A Review of Data Placement and Replication Strategies Based on Machine Learning

Amir Najjar, Riad Mokadem, Jean-Marc Pierson

A Review of Data Placement and Replication Strategies Based on Machine Learning

Amir Najjar, Riad Mokadem, Jean-Marc Pierson
Institut de Recherche en Informatique de Toulouse (IRIT)
Université de Toulouse
Toulouse, France
{amir.najjar, riad.mokadem, jean-marc.pierson}@irit.fr

***Abstract*—The global increase in data volumes has brought forth the need for scalable distributed systems that can provide satisfactory quality of service. Data placement and replication are well known techniques that provide increased performance, improved fault tolerance and higher availability. These techniques often require threshold-based activation mechanisms that can vary due to the nature of the workload and the underlying system architecture. Hence, setting and adjusting those thresholds usually require human intervention. In this context, machine learning presents a promising facet to automatically define such thresholds to adapt to different workloads and architectures. In this paper, we study the data placement and replication strategies proposed in the literature that employ machine learning. We classify such strategies based on the machine learning method, the platform on which they are deployed, the dynamicity and the achieved objectives. We describe the approach applied by each strategy as well as possible limitations. In addition, we provide insights into the experimental environments and metrics used to evaluate the strategies. We highlight the need to design data placement and replication strategies that respond better to modern needs for distributed systems. We also motivate the use of machine learning to achieve autonomy in distributed systems.**

***Index Terms*—Distributed Systems, Data Replication, Data Placement, Machine Learning.**

I. Introduction

The rise of social networks has led to the emergence of very large volumes of data in distributed systems. In such systems, Quality of Service (QoS) [1] is measured through various metrics such as response time, throughput, availability and fault tolerance. Data replication and data placement strategies are possible mechanisms to satisfy such requirements [2]; placing data closer to the user allows for improved latency [3], while data replication provides improved response time (RT), fault tolerance and availability through efficient placement of the replicas [4].

As such, numerous data placement and replication strategies have been proposed in distributed systems [5]–[7]. In addition, concerns have been raised regarding the environmental impact of such systems [8]. Therefore, such strategies must take energy consumption into account. Furthermore, such strategies also need to take into account economic costs in order to ensure the profitability of providers in commercial usages, such as the cloud [9].

The previously proposed strategies in the literature are often based on certain thresholds that require prior knowledge of the system to be defined [10]–[12]. Statistical methods have been used to predict these thresholds automatically: Khatua et al. [13] use time series, Calheiros et al. [14] use an autoregressive integrated moving average (ARIMA), and Séguéla et al [15] use control charts to define such thresholds. Machine learning could be a promising facet to automatically adapt to the characteristics of the system and learn such thresholds automatically, without prior knowledge of the underlying architecture or workload. Hence, eliminating the need for human intervention. By providing key information such as workload traces and the state of the environment, a machine learning model can be built to predict the thresholds required in a data placement or replication strategy. They could also be used to predict performance given a certain state of the system, aiding in the decision of activating the data replication mechanism and choosing the placement location of data.

Machine learning (ML) has seen increased use in various fields such as image classification, object detection, and face recognition. It has also seen use in distributed systems [5]–[7]. ML can be divided into three methods [16]: (i) Supervised learning which is characterized by a labelled dataset, (ii) unsupervised learning which is characterized by an unlabeled dataset, and (iii) reinforcement learning which is characterized by an environment and one or more agents which learn a policy through exploring actions and receiving rewards from the environment accordingly.

Few works in the literature have experimented with integrating ML in data placement and replication strategies. In this paper, we classify and present an overview of the data placement and replication strategies that employ ML. We emphasize on the contribution each work has brought as well as its limitations. We present a taxonomy of the strategies based on ML method. We then take a bird's eye view to analyze the tendencies in the strategies with regards to the platform and the dynamicity. We also focus on the contemporary aspects of large scale distributed systems, specifically the economical and environmental aspects. Finally, we delve deeper into the evaluation objectives of each, classify them and quantify the improvements perceived.

The rest of this paper is organized as follows: Section II introduces data placement, data replication, and machine learning. Section III presents the studied strategies that employ ML classified by the learning method. Section IV provides a

bird's eye analysis of the strategies studied, with insights on tendencies in the strategies. Section V reports the tendencies in the experimental environments used for evaluation and reports the results found in the strategies. Section VI provides conclusive remarks and possible facets to explore in the future for data placement and replication.

II. Background

In this section, we provide an overview of data placement and replication. In addition, we present a brief description of machine learning and its three main methods.

A. Data Placement and Replication

Data placement aims to find the appropriate location for data in a distributed architecture in order to primarily achieve lower storage costs and/or higher performance [3]. Other objectives could be achieved such as reduced energy consumption [17] and increased availability [18]. As tasks and applications have become increasingly more resource intensive, it has become challenging to assure a satisfactory QoS. Designing an efficient data placement strategy therefore allows to meet QoS demands.

Data replication aims to create copies of data in different locations to provide fault tolerance [19], increased availability [20], and higher performance [21]. It is important to note that replicating all data in all locations is not feasible due to storage constraints and cost of replication. Therefore, a data replication strategy is required. It aims to answer the following questions: [9]:

- Which data to replicate?
- When is replication required? i.e., what is the activation condition for a replication to occur?
- If a replication is to occur, how many replicas are required? This is commonly referred to as the replication factor.
- Where should the replica(s) be placed? This is a question shared with data placement.
- How to minimize the economic cost of replication? This question is particularly relevant for strategies designed for the cloud for instance.

