Workload Cost Optimization Using Dynamic Replication in Decentralized Systems

Ryoga Yoshida[1,*,†], Chuan Xiao[1,†] and Makoto Onizuka[1,†]

[1]Osaka University, Yamadaoka, Suita, Osaka, 565-0871, Japan

Abstract

Data replication plays a crucial role in decentralized systems by enhancing durability and availability. The ADR algorithm is a dynamic replication method that optimizes communication costs by adaptively adjusting the number of replicas. However, it overlooks workload costs, which are critical in real-world applications, leading to suboptimal performance, especially in parallel processing environments. To address this limitation, we propose an enhanced ADR algorithm that incorporates both communication and computational costs. Our method refines the cost model by considering the maximum-cost execution path in update transactions, ensuring a more accurate workload estimation. Additionally, we introduce an improved expansion-contraction test that efficiently optimizes replication placement. Experimental evaluations across various network topologies demonstrate that the proposed method achieves up to 12% higher throughput than the existing ADR algorithm, particularly in read-heavy environments. These results indicate that our approach provides a more balanced and efficient replication strategy, adapting to diverse workload patterns in decentralized systems.

Keywords

dynamic replication, decentralized systems, transaction management

1. Introduction

Data replication is a fundamental technique in decentralized systems, where data is replicated and stored across multiple processors. When writing data, transactions synchronize update transactions on the replicas across multiple processors to ensure durability. When reading data, the data can be retrieved from any processor holding the latest version, thereby maintaining consistency.

The number of processors where the latest data is replicated (we call replication processors) is a critical factor in data replication and can significantly impact system performance; e.g., in systems with read-heavy workloads, if the number of replication processors is small, it leads to frequent data retrieval from remote replication processors, degrading performance. In contrast, in systems with update-heavy workloads, if the number of replication processors is large, it increases the update load and degrades performance. Thus, for read-heavy workloads, the number of replication processors should be large to reduce the number of data retrievals from remote replication processors, while for update-heavy workloads, the number of replication processors should be small to decrease the update loads.

The optimal number of replication processors typically depends on the frequency of read and update transactions on each processor. In many decentralized systems, system designers must define the number of replication processors statically during the design phase, and then manually adjust it during the production phase [1]. However, this approach is suboptimal in environments with frequently fluctuating read/update transactions and is also inefficient due to the manual effort required by system designers. To overcome this, dynamic replication techniques are promising in the sense that they adaptively adjust the number of replication processors.

Specifically, the ADR algorithm [1] is one of these dynamic replication techniques. It adaptively changes the number of replication processors according to the read/update transactions by periodically making the expansion and contraction tests. However, it has two issues: (1) it focuses solely on optimizing communication cost and does not consider workload cost, which is more critical in real-world applications, and (2) prioritizes minimizing communication cost, which can lead to longer overall transaction execution times.

To overcome the above issues, we propose an enhanced ADR algorithm. Specifically, in addition to communication costs, we consider processor computational costs, allowing for a more accurate estimation of overall workload cost. Furthermore, since the execution time of update transactions in parallel environments is determined by the heaviest computational path, we redefine update transaction cost by focusing only on the maximum cost path, rather than summing up the weights of all paths.

2. Preliminaries

In decentralized systems, minimizing the workload cost across the entire system is a critical factor. We focus on applications that perform system-wide operations with the objective of reducing overall workload cost. Various dynamic replication algorithms have been proposed [1, 2, 3, 4, 5], among which the ADR algorithm [1] is designed to optimize overall communication cost by adaptively modifying the replication scheme R. Given its objective, it is considered the most relevant algorithm for achieving the goal of this study.

