IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

Implementation of Cloud Data Storage Scheduling Algorithm

Supriya Pankaj Patare[1], Akanksha Sambhaji Raut[2], Vaibhav Pandurang Solat[3], Sakshi Lahanbhau Varpe[4], J. K. Shimpi[5], P. B. Todmal[6]
[1],[2],[3],[4]Student,[5] Project Guide, [6] Project Co-Guide
Department of Computer Engineering
Adsul's Technical Campus, Chas, Ahmednagar, India

Abstract: *Cloud storage services keep user data on many storage nodes, and two decisions decide how well they perform: which node serves a request, and how many copies of the data are kept. Common policies such as First-Come-First-Served (FCFS) and Round Robin ignore node capacity, load and bandwidth, and most file systems apply one fixed replication factor to all data. This paper presents a cloud data storage scheduling framework that takes both decisions together. An efficient scheduler gives every feasible node a composite score from its free capacity, load, bandwidth and latency and picks the best one. A replication manager then derives the replication factor from a target availability, the node failure probability, and the access frequency and criticality of the object, and places the replicas on distinct nodes with automatic failover. The environment is modelled in CloudSim and Storage CloudSim, and the scheduler will be compared with FCFS, Round Robin, Shortest Job First and Min-Min on makespan, response time, throughput, load imbalance, availability and storage cost. An administrator dashboard shows node load, replica status and alerts. The work is at the design stage, so the paper gives the architecture, the mathematical model and the evaluation plan, and no experimental results; the worked example is illustrative.*

Keywords: Cloud data storage, scheduling algorithm, storage node selection, dynamic replication factor, load balancing, makespan, CloudSim.

I. INTRODUCTION

Cloud computing offers storage, computing and networking as on-demand services over the Internet [1]. A cloud storage service keeps user data on many storage nodes, and every upload or download has to be handled by one of them. How quickly requests are served, how evenly the nodes are used and how safe the data is all depend on two choices: which node should serve a request, and how many copies of the data should be kept and where.

Both choices are usually made by simple rules. FCFS and Round Robin serve requests in a fixed order and look at neither the free space of a node, nor its current load, nor its bandwidth, so on nodes of unequal strength some become hotspots while others stay idle. Widely used distributed file systems such as GFS and HDFS keep three copies of each block by default [2], [3]. One factor for all data wastes space on rarely used objects and may be too low for critical, heavily read ones. The scheduling studies we reviewed mostly handle either placement or replication, not both together.

This paper proposes a framework that decides the node and the replication factor together. Its main contributions are:

- a storage-aware node score that combines free capacity, load, bandwidth and latency (Section IV);
- a replication factor derived from a target availability and from the access frequency and criticality of each object;
- load balancing with automatic failover to a healthy replica, and an administrator dashboard;
- an evaluation plan in CloudSim that compares the framework with FCFS, Round Robin, Shortest Job First and Min-Min (Section VII).

Copyright to IJARSCT
www.ijarsct.co.in
DOI: 10.48175/IJARSCT-40822

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

Section II reviews related work, Section III states the problem, Section IV describes the proposed framework, Section V covers security, Section VI the implementation plan, Section VII the evaluation plan, and Sections VIII and IX discuss limitations and conclude.

II. RELATED WORK

Storage-aware scheduling. Bryk et al. [4] model file transfers and bandwidth in the scheduling of workflow ensembles and show that transfer cost must be part of the decision. Yao et al. [5] formulate the placement of storage workloads as multi-dimensional vector bin packing and use a modified best-fit-decreasing heuristic to balance I/O, capacity and energy, which is close to our multi-criteria score. Han et al. [6] reduce tail latency in cloud storage while keeping the load balanced.

Task scheduling studies. Ali and Alam [7] compare task schedulers on execution time, throughput, makespan, utilisation and cost, and Soltani et al. [8] survey heuristic schedulers by quality-of-service parameters. Zuo et al. [9] optimise completion time and cost together under deadline and cost constraints. These studies guide our choice of baselines and metrics, but they do not choose the replication factor.