B. Machine Learning

Machine learning employs the theory of statistics to make inference from samples or experience [22]. It can be divided into three methods:

- Supervised Learning: can be characterized by labelled data. The goal is to minimize the difference between the predicted output and the true labels. Common supervised learning tasks include regression and classification.
- Unsupervised Learning: can be characterized by unlabelled data. The goal is to learn the underlying relationships in the data, reduce the dimensionality of the data or clustering the data.
- Reinforcement Learning: one or more agents learn a policy to interact with an environment through an action-reward mechanism whereby the actions are performed by the agent impacting the environment, which in turn sends back a reward signal to the agent. The objective is to maximize the long term rewards. In contrast to supervised and unsupervised learning, a feedback mechanism is provided through the agent's exploration of the actions instead of an explicit dataset or labels.

As opposed to distributed machine learning [23], in this paper, we study the data placement or replication strategies that employed ML in their design.

III. ML-Based Data Placement and Replication Strategies

This section explores the data placement and replication strategies proposed in the literature. These strategies have employed ML in their design and hence are classified based on the ML method.

A. Supervised Learning

Supervised learning can be used in the design of strategies by using existing traces or by predicting the system performance given a certain resource allocation. Curino et al. [5] use decision trees to explain obtained partitions using predicates on frequently used attributes. Their system is tested against other methods such as manual partitioning, replication of all tables and hash partitioning. The performance of their system often matches or exceeds the best manual partitionings. A certain limitation of their strategy is the fact that the transactions are not weighted by their size. It would be more interesting to minimize the total amount of data transferred from multi-sited transactions as opposed to the number of transactions. Moreover, the strategy is static and does not take into account workload changes over time.

Xiong et al. [24] propose SmartSLA, a cost-sensitive virtualized resource management system for CPU-bound database services. The authors tested linear regression, regression trees, boosting and k-nearest neighbors to predict resource allocation. They have found boosting to be the most accurate predictor. They evaluate the performance of their system using the TPC-W benchmark [25] and the Yahoo Cloud Serving Benchmark (YCSB) [26]. A baseline is constructed with a static allocation of system resources to compare the performance. The results show the ability to minimize costs under dynamic workloads. The main limitation of their strategy is the fact that the best performing supervised learning technique still has a relatively high error of 28.6%. The authors could have explored other ML techniques such as Deep Learning to obtain more accurate predictions.

Bui et al. [27] tackle the primary issue of static replication in the Hadoop Distributed File System (HDFS). They implement Bayesian learning as a means of inference and a Gaussian process as a probability framework. To evaluate their experiments, they use the Facebook Statistical Workload Injector for MapReduce (SWIM) [28]. They compare their strategy with the default replication mechanism and two other strategies in the literature: ERMS [29] and OPTIMIS [30]. It is worthy to note that their strategy is based on an underlying assumption

that the percentage of high potential files is less than 10%. This assumption may not adapt to workloads where data potential distribution is more uniform.

Shwe and Aritsugi [31] study re-replication in HDFS. They present a proactive strategy that utilizes local regression [44] to estimate future CPU and disk utilization for the nodes in the cluster. They use an extended version of CloudSim [45], CloudSimEx [46] to compare the performance of their strategy with the baseline strategy used by HDFS, which simply chooses random nodes for re-replication. The authors did not discuss the ability of local regression to accurately predict the future resource utilization. In addition, while they found that local regression achieved better results that linear regression, they have not implemented any other ML techniques, which could predict future utilization more accurately.

Symvoulidis et al. [32] propose a data placement strategy in edge systems that classifies users based on mobility: static, local and mobile users to which different cluster assignment policies are applied. The classification of users is done via a deep learning model whose features are built using the fast causal interference algorithm [22]. They utilize a neural network whose output layer predicts the mobility class of the user. The authors find an accuracy of 81% for predicting user classes. While the improvements for static and local user classes are substantial, they are not significant for the mobile class. This may be due to their overly broad classification of mobile users. A more specific classification with different policies depending on the mobility level could improve the performance of their strategy.

B. Unsupervised Learning

Strategies can employ unsupervised learning to group data or users into clusters for which different policies could be applied. Yuan et al. [6] utilize k-means clustering for data placement based on dependencies in scientific cloud workflows. Datasets that are frequently used together are more likely to be classified within the same cluster. A dependency matrix is constructed as input features for the clustering algorithm. They simulate their algorithm on the SwinDew-C environment [47] and compare the performance between: (i) a random placement of datasets, (ii) clustered datasets at build-time then random placement at run-time, (iii) random placement at build-time then clustering at run-time and (iv) clustering during build-time and run-time. A particular issue with this strategy is the absence of dataset size in the dependency calculation. For example, transferring a large data set could cost more than transferring multiple small datasets with less total size. Another limitation is the fact that this strategy works exclusively in contexts where dependencies between data can be created.

Wang et al. [33] propose a data placement and task replication scheme for scientific workflows. They indicate that the strategy proposed in [6] does not take into account the size of the datasets. To tackle this limitation, the value in the constructed dependency matrix is multiplied by the size of the dataset. The k-means algorithm is then run on the dependency matrix with the Euclidean distance as a measure of similarity between vectors. The authors then compare their build-time placement strategy to a random placement strategy and to the strategy proposed in [6]. The run-time task replication strategy is compared to a strategy where no task replication occurs and to another where tasks are only replicated on a single level. Similar to the previous strategy, this strategy is adapted to scientific workflows where the notions of dependency between data exists. As such, it may be difficult to adopt it for other types workflows.

Sellami et al. [34] propose a data placement strategy in HDFS. They utilize Kernel Density Estimation [48] to predict future resource utilization to devise a migration plan to reallocate data from overloaded workers to underloaded ones. They use Fuzzy Formal Concept Analysis [49] to model the relationships between tasks and their consumed partitions. They compare their strategy to the strategies proposed by Jyothi et al. [50] and Xu et al. [51]. The author compares the execution time of their framework under varying workloads but do not compare to other strategies.

