ISSN 1816-6075 (Print), 1818-0523 (Online)
Journal of System and Management Sciences
Vol.12 (2022) No.1, pp.175-188
DOI:10.33168/JSMS.2022.0113

Replicated Data Management using Scaled Segment Chain in Unstable IoT Environments

Siwoo Byun

Anyang University, Anyang, South Korea

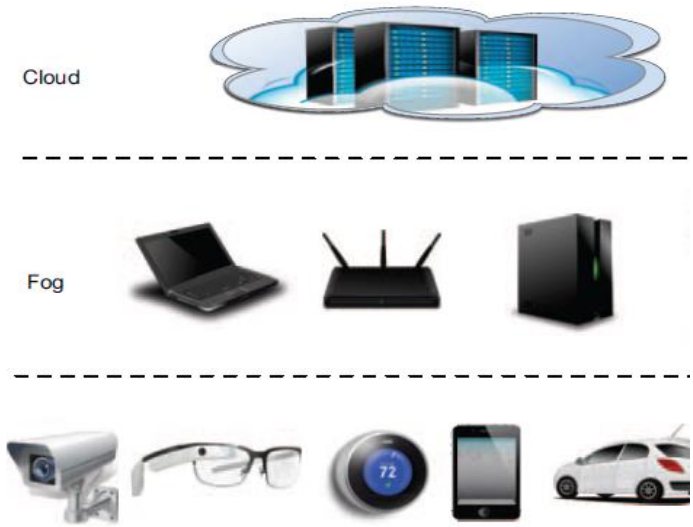
swbyun@anyang.ac.kr

Abstract. IoT edge gateway reduces cloud computing's overload that redirect sensor data to remote servers. For reliable and efficient IoT gateway, column-based flash memory has become a reasonable storage due to its space efficiency and compression performance. This paper introduces recent IoT network and edge computing technology. It proposes efficient replication management called Context-mapped Segment Submirroring to support stable data services for sensor data in the edge-based IoT environment. Sensor context scaling and chained segment submirroring schemes are presented to improve the reliability and performance using IoT edge gateway. In the chained submirroring scheme, the sensor data are kept in the space-efficient storage of IoT edge. Consequently, sensor data transmission and mirroring storage cost can be minimized. The simulation results show that the proposed scheme outperforms the traditional scheme in respect of operation throughput and its response time.

Keywords: Edge gateway, sensor data, submirroring, chained compression, IoT data management, Edge computing

1. Introduction

In Internet of Things (IoT) environments data offloading is required to decrease the volume of database storage and communication to the remote servers (Ashwini,2019), (Lee, 2020), (Park, 2019). In IoT edge computing environment, useful computing resources are placed on the edge of the IoT sensor network and very close to end-



devices such as IoT sensors (Figure 1). Therefore, IoT computing resources should be placed close to end-devices to reduce data traffic and latency. That is, a lot of smart IoT services can be provided to process and store data close to end-devices that generate the IoT big data (Wei, 2017), (Gopika, 2018), (Amir, 2018).

Fig. 1: Concept of IoT edge-based computing

Similar applications prior to the IoT edge technology includes content delivery or distribution networks (CDNs) that provided video content close to the user. Information-centric networking is also an improved form of content-based routing technology like CDN. In the future, IoT edge computing is considered to provide effective data filtering and a representative platform for processing big data from IoT devices. This means that big data should be optimized before it goes to the cloud to reduce communication costs and save energy of end-devices in the future. In addition, the IoT edge computing is also tightly coupled with key network technologies such as 5G which is characterized by low-delay, stable-communication, energy-efficiency, and high-connectivity to various devices.

Several approaches have been proposed to implement edge computing. The resource-rich model approach connects resource-rich servers to virtual machine-based 'cloudlet' directly after the end-devices via wireless access point. Cloudlet is a 'mini data center' which has multi-core CPU, flash storage, and wireless LAN to IoT edge. For example, sensor data and video collected through Goggle Glass are

processed on cloudlet for real-time services. Cloudlet can be used for LTE base stations and advanced vehicles in the future and can be expanded to a set of servers called micro-cloud.

