Conception of a load balancing strategy for CLOAK-Reduce

Diarra Mamadou [1], Tiendrebeogo B. Telesphore [2]
{diarra.md21@gmail.com[1], tetiendreb@gmail.com [2] }

Department of Mathematics and Computer Science, Nazi Boni University,
Bobo-Dioulasso, Burkina Faso [1] & [2]

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.

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

1 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 [2], controlled data placement with SkipNet [3], Pastry [4] routing and localization, and good consistency and reliable performance with Kademlia [5].

JRI 2021, November 11-13, Bobo-Dioulasso, Burkina Faso

DOI 10.4108/eai.11-11-2021.2317976

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 [6, 7, 8].

Like DHTs, several studies and many load-balancing algorithms [9, 10, 11] have been proposed for traditional cluster computing and distributed systems.

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 [6, 7, 8] and MapReduce [12, 13, 14].

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.

2 Related Works

2.1 Distributed Systems

A distributed system, also known as distributed computing, is a system with multiple components located on different machines that communicate and coordinate actions in order to appear as a single coherent system to the end-user[15].

The machines that are a part of a distributed system may be computers, physical servers, virtual machines, containers, or any other node that can connect to the network, have local memory, and communicate by passing messages [15, 16].

There are two general ways that distributed systems function:

[1]Covering Layer Of Abstract Knowledge

1. Each machine works toward a common goal and the end-user views results as one cohesive unit.

2. Each machine has its own end-user and the distributed system facilitates sharing resources or communication services.

Although distributed systems can sometimes be obscure, they usually have three primary characteristics: all components run concurrently, there is no global clock, and all components fail independently of each other [17].

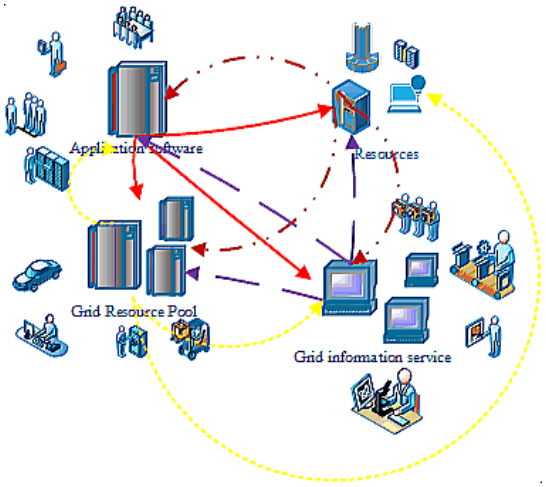

Fig. 1. Distributed systems computing environment.

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 [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].

2.3 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 [6, 7, 15].

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.

2.4 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.

2.5 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 [19].

In this paper, we mainly interested in dynamic load balancing which offers three categories of load balancing algorithms [20]: 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 [21].

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 [22, 23]. Dynamic environment is considered difficult and complex to apply but it balances a load in an efficient way.

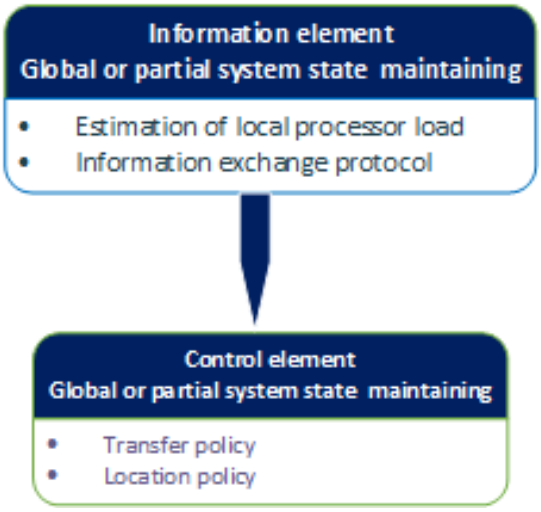

Fig. 2. Structure of a dynamic allocation algorithm.

2.6 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 [11, 24, 25]:

- **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.

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 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*.

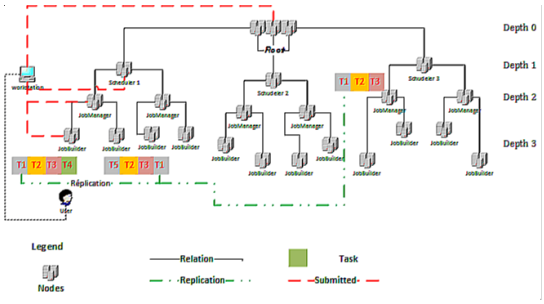

Fig. 3. Architecture of CLOAK-Reduce.

3.1.1 Summary of the invention

Root node The root is a node of coordinate (0,0), the minimal depth ($\mathscr{D}_0$) 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.

