ElasticCopyset: An Elastic Replica Placement Scheme for High Durability

Han Li Srikumar Venugopal

School of Computer Science and Engineering,
University of New South Wales, Australia
{hli,srikumarv}@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-201402
February 2014

Abstract

Distributed key-value stores (KVSs) are a standard component for data management for applications in Infrastructure-as-a-Service (IaaS) clouds. Replica placement schemes for KVSs on IaaS have to be adapted to on-demand node addition and removal, as well as, to handle correlated failures of the physical hardware underlying the cloud. Currently, while placement strategies exist for handling correlated failures, they tend to rely on static mapping of data to nodes, which is inefficient for an elastic system. This paper presents ElasticCopyset, a novel replica placement scheme, which fills in the gap of providing efficient elasticity while maintaining high data durability when multiple nodes simultaneously fail. We experimentally demonstrate that ElasticCopyset maintains a close to minimised probability of data loss at correlated failures under different scenarios, and exhibits better scalability and elasticity than state-of-the-art replica placement schemes.

1 Introduction

Distributed key-value stores (KVSs) [3, 9, 11] have become a standard component for data management for applications deployed on Infrastructure-as-a-Service (IaaS) clouds. Data in KVSs is organised in storage units, called variously as chunks, tablets, partitions or virtual nodes, that are replicated for improving fault tolerance and performance. A fundamental design choice in KVSs is the placement scheme for storing replicas across distributed nodes. We use the term *copyset*, derived from Cidon, et al. [7], to describe the set of nodes on which a particular storage unit is replicated.

IaaS clouds are built from commodity components. At large scales, hardware failure becomes the norm rather than an exception [28], causing multiple virtual instances, that form the nodes of a KVS, to fail concurrently. As documented in practice [15, 2, 26], data loss normally occurs in correlated failures, in which a non-negligible percentage of nodes (0.5%-1%) cannot be restored even after the failure is recovered [7]. Hence, the placement scheme must minimise the probability of data loss when multiple nodes simultaneously fail.

The defining characteristic of IaaS is *resource elasticity*, wherein extra resources, such as compute instances, can be acquired on-demand to deal with increasing workload, and can be dismissed later to save on operational costs. KVSs benefit from elasticity when node addition and removal are as efficient as possible. This means that a placement scheme should be able to minimise the impact of adding and removing a node to and from the KVS by requiring minimum changes in the mapping between data storage units and nodes.

However, current replica placement strategies do not simultaneously satisfy efficient elasticity, while maintaining high data durability in correlated failures. Random replication is employed by popular distributed storage systems [11, 26, 18]. It uses hash functions to disseminate each data storage unit across a set of randomly selected nodes. The random nature of this strategy provides efficient scalability and load balancing, but also creates a large number of copysets. As depicted in Figure 1, given the number of nodes that fail, a larger number of copysets gives higher probability of data loss.

Alternatively, Cidon, et al. [7] proposed the copyset-based replication strategy. The nodes are split into a number of copysets independently from the data, while the replicas of a data storage unit are required to be stored on the same copyset. This strategy builds on the insight that minimising the number of copysets can lead to minimised probability of data loss when multiple nodes fail simultaneously. However, current-state implementations [7, 8] focus on creating copysets for static system deployment, but handle dynamic node addition and removal inefficiently.

This paper presents ElasticCopyset, a novel replica placement scheme that provides efficient elasticity while guaranteeing high data durability. In ElasticCopyset, nodes are split into groups, and a minimum number of copysets are formed within each group. The contribution of this work is three-fold. First, a novel shuffle algorithm is designed to generate minimum non-overlapping copysets within each group, such that the probability of data loss is minimised when multiple nodes simultaneously fail. Second, a novel replacement algorithm is used to deal with node addition and removal efficiently. Third, a theorem is proposed to determine the number of nodes in a group, with a proof that shows the correctness of the algorithm. Through evaluation, we demonstrate that Elas-

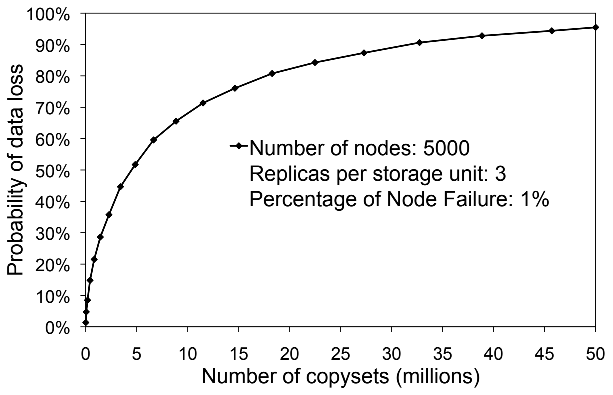


Figure 1.1: Probability of data loss versus number of copysets. Given the number of copysets required at each depicted point, distinct copysets were generated, based on permutation described in Copyset Replication [7]. The probability of data loss was calculated from simulating 10000 occurrences of correlated failure, as depicted in Section 5.

ticCopyset maintains a close to minimised probability of data loss at correlated failures under various scenarios, and exhibits better scalability and elasticity than state-of-the-art placement schemes.

The rest of this paper is structured as follows. In the next section, we review the state-of-the-art data placement strategies in data durability and scalability. In Section 3, we provide mathematical definitions for the durability problem. We present the design of ElasticCopyset in Section 4. The experimental evaluations are presented in Section 5. Finally, we present our conclusions in Section 6.

2 Related Work

Replica placement and its impact on data durability has been studied extensively in the past. Peer-to-peer system evaluations [29, 24] have considered replication [19, 10, 14] versus coding techniques [22] to achieve high data durability, and concluded that replication provides better robustness to survive the high rate of failures in distributed infrastructures. In the context of DHT systems, Glacier [16] uses massive replication to deal with large-scale correlated failure events. In contrast, Carbonite [6] creates and keeps track of additional replicas to handle small-scale failure events at a low cost, which is similar to the failure scenario studied in this paper.

The relation between data durability and the number of possible replicas has also been discussed previously [25, 27]. The trade-off here is between reducing the probability of losing a data item during a simultaneous failure (by limiting the number of replicas) and improving the robustness to tolerate a higher average failure rate (by increasing the number of replicas). However, none of these have discussed the trade-offs between data durability and scalability.

Cidon,et al. [7] introduce the concept of *scatter width*, and discuss its relationship to data durability and scalability performance. Scatter width is defined as the number of nodes that store copies for each nodes data. As discussed [7], using a high scatter width means that each node shares data with more nodes. Thus, it improves the performance to incorporate or recover a node, but creates more opportunities of data loss under simultaneous failure of nodes. In contrast, using a low scatter width limits the number of nodes that each node can share data with. It reduces the probability that all the replicas of some data are lost when multiple nodes fail, with the trade-off of slow recovery time from independent node failures.

This paper borrows the notions of scatter width and copyset from Cidon,et al. [7, 8]. As the copyset is a set of nodes that store the replicas for a data storage unit, the failure of a copyset is equivalent to the loss of that unit. The copyset-based replication schemes (including this paper) focus on minimising the number of copysets in the system to provide high data durability. However, existing implementations of copyset-based replication [7, 8] rely on random permutation to create copysets. For each node addition, new copysets are formed without altering any existing copyset, while no affected copysets are dismissed at each node removal. Instead, an existing node is randomly selected to replace the removed node. This approach ends up in increasing number of copysets if nodes join or leave dynamically. In contrast, node addition and removal are efficiently dealt with in this paper.

This paper proposes to split the nodes into groups before forming copysets. Yu, et. al [30] proposed Group DHT to constrain the placement of replicas in a group of nodes, but this is similar to the copyset idea [7]. In contrast to more structured placement such as group and copyset, popular key value stores [11, 12, 18] prefer randomised replication for the purpose of efficient load balancing. However, Carbonite [6] has identified that randomly replicating data across a large set of nodes increases data loss probability under simultaneous failures. Cidon,et al. [7, 8] have demonstrated that random replication is nearly guaranteed to cause a data loss event once the size of the system scales beyond hundreds of nodes.

There are other research efforts on data durability. Large companies [17, 13] consider geo-replication as an effective technique to prevent data loss under large scale concurrent node failures. However, this approach is not feasible for storage providers that operate within one region. Disaster recovery systems [4, 23] use replication and mirroring to increase durability. These systems focus on the cost of propagating updates on mutable data, which is not applicable to the immutable data in key-value stores [11, 18].

3 Problem Definition

3.1 Parameter Definitions

We focus on the issue of data placement in distributed KVSs across a cluster of N data nodes. Typically, the data is horizontally partitioned, and stored as consolidated replicas. In this paper, we define the *storage unit* as the basic unit for replication, which can refer to a key-value pair, a tablet or a data chunk in different storage systems. Each storage unit is stored on a set of distinct data nodes, which is called the *copyset* for this storage unit. The notational conventions used in this paper are summarised in Table 3.1.

The number of distinct nodes in the copyset is defined by the *replication*

Table 3.1: Notational Conventions

Notation	Description
R	The replication number, i.e., #replicas of each storage unit
S	The scatter width
P_{ran}	The probability of data loss using random replication
P_{cs}	The probability of data loss using copyset replication
N	The number of nodes in the system
F	The number of nodes failed in a correlated failure
N_G	The number of nodes in each group
N_E	The number of extra nodes that cannot form a complete group
N_{su}	The number of storage units in the database
N_{cs}	The number of copysets in the system
C	The number of columns in the shuffle matrix
L	The number of rows in the shuffle matrix
a, b, c	The array of nodes to shuffle
m_a, m_b, m_c	The matrix of nodes after shuffle
x, y	The x, y coordinate of the shuffle matrix
i, j, k, n	Non-negative integers
r, s, t	Integers

number R. To reduce complexity, we use the same replication number for all storage units in the system. Hence, every storage unit is stored by R nodes in one copyset.

