RIoTstore: Resilient Data Storage for Spatial IoT Applications

Vinayak Gajjewar
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, USA

Rich Wolski
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, USA

Chandra Krintz
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, USA

***Abstract*—In IoT deployments, it is often necessary to replicate data in failure-prone and resource-constrained computing environments to meet the data availability requirements of smart applications. In this paper, we evaluate the impact of correlated failures on an off-the-shelf probabilistic replica placement strategy for IoT systems via trace-driven simulation. We extend this strategy to handle both correlated failures as well as resource scarcity by estimating the amount of storage capacity required to meet data availability requirements. These advancements lay the foundation for building computing systems that are capable of handling the unique challenge of reliable data access in low-resource environments.**

I. Introduction

In this work, we study the problem of replicating immutable data in a deployment of resource-constrained devices that are subject to frequent failures. Such deployments are common in Internet of Things (IoT) settings sited in remote outdoor locations, where grid power is unavailable and wireless network connectivity can fluctuate.

A large body of previous work [1], [2], [3], [4], [5] has considered data replication strategies for IoT deployments, also termed "fog and edge systems." ElfStore [6] is a system for replicating data that is attractive in this problem domain because it makes placement decisions once, before the data is distributed, based on probabilistic assumptions of device availability. For data that is immutable or does not change rapidly, this static approach does not require the network and computational overheads associated with distributed consensus protocols [7], [8] that can be difficult to support using power-constrained devices and intermittent network connectivity.

The ElfStore [6] model assigns a reliability to each device, which is the probability that the device is available at a particular point in time. It then computes the probability that at least one device in a data item's replica set is available at a particular point in time, and uses this to determine which sets of devices to use for replica placement. Critically, however, the ElfStore results (which are analytical) require an IoT deployment to include a class of devices (termed "fog nodes") that do not fail. These devices implement a distributed index of the data blocks. However, if a fog node fails, all of the data storage nodes (termed "edge nodes") become unreachable, thereby introducing correlated failures in which multiple nodes "fail" together. Thus, ElfStore's assumption of statistical independence of failures is invalid.

In this work, we answer the following research questions:

- How does the presence of correlated failures in an IoT deployment affect the availability of data replicated via a probabilistic model?
- How can we determine the amount of additional storage capacity that is required to meet data Service Level Agreements (SLAs) in the face of correlated failures?

To answer these questions, we make the following contributions. We evaluate the efficacy of ElfStore, a probabilistic replication model, via discrete simulation driven by real-world failure traces and data access patterns. In particular, we evaluate replica placements against a real-world failure trace dataset and we use spatial database benchmarking software to generate realistic access patterns. We also propose RIoTstore, a replica placement strategy that accounts for the presence of correlated failures when replicating data on unreliable devices. As part of RIoTstore, we present a heuristic to determine the additional storage capacity required of each edge node to meet data block SLAs in the presence of correlated failures.

II. Background

There exists a body of work ([2], [3], [4], [5]) that studies the problem of replica allocation in failure-prone environments. However, there is a gap in the literature when it comes to replication in resource-scarce regimes. [2] explores replica allocation in the context of mobile ad hoc sensor networks. The authors propose methods to improve data availability when there is a strong correlation between data item requests. [3] introduces the concept of a Probabilistic Shared Risk Group (PSRG), a model for how multiple computing elements can fail simultaneously. The authors use Grid'5000 [9] failure traces and a combination of analytical and Monte Carlo simulation to evaluate service reliability. [4] uses dynamic Bayesian networks trained on historical failure logs to infer the failure dependencies between edge servers. The authors in [5] explore replica placement algorithms that tolerate correlated failures in cloud data centers. They find that the single-block problem has an efficient solution but the complexity of replicating multiple blocks depends on the maximum difference in replication factor.

This prior work attempts to handle correlated failures in data replication, but does not explore the impact of resource constraints, particularly storage capacity constraints that are

common in IoT settings. Additionally, it is an open question as to how to estimate the computing resources required to meet a given level of availability.

