ISSN:0975-9646
Lacine Kabre et al/ (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 15 (5), 2024, 31-39

Big data storage and processing with a peer-to-peer Content Addressable Network Protocol

Lacine KABRE[1*] and Tiendrebeogo Telesphore[2†]

[1]Exact and applied science, Joseph Ki-Zerbo,*
03 BP 7021 Ouagadougou 03, Ouagadougou, 7021, Burkina Faso.
Email: contact@ujkz.bf

[2]Exact and Applied Science,
Nagzi Institute University, Bododioulasso, Burkina Faso.
info@u-naziboni.bf,

*Corresponding author(s). E-mail(s): lacinekabre1@gmail.com
Contributing authors: tetiendreb@gmail.com
†These authors contributed equally to this work.

Abstract

Big Data refers to enormous amounts of heterogeneous data from various traditional and new data sources. These data are constantly changing. However, due to their great heterogeneity, several architectural approaches have been proposed to centrally process and analyze large amounts of data. For data processing, platforms using MapReduce are also designed for data centers, which are generally centralized. These platforms generally have a single node to maintain and coordinate MapReduce tasks, leading to a single point of failure. All of these architectures have advantages and limitations, especially in terms of time and processing mode. In our paper, we propose a decentralized architecture based on a content addressable network peer-to-peer protocol using image properties associated with SHA-512 keys to secure the data. First, we evaluated the existing Big Data technologies from the perspective of architecture to the use of a data storage model. Then, we presented all the conceptual aspects of our architecture, and we ended with the evaluation of a symbolic version developed and deployed in a local network. We implemented the MapReduce algorithm to evaluate the RGB architecture for data processing by the peer node. Our study showed that a decentralized environment can be created for big data processing via the P2P protocol.

Keywords: CAN protocol, MapReduce, RGB architecture, Big data storage.

1. Introduction

Today, the world population is estimated to be 8 billion, and more than 3.4 billion people are connected to the internet according to "internet live stats" [1]. For the last twenty years, the amount of generated data has been increasing rapidly. We produce annually very important data estimated at nearly 3 trillion (3.1018) bytes of data. It is estimated that in 2016, 90% of the world's data were created in the previous two years. It is in this context that the term "big data" appeared[2]. The term "big data" refers to databases that are too large and complex to study with traditional statistical methods and, by extension, to all the new tools for analyzing these data. In this article, we focus on the collection, storage and processing of massive amounts of data in a P2P network context using the CAN protocol.

Big data architectures such as lambda[3] or kappa use datalakes or smart data for batch or streaming processing. The MapReduce(4) computing model is also much better suited to deployment in data centers with highly centralized architectures.

Although omnipresent in our current events, massive amounts of data and artificial intelligence are new phenomena and are sometimes difficult to define[2]. Big data is therefore a new discipline that brings together techniques for managing and capturing data and tools and platforms for storing, preprocessing, processing and securing data. The fastest growing data type is unstructured data. This type of data is characterized by human information such as high-definition videos, movies, photos, scientific simulations, financial transactions, phone records, genomic datasets, seismic images, geospatial data, maps, email, tweets, Facebook data, call center conversations, cell phone calls, website clicks, documents, sensor data, telemetry, medical records and images, weather data records, log files and text. In the first part of this article, we presented related work on big data technologies from storage to processing. In the second part, we recalled the operating principle of the CAN peer-to-peer protocol. In the third part, we propose, implement and evaluate an architecture for storage and big data processing with MapReduce based on a CAN P2P network protocol [7].

2. Related Work

Big data storage

Big Data storage is generally based on an architecture that allows calculations to be performed and large amounts of data to be managed. In most cases, Big Data storage uses low-cost hard disks [5]. In big data, the data are mostly unstructured, which means that the storage is primarily file- and object-based. Although a specific volume or capacity is not formally defined, Big Data storage generally refers to volumes that grow exponentially on the terabyte or petabyte scale [6]. In addition, there are three storage methods: file storage, block storage and object storage. Object storage is the most widely used data storage method in the Cloud. It is a nonhierarchical storage method generally used for cloud storage[7]. It consists of storing data in the form of units called objects. Each object consists

of the data, a unique identifier and metadata. The unique identifier corresponds to the access path of the data. Block storage consists of dividing a file into several blocks of data and then storing them separately. This makes it possible to distribute the blocks within the storage system in a more efficient way. To retrieve a file, the system reassembles the blocks of the file[7]. Most of these storage infrastructures support Hadoop and NoSQL storage solutions. In file storage, the data are stored in a file unit and placed in a folder. It is a hierarchical storage method(8).

Big data storage models

The advent of big data(6) has also given birth to a series of technologies in data storage, including Not Only SQL (NoSQL)(9). NoSQL is a technology that was developed to solve the problems presented by relational databases. This technology (NoSQL) is implemented in different ways and has different models. The common characteristics of these models include efficient storage, low operational costs, high availability, high concurrency, minimal management, high scalability and low latency. In 2019, Khan, Samiya, and Liu conducted a qualitative study to rank NoSQL solutions based on several criteria (10). Rinkle Rani and other researchers(11) have provided a detailed classification of NoSQL solutions by dividing them into nine categories: columnar storage, document storage, object databases, columnar storage, data structure server, key-value storage, cached key-value storage, ordered key-value storage and finally coherent key-value storage. Cloud solutions can be classified into six categories: entity-attribute and value data storage, Amazon Platform columnar storage, key-value storage and document storage with a distributed hash table(11).