The number of nodes that store copies for each node's data is defined as the scatter width S [7]. Thus, each node should share copysets with S other distinct nodes. Since each node has $R-1$ other nodes in each copyset, then each node should be assigned to at least $\frac{S}{R-1}$ copysets. Hence, $S \geq (R-1)$. The scatter width is determined by the system administrators, and is usually a multiple of $R-1$. Theoretically, for a cluster of N nodes, the minimum number of copysets is given by Equation 3.1, which is divided by R because each copyset is counted R times. Thus, adding (or removing) a node requires $\frac{S}{(R-1)R}$ copysets to be created (or dismissed).

$$MinCopysets(N) = \frac{S}{R-1}\frac{N}{R} \tag{3.1}$$

The challenge to achieve the minimum copysets lies on guaranteeing that any two copysets overlap by at most one node. For any given node n_i, there are $\frac{S}{R-1}$ copysets that share n_i. Amongst these copysets, if there exists two copysets with another common node n_j, then n_i has only $S-1$ other distinct nodes to share its data. In this case, one more copyset is created for n_i. Then, the number of copysets will exceed the minimum number.

3.2 Probability of Data Loss

A correlated failure occurs when F nodes fail simultaneously, where $F > R$. In random replication, each storage unit is randomly assigned to a set of R nodes. The probability of losing at least one storage unit in this scenario [8] is given by Equation 3.2, wherein N_{su} is the number of storage units. $\binom{N}{R}$ and $\binom{F}{R}$ denote the number of ways of picking R nodes unordered out of N and F nodes, respectively.

$$P_{ran} = 1 - \left(1 - \frac{\binom{F}{R}}{\binom{N}{R}}\right)^{N_{su}} \tag{3.2}$$

In contrast, in a copyset-based scheme, the copysets are generated independently from the data. Given N_{cs} copysets, the probability of losing a single copyset is $\frac{\binom{F}{R}}{\binom{N}{R}} \frac{N_{cs}}{\binom{N}{R}}$. Hence, the probability of incurring any data loss (i.e., losing at least one copyset of nodes) is given by Equation 3.3.

$$P_{cs} = 1 - \left(1 - \frac{\binom{F}{R}}{\binom{N}{R}} \frac{N_{cs}}{\binom{N}{R}}\right)^{N_{cs}} \tag{3.3}$$

Each copyset of nodes hosts multiple storage units. Thus, the number of copysets is less than the number of storage units, i.e., $N_{cs} < N_{su}$. Two conclusions can be deducted from Equation 3.3. Firstly, since $\frac{N_{cs}}{\binom{N}{R}} < 1$ and $N_{cs} < N_{su}$, then the copyset-based scheme gives a smaller probability of data loss, i.e., $P_{cs} < P_{ran}$. Secondly, given the same R, N, and F, then P_{cs} is an increasing function of N_{cs}. It means reducing the number of copysets can lead to a decrease in the probability of data loss. This conclusion is consistent with Figure 1.1.

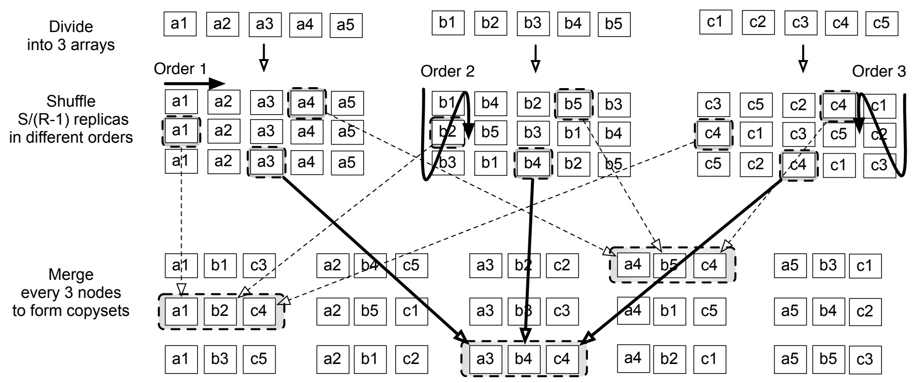
Hence, we focus on designing a copyset-based scheme that minimises the number of copysets. Moreover, the elasticity characteristic of IaaS cloud means that nodes are dynamically added or removed from the system. The minimum number of copysets should be maintained under dynamic node changes.

4 Design of ElasticCopyset

This section describes the design of ElasticCopyset, a replication scheme that efficiently creates and dismisses copysets as the nodes are added or removed dynamically, such that the number of copysets is close to minimum (Equation 3.1) for higher data durability against correlated failures.

There are properties that form the basis for an elastic, minimal copyset scheme. Firstly, given R and S defined in Section 3, each copyset contains R nodes. Each node belongs to $\frac{S}{R-1}$ copysets, in which there are S other distinct nodes. Secondly, any two copysets overlap by at most one node, such that the total number of copysets can be minimised. Lastly, the addition or removal of a node affects only S other nodes in the $\frac{S}{R-1}$ copysets it belongs to.

To achieve these properties, we propose that the nodes should be split into groups, and that each node forms copysets only with nodes in the same group. There are advantages of isolating nodes into groups. First, as the number of nodes changes, the number of copysets can be changed by forming or dismissing a group of nodes accordingly. Second, the effect of adding or removing a node is restricted to one group, rather than the entire system.



Divide into 3 arrays
Order 1
Order 2
Order 3
Shuffle S/(R-1) replicas in different orders
Merge every 3 nodes to form copysets

Figure 4.1: Create a minimum group of non-overlapping copysets by shuffling. In this example, there are $N_G = 15$ nodes, divided into 3 arrays: $\{a1, a2, a3, a4, a5\}$, $\{b1, b2, b3, b4, b5\}$ and $\{c1, c2, c3, c4, c5\}$. The scatter width $S = 6$. Hence, $C = 5$, $L = 3$. The three arrays are replicated into three 3×5 matrices.

In ElasticCopyset, each group has the same number of nodes, denoted as N_G, which is pre-defined using a theorem described in Subsection 4.2. Every N_G nodes forms one complete group, wherein the minimum number of *non-overlapping* copysets (i.e., share at most one node) are generated using a shuffle algorithm presented in Subsection 4.1.

However, the number of nodes N is usually not a multiple of N_G. There are extra nodes, $N_E = (N \mod N_G)$, wherein $0 \leq N_E < N_G$. The N_E nodes are insufficient to form a new group. When $N_E > 0$, ElasticCopyset randomly selects $(N_G - N_E)$ other nodes that are already in a complete group, to form one extra group of N_G nodes. This group is called an *incomplete group*, as there are nodes selected from other complete copyset groups. ElasticCopyset uses the same shuffle algorithm to generate copysets within incomplete groups. Subsection 4.3 describes a replacement algorithm that handles node addition and removal in these groups.

4.1 Generating Copysets in Groups

ElasticCopyset generates copysets with a shuffle algorithm that uses three distinct shuffle orders. It requires that the replication number $R = 3$, which is the default value for many distributed storage systems [3, 18, 21, 26]. Cidon et. al [7] had evaluated the effect of varying replication number against the data durability and system performance and report that "Increasing $R = 3$ to 4 does not provide sufficient durability, while using $R = 5$ or more significantly hurts the systems performance and almost doubles the cost of storage". Hence, we use $R = 3$.

Figure 4.1 illustrates the shuffle algorithm of generating copysets for a group of nodes. It consists of three steps: divide, shuffle and merge. The number of nodes in each group N_G is a multiple of R, such that the nodes are equally divided into $R = 3$ arrays, denoted as a, b, and c, each containing $C = \frac{N_G}{R}$ nodes (and columns). The i^{th} node of the array is denoted as $a[i]$, $b[i]$, and $c[i]$, respectively, where $0 \leq i < C$.

$$\begin{cases} C = \frac{N_G}{R} \\ L = \frac{S}{R-1} \end{cases} \tag{4.1}$$

In the *shuffle* phase, each node is replicated $L = \frac{S}{R-1}$ times, such that each array is converted into an $L \times C$ matrix, as defined in Equation 4.1. The three matrices are denoted as m_a, m_b, and m_c. The node located at the x^{th} row, y^{th} column is denoted as $m[x][y]$, wherein $0 \leq x < L$ and $0 \leq y < C$. As depicted in Figure 4.1, the arrays $a[i], b[i]$, and $c[i]$ are replicated into the corresponding matrices in Order 1, 2, and 3 respectively. The orders are described as follows. For each node, k increases from 0 to $L - 1$ inclusively.

Order 1. *Left-to-right, then up-to-down. That is, the k^{th} replica of $a[i]$ is placed at $m_a[k][i]$.*

Order 2. *Up-to-down, then left-to-right. That is, the k^{th} replica of $b[i]$ is placed at $m_b[x][y]$, wherein*
$x = (i + k * C) \mod L$*, and* $y = \lfloor \frac{i+k*C}{L} \rfloor$.

Order 3. *Up-to-down, then right-to-left. That is, the k^{th} replica of $c[i]$ is placed at $m_c[x][y]$, wherein*
$x = (i + k * C) \mod L$*, and* $y = C - 1 - \lfloor \frac{i+k*C}{L} \rfloor$.