Simulation. CloudSim [10] is a toolkit for modelling cloud environments, and Storage CloudSim [11] extends it with object storage, so that the cost and utilisation of storage strategies can be studied without a physical data centre. We use both. Table I lists the baselines.

TABLE I

Baseline algorithms and the proposed scheduler

Algorithm	Decision rule	Weak point
FCFS	Arrival order	Ignores load
Round Robin	Nodes in turn	Ignores node speed
SJF	Smallest request first	Large requests wait
Min-Min	Smallest completion time first	Overloads fast nodes
Proposed	Composite node score	Needs weights

III. PROBLEM STATEMENT

Given a set of storage nodes N = {n1, …, nm}, where node nj has capacity Cj, free space Fj, load Lj in [0, 1], bandwidth Bj and latency Latj, and a stream of requests Q = {q1, …, qk}, where request qi has a size si, an access frequency fi and a criticality ci in [0, 1], select for each request a node and a replica set so that makespan, response time and load imbalance are small, every object meets a target availability Atarget, and the storage cost stays low. Since scheduling on unrelated machines is NP-hard [12], a greedy heuristic is used rather than an exact search.

IV. PROPOSED FRAMEWORK

A. Architecture

The framework has four layers (Fig. 1). The presentation layer is the web interface for users and the administrator. The application layer holds authentication and access control, the request handler, the efficient scheduler, the replication manager and the monitoring module. The data layer is a metadata database, and the storage layer is the set of heterogeneous storage nodes that keep the objects and their replicas.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

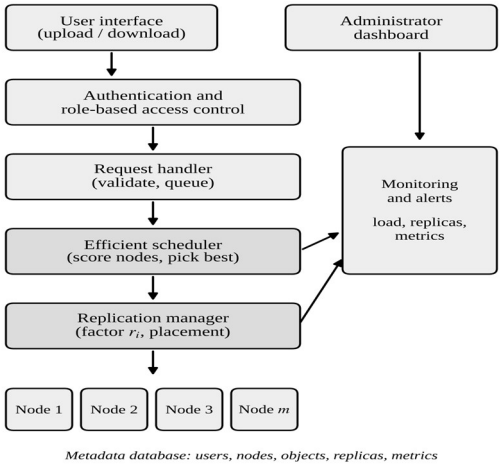


Fig. 1. Architecture of the proposed framework.

B. Node Selection

A request is first authenticated and validated, and then placed in a queue. For every feasible node, that is, every node with Fj ⩾ si, the scheduler computes the composite score

$$\text{Score}(n_j, q_i) = w_1 \frac{F_j}{C_j} + w_2(1 - L_j)$$

$$+w_3 \frac{B_j}{B_{\max}} + w_4 \left(1 - \frac{\text{Lat}_j}{\text{Lat}_{\max}}\right), \quad (1)$$

where Bmax and Latmax are the largest bandwidth and latency among the nodes, so that every term lies between 0 and 1, and the four weights w1 to w4 are non-negative, sum to 1 and are set by the administrator. The request goes to the node with the best score:

$$n_i^* = \arg\max_{F_j \geq s_i} \text{Score}(n_j, q_i) \quad (2)$$

The score of a node depends only on that node, so the m scores can be computed in parallel and combined by one arg max step. The expected completion time of request qi on node nj and the makespan of an allocation are

$$\text{ECT}_{ij} = \text{Ready}_j + \frac{s_i}{B_j} + \text{Lat}_j \quad (3)$$

$$\text{Makespan} = \max_{n_j} \sum_{q_i \in Q_j} \frac{s_i}{B_j} \quad (4)$$

where Readyj is the time at which the node becomes free and Qj is the set of requests assigned to it.

