Conception of a Load Balancing Strategy for CLOAK-Reduce

Diarra Mamadou and Tiendrebeogo B. Telesphore

Conception of a load balancing strategy for CLOAK-Reduce

DIARRA Mamadou
Department of Mathematics and Computer Science
Nazi Boni University
Bobo-Dioulasso, Burkina Faso
email diarra.md@univ-bobo.bf

TIENDREBEOGO B. Telesphore
Department of Mathematics and Computer Science
Nazi Boni University
Bobo-Dioulasso, Burkina Faso
email tetiendreb@gmail.com

Abstract—Distributed systems are highly heterogeneous, dynamic and unstable. It is therefore realistic to expect that some resources will fail during use. To overcome these problems and achieve better performance, it is necessary to implement load balancing algorithms that are adapted to any situation where some nodes are overloaded while others are less so or are even idle.

Load balancing between JobManager and JobManagers candidates, and between JobManagers of the same scheduler or load balancing between Schedulers, implies that additional loads are only done hierarchically.

In this paper, we propose a two-level dynamic, hierarchical and decentralised load balancing strategy focusing on three performance indicators namely: response time, process latency and running time of MapReduce jobs.

The first level of load balancing is intra-scheduler, in order to avoid the use of the large-scale communication network, and the second level of load balancing is inter-scheduler, for load regulation of our whole system. The proposed solution provides a better optimisation of the load balancing process and an improvement of the task mean response time with minimal communication.

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

I. Introduction

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

DHTs have been widely studied because of their attractive properties: efficiency and simplicity with Chord [3], controlled data placement with SkipNet [4], Pastry [5] routing and localization, and good consistency and reliable performance with Kademlia [6].

CLOAK[1] DHT, the basis of our work, is based on a hyperbolic tree constructed in hyperbolic space *(hyperboloid)* and projected stereographically into a Poincaré disk (Euclidean plane) of radius 1 centred at the origin where the tree node uses a virtual coordinate system [7]–[9].

Like DHTs, several studies and many load-balancing algorithms [10]–[12] have been proposed for traditional cluster computing and distributed systems..

Identify applicable funding agency here. If none, delete this.
[1]Covering Layer Of Abstract Knowledge

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

Although the problem of load balancing of homogeneous distributed systems has been intensively studied, the new challenges related to heterogeneous distributed systems make it an interesting topic for many projects because of the management of node heterogeneity.

Our objective is to propose a load balancing and task allocation strategy for our distributed CLOAK-Reduce model inspired by the CLOAK DHT [7]–[9] and MapReduce [13]–[15].

We will then evaluate the performance of this strategy by simulations.

Our study focuses on the following aspects:

1) Our first contribution is the summary of our model of distributed processing platform CLOAK-Reduce.
2) Our second contribution is the description of the load balancing strategy of CLOAK-Reduce.
3) the third 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 covers related works;
2) a second step, we describe CLOAK-Reduce platform;
3) the third part, we analyse the results of the different simulations.

II. Related Works

A. Distributed Systems

Distributed System is a heterogeneous set of independent computers that appears to its users as a single coherent system [16].

It 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 take into account the communication delay between the different nodes [17] .
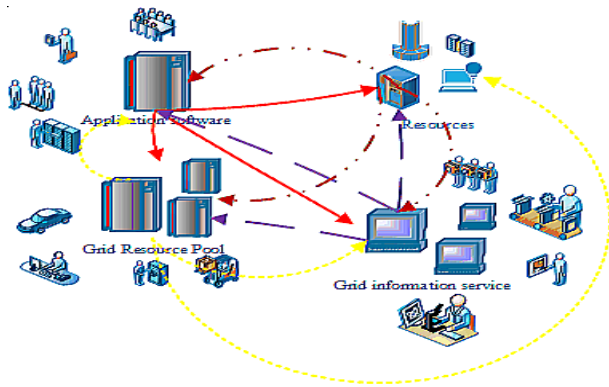
Fig. 1. Distributed systems computing environment

B. Grid Computing

Grid computing is a virtual infrastructure made up of a set of potentially shared, heterogeneous, distributed, autonomous and delocalised computing resources [18].