Ahmed et al. [35] propose a solution that clusters files into three groups in HDFS: hot, warm and cold. The replication policies are 3x and 2x for the hot and warm clusters respectively, while the erasure coding policy is applied to the files in the cold cluster. A k-means clustering algorithm then uses features such as access date and frequency to separate the files into the three aforementioned clusters. They compare the performance of their solution to the default solutions proposed by Hadoop as well as a solution proposed by Kaseb et Al. [52] (CPHARIF). The authors have not experimented with changing the default replica placement location.

In addition to incorporating supervised learning, Symvoulidis et al. [32] also use k-means clustering to separate the servers into different clusters, to which the clients will be assigned. The clustering is based on the proximity between the nodes. Their strategy has been described in detail previously. (Refer to Section III-A)

C. Reinforcement Learning

Reinforcement learning can be used to train agents to interact with the environment defined by distributed systems to learn a strategy or adjust the thresholds required to activate the placement or replication mechanisms. Morffi et al. [7] utilize Q-Learning to allocate replicated fragments in a distributed database. They utilize an ϵ-greedy policy [53] to train their agent. The results are compared to other metaheurisitc solutions such as simulated annealing and genetic algorithms. The main shortcoming of this strategy is its static nature. The obtained results are also worse compared to other metaheuristic algorithms.

Cano et al. [36] propose a background execution framework for cluster management tasks, such as data migration between storage tiers, disk balancing and garbage collection. The authors have found that dynamically executed tasks based on fixed thresholds suffer from performance degradation due to heterogeneity and the inability to adapt to varying workflows.

TABLE I
OVERVIEW AND CLASSIFICATION OF STUDIED STRATEGIES

Strategy	Replication	Platform	Dynamic	Objective	Learning	Sim	Real
Curino et al. (2010) [5]	✓	Cloud		Response Time	Supervised	✓	✓
Xiong et al. (2015) [24]	✓	Cloud	✓	Economic	Supervised		✓
Bui et al. (2016) [27]	✓	Cloud	✓	Availability Fault Tolerance	Supervised		✓
Shwe and Aritsugi (2017) [31]	✓	Cloud	✓	Availability Response Time	Supervised	✓	
Symvoulidis et al. (2023) [32]		Edge	✓	Response Time	Supervised Unsupervised	✓	
Yuan et al. (2010) [6]		Cloud	✓	Bandwidth Consumption	Unsupervised	✓	
Wang et al. (2014) [33]	✓	Cloud	✓	Response Time	Unsupervised	✓	
Sellami et al. (2021) [34]		Cloud	✓	Response Time	Unsupervised		✓
Ahmed et al. (2023) [35]	✓	Cloud	✓	Availability Response Time	Unsupervised		✓
Morffi et al. (2007) [7]	✓	Unspecified		Response Time	Reinforcement		✓
Cano et al. (2017) [36]		Cloud	✓	Response Time	Reinforcement		✓
Noel et al. (2019) [37]		Cloud	✓	Response Time	Reinforcement		✓
Liu et al. (2019) [38]	✓	Cloud	✓	Response Time	Reinforcement	✓	
Ferreira et al. (2020) [39]	✓	Cloud	✓	Throughput	Reinforcement		✓
Lu et al. (2022) [40]	✓	Cloud	✓	Response Time Throughput	Reinforcement	✓	
Alkassab et al. (2023) [41]	✓	Edge		Response Time	Reinforcement	✓	
Javed et al. (2023) [42]	✓	Edge		Throughput	Reinforcement	✓	
Zhang et al. (2023) [43]		Cloud	✓	Response Time	Reinforcement	✓	✓

Hence, they propose Q-learning to trigger such threshold-based tasks. They provide hierarchical data movement, known as tiering, as an example. It is important to note that Curator is only involved in moving data downwards (e.g. from SSD to HDD). This could be seen as a limitation of their framework.

Noel et al. [37] design an Adaptive Resource Management (ARM) system for Ceph [54], an object-based storage database. The major factors that cause performance degradation in such systems are hardware heterogeneity and the interference from background tasks such as data scrubbing. They use a stochastic policy gradient to mitigate performance hotspots. A shortcoming of their strategy is a static assumption of the workload nature. While the aim of the authors was for the system to be autonomous, information about the nature of the workload may not always be available, requiring human intervention. The authors compare the performance of their algorithm with the baseline performance of Ceph and another state of the art method called DLR [55].

Liu et al. [38] propose DataBot+, a framework for low latency in data center networks that uses deep Q-networks to automatically learn the optimal placement policies. They use the MSR Cambridge Traces [56] to compare against HASH (which is used by HDFS), commonIP [57], Sinbad [58], and DataBot [59] (which did not consider data analytical latency). A certain limitation of their strategy is that the replication factor is a fixed hyperparameter.

Ferreira et al. [39] utilize Deep Reinforcement Learning to dynamically tune the parameters of a replication middleware to achieve performance gains in terms of throughput. The replication middleware used is the Apache Bookkeeper Distributed Log. Evaluation was performed using the TPC-C benchmark [60]. While significant improvements over the baseline configuration were perceived, no other configurations were used for comparison.

Lu et al. [40] propose RLRP, a deep reinforcement learning replica placement strategy. The underlying architecture is Deep Q-Networks with two types of agents: placement agents and migration agents. The ϵ-greedy policy [53] is used to explore the state space. Their system is also extended to heterogeneous environments. The state is augmented by additional features: CPU utilization, I/O access rate and network resources. The underlying Deep Q-Network architecture is also changed from a multi-layer perceptron to an encoder-decoder based on stacked LSTM cells with attention mechanism. The system is tested in both homogeneous and heterogeneous systems against multiple strategies in the literature. A key limitation of this research is the fact that the replication factor is determined by the upper-layer application. Therefore, the authors exclusively treat the placement of replicas and do not discuss the number of replicas.

