International Journal of Data Science and Analysis
2024, Vol. 10, No. 5, pp. 86-99
https://doi.org/10.11648/j.ijdsa.20241005.11

Review Article

A Comparative Study of Parallel Processing, Distributed Storage Techniques, and Technologies: A Survey on Big Data Analytics

Saliha Mezzoudj[1, *], Meriem Khelifa[2], Yasmina Saadna[3]

[1]Department of Computer Science, University of Algiers, Algiers, Algeria
[2]Department of Computer Science and Information Technologies, University of Kasdi Merbah Ouargla, Ouargla, Algeria
[3]Department of Computer Science, University of Batna 2, Batna, Algeria

Abstract

The significance of developing Big Data applications has increased in recent years, with numerous organizations across various industries relying more on insights derived from vast amounts of data. However, conventional data techniques and platforms struggle to cope the Big Data, exhibiting sluggish responsiveness and deficiencies in scalability, performance, and accuracy. In response to the intricate challenges posed by Big Data, considerable efforts have been made, leading to the creation of a range of distributions and technologies. This article addresses the critical need for efficient processing and storage solutions in the context of the ever-growing field of big data. It offers a comparative analysis of various parallel processing techniques and distributed storage frameworks, emphasizing their importance in big data analytics. Our study begins with definitions of key concepts, clarifying the roles and interconnections of parallel processing and distributed storage. It further evaluates a range of architectures and technologies, such as MapReduce, CUDA, Storm, Flink, MooseFS, and BeeGFS and others technologies, discussing their advantages and limitations in managing large-scale datasets. Key performance metrics are also examined, providing a comprehensive understanding of their effectiveness in big data scenarios.

Keywords

Parallel Processing Frameworks, Distributed Storage Frameworks, MapReduce, CUDA, Storm, Flink, MooseFS, BeeGFS

1. Introduction

Big data is a term that refers to the astronomical amount of data that is produced and stored daily. This data comes from different sources, such as networks social media, e-commerce sites, mobile apps, etc. Businesses and organizations increasingly need this data to develop new products and services, improve internal processes and better understand the behaviors of their client. Big data is characterized by its volume, velocity, variety, veracity, and value, often referred to as the "5Vs." Volume refers to the vast amounts of data generated every second, while velocity indicates the speed at which this data is processed and analyzed. Variety highlights the diverse formats and sources of data, including structured, semi-structured, and unstructured forms. Veracity addresses the accuracy and reliability of the data, which is crucial for

*Corresponding author: s.mezzoudj@univ-alger.dz (Saliha Mezzoudj)

Received: 29 September 2024; **Accepted:** 14 October 2024; **Published:** 12 November 2024

making informed decisions. Finally, value represents the insights and benefits that can be extracted from analyzing big data, enabling organizations to drive innovation, improve operations, and enhance customer experiences.

Big data has also become a powerful tool for researchers and analysts. They cause the data to discover new trends and analyze consumer behaviors. Thanks to big data, it is also possible to improve predictions and economic models.

Although big data is a powerful force, it is important to note that it also involves of the risks. For example, the data can be used for malicious purposes, and companies and organizations are finding it increasingly difficult to manage and analyze this amount of data.

Big data is a large amount of data that exceeds processing capabilities of a conventional computer. This data is usually stored and analyzed in the goal of discovering trends, patterns and insights. The benefits of big data are many. Among the main ones are: A better understanding of customers and their needs, a better ability to anticipate changes, process and analyze ever-increasing volumes of data, making it easier to decision-making and process optimization [28].

Indeed, to manage complex situations in big data generally use:

1) Parallel processing: that is a computing technique that consists of executing several tasks simultaneously on computers connected together. This technique is particularly used in big data to process mass data. With parallel processing, computers can process data faster and reduce latency.
2) Distributed data: Distributed data in big data is data that is distributed over several computers connected together. These data are processed by specific algorithms. Which reduce latency. These algorithms harness the power of calculation of computers connected together to process data faster.
3) Flexibility: It is the ability to adapt processes and tools to changes. Rapid and. constant movements. Companies that succeed in big data are flexible and able to adapt quickly to changes. They also have the ability to process and analyze a large amount of data in real time.
4) Scalability: Big data scalability or scaling up is the ability to process a larger amount of data and to meet an ever-increasing demand in terms of capacity processing and analysis. Companies that succeed in big data have an architecture and scalable processes, allowing them to process a larger amount of data without compromising perform.

In this paper, we provide an overview of the latest technologies created for Big Data. We classify and thoroughly compare these technologies based not only on their applications, advantages, limitations, and characteristics, but also according to different layers, including Data Storage Layer, Data Processing Layer, Data Querying Layer, Data Access Layer, and Management Layer. This approach enhances the understanding of the relationships between different Big Data technologies and how they operate.

This paper is organized as follows. Section 2 defines Big Data preprocessing and presents some of its applications. Section 3 identifies parallel computing and its applications API. Section 4 presents big data processing frameworks and compares some main frameworks developed on top of it. Section 5 presents comparison between Big data parallel and distributed storage systems. Section 6 presents future in Big data analytics.

2. Big Data Pre-processing

Big Data pre-processing is an important step in the value creation process in big data. It consists of transforming raw data into structured and cleaned data to facilitate analysis and management. This step is essential because it allows organizations to discover and act on insights from their thee pre-processing steps for big data are as follows:

2.1. Big Data Cleaning

Refers to identifying and cleaning up inconsistencies. Inaccuracies and inaccuracies in data. This process may involve identifying and correction of errors, filling of missing values and standardization of formats of data.

2.2. Big Data Transformation

Big data transformation is a data processing technique that consists of modifying data from one format to another, or to transform it into a form that is easier to use or analyze. This may include converting raw data into a form structured, merging data from different sources, or creating tables and graphs from raw data.

2.3. Big Data Reduction

It is a data processing technique that consists of reducing the volume of data to be processed by deleting unnecessary data or reducing the size of data. This can reduce the time required for data processing, reduce storage and transmission costs, or facilitate analysis and interpretation Data.

2.4. Big Data Integration

It is a technique of integrating and merging data from different sources to create a single database. This integration makes it possible to have a global and coherent view of the data, which facilitates their use. For this step, the ETL tool is generally used.