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 for 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 [19].

C. Distributed Hash Tables

A 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 [7], [8], [16].

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.

D. Load Balancing

Load balancing is the process of allocating workloads and resources between nodes so that no node is overloaded, underloaded or idle. With the advent of Big Data and the proliferation of connected objects, the world has become digital, which can lead to system failure, as the number of requests submitted leads to system overload.

Therefore, to solve this overload problem, load balancing algorithms are used to distribute the tasks among several nodes. These algorithms track and manage the demand of running applications by distributing resources across multiple systems. Load balancing also aims to optimise the use of resources.

CLOAK DHT, to solve this problem uses the root as the only controller so that it can handle the workloads, but after some time, if more requests are made, it will cause the system to fail again. This process is considered onerous because it is also costly. This is why a hierarchical solution is considered the best, as it avoids the problem of bottlenecks by distributing the loads according to the capacity of the nodes.

E. Dynamic Load Balancing Algorithms Classification

To activate the load balancing and to decide which nodes to give the next job, load balancing algorithms are used. So load balancing algorithms are classified into Static and Dynamic load balancing algorithms.

In this paper, we mainly interested in dynamic load balancing which offers three categories of load balancing algorithms [27]: decentralized, centralized or hierarchical. 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 [29].

In a dynamic environment the lightest node among whole nodes is searched and that nodes is selected for balancing a load. These algorithms easily adapt to the changes made in the load during runtime [30], [31]. Dynamic environment is considered difficult and complex to apply but it balances a load in an efficient way.
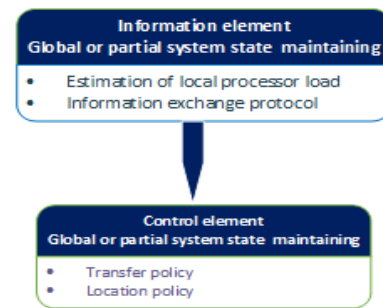

Fig. 2. Structure of a dynamic allocation algorithm

F. Some Load Balancing Performance Indicators

Some performance indicators are needed to measure the effectiveness of the different load balancing algorithms. They have been the subject of many studies which can be resumed as follows [12]:

- **Response Time:** This is the time spent in the queue of ready processes before the first execution.
- **Process latency:** or Waiting Time for Jobs is the time a process spends waiting before started.
- **Running time:** or Restitution time is the time that elapses between the submission of the job and its completion.

- **Throughput:** It indicates that how many requests (tasks) completed execution per unit time.
- **Makespan:** It is the total time required to complete all the tasks submitted to a virtual machine.
- **Fault tolerance:** It is the capabilities of the system to perform uninterrupted and uniform service even if one or more arbitrary nodes fail.
- **Migration time:** The time required to transfer a task or a virtual machine from one physical machine to another.
- **Degree of Imbalance:** It measures the imbalance among virtual machines.
- **Energy Consumption:** It is the amount of energy consumed by the devices used in the cloud computing or by the particular data centers.
- **Resource utilization:** It is the notch to which the resources of the system like CPU, Memory, Storage, and Networking etc. are uniformly utilized.
- **Reliability:** The task is transferred to any other virtual machine in case of any system failure to enrich the reliability of the system.
- **Band width (BW):** It determines the regulating of outgoing traffic from the local network and incoming traffic sent by an internet agent. This disparity of traffic over a network needs to be managed.

It should also be noted that this is not an exhaustive list, many other parameters can be taken into account depending on the needs of the researcher. As outlined in the abstract, this study is concerned with the Response time, Process latency, and Running time of a task.