The heterogeneous edge model leads the edge platform to increase the utilizations of various computing resources. That is, it integrates various devices such as IoT servers, access points, set-top boxes, and smart phones. It also provides various communication requirements such as wireless connectivity, routing, streaming, sharing, and mobile access. Examples of this include a mini-cloud clustered with Raspberry Pi and a mobile-cloud composed of various mobile devices. For example, mobile learning, mobile games, natural language processing, and mobile health care can be operated smoothly on end devices with poor resources.

Some of the features analyzed in the actual IoT application cases are as follows.

a) A large portion of sensor data sent from IoT devices has bandwidth-intensive characteristics. Placing computing resources right next to high-bandwidth data sources and processing them within the region, can reduce the amount of data sent to distant servers.

b) Low-latency network is important to many applications such as smart cars and IoT home network. They should process sensor data quickly to ensure real-time action and responsiveness.

c) Geographic distribution is an important characteristic in IoT sensor network. One realistic example of this edge computing is a collision avoidance system. This edge platform includes local intelligence and low-latency communication to vehicle networks and smart road facilities, and utilizes various sensor data such as location, speed, and acceleration. These sensor data should be processed locally with less delay than in long-distance clouds.

The column-oriented database is the opposite database of the traditional record-oriented storage. That is, the column-oriented storage saves sensor data values vertically, so sensor data values are basically clustered for effective data compression and fast data retrieval. Using the column-based database storage rather than general database storage for the IoT sensor data is more advantageous for data compression and data transmission (Byun, 2016), (Ahn, 2013), (Abadi, 2008).

2. Related works

2.1. IoT component model

Sensor data consists of measured data such as values of light, temperature, and humidity and light, as well as non-sensory data such as predefined identification data and default configuration. These sensor data are frequently used for cloud analysis after being collected from the sensor nodes. To store sensor data, sensor nodes use flash memory owing to the low price, compactness, portability, and stability.

An IoT component model consists of a sensor node, an sensor gateway, and a cloud server, each meaning a sensor data source, a IoT edge gateway, and sensor database server (Karrar, 2018).

a) *Sensor node:* IoT sensor node sends large volume of measured sensor data to IoT gateway. These sensor nodes and gateway devices are all interconnected to serve various IoT sensor monitoring services.
b) *Sensor gateway*: IoT sensor gateway forwards the measured sensor data to cloud servers. The gateway system preprocesses large volume of sensor data in the course of filtering and offloading process to remove data redundancy and unnecessary transmission cost.
c) *Cloud server*: Reliable cloud servers have sufficient computing resources such as larger main memory, multi-core processor, security software and big database system for secure IoT services.

2.2. Reliable data management for IoT sensor networks

Because IoT sensor network has unstable wireless channel and limited bandwidth, the IoT users could suffer from unsafe and slow IoT services. One way to reduce the likelihood of undesirable services is to copy sensor data to many sensor devices. The example of critical data replication may be a human location detected from body heats in disaster areas or battle fields.

In an IoT sensor applications, replicating sensor data can contribute to high level reliability where each independent sensor node can use its copy if sensor node fails or wireless network fails for a long time. Moreover, the actual operation cost of wireless communication is very expensive compared to the other operations. Thus, data replication and recovery scheme is more required in the sensor network environment rather than in the general distributed network environment.

In general database management, data recovery refers to the function of restoring important data to its pre-fault state in the event of a disaster such as a physical failure and incorrect operation. The disaster-oriented failure can be recovered using backup storage devices such as hard disks. The user-oriented failure can be recovered by redoing/undoing some tasks to the closest backup state from the failure point. In general database, data manager should access the actual storage via the recovery manager.