ElfStore is a distributed data storage service designed for IoT deployments that are heterogeneous, resource constrained, and failure prone. ElfStore provides high availability and content-based discovery for IoT data by combining the use of a super-peer overlay and differential replication. The super-peer overlay clusters IoT devices into partitions and differential replication accounts for device reliability and available storage capacity in its replica placement decisions.

A partition consists of a parent node (called a fog node) and a set of child nodes (called edge nodes). Each fog manages a non-overlapping subset of edges, coordinates with other fog nodes, and routes client requests to its edges or to other fog nodes. Each edge node stores replicas of data blocks and uses them to serve client requests that have been routed to it by its parent fog.

ElfStore makes a number of assumptions to simplify block placement and discovery across edge devices. First, it assumes that fog nodes never fail and that edge nodes have a predefined failure rate that doesn't change. When a device fails, all data blocks stored on it are inaccessible. Second, devices have a predefined storage capacity. Blocks are placed on devices up to this capacity. Third, data blocks are immutable (precluding the need for replica consistency protocols). Finally, each data block has a known size and reliability target specified as the probability of one or more of the edge nodes hosting a replica of the data block being available at all times. In the ElfStore study, this availability target is chosen, for each data block, at random to be either 0.9, 0.99, 0.999, or 0.999.

Using this set of assumptions, ElfStore determines the number and locations of data block replicas such that devices come from different partitions (if possible), have sufficient storage capacity, and the resulting placement of replicas meets the block's availability target. The replica placement algorithm balances the use of fog nodes with both high and low reliability edges, gives preference to fog nodes responsible for large amounts of storage, and distributes replicas across fog nodes to reduce the impact of correlated failures.

III. Evaluating the Impact of Correlated Failures

We first investigate the impact of correlated failures on the reliability of ElfStore [6] for spatial workloads in IoT deployments. Spatial applications rely on low latency access to location-aware data streams to provide real-time understanding and management of complex, dynamic systems. ElfStore provides some of the features that spatial applications require including low latency access to highly available data sets. Moreover, its data block immutability assumption, which significantly simplifies replication, works well for many spatial applications which operate on map data that changes infrequently or on streams of data that are append-only.

ElfStore has not been evaluated for this application domain where all nodes have a non-vanishing probability of failure. In particular, we study the effect of fog node failure which causes all of the child nodes in a partition to become unreachable. The assumption of 100% availability for all fog nodes is not typically achievable in remote IoT settings where power and network infrastructure can periodically become unstable. Specifically, ElfStore assumes that fog nodes are 100% available and that edge nodes in a deployment have unchanging and independent probabilities of being available, that, together, form a sample from a Normal distribution with mean 0.8 and standard deviation 0.05. As such, the joint probability that all edge nodes within a partition will fail is vanishingly small for even a modest number of nodes. Relaxing the assumption of perfect fog node availability introduces the possibility that all edge nodes within a partition are inaccessible (regardless of whether they have failed individually). In this work, we investigate the effect of fog node failure, and the correlated edge node failure it induces, on the reliability of ElfStore and its storage requirements.

To investigate ElfStore's suitability for replicating spatial data in IoT settings where fog node availability cannot be 100% guaranteed, we have developed a trace-based, discrete-event simulator that is capable of using either real-world or synthetically generated failure event traces to represent fog and edge node behavior. In contrast, the original ElfStore results are purely analytical, relying on standard distributional assumptions of availability and failure independence. In addition, the simulator includes the ability to replay actual query traces. The ElfStore work assumes a purely random access pattern in the query stream. ElfStore also reports reliability as the joint probability of replica availabilities. The simulator uses the joint probability to determine replica placement (see Equation 1) but it measures reliability as a fraction of successful replica accesses resulting from replay of a workload trace.