Key-Value Storage Model

In this section, we are interested in the NoSQL database type with a key-value storage, which implements a schema-less storage policy. The structure is formally defined for data storage. The data can be strings, numbers, images, binaries, XML, JSON, HTML or video format, in addition to many others. The stored values can be accessed by using a key. The flexibility of the database manifests by having the application completely control the value of the data[12]. The main advantages of key-value storage are as follows:

- The database does not force the application to structure its data in a specific form. Therefore, the application is free to model its data according to the requirements of the use case.
- Access to the objects occurs simply through the key assigned to the object. When using this database, it is not necessary to perform operations such as union, join and lock on the objects, which makes this data model more efficient and highly effective.
- Most of the available key-value databases allow you to adapt according to the demand.
- These databases are designed such that they are easy to add and remove. In addition, these databases are better equipped to handle network and hardware failures, which greatly reduces downtime.

CAN Protocol

The content addressable network (CAN)[13 protocol is a peer-to-peer network protocol in which distributed hash tables (DHTs) are used. The CAN design described is based on a virtual Cartesian coordinate space (Figure 2). This coordinate space is completely logical and has no relation to any physical coordinate system. The coordinate space is shared dynamically among all the nodes in the system so that each node has its own individual distinct area in the space.

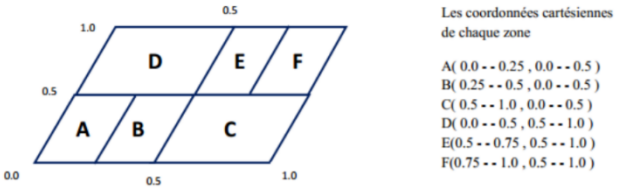

Figure 1: CAN coordinate space [14]

In the space managed by the CAN protocol, each portion of the space can be adopted by a node, and each node acquires information about its directly connected neighbors. The routing in the CAN is performed from close to close by passing through all the neighbors until arriving at the target peer. To make a hop, the node receiving the request communicates it to the neighbor whose coordinates overlap on d-1 dimensions and are contiguous on the remaining dimension. At each hop, one can change coordinates in only one dimension[14]. This virtual coordinate space is used to store (key, value) pairs. To store a pair of information (key, Value), the key is determined in a deterministic way on a point *P* in the coordinate space using a uniform hash function. The corresponding pair (key, value) is then stored on the node that owns the field. To retrieve an entry corresponding to a key, any node can apply the same deterministic hash function to map the given key onto the point *P* and retrieve the corresponding value. If the point is not owned by the requesting node or its immediate neighbors, the request must be routed through the CAN infrastructure until it reaches the node in the area where the information peer is located. Efficient routing is therefore a critical aspect of the CAN protocol. The nodes in the CAN automatically organize themselves into an overlay network that represents this virtual coordinate space. A node learns and maintains the addresses of nodes that contain coordinate areas adjacent to its own area. This set of nearest neighbors in the coordinate space serves as a coordinate routing table[14]. Our comparative study[15] performed using the simple network layer in the PeerFactSim simulator showed a discrepancy between the theoretical values and the values from the simulation in terms of routing. The number of hops was greater than the number estimated theoretically. For a space divided into n equal areas, the average routing path length is *O(d/4) (n 1/d),* with hops and individual nodes maintaining 2d neighbors. These scaling results mean that for a d-dimensional space, we can increase the number of nodes (and thus the areas) without increasing the state of the nodes[14]. However, the average path length increases and can tend to *O(n 1/d).*

Our comparative study[15] allowed us to analyze initialized messages and correctly transmitted messages to determine the average number of failed messages as a function of time.
The study showed that the CAN protocol sends more messages. However, this number decreases slightly with time, and the number of failed messages is low. The number of messages sent by Pastry increases with time, and the number of failed messages increases proportionally with the number of messages sent. Chord sends fewer messages and does not fail. We concluded that the CAN protocol has a longer routing time than the other protocols but ensures that a very large number of messages are successfully sent. This analysis oriented our choice to the CAN protocol for the implementation of a self-managing and scalable Big Data architecture.

3 PROBLEMATIC

Currently, the demand for storage solutions is increasing due to the challenges of managing data for connected devices, sharing data and customizing applications. There has been a shift toward data-intensive applications, especially because of the dependence of companies on large amounts of data. However, these data are stored in centralized data centers, which leads to multiple security problems and problems with the architecture of the storage environment. This argument supports the use of decentralized storage solutions. This argument supports the use of decentralized storage solutions. We also have cloud storage solutions that have a centralized architecture and are therefore easy to target for hackers. A decentralized system is devoid of such problems since the nodes are geographically distant and physically
independent. Therefore, a failure or attack at a point of failure should only affect the affected node. The overall system will remain unchanged, as the other nodes will continue to operate as usual. In addition to ensuring that the system is reliable and available, decentralization also contributes to system scalability and performance. However, massive data distribution in a decentralized environment presents significant challenges, especially in terms of access time when the nodes that compose it are operating beyond their capacity. In this research, we propose a decentralized architecture for scalable, highly available and securely stored massive amounts of data. Its particularity lies in the use of distributed hash tables (DHTs) in the peer-to-peer protocol Content Adressable Network (CAN) and the use of image properties coupled with the SHA-512 key to secure the data.