C. Usage-Driven Replication

If each node fails independently with probability p, an object kept on r replicas is available with probability A(r) = 1 − pr. The smallest number of replicas that reaches the target availability is

$$r_{\min} = \left\lceil \frac{\ln(1 - A_{\text{target}})}{\ln p} \right\rceil \quad (5)$$

For example, p = 0.05 and Atarget = 99.9% give rmin = 3 and A(3) = 99.9875%. Fig. 2 shows how fast unavailability falls as copies are added. The factor actually used for object i grows with its access frequency and criticality:

$$r_i = \min(r_{\max}, r_{\min} + u_i) \quad (6)$$

where ui = ⌈α fi/fmax⌉ + ⌈β ci⌉ is the extra copies that frequent or critical data needs, α and β are tuning constants and rmax ⩽ m caps the cost. Replicas are placed on distinct nodes chosen by the same score, and copied with rsync [13], which sends only the changed parts of a file.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

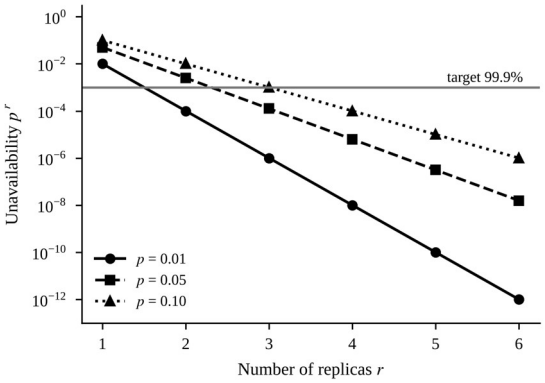


Fig. 2. Unavailability pr against the number of replicas for three node failure probabilities.

D. Load Balancing and Failover

Because every request goes to the best-scoring node, a heavily loaded node scores lower and receives less new work. Load balance is measured by the load imbalance factor, the coefficient of variation of the node loads,

$$\mathrm{LIF} = \frac{\sigma_L}{\mu_L} \qquad (7)$$

where 0 means a perfect balance. When a node fails, the monitoring module marks it unhealthy, requests are served from another replica, and the replication manager restores the replica count on a healthy node. If fewer than rmin healthy nodes remain, an alert is raised.

E. Scheduling Procedure

For each queued request the framework (1) checks authentication and rights; (2) discards nodes that lack space; (3) scores the remaining nodes with Eq. (1) and picks the best by Eq. (2); (4) stores the object; (5) computes ri by Eq. (6) and places the other replicas on the next best distinct nodes; (6) updates the metadata and the monitoring data. The procedure is a single pass over the nodes per request, so its cost grows linearly with m.

V. SECURITY

Users sign in with authentication, and role-based access control separates the rights of users and the administrator. Data in transit is protected by HTTPS/TLS and data at rest is encrypted. Replication traffic uses the same protection, and every action on the metadata is logged, so that a failed authentication or a failed replica synchronisation can be traced.

VI. IMPLEMENTATION PLAN

The storage environment is simulated in CloudSim and Storage CloudSim (Java) [10], [11]. The backend is written in Python with Django or Flask, the metadata is kept in MySQL or PostgreSQL, and the dashboard uses HTML, CSS and JavaScript (Table II). The work follows nine phases over 16 weeks: requirements, literature survey, design, environment set-up, baseline and proposed schedulers, replication and failover, load balancing and dashboard, security and testing, and evaluation.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

TABLE II
Technology stack

Layer	Tools
Simulation	Java, CloudSim, Storage CloudSim
Backend	Python, Django or Flask
Database	MySQL or PostgreSQL
Dashboard	HTML, CSS, JavaScript
Replica copy	rsync
Security	HTTPS/TLS, role-based access

VII. EVALUATION PLAN

Workloads. The proposed scheduler and the four baselines will run on identical synthetic and trace-based workloads, with growing numbers of nodes and requests, so that scalability can be checked, and with injected node failures for the availability tests.

Metrics. Makespan (Eq. (4)), average response time, throughput, utilisation, load imbalance factor (Eq. (7)), availability, and storage cost

$$\text{Cost} = c_s \sum_{i=1}^{k} r_i s_i \quad (8)$$

where cs is the cost per GB and risi is the stored volume of object i. The gain of the proposed scheduler over a baseline, in percent, for any metric M is

$$\text{Gain} = \frac{M_{\text{base}} - M_{\text{prop}}}{M_{\text{base}}} \times 100\% \quad (9)$$

Success conditions. Every valid request goes to a feasible node and its replicas to distinct healthy nodes; each object reaches Atarget; and the makespan and imbalance factor are not worse than those of the best baseline on the same workload.

