Sustainable MapReduce: Optimizing Security and Efficiency in Hadoop Clusters with Lightweight Cryptography-based Key Management

Marwa Khadji[1], *Samira* Kholji[1], *Salmane* Bourekkadi[2], *Mohamed* Larbi kerkeb[2]

[1]Abdelmalek Essaadi University, Morocco
[2]Ibn Tofail University, Morocco

Abstract. The exponential growth of big data has led to a significant increase in the volume and complexity of data being generated and stored. This trend has created a huge demand for secure storage and processing of big data. Cryptography is a widely used technique for securing data, but traditional cryptography algorithms are often too resource-intensive for big data applications. To address this issue, light weight cryptography algorithms have been developed that are optimized for low computational overhead and low memory utilization. This research paper explores the use of a new sustainable algorithm that utilizes a lightweight cryptography-based key management scheme to optimize MapReduce security and computational efficiency in Hadoop clusters. The proposed sustainable MapReduce algorithm aims to reduce memory and CPU allocation, thereby significantly reducing the energy consumption of Hadoop clusters. The paper emphasizes the importance of reducing energy consumption and enhancing environmental sustainability in big data processing and highlights the potential benefits of using sustainable lightweight cryptography algorithms in achieving these goals. Through rigorous testing and evaluation, the paper demonstrates the effectiveness of the proposed sustainable MapReduce algorithm in improving the energy efficiency and computational performance of Hadoop clusters, making it a promising solution for sustainable big data processing.

Index Terms— Big Data, Hadoop, Data Security, Sustainable MapReduce, Energy Consumption, Lightweight Cryptography Algorithms, Environmental Sustainability.

1 Introduction

Nowadays, there is no place where Big Data does not exist. In fact, the curiosity about what is Big Data has been soaring in the past few years. The amount of data created and stored globally is growing faster than ever before. Here are some overwhelming facts! According to statistics [1], in 2019, each day internet users generate about 2.5 quintillion bytes of data. And by 2020, every person will generate 1.7 megabytes in just a second, the Big Data analytics market is set to reach $103 billion by 2023.

Big Data refers to the big amounts of data (structured or unstructured) that feeds a company's daily business. However, it isn't the number or the types of data that counts, it's what organizations do. Big Data is used by different type of projects to extract valuable information either to take marketing decisions, track specific behaviors or detect threat attacks. However, Big Data is a double-edged sword. It brings convenience to people and brings certain risks. In the process of data collection, storage, and use, it can easily lead to the leakage of personal information, because data is difficult to discern. Therefore, security should be considered while storing and processing large amount of sensitive data.

One of the most common platforms used to store and process large amount of data is Hadoop. However, when Hadoop was originally designed, the security aspect was not considered [2]. It is fully true that the security system of Hadoop has been improved since it was de-signed. In fact, different projects have started to evolve the security of Hadoop such as Project Rhino. This project provides the ability to encrypt or decrypt the data stored in HDFS. It is important to highlight that Rhino uses AES, which is a good encryption standard. However, it's undoubtedly having higher memory requirement and might degrade performance since client node has limited memory and stored data is voluminous. We are motivated by the fact that traditional cryptographic algorithms rely on the secrecy of encryption algorithms. Such algorithms are now only of historical interest and are not sufficient for real-world needs especially for Big Data context, which requires efficient encryption and decryption algorithms.

2 Hadoop ecosystem and security overview

The Hadoop ecosystem is a comprehensive, open-source platform designed for managing and processing large data sets in a distributed environment. This platform is widely used by organizations to store and analyze massive amounts of data and is known for its scalability and versatility. However, as with any technology, the Hadoop ecosystem presents a unique set of security challenges that must be addressed to protect sensitive data and prevent external threats such as data breaches and malware attacks.

2.1 Hadoop security

When Hadoop was first released in 2007 it was intended to manage large amounts of web data in a trusted environment, it did not have a security mechanism, a security model, or an overall security plan. Effectively, security was not a significant concern or focus. With the increasing use of Hadoop, malicious behaviors such as unauthorized job submission, Job Tracker status change, and data falsification continue to occur. The Hadoop open-source community began to consider security requirements and added security mechanisms such as Kerberos authentication, ACL file access control, and network layer encryption. The Hadoop ecosystem consists of various components. We need to secure all the other Hadoop ecosystem components. In this section, we will look at the each of the ecosystem components security and the security solution for each of these components, each component has its own security challenges, issues and needs to be configured properly based on its architecture to secure them.