III. OUR CONTRIBUTIONS

A. 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 3)

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 balacing and Inter-schedulers for global load balacing (Figure 4).

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

1) Summary of the invention:

a) Root node: The root is a node of coordinate (0,0), the minimal depth ($\mathcal{D}_0$) tree, of the Poincaré disk. It allows to :

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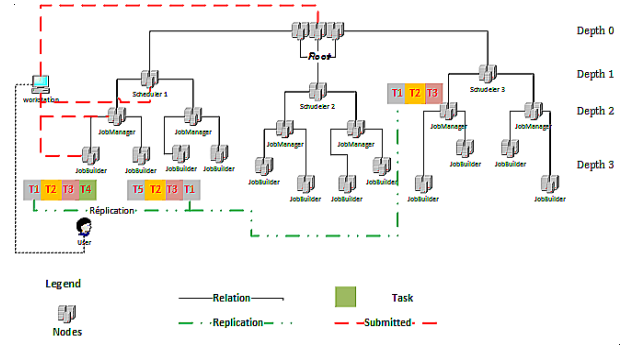- Decide to submit a task to a scheduler.

Fig. 3. Architecture of CLOAK-Reduce

b) Schedulers: The schedulers of depth ($\mathcal{D}_1$), have the function of :

- Maintain the load information of all their JobManagers;
- Decide their JobManagers load balancing;
- Synchronise their load information with their replicas to manage their failure and the others schedulers for load balacing;
- Transfer unsubmitted tasks to the root;

c) JobManagers: JobManagers candidates have a minimum depth ($\geq \mathcal{D}_2$). They are elected for:

- Supervise MapReduce jobs running;
- Manage the 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.

d) JobBuilders: The JobBuilders are of minimum depth ($\geq \mathcal{D}_3$). Their function is to:

- Execute MapReduce jobs;
- Synchronize the images of their different works as they go on their radial replicas of the depth $> \mathcal{D}_3$.
- 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 Job-Managers.
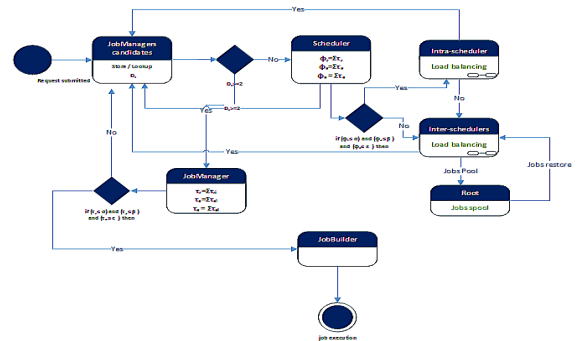

Fig. 4. CLOAK-Reduce function diagram

B. Load balancing strategy

Our strategy is to load balance at each JobManager, between JobManagers, and between Schedulers. This load balancing approach using key hashing is more deterministic and allows for more or less efficient affinity (under certain conditions) between nodes.

Load balancing is performed between nodes and their circular replicas (JobManagers candidates). It saves response time on the one hand, and reduces the communication time of the model on the other.

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.

2) Intra-scheduler load balancing:
Each JobManager can trigger a load balancing operation based on a threshold load. This threshold is estimated from cumulative information collected periodically from JobBuilders. The load balancing of the JobManager takes place between its candidate JobManagers. The inter-JobManager load balancing is ordered by the Scheduler when that of a JobManager fails. This locality approach aims to reduce communication costs.

3) Inter-scheduler load balancing :
Inter-schedulers load balacing start only when intra-scheduler balancing fails. In this case, all the new submissions of the overloaded scheduler will be transfered to the root.