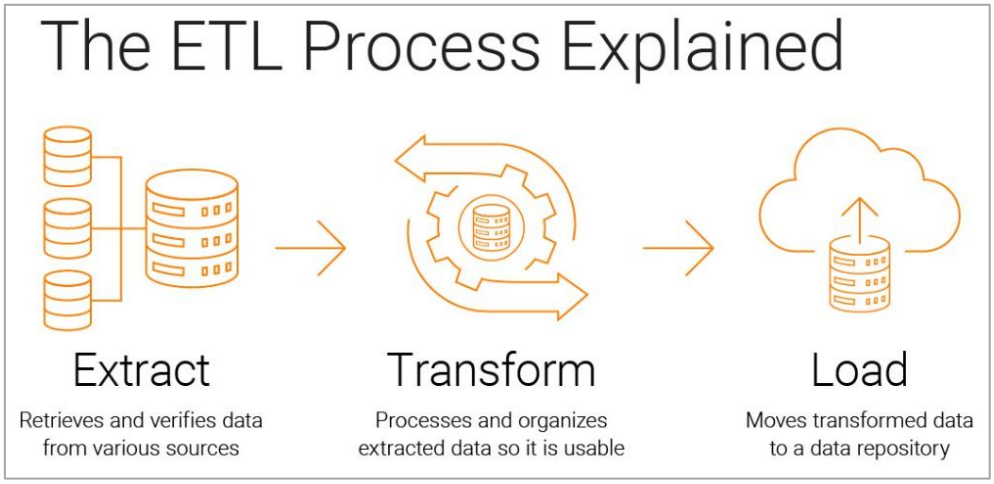

Figure 1. *ETL process.*

2.4.1. ETL Approach [1]

In English ETL is (extract, transform, load), which is an automated process consisting of three main tasks: extraction, transformation, loading. This process extracts information from raw data, then transforms this information in order to be able to upload it and integrate it into a data warehouse.

Among the benefits of ETL we can mention:

1) Quickly integrate a large amount of data.
2) No additional system installed, so the ETL is non-informative.
3) The load can be smoothed over the entire system.
4) Simplify and streamline the architecture.
5) Allows to have a coherent and homogeneous database.
6) Optimize performance, by performing data cleansing.

2.4.2. The Phases of the ETL Process

A. Extracting data

This task makes it possible to extract the most relevant data from different data sources such as spreadsheets, applications or database systems data. We have two types of data extraction [1]. The following table will allow us to explain these two types, thus to establish a comparison between the two:

Table 1. *The difference between the two types of extraction in an ETL process [2].*

Complete extraction	Incremental extraction
Captures the dataset.	Captures only data that has changed since the last extraction.
Typically used in initial data loading or the refresh case full of data.	There are two use cases possible: 1-Real-time extraction: the extraction is do neat the time of transaction in the of transaction in the source system. 2-Deferred extraction: the extraction of all loadings in a given period.

B. Data transformation

This task aims to clean the data to improve their quality and establish the consistency. The different data processing options possible in this phase [1]:

1) Filtering, cleaning, deduplication, deletion, decoding, conversion.
2) Perform calculations, translations or summaries based on the raw data.
3) Data validation and authentication.
4) Formatting data in tables or joined tables.

C. Loading data

The last phase consists of sending the transformed data from the transformation zone to the target data warehouse. There are three types of loading:

1) Initial loading: the out of the data warehouse activation.
2) Incremental loading: is done once the initial loading is complete, to overwrite and delete data in the data warehouse.
3) Full load: this type of loading requires complete unloading data loaded off initial load [2].

2.4.3. Other Approaches

There are other approaches that are also used to facilitate the integration of data. We will present some of these approaches: [2]

(i). CDC (Change Data Capture)

Identifies and captures only source data that has changed and moves it to the target system. CDC can be used independently for data movement transformed into a data lake or other repository in real time, as it can also be used during the extraction step of the ETL for the reduction of the necessary resources.

(ii). SDI (Stream Data Integration)

SDI continuously integrates data as it becomes available. sources instead of integrating the extracted data instantly at a given time. It activates a data store to power analytics, machine learning, and applications in real time to improve the customer experience.

(iii). Data virtualization

This approach consists of using a software abstraction layer to create a view: unified, integrated and fully usable data, without copying, transforming or loading physically the source data on a target system.

Data virtualization enables the creation of virtual data warehouses, data lakes and data stores from the same source data for thedata storage without the expense and complexity of creating and managing separate platforms for each.

3. Parallel Computing

Parallel computing is the process of implementing electronic architectures. digital electronics allowing information to be processed simultaneously. Perform these parallel tasks is possible thanks to the use of several processors or cores si- simultaneously. Unlike the time-sharing technique used by a system operating system (OS) to run multiple programs on a single processor, or when a operating system chooses to run multiple programs on a single processor without interaction between them. Parallelism relies on specialized algorithms and sometimes specific language sand platforms dedicated to parallel computing. These techniques have aim of carrying out the greatest number of operations in the shortest possible time.

3.1. The goal of Performing Calculations in Parallel

Parallel architectures have become the dominant paradigm for all computers since the end of the 1950s, The main reasons for parallel computing:

1) Saves execution time and consequently decreases the loss of money linked to the expenses (electricity for the operation and cooling of the machines).
2) Allows you to deal with larger and more complex problems using the high performance computing systems (intensive computing) for industrial applications trills or simulation of scientific phenomena (eliminates the limits and reduces the cost physical experience).

3.2. Automatic Parallelization

Represents the compilation procedure of a program that transforms the source code written for sequential machines to parallelized executables for multi-computers. The purpose of the latter is to simplify and reduce development time parallel programs because they are much more complicated to write than programs sequential. For this, from different platforms API (Application Programming Interface for parallel computing have been implemented, in particular:

3.2.1. POSIX Threads (Pthreads)

POSIX threads, commonly known as pthreads, are a threading API based on standards for C/C++ [3], it allows one program to control multiple workflows different overlapping in time, each workflow is called a thread hence the pthreads name was inspired. This API defines a standard interface that libraries can are going to use to create and manage threads, while allowing the operating system manage the low-level details needed to create and maintain threads [4].