2.2 Exploring the Limitations of Hadoop Security and Computational Efficiency Projects

Several studies have proposed various approaches to optimize the security and computational efficiency of Hadoop clusters. Some studies have focused on the use of lightweight cryptography to reduce the computational overhead of cryptographic operations.

Hsiao-Ying Lin et al.[5] have achieved data confidentiality in HDFS by implementing two integrations HDFS-RSA uses AES with RSA and HDFS-Pairing uses a pairing-based encryption scheme and AES.In this paper, their integrations provide alternatives toward achieving data confidentiality for Hadoop by integrating hybrid encryption schemes and the Hadoop distributed file system (HDFS).

In 2012 a hybrid encryption scheme is proposed [6] to ensure data confidentiality in HDFS, which using DES algorithm to encrypt files and RSA for key encryption, and finally IDEA for the user's RSA private key encryption, and finally RSA for key encryption.

One year later Seonyoung Park Youngseok Lee [7] proposed a secure Hadoop architecture by applying encryption and decryption functions to the HDFS. AES encrypt/decrypt classes are added for encryption and decryption of data to (Compression Codec) in Hadoop.

Experiments on Hadoop showed that the representative MapReduce job on encrypted HDFS generates affordable computation overhead less than 7%.

In 2017, Youngho Song et al. suggested a HDFS data encryption scheme which supports both ARIA (the Korean government selected algorithm as a standard data encryption scheme for domestic usages) and AES (international standard data encryption algorithm) algorithms on Hadoop [8].

In 2018, a new approach was proposed in [9] to improve the performance of encryption /Decryption file by using AES and OTP algorithms integrated on Hadoop. In this study, the files are encrypted within the HDFS and decrypted within the Map Task.

In 2021 a Research study in [10]: "Security Challenges and Solutions in Hadoop-based Big Data Analytics" provides a comprehensive review of the security challenges and solutions in Hadoop-based big data analytics. The paper covers various aspects of security in big data analytics, including data privacy and confidentiality, access control, data integrity, and authentication and authorization. The authors discuss the existing security solutions in Hadoop, such as HDFS encryption, Kerberos authentication, and access control mechanisms. They also highlight the limitations of these solutions and propose future research directions to overcome these limitations. This paper provides a valuable resource for researchers and practitioners working on security in Hadoop-based big data analytics.

Finally, Hadoop security projects can also be limited by the available resources and budget allocated for security. Organizations may not have the resources to implement all the necessary security measures, making it difficult to fully secure the Hadoop ecosystem. In addition, the cost of implementing and maintaining security tools and frameworks can be a significant barrier, especially for small and medium-sized organizations.

3 Lightweight cryptography

Different types of cryptographic solutions are available to protect our important data but unfortunately not all of them are suitable for Big Data environments. In fact, standard cryptographic algorithms can be too slow and heavy when encrypting voluminous data. For this, new algorithms such as lightweight cryptography have been proposed to overcome these

problems. Lightweight cryptography (LWC) is a research field that has been developed in recent years. It aims to design schemes for devices with constrained capabilities in power supply, connectivity, hardware and software. LWC is currently used in the Internet security protocols due to its sufficient security. In fact, it is a promising technique for different smart applications that require fewer loads on the CPU, less memory, and higher throughput. Light weighted encryption algorithms are preferred over heavyweight encryption algorithms in low power designs and devices mainly because of their reduced resource requirements. In fact, a light weighted encryption technique takes less time for encryption and provides better security than existing heavyweight algorithms such as AES.

Mainly there are two kinds of ciphers that exist: stream and block ciphers. Stream ciphers allows to encrypt each bit at a time in a stream of input bits while the block ciphers encrypt a block of data rather than just bits.

3.1 Lightweight Block Ciphers

A block cipher is a symmetric cryptographic algorithm that operates on currently on larger pieces of data that is, blocks, frequently joining blocks in order to provide extra security. The concept of a block cipher is to split the file into fairly large blocks, for instance, then to encrypt every block individually. Confusion and Diffusion are two operations used in block cipher for encryption. Confusion makes complex relationship among encryption key and cipher text. There are many lightweight block ciphers such as NOEKEON, SKIPJACK, XTEA and AES.

3.2 Lightweight Stream Ciphers

Lightweight stream ciphers are symmetric cipher in which each character of plaintext is transformed into a symbol of the cipher text.