In the *merge* phase, the three small matrices created via *shuffle* are merged into one larger $L \times (3C)$ matrix. Given each pair of (x, y), wherein $0 \leq x < L$ and $0 \leq y < C$, every three nodes $(m_a[x][y], m_b[x][y], m_c[x][y])$ form a copyset. For example, in Figure 4.1, $(a1, b2, c4)$, $(a3, b4, c4)$ and $(a4, b5, c4)$ form three different copysets. Hence, the large matrix generates $L * C$ copysets. By comparing Equation 3.1 and 4.2, it can be deducted that the algorithm generates the minimum number of copysets for N_G nodes.

$$L * C = \frac{S}{R-1} \frac{N_G}{R} \tag{4.2}$$

Hence, given N_G for a cluster of N nodes, there are $\lfloor \frac{N}{N_G} \rfloor$ complete groups. One extra incomplete group will be created, if $N_E = (N \mod N_G) > 0$. Thus, there are $\lceil \frac{N}{N_G} \rceil$ groups, each containing $\frac{S}{R-1} \frac{N_G}{R}$ copysets. Therefore, the total number of copysets generated by ElasticCopyset $TotalCopysets(N, N_G)$, is given by Equation 4.3 as a function of N and N_G. Compared to Equation 3.1, ElasticCopyset generates more copysets than minimum. The difference is given by Equation 4.4, which is linear with S and N_G.

$$TotalCopysets(N, N_G) = \frac{S}{R-1} \frac{N_G}{R} \lceil \frac{N}{N_G} \rceil \tag{4.3}$$

$$ExtraCopysets(N, N_G) = \frac{S}{R-1} \frac{N_G - N_E}{R} \tag{4.4}$$

4.2 The Minimum Group with Non-overlapping Copysets

The key design of ElasticCopyset is to determine the minimum number of nodes required to form a group. As shown in Equation 4.4, using a smaller N_G can produce close to the minimum number of copysets required, which reduces the probability of incurring data loss under simultaneous failures.

However, it is also required that each node share copysets with S other distinct nodes. ElasticCopyset shuffles each node into $\frac{S}{R-1}$ copysets, in which there are exactly S nodes other than this node. Yet, it is uncertain that these S nodes are mutually distinct. Hence, our task is to find out the minimum value of N_G, such that there are exactly S other distinct nodes in the $\frac{S}{R-1}$ copysets that each node belongs to.

Theorem 1. *Given the shuffle algorithm and a group of $N_G = R * C$ nodes. When C is the smallest odd number that is greater than L, wherein $L = \frac{S}{R-1}$, any two copysets in the group overlap with each other by at most one node.*

In the following, we will prove Theorem 1 by deduction. Note that given each pair of (x, y), wherein $0 \le x < L$ and $0 \le y < C$, every three nodes $(m_a[x][y], m_b[x][y], m_c[x][y])$ form a copyset.

Requirements for Column-wise Shuffle

As we think of the shuffle algorithm in column-wise, each node in array $a[i]$, which is shuffled in Order 1, always and only appears in the i^{th} column of the resulted matrix m_a. In contrast, the node localities for the 2^{nd} array $b[i]$ and the 3^{rd} array $c[i]$ depend on C, which is constrained by Lemma 1, proved in Appendix A.1.

Lemma 1. *When $C \ge L$, any $b[i]$, after shuffled by L times in Order 2 (or $c[i]$ in Order 3), will appear once at most in any column of the resulted matrix.*

Since each $a[i]$ appears in only one column, Lemma 2 can be deducted from Lemma 1, as proved in Appendix A.2.

Lemma 2. *When $C \ge L$, given $0 \le i, j, k < C$, any combination of $(a[i], b[j])$ or $(a[i], c[k])$ appears in at most one copyset.*

Requirements for Row-wise Shuffle

As we consider the shuffle algorithm in row-wise, each $a[i]$ appears in each and every row of m_a, while the localities of $b[i]$ and $c[i]$ still depend on C. Similarly, if each $b[i]$ (or $c[i]$) is also placed in each and every row of the resulted matrix m_b (or m_c), it will simplify the operation of restricting any two nodes from being allocated together more than once. According to Lemma 3, C and L should be co-prime (i.e. mutually prime). The proof is given in Appendix A.3.

Lemma 3. *When $C \ge L$, if C and L are co-prime, every node, after replicated by L times (either in Order 1, 2 or 3), will appear once in each and every row of the resulted matrix.*

The Minimum C for Non-overlapping Copysets

According to Lemma 3, the minimum C for ElasticCopysets is one of L's co-prime numbers that are greater than L. One candidate of C is the smallest odd number greater than L. We have proved Lemma 4 in Appendix A.4, given two possible values of L: i) L is even; ii) L is odd. In both circumstances, there does not exist a pair of $(b[i], c[j])$ that appears in more than one copyset.

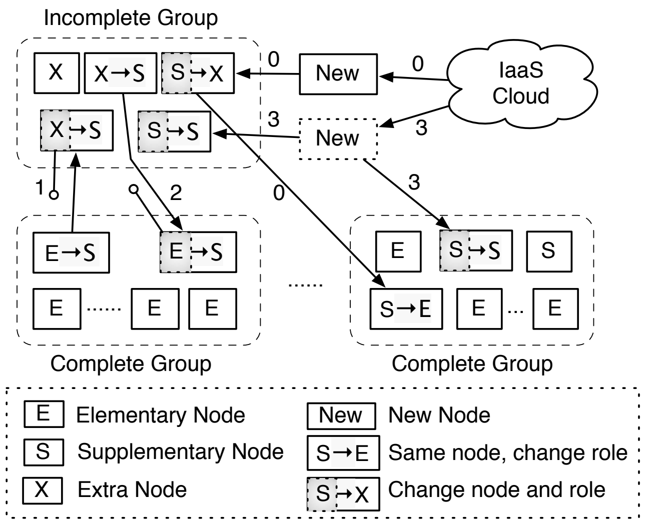


Figure 4.2: Adding and removing nodes in ElasticCopyset.

Lemma 4. *When C is the smallest odd number greater than L, given $0 \leq i, j < C$, any combination of $(b[i], c[j])$ appears in at most one copyset.*

Hence, based on Lemma 2 and 4, the value of C for the minimum group with non-overlapping copysets is, the smallest odd number that is greater than L (Lemma 5, proved in Appendix A.5). Given such C, each node shares copysets with exactly S other distinct nodes in each group, as depicted in Lemma 6, proved in Appendix A.6.

Lemma 5. *When C is the smallest odd number greater than L, any two copysets share at most one common node.*

Lemma 6. *When C is the smallest odd number greater than L, every node belongs to exactly L copysets, in which there are exactly S other distinct nodes.*

4.3 Handling Node Addition and Removal

In a distributed storage system running on an IaaS cloud, nodes can spontaneously join or leave the system. ElasticCopyset aims at minimal disruption of existing copysets when handling node addition and removal. Since there are complete and incomplete groups, the change of nodes is handled according to the type of groups the node belongs to. We have defined three roles of a node, as follows.

- Extra: the node is in the incomplete group, and not in any complete group.
- Elementary: the node is in a complete group, and not in the incomplete group.
- Supplementary: the node is in both complete and incomplete groups.

Figure 4.2 depicts the operations to deal with node changes. A new node is always added to the incomplete group, in which there are nodes selected from other complete groups as the supplement. Shown as Scenario 0, one of the *Supplementary* nodes is chosen and replaced by the new node. The chosen *Supplementary* node becomes *Elementary*, while the new node is marked as *Extra*. In addition, if $(N \mod N_G) = 0$ where there is no incomplete group, ElasticCopyset will randomly select $(N_G - 1)$ *Elementary* nodes from the complete groups, to form a new incomplete group with the new node. The selected *Elementary* nodes then become *Supplementary*.

An existing node can fail or be consciously removed from the system at any time. Node removal is handled according to the role of the removed node. There are three scenarios, labeled as 1, 2, and 3 in Figure 4.2, respectively.

Scenario 1: the removed node is *Extra*. An *Elementary* node is randomly selected from one complete group to replace the removed node. The selected node becomes *Supplementary*.

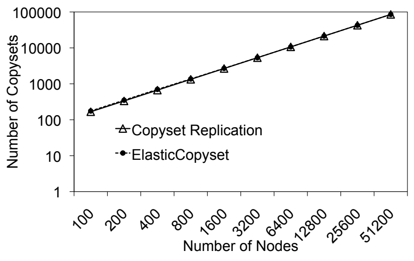
Scenario 2: the removed node is *Elementary*. The incomplete group offers one of the *Extra* nodes. This node replaces the removed node in the affected complete group, and becomes *Supplementary* since it has been assigned to a complete group. In addition, if $N_E = 0$ where there is no *Extra* node, the incomplete group is dismissed. The complete group with the removed node becomes the new incomplete group, wherein the existing nodes become the *Extra* nodes, thus $N_E = N_G - 1$. Another node is randomly selected from a complete group to replace the removed node, and becomes *Supplementary*.

Scenario 3: the removed node is *Supplementary*. There are two solutions. The first solution consists of two steps: i) select an *Elementary* node to replace the removed node in the incomplete group as in Scenario 1; ii) select an *Extra* node to replace the removed same node in the complete group as in Scenario 2. This approach is involved with two operations of node replacement. An alternative is to provision a new node from the cloud, and use it to replace the removed node.