The simplest recovery way to control data copy is read-one write-all method. However, since write transactions cannot be allowed after one single node failure, this scheme naturally limits the data availability (Bernstein, 1987). To cope with the low availability problem, the voting methods were developed. In voting methods, write transactions should have [n/2+1] of copies, instead of all copies in read-one write-all scheme (Pasojevic, 1994).

The key idea of voting has become generalized to general quorum-consensus method. In the quorum-consensus architecture, a replicated data access must establish

a quorum before an actual read or write. The quorum-consensus methods are recognized as a well-known abstraction technique for ensuring data consistency in secure distributed systems (Lin, 1997), (Marko, 2012). In recent years, the quorum-based skill has been developed into robust loggers (Edson, 2017)and block chaining applications (Christian, 2017).

2.3. Proposed IoT replication control

2.3.1. Edge-based sensor data mirroring

This study proposes a technique called Context-mapped Segment Submirroring (CmSS) using IoT edge gateway that improves general mirroring techniques for improving storage performance and reliability of sensor data recovery. Several RAID techniques (Byun, 2019) were also considered in the design phase, but the practical choice for edge gateway is considered to be mirrored RAID, because the computation and the space overhead of other techniques was too large to endure.

The proposed technique also keeps replicated data in different location like RAID. However, instead of using the same copy of the original data, the edge gateway exploits variably scaled data after column-wise compression is performed block by block.

The differences between this technique and general mirroring are as follows. First, whereas normal mirroring keeps the replica inside its sensor node, this technique sends the replica to the edge gateway connected to the local network. Second, the replicated sensor data can be compressed unlike the original mirroring. Thus, the researcher uses the term ‘submirroring’ instead of mirroring. Third, the replicated sensor data can be stored at a lower resolution than the original, considering the limited computing power of the sensor node. Fourth, the replication timing of the sensor data can be a variable depending on the characteristics of the sensor, rather than being performed immediately.

For the operational efficiency, sensor data is generated every second in the sensor node but is aggregated on the edge every minute and uploaded to the cloud every hour. This is because the large amount of sensor data generated must be filtered or aggregated through the edge gateways to reduce data transmission overhead in the first phase. In addition, the data required in the cloud needs to be refined for the same reason.

The first reason for mirroring on IoT edges is the instability of the sensor node. Second, there is too much data to mirror in the cloud. Therefore, it is necessary to reduce it to a transferable level through data processing such as filtering on the edges. Third, the edge gateway is connected to the sensor nodes by local network, so the communication load is relatively low and stable. Fourth, compared to sensor nodes, IoT edge gateway has stable hardware and power supplies and not exposed to the outside damage, thus ensuring the stability of the data. However, efficient storage control techniques are needed, since the amount of sensor data must be significantly

reduced and mirrored to run on resource-poor small gateways compared to general IoT servers.

2.3.2. Scaled context mapping and variable transmission

The purpose of using sensors is mostly to check that the desired devices are functioning normally within the acceptable range or where abnormalities have occurred. Therefore, it is not necessary to send too much detailed values to the server if the values are within acceptable range. In a typical case, statistics of numerous sensor values or sensor information where anomalies occur are important. Therefore, it is inefficient to divide the range of sensor values into equally spaced areas. That is, it is much more efficient to divide the range of sensor values into multiple variable areas and send only the code values of those areas. This means that the volume of data storage and communication can be greatly reduced by sending variably scaled code rather than fixed scaled data.

In multi-scale related research (Marco, 2019), mathematical, computational, and statistical methods were used together. This study expands multi-scaling using variable scale mapping that considers the characteristics of sensor context, compresses it on a column basis, and utilizes it for submirroring as shown in Figure 2.

For simple example, there are 100 temperature sensors in a particular workshop and the normal range of sensors is 10-40 degrees and can be reported with a precision of 5 degrees. However, below or above this range is defined to be abnormal and should be reported within a precision of 1 degree. The total sensing range is 1 to 50 degrees.