Worked example. Take three nodes and a request of si = 50 GB, with weights 0.3, 0.3, 0.2 and 0.2, Bmax = 200 and Latmax = 30. Node 1 has C = 1000 GB, F = 600 GB, L = 0.7, B = 100, Lat = 20; node 2 has 1000, 400, 0.3, 200, 10; node 3 has 500, 300, 0.2, 150, 30. Table III gives the terms of Eq. (1). Node 2 wins with 0.663 although it has the lowest share of free space, because it is lightly loaded and fast. With α = β = 1, fi/fmax = 0.5, ci = 0.8, rmin = 3 and rmax = 5, Eq. (6) gives ri = 3 + 1 + 1 = 5. All values are made up to show the method, not measured.

TABLE III
Illustrative node scores

Node	*F/C*	1−*L*	$B/B_{\max}$	$1-Lat/Lat_{\max}$	Score
1	0.60	0.30	0.50	0.33	0.437
2	0.40	0.70	1.00	0.67	0.663
3	0.60	0.80	0.75	0.00	0.570

VIII. LIMITATIONS

The framework is evaluated only in simulation, and a real deployment will see network effects that a simulator does not capture. The weights and the constants α and β must be tuned, and the independence assumption behind A(r) does not hold when nodes share a rack or a power source. A heuristic cannot promise the optimal allocation. The score also depends on fresh load and bandwidth readings, so stale monitoring data would weaken it.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

IX. CONCLUSION AND FUTURE WORK

This paper presented a cloud data storage scheduling framework that selects a node by a composite score and sets the replication factor from a target availability and from the use of the data, with load balancing, failover and an administrator dashboard. The design and the mathematical model are complete and the implementation is planned; the comparison with FCFS, Round Robin, SJF and Min-Min will show how far the idea holds. Future work includes learning the weights from past load, handling correlated failures, and testing on a small physical cluster.

ACKNOWLEDGMENT

The authors thank the Department of Computer Engineering, Adsul's Technical Campus, Chas, Ahmednagar, for the facilities and guidance.

REFERENCES

1. P. Mell and T. Grance, "The NIST definition of cloud computing," NIST Special Publication 800-145, 2011.
2. S. Ghemawat, H. Gobioff and S.-T. Leung, "The Google file system," in Proc. 19th ACM Symp. Operating Systems Principles (SOSP), 2003.
3. K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop distributed file system," in Proc. IEEE 26th Symp. Mass Storage Systems and Technologies (MSST), 2010.
4. P. Bryk, M. Malawski, G. Juve and E. Deelman, "Storage-aware algorithms for scheduling of workflow ensembles in clouds," J. Grid Comput., vol. 14, pp. 359–378, 2016.
5. Y. Yao, I. Papapanagiotou and R. Callaway, "Multi-dimensional scheduling in cloud storage systems," in Proc. IEEE Int. Conf. Communications (ICC), 2015.
6. Y. Han et al., "A tail-tolerant cloud storage scheduling based on precise periodicity detection," CCF Trans. High Perform. Comput., vol. 4, pp. 321–338, 2022.
7. S. A. Ali and M. Alam, "A relative study of task scheduling algorithms in cloud computing environment," arXiv:1612.05633, 2016.
8. N. Soltani, B. Soleimani and B. Barekatain, "Heuristic algorithms for task scheduling in cloud computing: a survey," Int. J. Comput. Netw. Inf. Secur., vol. 9, no. 8, 2017.
9. L. Zuo et al., "A multiobjective hybrid cloud resource scheduling method based on deadline and cost constraints," IEEE Access, vol. 5, 2017.
10. R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Softw. Pract. Exper., vol. 41, no. 1, pp. 23–50, 2011.
11. R. Sturm, F. Jrad and A. Streit, "Storage CloudSim: a simulation environment for cloud object storage infrastructures," in Proc. 4th Int. Conf. Cloud Computing and Services Science (CLOSER), 2014, pp. 186–192.
12. M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979.
13. Tridgell and P. Mackerras, "The rsync algorithm," 1996. [Online]. Available: https://rsync.samba.org/