2.1. Replication Scheme

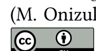
A replication scheme R represents the set of processors that hold the latest replicas and forms a variable-sized "amoeba" that shifts toward the center of the network of read/write (read/update) requests. R is created for each data object. When the number of read requests increases, the ADR algorithm expands R to reduce the communication cost by responding to read requests from a local processor or nearby

DOLAP 2025: 27th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT 2025, March 25, 2025, Barcelona, Spain

*Corresponding author.

yoshida.ryoga@ist.osaka-u.ac.jp (R. Yoshida); chuanx@ist.osaka-u.ac.jp (C. Xiao); onizuka@ist.osaka-u.ac.jp (M. Onizuka)

https://sites.google.com/site/chuanxiao1983 (C. Xiao); http://www-bigdata.ist.osaka-u.ac.jp/professor/onizuka/onizuka_en.html (M. Onizuka)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

CEUR-WS.org/Vol-3931/short6.pdf

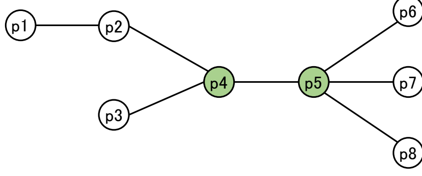


Figure 1: Replication scheme example, which consists of processors $p4$ and $p5$, depicted in green.

processors. In contrast, when the number of write requests increases, the ADR algorithm shrinks R to reduce the overhead of updating replicas in R. Hereafter, the processors included in R are referred to as R processors.

For data reading, if the processor where a read request occurs belongs to R, the processor reads the local replica. If the processor does not belong to R, the processor sequentially sends read requests to its neighboring processors. Once the request reaches an R processor, it returns the replica to the requesting processor. For data writing, the replicas in all R processors are updated synchronously by repeatedly sending the data to neighboring processors.

As an example, consider the communication network shown in Figure 1. The replication scheme R consists of processors $p4$ and $p5$, depicted in green. When reading data at processor $p1$, the nearest R processor is processor $p4$, from which the data is fetched. When writing data at processor $p1$, the update is sequentially sent to $p4$ and $p5$ via $p2$, and the replicas are updated on those processors.

Under the ADR algorithm, the replication scheme R always forms a single connected set of processors. In addition, R is created in various object units, such as a tuple, block, or text file. It is guaranteed that when the read-write pattern at each processor – the number of reads and writes issued by each processor – is regular, the replication scheme converges to the optimal configuration, regardless of the initial scheme [1].

2.2. Expansion and Contraction

R is periodically adjusted through expansion and contraction[1] every fixed period. Expansion occurs in systems with read-intensive workloads, increasing the size of R (i.e., adding more processors to R) to reduce the communication cost between the requesting processor and the R processors. In contrast, contraction occurs in systems with write-intensive workloads, decreasing the size of R (i.e., removing processors from R) to reduce the communication cost between the R processors. Whether to expand or contract R is determined by executing the expansion and contraction tests, respectively.

2.3. Issues of the ADR Algorithm

The ADR algorithm focuses solely on optimizing communication cost and does not consider workload cost, which is more critical in real-world applications. As a result, it fails to consider disparities in processing costs among processors or disparities in the execution times of transactions between read and write operations.

Additionally, in parallel processing environments, there are cases where communication time increases, but transaction execution time decreases. In such situations, the ADR algorithm prioritizes minimizing communication cost, which can inadvertently prolong overall transaction execution time.

[1]There is also an operation called "Switch". However, since the main algorithm consists primarily of expansion and contraction, it is omitted here for simplicity.

3. Proposed Method

This section introduces an improved version of the ADR algorithm. As noted earlier, the ADR algorithm optimizes only communication cost, neglecting workload cost, which is more crucial in real-world applications. To overcome this issue, the proposed method modifies the cost function of the ADR algorithm and redefines the optimization equation for a more realistic workload representation.

Specifically, in addition to communication costs, the proposed method considers processor computational costs, allowing for a more accurate estimation of overall workload cost. Furthermore, since update transaction processing time in parallel execution environments is determined by the heaviest computational path, the proposed method redefines update transaction cost by focusing only on the maximum cost path, rather than summing up the weights of all paths.

3.1. Optimization Formula

We revise the ADR algorithm to optimize the workload cost across the entire system. To optimize the workload cost rather than the communication cost, we extend the objective formula using not only the number of communications but also its associated read/update cost.