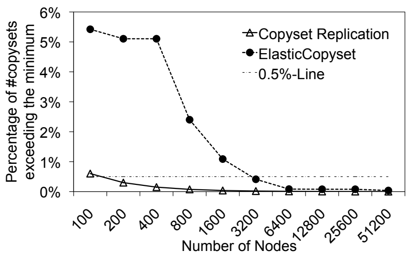
Now we discuss the number of copysets involved in each node operation. In each group, each node is associated with $\frac{S}{R-1}$ copysets. Hence, replacing a node in a group affects $\frac{S}{R-1}$ copysets, while changing the role of a node does not affects any copyset. As we can see in Figure 4.2, each node addition or removal is involved with only one replacement of node in the groups, thus affecting only $\frac{S}{R-1}$ copysets.

However, there are extreme cases where the whole incomplete group needs to be altered. When a new incomplete group is created (due to node addition), although there are $\frac{S}{R-1}\frac{N_G}{R}$ copysets formed in the new group, data movement is required in only $\frac{S}{R-1}$ copysets that contain the new node. When an existing incomplete group is dismissed, it does involve data movement for all its $\frac{S}{R-1}\frac{N_G}{R}$ copysets. Nevertheless, as discussed in Scenario 2, the incomplete group is dismissed when all its nodes have become *Supplementary*. In order to reduce the amount of data moved during this process, ElasticCopyset requires that no data should be assigned to the copysets that are in the incomplete group whilst having no *Extra* node. Hence, the data is moved out gradually as more *Extra* nodes become *Supplementary*. By the time when the incomplete group is to be dismissed, there is already no data left.

To summarise, in ElasticCopyset, each node addition or removal requires data movement in at most $\frac{S}{R-1}$ copysets, which involves only one scatter width of nodes.



(a) Number of copysets



(b) Percentage exceeding the minimum copyset number, i.e., Equation 3.1.

Figure 5.1: Number of copysets generated by different placement strategies.

5 Evaluation

In this section, we provide a set of experimental results to evaluate the impact of ElasticCopyset against Copyset Replication [7] and Random Replication, on data durability under correlated failures. We have used simulations to evaluate the algorithms as we do not have access to the thousands of compute nodes needed for real-world experiments. Nevertheless, ElasticCopyset is a general-purpose data placement scheme that can be implemented on a wide range of distributed storage systems that distribute data across a cluster of nodes.

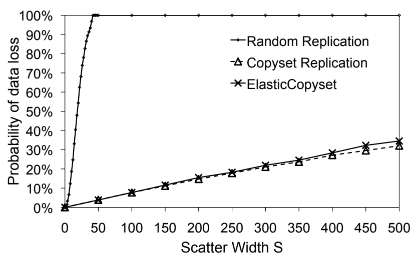
For all the experiments, the replication number is set as $R = 3$, as explained in Section 4.1. The number of nodes N changes between 1000 and 10000, and there are 10000 storage unit (e.g., data chunk, partition) assigned to each node. This is the typical cluster size at Facebook [2], LinkedIn [5] and RAMCloud [20], and is also consistent with the experiments conducted in Copyset Replication [7].

We have evaluated data durability under three scenarios: i) static deployment with varied settings of number of nodes, scatter width and percentage of nodes that failed; ii) scaling the system, both up and down, at a constant rate; and iii) elastic scaling based on workload demands. We have used the probability of data loss to quantify data durability. Each loss probability is calculated in this way: we simulate 10000 times of correlated failures. Each time, certain percentage of the nodes are randomly chosen to fail. This correlated failure is considered to have caused data loss if there exists at least one copyset in which all nodes have failed. The probability of data loss is equal to the number of correlated failures that have caused data loss, divided by 10000.

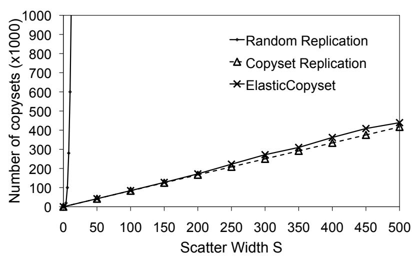
5.1 Evaluations on Static Deployment

Varied Numbers of Nodes

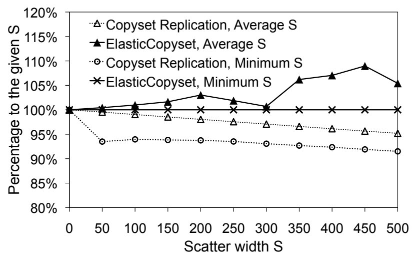
As Figure 1.1 has demonstrated, the probability of data loss is dominated by the number of copysets in the system. In this experiment, we used different numbers of nodes, to study the number of copysets generated using different placement schemes. The number of nodes is doubled at every step, starting from 100 up to 51200. The scatter width is set as $S = 10$, which is consistent with that reported for Facebook deployment [2] and in the evaluation of Copyset Replication [7].



(a) Probability of data loss, with 1% of nodes failed.



(b) Number of copysets



(c) Average/Minimum Scatter Width

Figure 5.2: Testing how the varied scatter width S will impact the system of 5000 nodes

Figure 5.1 compares the number of copysets generated by Copyset Replication and ElasticCopyset. Figure 5.1(a) shows that, the numbers of copysets generated in both schemes increase linearly with the number of nodes. In order to clearly differentiate the two numbers, we present Figure 5.1(b), which compares the resulted number of copysets towards the minimum copysets defined in Equation 3.1. As can be seen, when the number of nodes $N > 100$, the percentage of copysets exceeding the minimum number is less than 0.5% using Copyset Replication. In contrast, ElasticCopyset generates slightly more copysets. When $N \leq 400$, the number of copysets generated exceeds the minimum by 5%. This percentage of excess falls below 0.5% when $N \geq 3200$. Overall, both schemes generate a close to minimum number of copysets.

Varied Scatter Widths

In a data placement scheme, the value of scatter width is determined by the system administrators. Figure 5.2 depicts how a system of 5000 nodes will be affected when the scatter width S increases from 0 up to 500.

Figure 5.2(a) depicts the probability of data loss when 1% of the nodes fail simultaneously. As can be seen, the probability of data loss using Copyset Replication and ElasticCopyset is below 40% even when $S = 500$, while with Random Replication, the data is almost guaranteed to be lost when $S > 50$. Figure 5.2(b) shows that the number of copysets generated by Copyset Replication and ElasticCopyset grows steadily and linearly as S increases. In contrast, the number of copysets generated by Random Replication goes beyond one million even when S is small, because Random Replication forms one copyset for each storage unit, and there are 10000 storage units per node. By comparing

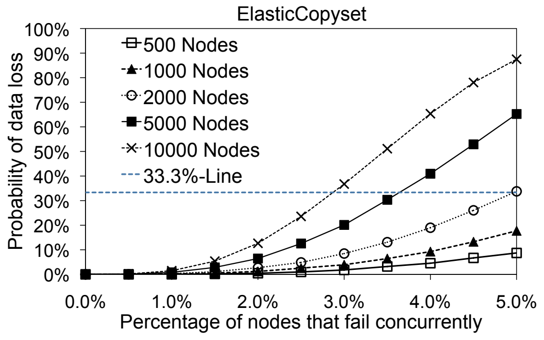


Figure 5.3: Probability of data loss with varying percentages of nodes failed simultaneously, using ElasticCopyset, given S=10.

Figure 5.2(a) and 5.2(b), it shows that the probability of data loss is determined by the number of copysets in the system. This is also consistent with the conclusion of Figure 1.1.

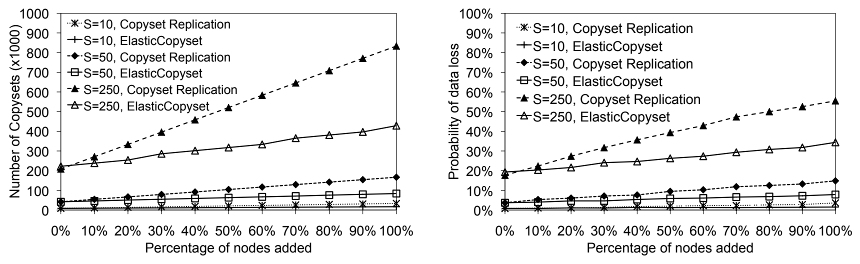
Figure 5.2(c) shows the percentage of the average and minimum scatter width of all the nodes, towards the S given by system administrators. As S increases, ElasticCopyset presents a wave-like ascent in the average scatter width, and a constant minimum scatter width equal to the given S. By comparison, Copyset Replication shows a steady decline in both the average and minimum scatter widths. The latter is less than 94% of S when $S > 50$. Overall, ElasticCopyset exhibits higher scatter width than Copyset Replication, but the percentages of deviation to the given S are less than 10% in both schemes.

The result of Figure 5.2(c) can be explained. ElasticCopyset guarantees that each node is put into copysets with S distinct nodes, so the minimum scatter width is always equal to S. Moreover, the incomplete group generates extra copysets, the number of which, i.e., Equation 4.4, is determined by both S and N_E, causing a non-linear increase in the average scatter width of ElasticCopyset. In contrast, Copyset Replication uses random permutation to form $\frac{S}{R-1}$ copysets for each node, without guarantee that there are S distinct nodes. A greater S results in more duplicating nodes in the copysets that a node belongs to. Hence, the average and minimum scatter widths both decrease as the given S grows.