For this case, representing the entire temperature area of the Scaled Context Map (SCM_temp) in order of (#area code, temperature range) is shown below, meaning that data range is compressed to 26/50 and can be reduced by almost half.

SCM_temp = { (#1,1), (#2,2), ..., (#10,10), (#11,11~15), (#12,16~20), (#13,21-24), (#14,25~30), (#15,31~35), (#16,36~40), (#17,41), (#18,42), ..., (#26,50) }

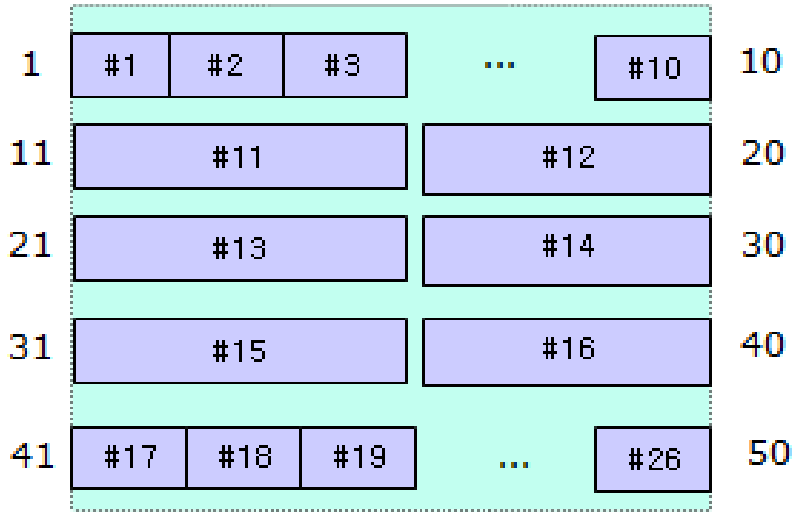


Fig. 2: Example of multi-scaled context mapping

In addition, it is possible to reduce the transmission volume using variable-cycle transmission. If the sensor data is in the normal range, it can be sent on a three second basis, and if the data in the non-normal range is sent on a one second basis, the amount of data transmission can be reduced by a third. Additionally, data transfers can be further reduced by inhibiting repetitive data transfers or by excluding previous similar data from the transfer data sets.

2.2.3. Segment-chained compression and submirroring

CmSS exploits column-based database skill to maximize data compression. This study tested the efficiency of compression based on network flow data (Table 1). It uses the Lzo API module (Lzo, 2020) for column-based decompression or compression, and the lzo performance test showed that the compressed ratio is about 20%. Due to the high locality of column-based sensor data, the capability of data compression and decompression is inherently good, especially for text type and code type data. The compressed size of 10,000 sensor data is less than 33KB. The columns associated with queries are selected randomly by workload generator.

The column-by-column compression procedure consists of multi-chain concatenation that uses small-size column segments rather than full-size one segment to minimize compression and decompression overhead.

The decompression and compression costs of small data segments are essentially reduced compared to one large segment. Since CmSS can move to the beginning of the original column and only small segments can be decompressed, the access speed is greatly improved. Since recent multi-core processors are much powerful than normal storage devices, the overhead of segment decompressing is negligible. The system architecture of CmSS is described in Figure 3.

Table 1: Compression test of IPTV NetFlow

Test	*CliRttTm*	*SvrRttTm*	*FstFinTm*	*SessStartTm...*
Lzo Time for Compression	30 ms	36 ms	40 ms	82 ms
Lzo size of compressed data	699 Bytes	1,562 Bytes	1,782 Bytes	32,200 Bytes
Example.1	0.072	16.182	183.168	2021-02-01 10:53:05.000000
Example.2	0.076	16.179	183.192	2021-02-02 17:15:43.000000