This process depends not only on the used key, but also on its position in the flow of the plaintext. Stream ciphers use a different approach to symmetric encryption, rather than block ciphers. In a stream cipher, the plaintext is encrypted one bit at a time. In a block cipher, the plaintext is broken into blocks of a set length and the bits in each block are encrypted together.

There are many lightweight stream ciphers such as CHACHA20, Rabbit, HC-128 and AES-CTR.

4 Complexity and Challenges to secure Hadoop using light weight cryptography algorithms.

Integrating light weight cryptography algorithms into Hadoop can present several complexities and challenges:

- **Compatibility:** Light weight cryptography algorithms may not be compatible with Hadoop's existing architecture and infrastructure, requiring significant modifications to integrate them effectively.
- **Performance overhead:** Integrating cryptography algorithms into Hadoop can introduce additional processing overhead, which can negatively impact the overall performance of the system. This can be especially challenging in the context of big data, where high performance is essential.

- **Lack of standardization:** There is a lack of standardization among light weight cryptography algorithms, which can make it difficult to choose the best algorithm for a given use case and to integrate it into Hadoop.
- **Key management:** Cryptography algorithms require secure key management, which can be a challenge in a distributed Hadoop environment. Key management issues can arise, such as key distribution, key storage, and key rotation.
- **Complexity:** Integrating cryptography algorithms into Hadoop can be a complex process that requires expertise in both cryptography and Hadoop.

Despite these challenges, light weight cryptography algorithms have the potential to provide improved security for big data applications in Hadoop. However, it is important to carefully evaluate the trade-offs between security and performance, and to choose the right cryptography algorithm and integration strategy based on specific requirements and constraints.

5 Enhancing MapReduce Security and Performance With A Hybrid Key Management Scheme That Utilizes Lightweight Cryptography Algorithms

To the best of our knowledge, no comprehensive study has been conducted to implement a light wight cryptography algorithm using a hybrid key management scheme in MapReduce and evaluate the performance of lightweight cryptography algorithms in Big Data environments. This lack of research leaves a crucial gap in our understanding of how these algorithms can be effectively utilized in large-scale data processing systems. Thus, in this paper, we study the trade-off between security and performance of some lightweight ciphers. This is where the MapReduce security scheme comes into play. By utilizing the parallel processing capabilities of the MapReduce framework, the security of lightweight cryptography algorithms can be enhanced. This approach allows for the efficient processing of large amounts of data in a secure manner, making it ideal for use in a variety of applications.

We evaluated the performance of our proposed scheme using a Hadoop cluster consisting of eight worker nodes. We compared our scheme with the standard Hadoop security model, which uses the Advanced Encryption Standard (AES) algorithm for encryption. We measured the energy consumption and computational efficiency of both schemes.

This research has shown that the use of the MapReduce security scheme can significantly enhance the security of lightweight cryptography algorithms. In addition, the parallel processing capabilities of MapReduce can also improve the processing time of the algorithms reduces the energy consumption of Hadoop clusters and enhances their environmental sustainability by minimizing the amount of computational resources required for cryptographic operations.

5.1 Encryption Process

Encryption process is done using MapReduce using file stored in the HDFS. The steps used in performing encryption process as shown in Figure 1 are:

- The data are taken from the Hadoop Distributed file system in the form of blocks of fixed sizes.
- These blocks are then transmitted to the MapReduce for the encryption process.

• The Map function contains the code for the encryption where the map function is applied to data and produces intermediate outputs in the form of (key, value) pair. It encrypts the data block by block in parallel and converts them into encrypted chunks.
• These encrypted blocks are transmitted to the Reduce function that's applied on intermediate outputs in the Reducer Phase. It merges all the encrypted blocks in a single encrypted file.
• This single encrypted file of the original file is stored in the HDFS and the process of encryption is completed.

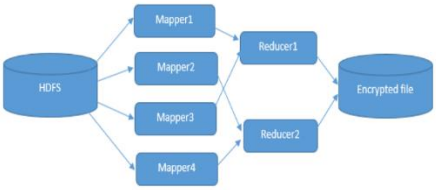

Fig 1: Encryption process

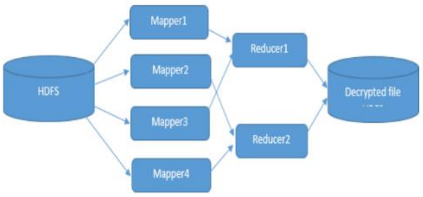

Fig 2: Decryption process

5.2 Decryption Process

Decryption process it is the reverse process of encryption is the step where decrypt the encrypted data. It is done using MapReduce using encrypted file stored in the HDFS. The steps used in performing decryption process as shown in Figure 2 are:
• The input the decryption phases are taken from HDFS in the Encrypted format.
The encrypted data are taken from the Hadoop Distributed file system This encrypted file is broken into Blocks and then transmitted to the Map Reduce Functions Mapper class
• The Map function of the Mapper class contains the decryption code. It decrypts the encrypted blocks one by one in parallel and converts it to plaintext.
• These unencrypted data blocks are then transmitted to the Reducer function of the Reduce class. It merges clear data blocks into a single unencrypted file.
• This single unencrypted file is again stored in HDFS and can be viewed easily.