Varied Percentages of Node Failure

In this experiment, we study data durability under varied percentages of nodes that fail in a correlated failure. There are five tests, wherein the number of nodes grows from 500 to 10000. In each test, the percentage of nodes that are selected to fail increases by 0.5% from 0 up to 5%. The scatter width is $S = 10$.

Figure 5.3 depicts the probability of data loss using ElasticCopyset. For a system of 500 nodes, the loss probability rises, but remains below 10% even when 5% of the nodes fail simultaneously. As the number of nodes increases, there is a clear trend that the probability of data loss grows more quickly. For example, when 5% of 10000 nodes fail concurrently, the probability of data loss is almost 90%. The reason is that, given the same percentage of node failure, there are more nodes failed in a system with more nodes, which naturally leads to higher data loss probability.



(a) Number of copysets (b) Probability of data loss

Figure 5.4: Scale up the system by adding nodes one-by-one at runtime. Given initial number of nodes $N = 5000$ that scales up to 10000. The figures show varying percentage that the system is scaled, using varying setting of scatter width. The percentage of node failure is 1%.

We have added a horizontal line at 33.3% in Figure 5.3. If a system can tolerate one time of data loss in every three system failures, ElasticCopyset can sustain 5% of node failure rate for up to 2000 nodes. Alternatively, it can support a system of 10000 nodes if no more than 3% of the nodes would fail simultaneously. We have also conducted this experiment using Copyset Replication. The results (not shown due to page limit) are almost identical to Figure 5.3. This is because both Copyset Replication and ElasticCopyset generate a close to minimum number of copysets (Figure 5.1).

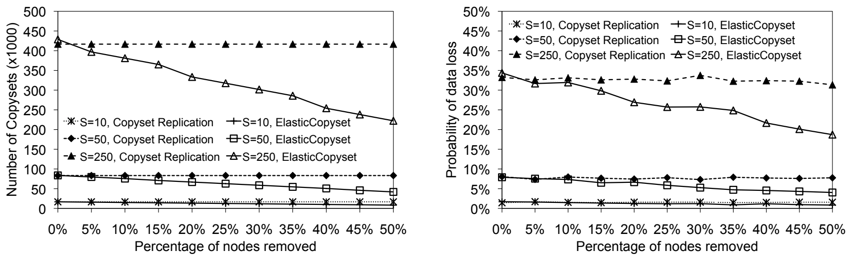
5.2 Scaling at a constant rate

We have studied the data durability with static deployment, in which the copysets were statically generated based on the settings of the system. In this subsection, we study the scalability of the placement schemes with two online evaluations, wherein the system is scaling during runtime, both up and down at a constant rate.

The first experiment is the evaluation on system scale-up. The placement schemes are required to handle node additions. The initial number of nodes is $N = 5000$. The new nodes are added one after another during the runtime, until the system scale is doubled (i.e., $N = 10000$). There are three tests, with the scatter width S set as 10, 50, and 250, respectively.

Figure 5.4(a) compares the number of copysets generated by Copyset Replication and ElasticCopyset during scale-up. In all the tests, as the system scales up, the number of copysets increases more quickly when using Copyset Replication than when using ElasticCopyset. Consequently, as shown in Figure 5.4(b), given 1% of nodes that simultaneously fail, ElasticCopyset consistently presents a smaller probability of data loss than Copyset Replication. When the system is scaled up by 100%, the probability of data loss in Copyset Replication is also almost 100% greater than ElasticCopyset.

The second experiment is to study system scale-down, which is exactly the reverse of the scale-up experiment. The number of nodes is $N = 10000$. The nodes were removed from the system one-by-one during runtime, until the system scale was reduced to a half (i.e., $N = 5000$). Similarly, there are three tests, with $S = 10$, 50 and 250. The percentage of nodes that fail is also 1%.



(a) Number of copysets (b) Probability of data loss

Figure 5.5: Scale down the system by removing nodes one-by-one at runtime. Given initial number of nodes N=10000 that scales down to 5000. The figures show varying percentage that the system is scaled, using varying setting of scatter width. The percentage of node failure is 1%.

As depicted in Figure 5.5(a), as the system scales down from 10000 to 5000, the number of copysets remains constant when using Copyset Replication. In contrast, ElasticCopyset is able to steadily reduce the copyset number during the scale-down. As a result, ElasticCopyset exhibits a smaller probability of data loss than Copyset Replication in Figure 5.5(b). When the system is scaled down by 50%, the probability of data loss in ElasticCopyset is 50% smaller than Copyset Replication.

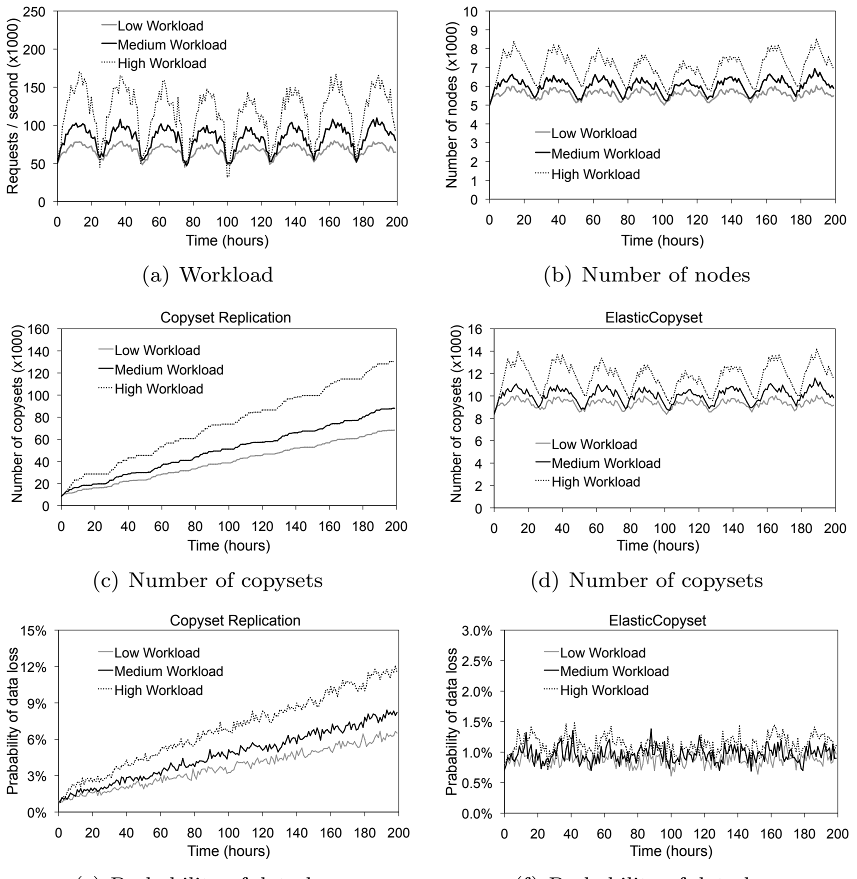
The experiments have demonstrated that, ElasticCopyset handles online node addition and removal better than Copyset Replication, and maintains a smaller probability of data loss than Copyset Replication during both scale-up and scale-down.

5.3 Elastic scaling under workload

In this experiment, we study the scalability of the placement schemes when the system is subject to a dynamic workload. The test scenario is based on the workload analysis on Facebook's Memcached deployment [1]. The workload (e.g. number of requests per second) follows the diurnal pattern, wherein there are peaks in day time and bottoms in night time. The system is, either automatically or manually, scaled up and down incrementally according to the changes of workload. During the process of dynamic scaling, we compared ElasticCopyset against Copyset Replication, on data durability.

The input of this experiment is depicted in Figure 5.6(a). We simulated the workload to match the temporal patterns described in the Facebook deployment [1]. Every 24 hours is a period of wave, and there are 200 hours in total. The bottom workload is set as 50,000 requests per second. Depending on the peak workload, there are three types of patterns, namely *low*, *medium*, and *high*. The peak workload are 75000, 100000, and 150000 reqs/sec, respectively, which are 50%, 100% and 200% greater than the bottom workload. Hence, in the remaining of Figure 5.6, the results are labeled as Low, Medium, and High Workload, respectively.

Based on these workload, we then simulated the changes of number of nodes. As shown in Figure 5.6(b), when the workload is at the bottoms, there are 5000 nodes. The number of nodes is increased as the workload rises. The system scale



(a) Workload (b) Number of nodes

(c) Number of copysets (d) Number of copysets

(e) Probability of data loss (f) Probability of data loss

Figure 5.6: Compare the performance of dynamic scaling between Copyset Replication and ElasticCopyset. The system is scaled based on different types of workloads. Initial number of nodes N=5000, with 1% of nodes fail.

peaks approximately at 6000, 7000, and 8500 nodes, for Low, Medium, and High Workload, respectively. Hence, given $N = 5000$ at the bottom, the percentages of scaling up are respectively 20%, 40% and 80% under the workloads.

The number of nodes in Figure 5.6(b) is served as the direct input for evaluating the data durability at elastic scaling. The numbers of copysets generated are shown in Figure 5.6(c) and 5.6(d), respectively. Initially, about 8500 copysets were generated in both schemes. However, as the time elapses, the two schemes exhibit two disparate trends. In Copyset Replication, the number of copysets cumulates step by step every 24 hours, and grows steadily each day. While in ElasticCopyset, the number of copysets rises and declines according to the number of nodes, and remains at a horizontal level in the long run.