In addition, the root, depending on the scheduler's information, may instruct the overloaded scheduler to perform inter-schedulers load balancing, taking into account allocation costs and the choice of tasks to select.
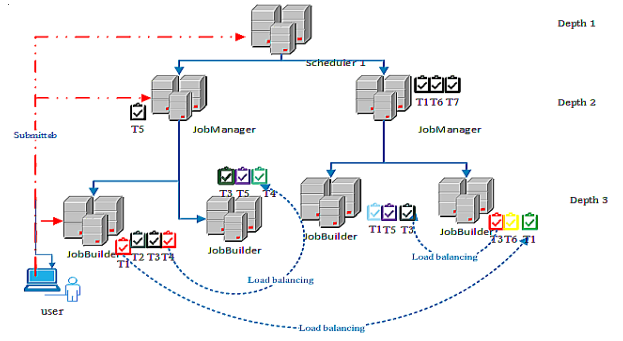

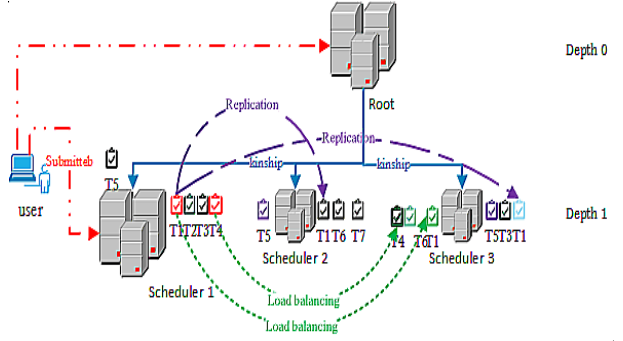Fig. 5. Intra-scheduler load balancing

Fig. 6. Inter-schedulers load balancing

4) Brief description of some load balancing algorithms:
We are focused on the Response time, the Process latency, and the average execution time of a task as a JobManager load.

Notations :

- **Break-even point for JobManagers :**
  - α_j : Response time,
  - β_j : Waiting for jobs,
  - ϵ_j : Running time for jobs.
- **Break-even point for Scedulers :**
  - α_s : Response time,
  - β_s : Waiting for jobs,
  - ϵ_s : Running time for jobs.
- **Nodes :**
  - J_b : JobBuilder,
  - J_m : JobManager,
  - J_{mc} : JobManager candidate,
  - $\mathcal{S}$: Scheduler,
  - $\mathcal{R}$: Root
- **Variables :**
  - τ_r : J_b response time ,
  - τ_a : J_b wait time ,
  - τ_e : J_b run time,
  - Φ_r : J_m response time,
  - Φ_a : J_m wait time,
  - Φ_e : J_m run time

Algorithms:

Algorithm 1: Intra-Scheduler load balancing

---

1 **for** *Every J_m of $\mathcal{S}$* **do**
2   **for** *Each time period* **do**
3   /* Receive from every J_b of J_m */
4   Calculate $\tau_r = \Sigma\tau_{ri}$;
5   Calculate $\tau_a = \Sigma\tau_{ai}$;
6   Calculate $\tau_e = \Sigma\tau_{ei}$;
7   Send τ_r, τ_a, τ_e to their $\mathcal{S}$ associated ;
8   **if** *$((\tau_r > \alpha_j)$ and $(\tau_a > \beta_j)$ and $(\tau_e > \epsilon_j))$* **then**
9   Transfer submissions from J_m to J_{mc} ;
10   **else**
11   Transfer submissions from J_m to J_b ;

Algorithm 2: Tasks allocation

Input: $Tr[3], Ta[3], Te[3], \Phi_r, \Phi_a, \Phi_e$
1 **for** *Every $\mathcal{S}$ of CLOAK-Reduce* **do**
2   /* Calculate the capacities of $\mathcal{S}$*/
3   Calculate $\Phi_r = \Sigma\tau_r$;
4   Calculate $\Phi_a = \Sigma\tau_a$;
5   Calculate $\Phi_e = \Sigma\tau_e$;
6    $Tr_S[i] = \Phi_r$;
7    $Ta_S[i] = \Phi_a$;
8    $Te_S[i] = \Phi_a$;
9   /* List $\mathcal{S}$ capacities by descending order relative to their load */
10   Sort $Tr_S[i]$; Sort $Ta_S[i]$; Sort $Te_S[i]$
11   **if** *$((\Phi_r > \alpha_s)$ and $(\Phi_a > \beta_s)$ and $(\Phi_e > \epsilon_s))$* **then**
12   Transfer submissions from $\mathcal{S}$ to $\mathcal{R}.JobSpool$;
13   **else**
14   Transfer submissions from $\mathcal{S}$ to J_{mc} ;