5.3 Proposed algorithm with a hybrid key management scheme that utilizes lightweight cryptography algorithms

Algorithm 1 Proposed algorithm with a hybrid key management scheme that utilizes lightweight cryptography algorithms

```
Start
Key and value generation:
for each file in files do
    key = generate_key()
    value = generate_value(file)
    keys[file] = key
    values[file] = value
end for
Map phase:
File encryption:
for each file in files do
    encrypted_values[file] = LightWeight_Algorithm_encrypt(values[file], keys[file])
    encrypted_files[file] = encrypted_file
end for
Key and value sharing:
for each party in parties do
    share_key_value(party, keys, values)
end for
for each encrypted_file in encrypted_files do
    key = keys[encrypted_file]
    value = values[encrypted_file]
    map_output = map_function(key, value)
    map_outputs[encrypted_file] = map_output
end for
Shuffle phase:
for each map_output in map_outputs do
    shuffle_output = shuffle_function(map_output)
    shuffle_outputs[map_output] = shuffle_output
end for
Reduce phase:
for each shuffle_output in shuffle_outputs do
    key = keys[shuffle_output]
    decrypted_data = decrypt(shuffle_output, key)
    reduced_output = reduce_function(decrypted_data)
    reduced_outputs[shuffle_output] = reduced_output
end for
Decryption:
final_output = decrypt(reduced_outputs, keys[reduced_outputs])
End
```

Fig 3: Proposed algorithm with a hybrid key management scheme that utilizes lightweight cryptography algorithms

In our proposed algorithm, the key and value are generated for each file in the first step of the algorithm in order to ensure the security and privacy of the data. The key is used to encrypt the file, while the value is used to identify the file.

In our proposed algorithm, the key and value are generated for each file in the first step of the algorithm to ensure the security and privacy of the data. The key is used to encrypt the file, while the value is used to identify the file.

Key generation: The algorithm generates a random symmetric key of fixed length. The key is shared among the MapReduce nodes and used to encrypt and decrypt the data.

Map phase: The input data is divided into fixed-sized blocks and distributed among the MapReduce nodes. Each node encrypts its assigned block using the shared key. The encryption process use a lightweight symmetric encryption algorithm such as Rabbit or AES with a small key size to optimize and enhance encryption time and memory usage.

Shuffle phase: After encryption, the encrypted data blocks are shuffled and redistributed among the MapReduce nodes. This step ensures that the data is evenly distributed and prevents any one node from having access to all the encrypted blocks.

Reduce phase: Each node reduces its assigned blocks by performing a computation or operation on the encrypted data. The reduced data is then re-encrypted using the shared key.

Data merging: Finally, the reduced and re-encrypted data blocks are merged to form the final output. The merged data can be decrypted using the shared key to obtain the original input data.

The MapReduce security scheme enhances the algorithm's security and performance by distributing the data and encryption process among multiple nodes, preventing any single node from having access to the entire input data or encryption key. Additionally, the data shuffling step adds another layer of protection against attacks that might exploit patterns in the input data.

6 Results

6.1 Encryption/Decryption time

In our study we compare our proposed algorithm with two categories of algorithms lightweight block ciphers and lightweight stream ciphers. Stream ciphers are faster than block and are more difficult to implement correctly while block ciphers typically require more memory. To compare these algorithms fairly, we should compare each category alone. The table below present encryption time in seconds of algorithms with varying files sizes from 1 Megabytes to 1000 Megabytes.

Table 1: Encryption time in seconds.

