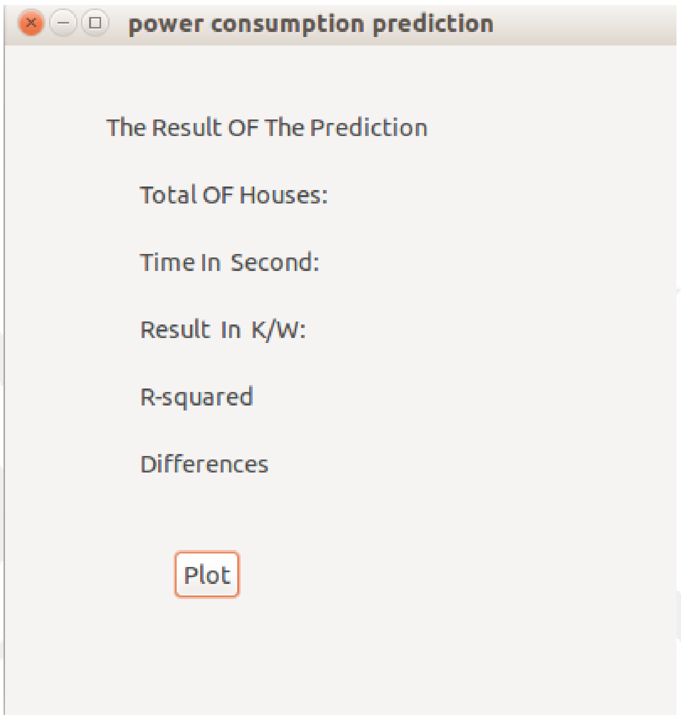


FIGURE 3.25: The results of the predict

Chapter 4

Conclusion

In this thesis, we explored the power of machine learning and in-memory databases. With the development of technologies, the RAM is getting cheaper and offers the advantage of speed over Hard Disk. We proved using machine learning to predict electricity consumption, that the process of reading and writing data is much faster using the main memory.

This thesis is the first application of machine learning with an in-memory database. We chose the Aerospike In-memory database. And using the SmartMeter Energy Consumption Data in London Households data set, we applied many machine learning algorithms for prediction, before selecting the best one: Bagging. During the process, we demonstrated that R squared error is not always the best indicator of the algorithm's success.

In addition, based the SmartMeter Energy Consumption Data in London Households data set, we built a graphical user interface for power prediction that can easily be used by non-experimented users. The usage of main memory is definitely going to be more and more important in the future. The limitation of I/O delays is opening doors for more researches in developing this technology. All applications that need real-time response will start using the main memory instead of HDD.

Bibliography

[1] The memory hierarchy. URL http://blog.csdn.net/cinmyheart/article/details/36687963. Accessed on 2016.

[2] Hasso Plattner. *A Course in In-Memory Data Management.* Springer, German, 2 edition, 2014. ISBN 978-3-642-55269-4.

[3] nosql database comparison. URL http://refreshmymind.com/nosql-database-comparison/. Accessed on 2016.

[4] How do column stores work? URL http://kejser.org/how-do-column-stores-work/. Accessed on 2016.

[5] Comparing document-oriented and relational data. URL http://docs.couchbase.com/developer/dev-guide-3.0/compare-docs-vs-relational.html. Accessed on 2016.

[6] Where is nosql practically used? URL http://scraping.pro/where-nosql-practically-used. Accessed on February 20- 2016.

[7] Hasso Plattner. Sanssoucidb: An in-memory database for processing enterprise workloads. *Datenbanksysteme für Business*, 9, March 2011.

[8] Height balanced binary trees. URL http://www.csee.umbc.edu/courses/undergraduate/341/fall98/frey/ClassNotes/Class18/avl.html.

[9] Tobin J. Lehman and Michael J. Carey. A study of index structures for main memory database management systems. *Proceedings Conference of the Twelfth International on Very Large Data Bases*, August 1986.

[10] Anand Deshmukh and Danny Lohan. Electric load identification using machine learning. 2002. URL https://www.cs.ucsb.edu/research/tech-reports/2002-20.

[11] Atinuke Ademola-Idowu, Pawel Kupsc, and Sonya Mollinger. Home electricity forecasting. December 2014. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1469097.

[12] Eleftherios Soulas Dennis Shasha. Online machine learning algorithms for currency exchange prediction. Technical report, NYU, New York Courant Department, 2013.

[13] Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. Predicting the energy and power consumption of strong and weak scaling hpc applications. *International Journal of Supercomputing Frontiers And Innovations*, September 2014. URL http://superfri.org/superfri/article/view/9/8.

[14] Renu Chaudhary and Gagangeet Singh. A novel technique in nosql data extraction. *INTERNATIONAL JOURNAL of RESEARCH – GRANTHAALAYAH*, 1, August 2014.

[15] Rick Cattell. Scalable sql and nosql data stores. *SIGMOD ACM Special Interest Group on Management of Data*, 39:12–27, December 2010. URL http://dl.acm.org/citation.cfm?id=1978919.

[16] Raghunath Nambiar and Meikel Poess. Performance evaluation and benchmarking: Traditional to big data to internet of things. *7th TPC Technology Conference*, Semtember 2015.

[17] Zhilin Zhu, Jinxue Sui, and Li Yang. A novel main memory database t-tree index optimization algorithm. *International Journal of Advancements in Computing Technology*, 3(5), 2011.

[18] DanielOhene-Kwofie and Gideon Nimako. A fast memory resident index for in-memory databases. *International Conference on Information and Knowledge Management*, 45, 2012.

[19] Poznań. Business information systems. *16th International Conference*, 2013.

[20] Harsh Bhasin and Surbhi Bhatia. Application of genetic algorithms in machine learning. *(IJCSIT) International Journal of Computer Science and Information Technologies*, 2:2412–2415, 2011.

[21] Pat Langley. Editorial: Human and machine learning. *Machine Learning*, 1: 243–248, Sep 1986.

[22] Smartmeter energy consumption data in london households. URL http://data.london.gov.uk/dataset/smartmeter-energy-use-data-in-london-households.

[23] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas. Data preprocessing for supervised leaning. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 12, 2007.