POSIX threads are primarily useful for multi-processor or multi-processor systems, where threads can be scheduled to run on separate processors and run in parallel with other threads. They can also be used to create fine parallelism within a single processor [5]. Pthreads defines a set of types, functions and constants of the programming language. C grammar provided by the “pthread.h” library. Threads have running status independent and an independent address space; they are not required to share the same process or memory space than their creator. There are about 100 thread procedures, all prefixed with "pthread" and they can be classified into four groups:

1) Thread management to create, join threads, etc.
2) Mutex that represent a lock (mechanism that imposes access limits to A resource)
3) Conditional variables or monitor, a synchronization construct that per- sets threads to have both mutual exclusion and the ability to wait (bloquer) that a certain condition becomes false.
4) Synchronization between threads using locks and read-barriers writing.
5) Twist locks which make a thread simply wait in a loop ("spin") while repeatedly checking if the lock is available [6].

3.2.2. Open Multi-Processing (OpenMP)

Represents a programming interface for parallel computing on an architecture with shared memory [7]. This API supports cross-platform parallel programming with shared memory in C/C++ and Fortran. Supported by many platforms including GNU/Linux, OS X and Windows, it defines a portable and scalable model with a simple and flexible interface to develop

parallel applications on platforms ranging from desktop computers to supercomputers [7].

OpenMP is an implementation of multithreading, a parallelization method in which a main thread (a series of sequential instructions) is divided into sub-threads, and the system distributes the tasks between them to then be executed simultaneously and the environment runtime assigns them to different processors. In other words, just add “#pragma omp” directives in the code to indicate the parts to be parallelized (and com- how to do it), then the compiler and the operating system take care of the rest [7].

Sections of code that run in parallel are marked accordingly with compiler directives that form threads before section execution. Each thread is assigned an ID. After executing the parallelized code, the thread reconnects to the main thread and continues until the end of the program. OpenMP functions arecontained in a C/C++ header file called "omp.h" [7], Basic elements can be classified into six groups:

1) Thread creation: The omp parallel pragma is used to spawn the sub threads necessary to perform the work included in the parallel construction. the thread origin will be designated as the master thread with thread ID 0.
2) Work-sharing constructs: Is used to specify how to assign an independent on one or all threads (omp for or omp do to divide the loop iterations, sections for assignment of consecutive but independent code blocks ants to sub-threads, single and master specifying a block of code that is executed by a single thread and a block of code will be executed by the master thread only respectively) [8].
3) Variant directives: are one of the main features introduced in the. OpenMP 5.0 specification to help programmers improve the portability of forms. They allow the adaptation of OpenMP pragmas and user code to compile time [7].
4) Clauses: to keep the integrity of the parallelism of the code, most of the variables of the code OpenMP are visible by default to all threads. However, private variables are sometimes necessary to avoid race conditions and values must be transmitted between the sequential part and the parallel region (Attribute clauses of data sharing, synchronization clauses, scheduling clauses, IF control, Initialization, Data Copy...)
5) User-level: runtime which represents routines used to modify/check the number of threads, detect if the execution context is in a parallel region, how many number of processors in the current system, set/disable locks, security functions synchronization, etc.
6) Environment variables: which represents method to modify features runtime of OpenMP applications and used to control the scheduling of loop iterations, default thread count, etc. [8].

3.2.3. Message Passing Interface (MPI)

Represents an API born from a standardization effort, defining the standardized and portable messaging mission designed to run on architectures parallel computing. The MPI standard is not a library, but specifies what such a library should provide as functionality such as syntax and semantics of routines, this makes it very useful to a wide range of users writing portable messaging programs in C, C++ and Fortran, and any language able to interface with such libraries including C#, Java or Python [9]. There are several open-source MPI implementations that have helped make it a standard for parallel programming by message passing [10]. MPI's objectives are high performance, scalability a normed in parallel on several computers. MapReduce is often used for bulk data processing applications, such as the processing of logs, click data analysis and pattern recognition.

MapReduce was popularized by Google and is now implemented in several frameworks open source, such as Apache Hadoop and Apache Spark. Additionally, libraries MapReduce are available for different programming portability.

The MPI is process-oriented, the problem to be addressed is broken down into several processes, each of these processes is generally associated with a calculation node. An MPI process can contain a single thread (common case) or several threads. MPI processes are identified by their ranks, if n-number of processes are running, the rank of the processes goes from 0 to n-1. MPI provides several features. The following concepts provide a context for all of these capabilities and help the programmer decide which feature use in its application programs. The four basic elements can be classified as following:

Introduced with the first MPI-1 release:

1) Communicator which is responsible for creating objects connect groups of processes in the MPI session. Each communicator gives each contained process a independent identifier and organizes its contained processes into an ordered topology.
2) Point-to-point basics represented by a number of important MPI functions aunts involve communication between two specific processes (MPI_Send and MPI_Recv...) Point-to-point operations, as they are called, are particular- useful in structured or irregular communications.
3) collective basics which represents collective functions involve communication between all the processes of a process group (MPI_Bcast, MPI_Reduce and MPI_alltoall...). Other operations perform more sophisticated tasks, such as that rearranging n data items.
4) Derived data types predefined constants (MPI_INT, MPI_CHAR, MPI_DOUBLE) so that MPI can support heterogeneous environments where types data can be represented differently on different nodes. [10]

3.2.4. Compute Unified Device Architecture (CUDA)

Represents an API created by Nvidia that allows software to use certain types graphics processing units (GPUs) for gen-

eral-purpose processing, this platform parallel computing form is a software layer that provides direct access to the GPU, this is an approach called General Purpose Computing on GPU (GPGPU) [11]. This eliminates the bottlenecks caused by software running on the host CPU, resulting in a dramatic increase in application performance. CUDA allows programs to be compiled into device code that can run directly on GPUs [12].

CUDA is designed to work with programming language such as C, C++and Fortran, Third-party wrappers are also available for Python, Java, R, MATLAB. This accessibility allows specialists in parallel programming to more easily use GPU resources, unlike earlier APIs like Direct3D and OpenGL, that required advanced graphics programming skills. Powered GPUs by CUDA also support programming frameworks such as OpenMP, OpenACC and OpenCL and HIP by compiling this code in CUDA [12].

Graphics Processing Units (GPUs) are highly parallel allowing efficient manipulation of large blocks of data. This design has makes them a more efficient tool than general-purpose (CPUs) for algorithms in situations where the processing of large blocks of data is carried out in parallel (learning automation, cryptographic hash functions, molecular dynamics simulations, physics engines...). CUDA provides both a low-level API (CUDA Driver API, non-single source) and higher level API (CUDA Runtime API, single source) and handles a collection of libraries, tools, and technologies that deliver performance high in multiple application areas ranging from artificial intelligence to computing high performance [11].