Algorithm 3: Inter-Schedulers load balancing

---

1 Call Algorithm 2
2 **while** $Capacity(S_i) > Capacity(S_{i+1})$ *And* $\mathcal{R}.JobSpool \neq\phi$ **do**
3   **if** $\mathcal{R}.JobSpool \neq\phi$ **then**
4   Transfer submissions from $\mathcal{R}.JobSpool$ to $\mathcal{S}$;
5   Underloaded $\mathcal{R}.JobSpool$;
6      $\mathcal{R}.JobSpool --$;
7   **else**
8      $Capacity(Capacity(S_{i+1}))$

C. Performance analysis

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 repetitions of each simulation phase in order to extract a significant average.

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.

2) Simulator:
- **PeerSim** [33] is a simulator whose main objective is to provide high scalability, with network sizes up to 10^6 nodes, which characterises its dynamicity and extensibility. Its modularity facilitates the coding of new applications. The PeerSim configuration file has three types of components: protocols, dynamics and observers.
- **Simulation** 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 by direct measurement becomes very expensive [34].

3) Some performance indicators: We are interested by 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.

- To calculate the mean response time (MRT) of the processes we use the following formula:

$$\textbf{MRT} = \sum_{i=0}^{n} \tau_{ri}/n \qquad (1)$$

With τ_{ri}= completion time - arrival date

- The mean process latency (MPL) is calculated as follows

$$\textbf{MPL} = \sum_{i=0}^{n} \tau_{ai}/n \qquad (2)$$

With $\tau_{ai}= \tau_{ri}$ - 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.

$$V = \frac{n_1 \times (x_1 - m)^2 + n_2 \times (x_2 - m)^2 + ... + n_p \times (x_p - m)^2}{N}$$

- **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. .

D. Load balancing performances

TABLE I
RATIO OF PERFORMANCE TO RESPONSE TIME

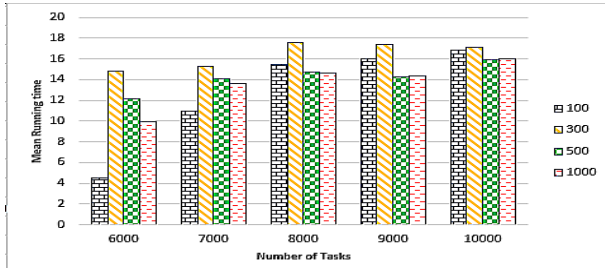| Nœuds	Nombres de tâches soumises					X
	6000	7000	8000	9000	10000	
100	41.22	47.11	46.50	44.21	43.21	44.45
300	34.65	45.92	48.34	47.23	45.16	44.26
500	33.93	37.49	36.42	33.59	37.44	35.77
1000	33.35	36.89	36.15	36.69	41.99	37.01

