CLOAK-REDUCE LOAD BALANCING STRATEGY FOR MAPREDUCE

Mamadou Diarra and Telesphore Tiendrebeogo

Department of Mathematics and Computer Science, Nazi Boni University, Bobo-Dioulasso, Burkina Faso

ABSTRACT

The advent of Big Data has seen the emergence of new processing and storage challenges. These challenges are often solved by distributed processing.

Distributed systems are inherently dynamic and unstable, so it is realistic to expect that some resources will fail during use. Load balancing and task scheduling is an important step in determining the performance of parallel applications. Hence the need to design load balancing algorithms adapted to grid computing.

In this paper, we propose a dynamic and hierarchical load balancing strategy at two levels: Intra-scheduler load balancing, in order to avoid the use of the large-scale communication network, and inter-scheduler load balancing, for a load regulation of our whole system. The strategy allows improving the average response time of CLOAK-Reduce application tasks with minimal communication.

We first focus on the three performance indicators, namely response time, process latency and running time of MapReduce tasks.

KEYWORDS

Big Data, Distributed processing, Load balancing, CLOAK-Reduce, Task allocation

1. INTRODUCTION

Information's increasing due to connected objects has seen the emergence of new technology domains offering an alternative to traditional databases. Distributed systems offer solutions for high scalability of the information system through service-oriented architectures.

Big Data processing can be considered in two main phases: storing the data on a Distributed File System (DFS) and running distributed computations on the data. Different approaches to distributed computing have been proposed for Big Data processing, the most widely used being MapReduce [1].

However, the implementation of distributed systems generates difficulties such as managing node heterogeneity, task governance, degraded mode operation and load balancing.

However, load balancing, which consists in allocating data initially and then eventually redistributing it over a set of processors in order to minimise their processing time, remains a real problem for distributed systems.

Our objective is to propose a load balancing and task allocation strategy for our distributed CLOAK-Reduce model inspired by the CLOAK DHT [2], [3], [4] and MapReduce [1], [6], [7]. We will then evaluate the performance of this strategy simulations.

Our study focuses on the following aspects:

1. Our first contribution is the description of the load balancing strategy of CLOAK-Reduce.
2. Our second contribution is to study the quality of the results produced by the analysis of the proposed load balancing strategy.

Our paper will be organized as follows:

1. A first step, we discuss the relative work;
2. A second step, we describe our model of the distributed processing platform;
3. The third part, we analyse the results of the different simulations.

2. Related Works

2.1. Distributed Systems

A distributed system is a heterogeneous set of independent computers that appears to its users as a single coherent system [8].

A distributed system (Figure 1) is an arbitrary set of computing units (Nodes) with its own address space, linked in a network so that they can exchange messages and coordinate their activities. A distributed system is capable of handling any number of processes.

Communication between the different nodes is done by messages. It is important, to consider the communication delay between the different nodes [9].

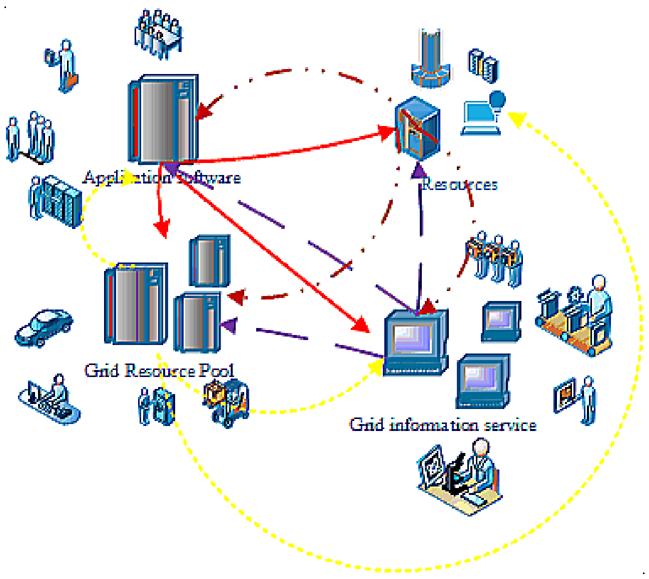


Figure 1. Distributed systems computing environment

2.2. Grid Computing

Grid computing is a virtual infrastructure made up of a set of potentially shared, heterogeneous, distributed, autonomous and delocalised computing resources [10].

Grid computing is an autonomous, dynamically reconfigurable, scalable computing infrastructure aggregating a large number of computing and data storage resources. It is intended to make resources available to users of distributed computing.

More concretely, a computing grid is made up of a large number of heterogeneous and often delocalized machines linked by an Internet network and made homogeneous to users by middleware [11].

2.3. Distributed Hash Tables

DHT is a technology for constructing a hash table in a distributed system where each piece of data is associated with a key and is distributed over the network. DHTs provide a consistent hash function and efficient algorithms for storing STORE (key, value) and locating LOOKUP (key) of the node responsible for a given (key, value) pair.

It is therefore important to specify that only a reference of the declared object (Object IDentifier (OID)) is stored in the DHT. Concerning the search for an object, only by routing the searched key can the set of associated values be found [2], [3], [5].