CUDA has several particularities compared to parallel programming, in particularly are:

1) explicit prioritization of memory areas (private, local, global) allows so much to finely organize the transfers between them.
2) grouping threads into grids of grids (local 1D, 2D or 3D grids threads that can quickly share local memory) Local grids are ordered in a global grid allowing access to the global memory.

3.2.5. MapReduce

Represents a distributed programming API for processing massive data in parallel on clusters of servers. It was introduced by Google in 2004 and is become popular for data processing on large amounts of data. The tasks are distributed on the different nodes of the cluster for efficient parallel execution. MapReduce is often used in conjunction with the Hadoop Distributed File System to process data in parallel on large server clusters [13].

MapReduce is used to process large amounts of data across many parallel computers. This allows complex tasks to be broken down into simple sub-tasks which can be perf languages, such as Java, python, Ruby, and C++. It is a powerful tool for data processing applications in mass and is widely used in industry [14]. The MapReduce model consists of two main steps:

1) The "Map" step which takes as input a list of key-value pairs and returns a list intermediate key-value pairs. It filters and distributes the work on different nodes of the cluster or map, using a function sometimes called a mapper.
2) The "Reduce" step which takes as input the list of intermediate key-value pairs and returns a list of final key-value pairs. It organizes and reduces the results of each node into a consistent response to a query, using a function called reducer.

3.3. Comparative Table Between APIs

To better understand the differences between the APIs, we present the following table:

Table 2. *Comparison between APIs.*

POSIX	OpenMP	MPI	CUDA	MapReduce
USE				
Programming systems	Parallel treatment on a single processor	Treatment parallel on several processor	Treatment by- runs on GPU	Distributed of treatment on data
LANGUAGE				
Fortran, C, C++	Fortran, C, C++	Fortran, C, C++	Fortran, C, C++	Java,Python,C++
PLATFORM				
Executive hardware	Processor	Cluster	GPUs	Hadoop
ABSTRACTION				
-	-	-	+	+ +
COMMUNICATION				
Shared memorygey	Shared memory	Network	Shared memory	Network

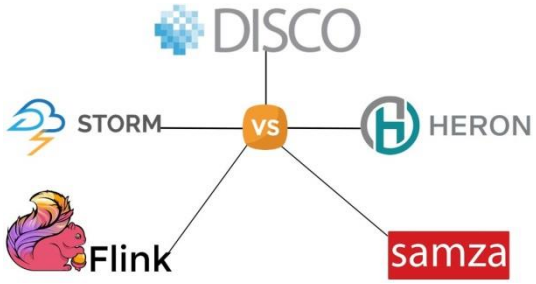

Figure 2. Big data frameworks.

4. Big Data Processing Framework's

The rapid growth of digital data generated by various sources makes it inefficient the use of traditional methods of storage, processing and analysis. These limitations have led to the development of new technologies to process and store very large data sets. As a result, several implementation frameworks have emerged to big data processing. We will focus in this research on the comparison of the most important and most frequently used frameworks in the free software landscape [25, 27].

4.1. STORM

Apache Storm is a distributed processing framework for streaming data in time real. Storm was originally developed by Nathan Martz of Back Type, acquired by Twitter in 2011 and was incorporated into Apache projects later in 2014. Apache Storm is designed to process and analyze large amounts of unlimited data streams that can come from various sources and post real-time updates to the user interface or other locations without storing any actual data. Storm is highly scalable in nature and offers low latency with an easy-to-use interface through which developers can program in virtually any programming language [15].

4.2. DISCO

Disco open-source framework was developed in 2008 at Nokia research center in order to meet the real challenges of handling massive amounts of data and the framework has since been actively developed by Nokia. Disco is the latest addition to the list growing use of Big Data tools, which allows the parallel processing of large amounts of data and using Python makes it robust and easy to use [16].

4.3. HERON

Apache Heron is a distributed stream processing framework developed by Twitter. According to the creators of Twitter, the scale and diversity of Twitter data has increased, and Heron is a real-time analytics platform for processing streaming and scalability. It was presented at SIGMOD. Heron is compatible with the Apache Storm API [17].

4.4. SAMZA

Developed internally at LinkedIn in 2013 and later donated to Apache Software Foundation. Samza is now adopted by several large companies it is based on a unified design for stateful processing of batch data and data streams in real time at high volume. Samza is designed to support high throughput for streams data while providing fast fail over and high reliability. For at-To tint these design goals, Samza uses some key abstractions, such as partitioned streams, change log capture, and local state management [18].

4.5. FLINK

Apache Flink is a distributed processing framework for processing of unlimited data streams. Flink was started in 2009 at the Technical University of Berlin under the name Stratosphere. Flink is known to process data a hundred times faster than MapReduce. Thanks to a very flexible windowing mechanism, Flink programs can calculate early and approximate results, as well as delayed and precise results of early and approximate results, as well as late and precise results through the same process of combining different systems for both use cases [19].

4.6. Comparison between Big Data Processing Frameworks's

Table 3. Comparison of different frameworks according to architecture and processing model.

ARCHITECTURE				
FLINK	**SAMZA**	**STORM**	**HERON**	**DISCO**
A four-layer architecture. The core is a motor data flow distributed who	Three-layer architecture. A streaming layer that is responsible for the sup-	Architecture-based on master/slave, it allows only one node master.	The overall architecture of Heron based on usage of API Storm used to	Architecture who is composed of two elements main: the file

receives the programs of the users. Flink possesses two APIs main: API data set and the Stream API data. Flink is a processing system stream does not offer storage; it is designed to read data from various streaming systems and various storage.	ply of a source replayable data, the execution layer which is responsible of scheduling and the management of resources, and the treatment layer is responsible for data processing and stream management.	Three components main. Nimbus, Zookeeper and the Super view finder. Nimbus is a master who distributes the work between all available workers Zookeeper is used for all the coordination between Nimbus and supervisors.	create and submit topologies to a planner. The planner executes each topology act like a task composed of several containers. One of the containers executed the topology Master, that is responsible topology management.	system distributed Disco (DDFS). The DDFS provides access to data of the application, MAP/Reduce is responsible of the task combinations chained and scheduling.
PROCESSING MODEL				
FLINK	SAMZA	STORM	HERON	DISCO
The programs Flink are compiled and converted to graphic flow data. Each edge represents a flow of data, and each vertex represents a operator that used a logic defined by the application for data processing.	Consists of four abstractions, topology, spouts, bolts, stream. The bolts are the units of treatment of one application of Storm which can perform various tasks.	Consists of four abstractions, topology, spouts, bolts, stream. The bolts are the units of treatment of one application of Storm which can perform various tasks.	Heron is suitable for processing engine flow developed by Twitter and given to Apache. According to Twit- ter, the flow rate is 10-14x higher than that of Storm in all experiences, its latency is lower to that of Storm.	MapReduce is the only framework of data processing for Disco. The phases Map and Reduce have each their own flow concepts data. Thanks to task combinations chained, Disco supports a remarkable variety of data flow.