4. CONTRIBUTION

Big Data architecture based on the CAN Protocol

The purpose of Big Data architecture is not only to store data but also to make it available for other applications to exploit and extract value from it. It must be possible to perform customized analyses on these data in an easy way. Therefore, the time needed to access the data is crucial. A big data architecture is also an organization of a set of technologies allowing us to collect and exploit the data. These technologies define the environment of the architecture. The main objective of this architecture proposal is to ensure a minimum of security both at the level of the data in transit and at the level of the various components. In this paper, we propose a massive data storage architecture based on the P2P CAN protocol to improve access time and facilitate access to data. In this architecture, we also add an additional layer of security by using SHA-512 keys (512 bits) and image properties. This architecture is called the RGB architecture. A comparative study(15) on the P2P protocols CAN, Pastry, Chord and Kademlia showed us that the CAN protocol is effective at delivering messages compared to the other protocol. The successful delivery of messages implies a rate of success of storage requests compared to those of Chord and Pastry. These results show that CAN could guarantee a very low failure rate in the storage of big data objects.

RGB Architecture

As shown in the image (Figure 2), our model has one main node and three secondary nodes. The main node is called the bootstrap node, and the other three nodes are called bootstrap secondary nodes. The initialization of the architecture is performed by the BootStrap node, which assigns a label and an address to each secondary node. These labels are R, V, and B. Each label is associated with a secondary node. The secondary nodes can in turn initialize other nodes called data nodes. Our architecture is a key-value-oriented Big Data storage solution that takes advantage of the CAN protocol properties for node management.

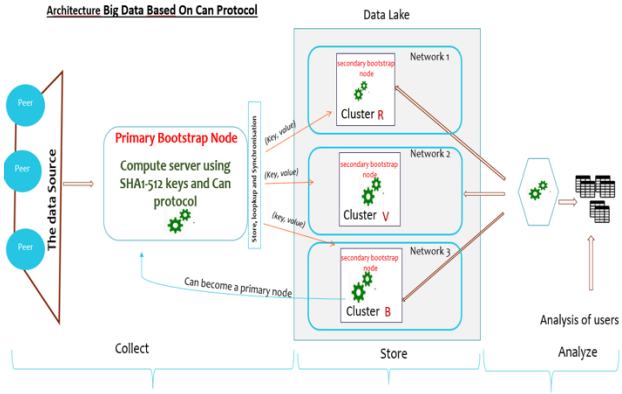

Fig **2**:Big data Architecture based on CAN protocol

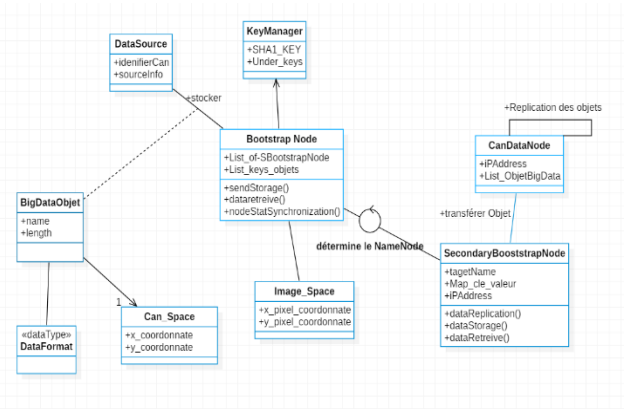

Fig 3: Basic UML diagram class for the RVB Architecture

Operating principle

4.3.1 The storage of Big Data objects

For the majority of the numerical applications consisting of representative images on a screen, the trichromic basis of additive synthesis RGB (red green blue) is used (screens with cathode ray tubes or electroluminescent diodes use these three primary colors to synthesize visible colors by the man)[16]. Each color is then represented by a triplet of real in £ 0; 1¤ (the triplet (1, 1, 1) is the white color and (0, 0, 0) the black color). The quasitotality of the formats of images makes it possible to manage images in base RGB where each component is generally coded on 8 bits (24 bits for a pixel); thus, this system of coding will be used to identify and make an object big data. Each big data object in this architecture will have a *Globally Unique Identifier (GUID)* based on a 512-bit SHA3 key. This means that each object (file, unstructured data, etc.) is associated with a *512-bit SHA key*. For storage, the process is described as follows:

• ***Step 1:*** First, generate a virtual unique point in the CAN space via the CAN hash function. This virtual and unique point will be associated with an object to be stored.

• ***Step 2*****:** The encryption function SHA-512 is used to encrypt the unique virtual points associated with the object whose result is a 512-bit key also associated with the object.

• ***Step 3*****:** The generated 512-bit key is split into eight (08) subkeys, each composed of 64 bits.

Let $\forall x \in O\ ID = (X_1,X_2,X_3,X_4,X_5,X_6,X_7,X_8)$.

• ***Step 4*****:** Each subkey of the previous step is broken down into 24 bits and 40 bits. The first 24 bits define the color (or value) of a virtual image pixel. The remaining 40 bits are split into two parts of 20 bits each, and the X and Y decimal coordinates of the image pixel are represented in Euclidean space.

• ***Step 5*****:** To determine the group of servers in which the object will be stored, a calculation is carried out to find the higher value of *SUP(R,G,B)* associated with each of the subkeys to select the server on which to store the information. For example, if we have *SUPxi(R,V,B) = B,* storage will occur on server *B*. Xi is a subkey. We will have a cluster of servers categorized into R, G, and B. Given that the identifier of an object (the key SHA-512) is split into eight (08) subkeys, we will have eight (08) redundancies in all three clusters of the data server. This set of servers constituted the Data Lake in our architecture.