The workload cost and optimization formulation are defined as follows:

$$\begin{aligned} \mathrm{C}_{\mathrm{workload}}(R) &:= \sum_{v \in V} (\#\mathrm{U}(v, R) \times \mathrm{C}_{\mathrm{u}}(v, R) \\ &\qquad + \#\mathrm{F}(v, R) \times \mathrm{C}_{\mathrm{r}}(v, R)) \\ \operatorname*{argmin}_{R} &\ \mathrm{C}_{\mathrm{workload}}(R) \end{aligned}$$

where v denotes a processor in the network, V denotes the set of all processors, R denotes the replication scheme, $\#\mathrm{U}(v, R)$ denotes the number of update transactions, $\mathrm{C}_{\mathrm{u}}(v, R)$ denotes the cost of an update transaction, $\#\mathrm{F}(v, R)$ denotes the number of fetch transactions, and $\mathrm{C}_{\mathrm{r}}(v, R)$ denotes the cost of a read transaction. The goal of the optimization formula is to find the replication scheme R that minimizes the workload cost. In practice, R is gradually adjusted to progressively reduce the workload cost as much as possible.

In the proposed method, $\#\mathrm{F}(v, R)$, $\mathrm{C}_{\mathrm{r}}(v, R)$, and $\mathrm{C}_{\mathrm{u}}(v, R)$ are defined as follows:

$$\begin{aligned} \mathrm{C}_{\mathrm{u}}(v, R) &:= \mathrm{L}_{\mathrm{u}}(v, R) \\ \#\mathrm{F}(v, R) &:= \#\mathrm{R}(v, R)\beta(v, R) \\ \mathrm{C}_{\mathrm{r}}(v, R) &:= \mathrm{L}_{\mathrm{r}}(v, R) \end{aligned}$$

where $\mathrm{L}_{\mathrm{u}}(v, R)$ is the distance to the farthest R processor from processor v, $\beta(v, R)$ is the cache miss rate, and $\mathrm{L}_{\mathrm{r}}(v, R)$ is the distance from the nearest R processor to processor v.

When a cache hit occurs, no fetch operation is triggered, allowing local reads with zero cost. Therefore, the workload cost at a processor considers only the read costs incurred by fetch operations, which is determined by multiplying the number of read transactions $\#\mathrm{R}(v, R)$ by the cache miss rate $\beta(v, R)$, resulting in $\#\mathrm{F}(v, R)$.

3.2. Expansion-Contraction Test

The workload cost is optimized through an expansion-contraction test, which is executed after every k successfully completed transactions. In the expansion-contraction test, for each expansion-contraction pattern, the system determines whether to expand expandable processors or shrink shrinkable processors by calculating the differential workload cost.

Similar to the ADR algorithm, each expansion-contraction test restricts operations such as expanding or contracting beyond one hop and forming a discontinuous R. These restrictions are imposed due to computational complexity concerns and the potential excessive fluctuation in $\#\mathrm{F}(v)$ and $\#\mathrm{U}(v)$ before and after expansion-contraction.

Next, we explain the method for computing $\delta(R)$, the optimal expansion-contraction pattern set that minimizes the differential workload cost. $\delta(R)$ is calculated as follows:

$$\begin{aligned}\delta(R) = &\operatorname*{argmin}_{E_i \subseteq E, C_j \subseteq C} \mathrm{C}_{\text{workload}}(R_{E_i,C_j}) - \mathrm{C}_{\text{workload}}(R) \\ = &\operatorname*{argmin}_{E_i \subseteq E, C_j \subseteq C} \sum_{v \in V} \#\mathrm{U}(v, R_{E_i,C_j}) \times \mathrm{L_u}(v, R_{E_i,C_j}) \\ &+ \sum_{v \in V} \#\mathrm{F}(v, R) \times \Delta_{E_i,C_j} \mathrm{L_r}(v, R)) \\ &- \sum_{v \in V} \#\mathrm{U}(v, R) \times \mathrm{L_u}(v, R) \qquad (1)\end{aligned}$$

where E denotes the set of expandable processors, and C represents the set of contractible processors. R_{E_i,C_j} denotes the replication scheme after expanding processors E_i and contracting processors C_j. $\Delta_{E_i,C_j} f(R)$ denotes $f(R_{E_i,C_j}) - f(R)$. Here, assuming that $\Delta_{E_i,C_j} R = R_{E_i,C_j} - R$ is sufficiently small, it is approximated that $\#\mathrm{U}(v, R) = \#\mathrm{U}(v, R_{E_i,C_j})$ and $\#\mathrm{F}(v, R) = \#\mathrm{F}(v, R_{E_i,C_j})$.