Fig. 7. Mean Response Time comparison for varying number of tasks

TABLE II
RATIO OF PERFORMANCE TO WAITING TIME FOR JOBS

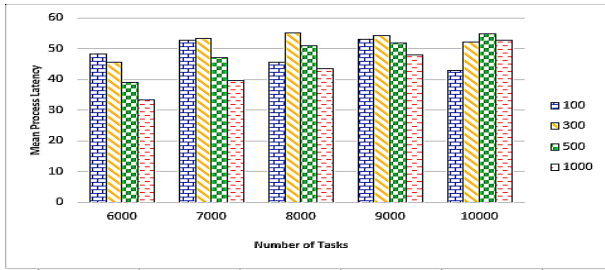| Nœuds	Nombres de tâches soumises					$\bar{X}$
	6000	7000	8000	9000	10000	
100	48.17	52.75	45.50	53.12	42.98	48.50
300	45.56	53.41	55.02	54.09	52.22	52.06
500	39.09	47.14	50.82	51.83	54.69	48.71
1000	33.40	39.47	43.46	48.02	52.82	43.34

Fig. 8. Mean Process Latency comparison for varying number of tasks

TABLE III
RATIO OF PERFORMANCE TO RUNNING TIME

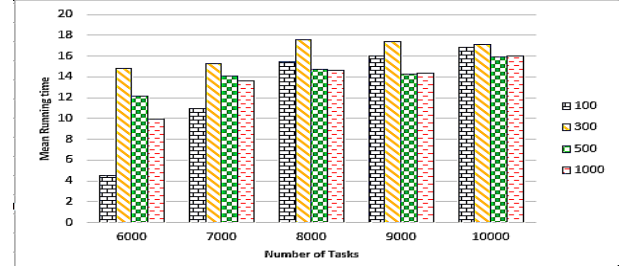| Nœuds	Nombres de tâches soumises					X
	6000	7000	8000	9000	10000	
100	4.51	10.97	15.44	15.97	16.88	12.75
300	14.83	15.28	17.61	17.40	17.16	16.46
500	12.11	14.10	14.76	14.30	15.92	14.24
1000	9.93	13.62	14.67	14.37	16.02	13.72

Fig. 9. Mean Running Time comparison for varying number of tasks

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 schedulers.

Figure 7 illustrates the improvement in mean response time obtained by our load balancing model for different numbers of nodes by varying the number of jobs. We obtain a minimum average response time gain of 35.77% and a maximum of 44.75% with an overall average of 40.37%.

Figure 8 shows the mean process latency improvement obtained by our load balancing model for 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 average of 48.18%.

Figure 9 illustrates the mean running time improvement obtained by our load balancing model for different numbers of nodes by varying the number of tasks. We obtain a minimum average response time gain of 12.75% and a maximum of 16.46% with an overall average of 14.29%.

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 III-A1.

IV. CONCLUSIONS AND PERSPECTIVES

This paper presented a new load balancing and task scheduling strategy for the CLOAK DHT. It proposed a hybrid hierarchical and decentralised load balancing model. The simulation

results show an overall improvement of our three performance indicators. It confirms our theoretical approach regarding the load balancing strategy between multiple JobManagers in charge of job submission and their circular replicas (candidate JobManagers) or even JobManagers of the same scheduler. Furthermore, inter-Scheduler load balancing gives the privilege of balancing between Schedulers and not a global balancing of the whole system. The queue role assigned to the level 0 node. This removal of the root in our load balancing strategy improves the balancing performance of the CLOAK DHT.

In perspective, we will perform simulations to highlight its performance in our CLOAK-Reduce model. Then we will conduct a comparative study with distributed processing architectures using tree structures.

REFERENCES

[1] Thomas Bourany. Les 5v du big data. Regards croises sur l'economie, (2) :27–31, 2018. lu 05 sept 2019

[2] Zhao J., Pjesivac-Grbovic J., "MapReduce : The programming model and practice ", 2009.

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

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

[5] Antony Rowstron and Peter Druschel. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems.In IFIP/ACMInternationalConferenceonDistributed Systems Platforms and Open Distributed Processing, pages 329–350. Springer, 2001.

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

[7] Telesphore Tiendrebeogo, Daouda Ahmat, and Damien Magoni. Évaluation de la fiabilité d'une table de hachage distribuée construite dans un plan hyperbolique. Technique et Science Informatique, TSI, Volume 33 - no 4/2014, Lavoisier, pages 311–341, Juin 2014

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

[9] Tiendrebeogo, Telesphore. "A New Spatial Database Management System Using an Hyperbolic Tree." DBKDA 2015 (2015): 53.