Schedulers The schedulers of depth ($\mathscr{D}_1$), 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;

JobManagers JobManagers candidates have a minimum depth ($\geq \mathscr{D}_2$). 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.

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

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

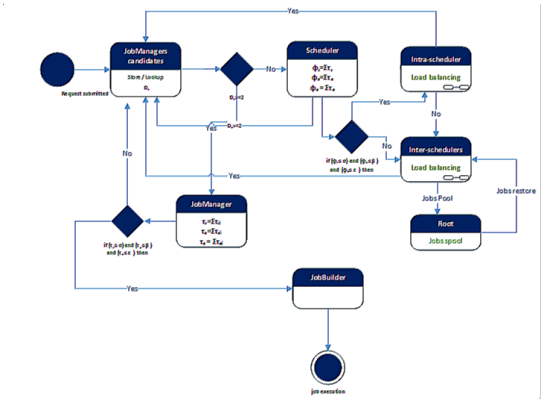

Fig. 4. CLOAK-Reduce function diagram.

3.2 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.

3.2.1 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.

3.2.2 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.

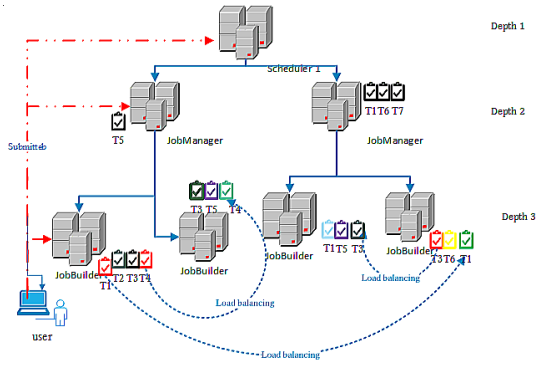

Fig. 5. Intra-scheduler load balancing.

3.2.3 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.

3.2.4 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.

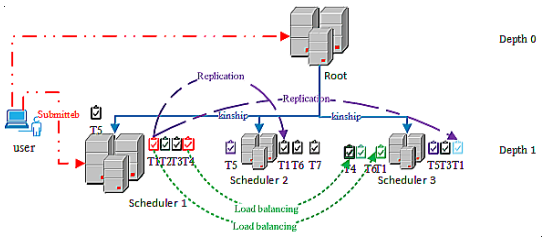

Fig. 6. Inter-schedulers load balancing.

Notations :

Table 1: Break-even point for JobManagers and Schedulers.

Variables	JobManagers	Scedulers
Response time	α_j	α_s
Waiting for jobs	β_j	β_s
Running time for jobs	ε_j	ε_s

Table 2: time variables for JobBuilders and JobManagers.

Variables	JobBuilders	JobManagers
Response time	τ_r	Φ_r
Waiting for jobs	τ_a	Φ_a
Running time for jobs	τ_e	Φ_e

Nodes :
JobBuilder (J_b), JobManager (J_m), JobManager candidate (J_{mc}), Scheduler ($\mathscr{S}$), Root ($\mathscr{R}$).

Algorithms:

for *Every* J_m *of* $\mathscr{S}$ **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 τ_r, τ_a, τ_e to their $\mathscr{S}$ associated ;
    if $((\tau_r > \alpha_j)$ *and* $(\tau_a > \beta_j)$ *and* $(\tau_e > \varepsilon_j))$ **then**
      Transfer submissions from J_m to J_{mc} ;
    else
      Transfer submissions from J_m to J_b ;
    end if
  end for
end for

Algorithm 1: Intra-Scheduler load balancing

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

Algorithm 2: Tasks allocation

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

Algorithm 3: Inter-Schedulers load balancing

3.3 Performance analysis

3.3.1 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.

3.3.2 Simulator

- **PeerSim** [26] 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 [27].

3.3.3 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:

$$\mathbf{MRT} = \sum_{i=0}^{n} \tau_{ri}/n \tag{1}$$

  With τ_{ri}= completion time - arrival date

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

$$\mathbf{MPL} = \sum_{i=0}^{n} \tau_{ai}/n \tag{2}$$

  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.
- Pre-strategy estimation in milliseconds (PSE),
- After strategy estimation in milliseconds (ASE),
- Percentage of time saved (Gain). Gain = (PSE - ASE) / PSE .

3.4 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 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%.