• ***Step 6***: The object identification information is added to the DHT CAN. A pseudocode of this approach is presented in Algorithm 1. This technique ensures that the data are distributed to all nodes or clusters. Objects sent to the R or G server cluster will be replicated to the data nodes (peer of data). Algorithm 1 is a pseudo code for the procedure described.

Algorithm 1: Pseudo-code

Initialize
Compute
Require: OID,CANDHT,File,Pv ,P,Statut
Ensure: SubKeys, STORE , REPLICATE, Pv,Target Server

```
BEGIN OID ← hascode(x, y)
|    objetKey ← SH A −512(O ID)
|    tmp ← lengthe Of (OID)
|     if  (tmp = 64 octets)
|       |    then P ← 1 for P > 0 and P < 64
|       |   SubKeys ← 8 octets
|       |    Pv ← 5 octects
|       |    TargetServer ← SUP(3octets)
|       |    CANDHT ← (SubKey,FileName)
|       |    BootstrapNode ← STORE(File)
|      | BootstrapSecondaryNode ← REPLICATE(File)
|       |   P ← P +1
|       |   STATUS ← 1
|    else
|      STATUS← 0
|    end if
END
```

Node dynamics

In the RGB architecture, nodes are managed using the CAN protocol. The primary bootstrap node initializes the secondary nodes by assigning them logical addresses. The secondary bootstrap nodes manage the data peers. It is therefore responsible for adding or removing data peers. It also has a routing table for the nodes for which it is responsible and is informed of the status of the primary node. The primary bootstrap node is used to perform storage calculations and maintain the state of the secondary nodes. When a secondary bootstrap node fails, storage is performed on the secondary node with the closest value (according to the *SUP(R,G,B)* calculation) to the failing node. When the primary node fails, the first secondary node becomes the primary bootstrap node until the true primary node is restored. This dynamic architecture makes it a self-managed, high-availability architecture. As the storage mode is key-value oriented, it is possible to parallelize or independently process the execution of search requests on secondary nodes.

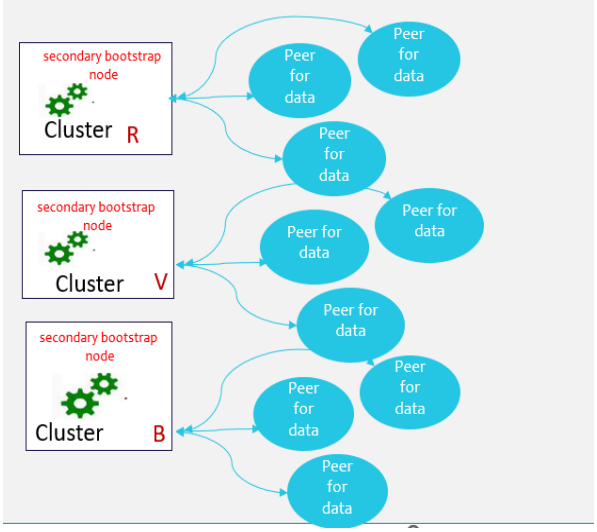

Figure 4: Data nodes in the RGB Architecture

Securing and verifying the integrity of the stored data

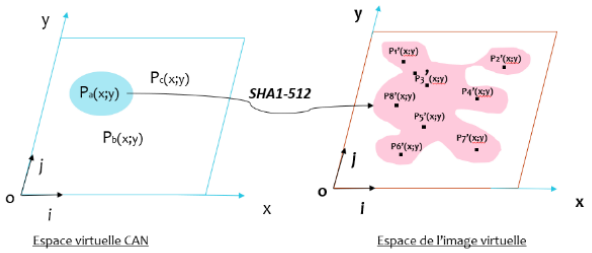

Figure 5: Virtual space of keys

In our model, we have oriented the security at two levels. The use of the SHA3 key applied to the unique identifier generated by the CAN hash function allows for double verification of the object identifier. Indeed, each unique CAN identifier is associated with a set of image points (pixels). Let *Pu* be the unique point of the CAN and SHA3 be the encryption function; the following formula verifies the object identifier:

$$SHA3(Pu) = \sum_{i=1}^{8} (\mathrm{P}'_i)$$

P ' I represents the position of the image points (pixel).

In addition, protecting data through encryption, tokenization and masking are complex and time-consuming processes. This new architectural approach directly integrates data-centric security through the use of secureData Format Preserving Encryption (FPE), which combines both encryption and data masking technology[17]. This is the second level of security. This technique can greatly simplify data confidentiality while mitigating data leakage. This technology allows local encryption of our Data Lake without significant impacts from IT. SecureData protects sensitive data from the moment they are acquired and ensures that theyare always used, transferred and stored in a protected form[17].