In practice, DHTs are substituted by "overlay" networks on top of the physical networks, thus reducing the size of the table at each node while considerably increasing the efficiency of the search algorithm.

DHTs have been widely studied because of their attractive properties: efficiency and simplicity with Chord [12], controlled data placement with SkipNet [13], Pastry [14] routing and localization, and better consistency and reliable performance with Kademlia [15].

CLOAK DHT, the basis of our work, is based on a hyperbolic tree constructed in hyperbolic space (hyperboloid) and stereographically projected into a Poincaré disk (Euclidean plane) of radius 1 centred at the origin where the tree node uses a virtual coordinate system.

2.4. Load balancing

A node load is driven by the activity of the processes running locally. Load balancing is the process of distributing network traffic across multiple servers. This ensures no single server bears too much demand. By spreading the work evenly, load balancing improves application responsiveness. It also increases the availability of applications and websites for users [16], [17].

The main goals of load balancing, in a distributed environment, are improving system performance, maintaining stability in the system, building a system which is tolerant to faults and to be able to make modifications in the future.

However, the load balancing problem has two objectives, to consider [18], [19], [20]: minimising user response time and reducing load imbalance between servers.

There are actually two types of load balancing algorithms:

- Dynamic load balancing [21]: In a dynamic algorithm, the least loaded server in the whole system is looking for and preference is given to it to load balancing. Dynamic load balancing offers better performance;

Static load balancing [22], [23]: This algorithm needs to have prior information about the system resources in order to be able to make sure that decision of load shifting does not depend on the current system state. Static algorithm is suitable for systems with low load variation.

In this paper, we mainly interested in dynamic load balancing which offers us three architectures: centralized, decentralized and hierarchical load balancing algorithms [24].

A dynamic process allocation algorithm consists of two basic elements: an information element and a control element (Figure 2). The role of the information element is to maintain information about the state of the distributed system, which is used by the control element to perform the actual placement of processes [25].

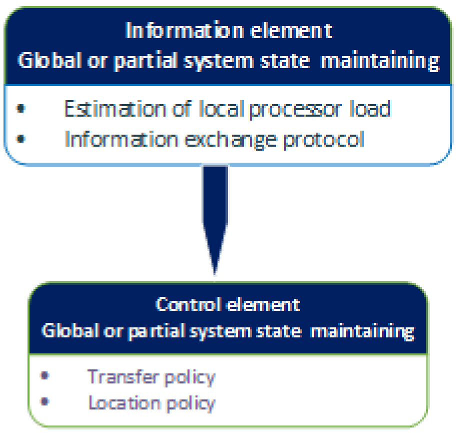

Information element
Global or partial system state maintaining
Estimation of local processor load
Information exchange protocol
Control element
Global or partial system state maintaining
Transfer policy
Location policy

Figure 2. Structure of a dynamic allocation algorithm

2.5. Data Replication

The objective of data replication is to increase data availability, reduce data access costs and provide greater fault tolerance [26].

By replicating datasets to an off-site location, organisations can more easily recover data after catastrophic events, such as outages, natural disasters, human errors or cyberattacks.

3. Our Contributions

3.1. Cloak-Reduce

CLOAK-Reduce is built by implementing, on the CLOAK DHT, a module to create and execute MapReduce operations. It is a distributed model that exploits the advantages of CLOAK DHT and MapReduce. CLOAK DHT to submit Map () and Reduce () jobs in a balanced way thanks to the replication mechanisms it offers in addition to the task scheduling strategy we provide (Figure 4)

The tree structure of the CLOAK DHT allowed us to define a hierarchical architecture of load balancing at two levels: Intra-scheduler for local load balancing and Inter-schedulers for global load balancing (Figure 3).

Our model consists of a root node, three schedulers, several candidate nodes or JobManagers candidates and builder nodes or JobBuilders.

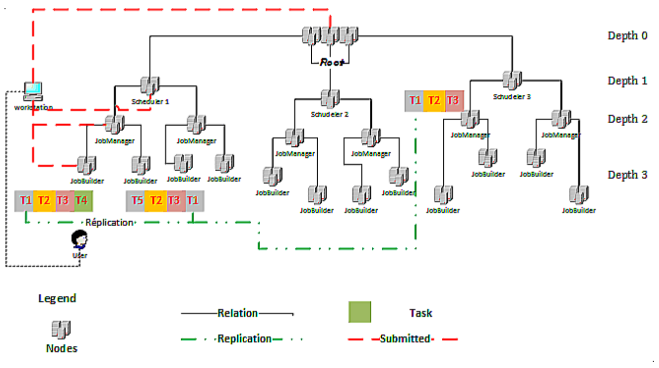


Figure 3. CLOAK-Reduce Architecture

3.1.1. Summary of the Invention

Root node: The root is a node of coordinate (0,0), the minimal depth (D_0) tree, of the Poincaré disk. It allows to:

- Keep tasks that cannot be executed immediately in the root task spool;
- Maintain schedulers load information;
- Decide to submit a task to a scheduler.

Schedulers: The schedulers of depth (D_1), have the function of:

- Maintain the load information of all their JobManagers;
- Decide global balancing of their JobManagers;
- Synchronise their load information with their replicas to manage their failure and the others schedulers for load balancing.