Table 4. *Comparison of different frameworks according to scheduling and fault tolerance.*

SCHEDULING				
FLINK	**SAMZA**	**STORM**	**HERON**	**DISCO**
There are three strata planning strategies in Flink for the allocation resource: at the same time planning, planning paresuse from sources planning and pipeline region. All planning strategies tion tries at the same time to allocate resources sources needed in only once when the task begins.	For planning tasks and the resource negotiation, Samza relies on managers of clusters like YARN and mesos.	Storm has four built-in schedulers: Default, Insulation, Multitenant, and Resources Aware. Tea default Storm scheduler is fair scheduler that takes into account each node when scheduling tasks.	By disregarding order component nuancing, we have ease the deployment on a infrastructure shared running different order frames such as Mesos, YARN or a personal environment read.	The scheduler of DISCO done part Of the manager of resources and is purely responsible of scheduling of the resources for the execution of applications. The scheduler offer two policies scheduling, FIFO and Fair.
FAULT TOLERANCE				
FLINK	SAMZA	STORM	HERON	DISCO
Flink provides a reliable execution with guarantees of cohesion "exactly-once". It takes regularly of snapshots distributed from the flow of data and states opertors. These snapshots serve checkpoints coherent, in case of failure, the system can go back to these checkpoints.	Samza executed in background a service log capture of the changes Who checked in THE changes incre-mental to a place known to the system native files. A failed task maybe restarted replaying the news paper of the changes. When a container restart after a failure, it research the last points of check and start to accept messages from the last point of control.	Nimbus and the Super visors are designed like demons rapid, but resilient and stateless, with all their stored data in Zookeeper or on local disks, avoiding. Thus all loss catastrophic in case of failure processes or processor.	Highly tolerant to break downs with data stored Zookeeper or on the local disks, Avoiding all loss catastrophic in case of failure processes or processor.	Disco is highly fault tolerant. The DDFS ensures the fault tolerance using the replication of data. The data blocks stocked in the Data-Nodes are replicated and distributed in the cluster to ensure fiability and high availability.

Table 5. Comparison of different frameworks according to speed.

SPEED				
FLINK	**SAMZA**	**STORM**	**HERON**	**DISCO**
It handles speeds more fast by allowing to iterative processing take place on the same node rather than having the cluster who performs them independently.	The tasks in Samza may present a latency of the order of a few milliseconds when they are run with Apache Kafka. The approach of memorization buffer is different.	Due to the treat-lie in almost in real time. Storm processes data with very low latency for give a result with minimal delay.	Heron is built with a wide range improvements architectural with an execution engine fast, latency can be as weak as possible.	Due to the treatment in batches of one large volume of data, Disco takes more computation time, which means that the latency is more important aunt and that Disco is relatively slow.

5. Parallel and Distributed Storage Systems

There are several systems for parallel and distributed file storage such as Hadoop Distributed File System (HDFS) which is the storage manager in Hadoop. It makes it possible to store the different blocks of partitions on the different nodes of the distributed architecture. Tachyon which is a memory-centric distributed storage system similar to HDFS and many other systems like [24-27].

5.1. GlusterFS (Gluster File System)

It Is an open-source distributed high-availability file system that can scale from modular way to store several petabytes of data with a simple structure (two software elements): server and client. This is commercial support provided by redhat which is accessible through NFS or FUSE. Scale-out storage systems based on GlusterFS are suitable for unstructured data. data such as documents, images, audio and video files, and log files [20].

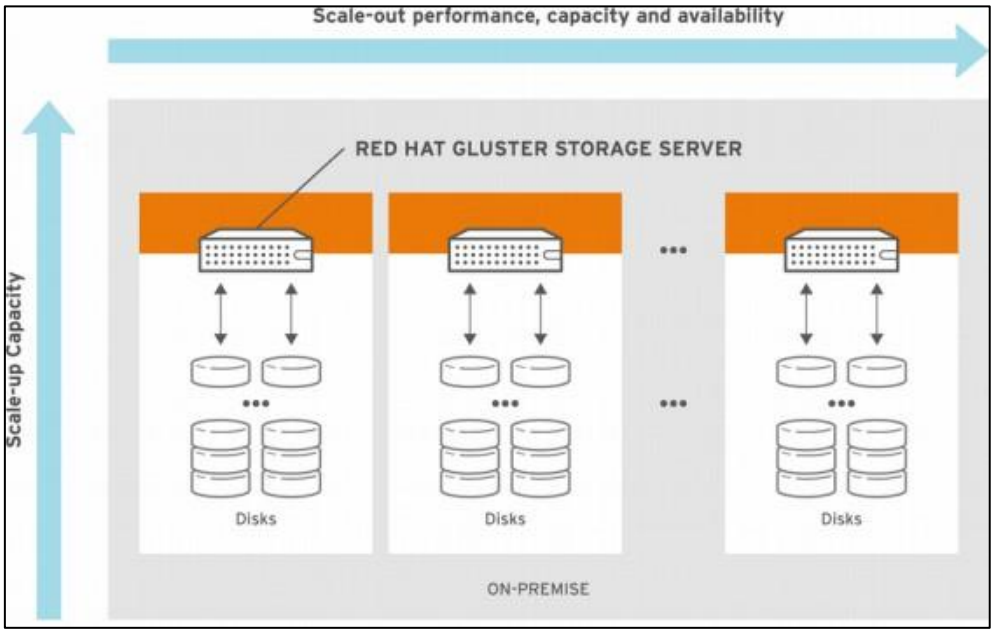

Figure 3. The architecture of a GlusterFS.