Note that ElfStore includes an efficient gossip protocol for fog nodes to maintain a consistent view of the replica index when new data blocks are added to the collection of data blocks. In this work, we presuppose that such a protocol exists for this purpose and we do not include its function in the RIoTSTORE simulations. That is, each simulation is for a query stream against a fixed set of data blocks whose replica index has already been distributed across fog nodes.

A. Trace-Driven IoT Deployment Simulation

The original ElfStore results are analytical. That is, they show that the placement algorithm (described in Section II) is able to maintain a minimum replica availability probability for all replicated data blocks and sufficient storage. We use the term "Service Level Agreement" or SLA to refer to the collection of availability probabilities that must be maintained for all data blocks. Specifically, the SLA specifies that the fraction of accesses for any data block, when repeatedly requested by a set of queries, will be greater than or equal to a minimum target fraction for the block. ElfStore assigns each block a target fraction of 0.9, 0.99, 0.999, or 0.9999, chosen at random.

In order to determine whether a particular replica placement meets data SLAs, we have developed a trace-based simulator that is extensible in three ways. First, the RIOTSTORE simulator can assign individual edge node availability probabilities using different methods. Recall that ElfStore draws this probability from a Normal distribution with mean 0.8 and standard deviation 0.05. The simulator supports this method but also one that computes availability probabilities from failure event traces taken from real machines. Secondly, the workload model investigated by ElfStore is one in which each data block is equally likely to be queried (i.e. block accesses are chosen randomly). The RIOTSTORE simulator is able to support this workload model but also one based on real query traces that specify a sequence of accesses to specific blocks. Finally, the simulator is modularized so that it can use different replica placement algorithms. We implement both ElfStore's placement algorithm and RIOTSTORE , which accounts for the possibility of correlated node failures. The modules that control node availability, query traces, and replica placements can be changed independently, allowing for a thorough evaluation of replica placement strategies.

Regardless of how the simulator modules are configured, the RIOTSTORE simulator implements the ElfStore SLA by assigning a minimum availability threshold to each data block. In this work, to match the original ElfStore results, we assign each data block a minimum availability of 0.9, 0.99, 0.999, or 0.9999 (chosen at random) representing the minimum probability that the data block will be available at the time of a query. The simulator then uses whatever placement algorithm has been configured to replicate the data blocks across edge nodes subject to the constraint that a replica of a data block will be accessible with at least its minimum availability specification in the SLA.

Our simulator decomposes a given workload trace (consisting of a set of queries) into a sequence of individual data block requests. A request for a data block is deemed to be successful if at least one edge node that stores a replica of it is available at the time that the access was made. For an edge node to be considered available at a particular time, both it and its parent fog node must be available at that time. After a workload trace terminates, we can determine per block whether that block's SLA was met by dividing the number of successful accesses for that block by the number of total accesses for that block. Therefore, we generate workload traces consisting of 100000 queries for our simulation to ensure, with high probability, that each data block is accessed enough times to accurately compute its availability.

To assign availability probabilities to each edge node, the simulator first assigns a failure event trace, represented as a time series of failure events, to each node. It then computes the node's availability as the fraction of uptime in the trace divided by the total length of the trace. For example, if a failure trace is assigned to an edge node that spans 24 hours, with 21 hours of uptime and 3 hour to total downtime, that edge node will be assigned an availability of $21 \div 24 = 0.875$.

Note that the ElfStore placement algorithm requires that the availability probabilities for each edge node and the availability threshold for each data block be determined before the algorithm begins. The simulator uses the availability probabilities computed from the failure traces assigned to each edge node to compute the joint availability probability for each data block during placement.