JobManagers: JobManagers candidates have a minimum depth ($\geq D_2$). They are elected for:

- Supervise MapReduce jobs running;
- Manage the load information related to JobBuilders;
- Maintain their load state;
- Decide local load balacing with their circulars replicas JobManager candidate of the same scheduler;
- Inform JobBuilders of the load balancing decided for the backup of intermediate jobs.

JobBuilders: The JobBuilders are of minimum depth ($\geq D_3$). Their function is to:

- Execute MapReduce jobs;
- Synchronize the images of their different works as they go on their radial replicas
- of the depth > D_3.
- Keep the information on the state of their charge up to date;

- Update this workload information at the JobManager;
- Perform the balancing necessities ordered by their JobManagers.

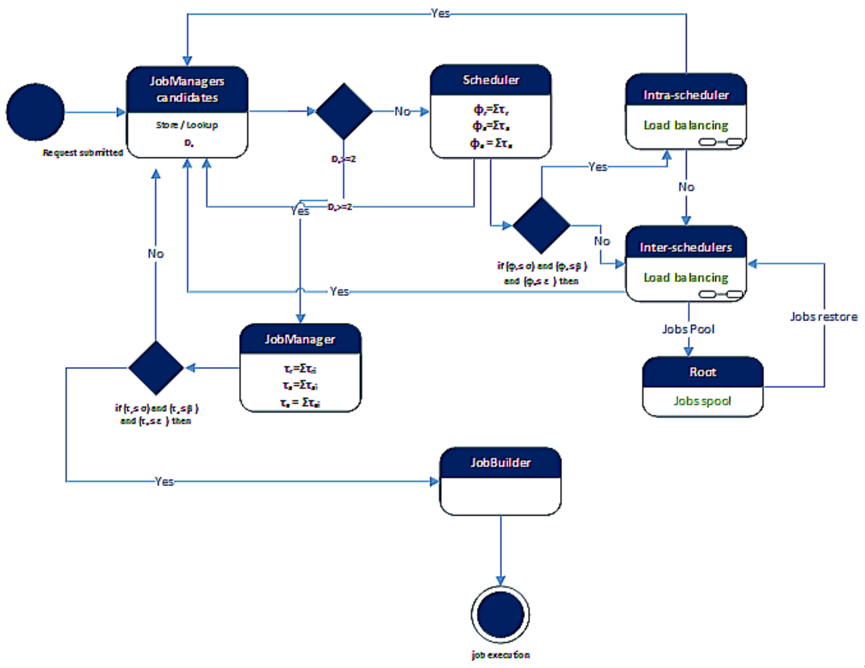


Figure 4. CLOAK-Reduce function diagram

3.1.2. Load Balancing Strategy

Our work consists of load balancing at the JobManager level in a decentralised way, since the JobManagers are independent of each other, which saves response time on the one hand, and allows the selection of the circular replica that best matches the received task on the other.

The Scheduler allows local overloaded and underloaded JobManagers to be brought into contact with each other so that they can work together to migrate tasks from one JobManager to the other.

3.1.2.1. Brief Description

- Each JobManager has a period, during which it sends its load information to the JobManagers candidates of the same scheduler.
- Each JobManager has a period, during which it sends its load information to its scheduler.
- Each scheduler has a period, during which it sends its load information to the other schedulers.
- Each scheduler has a period, during which it sends its load information to the root.
- Intra-scheduler load balancing is triggered by schedulers or their JobManagers.
- Intra-scheduler load balancing of JobManager is performed between the JobManagers and its JobManagers candidates to avoid the use of any scheduler communication network.
- Scheduler's intra-scheduler load balancing favours, where possible, load balancing between JobManagers to avoid the use of any Scheduler communication network.
- The overloaded scheduler redirects the task to be submitted to the root spool.
- Inter-scheduler load balancing is triggered by the overloaded schedulers in case intra-scheduler load balancing is not successful.

- Inter-scheduler load balancing favours load balancing between schedulers.
- The root submits pending tasks to schedulers based on a load level.

3.1.2.2. Intra-Scheduler Load Balancing

Figure 5, Every JobManager can trigger a load balancing operation based on the current load it is managing. This load is estimated from the various load information sent periodically by the JobBuilders. The information received by JobManager allows it to distribute its load to its candidate JobManagers. When the JobManager load balancing operation fails, the scheduler takes over.

The scheduler tries, as a priority, to balance the load of the JobManager locally by distributing it among the JobManagers that are under its control. This locality approach aims to reduce communication costs, by avoiding inter-scheduler communications.

