A SUPPLEMENTARY

A.1 Implementations in IoTDB

All algorithms we presented in this paper have been fully implemented in Apache IoTDB, an open-source time series database system. The corresponding Java classes are Data allocation table [N7], Data partition table [N8], Greedy CopySet replication algorithm [N9] and Min cost flow distribution algorithm [N10].

A.2 Customers Scenarios

IoTDB has served over 1000 backbone and industrial leading enterprises [N11]. This section presents typical user scenarios.

A.2.1 Changan Automobile. Changan Automobile [N12] is one of the "Big Four" state-owned car manufacturers of China. Changan Automobile built a smart platform for querying massive amounts of connected vehicle data and remote diagnostics based on Apache IoTDB.

A.2.2 Mcc Cisdi. Mcc Cisdi [N13] is one of the core subsidiaries of MCC under China Minmetals, the Fortune Global 500. Mcc Cisdi provides overall solutions for domestic and foreign steel industry customers. Mcc Cisdi built the core part of the Water and Soil Cloud Industrial IoT Platform for data development governance base on Apache IoTDB.

A.2.3 AUTO AI. AUTO AI [N14] is a provider of technologies and integrated hard/software solutions in the field of intelligent cockpit. It uses Apache IoTDB to manage the driving behavior data of all service vehicles, with a total of 1.6 million online vehicles every day.

The real customer workload is shown at Table 3. Taking AUTO AI as an example, we collected its production cluster metric from May 2, 2024 to May 9, 2024 at Table 4. It can be seen that the IoTDB cluster is balanced in both computing and storage since the write throughput and the used disk space fluctuate very little, where the coefficient of variation is less than 4%.

A.3 Experiment Setup

We first conducted the simulated evaluation on a personal Mac computer, which is equipped with 10-core M1 Pro chip and 16 GB memory. Then we deployed an IoTDB cluster consisting of 11-nodes on Tencent Cloud for processing the IoTDB evaluation, which covers the typical cluster size as shown in Table 3. Each node is equipped with a 4-core CPU, 8 GB memory, 100 GB SSD storage and 1.5 GB ethernet. Of these 11 nodes, 8 are used to deploy DataNodes, 2 are used to deploy IoT-benchmark, and the remaining 1 is used to deploy ConfigNode.

A.3.1 IoT-Benchmark. IoT-benchmark [N15] is an open-source benchmark for TSDB in IoT scenario, which we used for generating periodic time series data according to the configuration and send the data batch by batch to IoTDB cluster. We configured IoT-benchmark to simulate sensors with different load, configured batch size to 100, and assembled the data generated by every 100 sensors into one write request, with the average size of each write request being 60 KB. Finally, the IoTDB cluster is written about 40 million data points per second. The benchmark load covers real customer scenarios as shown in Table 3.

Table 3: Customer workload

Enterprise	Changan Automobile	Mcc Cisdi	AUTO AI
Cluster Size (DataNodes)	3	3	6
CPU core (per DataNode)	64	8	16
Memory (per DataNode)	256 GB	32 GB	128 GB
Disk (per DataNode)	10 TB	2 TB	1.2 TB
Time Series	1.2 billion	2 million	64.8 million
Write Frequency	Per 30 s	70% per 1 s 20% per 10 s 10% per 0.1 s	Per 3 s
Write Throughput (Data Points)	avg 6 m/s max 10 m/s	avg 3 m/s	avg 1.55 m/s max 3 m/s
TTL	3 months	7 days	5 days
Time Partition	7 days	7 days	7 days

Table 4: IoTDB cluster in AUTO AI

Load Metric	Write Throughput	Used Disk Space
DataNode 1	516113/s	231 GB
DataNode 2	516792/s	235 GB
DataNode 3	521633/s	238 GB
DataNode 4	523417/s	229 GB
DataNode 5	508919/s	244 GB
DataNode 6	510220/s	252 GB
Mean	516182.33	238.17
Standard Deviation	5843.49	8.61
Coefficient of Variation	1.13%	3.62%

A.3.2 Alternative Algorithms. We implemented the following DataRegion placement algorithms [N16] to compare with our GCR algorithm. The first one is a Greedy algorithm which always place the new DataRegion to the DataNode with the smallest number of DataRegion since it's a intuitive solution. We then implemented the Copyset [4] and Tiered Replication [3] algorithms. Because the optimization of GCR algorithm in disaster recovery capability is inspired by these two papers. For all DataRegion placement algorithms, we have them generate the same number of DataRegion placement schemes so that each DataNode ends up holding an average of 6 DataRegions, where the load factor 6 is a reasonable amount we tested beforehand to determine the appropriate number of DataRegions hosted by each DataNode.