To introduce the effect of fog node failure, our simulation also assigns a failure trace to each fog node and computes from it the fraction of time the node is "up" to use as its availability probability. If the availability probability for a for a fog node is a_f and the availability probability for an edge node is a_e then the availability probability A for a replicated data block is

$$A = 1 - \prod_{\text{fog nodes}} [1 - (a_f \times (1 - \prod_{\text{edge nodes}} (1 - a_e)))] \quad (1)$$

where the a_e probabilities are for the edge nodes where the block is replicated, and the a_f probabilities are for the fog nodes serving the partitions that include *all* of the edge nodes replicating the data. That is, the joint failure probability associated with fetching a replica within a partition is $1 - \prod_{\text{edge nodes}}(1 - a_e))$ where the a_e probabilities are for the edge nodes replicating the data within the partition. Multiplying this probability by the availability probability of the fog node for the partition, a_f, gives the joint probability for successfully fetching a replica from the partition and thus subtracting this product from 1 yields the failure probability for the partition. The joint probability of failing to fetch the replica from any partition in which it is stored is the product of the partition failure probabilities from all of the partitions in which it is stored. Subtracting this probability from 1 yields the probability A that at least one replica will be available. Note that Equation 1 captures the original ElfStore measure of replica availability when all values if a_f as set to 1.0, and each replica is assigned the value of a_e from the node to which it has been assigned by the replica placement algorithm.

B. Real-World Failure Traces

We also evaluate RIOTSTORE using failure traces from a 1050-node Raspberry Pi cluster located at a research university. Each node of this cluster shares a USB power distribution unit with 41 other nodes connected via a single network switch. There are 7 switches within the cluster. Node failures from this system are recorded as true node failures (the node is unresponsive), switch failures (causing all 42 to record failure), and power supply undervoltage (which causes a node to throttle its CPU frequency to a minimum value). Note that undervoltage periods in this cluster vary quite substantially across USB power supplies and individual nodes.

In this study, we consider only undervoltage failures as they are, by far, the most frequent and also the most indicative of IoT behavior in remote settings. Often, when IoT devices (such as Raspberry Pi single board computers [10]) are deployed outdoors, where there is no reliable network or power infrastructure, Lithium Ion batteries recharged by PV solar

panels are also deployed to power them. Undervoltage, in these settings, is a common occurrence and one which (to prevent device damage) necessitates that the device remain idle. We consider a node from the cluster to be unavailable with respect to data block replica access if it is in an undervolt state. We use the `vcgencmd` command-line tool to determine if a node is currently in this undervolt state.

There are 760 unique failure traces in this dataset, which span the time period from Wednesday, June 4, 2025 to Thursday, June 5, 2025. Because not every single Pi experienced unavailability due to undervoltage during this time period, we sample traces from the 20 Pis with the least uptime. For the remainder of this paper, we will refer to this source of failure traces as "Pi1050".

C. Spatial Application Traces

To drive our simulation with realistic data access patterns, we generate query traces using the Jackpine spatial database benchmark [11]. Jackpine models how end users and applications interact with spatial data, modeling spatial joins, analysis functions, and queries based on real-world spatial applications.

We use the US Census Bureau's TIGER [12] dataset for the state of Texas as a dataset against which Jackpine generates query traces. This dataset contains $13,440$ places of interest (POIs), each one represented by a point with a given latitude and longitude, as well as some additional metadata. These POIs are the fundamental unit of replication, meaning that each one is stored as a 10 MB data block and replicated across nodes. This size matches the one given in the ElfStore paper.

D. RIoTStore Replica Placement and Storage Sizing

We next consider the problem of determining the per-device storage capacity required to meet the SLAs of a dataset replicated on unreliable devices. For RIoTStore, we define a replica placement strategy that both automatically determines storage capacity requirements and determines a placement of replicas that will meet a set of SLA targets.

The heuristic orders fog nodes by decreasing a_f from Equation 1, and for each fog node, it orders its edge nodes by the decreasing product of a_f and a_e. It then orders data blocks in terms of SLA target, from most reliable to least, so that the data blocks that require the most reliability are prioritized over those with weaker targets. The heuristic generates an ordered list of fog nodes where each element contains a list of edge nodes, annotated by available storage capacity and availability set to $a_f \times a_e$. It also generates an ordered list of data blocks sort by SLA target.

When a traversal completes and all blocks from the block list have been replicated so that their A values meet the SLA requirement for each, the algorithm terminates. Upon termination, the algorithm reports the storage required on each edge node and the replica mapping to that storage that meets the SLA.