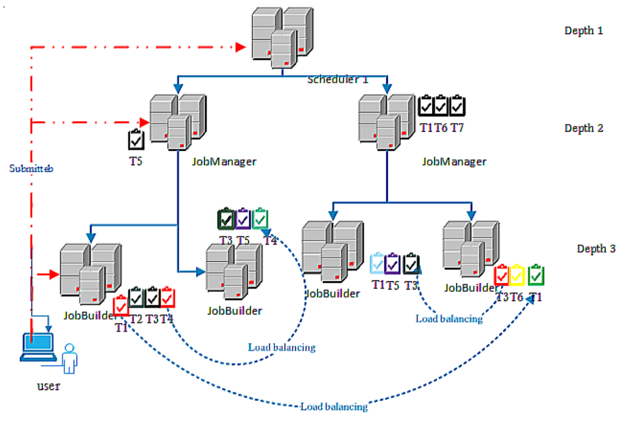


Figure 5. Intra-scheduler load balancing

3.1.2.3. Inter-Scheduler Load Balancing

Figure 6, Inter-scheduler load balancing is triggered only when intra-scheduler balancing fails due to either JobManager saturation or insufficient load supply from candidate JobManagers compared to the demand from overloaded JobManagers. In this case, the overloaded scheduler transfers the new submissions to the root.

Also, the root, according to the scheduler information, can ask the overloaded scheduler to perform an inter-scheduler load balancing, considering the allocation costs and the choice of the tasks to be selected.

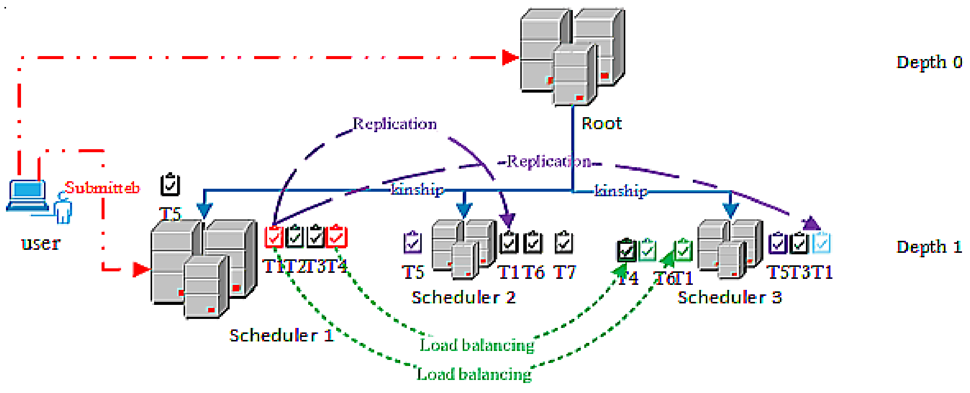


Figure 6. Inter-schedulers load balancing

3.1.3. Brief Description of Some Load Balancing Algorithms

We focus on the mean response time, the mean process latency, and the mean running time of a task as a JobManager load.
Notations:

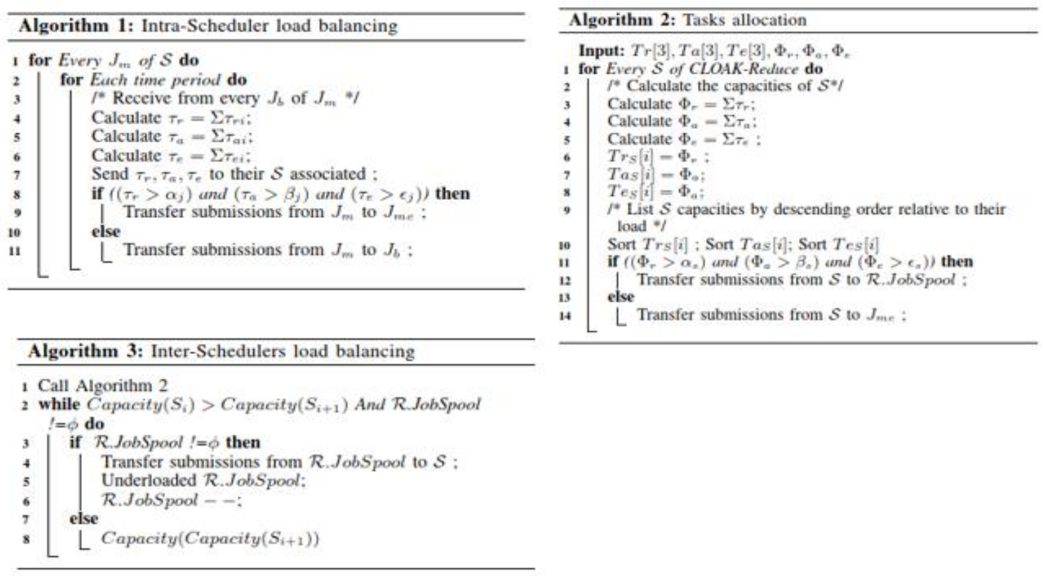
- Break-even point for JobManagers:

  ✓ $\alpha_{j:}$ Response time,
  ✓ β_j: Process latency,
  ✓ ϱ_j: Running time for jobs.

- Break-even point for Scedulers:

  ✓ α_s: Response time,
  ✓ β_s: Process latency,
  ✓ ϱ_s: Running time for jobs.