We implemented the following primary replica selection algorithms [N17] to compare with our CFD algorithm. The first one is a Greedy algorithm which always select the DataRegion on the DataNode that holds the fewest number of primary replica currently as the primary one to act as an intuitive solution. The second one is a Random algorithm which select a random DataRegion as the primary one. We want to simulate the situation where the DataRegionGroup elects the primary replica by itself, and each

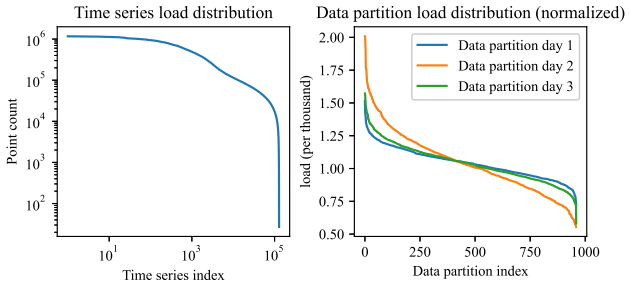

Figure 21: Load distribution of real world dataset

DataRegion has the same weight to be elected as the primary one by using this algorithm.

A.3.3 Metrics. The following metrics point count, used disk space and client requests per second are selected. The point count is used to present the load distribution of various time series in the given real world dataset, as well as the load distribution of data partitions generated through our data partition algorithm. The used disk space is used to demonstrate the ability to keeping storage load balance of our replica placement algorithm. The client requests per second is picked to prove both disaster recovery and computing load balance capability.

A.4 Supplementary experiments

We will redo the Section 6.2 IoTDB evaluation, which we hope can answer the following questions that reviewers are concerned about:

- R2D1: Add the comparison of time series dimension. The number of time series will be increased proportionally after the cluster is expanded in expansion experiment.
- R3O2: Broader deployment and evaluation. As shown in Table 3, most of our customers can meet their business requirement through deploying an IoTDB cluster with not exceeding six DataNodes. Thus, to better demonstrate the expansion capabilities that benefit from our algorithms. We will set the number of DataNodes to 16 and deploy a 19-nodes IoTDB cluster for expansion experiment.
- R1D6, R3D1: Create Unbalanced partitions, using real world dataset. We have configured IoT-benchmark to generate sensors with different load previously. However, since real production environments tend to be more complex, we'll use IoT-benchmark to read a real dataset and then write to the IoTDB cluster.

To sum up, we will redo the IoTDB evaluation mentioned at Section 6.2 by reconfiguring IoT-benchmark to read a real dataset and then write to the IoTDB cluster. Furthermore, we'll start from an 8-DataNode IoTDB cluster and extends it to 16 DataNodes, exponentially increase the number of time series after expansion for the expansion experiment.

A.4.1 Data partition algorithm. Fig 21 (a) presents the diverse load of time series in real production environment. We used three days of real data from one of our customers and showed the count of data points the time series generated. Most time series produce

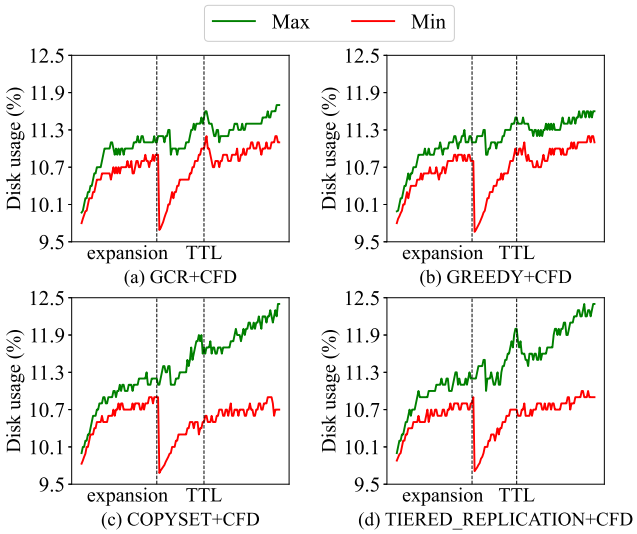

Figure 22: Disk usage distribution after cluster expansion

$(10^3, 10^5)$ data points while some time series are sampling in higher or lower frequencies.

The Data partition algorithm as shown in Sec 3 is first used to equalize the diverse loads generated by the time series. We set the time partition interval to 1 day as an example and Fig 21 (b) shows the load (per thousand) of each data partition generated by the algorithm. Though the load of different time series are various, data partitions generated by the algorithm that belong to the same time partition always have approximately the same load.

A.4.2 Replica placement algorithm. Figure 22 presents the benefit of our replica placement algorithm in storage load balance. We first deploy an IoTDB cluster that consists of 8 DataNodes. And a IoT-benchmark that containing half of the time series in the data set is started for writing data to the IoTDB cluster. After running for a period of time, we expanded 8 more DataNodes, and started a new IoT-benchmark containing the remaining half of the time series to test the capability of our algorithm in the cluster expansion scenario. As mentioned at Sec 3, our data partitioning algorithm does not actively migrate data. Since historical data is gradually deleted over time, the IoTDB cluster is expected to reach storage load balance again after the expansion and TTL time. Figure 22 shown the minimum and maximum disk usage of all DataNodes in the IoTDB cluster. Both GCR and GREEDY achieves the best storage load balance. While the COPYSET and TIERED_REPLICATION algorithms varies the disk usage greatly. Because of their randomness, different DataNodes tend to hold different numbers of replicas.