3.3. Reduction of the Search Space

Equation 1 requires evaluating all possible patterns, where each expandable processor can either be expanded or not, leading to $2^{|E|}$ possibilities, and each contractible processor can either be contracted or not, leading to $2^{|C|}$ possibilities. A straightforward computation results in an exponential search space of $O(2^{|E|+|C|})$, which is impractical for scalability. Thus, reducing the search space is necessary.

Figure 2 illustrates the concept of reducing the search space (processors and nodes are treated as equivalent in this figure). For simplicity, assume that updates originate only from the center processor of the R-tree (we call R-center processor) and that all processor-to-processor distances are 1. After expansion-contraction, processors can be grouped based on their distance from the R-center processor, referred to as the maximum R-center distance in this paper. Graphs with identical maximum R-center distances exhibit the same update costs, making total cost dependent solely on read operations. Since read costs decrease as R expands, only the case with the largest R set within each group needs to be considered. Thus, the number of such groups determines the search space, which corresponds to the possible values of the maximum R-center distance after expansion-contraction, resulting in a complexity of $O(|E| + |C|)$.

Only R-leaf processors can become maximum R-center processors after expansion-contraction. The number of possible types of R-leaf processors after expansion-contraction consists of the original R-leaf processors ($|C|$), newly expanded R-leaf processors ($|E|$), and processors that became

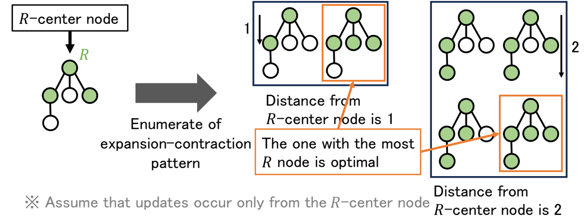


Figure 2: Illustration of search space reduction in expansion-contraction tests.

R-leaf processors due to contraction ($|C|$). Since all pre-expansion R-leaf processors must be contractible, there are exactly $|C|$ such processors. Consequently, the worst-case search space is $O(|C| + |E| + |C|) = O(|E| + |C|)$. In Figure 2, there are only two groups based on the maximum R-center distance: 1 and 2, meaning that only these two groups need to be considered for optimal expansion-contraction patterns.

However, in real scenarios, updates originate from multiple processors, not just the center processor. In such cases, even if the maximum R-center distance remains unchanged, the R-eccentric distance (the maximum shortest distance from an R processor to any other R processor) may vary, requiring separate calculations. By treating these separately, the search space is proven (proof omitted) to be $O((|E| + |C|)^2 |N_R(\sigma_R)|)$ where $N_R(\sigma_R)$ denotes neighboring R processors of the R-center processor. Additionally, using tree dynamic programming (DP) and the sliding window technique, the expansion-contraction test can be computed in $O(|V| + |N_R(\sigma_R)|(|E| + |C|) \log(|E| + |C|))$ time.

4. Experiments

This section presents experimental evaluations comparing the proposed method [2] with the ADR algorithm across three characteristic topologies.

4.1. Experimental Setup

We conducted the experiments on an EC2 m5.16xlarge instance using Dejima [6, 7, 8, 9, 10]. Dejima is a decentralized data management system designed for flexible data integration at the database level with global consistency. Each processor was represented by deploying multiple Docker containers on a single machine. For concurrency control, the Two-Phase Locking (2PL) protocol [11, 12, 13] was adopted. The evaluation criterion is throughput. Throughput was calculated by dividing the total number of successful transactions (reads and updates) executed across all processors by the execution time of 300 seconds. Additionally, the throughput was measured after the replication scheme R had converged and stabilized. The replication scheme R was created at the record level to minimize expansion cost. The expansion-contraction test was triggered every $k = 5$ transactions. The topologies used in the experiments are shown in Figure 3. The numbers in parentheses indicate the number of processors (nodes) in each topology.