Alkassab et al. [41] deploy Deep Q-Networks and Deep Recurrent Q-Networks to implement a prefetcher for video content on edge networks. To evaluate their prefetcher, they use various other prefetching algorithms such as Belady-Prefetch and top-k Popularity [61]. The authors have compared their strategy to baseline strategies only and have not explored other strategies in the literature.

Javed et al. [42] develop a Deep Q-Network agent to optimize the placement of data requested by users on multiple base stations in the context of mmWave networks for Extended Reality applications. The ϵ-greedy policy [53] is used to explore the solution space. The performance of their agent is evaluated against the BEST-CQI heuristic, which chooses the set of base stations with the best channel quality to the user. While the authors compare the performance of their strategy on different replication factors, the replication factor is a fixed

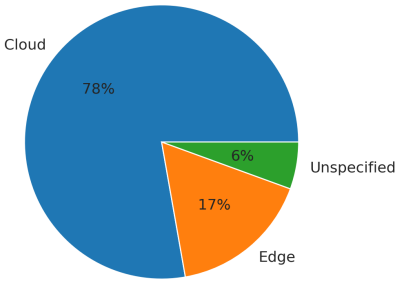

Fig. 1. Strategy distribution by platform

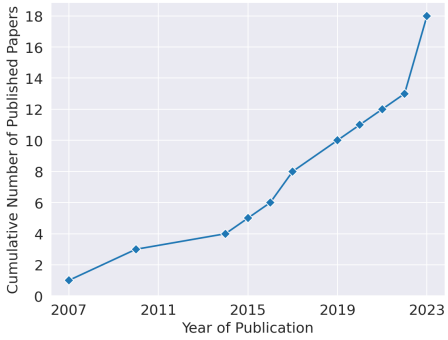

Fig. 2. No. of publications between 2000 and 2023

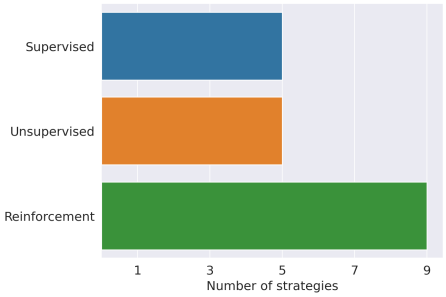

Fig. 3. Strategy distribution by ML method

hyperparameter.

Zhang et al. [43] propose a hierarchical storage framework with dynamic data placement in which storage devices with different properties in terms of speed and volume are available. Frequently accessed data can be placed in higher tiers, which are faster but typically are less in volume than lower tiers. The authors propose the implementation of a reinforcement learning agent at each tier to manage the migration of data between different tiers. They use the temporal difference (TD(λ)) learning [53]. They compare the performance of their system to other common rule based policies. In [62], they extend their system for scientific datasets, where some of the state variables are better adapted to the nature of the dataset such as the use of an interestingness measure over the size of the file in the protein translocation dataset [63]. While the authors compare their policy with three other rule-based policies based on common rules, much more common policies such as Least Recently Used or Least Frequently Used are not included in the comparison [64].

IV. Analysis

Table I presents an overview and classification of the strategies proposed in the literature. Most of the strategies study placement as well as replication, whereas [6], [32], [34], [36], [37], [43] study placement exclusively. The majority of the strategies proposed are dynamic, with only four of the strategies being static [5], [7], [41], [42]. Dynamic strategies allow to adapt the placement and replication mechanisms to changing workloads and can therefore perform better than static strategies in such circumstances.

Fig. 1 presents a pie chart of the platforms on which the strategies are deployed. Achieving satisfactory QoS in large scale systems is more challenging. Thus, most of the strategies proposed (78%) are adopted to the cloud architecture. Few other strategies are adopted to the the edge architecture (17%) [32], [41], [42]. Finally, Morffi et al. [7] did not specify the platform on which their strategy is implemented.

The importance of optimization in large-scale distributed systems can be further confirmed by studying the amount of papers published per year. Fig. 2 shows the cumulative number of published papers by year of publication. We notice a steady increase in the number of papers published throughout the years and a sharp increase in 2023.

As for optimization objectives, response time improvement was dominant, with only few strategies studying other objectives such as availability and fault tolerance. Objectives are more deeply explored in Section V.

It is also important to highlight that most strategies have aimed to optimize the objective studied. However, in platforms such as the cloud, rather than maximizing performance, achieving a satisfying QoS consistently to clients can be sufficient. Such leniency can allow to introduce other objectives such as the reduction of energy consumption or the reduction of economic costs. However, we notice a lack in the literature when it comes to the aforementioned aspects. The economical aspect of large scale systems has not been studied sufficiently despite its importance in commercial contexts. In addition, with increasing power costs and concerns about the environmental impact of distributed systems, energy consumption needs to be considered when designing a data placement and/or replication strategy. Current research fails to take into account this aspect despite the fact that ML techniques can be adapted to multiple dimensions of optimization.

The machine learning methods in the strategies proposed varied, with five strategies using supervised learning, five strategies using unsupervised learning, and nine strategies using reinforcement learning. Fig. 3 shows the number of strategies that employed each machine learning method. A more detailed taxonomy of learning methods for each strategy is shown in Fig. 4. Strategies that employed supervised learning focused primarily on predicting the expected objective given a certain resource allocation. The use of unsupervised learning remains limited to specific contexts where it is possible to form clusters for users or datasets based on their characteristics. Reinforcement learning allows for the use of simulated environments or real scenarios instead of relying on workload traces.

V. Evaluation

This section explores the different evaluation environments used by the previously discussed strategies. We also discuss the different evaluation objectives and provide insights about the perceived improvements by these strategies.