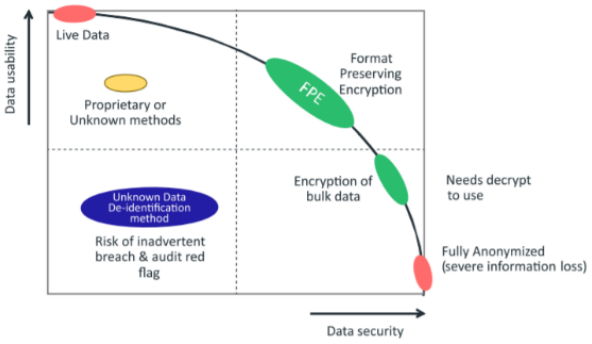

Figure 6: Mode of operation of FPE technology

5 Implementation And Simulation

In this research, we developed the entire module of the proposed architecture based on the CAN protocol[14]. We subsequently integrated this module into a functional CAN P2P protocol version. It is a module programmed entirely in JAVA language using APIs to manipulate files, SHA3 keys, hash tables, and networks. This implementation is accessible and available on GitHub[18]. For the simulation, we configured a local network with a speed of 100 Mbs. The client peers (or sources of data), which are the client programs in our case, are executed on a computer with a capacity of 4 Go Ram and a processor of 2 GHz. A computer with a capacity of 32 GB of RAM (Ram), 2.6 GHz as the processor speed, and an SSD disk was used to deploy the server program (primary bootstrap node). We also configured three virtual computers serving as secondary bootstrap nodes. The simulation was carried out over several hours according to the chosen metrics. In this simulation, we perform several storage operations (LOOKUP) and search operations (STORE) by gradually increasing the number of queries and data sources. The idea is to determine the average time of the storage and search operations and to evaluate the influence of the generation of subkeys on the search for data in large volumes of data. For the calculation of the means, we realize ten [10] repetitive simulations with fixed parameters. To find the mean, we apply the following operation:
Let $x_1,x_2,x_3. . ..,x_{10}$ the means obtained for each simulation and Y the mean to be calculated. $Y = (x_1,x_2,x_3. . ...,x_{10})/10$

6 Results Analysis

Evaluation of the average number of Store requests according to the size of the files

To evaluate the average time of STORE requests, we decided to vary the size of the files. The tool fsutil of Windows allowed us to create files of different sizes. The size of the files varies from 1 Mo to 1024 Mo soit 1 Go. The size of a file represents the size of the flow entering the system, not the size of the data warehouse.
On the graph (Figure 8), we have curves that show the evolution of the average times according to the size of the files. For files from 1 to 32 Mb, the latency of the STORE requests remains almost the same. However, the results show that the storage time increases with the file size. For files ranging from 64 MB to 1024 MB, the runtime increases slightly with the number of STORE requests. Indeed, for each file, 10 000 STORE requests are sent to the server Primary BootStrap Node simultaneously by 10 data sources or peers. These averages were therefore calculated progressively. For 1000 requests, then 2000 requests, then for 3000. This approach allows us to evaluate the latency times according to the number of requests. For a data file of 64 MB and 128 MB, the latency times remain almost constant. There are 10.49 seconds and 31.17 seconds between 1000 and 10000 requests, respectively. The average time for the 256 MB file is 50 seconds, whereas it is 75.23 seconds for a 512 MB file. We have 147.44 seconds as the latency time for a 1024 MB file (1 GB). The curve (Fig. 7) shows that there is a corollary between the latency times and the file sizes. The latency of STORE requests is a function of the file size.

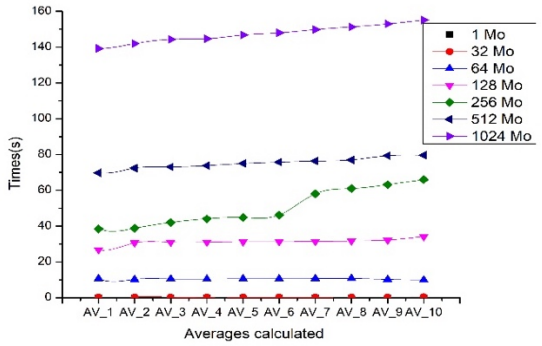

Figure 7: Latency according to the number of requests

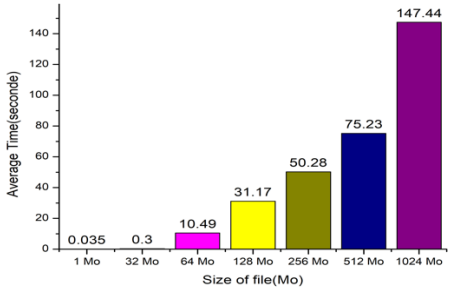

Figure 8: Latency times and file sizes

Evaluation of failure and success rates

As with any computer process, there are often failures in the execution of tasks. In this experiment, we evaluated the failure rate *TE* for 10000 STORE requests sent. Figure 9 shows that the failure rate is low. The failure rate was calculated with the following formula:

$$TE = \frac{Qe}{Qr}$$

Qe is the number of failed requests, and *Qr* is the number of successful requests. In our experiment, failures occurred from 8000 queries, for a rate of 0.06% or 99.94% success. For 9000 requests, the failure rate is 1.9%, or the success rate is 98.1%. For 10000 requests

The success rate was 97.67%, or the failure rate was 2.33%. An analysis of our architecture allowed us to identify the reason for these failures. This is due to the transfer of files. The key generation and identifier assignment processes are successfully executed. However, the possibility that a failure could occur during the first 8000 requests cannot be excluded. Moreover, the failure in our case does not necessarily mean an absence of the file on all the storage clusters because our approach implies eight redundancies, all independent, during storage.