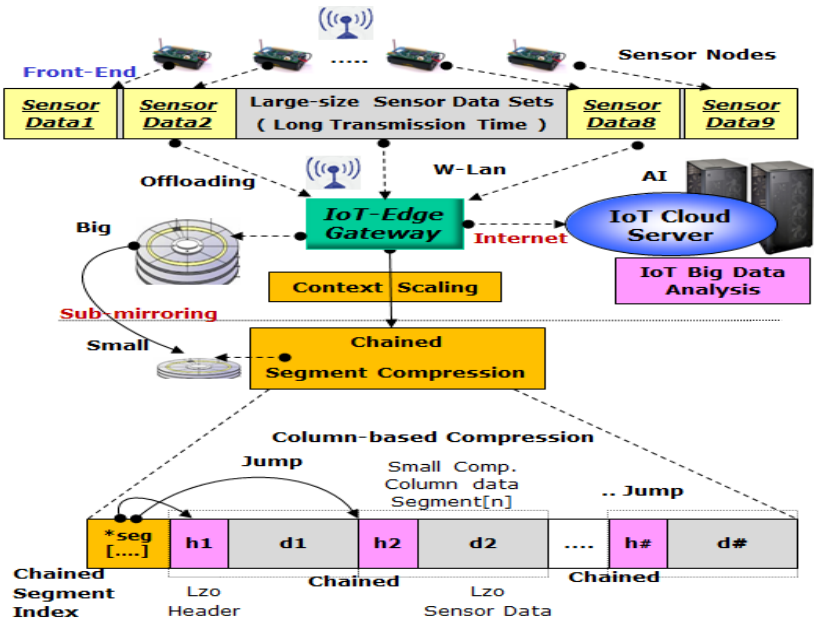


Fig. 3: Architecture of context-mapped segment submirroring

The chained segment indexing method divides the original node into several short segment which can reduce the decompression workload. Then, it connects the short segments together and stores them in a built-in jump table (chained segment) that can go directly to the detailed location as needed. Decompressing a longer index node requires longer restoration time.

For this built-in index configuration, one small index header is added to the segment area, and thus compression time may take a little longer. However, in case of retrieving data from a leaf index node, the data in the desired column data can be fetched directly from the segments. Then, only the segment can be unzipped without

wasting time to find it from the starting point of original long cluster. As a result, data search operations can be performed efficiently.

2.4. Computer simulation

2.4.1. Simulation model

The researcher compared the performance of three types of data mirroring. The basic original mirroring scheme is denoted as OrgMr, shortly in the test. As mentioned above, CmSS scheme exploits both submirroring and chained segment for space-efficiency. The CmSS version only with submirroring is denoted as SubMr.

In this study, simulation system was programmed using C++ and CSIM (Csim, 2020) for workload generation and performance analysis. The key performance metrics are response time and throughput rate for mirroring operations. The response time is the time that takes between the request and the execution of the mirroring operations, and the throughput rate is the counter of mirroring operations processed per second.

The experimental system consists of a Mirroring Operation Generator(MOG), a Mirroring Manager (MM) for the operation, a Compression Manager (CM), and a Segment Manager (SM), as shown in Figure 4. The MOG generates mirroring operation at predefined intervals to make the system workload required for the simulation. The MM manager manages and analyzes the operation, and then sends its data to CM. After compressing the data using Lzo algorithm, the CM sends the compressed data to SM. The SM makes chained segments and saves them in the file system.

2.4.2. Simulation results and interpretation

The computer simulation was performed to check the effect of the mirroring operation in which number of IoT sensor node (No_ISN) increases. The result graphs for the three schemes, OrgMr, SubMr, and CmSS are as follows.

The overall search throughput is shown in Figure 5, and response time is shown in Figure 6. In Figure 5, the *x* axis is the mirroring input which is the number of generated mirroring requests per second. Thus, changing the value of generated mirroring requests means the workload variation of mirroring operations from IoT sensor nodes. The *y* axis is the transaction processing results which is the counter of finished mirroring operations per second. In Figure 6, the overall response time smoothly increases with the mirroring request workload. The main reason of this incensement is the mirroring contention that caused by the increment of the simulation workload. In the result of this simulation, the best performance is shown by CmSS, followed by SubMr and OrgMr.