As with ElfStore, RIoTStore determines replica placements once, before receiving a stream of queries. Determining a placement strategy that can handle a dynamically changing population of data blocks is the subject of our ongoing research.

Note that we explicitly disperse replicas of a data block across fog partitions because placing replicas of a block on edges with the same parent fog may not be able to satisfy that block's availability requirement. This case is handled implicitly by equation 1, which implies that if a data block is replicated on only one fog partition, its availability is constrained by the availability of that fog node. The algorithm thereby dispurses the replicas of a data block across fog nodes, if possible. The pseudocode for the algorithm is shown below.

```
1  sort Blocks by SLA in descending order
2  sort Fogs by reliability in descending order
3  GroupedEdges := empty list of lists
4  for each Fog in Fogs
5      sort Fog.edges by reliability in descending
           order
6      append Fog.edges to GroupedEdges
7  Edges := concatenate all lists in GroupedEdges in
       order
8  StoragePerEdge := map from Edge -> integer,
       default 0
9  BlockReplicas := map from Block -> list of Edges,
       default {}
10 EdgeIndex := 0
11 for each Block in Blocks:
12     StartIndex := EdgeIndex
13     while (availability(BlockReplicas[Block]) <
           Block.SLA:
14         CurrEdge := Edges[EdgeIndex]
15         EdgeIndex := (EdgeIndex + 1) mod Edges.
               size
16         if EdgeIndex == StartIndex: break
17         append CurrEdge to BlockReplicas[Block]
18         StoragePerEdge[CurrEdge]++
19 for each [Edge, Capacity] in StoragePerEdge:
20     print Capacity
21 return BlockReplicas
```

IV. Evaluation

To evaluate the efficacy of this rightsizing heuristic and replica placement strategy, we simulate a set of query traces. We present exceedence curves for the resulting data block availabilities. We comput availability of a data block by dividing the number of successful requests (i.e., the number of requests that occur when at least one replica is up) by the number of total requests for that block, to produce a fraction between 0 and 1. The y-axis of the exceedence curve shows the cumulative fraction of data blocks with a simulated availability greater than or equal to a given value on the x-axis. In other words, the height of the exceedence curve at a given availability level is the fraction of data blocks that meets that availability threshold.

To ensure a fair comparison with ElfStore, we use the same deployment configuration described in the original work. Specifically, each simulation describes a deployment of 272 nodes in which 16 fog nodes are each responsible for 16 edge nodes. We also use the same SLA targets as the ElfStore paper, with each data block being randomly assigned to one of four availability thresholds: 0.9, 0.99, 0.999, and 0.9999. A particular block's SLA is satisfied if its computed availability

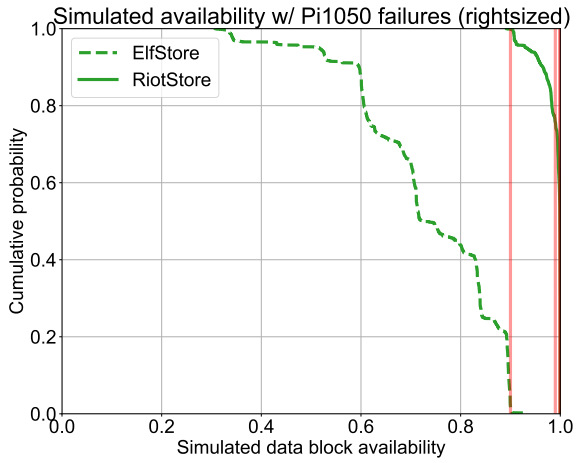

Fig. 1. Exceedance curves of simulated data block availability generated by unreliable fog nodes. Dotted lines are for ElfStore. Solid lines are for RIOTSTORE . The results are for Jackpine-generated workloads, the Pi1050 failure source, and the RIOTSTORE replica placement method described in Subsection III-D.

is greater than or equal to the threshold assigned to it. With this set of SLAs, we expect that, after enough queries have been made, 100% of data blocks meet the 0.9 SLA, 75% of blocks meet the 0.99 SLA, 50% of blocks meet the 0.999 SLA, and 25% of blocks meet the 0.9999 SLA.