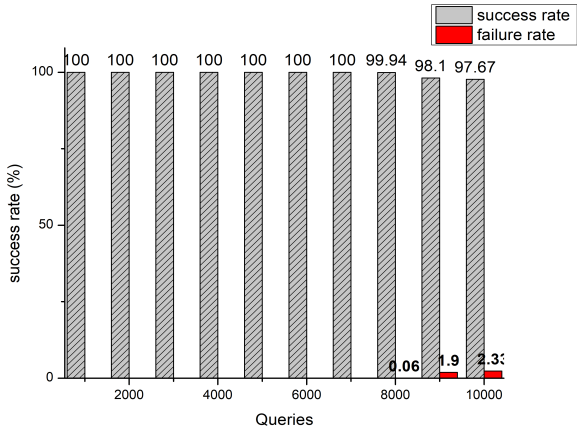

Figure 9: Failure and success rates

Evaluation of the average number of LOOKUP requests

Currently, one of the crucial points in Big Data architecture is latency in the search for information. Even when the functions of MapReduce are applied to the data, their efficiency depends on the time needed to access the data to be treated. One of the objectives of our architecture is to reduce the access time to the data by multiplying the access keys to the data using the DHT CAN. This section shows the evaluation of the results of our architecture. The search operations are called LOOKUP. For this experiment, we first performed 1000 STORE queries. These STORE requests generate 1000*8 keys in the hash table that can identify the 1000 stored objects. Then, we performed 10 LOOKUP experiments. Each LOOKUP experiment (lookup 1, lookup 2, etc.) recorded in the table above corresponds to 1000 LOOKUP requests sent to search for the 1000 objects stored on all three clusters of servers R,V,B. Additionally, for each lookup experiment (from 1 to 10), the data were not initialized. That is, the STORE operation was not resumed. This implies that 10000 LOOKUP requests were sent in total during this experiment. The curve in the figure (FIG. 8) shows the evolution of the average time taken by the LOOKUP requests (T ML). The smallest LOOKUP latency was observed during the first lookup experiment. It is 2.9 ms (6illisecond). The largest value is observed during the third experiment. It is 5.7 ms. The average time decreases progressively to reach the average value of 4.9 ms before starting again at 5.5 ms during the last experiment. The spike observed during the third
phase of the experiment could be explained by memory leakage. The calculated median gives us an average LOOKUP time of 5.5 ms. In the figure (FIG: 9), we have a graphic representation of the success rate and the failure rate. We notice a very low failure rate. The failure rate varies between 0.1% and 0.2%, against a success rate that varies between 99.98% and 100%.

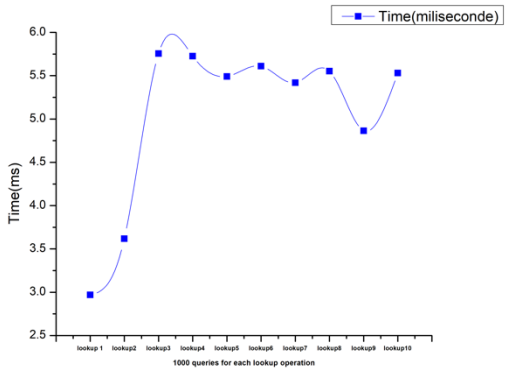

Figure 10: Evolution of the average time taken by the LOOKUP requests

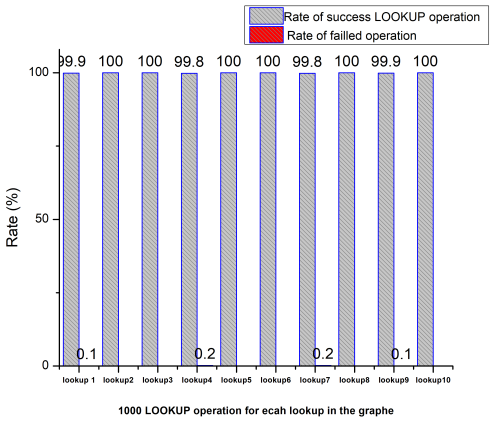

Figure 11: Rate of success of LOOKUP operations

5. MapReduce and the RGB Architecture

Popular platforms for MapReduce, such as Hadoop(19), are extremely powerful but have certain inherent limitations. These platforms are designed to be deployed in a data center. Their architecture relies on several nodes with specific roles to coordinate work, such as the NameNode and JobTracker. These nodes perform scheduling and distribution tasks and contribute to the fault tolerance of the network as a whole; however, in doing so, they themselves become single points of failure. Our MapReduce implementation on the RGB Architecture provides a dynamic framework for MapReduce and is capable of running on any arbitrarily distributed configuration. Our framework exploits the characteristics of CAN[14] distributed hash tables coupled with our color-coded computing approach to manage distributed file storage, fault tolerance and data retrieval.

Our approach to implementing MapReduce has been to develop modules as extensions to the CAN protocol, taking advantage of existing functionality. By treating each task as a data object, we can distribute them in the same way as files, relying on the protocol to route them and ensure their robustness.

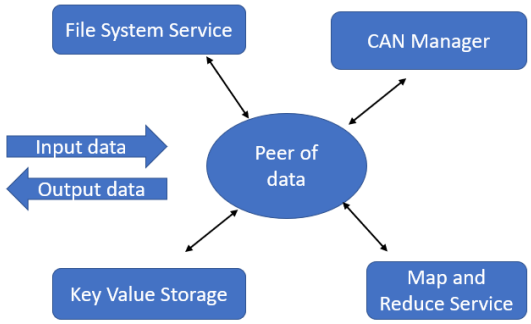

Figure 12: Basic architecture of nodes in the RGB architecture

To evaluate the performance of our MapReduce implementation, we chose to deploy it on a local network. This implementation was entirely realized in Java using the java.net, File, and Stream API and regular expressions. Our implementation implements all the routing and maintenance procedures defined by the CAN protocol, which is used to implement the RGB architecture. The machines used were configured on the Windows file system. Our implementation is therefore able to easily manipulate (create, read and write) files. To start the experiment, MapReduce commands and job descriptions are sent to the primary bootstrap node, which performs a file search operation before the commands are transferred to one of the secondary nodes. We tested our computing system by running a word frequency count. The tasks were tested in several configurations; we varied the initial network size and the size of the jobs. Each map job is defined by the number of nodes that must execute it, and a result that constitutes an input for the "Shuffle" process is produced. Reducing these results involves adding up the respective fields. Our experiment counts the occurrence of each word in a file stored on the RGB architecture.