Consequently, given 1% of nodes that fail, the probability of data loss, which is determined by the number of copysets, also exhibits two disparate trends. As shown in Figure 5.6(e), the loss probability in Copyset Replication increases steadily day after day. During 200 hours, the probability grows from 1% to 12% under the high workload. At this rate, the probability of data loss will exceed 50% within 1000 hours (i.e. 40 days). In contrast, ElasticCopyset manages to maintain the data loss probability below 1.5% under varied workloads (Figure 5.6(f)).

Figure 5.6 has demonstrated that, when the system is dynamically scaled up and down within a range over a long period of time, ElasticCopyset is able to maintain data loss probability at a close to minimised level, while Copyset Replication is not.

5.4 Discussions

We have presented the experimental results. The evaluations on static deployment have demonstrated that, ElasticCopyset exhibits a close to minimised probability of data loss under varied system setups and failure rates. Yet, it is noticeable (Figure 5.1(b) and 5.2(b)) that ElasticCopyset generates more copysets than Copyset Replication, resulting in slightly higher probability of data loss shown in Figure 5.2(a). This is due to the extra copysets generated in the incomplete group (Equation 4.4). Nevertheless, as shown in Figure 5.1(b), the percentage of extra copysets is less than 6%.

The evaluations on scalability have shown that ElasticCopyset greatly outperforms Copyset Replication by maintaining the minimum copysets during online node addition and removal. The results can be explained. During scale-up, Copyset Replication requires $\frac{S}{R-1}$ new copysets to be generated for each node addition, while ElasticCopyset generates new copysets only when a new incomplete group is required. To illustrate, we assume n new nodes are added. Copyset Replication creates $\frac{S}{R-1}n$ new copysets, while ElasticCopyset creates $\frac{S}{R-1}\frac{N_G}{R}\lceil\frac{n}{N_G}\rceil$ new copysets (Equation 4.3). When $n \gg N_G$, then $\frac{S}{R-1}\frac{N_G}{R}\lceil\frac{n}{N_G}\rceil \approx \frac{S}{R-1}\frac{n}{R}$. Since $\frac{S}{R-1}n$ is R times as many as $\frac{S}{R-1}\frac{n}{R}$, given $R = 3$, then the number of copysets generated for new nodes is almost tripled in Copyset Replication than in ElasticCopyset.

During scale-down, when a node is removed, Copyset Replication uses an existing node to replace it, without destroying any existing copyset. While in ElasticCopyset, the whole incomplete group is dismissed when there is no *Extra* node. Hence, when the system is scaled down, the number of copysets in ElasticCopyset is reduced accordingly, while the number in Copyset Replication remains unchanged.

Overall, ElasticCopyset leverages the incomplete group to create and dismiss copysets for dynamic node addition and removal, and also maintains a close to minimised probability of data loss in both static deployment and dynamic scaling.

6 Conclusion

This paper addressed the issue of data durability under correlated node failures for distributed storage systems that are required to efficiently scale up and down in direct response to workload demands. We have proposed ElasticCopyset, a novel, general-purpose replica placement scheme that builds on the concept of "copysets" [7, 8]. Given the scatter width of the data placement, ElasticCopyset defines the minimum number of nodes that can form a group, and splits the nodes in the system into a list of complete group and one incomplete group.

ElasticCopyset uses a novel shuffle algorithm to generate a minimum number of non-overlapping copysets within each group to minimise the probability of data loss. ElasticCopyset also leverages the incomplete group to efficiently handle node addition and removal, such that each node operation affects only one scatter width of nodes.

We have evaluated ElasticCopyset against the current-state replica placement schemes including Random Replication and Copyset Replication. The evaluation has demonstrated that ElasticCopyset is able to maintain a close to minimum probability with the setting of varying values of scatter width, system scales, and percentage of node failure. In contrast to the current-state Copyset Replication, ElasticCopyset has also exhibited much better scalability and elasticity in the scenario where the distributed system is required to dynamically scale up and down under the diurnal workload pattern.

Future plans include evaluating ElasticCopyset in an actual distributed storage system such as HDFS or Cassandra. We would also like to leverage distributed protocols in order to co-ordinate the placement of data storage units across the system.

Bibliography

[1] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload analysis of a large-scale key-value store. In *ACM SIGMETRICS Performance Evaluation Review*, volume 40, pages 53–64. ACM, 2012.

[2] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, et al. Apache hadoop goes realtime at facebook. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1071–1080. ACM, 2011.

[3] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):1–26, 2008.

[4] F. W. Chang, M. Ji, S.-T. Leung, J. MacCormick, S. E. Perl, and L. Zhang. Myriad: Cost-effective disaster tolerance. In *FAST*, volume 2, page 8, 2002.

[5] R. J. Chansler. Data availability and durability with the hadoop distributed file system. *The USENIX Magazine, FILESYSTEMS*, 2012.

[6] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris. Efficient replica maintenance for distributed storage systems. In *NSDI*, volume 6, pages 4–4, 2006.

[7] A. Cidon, S. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum. Copysets: Reducing the frequency of data loss in cloud storage. *Proceedings of the 2013 USENIX conference on Annual Technical Conference*, 2013.

[8] A. Cidon, R. Stutsman, S. Rumble, S. Katti, J. Ousterhout, and M. Rosenblum. Mincopysets: Derandomizing replication in cloud storage. In *Networked Systems Design and Implementation (NSDI)*, 2013.

[9] B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proceedings of the VLDB Endowment*, 1(2):1277–1288, 2008.

[10] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. *ACM SIGOPS Operating Systems Review*, 35(5):202–215, 2001.

[11] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[12] B. Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004(124):5, 2004.

[13] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *OSDI*, pages 61–74, 2010.

[14] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[15] Z. Guo et al. Failure recovery: When the cure is worse than the disease. In *Proceedings of the 14th USENIX Conference on Hot Topics in Operating Systems*, HotOS'13, page 88, Berkeley, CA, USA, 2013. USENIX Association.

[16] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 143–158. USENIX Association, 2005.

[17] J. Hamilton. Geo-replication at facebook.

[18] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[19] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *ACM SIGOPS Operating Systems Review*, volume 30, pages 84–92. ACM, 1996.

[20] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum. Fast crash recovery in ramcloud. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 29–41. ACM, 2011.

[21] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, et al. The case for ramclouds: scalable high-performance storage entirely in dram. *ACM SIGOPS Operating Systems Review*, 43(4):92–105, 2010.

[22] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 109–116. ACM, 1988.

[23] H. Patterson, S. Manley, M. Federwisch, D. Hitz, S. Kleiman, and S. Owara. Snapmirror®: file system based asynchronous mirroring for disaster recovery. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, pages 9–9. USENIX Association, 2002.

[24] R. Rodrigues and B. Liskov. High availability in dhts: Erasure coding vs. replication. In *Peer-to-Peer Systems IV*, pages 226–239. Springer, 2005.

[25] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. Fab: building distributed enterprise disk arrays from commodity components. *ACM SIGOPS Operating Systems Review*, 38(5):48–58, 2004.

[26] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE, 2010.

[27] R. van Renesse and F. B. Schneider. Chain replication for supporting high throughput and availability. In *OSDI*, volume 4, pages 91–104, 2004.

[28] K. V. Vishwanath and N. Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM Symposium on Cloud computing*, pages 193–204. ACM, 2010.

[29] H. Weatherspoon and J. D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Peer-to-Peer Systems*, pages 328–337. Springer, 2002.

[30] H. Yu, P. B. Gibbons, and S. Nath. Availability of multi-object operations. In *Proc. of the Third USENIX Symp. on Networked Systems Design and Implementation*, pages 211–224, 2006.

A Proof of Lemmas

A.1 Proof of Lemma 1

Proof. According to Order 2, $b[i]$ appears at $m_b[x][y]$, wherein $x_b = (i + k * C) \bmod L$, $y_b = \lfloor \frac{i+k*C}{L} \rfloor$, given $i \in [0, C)$ and $k \in [0, L)$. When $C \geq L$, let $C = L + t$, wherein $t \geq 0$. Thus, for any $k_1 < k_2$, i.e. $k_1 + 1 \leq k_2$, we have:
$y_b(k_2) \geq \lfloor \frac{i+(k_1+1)*(L+t)}{L} \rfloor > \lfloor k_1 + \frac{i+k_1*t+t}{L} \rfloor \geq \lfloor k_1 + \frac{i+k_1*t}{L} \rfloor$
$= \lfloor \frac{i+k_1*(L+t)}{L} \rfloor = y_b(k_1)$.

That is, $y_b(k_1) \neq y_b(k_2)$. There are no two replicas of $b[i]$ appearing in the same column. However, when $C < L$, there exists at least one violation. For example, let $i = 0$, $k_1 = 0$ and $k_2 = 1$, then $y_b = \lfloor k_1 \frac{C}{L} \rfloor = \lfloor k_2 \frac{C}{L} \rfloor = 0$. That is, $b[0]$ appears twice in Column $y_b = 0$.

According to Order 3, for each $c[i]$, $y_c = C - 1 - y_b$. Given $C \geq L$ and $k_1 \neq k_2$, since $y_b(k_1) \neq y_b(k_2)$, then $y_c(k_1) \neq y_c(k_2)$. Hence, Lemma 1 is also valid for $c[i]$.