A. RIOTSTORE *Replica Placement*

Figure 1 shows the exceedence curves for this experiment. The graph shows that the RIOTSTORE replica placement strategy is able to meet the SLA targets by determining what storage requirements are necessary for each edge node based on Equation 1 (cf. Subsection III-D). RIOTSTORE meets the SLA targets by setting extending each edge node's capacity to support 9666 replicas (from the 1600 limit used in the original ElfStore work). The graph also shows that ElfStore is unable to take advantage of the increased storage capacity and fails to meet the SLA for the Pi1050 real-world error trace.

To summarize, the contributions of RIOTSTORE are twofold. First, we use a joint probability calculation (Equation 1) to determine the availability of a data block in a way that accounts for failure-prone fog nodes. Second, we present a replica placement strategy that simultaneously determines both the storage capacity needed by each edge node as well as a replica placement that meets a given set of SLA targets. Furthermore, we use discrete-even simulation to validate the efficacy of our proposed solution.

V. CONCLUSION

In this paper, we use discrete event simulation driven by real-world failure traces to show that meeting data availability requirements requires a replica placement strategy that considers both correlated and individual IoT node failures. We present such a strategy (called RIOTSTORE) that extends the ElfStore IoT data storage system to account for potential fog node failures. RIOTSTORE simultaneously determines the storage capacity needed by each edge node in a fog partition to meet a set of SLA targets and a replica placement that meets those targets. These advancements deepen our understanding of how to ensure reliability in failure-prone and resource-scarce settings, and lay the foundation for software infrastructure capable of handling this challenge.

REFERENCES

[1] M. I. Naas, L. Lemarchand, P. Raipin, and J. Boukhobza, "Iot data replication and consistency management in fog computing," *Journal of Grid Computing*, vol. 19, no. 3, pp. 1–25, 2021.

[2] T. Hara, N. Murakami, and S. Nishio, "Replica allocation for correlated data items in ad hoc sensor networks," *ACM Sigmod Record*, vol. 33, no. 1, pp. 38–43, 2004.

[3] J. E. Pezoa and M. M. Hayat, "Reliability of heterogeneous distributed computing systems in the presence of correlated failures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 4, 2013.

[4] A. Aral and I. Brandic, "Dependency mining for service resilience at the edge," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 228–242.

[5] K. A. Mills, R. Chandrasekaran, and N. Mittal, "Algorithms for optimal replica placement under correlated failure in hierarchical failure domains," *Theoretical Computer Science*, vol. 809, pp. 482–518, 2020.

[6] S. K. Monga, S. K. Ramachandra, and Y. Simmhan, "Elfstore: A resilient data storage service for federated edge and fog resources," in *IEEE International conference on web services*, 2019.

[7] D. Ongaro and J. Ousterhout, "The raft consensus algorithm," *Lecture Notes CS*, vol. 190, p. 2022, 2015.

[8] R. Van Renesse and D. Altinbuken, "Paxos made moderately complex," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, pp. 1–36, 2015.

[9] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab *et al.*, "Grid'5000: a large scale and highly reconfigurable experimental grid testbed," *Journal of High Performance Computing Applications*, vol. 20, no. 4, 2006.

[10] O. Nath, "Review on raspberry pi 3b+ and its scope," *Int. J. Eng. Appl. Sci. Technol*, vol. 4, no. 9, pp. 157–159, 2020.

[11] S. Ray, B. Simion, and A. D. Brown, "Jackpine: A benchmark to evaluate spatial database performance," in *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 2011, pp. 1139–1150.

[12] R. W. Marx, "The tiger system: automating the geographic structure of the united states census," *Government publications review*, vol. 13, no. 2, pp. 181–201, 1986.