Results

In the test context, we evaluated the latency of MapReduce requests. We chose a file with a fixed size of 120 MB. This file contains a set of words. The Map and Reduce tasks consist of counting certain keywords that we specified as arguments at the start of the program launch. First, we carried out an initial test to ensure that all the steps would run successfully. To do this, we configured the RGB architecture and the nodes on a single machine with 32 GB RAM capacity and an SSD disk. The addresses of the computing peers and secondary nodes are managed using .txt files. We ran the same job several times, varying the number of nodes from 1 to 10.

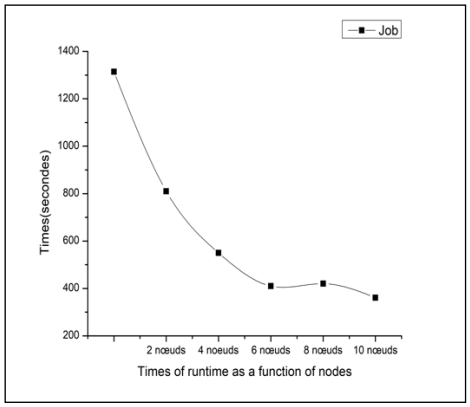

Figure 13: Job execution time as a function of nodes

Figure 10 shows the evolution of the calculation times for the same case. We obtained an average value of 1214 milliseconds, i.e., approximately 1.3 seconds for one node and an average value of 361 milliseconds for 10 nodes. The greater the number of nodes is, the longer the execution time. This implies that the processes of dividing files into blocks, distributing these blocks, counting and sorting are successfully completed.

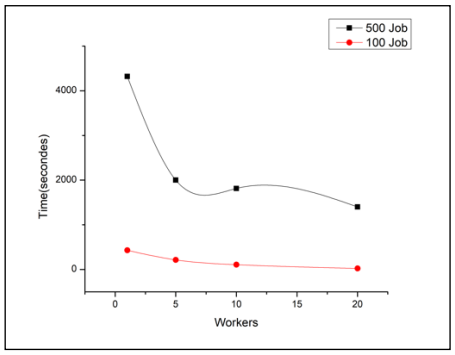

Figure 14: Processing time as a function of the number of computing nodes

Fig. 14 shows the results of the experimenting with MapReduce on the RGB architecture. For the 100 jobs, we have 429.4 seconds for 1 node versus 22.10 seconds for 20 nodes. For 500 jobs, we have 4322 seconds for 1 node versus 1400 seconds for 20 nodes. For this experiment, we observe a progressive decrease in processing time, as shown in Figure 14. We can therefore deduce an acceleration factor by calculating *(T1/Tn)*. This yields 19.41 for 100 jobs and 3.08 for 500 jobs. Note that the greater the number of jobs is, the longer the computation time, but the shorter the computation time if several nodes are assigned to the jobs.

The graphs *(Fig. 15)* show the evolution of computation times for jobs between 1 and 20 compute nodes. This estimate is based on a proportional calculation and the data collected in the previous analyses.

The curves all have the same shape, showing an improvement in calculation time despite the large number of jobs submitted.

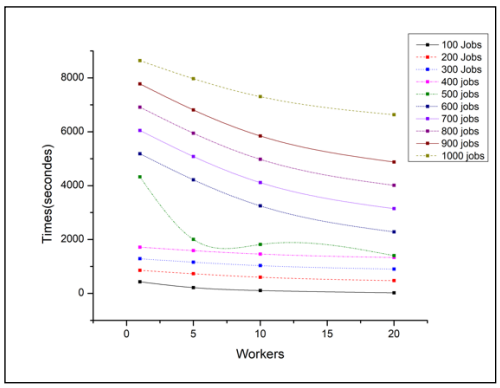

Figure 15: Estimating execution time as a function of nodes

Figure 16 shows a theoretical estimate of the execution speed as a function of the number of nodes. Based on 500 jobs submitted, for 100 nodes, we have an execution speed 66.42 times the execution speed of a node loaded with the same number of jobs. At 10000 nodes, the speed can reach 97.92 times the speed of a loaded node.

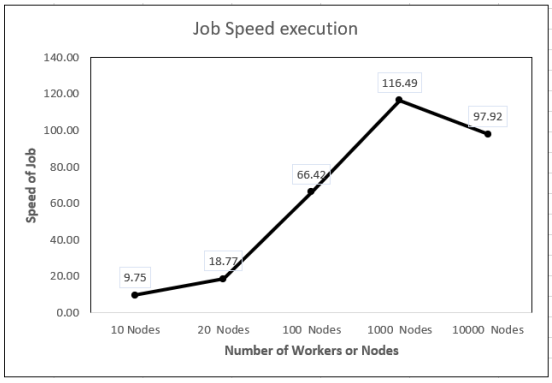

Figure 16: Job execution speed by node

In the following table, we provide a comparison of certain architectures, including the RGB architecture, based on the study of [20].