- Nodes:
- J_b: JobBuilder,
- J_m : JobManager,
- J_{mc}: JobManager candidate,
- S: Scheduler,
- R: Root
- Variables:
- τ_r : Jb response time ,
- τ_a : Jb process latency ,
- τ_e : Jb running time,
- Φ_r : Jm response time,
- Φ_a :Jm process latency,
- Φ_e : Jm running time



Algorithm 1: Intra-Scheduler load balancing

```
for Every $J_m$ of $\mathcal{S}$ do
    for Each time period do
        /* Receive from every $J_b$ of $J_m$ */
        Calculate $\tau_r = \Sigma\tau_{ri}$;
        Calculate $\tau_a = \Sigma\tau_{ai}$;
        Calculate $\tau_e = \Sigma\tau_{ei}$;
        Send $\tau_r, \tau_a, \tau_e$ to their $\mathcal{S}$ associated ;
        if $((\tau_r > \alpha_j)$ and $(\tau_a > \beta_j)$ and $(\tau_e > \epsilon_j))$ then
            Transfer submissions from $J_m$ to $J_{me}$ ;
        else
            Transfer submissions from $J_m$ to $J_b$ ;
```

Algorithm 2: Tasks allocation

```
Input: $Tr[3], Ta[3], Te[3], \Phi_r, \Phi_a, \Phi_e$
for Every $\mathcal{S}$ of CLOAK-Reduce do
    /* Calculate the capacities of $\mathcal{S}$*/
    Calculate $\Phi_r = \Sigma\tau_r$;
    Calculate $\Phi_a = \Sigma\tau_a$;
    Calculate $\Phi_e = \Sigma\tau_e$ ;
    $Trs[i] = \Phi_r$ ;
    $Tas[i] = \Phi_a$;
    $Tes[i] = \Phi_a$;
    /* List $\mathcal{S}$ capacities by descending order relative to their
      load */
    Sort $Trs[i]$ ; Sort $Tas[i]$; Sort $Tes[i]$
    if $((\Phi_r > \alpha_s)$ and $(\Phi_a > \beta_s)$ and $(\Phi_e > \epsilon_s))$ then
        Transfer submissions from $\mathcal{S}$ to $\mathcal{R}.JobSpool$ ;
    else
        Transfer submissions from $\mathcal{S}$ to $J_{me}$ ;
```

Algorithm 3: Inter-Schedulers load balancing

```
Call Algorithm 2
while $Capacity(S_i) > Capacity(S_{i+1})$ And $\mathcal{R}.JobSpool$
  $!=\phi$ do
    if $\mathcal{R}.JobSpool$ $!=\phi$ then
        Transfer submissions from $\mathcal{R}.JobSpool$ to $\mathcal{S}$ ;
        Underloaded $\mathcal{R}.JobSpool$;
        $\mathcal{R}.JobSpool--$;
    else
        $Capacity(Capacity(S_{i+1}))$
```

Figure 7. Some Algorithms

3.2. Performance Analysis

3.2.1. Experimental setup

For data collection, the circular and radial replication mechanisms of CLOAK-Reduce have been set to five (05), in a dynamic network with 10% churns, have been required.

In order to study the scalability of the system in addition to load balancing, we opted to consider a random number of nodes as follows: 100, 300, 500 and 1000, with the number of tasks varying from 6000 to 10000 in steps of 1000.

The data processed in the rest of this paper are the result of ten (10) independent recurrence of each simulation phase in order to extract a significant mean.

This collection allowed us to conduct a comparative study on the Intra-Scheduler and Inter-Schedulers load balancing strategy.

This limitation is due to the hardware constraints of the machine to run the simulation.

All experiments were performed on a 2.60 GHz Intel Core i5 CPU PC with 8 GB memory running Windows 10 Professional.

To achieve our simulation objectives, we opted to use the PeerSim simulator.

3.2.2. Simulator

- PeerSim [27] is a simulator whose main objective is to provide high scalability, with network sizes up to 1000000 nodes, which characterises its dynamics and extensibility. Its modularity facilitates the coding of new applications. The PeerSim configuration file has three types of components: protocols, dynamics and observers.
- Simulation [27] is a widely used method for performance evaluation. It consists of observing the behaviour of a simplified model of the real system with the help of an

appropriate simulation program which will result in graphs that are easy to analyse and interpret. This method is closer to the real model than analytical methods, and is used when evaluation of direct measure becomes very expensive.

3.2.3. Some Performance Indicators

We are interested in reducing the response time, the process latency for jobs, and the running time of MapReduce tasks. The following performance metrics have been analysed in order to measure the performance.

- Mean Response Time: This is the time spent in the queue of ready processes before the first execution. To calculate the mean response time (MRT) of the processes we use the following formula: $\mathbf{MRT} = \sum_{i=0}^{n} \tau_{ri}/n$ With τ_{ri}= completion time - arrival date
- Mean Process Latency: This is the mean time a process spends waiting. The mean process latency (MPL) is calculated as follows $\mathbf{MPL} = \sum_{i=0}^{n} \tau_{ai}/n$ With τ_{ai}= τ_{ri} - running time
- Running Time: or Restitution time is the time that elapses between the submission of the job and its completion.
- Variance: The variance of a series, noted V, is the average of the squares of the deviations of each value from the mean m of the series.
  $V= (n_1 \times (x_1 - m)^2 + n_2 \times (x_2 - m)^2 + ... + n_p \times (x_p - m)^2)/N$
- Standard deviation: The standard deviation of a series is the number, noted σ, with $\sigma = \sqrt{V}$, where V is the variance of the series.
- Max / Min: represented respectively the maximum and minimum mean of the results of the ten (10) independent runs of each simulation phase.