Ciphers \ File size		1Mb	64 Mb	128 Mb	256 Mb	512 Mb	1 Go
Stream Ciphers	**AES(CTR)**	79s	91s	102s	139s	400s	802s
	Chacha20	70s	96s	152s	187s	409s	820s
	RABBIT	80s	89s	97s	117s	322s	612s
	HC128	99s	98s	167s	193s	587s	1020s
Block Ciphers	**AES(CBC)**	51s	200s	239s	386s	1072s	1900s
	NOEKEON	58s	110s	135s	240s	309s	601s
	Skipjack	43s	105s	152s	257s	517s	940s
	XTEA	44s	216s	243s	348s	600s	1023s

The table below present Decryption time in seconds of algorithms with varying files sizes from 1 Megabytes to 1000 Megabytes:

Table 2: Decryption time in seconds.

Ciphers \ File size		1Mb	64 Mb	128 Mb	256 Mb	512 Mb	1Go
Stream Ciphers	**AES(CTR)**	75s	90s	100s	135s	398s	710s
	Chacha20	70s	95s	116s	182s	279s	520s
	RABBIT	72s	86s	98s	122s	200s	398s
	HC128	99s	98s	167s	194s	588s	1020s
Block Ciphers	**AES(CBC)**	57s	207s	246s	395s	1085s	1997s
	NOEKEON	53s	103s	132s	228s	305s	589s
	Skipjack	46s	98s	143s	249s	512s	945s
	XTEA	44s	210s	238s	337s	593s	997s

6.2 Resource Allocation

Memory allocation

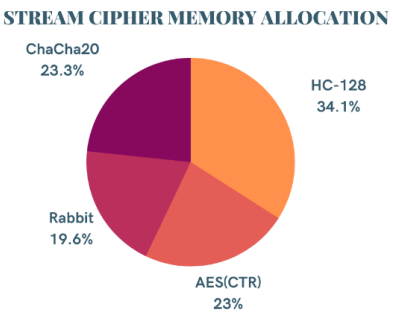

Fig 4: Stream cipher memory allocation

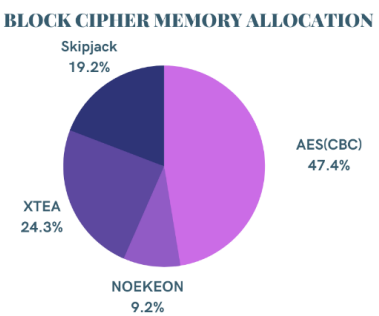

Fig5: Block cipher memory allocation

6.3 CPU allocation

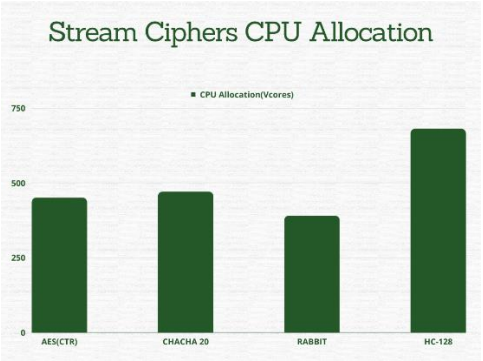

Fig6 : Stream ciphers CPU Allocation

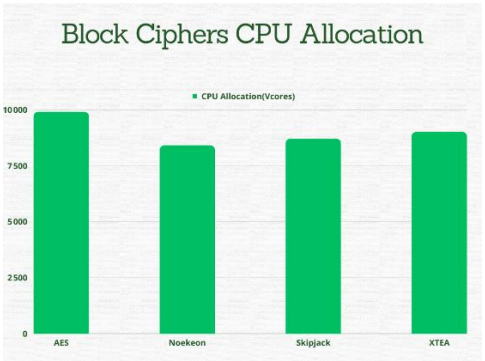

Fig7 : Block ciphers CPU Allocation

7. Discussion

7.1 Encryption and Decryption time

A. Stream ciphers category

The results showed that for small files encryption/decryption (1MB), that Chacha20 consumes least encryption/decryption time, being fast because of its short initialization phase. On the other hand, HC-128 takes the longest time to encrypt and decrypt small data because of the initialization overhead, when small files are processed, the performance is degraded.

Therefore, Chacha20 can be the best candidate used for applications when only small data needs to be processed. For large amounts of data (1Go) the lowest encryption/decryption time was achieved by Rabbit due to the simplicity of its design, it is the most suitable stream cipher to be used in Big Data environment since it has the lowest encryption/decryption time, because it generates a keystream based on a 128-bit key and a 64-bit initialization vector (IV) using simple operations such as bitwise XOR and addition. This enables Rabbit to generate the keystream quickly and with minimal computational overhead.

On the other hand, the highest encryption/decryption time was achieved by HC-128, Because it uses two secret tables, which are essentially arrays of numbers that are used to perform calculations on the data being encrypted or decrypted.