Features	Lamda	Zeta	kappa	RVB
Data type	Reference data or metadata	transactional data	transactional data	Reference data or metadata
Frequency of Data	Feeds in real-time	Feeds on demand	Feeds on demand	Feeds on demand
Content Format	structured, Semistructured, unstructured	structured, Semistructured, unstructured	structured, Semistructured, unstructured	structured,Semi, unstructured
Data Source	Man made, by computers	web, Internal source	Man made, computers, web	Man made Internal source

Figure 17: Comparison table

7. CONCLUSION

Today, in the world of big data storage, centralized storage is becoming obsolete in favor of decentralized storage. Moreover, decentralized storage with blockchain has been identified as the future of Big Data technology. In this paper, we propose a big data architecture called the RGB architecture based on the P2P CAN network protocol whose storage is an oriented key value. CAN is traditionally a P2P protocol for file distribution and sharing. We implemented a fully functional version of the RGB architecture and performed detailed experiments to test its performance. These experiments have shown that our architecture is robust and dynamic and is able to support the storage of large volumes of data. The use of the CAN protocol as middleware allows us to exploit its efficiency for message sending and data distribution. The efficiency of the CAN protocol helps to improve the data storage processes and to guarantee the scalability of a dataLake. This architectural approach allows us to construct a secure collection and storage system with dynamic processing nodes. Thus, the great advantage of this approach is that it can guarantee low data loss during a collection operation because these secondary nodes can replace the central node without taking too much time. We also implemented a fully functional version of MapReduce on the RGB architecture and carried out detailed experiments to test its performance. These experiments confirmed that the architecture is robust and efficient. P2P network protocols are traditionally known for file sharing. We have demonstrated that this approach can also be used to construct a data pipeline and perform distributed computations on large volumes of data.

Declarations

Ethics approval and consent to participate-Not applicable
Consent for publication-All the authors have given their consent for the publication of the manuscript.
Availability of data and materials-All the data sets on which the conclusions of the manuscript rely are available upon request.
Competing interests-The authors declare no conflicts of interest. Funding This study was not funded.
Author contributions-The authors declare that this study received no contributions from other authors.

Acknowledgments

The authors would like to thank all the teachers of the Training and Research Unit of Joseph Ki-Zerbo University. I would also like to thank the Laboratory of Mathematical and Computer Analysis (LAMI) for allowing me to perform the research.

BIBLIOGRAPHY

1. Number of Internet Users (2016) - Internet Live Stats [Internet]. [cited 2024 Jun 24]. Available from: https://www.internetlivestats.com/internet-users/
2. Labrinidis A, Jagadish HV. Challenges and opportunities with big data. Proc VLDB Endow. 2012 août;5(12):2032–3.
3. PoweredBy - HADOOP2 - Apache Software Foundation [Internet]. [cited 2023 Jun 7]. Available from: https://cwiki.apache.org/confluence/display/HADOOP2/PoweredBy
4. Lämmel R. Google's MapReduce programming model — Revisited. Science of Computer Programming. 2008 Jan;70(1):1–30.
5. Optical storage arrays: A perspective for future big data storage - RMIT University [Internet]. [cited 2024 Jul
6. Available from: https://researchrepository.rmit.edu.au/esploro/outputs/journalArticle/Optical-storage-arrays-A-perspective-for/9921862898101341
7. De Mauro A. What is big data? A consensual definition and a review of key research topics. 2014.
8. Stockage objet : la principale méthode de stockage de données Cloud [Internet]. [cited 2024 Jul 8]. Available from: https://www.lebigdata.fr/stockage-objet-la-principale-methode-de-stockage-de-donnees-cloud
9. Factor M, Meth K, Naor D, Rodeh O, Satran J. Object storage: The future building block for storage systems. In 2005. p. 119–23.
10. Bathla G, Rani R, Aggarwal H. Comparative study of NoSQL databases for big data storage. International Journal of Engineering & Technology. 2018 Mar 11;7(2.6):83–7.
11. Khan: Storage solutions for big data systems: A qualitati... - Google Scholar [Internet]. [cited 2024 Jul 10]. Available from: https://scholar.google.com/scholar_lookup?arxiv_id=1904.11498
12. Saxenna M, Singh V. NoSQL Databases- Analysis, Techniques, and Classification. Journal of Advanced Database Management & Systems (EISSN: 2393-8730). 2014 Jul 1;1:1–11.
13. Flesca S, Greco S, Masciari E, Saccà D. A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years. Vol. 31. 2018.
14. Ratnasamy S, Francis P, Handley M, Shenker S, Karp R. A Scalable Content-Addressable Network
15. Ratnasamy S, Francis P, Handley M, Shenker S, Karp R. A Scalable Content-Addressable Network.
16. KABRE L, Tiendrebeogo T. Comparative Study of can, Pastry, Kademlia and Chord DHTS. International Journal of Peer to Peer Networks. 2021 Aug 31;12:1–22.
17. Fofi D, Mouaddib EM, Salvi J. Décodage d'un motif structurant codé par la couleur.
18. Bellare M, Ristenpart T, Rogaway P, Stegers T. Format-Preserving Encryption. 2009. 295 p.
19. KABREGIT/BigDataStorageBasedOnCan: Data storage based on CAN DHT [Internet]. [cited 2024 Jul 18]. Available from: https://github.com/KABREGIT/BigDataStorageBasedOnCAN
20. Rajeh W. Hadoop Distributed File System Security Challenges and Examination of Unauthorized Access Issue. Journal of Information Security. 2022 Feb 16;13(2):23–42.
21. KoffiKalipe G, Behera R. Big Data Architectures: A Detailed and Application Oriented Analysis. International Journal of Innovative Technology and Exploring Engineering. 2019 Jul 30;8:2182–90.