Fig 23 shows the benefit of our GCR algorithm in disaster recovery capability. We deployed a 8 DataNodes IoTDB cluster and keep writing data to it. A random DataNode is then shutdown, and the client requests per second is used to evaluate the computing load distribution. As shown in Fig 23 (b), only the GREEDY algorithm can't distribute the computing load evenly. Because it tends to form fixed DataNode pair as the Fig 3 presents. The GCR, COPYSET and TIERED_REPLICATION algorithms makes the cluster maintain a

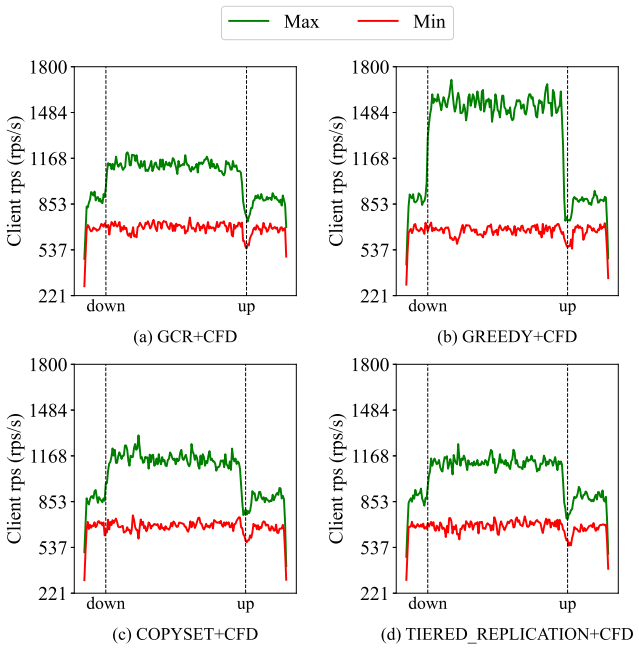

Figure 23: Client rps distribution in disaster scenario

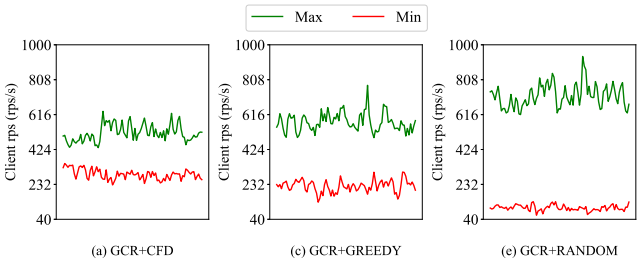

Figure 24: Client rps distribution in disaster scenario

better balance of computing load by giving the ability to equalize the primary replicas in case of node failure.

To sum up, our GCR algorithm has the best ability to maintain cluster computing and storage load balance across different production environments.

A.4.3 Primary replica selection algorithm. An 11 DataNodes IoTDB cluster is deployed for evaluating the computing load balance capability of our primary replica selection algorithm as shown in Fig 24. Comparing with the Random and Greedy algorithms, our CFD algorithm reaches the lowest std in client requests per second. This metric counts the number of client write requests handled by the DataNode per second, and client write requests are always delivered to the primary replica of the relevant DataRegionGroup. Our algorithm always guarantees the most even distribution of primary replicas and thus achieves the best results.

[N7] Data allocation table. https://github.com/apache/iotdb/blob/master/iotdb-core/confignode/src/main/java/org/apache/iotdb/confignode/manager/load/balancer/partition/DataPartitionPolicyTable.java
[N8] Data partition table. https://github.com/apache/iotdb/blob/master/iotdb-core/node-commons/src/main/java/org/apache/iotdb/commons/partition/DataPartitionTable.java
[N9] Greedy CopySet replication algorithm. https://github.com/apache/iotdb/blob/master/iotdb-core/confignode/src/main/java/org/apache/iotdb/confignode/manager/load/balancer/region/GreedyCopySetRegionGroupAllocator.java
[N10] Min cost flow distribution algorithm. https://github.com/apache/iotdb/blob/master/iotdb-core/confignode/src/main/java/org/apache/iotdb/confignode/manager/load/balancer/router/leader/MinCostFlowLeaderBalancer.java
[N11] User cases. https://www.timecho-global.com/product
[N12] Changan Automobile. https://en.wikipedia.org/wiki/Changan_Automobile
[N13] Mcc Cisdi. http://www.cisdigroup.com.cn/html/2/abou
[N14] AUTO AI. https://www.autoai.com/en
[N15] IoT-benchmark. https://github.com/thulab/iot-benchmark
[N16] Alternatives of replica placement algorithms. https://github.com/CRZbulabula/iotdb/tree/elastic-storage/iotdb-core/confignode/src/main/java/org/apache/iotdb/confignode/manager/load/balancer/region
[N17] Alternatives of primary replica selection algorithms. https://github.com/CRZbulabula/iotdb/tree/elastic-storage/iotdb-core/confignode/src/main/java/org/apache/iotdb/confignode/manager/load/balancer/router/leader