Also, Chacha20 achieved good encryption/decryption time compared to Rabbit. Besides, traditional encryption is not practical to encrypt massive data, although we can see that AES(CTR) refute the theory, it was noted that the AES(CTR) algorithm ranked second for the lowest encryption time after Rabbit.

B. Block ciphers category

The results showed that for small files encryption/decryption (1MB), Skipjack and XTEA achieved the lowest encryption/decryption time, and the highest encryption/decryption time was achieved by NOEKEON.

For large amounts of data (1Go), we can notice that NOEKEON achieved the lowest encryption/decryption time, and the highest encryption/decryption time was achieved by AES (CBC).

Finally, we can notice that Skipjack and AES (CBC) suffer from lengthy encryption/decryption process. Apparently, this is undesirable when handling massive data, when Big Data paradigm demands for faster and efficient encryption process.

7.2 Ressource Allocation

7.2.1 Memory Allocation

A. The memory used in each lightweight stream cipher is shown in percentages Fig4.

Referring the result from the graphs fig 4, it is shown that HC-128 algorithm has large memory requirement as compared to other stream ciphers, because HC-128 uses two secret tables, this algorithm needs to access and perform calculations on these tables, which can impact the amount of memory needed to execute the algorithm, and ultimately affect the performance of the encryption and decryption process. Therefore, this memory used negatively impacts cost of the system.

In other side, the results show that Rabbit takes the lowest memory for encryption due to its low encryption time, Because Rabbit generates a keystream based on a 128-bit key and a 64-bit initialization vector (IV) using relatively simple operations such as bitwise XOR and addition. It does not rely on large lookup tables or complex mathematical operations like some other stream ciphers. This design allows Rabbit to use less memory for its implementation compared to other ciphers, particularly block ciphers which require larger amounts of memory.

Additionally, Rabbit is known for its high speed and low latency, which makes it a popular choice for applications that require fast and efficient encryption. Its simple design and efficient implementation enable it to encrypt and decrypt data quickly and with minimal memory usage.

After analyzing the figure, it can be concluded that Rabbit, Chacha20, and AES (CTR) are the most suitable ciphers in terms of memory usage. These ciphers require smaller amounts of memory engagement, making them favorable for Big Data applications where efficient memory usage is critical for performance. However, the specific choice of cipher may depend on other factors such as encryption speed, security, and compatibility with the application's hardware and software environment.

Moreover, using HC-128 to encrypt the Big Data will consume the computing resources and decrease the speed making them unsuitable to be utilized in Big Data environments.

B. The memory used in each lightweight block cipher is shown in percentages Fig5.

For block ciphers, AES (CBC) occupy the highest memory space when encrypting a file within 1 Go. Also, we can observe that NOEKEON requires lesser overall storage as compared to the AES (CBC) cipher.

Based on this figure, the more suitable ciphers in terms of memory usage are (NOEKEON, XTEA and Skipjack where smaller amount of memory engagement will be favorable for Big Data applications.

Furthermore, utilizing AES (CBC) to encrypt Big Data will consume the computing resources and decrease the speed making them unsuitable to be utilized in Big Data environments.

7.2.2 CPU Allocation

A. CPU allocated to each lightweight stream cipher is shown in percentages Fig6.

The given graph provides a comparison of CPU usage in terms of Vcores and time for four different stream ciphers: Rabbit, AES (CTR), Chacha20, and HC-128. From the graph, it is evident that Rabbit cipher has the least number of Vcores allocated (388) and the shortest time taken (322 seconds) to complete the task, while HC-128 has the highest CPU usage with 690 Vcores .

The reason for Rabbit cipher's low Vcores allocation can be attributed to the algorithm structure on which it is written. The Rabbit cipher uses a simple and efficient algorithm that makes it faster and more efficient compared to other stream ciphers. Therefore, in scenarios where resources such as CPU power are limited, the Rabbit cipher can be a suitable option.

In contrast, HC-128 has the highest CPU usage, indicating that it requires more resources to complete the task. The algorithm used by HC-128 is more complex than Rabbit, which makes it slower and more resource-intensive. Therefore, it may not be a suitable option in scenarios where resources are limited.

Regarding AES (CTR) and Chacha20, their CPU usage is almost the same with AES (CTR) using 460 Vcores and Chacha20 using 480 Vcores. These two stream ciphers have similar efficiency and can be good options for scenarios that require stream ciphers with moderate resource requirements.