5.2. CEPH

It is a free distributed storage platform designed for high availability, using readable by any type of service, the three types of storage are available (object, block and file). CEPH uses a hashing algorithm to determine the location of a block and allows a CEPH client to talk directly to the ods service in

order to access data, CEPH supports "erasure-coding" as well as replication to avoid data loss and possible to use with Hadoop. It is built of two types of servers: Server Monitor (MON) (3 Minimum), and Object Storage Server (OSD) [21].

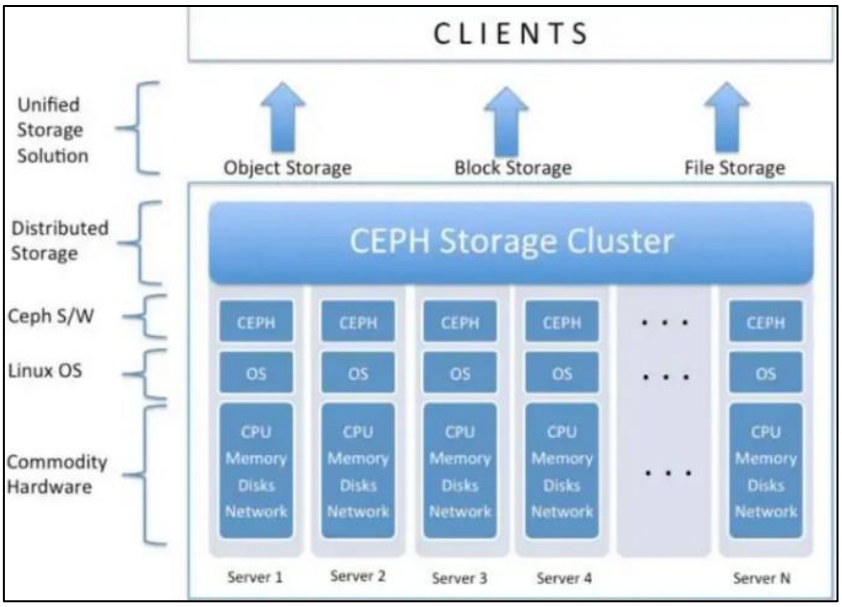

Figure 4. *The architecture of a CEPH.*

5.3. NFS (Network File System)

It is a distributed file system protocol originally developed by Sun Micro historical network sharing systems under Unix.NFS follows the client-server computing model. The server implements the file system shared and the storage to which the clients connect, the clients implement the interface user on the shared file system, mounted in the client's local file space [20].

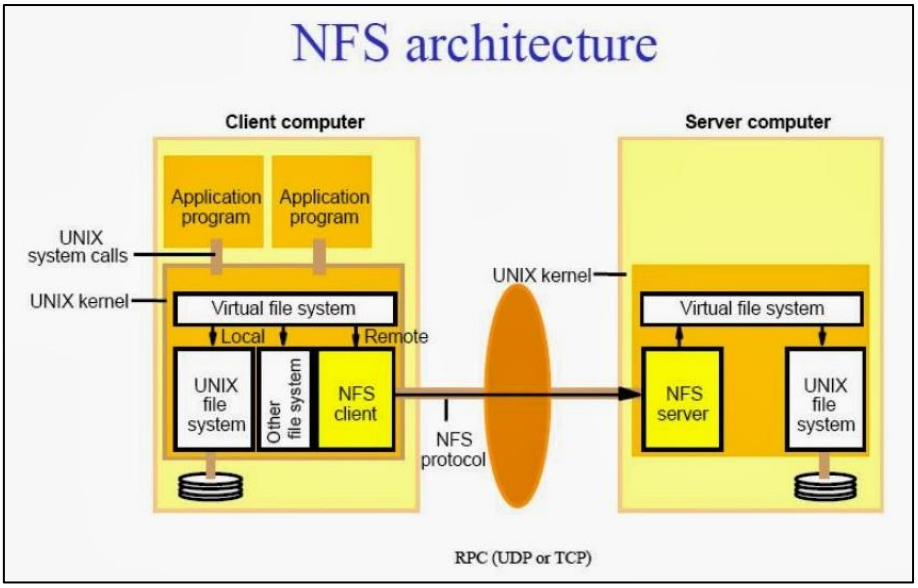

Figure 5. *The architecture of an NFS.*

5.4. BeeGFS

It is a parallel cluster file system, it combines multiple storage servers to provide a highly scalable shared network file system with content from interlaced files BeeGFS is an Open Source software (and not free for the server part). In

such a system, high throughput demands of a large number of customers can easily be met, but a single customer can benefit from the aggregated performance of all storage servers of the system, there are 3 types of servers: Management Server for Configuring and Monitoring Services, Metadata Server Dedicated to File Metadata, Data Server [22].

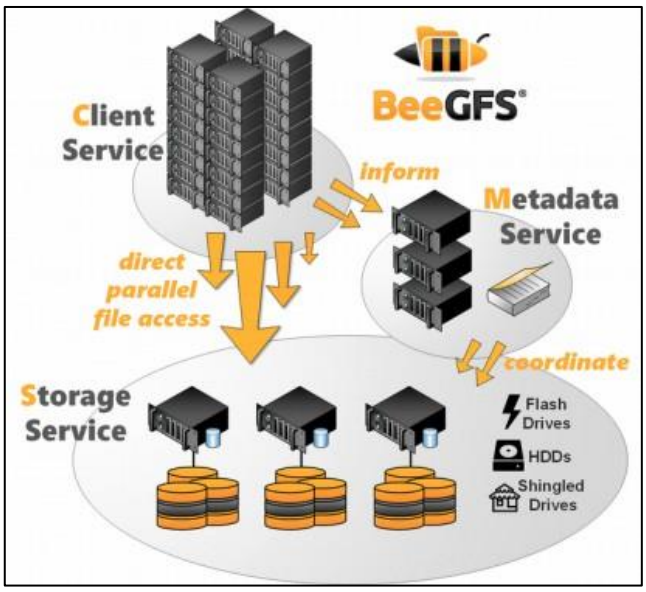

Figure 6. The architecture ofBeeGFS process.