Table 3: Response time evolution

Plateforms		Number of submitted tasks				
Nodes	Measure	6000	7000	8000	9000	10000
100	PSE	27.83	28.21	28.58	28.95	29.31
	ASE	16.36	14.92	15.29	16.15	16.65
	Gain	41.22	47.11	46.5	44.21	43.21
300	PSE	24.57	25.40	26.24	27.11	27.98
	ASE	16.06	13.74	13.56	14.30	15.35
	Gain	34.65	45.92	48.34	47.23	45.16
500	PSE	21.31	22.11	22.91	23.72	24.55
	After	14.08	13.82	14.56	15.75	15.36
	Gain	33.93	37.49	36.42	33.59	37.44
1000	PSE	14.08	13.82	14.56	15.75	15.36
	ASE	12.86	12.69	13.37	13.81	13.14
	Gain	33.35	36.89	36.15	36.69	41.99

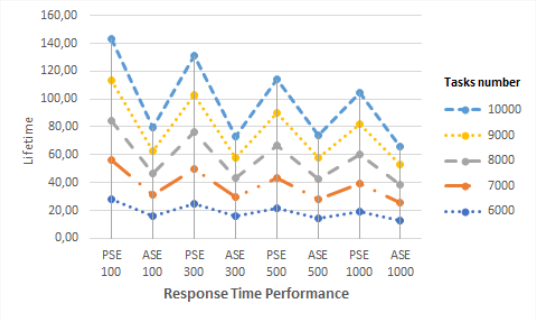

Fig. 7. Response time according tasks number variation

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%.

Table 4: Waiting time for jobs evolution

Plateforms		Number of submitted tasks				
Nodes	Measure	6000	7000	8000	9000	10000
100	PSE	34.78	35.18	35.58	35.97	36.36
	ASE	18.03	16.62	19.39	16.86	20.73
	Gain	48,17	52,75	45,5	53,12	42,98
300	PSE	35.48	36.38	37.32	38.23	39.20
	ASE	19.32	16.95	16.79	17.55	18.73
	Gain	45,56	53,41	55,02	54,09	52,22
500	PSE	25.03	25.84	26.65	27.47	35.48
	ASE	15.25	13.66	13.11	13.23	16.08
	Gain	39,09	47,14	50,82	51,83	54,69
1000	PSE	19.34	20.11	20.87	21.64	22.41
	ASE	12.88	12.17	11.80	11.25	10.57
	Gain	33,4	39,47	43,46	48,02	52,82

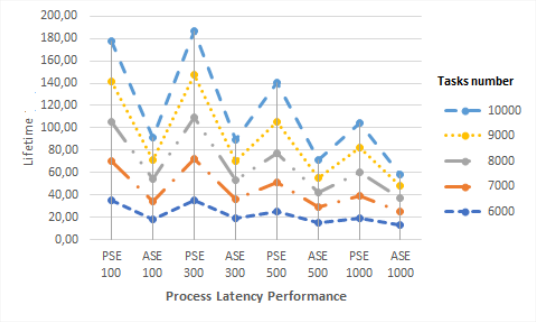

Fig. 8. Process Latency according tasks number variation.

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.

Table 5: Running time evolution

Plateforms		Number of submitted tasks				
Nodes	Measure	6000	7000	8000	9000	10000
100	PSE	4.51	4.68	4.85	5.03	5.20
	ASE	4.31	4.17	4.10	4.22	4.32
	Gain	4,51	10,97	15,44	15,97	16,88
300	PSE	14.83	15.96	16.97	17.99	19.01
	ASE	12.63	13.52	13.98	14.86	15.75
	Gain	14,83	15,28	17,61	17,4	17,16
500	PSE	12.11	13.02	13.92	14.84	15.79
	ASE	10.64	11.19	11.87	12.72	13.28
	Gain	12,11	14,1	14,76	14,3	15,92
1000	PSE	9.93	10.80	11.68	12.61	13.52
	ASE	8.94	9.33	9.97	10.79	11.35
	Gain	9,93	13,62	14,67	14,37	16,02

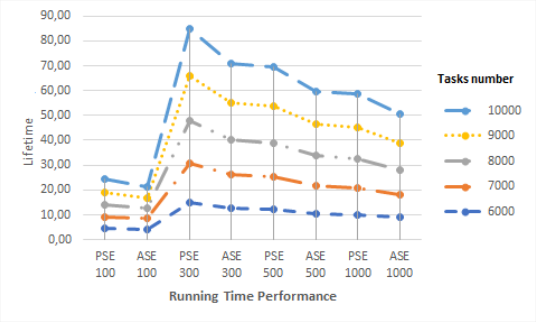

Fig. 9. Mean Running Time according tasks number variation

4 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] Bourany T. Les 5V du big data. Regards croises sur l'economie. 2018;(2):27-31.

[2] Stoica I, Morris R, Liben-Nowell D, Karger DR, Kaashoek MF, Dabek F, et al. Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on networking. 2003;11(1):17-32.

[3] Harvey NJ, Dunagan J, Jones M, Saroiu S, Theimer M, Wolman A. Skipnet: A scalable overlay network with practical locality properties. 2002.