B. CPU allocated to each lightweight block cipher is shown in percentages Fig7.

The given graph provides a comparison of CPU usage in terms of Vcores for four different block ciphers: NOEKEON, AES (CBC), Skipjack, and XTEA. From the graph, it is evident that NOEKEON has the lowest number of Vcores allocated (8400), while AES (CBC) has the highest CPU usage with 9900 Vcores.

NOEKEON is a block cipher that uses a simple and efficient algorithm, which makes it faster and more efficient compared to other block ciphers. Therefore, in scenarios where resources such as CPU power are limited, NOEKEON can be a suitable option.

On the other hand, AES (CBC) has the highest CPU usage compared to all other block ciphers in the study. The algorithm used by AES (CBC) is more complex than NOEKEON,

which makes it slower and more resource-intensive. Therefore, it may not be a suitable option in scenarios where resources are limited.

Regarding Skipjack and XTEA, their CPU usage is lower compared to AES (CBC), but still higher than NOEKEON. Skipjack requires 8700 Vcores, and XTEA requires 9000 Vcores. These block ciphers have a moderate level of efficiency and can be good options for scenarios that require block ciphers with moderate resource requirements.

7.3 Efficient MapReduce Encryption and Decryption: Impact on Energy and Environment

Efficient MapReduce Encryption and Decryption can have a significant impact on energy and the environment.The encryption and decryption of data in MapReduce can consume a significant amount of energy and processing power. This can have a negative impact on the environment, as the increased energy consumption can result in higher carbon emissions and contribute to global warming.

However, if the encryption and decryption process is made more efficient, it can significantly reduce the amount of energy required to process large amounts of data. This can lead to a reduction in carbon emissions and a positive impact on the environment. Here are some ways in which optimizing encryption and decryption time in MapReduce, and reducing memory and CPU allocation, can impact the use of energy and the environment:

- **Reduced computational load:** Optimizing encryption and decryption time in MapReduce reduces the computational load on the cluster. This means that the cluster can perform the same amount of work in less time, which translates to reduced energy consumption.[11] By reducing the computational load, the cluster can also run more efficiently, which can further reduce energy usage.
- **Efficient resource utilization:** When memory and CPU allocation are reduced, MapReduce clusters can use their resources more efficiently.[12] This means that the cluster can complete its work with fewer resources, which reduces the overall energy consumption of the cluster.[13]
- **Reduced cooling requirements :** MapReduce clusters generate a lot of heat, which requires cooling systems to keep the machines within safe operating temperatures. When the cluster is running more efficiently, it generates less heat, which reduces the cooling requirements.[14] This, in turn, reduces the energy consumption of the cooling systems.
- **Reduced carbon footprint:** The reduced energy consumption of an optimized MapReduce cluster translates to a reduced carbon footprint. This is because less energy is consumed from non-renewable sources, which reduces the greenhouse gas emissions associated with energy production.[15]
- **Longer lifespan of hardware:** By reducing the computational load, memory usage, and CPU allocation of MapReduce clusters, the lifespan of the hardware used in the cluster can be extended. This means that the hardware can be used for a longer period of time, which reduces the need for frequent upgrades and replacements.[15] This, in turn, reduces the environmental impact associated with the production and disposal of hardware.

8 CONCLUSION

Through the analysis and comparison of experimental data results, it became evident that each cryptographic algorithm has its own strengths and weaknesses. Therefore, the selection of a cryptographic algorithm should be based on the specific demands of the application it will be used for.

Based on the experimental results and comparison, Rabbit stream cipher and NOEKEON block cipher are suitable choices in terms of CPU and memory allocation. If confidentiality and integrity are major factors, AES and Chacha20 algorithms can be selected. Additionally, if high speed is a significant requirement, Rabbit stream cipher and NOEKEON block cipher are the best options.

Overall, the choice of cipher depends on the specific requirements of the application, such as the level of security, the desired encryption/decryption speed,the available hardware resources and energy consumption . By carefully considering these factors, it is possible to select the most appropriate cryptographic algorithm for the application, which can improve both the security , performance and have suitable environment impact on the the system.

In conclusion, optimizing encryption and decryption time in MapReduce, and reducing memory and CPU allocation, can have a significant impact on the use of energy and the environment. By reducing the computational load, using resources more efficiently, reducing cooling requirements, reducing the carbon footprint, and extending the lifespan of hardware, an optimized MapReduce cluster can have a positive impact on the environment.