5.5. MooseFS

It is a fault-tolerant distributed file system, developed by Genius SA, it is based on the use of FUSE, the file system is POSIX compatible and does not require adaptation of the programs to be able to use it. The system is distributed: the files are cut into packets of 64 MB and distributed over the machines called "*chunk server*», or data servers. MooseFS consists of three types of servers [20, 23]:

1) The MasterServer
2) Met logger Server
3) The Chunck Server

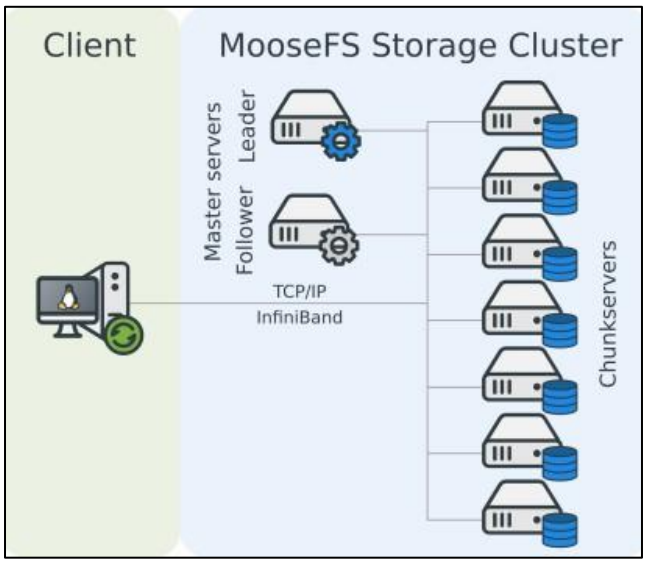

Figure 7. The architecture of MooseFS Process.

5.6. IBM GPFS (IBM General Parallel File System)

It is a file system used to distribute and manage data across multiple servers, and is implemented in many computing environments for high large-scale performance and storage. It is described as a file system parallel because GPFS data is divided into blocks and spread across multiple disks in an array, then read in parallel when accessing the data. This allows speeds of higher reading and writing.

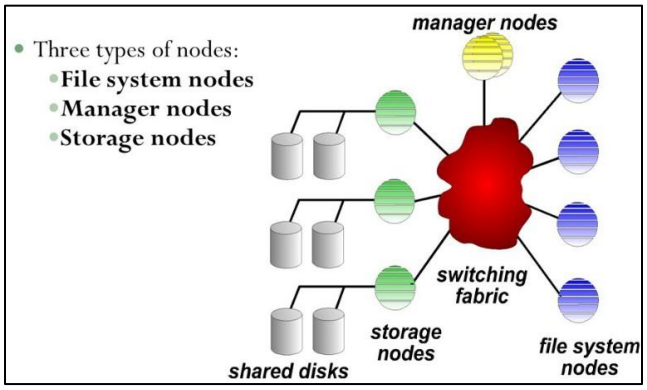

Figure 8. The architecture of an IBM GFS.

5.7. A Comparison of Previous Storage Systems

Table 6. Comparison of different distributed storage systems.

	GlusterFS	CEPH	NFS	BeeGFS	MooseFS	IBMGFS
Get to API	libglusterfs,	Librados	NFS	POSIX	POSIX, FUSE	POSIX, NFS,

	GlusterFS	CEPH	NFS	BeeGFS	MooseFS	IBMGFS
	NFS, Swift FUSE	(C,C++, Swift, FUSE)				HDFS, FUSE, SMB
open source	YES	YES	YES	Customer=YES Server=EULA	YES	NO
MDSnecessary	NO	NO	NO	Metadata+ Manage	Metadata+ Manage	NO
Tools of move- lies of data	YES	YES	NO	YES	NO	YES
Easy to put in place	+ +	+	+ +	+	+	+
Consistency of data	+ +	-	–	+ +	+ +	+ +
Safety and availability of data	+	+ +	–	+ +	+ +	+ +
Scalability	+	+ +	–	+	+ +	+
performance	+	+ +	+	+ +	+	+ +
Replication of data	YES	YES	NO	YES	YES	YES
Cost	+	+ +	-	+	-	+ +

6. Future in Big Data Analytics

Future trends in big data analytics focus on the integration of parallel processing techniques with advanced distributed storage solutions. Key trends include:

1) Increased Scalability: As data volumes grow, frameworks will evolve to better handle large-scale data processing by effectively distributing workloads across resources.
2) Enhanced Performance: Innovations in algorithms and hardware will drive improvements in processing speeds and efficiency, making real-time analytics more feasible.
3) Cloud Integration: The shift towards cloud-based services will facilitate more flexible and cost-effective storage and processing options, catering to diverse data needs.
4) Deep learning Integration: There will be a rise in frameworks that support deep learning workloads, allowing for better data-driven insights and predictive analytics.
5) Hybrid Architectures: The combination of on-premises and cloud resources in hybrid architectures will gain traction, allowing organizations to optimize their data processing strategies.
6) Improved Data Governance: As data privacy concerns increase, frameworks will incorporate more robust data governance and security measures to protect sensitive information.
7) Interoperability: There will be a push for greater interoperability among different frameworks and tools, enabling more seamless data processing across varied systems.

Overall, the future landscape of big data analytics will be characterized by more efficient, flexible, and intelligent processing methods, driven by ongoing technological advancements.

7. Conclusion

Big Data is a technology that provides significant advantages to businesses, including enhancements in business processes, improved decision-making, and increased operational efficiency. The advantages of Big Data are numerous, enabling companies to boost their performance and make better-informed decisions. Although Big Data can be intricate and demanding in terms of resources, it can also be very lucrative and contribute to long-term business success. We've observed that parallel processing of large datasets is an effective and cost-efficient method to accelerate the handling of substantial volumes of data. This approach facilitates the execution of multiple tasks simultaneously, reducing processing times. It is especially beneficial for applications that require simultaneous data processing.

Indeed, the benefits of parallel processing are substantial, which encompass quicker processing times, more efficient utilization of IT resources, and reduced processing costs. Nonetheless, parallel data processing can also introduce challenges, especially related to synchronization and communication among the various nodes. To maximize the advantages of parallel data processing, it is crucial for developers and administrators to grasp the underlying principles and techniques associated with this technology. In addition to various frameworks and distributed data storage systems, this form of storage enables users to store and share data across multiple systems or network devices. It provides enhanced scalability and improved data availability, with data being spread across several systems or connected devices. Distrib-

uted storage is particularly well-suited for large information systems and organizations that require the management of significant volumes of data. Furthermore, it offers increased flexibility and heightened data security.