[10] Altayeb, Dina Farouk, and Fatima Abdelghani Mustafa. "Analysis of load balancing algorithms implementation on cloud computing environment." Int J Innov Res Adv Eng 6.2 (2016): 1-32.

[11] Bouafia, Zouheyr, Badr Benmammar, and Mourad Hakem. "Algorithmes d'ordonnancement des tâches dans un environnement Cloud." Revue Méditerranéenne des Télécommunications 5.2 (2015).

[12] Mishra, Kaushik, and Santosh Majhi. "A state-of-art on cloud load balancing algorithms." International Journal of computing and digital systems 9.2 (2020): 201-220.

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

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

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

[16] Andrew S. Tanenbaum and Maarten Van Steen. Distributed System : Priniples and Paradigms. Prentie Hall, 2002.

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

[18] Tian, Wenhong, and Yong Zhao. "Big data technologies and cloud computing." Optimized Cloud Resource Management and Scheduling Theory and Practice (2015): 17-49.

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

[20] J.Li, M.Qiu, J.Niu, W.Gao, Z.Zong, "Feedback dynamic algorithms for preemptable job scheduling in cloud systems". Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology, Aug. 31-Sep. 3, IEEE Xplore Press, Toronto, ON, pp: 561-564. DOI: 10.1109/WI-IAT.2010.30, 2010.

[21] J.Li, M.Qiu, Z.Ming, G.Quan, X.Qin, Z.Gu, "Online optimization for scheduling preemptable tasks on IaaS cloud systems". J. Parallel Distributed Computing. Elsevier, 72(5):666-677, May 2012.

[22] X. Xu, H. Yu, A game theory approach to fair and efficient resource allocation in cloud computing, Math. Probl. Eng. 2014 (2014) 1–14, http://dx.doi.org/10.1155/2014/915878

[23] Q. Zheng, B. Veeravalli, On the design of mutually aware optimalpricing and load balancing strategies for grid computing systems, IEEE Trans. Comput. 63 (7) (2014) 1802–1811.

[24] V. Jalaparti, G.D. Nguyen, Cloud Resource Allocation Games, Tech. rep., 2010, URL http://hdl.handle.net/2142/17427.

[25] Olejnik, R., I. Alshabani, B. Toursel, E. Laskowski, et M. Tudruj (2009). Load balancing metrics for the SOAJA framework. Scientific International Journal for Parallel and Distributed Computing, Scalable Computing: Practice and Experience, 10 (4): 1-10.

[26] LE, Viet Dung. Équilibrage de charge dans un réseau pair-à-pair structuré: une méthode dynamique intégrée. 2006.

[27] Al-Rayis, Ektemal, and Heba Kurdi. "Performance Analysis of load balancing Architectures in Cloud computing." 2013 European Modelling Symposium. IEEE, 2013.

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

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

[30] L. Gu, D. Zeng, S. Guo, A. Barnawi, Y. Xiang, "Cost efficient resource management in fog computing supported medical cyberphysical system", IEEE Trans. Emerg. Topics Comput., vol. 5, no.1, pp. 108-119, Jan. 2017.

[31] Shah, Nadeem & Farik, Mohammed. (2015). Static Load Balancing Algorithms In Cloud Computing: Challenges & Solutions. International Journal of Scientific & Technology Research. 4. 353-355.

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

[33] Alberto Montresor and Márk Jelasity. Peersim : A scalable p2p simulator. In 2009 IEEE Ninth International Conference on Peer-to-Peer Computing, pages 99–100. IEEE, 2009.

[34] Chen, Ken. Evaluation de performances par simulation et analyse: applications aux réseaux informatiques. ISTE Group, 2014.