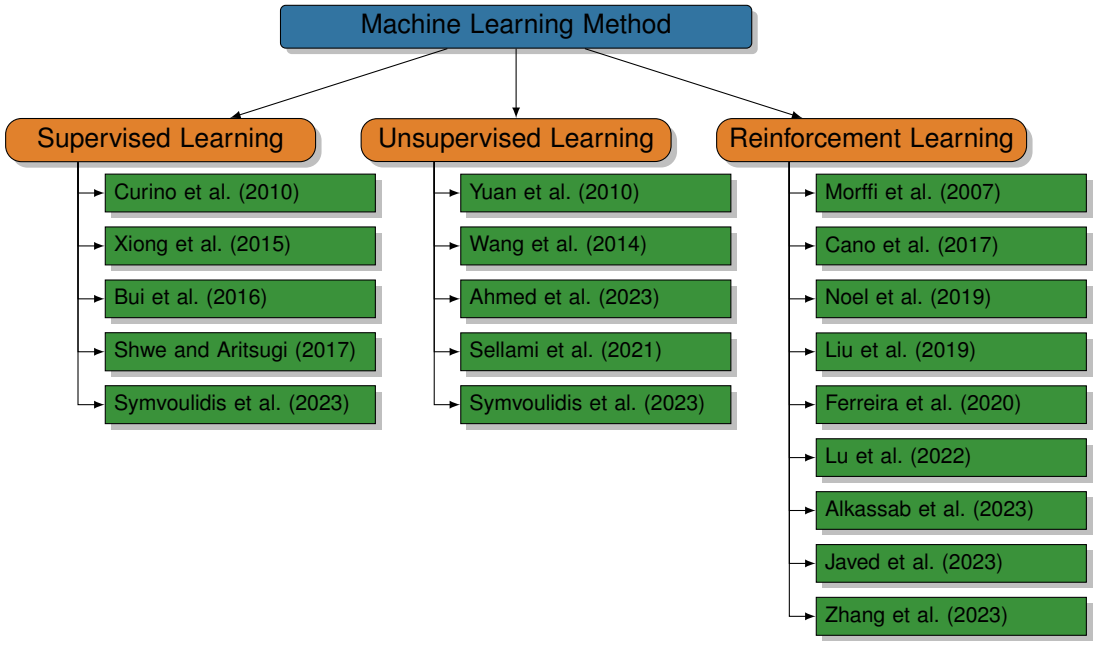

Fig. 4. Taxonomy based on ML method

A. Evaluation Environment

To validate their strategies, some works relied on simulation results [6], [31]–[33], [38], [40]–[42]. Other works relied on real environments [7], [24], [27], [34]–[37], [39]. Finally, a few works used both simulation and real environments [5], [43]. Fig. 5 demonstrates the percentages of strategies based on evaluation environments. Simulated environments were most frequently used (50%), while real environments were used slightly less (39%). Finally, few works relied on both simulation and real environments (11%).

Common benchmarks used for the evaluation of strategies were the YCSB Benchmark, [26] the TPC benchmark family, [25], [60] and the Facebook cluster traces [28]. We see the use of simulators such as CloudSimEx [46] in [31]. Other strategies developed ad-hoc simulation environments for their experiments. Python was a commonly used as a programming language for such environments [33], [42]. Finally, Yuan et al. [6] relied on SwinDeW-C, [47] a simulation environment developed in their institution.

B. Evaluation Objectives

We study objectives targeted by the different strategies discussed in this review. These objectives concern improving response time, increasing throughput, providing fault tolerance, increasing availability and reducing economic costs. Fig. 6 provides a summary of the objectives of these strategies.

1) Response Time & Throughput: Response time was the most common objective in the studied strategies. Here, we consider all measures having an impact on response time. Shwe and Aritsugi [31] study the improvement in the mean time to repair, which measures the time it takes for the system to re-replicate after a failure occurs. They find improvements between ∼2 minutes and ∼23 minutes compared to the default Hadoop re-replication policy. Symvoulidis et al. [32] measure improvements in latency. They report a reduction of 60% for static or local users and 10% for mobile users compared to other strategies in the literature. Wang et al. [33] measure the makespan of datasets, which is the total time required to execute a set of related tasks or operations. They measure improvements of 25% compared to a random placement strategy. Ahmed et al. [35] observe an improvement of 28.2% and 29% on read and write execution times compared to CPHARIF [52]. Cano et al. [36] observe improvements on latency in varying workloads compared to a strategy with thresholds fixed on heuristics. The improvements range from 2% on a varying workload to 20% on a read heavy workload. Noel et al. [37] compare the read and write response times of their strategy to the default scheme and to DLR [55]. Read response time is improved by 50% and 43% compared to the baseline and DLR respectively. Write response times are improved by 33% and 36% compared to the baselines and DLR respectively. Liu et al. [38] reduce user-experienced latency for 23.8% of users compared to HASH, a higher percentage than other strategies (up to ∼15% for Databot [59] and ∼10% for Sinbad [58] and commonIP [57]). Lu et al. [40] also tackle read response time and perceive an improvement of 10% - 50% compared to other schemes. Curino et al. [5] reduce bandwidth consumption by reducing the number of distributed transactions. They have found a reduction of 30% compared to simple partitioning schemes. Yuan et al. [6] measure data movements in scientific workflows and report a reduction of 50% compared to random strategies. Wang et al. [33] improve upon the aforementioned work by accounting for the size in data movements and measure an improvement of 10%. Sellami et al. [34] reduce