[4] Rowstron A, Druschel P. 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. Springer; 2001. p. 329-50.

[5] Maymounkov P, Mazieres D. Kademlia: A peer-to-peer information system based on the xor metric. In: International Workshop on Peer-to-Peer Systems. Springer; 2002. p. 53-65.

[6] Tiendrebeogo T, Ahmat D, Magoni D. Évaluation de la fiabilité d'une table de hachage distribuée construite dans un plan hyperbolique. Revue des Sciences et Technologies de l'Information-Série TSI: Technique et Science Informatiques. 2014;33(4):311-41.

[7] Tiendrebeogo T, Magoni D. Virtual and consistent hyperbolic tree: A new structure for distributed database management. In: International Conference on Networked Systems. Springer; 2015. p. 411-25.

[8] Tiendrebeogo T, Diarra M. Big Data Storage System Based on a Distributed Hash Tables system. International Journal of Database Management Systems (IJDMS) Vol. 2020;12.

[9] Altayeb DF, Mustafa FA. Analysis of load balancing algorithms implementation on cloud computing environment. Int J Innov Res Adv Eng. 2016;6(2):1-32.

[10] Bouafia Z, Benmammar B, Hakem M. Algorithmes d'ordonnancement des tâches dans un environnement Cloud. Revue Méditerranéenne des Télécommunications. 2015;5(2).

[11] Mishra K, Majhi S. A state-of-art on cloud load balancing algorithms. International Journal of computing and digital systems. 2020;9(2):201-20.

[12] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM. 2008;51(1):107-13.

[13] Verma N, Malhotra D, Singh J. Big data analytics for retail industry using MapReduce-Apriori framework. Journal of Management Analytics. 2020;7(3):424-42.

[14] Htay TT, Phyu S. 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.

[15] Lindsay D, Gill SS, Smirnova D, Garraghan P. The evolution of distributed computing systems: from fundamental to new frontiers. Computing. 2021;103(8):1859-78.

[16] Biswas A, Maurya AK, Tripathi AK, Aknine S. FRLLE: a failure rate and load-based leader election algorithm for a bidirectional ring in distributed systems. The Journal of Supercomputing. 2021;77(1):751-79.

[17] Gopi AP, Narayana VL, Kumar NA. Dynamic load balancing for client server assignment in distributed system using genetical gorithm. Ingénierie des Systèmes d'Information. 2018;23(6).

[18] Tian W, Zhao Y. Big data technologies and cloud computing. Optimized Cloud Resource Management and Scheduling Theory and Practice. 2015:17-49.

[19] Nandal P, Bura D, Singh M, Kumar S. Analysis of Different Load Balancing Algorithms in Cloud Computing. International Journal of Cloud Applications and Computing (IJCAC). 2021;11(4):100-12.

[20] Ping Y. Load balancing algorithms for big data flow classification based on heterogeneous computing in software definition networks. Journal of Grid Computing. 2020;18(2):275-91.

[21] Bhushan K, et al. Load balancing in cloud through task scheduling. In: Recent Trends in Communication and Intelligent Systems. Springer; 2020. p. 195-204.

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

[23] Abdalkafor AS, Jihad AA, Allawi ET. A cloud computing scheduling and its evolutionary approaches. Indonesian Journal of Electrical Engineering and Computer Science. 2021;21(1):489-96.

[24] Hassanzadeh-Nazarabadi Y, Küpçü A, Özkasap Ö. Decentralized and locality aware replication method for DHT-based P2P storage systems. Future Generation Computer Systems. 2018;84:32-46.

[25] Praveenchandar J, Tamilarasi A. Dynamic resource allocation with optimized task scheduling and improved power management in cloud computing. Journal of Ambient Intelligence and Humanized Computing. 2021;12(3):4147-59.

[26] Montresor A, Jelasity M. PeerSim: A scalable P2P simulator. In: 2009 IEEE Ninth International Conference on Peer-to-Peer Computing. IEEE; 2009. p. 99-100.

[27] Chen H, Rossi RA, Mahadik K, Kim S, Eldardiry H. Graph Deep Factors for Forecasting with Applications to Cloud Resource Allocation. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining; 2021. p. 106-16.

[28] Author A, Author B, Author C, Author D. Title of article. Abbreviated title of journal. Year of publication;volume number(issue number):page numbers.

[29] Author A. Title of book. Edition ed. Place of publication: Publisher; Year of publication.

[30] Author A, Author B. Chapter number. In: Chapter title. edition ed. Place of publication: Publisher; Year of publication. p. page numbers of chapter.

[31] Author A. Title of paper. In: Editor AA e, editor. Proceedings of the Title of the Conference. Place of publication: Publisher's name; Year of publication. p. page numbers.