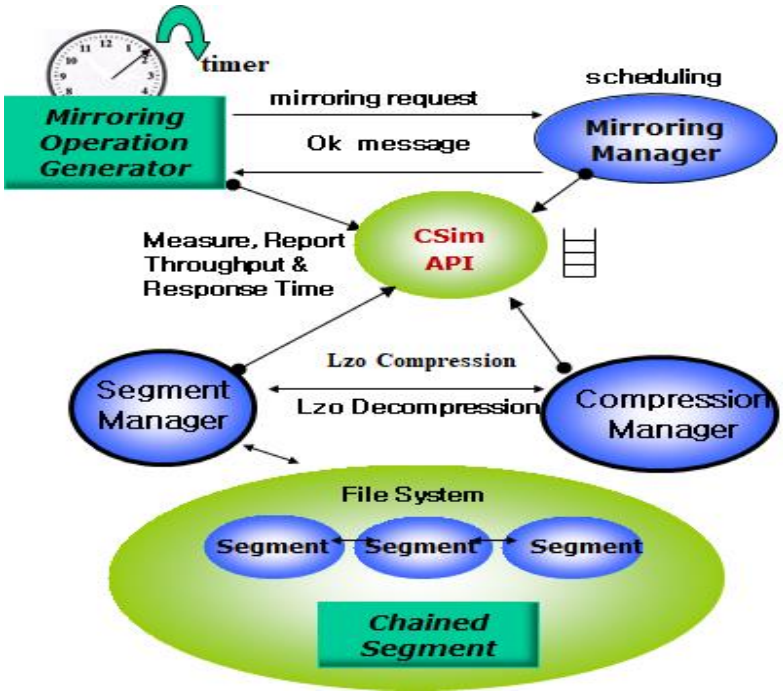


Fig. 4: Simulation model for performance evaluation

Figure 5 shows that the throughput of each scheme decreases beyond No_ISN of 120 to 160. Although the value of No_ISN has been increased beyond 120, the number of active mirroring requests that are currently being processed in the simulator appears to be reduced slightly. This reduction means that adding more mirroring requests beyond 120 to 160 simply leads to increasing the mirroring operation overload, not necessarily incurring an enhanced level of performance. From this simulation, the researcher can claim that the throughput rate of the mirroring operation are essentially influenced by the factor of mirroring workloads such as data scaling and segment compression.

Figure 6 shows that the throughput rate of SubMr begins to be degraded as No_ISN increases beyond 200, although SubMr shows enhanced throughput rate than OrgMr by shrinking the mirroring volume by the data scaling. This fact means that SubMr also suffers form the negative effect of mirroring overload under the high contention condition.