3.3. Load Balancing Performances

We compare CLOAK-Reduce and the CLOAK DHT, unlike CLOAK DHT, which centralized load balancing strategy, CLOAK-Reduce has a hierarchical load balancing strategy.

Therefore, our strategy performs a load balancing between a JobManager and its JobManagers candidate before considering a load balancing between the different JobManagers of a scheduler and in extreme cases a load balancing between the schedulers.

Table 1. Mean Response Time efficiency ratio (%)

Nodes	Taks number					Mean
	6000	7000	8000	9000	10000	
100	41.22	47.11	46.50	44.21	43.21	44.45
300	34.65	45.92	48.34	47.23	45.16	44.26
500	33.93	37.49	36.42	33.59	37.44	35.77
1000	33.35	36.89	36.15	36.69	41.99	37.01

Table 1, We obtain a minimum average response time gain of 35.77% and a maximum of 44.75% with an overall mean of 40.37%. Figure 8 illustrates the improvement in mean response time obtained by our load balancing model different numbers of nodes by varying the number of jobs

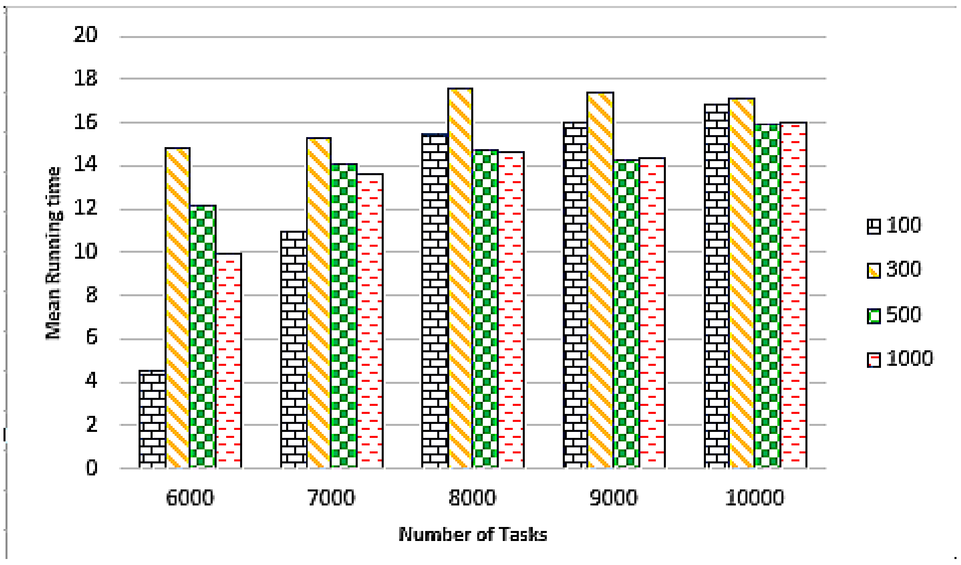


Figure 8. Mean Response Time comparison for varying number of tasks

Figure 9 shows the mean process latency improvement obtained by our load balancing model different numbers of nodes by varying the number of tasks. We obtain a minimum average response time gain of 43.43% and a maximum of 52.06% with an overall mean of 48.18% in Table 2.

Table 2. Mean Process Latency efficiency ratio (%)

Nodes	Taks number					Mean
	6000	7000	8000	9000	10000	
100	48.17	52.75	45.50	53.21	42.98	48.50
300	45.56	53.41	55.02	54.09	52.22	52.06
500	39.09	47.14	50.82	51.83	54.69	48.71
1000	33.40	39.47	43.46	48.02	52.82	43.34

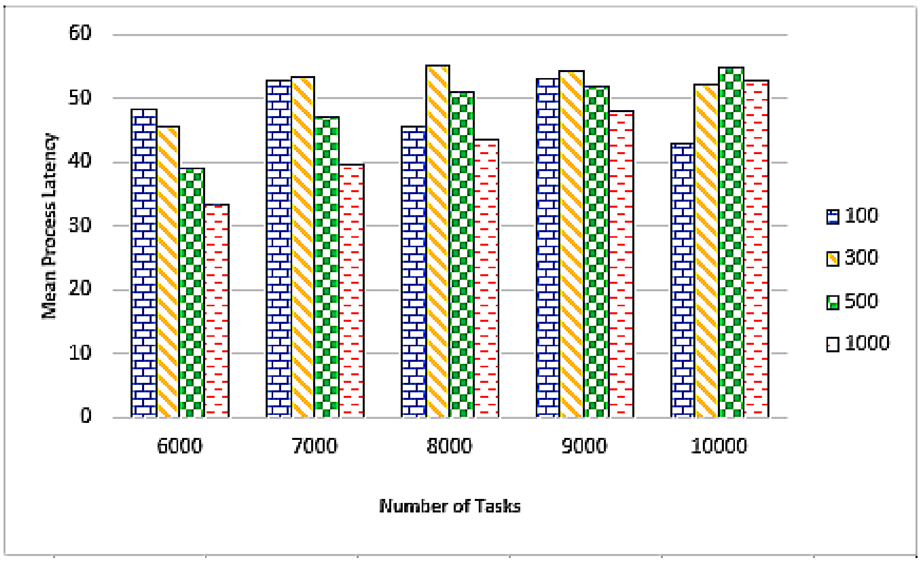


Figure 9. Mean Process Latency comparison for varying number of tasks

Figure 10 illustrates the mean running time improvement obtained by our load balancing model different numbers of nodes by varying the number of tasks. We obtain a minimum mean response time gain of 12.75% and a maximum of 16.46% with an overall average of 14.29% in Table 3.

