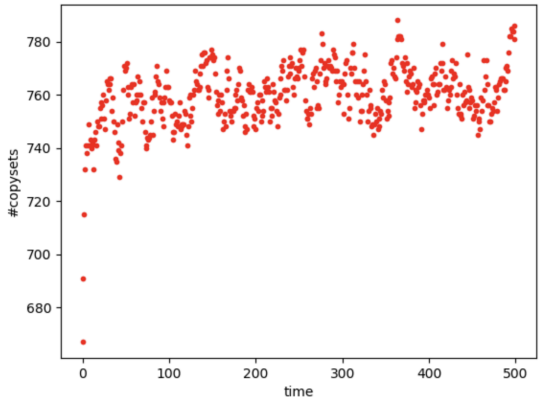
Abstract

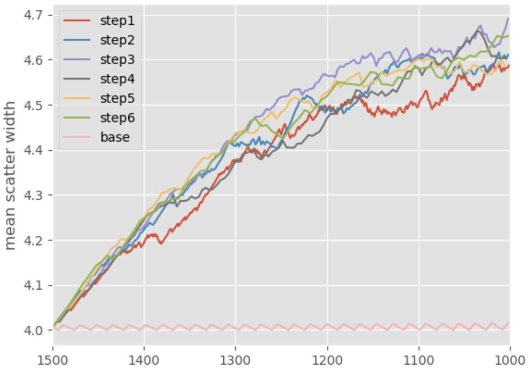
Cloud storage system use Replication Algorithm to prevent data loss. The most common algorithm is Random Replication. However, it is almost guaranteed to lose data due to cluster-wide power outages. To solve this problem, we can weigh between the replication groups and recovery time. Generally, fewer replication groups to reduce the possibility of data loss. It turns out that greedy algorithm is much more suitable to generate replication groups in the practical use. However, we found that with long time running of this algorithm, the result of generation is getting worse and divert from the original setting. So we introduce a merging algorithm which can be run periodically. This merging algorithm helps to neutralize the side effects of dynamic cluster changing and reduce the increasing number of replication groups. Then we evaluate the merging cost of this algorithm and give several suggestions of scheduling the merging.

Motivation/background:

What problem is the project trying to solve?

Random copyset replication use a static algorithm which is not good for practical use. Ring Placement is restricted to tuning variables. Tiered Replication use greedy algorithm to generate copyset groups. However, in the case of dynamic cluster, Tiered Replication tends to get worse on the number of copyset.





In real case, the number of nodes in the data center will dynamically changing. This changing will mess up with the generated groups. In the long run, the original purpose of the algorithm which is reduce the replication group will not take effect.

Why is this problem interesting/important?

There is a trade off between the number of replication group, recovery time and data loss possibility. To achieve optimal performance, all of these variables needs to be tuned with the practical case.

This problem is interesting because most of the existing system can not survive in cluster-wide power outages. Data availability is the most important consideration of cloud computing.

Design: Proposed design to solve the problem.

The algorithm we propose consist of two steps:

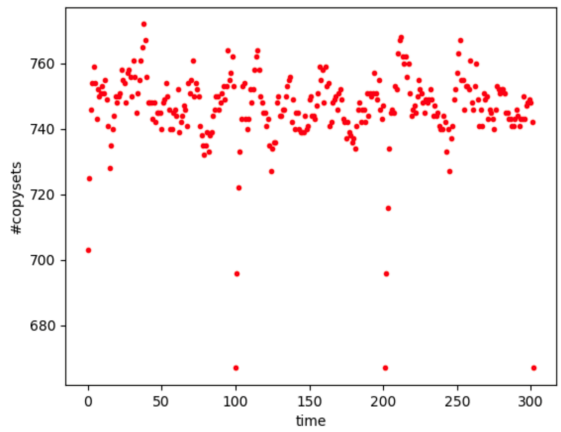
First, kick over-scattered nodes out of its copysets. Each node only belongs to P copysets, should leave other remaining copysets

Second, merging half-full copysets into full-sized a copyset.
Sort the half-full copyset by size, and merge from two ends: start from the biggest copyset and merge with smallest one.If two copysets can merge into a copyset greater than full-size(e.g. 3) size, merge them into one full-size copyset AND another half-full copyset (e.g. 2+2 --> 3+1)

What are some of the design choices you made and why did you make them?

Why is this the appropriate design to this problem?

Evaluation: How good is the proposed design in solving the problem? How does it compare to alternative approaches?'



Related work: what is prior work and why is this project new compared to prior approaches?

Related work:
Copysets: Reducing the Frequency of Data Loss in Cloud Storage
Tiered Replication: A Cost-effective Alternative to Full Cluster Geo-replication