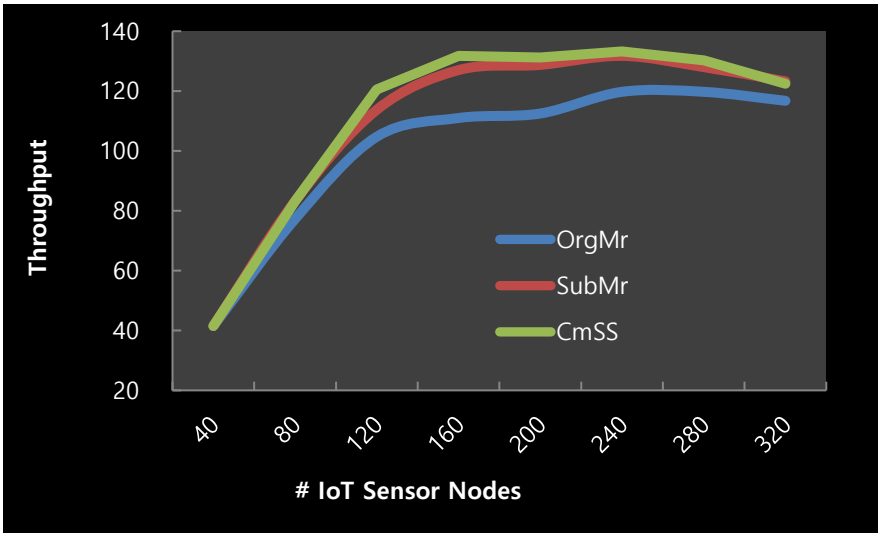


Fig. 5: Mirroring Throughput Rate Test

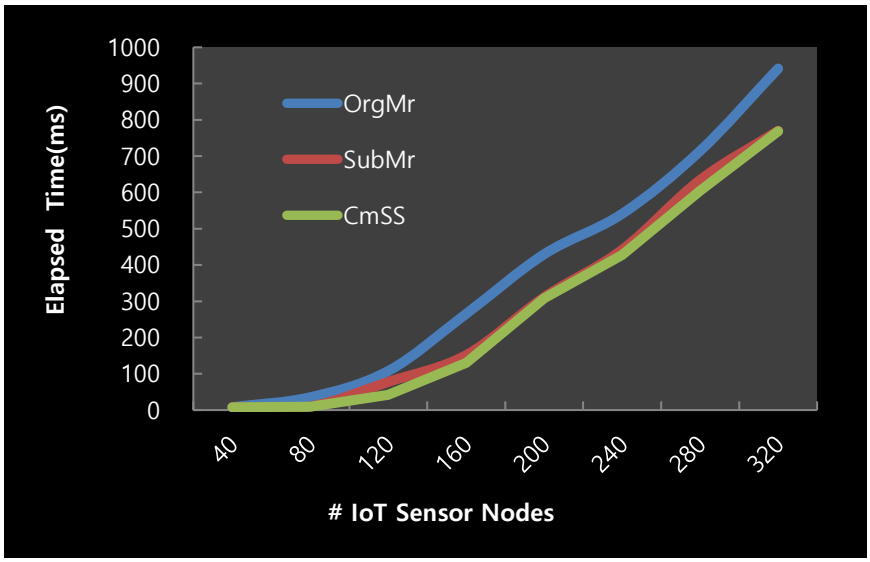


Fig. 6: Mirroring response test

CmSS shows slightly higher throughput rate than SubMr under the high contention condition. This means that CmSS successfully lessens the degree of Lzo compression in flash memory using chained indexing and small-size segmentation skills.

In Figure 6, the researcher observed that the mirroring response of OrgMr and SubMr increases slightly as the number of the mirroring requests increases. At the overall range of mirroring requests from IoT sensor nodes, the response time of CmSS is lower than that of OrgMr and SubMr. With respect to the mirroring throughput rate,

the mirroring operation gain of CmSS relative to OrgMr reaches about 10.3%. In terms of mirroring response, the time gain reaches to 25%. This achieved difference means that a large portion of mirroring gain in CmSS over OrgMr comes from the positive effect of sensor data scaling and index segmentation to reduce mirroring workload.

5. Conclusions

This study introduced recent IoT network and edge computing technology. It also proposes a technique called context-mapped segment submirroring using IoT edge that improves general mirroring techniques for improving storage performance and safety of sensor data.

To operate on small edge gateways that lack computing resources compared to resource-rich general computers, the volume of sensor data should be reduced and mirrored. While normal mirroring keeps the same copy of the original sensor data, proposed technique exploits variably scaled data and chained compression segment by segment. To minimize sensor data transmission and mirroring storage cost, the sensor data are kept in the column-based flash storage of IoT edge. Therefore, the replicated sensor data can be stored efficiently, considering the limited space and computing power of the sensor network.

There are small drawbacks in implementing the scaling and chained segment where the data scaling and segment compression processes essentially lead to additional system overhead, although it saves the mirroring space. The simulation results show that the proposed mirroring scheme using chained segment submirroring in column-based flash storage, outperforms the traditional scheme in respect of mirroring response time and its throughput rate.