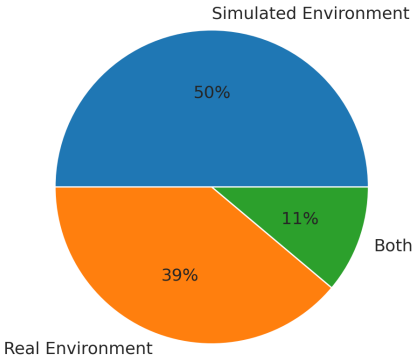

Fig. 5. Strategy distribution by evaluation environment

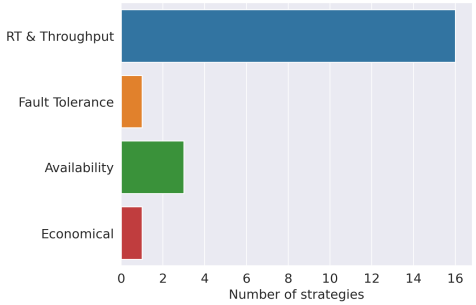

Fig. 6. Strategy distribution by objective

the amount of required workers by 2 to 15 compared to Xu et al. [51] and by 5 to 22 compared to Jyothi et al. [50] on a workload of jobs varying between 100 and 300, therefore limiting the amount of data migrations. Alkassab et al. [41] measure the prefetching accuracy and coverage in content delivery networks. These metrics insure efficiency in bandwidth consumption and improved user experience. They measure an improvement of 17% - 28% compared to the popularity-based strategies. Zhang et al. manage to reduce file transfers. They record 45 transfers per timestep using their RL-based policy compared to 311 per timestep using rule-based policies, making the RL-based policy about 6 times more efficient than rule-based policies.

Other strategies aimed to increase throughput. Ferreira et al. [39] measure throughput through the number of client replicated and written transactions and the number of client requests. They record an improvement of 370.99% on written transactions compared to a baseline configuration on some of the metrics. Lu et al. [40] increase read and write throughput by 30% - 40% and 10% respectively compared to other schemes. Javed et al. [42] measure the percentage of frames delivered within the deadline. They perceive an increase of up to 1.3% compared the BEST-CQI, a significant improvement in the context of extended reality.

2) Fault Tolerance: Fault tolerance is another objective that the discussed strategies aimed to improve. It is important to note that data placement strategies [6], [32], [34], [36], [37], [43] which do not replicate or apply erasure coding do not offer any fault tolerance. While data replication strategies provide fault tolerance, most strategies did not provide measurements. The strategy proposed by Bui et al. [27] provides the same fault tolerance as the default Hadoop replication strategy while having a lower average replication factor of 1.22 compared to 3 for Hadoop.

3) Availability: Availability has also been studied as an objective. Shwe and Aritsugi [31] use the overload probability as a metric for availability. An overloaded node may not be able to respond to user requests thus reducing availability. They reduce overload probability from 50% to 0% compared to the default policy. Ahmed et al. [35] provide the highest availability while consuming 50% and 1.12% less storage compared to the default Hadoop replication scheme and the strategy proposed in [52] respectively.

4) Economical Aspect: The rise of the cloud as a service and other architectures such as the fog or the edge has been mainly due to the existence of a profit venture. It is surprising then that aside from [24], none of the other strategies have tackled the economic perspective. Xiong et al. manage to reduce the weighted SLA penalty cost by 17.80% on average compared to a baseline resource allocation. In fact, there is a general lack in the literature when it comes to studying the economical aspect of data placement and replication strategies regardless of whether they are ML-based.

VI. Conclusion

As assuring satisfactory QoS has become essential in distributed systems, there is a need to design effective data placement and replication strategies. However, current strategies often rely on threshold-based activation mechanisms that require fine tuning according to the nature of the workflow and the underlying architecture, usually through human intervention. ML is a promising facet to define such thresholds in an automatic manner. This has led us to survey the data placement and replication strategies that employ ML. We have presented a taxonomy of the strategies based on the ML method while providing the concept behind each strategy as well as potential limitations. In addition, we characterized such strategies by the platform on which they are deployed, the dynamicity and the achieved objective. We highlighted the recent increase in use of ML to design strategies. Finally, we provided in-depth insights by analyzing the tendencies in the evaluation environment and evaluation metrics.

Current research fails to meet all contemporary expectations for distributed systems. In particular, distributed systems have seen increased commercial use. This is important to keep in mind as a replication incurs an economic cost. For example, in the cloud paradigm, assuring a satisfactory QoS to clients is sufficient, which allows to introduce other objectives such as the reduction of economic cost. In addition, concerns about the environmental impact of such systems have also been raised, as datacenters are increasingly consuming more energy. Few strategies in the literature tackle the aforementioned

aspects while achieving autonomy. We believe that ML-based strategies could incorporate the two aforementioned aspects. With the global increase of data volumes, a strategy should achieve the following requirements:

- Have both data placement and replication mechanisms; modern QoS requirements include response time as well as fault tolerance and availability.
- Have the ability to dynamically adapt to workload changes and evolution in the system.
- Satisfy application domain QoS requirements.
- Include the economical aspect in case of commercial use.
- Aim to reduce energy consumption.
- Employ ML in order to achieve autonomy and eliminate human intervention.

In terms of ML method, reinforcement learning seems to be the most suitable for a strategy design. The ability to learn directly from a simulated or a real environment presents an effective mechanism. On the other hand, supervised learning requires the presence of a labelled dataset, through the form of access traces. While certainly viable, acquiring traces can prove to be a difficult task. Finally, unsupervised learning requires specific circumstances where relations between data and/or between users can be established, such as scientific workflows or edge networks with different user mobility classes. Hence, the potential for unsupervised learning is fairly limited to specific domains.

In the future, we plan to incorporate the aforementioned requirements to design an autonomous data replication strategy based on reinforcement learning in the cloud. Our primary objectives are increasing provider profit and reducing the environmental impact. To validate our strategy, we plan to compare it to other state of the art strategies using simulations and real environments.

REFERENCES

[1] D. Armstrong and K. Djemame, "Towards quality of service in the cloud," in *Proc. of the 25th UK Performance Engineering Workshop*, 2009.