We consider two types of transactions: (1) **Update** that modifies a column in a record, and (2) **Read** that reads all columns of a record. The table structure, update method, and read method in the RDBMS adhered to the YCSB [14].

[2] source code is available at: https://github.com/OnizukaLab/dejima-dynamic-replication

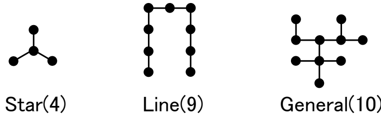


Figure 3: Topologies used in the experiments.

Table 1
Comparison of throughput between min $|R|$ and max $|R|$ in Star topology.

	Star(4)		
Read ratio	10	50	90
min $\|R\|$	41.5	**79.5**	242.8
max $\|R\|$	40.1	71.7	**327.4**

When generating the initial records for each table, record insertions into the base table of each processor were propagated to multiple processors via Dejima's data-sharing mechanism. In this experiment, 100 records were inserted into each processor as initial records, and these records were propagated across the entire system. For example, in General(10), 100 records are initially inserted into each processor, resulting in a total of 1,000 records.

4.2. Experimental Results

4.2.1. Star Topology

The experimental results for the star topology are shown in Table 1 and Table 2. Table 1 compares the case where the replication scheme R is minimized, meaning only the center processor is part of R, and the case where R is maximized, meaning updates are propagated to all processors. Table 2 shows the results of the existing method (the ADR algorithm) and the proposed method, along with their ratio (relative throughput).

For Star(4), as shown in Table 1, as the read ratio increases, max $|R|$ achieves higher throughput than min $|R|$, with the performance gap widening at higher read ratios. As the proportion of read transactions increases, their impact becomes greater than that of update transactions, making it more effective to expand R to reduce workload cost.

A comparison of the existing method and the proposed method in Star(4) is shown in Table 2. In the existing method, performance remains stable when the read ratio is low but degrades significantly as the read ratio increases. This is because the existing method tends to overestimate update costs in parallel processing environments, leading to an unnecessarily small $|R|$ and performance degradation in read-heavy environments. This overestimation occurs because the existing method only considers communication cost, ignoring cases where updates can be executed concurrently without increasing execution time.

In contrast, the proposed method mitigates performance degradation due to its consideration of parallel execution costs. However, a slight performance decline was still observed, possibly due to the small k value, which affects the accuracy of statistical data in the expansion-contraction test. Increasing k could improve the accuracy and lead to better performance.

Table 2
Comparison of throughput between the existing method and the proposed method in Star topology.

	Star(4)		
Read ratio	10	50	90
Existing	41.8	77.2	297.4
Proposed	41.4	76.5	**324.6**
Ratio	-1%	-1%	**+9%**

Table 3
Comparison of throughput between min $|R|$ and max $|R|$ in Linear and General topologies.

	Line(9)			General(10)		
Read ratio	10	50	90	10	50	90
min $\|R\|$	**57.9**	**99.4**	298.9	**38.0**	**76.8**	279.8
max $\|R\|$	34.2	62.5	286.5	32.8	59.5	284.5

Table 4
Comparison of throughput between the existing method and the proposed method in Linear and General topologies.

	Line(9)			General(10)		
Read ratio	10	50	90	10	50	90
Existing	53.7	94.5	291.0	37.5	68.0	255.1
Proposed	54.4	95.2	293.6	37.8	**74.9**	**286.1**
Ratio	+1%	+1%	+1%	+1%	**+10%**	**+12%**

4.2.2. Linear and General Topologies

The experimental results for Linear(9) and General(10) topologies are shown in Table 3 and Table 4.