Abbreviations

ETL	Extract Transform Load
CDC	Change Data Capture
SDI	Stream Data Integration
Pthreads	POSIX Threads
OpenMP	Open Multi-Processing
MPI	Message Passing Interface
CUDA	Compute Unified Device Architecture
GlusterFS	Gluster File System
NFS	Network File System
IBM GPFS	IBM General Parallel File System

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] Souibgui, M., Atigui, F., Zammali, S., Cherfi, S., & Yahia, S. B. (2019). Data quality in ETL process: A preliminary study. Procedia Computer Science, 159, 676-687. https://doi.org/10.1016/j.procs.2019.09.223

[2] Yang, S., & Kim, J. K. (2020). Statistical data integration in survey sampling: A review. Japanese Journal of Statistics and Data Science, 3(2), 625-650. https://doi.org/10.48550/arXiv.2001.03259

[3] Butenhof, D. R. (1993). Programming with POSIX threads. Addison-Wesley Professional.

[4] Shen, J. P., & Lipasti, M. H. (2013). Modern processor design: fundamentals of superscalar processors. Waveland.

[5] Culler, D., Singh, J. P., & Gupta, A. (1999). Parallel computer architecture: a hardware/software approach. Gulf Professional Publishing.

[6] Castelló, A., Gual, R. M., Seo, S., Balaji, P., Quintana-Orti, E. S., & Pena, A. J. (2020). Analysis of threading libraries for high performance computing. IEEE Transactions on Computers, 69(9), 1279-1292. https://doi.org/10.1109/TC.2020.2970706

[7] Silberschatz, A., Galvin, P. B., & Gagne, G. (2012). Operating system concepts.

[8] OpenMP, A. R. B. (2013, July). OpenMP application program interface version 4.0. In The OpenMP Forum, Tech. Rep.

[9] Nielsen, F., & Nielsen, F. (2016). Introduction to MPI: the message passing interface. Introduction to HPC with MPI for Data Science, 21-62. https://doi.org/10.1007/978-3-319-21903-5_2

[10] Sur, S., Koop, M. J., & Panda, D. K. (2006, November). High-performance and scalable MPI over InfiniBand with reduced memory usage: an in-depth performance analysis. In Proceedings of the 2006 ACM/IEEE conference on Supercomputing (pp. 105-es). https://doi.org/10.1109/SC.2006.34

[11] Tuomanen, B. (2018). Hands-On GPU Programming with Python and CUDA: Explore high-performance parallel computing with CUDA. Packt Publishing Ltd.

[12] Abi-Chahla, F. (2008). Nvidia's CUDA: The End of the CPU?. Tom's Hardware, (s 15).

[13] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113. https://doi.org/10.1145/1327452.1327492

[14] Hashem, I. A. T., Anuar, N. B., Gani, A., Yaqoob, I., Xia, F., & Khan, S. U. (2016). MapReduce: Review and open challenges. Scientometrics, 109, 389-422. https://doi.org/10.1007/s11192-016-1945-y

[15] Laku, L. I. Y., Mohammed, A. F. Y., Fawaz, A. H., & Youn, C. H. (2019, February). Performance Evaluation of Apache Storm With Writing Scripts. In 2019 21st International Conference on Advanced Communication Technology (ICACT) (pp. 728-733). IEEE. https://doi.org/10.1007/978-3-030-79478-1_24

[16] Mundkur, P., Tuulos, V., & Flatow, J. (2011, September). Disco: a computing platform for large-scale data analytics. In Proceedings of the 10th ACM SIGPLAN workshop on Erlang (pp. 84-89). https://doi.org/10.1145/2034654.2034670

[17] Wu, H., & Fu, M. (2021). Heron Streaming: Fundamentals, Applications, Operations, and Insights. Springer Nature.

[18] Gürcan, F., & Berigel, M. (2018, October). Real-time processing of big data streams: Lifecycle, tools, tasks, and challenges. In 2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT) (pp. 1-6). IEEE. https://doi.org/10.1109/ISMSIT.2018.8567061

[19] Friedman, E., & Tzoumas, K. (2016). Introduction to Apache Flink: stream processing for real time and beyond. "O'Reilly Media, Inc."

[20] Baker, M. G., Hartman, J. H., Kupfer, M. D., Shirriff, K. W., & Ousterhout, J. K. (1991, September). Measurements of a distributed file system. In Proceedings of the thirteenth ACM symposium on Operating systems principles (pp. 198-212). https://doi.org/10.1145/121133.121164

[21] Jin, L., Zhai, X., Wang, K., Zhang, K., Wu, D., Nazir, A., … & Liao, W. H. (2024). Big data, machine learning, and digital twin assisted additive manufacturing: A review. Materials & Design, 113086. https://doi.org/10.1016/j.matdes.2024.113086

[22] Abramson, D., Jin, C., Luong, J., & Carroll, J. (2020, February). A BeeGFS-based caching file system for data-intensive parallel computing. In Asian Conference on Supercomputing Frontiers (pp. 3-22). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-48842-0_1

[23] Liu, M. (2024). Key Technology of Distributed Memory File System Based on High-Performance Computer. International Journal of Cooperative Information Systems, 33(02), 2350019. https://doi.org/10.1142/S0218843023500193

[24] Mezzoudj, S., Behloul, A., Seghir, R., & Saadna, Y. (2021). A parallel content-based image retrieval system using spark and tachyon frameworks. Journal of King Saud University-Computer and Information Sciences, 33(2), 141-149. https://doi.org/10.1016/j.jksuci.2019.01.003

[25] Saliha, M., Ali, B., & Rachid, S. (2019). Towards large-scale face-based race classification on spark framework. Multimedia Tools and Applications, 78(18), 26729-26746. https://doi.org/10.1007/s11042-019-7672-7

[26] Mezzoudj, S. (2020). Towards large scale image retrieval system using parallel frameworks. In Multimedia Information Retrieval. IntechOpen. https://doi.org/10.5772/intechopen.94910

[27] Saadna, Y., Behloul, A., & Mezzoudj, S. (2019). Speed limit sign detection and recognition system using SVM and MNIST datasets. Neural Computing and Applications, 31(9), 5005-5015. https://doi.org/10.1007/s00521-018-03994-w

[28] Meriem, K., Saliha, M., Amine, F. M., & Khaled, B. M. (2024). Novel Solutions to the Multidimensional Knapsack Problem Using CPLEX: New Results on ORX Benchmarks. Journal of Ubiquitous Computing and Communication Technologies, 6(3), 294-310. https://doi.org/10.1007/11499305_