[2] D. J. Abadi, "Data management in the cloud: Limitations and opportunities," *IEEE Data Eng. Bull.*, vol. 32, no. 1, pp. 3–12, 2009.

[3] S. Mazumdar, D. Seybold, K. Kritikos, and Y. Verginadis, "A survey on data storage and placement methodologies for cloud-big data ecosystem," *Journal of Big Data*, vol. 6, no. 1, pp. 1–37, 2019.

[4] S. Goel and R. Buyya, "Data replication strategies in wide-area distributed systems," in *Enterprise service computing: from concept to deployment*, pp. 211–241, IGI Global, 2007.

[5] C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: a workload-driven approach to database replication and partitioning," *Proc. VLDB Endow.*, vol. 3, p. 48–57, sep 2010.

[6] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1200–1214, 2010.

[7] A. Rodriguez Morffi, D. Paz, M. Mainegra Hing, and L. González, "A Reinforcement Learning Solution for Allocating Replicated Fragments in a Distributed Database," *Computacion y Sistemas*, vol. 11, pp. 117–128, 12 2007.

[8] P. Lindberg, J. Leingang, D. Lysaker, S. U. Khan, and J. Li, "Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed systems," *The Journal of Supercomputing*, vol. 59, pp. 323–360, 2012.

[9] R. Mokadem and A. Hameurlain, "A data replication strategy with tenant performance and provider economic profit guarantees in Cloud data centers," *Journal of Systems and Software*, vol. 159, p. 110447, 2020.

[10] M.-C. Lee, F.-Y. Leu, and Y.-p. Chen, "PFRF: An adaptive data replication algorithm based on star-topology data grids," *Future generation computer systems*, vol. 28, no. 7, pp. 1045–1057, 2012.

[11] X. Bai, H. Jin, X. Liao, X. Shi, and Z. Shao, "RTRM: A response time-based replica management strategy for cloud storage system," in *Grid and Pervasive Computing: 8th International Conference, GPC 2013 and Colocated Workshops, Seoul, Korea, May 9-11, 2013. Proceedings 8*, pp. 124–133, Springer, 2013.

[12] U. Tos, R. Mokadem, A. Hameurlain, T. Ayav, and S. Bora, "A performance and profit oriented data replication strategy for cloud systems," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pp. 780–787, Ieee, 2016.

[13] S. Khatua, A. Ghosh, and N. Mukherjee, "Optimizing the utilization of virtual resources in Cloud environment," in *2010 IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems*, pp. 82–87, 2010.

[14] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2015.

[15] M. Séguéla, R. Mokadem, and J.-M. Pierson, "Dynamic Energy and Expenditure Aware Data Replication Strategy," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pp. 97–102, 2022.

[16] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[17] N. Maheshwari, R. Nanduri, and V. Varma, "Dynamic energy efficient data placement and cluster reconfiguration algorithm for MapReduce framework," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 119–127, 2012.

[18] H. Jin, X. Yang, X.-H. Sun, and I. Raicu, "Adapt: Availability-aware mapreduce data placement for non-dedicated distributed computing," in *2012 IEEE 32nd International Conference on Distributed Computing Systems*, pp. 516–525, IEEE, 2012.

[19] Y. Qu and N. Xiong, "RFH: A resilient, fault-tolerant and high-efficient replication algorithm for distributed cloud storage," in *2012 41st International Conference on Parallel Processing*, pp. 520–529, IEEE, 2012.

[20] D.-W. Sun, G.-R. Chang, S. Gao, L.-Z. Jin, and X.-W. Wang, "Modeling a dynamic data replication strategy to increase system availability in cloud computing environments," *Journal of computer science and technology*, vol. 27, no. 2, pp. 256–272, 2012.

[21] N. Mansouri, "Adaptive data replication strategy in cloud computing for performance improvement," *Frontiers of Computer Science*, vol. 10, pp. 925–935, 2016.

[22] P. Spirtes, C. Glymour, and R. Scheines, *Causation, prediction, and search*. MIT press, 2001.

[23] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *Acm computing surveys (csur)*, vol. 53, no. 2, pp. 1–33, 2020.

[24] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacgümüş, "SmartSLA: Cost-Sensitive Management of Virtualized Resources for CPU-Bound Database Services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1441–1451, 2015.

[25] D. A. Menascé, "TPC-W: A benchmark for e-commerce," *IEEE Internet Computing*, vol. 6, no. 3, pp. 83–87, 2002.

[26] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 143–154, 2010.

[27] D.-M. Bui, S. Hussain, E.-N. Huh, and S. Lee, "Adaptive Replication Management in HDFS Based on Supervised Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1369–1382, 2016.

[28] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: a cross-industry study of MapReduce workloads," *Proc. VLDB Endow.*, vol. 5, p. 1802–1813, aug 2012.

[29] Z. Cheng, Z. Luan, Y. Meng, Y. Xu, D. Qian, A. Roy, N. Zhang, and G. Guan, "ERMS: An Elastic Replication Management System for HDFS," in *2012 IEEE International Conference on Cluster Computing Workshops*, pp. 32–40, 2012.