In Linear(9), as shown in Table 3, as the read ratio increases, the throughput gap between min $|R|$ and max $|R|$ widens, favoring min $|R|$. This is because a lower read ratio results in a higher proportion of update transactions, making a smaller $|R|$ more advantageous. Table 4 shows that both methods achieve performance close to the optimal min $|R|$ case, demonstrating successful optimization. The reason for the lack of a significant difference between the two methods is that, unlike Star topology, Linear topology has a lower degree of parallelism, which reduces the performance gap between the methods.

In General(10), similar to Linear(9), as shown in Table 3, as the read ratio increases, the throughput gap between min $|R|$ and max $|R|$ widens, favoring min $|R|$. Additionally, at a 10% read ratio, Table 4 shows that both methods achieve optimal values with no notable difference. At 50% and 90% read ratios, the proposed method achieves higher throughput than the existing method. This result, as in Star topology, is attributed to the consideration of parallel computation and per-processor costs. At a 90% read ratio, the existing method underperforms both min $|R|$ and max $|R|$, while the proposed method surpasses both. This indicates that the ADR algorithm sometimes converges to a worse solution than either min $|R|$ or max $|R|$, whereas the proposed method has the potential to reach an optimal solution that is neither the smallest nor the largest $|R|$.

Acknowledgements

This work is supported by JSPS Kakenhi JP23K17456, JP23K25157, JP23K28096, and JST CREST JPMJCR22M2.

References

[1] O. Wolfson, S. Jajodia, Y. Huang, An adaptive data replication algorithm, ACM Trans. Database Syst. 22 (1997) 255–314.

[2] D.-W. Sun, G.-R. Chang, S. Gao, L.-Z. Jin, X.-W. Wang, Modeling a dynamic data replication strategy to increase system availability in cloud computing environments, in: Journal of Computer Science and Technology, 2012, pp. 256–272.

[3] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, D. Feng, Cdrm: A cost-effective dynamic replication management scheme for cloud storage cluster, in: 2010 IEEE International Conference on Cluster Computing, 2010, pp. 188–196.

[4] W. Li, Y. Yang, D. Yuan, A novel cost-effective dynamic data replication strategy for reliability in cloud data centres, in: 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, 2011, pp. 496–502.

[5] J.-W. Lin, C.-H. Chen, J. M. Chang, Qos-aware data replication for data-intensive applications in cloud computing systems, IEEE Transactions on Cloud Computing 1 (2013) 101–115.

[6] O. Lab, Dejima architecture, https://github.com/OnizukaLab/dejima-prototype, 2023.

[7] Y. Asano, S. Hidaka, Z. Hu, Y. Ishihara, H. Kato, H. Ko, K. Nakano, M. Onizuka, Y. Sasaki, T. Shimizu, V. Tran, K. Tsushima, M. Yoshikawa, Making view update strategies programmable - toward controlling and sharing distributed data, CoRR abs/1809.10357 (2018). arXiv:1809.10357.

[8] Y. Asano, Z. Hu, Y. Ishihara, H. Kato, M. Onizuka, M. Yoshikawa, Controlling and sharing distributed data for implementing service alliance, in: BigComp, IEEE, 2019, pp. 1–4.

[9] Y. Asano, D. Herr, Y. Ishihara, H. Kato, K. Nakano, M. Onizuka, Y. Sasaki, Flexible framework for data integration and update propagation: System aspect, in: BigComp, IEEE, 2019, pp. 1–5.

[10] Z. Hu, M. Onizuka, M. Yoshikawa, Bidirectional collaborative data management, Bidirectional Collaborative Data Management: Collaboration Frameworks for Decentralized Systems (2024) 63–119.

[11] P. A. Bernstein, V. Hadzilacos, N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley, 1987.

[12] C. H. Papadimitriou, The Theory of Database Concurrency Control, Computer Science Press, 1986.

[13] K. P. Eswaran, J. Gray, R. A. Lorie, I. L. Traiger, The notions of consistency and predicate locks in a database system, Commun. ACM 19 (1976) 624–633.

[14] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears, Benchmarking cloud serving systems with ycsb, in: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 143–154.