Table 3. Mean Running Time efficiency ratio (%)

Nodes	Taks number					Mean
	6000	7000	8000	9000	10000	
100	4.51	10.97	15.44	15.97	16.88	12.75
300	14.83	15.28	17.61	17.40	17.16	16.46
500	12.11	14.10	14.76	14.30	15.92	14.24
1000	9.93	13.62	14.67	14.37	16.02	13.72

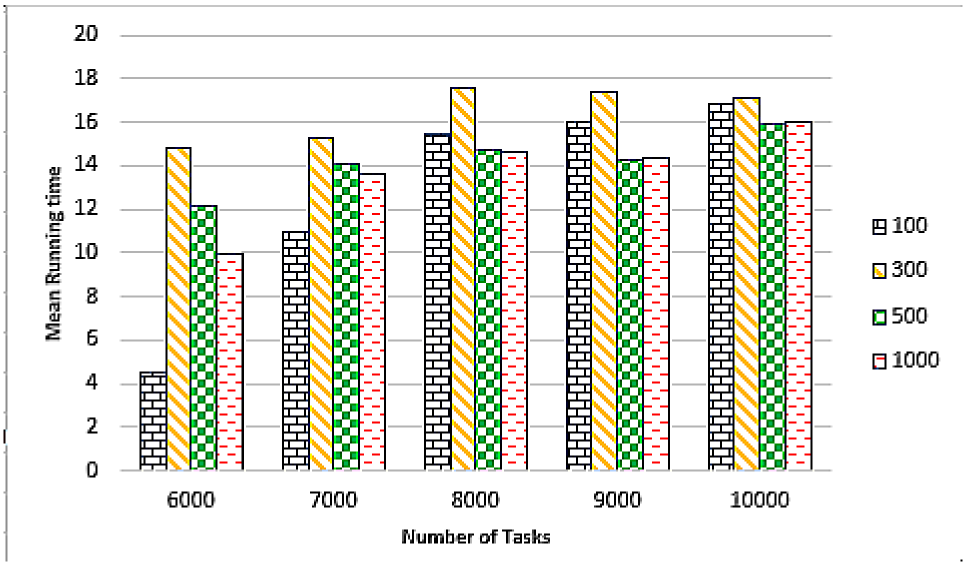


Figure 10. Mean Running Time comparison for varying number of tasks

From the different observations above, we can confirm that CLOAK-Reduce load balancing strategy certainly presents better results with respect to the three performance indicators that were the subject of this study. However, the irregularity of the time saving as a function of the number of nodes and tasks may be due to underloaded or even inactive nodes as the tasks are submitted according to a principle described in 3.1.1.

4. Conclusions and Perspectives

This paper addresses the problem of load balancing and scheduling of CLOAK-Reduce tasks. We proposed a hierarchical and dynamic load balancing model based on the tree structure of the CLOAK DHT. On this balancing approach using key hashing allowing more or less efficient affinity between nodes and more deterministic, we defined a hierarchical load balancing strategy to improve the mean response time of tasks in the model application with minimal communication.

The experimental results show that the hierarchical architecture of the model allows load balancing between several JobManagers in charge of job submission and their circular replicas (candidate JobManagers) or even JobManagers in the same scheduler. These algorithms complement each other while maintaining centralized management on the scheduler.

Therefore, we have assigned a queue role to the level 0 node. This removal of the root in our load balancing strategy improves the balancing performance of CLOAK-Reduce.

In the future, we will compare CLOAK-Reduce with other DHT-MapReduce platforms, and we will perform simulations to demonstrate its performance against distributed processing architectures.

Finally, we want to see to what extent we can apply this approach to other hybrid storage and distributed processing platforms.

REFERENCES

[1] Neha Verma, Dheeraj Malhotra, et Jatinder Singh (2020) “Big data analytics for retail industry using MapReduce-Apriori framework”, Journal of Management Analytics, p. 1-19.

[2] Telesphore Tiendrebeogo, Daouda Ahmat, and Damien Magoni, (2014) “Évaluation de la fiabilité d’une table de hachage distribuée construite dans un plan hyperbolique”, Technique et Science Informatique, TSI, Volume 33 - n◦ 4/2014, Lavoisier, pages 311–341

[3] Telesphore Tiendrebeogo and Damien Magoni, (2015) “Virtual and consistent hyperbolic tree: A new structure for distributed database management”. In International Conference on Networked Systems, pages 411–425. Springer, 2015.