[30] G. Kousiouris, G. Vafiadis, and T. Varvarigou, "Enabling Proactive Data Management in Virtualized Hadoop Clusters Based on Predicted Data

Activity Patterns," in *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pp. 1–8, 2013.
[31] T. Shwe and M. Aritsugi, "Proactive Re-replication Strategy in HDFS based Cloud Data Center," in *Proceedings of The10th International Conference on Utility and Cloud Computing*, UCC '17, (New York, NY, USA), p. 121–130, Association for Computing Machinery, 2017.
[32] C. Symvoulidis, A. Kiourtis, G. Marinos, J.-D. Totow Tom-Ata, G. Manias, A. Mavrogiorgou, and D. Kyriazis, "A User Mobility-Based Data Placement Strategy in a Hybrid Cloud/Edge Environment Using a Causal-Aware Deep Learning Network," *IEEE Transactions on Computers*, vol. 72, no. 12, pp. 3603–3616, 2023.
[33] M. Wang, J. Zhang, F. Dong, and J. Luo, "Data Placement and Task Scheduling Optimization for Data Intensive Scientific Workflow in Multiple Data Centers Environment," in *2014 Second International Conference on Advanced Cloud and Big Data*, pp. 77–84, 2014.
[34] M. Sellami, H. Mezni, M. S. Hacid, and M. M. Gammoudi, "Clustering-based data placement in cloud computing: a predictive approach," *Cluster Computing*, vol. 24, no. 4, pp. 3311–3336, 2021.
[35] M. A. Ahmed, M. H. Khafagy, M. E. Shaheen, and M. R. Kaseb, "Dynamic Replication Policy on HDFS Based on Machine Learning Clustering," *IEEE Access*, vol. 11, pp. 18551–18559, 2023.
[36] I. Cano, S. Aiyar, V. Arora, M. Bhattacharyya, A. Chaganti, C. Cheah, B. Chun, K. Gupta, V. Khot, and A. Krishnamurthy, "Curator: Self-Managing Storage for Enterprise Clusters," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, (Boston, MA), pp. 51–66, USENIX Association, Mar. 2017.
[37] R. R. Noel, R. Mehra, and P. Lama, "Towards Self-Managing Cloud Storage with Reinforcement Learning," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 34–44, 2019.
[38] K. Liu, J. Peng, J. Wang, B. Yu, Z. Liao, Z. Huang, and J. Pan, "A learning-based data placement framework for low latency in data center networks," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 146–157, 2019.
[39] L. Ferreira, F. Coelho, and J. Pereira, "Self-tunable DBMS Replication with Reinforcement Learning," in *Distributed Applications and Interoperable Systems* (A. Remke and V. Schiavoni, eds.), (Cham), pp. 131–147, Springer International Publishing, 2020.
[40] K. Lu, N. Zhao, J. Wan, C. Fei, W. Zhao, and T. Deng, "RLRP: High-Efficient Data Placement with Reinforcement Learning for Modern Distributed Storage Systems," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 595–605, 2022.
[41] N. Alkassab, C.-T. Huang, and T. L. Botran, "DeePref: Deep Reinforcement Learning For Video Prefetching In Content Delivery Networks," *arXiv preprint arXiv:2310.07881*, 2023.
[42] M. A. Javed, P. Liu, and S. S. Panwar, "Predictive Data Replication for XR Applications in Multi-Connectivity Enabled mmWave Networks," in *2023 International Balkan Conference on Communications and Networking (BalkanCom)*, pp. 1–5, 2023.
[43] T. Zhang, A. Hellander, and S. Toor, "Efficient Hierarchical Storage Management Empowered by Reinforcement Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 6, pp. 5780–5793, 2023.
[44] C. Loader, *Local regression and likelihood*. Springer Science & Business Media, 2006.
[45] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.
[46] N. Grozev and R. Buyya, "Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments," *The Computer Journal*, vol. 58, no. 1, pp. 1–22, 2015.
[47] Y. Yang, K. Liu, J. Chen, X. Liu, D. Yuan, and H. Jin, "An Algorithm in SwinDeW-C for Scheduling Transaction-Intensive Cost-Constrained Cloud Workflows," in *2008 IEEE Fourth International Conference on eScience*, pp. 374–375, 2008.
[48] M. Kristan and A. Leonardis, "Online discriminative kernel density estimator with gaussian kernels," *IEEE transactions on cybernetics*, vol. 44, no. 3, pp. 355–365, 2013.
[49] J. Poelmans, D. I. Ignatov, S. O. Kuznetsov, and G. Dedene, "Formal concept analysis in knowledge processing: A survey on applications," *Expert systems with applications*, vol. 40, no. 16, pp. 6538–6560, 2013.
[50] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, I. Goiri, S. Krishnan, J. Kulkarni, *et al.*, "Morpheus: Towards automated SLOs for enterprise clusters," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 117–134, 2016.
[51] M. Xu, S. Alamro, T. Lan, and S. Subramaniam, "Cred: Cloud right-sizing with execution deadlines and data locality," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3389–3400, 2017.
[52] M. R. Kaseb, M. H. Khafagy, I. A. Ali, and E. M. Saad, "An improved technique for increasing availability in Big Data replication," *Future Generation Computer Systems*, vol. 91, pp. 493–505, 2019.
[53] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, pp. 9–44, 1988.
[54] S. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI'06)*, pp. 307–320, 2006.
[55] R. R. Noel and P. Lama, "Taming Performance Hotspots in Cloud Storage with Dynamic Load Redistribution," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 42–49, 2017.
[56] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Transactions on Storage (TOS)*, vol. 4, no. 3, pp. 1–23, 2008.
[57] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *NSDI*, 2010.
[58] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging endpoint flexibility in data-intensive clusters," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 231–242, 2013.
[59] K. Liu, J. Wang, Z. Liao, B. Yu, and J. Pan, "Learning-based adaptive data placement for low latency in data center networks," in *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, pp. 142–149, IEEE, 2018.
[60] "TPC-C." https://www.tpc.org/tpcc/.
[61] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.
[62] T. Zhang, A. Gupta, M. A. F. Rodríguez, O. Spjuth, A. Hellander, and S. Toor, "Data management of scientific applications in a reinforcement learning-based hierarchical storage system," *Expert Systems with Applications*, vol. 237, p. 121443, 2024.
[63] V. Ljosa, K. L. Sokolnicki, and A. E. Carpenter, "Annotated high-throughput microscopy image sets for validation.," *Nature methods*, vol. 9, no. 7, pp. 637–637, 2012.
[64] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies," in *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 134–143, 1999.