□

A.2 Proof of Lemma 2

Proof. Given any two copysets, i.e., $(a[i_1], b[j_1], c[k_1])$ and $(a[i_2], b[j_2], c[k_2])$. We use proof by contradiction. We presume $i_1 = i_2$ and $j_1 = j_2$. According to Order 1, $a[i]$ is placed at $m_a[x][y]$ wherein $y_a = i$. Since $i_1 = i_2$, then $y_{a1} = y_{a2}$. Moreover, since $y_{a1} = y_{b1}$ and $y_{a2} = y_{b2}$ (defined in *merge* phase), then $y_{b1} = y_{b2}$. However, according to Lemma 1, if $j_1 = j_2$, then $y_{b1} \neq y_{b2}$, which contradicts $y_{b1} = y_{b2}$. That is, the presumption is not valid. Hence, there does not exist two copysets sharing the same $a[i]$ and $b[j]$. Similarly, if $k_1 = k_2$, then $y_{c1} \neq y_{c2}$. There does not exist two copysets that share the same $a[i]$ and $c[k]$, either. □

A.3 Proof of Lemma 3

Proof. According to Order 1, $a[i]$ is placed at $x_a = k$ wherein $k \in [0, L)$. Thus, $a[i]$ appears in each and every row.

According to Order 2, $b[i]$ appears at $x_b = (i + k * C) \bmod L$ wherein $k \in [0, L)$. We use proof by contradiction. We presume there exist $b[i]$ and $k_1 \neq k_2$, such that $x_b = x_{b1} = x_{b2}$, i.e., presuming two integers t_1 and t_2, such that:

$$\begin{cases} i + k_1 * C = t_1 * L + x_b \\ i + k_2 * C = t_2 * L + x_b \end{cases} \tag{A.1}$$

From Equation A.1, we have: $\frac{C}{L} = \frac{t_2 - t_1}{k_2 - k_1}$. Let $k_1 < k_2$, then $0 < k_2 - k_1 < L$. Since $(t_2 - t_1)$ and $(k_2 - k_1)$ are both integers, there must exist an integer $s > 1$, such that $\frac{C}{L} = \frac{s(t_2-t_1)}{s(k_2-k_1)}$, i.e., s is a common divisor of C and L. However, C and L are co-prime, meaning that their greatest common divisor is $s = 1$, which contradicts $s > 1$. Hence, the presumption is invalid. there is no $b[i]$ appearing more than once in any row of the matrix. Moreover, there are L replicas of $b[i]$ in m_b that has exactly L rows. Since each row contains at most one $b[i]$, the only possible layout is that each $b[i]$ appears once in each and every row. In addition, according to Order 3, $x_c = (i + k * C) \mod L = x_b$. The conclusion for $b[i]$ is also valid for $c[j]$. $\square$

A.4 Proof of Lemma 4

It consists of two parts: i) L is even; and ii) L is odd.

Proof for an Even L

Proof. Let $L = 2t$ and $C = 2t + r$, wherein $t \geq 1$ and $r \geq 0$. According to Order 2, $\forall k_b \in [0, L)$, each $b[i]$ appears at $m_b[x][y]$ where:

$y_b = \lfloor \frac{i+k_bC}{L} \rfloor = \lfloor \frac{i+k_b*(2t+r)}{2t} \rfloor = k_b + \lfloor \frac{i+k_br}{2t} \rfloor$

$x_b = (i + k_bC) \mod L \Longrightarrow x_b = i + k_br - 2t\lfloor \frac{i+k_br}{2t} \rfloor$

According to Order 3, $\forall k_c \in [0, L)$, each $c[j]$ appears at $m_c[x][y]$ where:

$y_c = C - 1 - \lfloor \frac{j+k_cC}{L} \rfloor = C - 1 - k_c - \lfloor \frac{j+k_cr}{2t} \rfloor$

$x_c = (j + k_cC) \mod L \Longrightarrow x_c = j + k_cr - 2t\lfloor \frac{j+k_cr}{2t} \rfloor$

If $(b[i], c[j])$ appears in the same copyset, i.e., $x_b = x_c$ and $y_b = y_c$, then we have Equation A.2.

$$\begin{cases} \lfloor \frac{i+k_br}{2t} \rfloor - \lfloor \frac{j+k_cr}{2t} \rfloor = \frac{i+k_br}{2t} - \frac{j+k_cr}{2t} \\ \lfloor \frac{i+k_br}{2t} \rfloor + \lfloor \frac{j+k_cr}{2t} \rfloor = 2t + r - 1 - (k_b + k_c) \end{cases} \tag{A.2}$$

From $\lfloor \frac{i+k_br}{2t} \rfloor - \lfloor \frac{j+k_cr}{2t} \rfloor = \frac{i+k_br}{2t} - \frac{j+k_cr}{2t}$, we know $(i + k_br)$ and $(j + k_cr)$ are congruent modulo $2t$. That is,

$$i + k_br \equiv j + k_cr \mod 2t \tag{A.3}$$

When $r = 0$, Simultaneous equations A.2 for (k_b, k_c) becomes $\begin{cases} \lfloor \frac{i}{2t} \rfloor - \lfloor \frac{j}{2t} \rfloor = \frac{i}{2t} - \frac{j}{2t} \\ \lfloor \frac{i}{2t} \rfloor + \lfloor \frac{j}{2t} \rfloor = 2t - 1 - (k_b + k_c) \end{cases}$, which has multiple valid solutions for (k_b, k_c), given certain $b[i]$ and $c[j]$. For example, let $i = j = 0$, then there are $2t$ possible solutions shown in Equation A.4. Thus, $r \neq 0$.

$$\begin{cases} k_b = 0 \\ k_c = 2t - 1 \end{cases} \quad \begin{cases} k_b = 1 \\ k_c = 2t - 2 \end{cases} \quad \begin{cases} ...... \\ ...... \end{cases} \quad \begin{cases} k_b = 2t - 1 \\ k_c = 0 \end{cases} \tag{A.4}$$

When $r = 1$, i.e., $C = L+1$, Simultaneous equations A.2 for (k_b, k_c) becomes

$$\begin{cases} \lfloor \frac{i+k_b}{2t} \rfloor - \lfloor \frac{j+k_c}{2t} \rfloor = \frac{i+k_b}{2t} - \frac{j+k_c}{2t} \\ \lfloor \frac{i+k_b}{2t} \rfloor + \lfloor \frac{j+k_c}{2t} \rfloor = 2t - (k_b + k_c) \end{cases} \tag{A.5}$$

Given any valid value of i, j and t, we use proof by contradiction to prove that Simultaneous equations A.5 have at most one solution for (k_b, k_c). We presume there are two solutions, i.e. (k_{b1}, k_{c1}) and (k_{b2}, k_{c2}), such that $k_{b1} \neq k_{b2}$ and $k_{c1} \neq k_{c2}$, then we have Equations A.6 and A.7.

$$\begin{aligned} (\lfloor \frac{i + k_{b2}}{2t} \rfloor - \lfloor \frac{i + k_{b1}}{2t} \rfloor) - (\lfloor \frac{j + k_{c2}}{2t} \rfloor - \lfloor \frac{j + k_{c1}}{2t} \rfloor) \\ = \frac{k_{b2} - k_{b1}}{2t} - \frac{k_{c2} - k_{c1}}{2t} \end{aligned} \tag{A.6}$$

$$\begin{aligned}(\lfloor\frac{i+k_{b2}}{2t}\rfloor - \lfloor\frac{i+k_{b1}}{2t}\rfloor) + (\lfloor\frac{j+k_{c2}}{2t}\rfloor - \lfloor\frac{j+k_{c1}}{2t}\rfloor) \\ = -(k_{b2}-k_{b1}) - (k_{c2}-k_{c1})\end{aligned} \tag{A.7}$$

From Equation A.3, we have $k_{b2} - k_{b1} \equiv k_{c2} - k_{c1} \mod 2t$, when $r = 1$. Thus, $(k_{b2} - k_{b1}) = (k_{c2} - k_{c1}) \pm n * 2t$, where $n \geq 0$. Let $k_{b1} < k_{b2}$. Since $\forall k \in [0, L) = [0, 2t)$, then we have $0 < k_{b2} - k_{b1} < 2t$ and $-2t < k_{c2} - k_{c1} < 2t$. Thus, we have Equation A.8, wherein $n \in \{0, 1\}$.

$$(k_{b2} - k_{b1}) = (k_{c2} - k_{c1}) + n * 2t \tag{A.8}$$

Presuming $n = 0$, i.e. $k_{b2} - k_{b1} = k_{c2} - k_{c1} > 0$, then,
$(\lfloor\frac{i+k_{b2}}{2t}\rfloor - \lfloor\frac{i+k_{b1}}{2t}\rfloor) + (\lfloor\frac{j+k_{c2}}{2t}\rfloor - \lfloor\frac{j+k_{c1}}{2t}\rfloor)$
$\geq (\lfloor\frac{i+k_{b1}}{2t}\rfloor - \lfloor\frac{i+k_{b1}}{2t}\rfloor) + (\lfloor\frac{j+k_{c1}}{2t}\rfloor - \lfloor\frac{j+k_{c1}}{2t}\rfloor) = 0$,
while $-(k_{b2} - k_{b1}) - (k_{c2} - k_{c1}) < 0$, which contradicts Equation A.7. Hence, $n \neq 0$.

When $n = 1$ and $k_{b1} < k_{b2}$, we have $-2t < k_{c2} - k_{c1} < 0$, and $(k_{b2} - k_{b1}) = (k_{c2} - k_{c1}) + 2t \Longrightarrow \frac{k_{b2}-k_{b1}}{2t} - \frac{k_{c2}-k_{c1}}{2t} = 1$.