References

Ashwini, L. K., Lee, H. & Hwang, M. (2019). Implementation of IoT Application using Geofencing Technology for Protecting Crops from Wild Animals. *Asia-pacific Journal of Convergent Research Interchange*, 6(6), 13-23.

Lee, Y. G. (2020). A Study on Development of an Integrated IoT Service Platform Using Spatial Information. *Asia-pacific Journal of Convergent Research Interchange*, 6(9), 73-80

Park, H. D. & Kim, S. G. (2019). A Study on Monitoring and Control Architecture for Smart Lighting System in IoT Environment. *Asia-pacific Journal of Convergent Research Interchange*, 5(3), 91-100.

Wei, Y., Fan, L., Xiaofei, H., William, G. H., Chao, L., Jie, L. & Xinyu, Y. (2017). A Survey on the Edge Computing for the Internet of Things. *IEEE Access.* 6, 6900-6919.

Gopika, P., Mario D. & Tarik, T.. (2018). Edge Computing for the Internet of Things: A Case Study. *IEEE Internet Of Things Journal*, 5(2), 1275-1284.

Amir, M. R., Nguyen Gia T., Negash, B., Anzanpour, A., Azimi, I., Jiang, M. & Liljeberg, P. (2018). Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach. *Future Generation Computer Systems*, 78(2), 641-658.

Byun, S. & Jang, S. (2016). Asymmetric Index Management Scheme for High-capacity Compressed Databases. *Journal of Korea Academia-Industrial*, 17(7), 293-300.

Ahn, S. & Kim, K. (2013). A Join Technique to Improve the Performance of Star Schema Queries in Column-Oriented Databases. *Journal of Korean Institute of Information Scientist and Engineers*, 40(3), 209-218.

Abadi, D., Samuel, R. and Madden, N.. (2008). ColumnStores vs. RowStores: How Different Are They Really?. *Proceedings of the ACM SIGMOD'08*. Vancouver, BC, Canada, 967-980.

Karrar, A. & Fadl, M. (2018). Security Protocol for Data Transmission in Cloud Computing. *International Journal of Advanced Trends in Computer Science and Engineering*, 7(1), 1-5.

Bernstein, P., Hadzilacos, V., & Goodman N. (1987) Concurrency control and recovery in database systems, *Addison-Wesley Press*.

Pasojevic, M. & Berman, P. (1994). Voting as the optimal static pessimistic scheme for managing replicated data. *IEEE Trans. Parallel Distrib. Syst*. 5(1), 64-73.

Lin, X., (1997). A fully distributed quorum consensus method with high fault-tolerance and low communication overhead. *Theor. Comput. Sci.* 185(2), 259-275.

Marko V.. (2012). Quorum Systems: With Applications to Storage and Consensus. *Synthesis Lectures on Distributed Computing Theory*. Morgan & Claypool Publishers

Edson Tavares de Camargo, Elias P. Duarte Jr., Fernando Pedonez. (2017). A Consensus-based Fault-Tolerant Event Logger for High Performance Applications, *Euro-Par 2017: Parallel Processing.* 415-427.

Cachin, C. & Vukolic M. (July 2017). Blockchain consensus protocols in the wild", Technical Report. arXiv:1707.01873, IBM Research - Zurich, 1-24.

Byun S. (2019). Modeling and simulation of the redundant array of inexpensive/independent disks storage for internet of things monitoring servers. *International Journal of Electrical Engineering Education*. 58(2), 156-167.

Marco S. Reis, (2019). Multiscale and Multi-Granularity Process Analytics: A Review. *Processes* 7(2), 61.

Lzo, (2020). Professional data compression,
http://www.oberhumer.com/products/lzo-professional.

Csim, (2020). Introduction to CSIM Modeling,
https://www.csim.com/overview.html