Futur work

Our futur work can be conducted on how to better manage the energy consumption and cooling requirements of MapReduce clusters. This could involve exploring new methods of resource allocation, load balancing, and temperature control, to ensure that energy is used more efficiently and that cooling requirements are minimized.

We could also focus on developing new tools and frameworks for monitoring and managing the energy usage and environmental impact of MapReduce clusters. This could include developing new monitoring tools that can track energy usage and carbon emissions, as well as developing new algorithms and techniques for optimizing energy usage and reducing environmental impact. By continuing to explore and develop new solutions in this area, we can further improve the efficiency and sustainability of MapReduce clusters, and reduce their overall impact on the environment.

References

1. Marr, Bernard. "How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read." Forbes, 21 May 2018.
2. Sharma, P.P. "Securing Big Data Hadoop: A Review of Security Issues, Threats and Solution."
3. Kadre, Viplove and Chaturvedi, Sushil. "AES-MR: A Novel Encryption Scheme for Securing Data in HDFS Environment Using MapReduce." International Journal of Computer Applications, Vol. 129, 2015, pp. 12-19.

4. Yang, Chao, Lin, Weiwei and Liu, Mingqi. "A Novel Triple Encryption Scheme for Hadoop-Based Cloud Data Security." Proceedings of the 2013 4th International Conference on Emerging Intelligent Data and Web Technologies (EIDWT), 2013, pp. 437-442.
5. Park, S. and Lee, Y. "Secure Hadoop with Encrypted HDFS." Proceedings of the 2013 IEEE 10th International Conference on e-Business Engineering (ICEBE), 2013, pp. 134-141.
6. Jayan, Anandu and Upadhyay, Bhargavi. "RC4 in Hadoop Security Using MapReduce." Proceedings of the 2017 2nd International Conference on Communication and Information Systems (ICCIDS), 2017, pp. 1-5.
7. Mahmoud, Hadeer, Hegazy, Abdelfatah and Khafagy, Mohamed. "An Approach for Big Data Security Based on Hadoop Distributed File System." Proceedings of the 2018 9th International Conference on Information Technology Convergence and Services (ITCS), 2018, pp. 109-114.
8. Lin, Hsiao-Ying, Shen, Shiuan-Tzuo, Tzeng, Wen-Guey and Lin, Bao-Shuh. "Toward Data Confidentiality via Integrating Hybrid Encryption Schemes and Hadoop Distributed File System." Proceedings of the 2012 IEEE 26th International Conference on Advanced Information Networking and Applications (AINA), 2012, pp. 740-747.
9. El imrani, O. et al (2022). The consumer price index and it effect in the new ecosystems and energy consumption during the sanitary confinement: The case of an emerging country. IOP Conference Series: Earth and Environmental Science , 975(1), 012006
10. Kassou, M., Bourekkadi, et al. (2021) . Blockchain-based medical and water waste management conception. E3S Web of Conferences, 2021, 234, 00070
11. Parmar, Raj, Roy, Sudipta, Bhattacharaya, Debnath, Bandyopadhyay, Samir and Kim, Tai-hoon. "Large Scale Encryption in Hadoop Environment: Challenges and Solutions." IEEE Access, Vol. 5, 2017, pp. 28945-28953.
12. Ali, Syed Raza and Javaid, Nadeem. "A novel approach for secure big data communication using hybrid encryption and MapReduce." International Journal of Distributed Sensor Networks, Vol. 17, No. 1, 2021, DOI: 10.1177/1550147721991358.
13. Chen, J., Liu, Q., Liu, Y., & Yang, L. T. (2017). Energy-efficient MapReduce encryption and decryption scheme for big data. Journal of Network and Computer Applications, 87, 88-97.
14. Shafi, S., Parvez, S., & Razaque, A. (2019). Efficient energy consumption in MapReduce through workload balancing and power-awareness. Sustainable Computing: Informatics and Systems, 21, 1-12.
15. Zhao, Y., Li, J., Zhang, X., & Li, Y. (2015). Energy-efficient MapReduce scheduling for big data applications in cloud. Journal of Parallel and Distributed Computing, 80, 14-26.
16. Ma, H., Huang, Y., Xu, Y., & Liu, W. (2016). Energy-efficient data processing in MapReduce for green cloud computing. Journal of Grid Computing, 14(2), 223-236.
17. Li, H., Li, Y., Wang, L., & Zhang, X. (2016). Energy-efficient data encryption scheme for MapReduce in cloud. Future Generation Computer Systems, 65, 65-72.