Thus, from Equation A.6, we have,
$(\lfloor\frac{i+k_{b2}}{2t}\rfloor - \lfloor\frac{i+k_{b1}}{2t}\rfloor) - (\lfloor\frac{j+k_{c2}}{2t}\rfloor - \lfloor\frac{j+k_{c1}}{2t}\rfloor) = 1$

Since $k_b < L$, let $k_b = L - s = 2t - s$, wherein $s > 0$. Since $i \leq C - 1 = 2t$, we have: $\lfloor\frac{i+k_b}{2t}\rfloor \leq \lfloor\frac{2t+2t-s}{2t}\rfloor = 2 + \lfloor\frac{-s}{2t}\rfloor < 2$. Thus, $\lfloor\frac{i+k_b}{2t}\rfloor \in \{0, 1\}$. Similarly, $\lfloor\frac{j+k_c}{2t}\rfloor \in \{0, 1\}$. Therefore,

$$\begin{cases}\lfloor\frac{i+k_{b2}}{2t}\rfloor - \lfloor\frac{i+k_{b1}}{2t}\rfloor \in \{-1,0,1\} \\ \lfloor\frac{j+k_{c2}}{2t}\rfloor - \lfloor\frac{j+k_{c1}}{2t}\rfloor \in \{-1,0,1\} \\ (\lfloor\frac{i+k_{b2}}{2t}\rfloor - \lfloor\frac{i+k_{b1}}{2t}\rfloor) - (\lfloor\frac{j+k_{c2}}{2t}\rfloor - \lfloor\frac{j+k_{c1}}{2t}\rfloor) = 1\end{cases} \Longrightarrow$$

$(\lfloor\frac{i+k_{b2}}{2t}\rfloor - \lfloor\frac{i+k_{b1}}{2t}\rfloor) + (\lfloor\frac{j+k_{c2}}{2t}\rfloor - \lfloor\frac{j+k_{c1}}{2t}\rfloor) = \pm 1$.

Simultaneous equations A.6 and A.7 become:

$$\begin{cases}(k_{b2} - k_{b1}) - (k_{c2} - k_{c1}) = 2t \\ -(k_{b2} - k_{b1}) - (k_{c2} - k_{c1}) = \pm 1\end{cases} \Longrightarrow$$

$$\begin{cases}k_{b2} - k_{b1} = t \mp 1/2 \\ k_{c2} - k_{c1} = -t \pm 1/2\end{cases}$$

However, since $\forall k$ are integers, $(k_{b2} - k_{b1})$ or $(k_{c2} - k_{c1})$ cannot be non-integers. That is, when $r = 1$, there does not exist two distinct solutions (k_{b1}, k_{c1}) and (k_{b2}, k_{c2}) for any two nodes $b[i]$ and $c[j]$. Hence, when L is even, any combination of $(b[i], c[j])$ appear together in at most one copyset.

$\square$

Proof for an Odd L

Proof. When L is an odd number, let $L = 2t - 1$ and $C = 2t + r$, wherein $t \geq 1$ and $r \geq -1$ and both are integers. Similar to the previous proof, given $\forall k_b \in [0, L)$ and $\forall k_c \in [0, L)$, we have,

$y_b = k_b + \lfloor\frac{i+k_b*(r+1)}{2t-1}\rfloor$
$x_b = i + k_b * (r+1) - (2t-1)\lfloor\frac{i+k_b*(r+1)}{2t-1}\rfloor$
$y_c = C - 1 - k_c - \lfloor\frac{j+k_c*(r+1)}{2t-1}\rfloor$
$x_c = j + k_c * (r+1) - (2t-1)\lfloor\frac{j+k_c*(r+1)}{2t-1}\rfloor$

If $(b[i], c[j])$ appears in the same copyset, i.e., $x_b = x_c$ and $y_b = y_c$. Then we have Equation A.9.

$$\begin{cases} \lfloor\frac{i+k_b*(r+1)}{2t-1}\rfloor - \lfloor\frac{j+k_c*(r+1)}{2t-1}\rfloor \\ \qquad = \frac{i+k_b*(r+1)}{2t-1} - \frac{j+k_c*(r+1)}{2t-1} \\ \lfloor\frac{i+k_b*(r+1)}{2t-1}\rfloor + \lfloor\frac{j+k_c*(r+1)}{2t-1}\rfloor \\ \qquad = 2t + r - 1 - (k_b + k_c) \end{cases} \tag{A.9}$$

When $r = -1$ or $r = 0$, Simultaneous equations A.9 for (k_b, k_c) have multiple solutions. We have to skip the proof due to page limit. Hence, $r \neq -1$ and $r \neq 0$.

When $r = 1$, i.e., $C = L + 2$, we have Equations A.10.

$$\begin{cases} \lfloor\frac{i+2k_b}{2t-1}\rfloor - \lfloor\frac{j+2k_c}{2t-1}\rfloor = \frac{i+2k_b}{2t-1} - \frac{j+2k_c}{2t-1} \\ \lfloor\frac{i+2k_b}{2t-1}\rfloor + \lfloor\frac{j+2k_c}{2t-1}\rfloor = 2t - (k_b + k_c) \end{cases} \tag{A.10}$$

Similarly, we use proof by contradiction to prove that there is at most one solution. Presume there are two solutions, i.e. (k_{b1}, k_{c1}) and (k_{b2}, k_{c2}), such that $k_{b1} \neq k_{b2}$ and $k_{c1} \neq k_{c2}$, then we have Equations A.11 and A.12.

$$\begin{aligned} &(\lfloor\frac{i+2k_{b2}}{2t-1}\rfloor - \lfloor\frac{i+2k_{b1}}{2t-1}\rfloor) - (\lfloor\frac{j+2k_{c2}}{2t-1}\rfloor - \lfloor\frac{j+2k_{c1}}{2t-1}\rfloor) \\ &\qquad = \frac{2(k_{b2}-k_{b1})}{2t-1} - \frac{2(k_{c2}-k_{c1})}{2t-1} \end{aligned} \tag{A.11}$$

$$\begin{aligned} &(\lfloor\frac{i+2k_{b2}}{2t-1}\rfloor - \lfloor\frac{i+2k_{b1}}{2t-1}\rfloor) + (\lfloor\frac{j+2k_{c2}}{2t-1}\rfloor - \lfloor\frac{j+2k_{c1}}{2t-1}\rfloor) \\ &\qquad = -(k_{b2}-k_{b1}) - (k_{c2}-k_{c1}) \end{aligned} \tag{A.12}$$

When $r = 1$, let $k_{b1} < k_{b2}$. Similarly, we have Equation A.13, wherein $n \in \{0, 2\}$.

$$2(k_{b2} - k_{b1}) = 2(k_{c2} - k_{c1}) + n * (2t - 1) \tag{A.13}$$

When $n = 0$, Equation A.12 is violated. The proof is skipped due to page limit. Hence, $n \neq 0$.

When $n = 2$, we have $\frac{k_{b2}-k_{b1}}{2t-1} - \frac{k_{c2}-k_{c1}}{2t-1} = 1$. From Equation A.11, we have $(\lfloor\frac{i+2k_{b2}}{2t-1}\rfloor - \lfloor\frac{i+2k_{b1}}{2t-1}\rfloor) - (\lfloor\frac{j+2k_{c2}}{2t-1}\rfloor - \lfloor\frac{j+2k_{c1}}{2t-1}\rfloor) = 2$. Similarly, it can be deducted that, $(\lfloor\frac{i+2k_{b2}}{2t-1}\rfloor - \lfloor\frac{i+2k_{b1}}{2t-1}\rfloor) + (\lfloor\frac{j+2k_{c2}}{2t-1}\rfloor - \lfloor\frac{j+2k_{c1}}{2t-1}\rfloor) = 2s$, where $s \in \{-1, 0, 1\}$.

Hence, the solution for Equations A.11 and A.12 is:

$$\begin{cases} k_{b2} - k_{b1} = t - s - 1/2 \\ k_{c2} - k_{c1} = -t - s + 1/2 \end{cases}.$$

However, since $\forall k$ are integers, $(k_{b2} - k_{b1})$ or $(k_{c2} - k_{c1})$ cannot be non-integers. Hence, when L is odd, any combination of $(b[i], c[j])$ appear together in at most one copyset.

□

A.5 Proof of Lemma 5

Proof. Given any two copysets, i.e. $(a[i_1], b[j_1], c[k_1])$ and $(a[i_2], b[j_2], c[k_2])$. We presume the same $a[i]$ is shared in these two copysets, i.e., $i_1 = i_2$. According to Lemma 2, $b[j_1]$ and $b[j_2]$ must be different, and $c[k_1]$ and $c[k_2]$ are also different nodes. Hence, $a[i]$ is the only common node. Otherwise, we presume the same $b[j]$ is shared. According to Lemma 4, $c[k_1]$ and $c[k_2]$ must be different. Hence, $b[j]$ is the only common node. Moreover, if both $a[i]$ and $b[j]$ are not shared, then $c[k]$ also becomes the only possible common node. To sum up, two copysets share at most one common node in any cases.

$\square$

A.6 Proof of Lemma 6

Proof. Given any node n_i in the group. Since C is the smallest odd number greater than L, then L and C are co-prime. According to Lemma 3, n_i appears once in each and every row in the matrix that consists of L rows. Since nodes from different rows cannot form a copyset, then n_i belongs to L distinct copysets. In each copyset that the node n_i belongs to, there are $R-1$ other distinct nodes. According to Lemma 5, any two copysets share at most one common node, which can only be n_i. Therefore, the L copysets contain $L * (R-1)$ distinct nodes other than n_i. Since $S = L * (R-1)$, then n_i shares copysets with S other distinct nodes.

$\square$