[4] Tiendrebeogo, Telesphore, (2015) “A New Spatial Database Management System Using a Hyperbolic Tree”, DBKDA 2015: 53.

[5] Telesphore Tiendrebeogo, Mamadou Diarra, (2020) “Big Data Storage System Based on a Distributed Hash Tables system” International Journal of Database Management Systems (IJDMS) Vol.12, No.4/5.

[6] Jeffrey Dean and Sanjay Ghemawat, (2008) “MapReduce simplified data processing on large clusters”, Communications of the ACM, 51(1) :107–113, 2008.

[7] Than Than Htay and Sabai Phyu, (2020) “Improving the performance of Hadoop MapReduce Applications via Optimization of concurrent containers per Node”, In: 2020 IEEE Conference on Computer Applications (ICCA). IEEE, p. 1-5.

[8] Dino, Hivi Ismat, et al. "Impact of load sharing on performance of distributed systems computations." International Journal of Multidisciplinary Research and Publications (IJMRAP) 3.1 (2020): 30-37.

[9] Gopi, Arepalli Peda, V. Lakshman Narayana, and N. Ashok Kumar., (2018). "Dynamic load balancing for client server assignment in distributed system using genetical gorithm." Ingénierie des Systèmes d’Information 23.6.

[10] Li, Yun, et al. "Big data and cloud computing." Manual of Digital Earth. Springer, Singapore, 2020. 325-355.

[11] Ian FOSTER., (2005) “Globus toolkit version 4: Software for service-oriented systems”. In Network and parallel computing. Springer, p. 2–13.

[12] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan., (2003) “Chord: a scalable peer-to-peer lookup protocol for internet applications”. IEEE/ACM Transactions on Networking (TON), 11(1) :17–32.

[13] Nicholas JA Harvey, John Dunagan, Mike Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman, (2002) “Skipnet: A scalable overlay network with practical locality properties”

[14] Antony Rowstron and Peter Druschel., (2001) “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems”. In IFIP / ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, pages 329–350. Springer.

[15] Petar Maymounkov and David Mazieres. Kademlia, (2002) “A peer-to-peer information system based on the xor metric”. In International Workshop on Peer-to-Peer Systems, pages 53–65. Springer.

[16] Ping, Y. (2020). Load balancing algorithms for big data flow classification based on heterogeneous computing in software definition networks. Journal of Grid Computing, 1-17.

[17] Bhushan, K. (2020). Load Balancing in Cloud Through Task Scheduling. In Recent Trends in Communication and Intelligent Systems (pp. 195-204). Springer, Singapore.

[18] Gao, X., Liu, R., & Kaushik, A. (2020). Hierarchical multi-agent optimization for resource allocation in cloud computing. IEEE Transactions on Parallel and Distributed Systems, 32(3), 692-707.

[19] Abdalkafor, A. S., Jihad, A. A., & Allawi, E. T. (2021). A cloud computing scheduling and its evolutionary approaches. Indonesian Journal of Electrical Engineering and Computer Science, 21(1), 489-496.

[20] Ebadifard, F., Babamir, S. M., & Barani, S. (2020, April). A dynamic task scheduling algorithm improved by load balancing in cloud computing. In 2020 6th International Conference on Web Research (ICWR) (pp. 177-183). IEEE.

[21] Ghomi, E. J., Rahmani, A. M., & Qader, N. N. (2017). Load-balancing algorithms in cloud computing: A survey. Journal of Network and Computer Applications, 88, 50-71.

[22] Belayadi, D., Hidouci, K. W., Bellatreche, L., & Ordonez, C. (2018). Équilibrage de Distribution de Données d'une Base en Mémoire Parallèle Partitionnées par Intervalle. Business Intelligence & Big Data.

[23] BAERT, Quentin, CARON, Anne-Cécile, MORGE, Maxime, et al., (2019) "Stratégie situationnelle pour l'équilibrage de charge.

[24] Neghabi, A. A., Navimipour, N. J., Hosseinzadeh, M., & Rezaee, A. (2018). Load balancing mechanisms in the software defined networks: a systematic and comprehensive review of the literature. IEEE Access, 6, 14159-14178.

[25] S.Banerjee , J.P. Hecker, (2015) "Multi-Agent System Approach to Load-Balancing and Resource Allocation for Distributed Computing", First Complex Systems Digital Campus World EConference .

[26] Yahya Hassanzadeh-Nazarabadi, Alptekin Küpçü, et Öznur Özkasap., (2018) "Decentralized and locality aware replication method for DHT-based P2P storage systems. Future Generation Computer Systems", vol. 84, p. 32-46.

[27] Surati, S., Jinwala, D. C., & Garg, S. (2017). A survey of simulators for P2P overlay networks with a case study of the P2P tree overlay using an event-driven simulator. Engineering Science and Technology, an International Journal, 20(2), 705-720..

AUTHORS

Diarra Mamadou, master degree in information systems and decision support system in nazi BONI university Burkina Faso. My interest research topic is image watermarking for decision support and multimedia system.

Telesphore Tiendrebeogo, PhD and overlay network and assistant professor at Nazi Boni University. I have a master's degree in multimedia and real time system. My current research is on big data and image watermarking