CRS: Consistent Replica Selection Algorithm for Distributed Key-Value Storage

Thazin Nwe
Faculty of Computer Sciene
University of Computer Studies, Mandalay
Mandalay, Myanmar
thazinnwe@ucsm.edu.mm

Tin Tin Yee
Faculty of Computer Sciene
University of Information Technology
Yangon, Myanmar
tintinyee@uit.edu.mm

Ei Chaw Htoon
Faculty of Computer Sciene
University of Information Technology
Yangon, Myanmar
eichawhtoon@uit.edu.mm

***Abstract*— The communication between data accessibility and consistency becomes a challenge of replica selection for a Key-Value Storage (KVS). Applying a fixed replica selection policy in numerous data centers may decrease the capacity of a data store. Dynamic scaling advances the KVS with dynamical addition or removing data nodes. To expand the effective addition of the data nodes, the consistent hashing method is applied in KVS because of the adjustability of node variants. The benefits of the hashing algorithms are that scalability, direct control, and adaptation to node transfers while it is not entirely restructuring data layout. The three algorithms are proposed in this paper for selecting the consistent replicas according to the order of the data nodes id's hash values, total Round Trip Time (RTT), and Probability of Bounded Staleness (PBS) among the replicas in a datacenter. By selecting the appropriate node, it can reduce the write execution time for regenerating the data. Without applying the random of the existing consistent hashing technique, it is effective in reducing latency cost, memory cost, and replication cost. The experimental results state that the node selection procedure significantly improves the throughput and consistency rate.**

Keywords*— *Key-Value Storage(KVS), Consistent hashing, Probability of Bounded Staleness (PBS), Round Trip Time (RTT)

I. INTRODUCTION

Essentially, KVS is a well-known strategy for data-intensive systems such as social webs, e-business, and enterprise because it is proficient in strongly handling big data traffic. Replica and data managing become an issue to provide consistency and accessibility of KVS [1]. Most of the latency-sensitive applications use eventual consistency. An approach from eventual consistency is implemented so that the master replica can solve the bottleneck of data transfer. However, eventual consistency does not guarantee to return of the last updated value. Such systems usually use weak consistency, which proposes them likely to decide fewer replicas in the read request for the performance, which points to the probability of becoming a stale replica. Inconsistency possibility varies on network transmission, system capacity, the number of replicas, high operation latencies, and bad performance.

In replicated KVS, consistency can be implemented by configuring read and write transactions to contact a quorum replica. Most of those KVS systems normally choose a static number of replicas of the read and write transactions in KVS. Quorum-based replication can solve this static replica selection problem to assure consistency. Data consistency and data availability are important for quorum replication. In this quorum system, the positions of the replicas are defined, and a set of replicas is selected to be a participant in the choosing. The master replica is needed to send a message from the chosen replicas and wait for acknowledgment from a set of quorum replicas. Read/write requests are followed according to their position of the replicas in the storehouse domain that varies the latency of the service.

Therefore, assigning a static replica selection strategy in several data centers may reduce the ability of the storage system such as write execution time, latency cost, throughput, staleness rate, etc. In the proposed system, dynamic consistent replica selection is offered to increase the performance of KVS with data nodes with dynamism adding or leave-taking. To improve the effective scaling of the data nodes, an enhanced consistent hashing approach is proposed in KVS with the adjustability of node variations. Therefore, this method is adjustable to definite replicas configurations and the user's requirement, such as changes in the network administration or server capacity.

This paper is arranged as follows. The related works for consistent replica selection methods are mentioned in section 2. Section 3 is the background theory of this system. The proposed algorithms are explained in section 4. The environmental setting and experimental results are presented in section 5. Finally, this paper is concluded in section 6.

II. RELATED WORKS

Current replica selecting methods for the KVS has been studied generally and matched their performances. Canini et al. offered a C3 procedure for example a replica collection approach that selects the data nodes depending on the level of the client with arranging the scores method. Although it declined the latency, it could not effort properly in heterogeneous loads [7]. Consequently, Quema et al. presented Heron as a replica evaluation strategy with the estimation of necessary requirements at a critical period. It proposed by defining the history of keys corresponding to significant values. This strategy designated the data node that an arriving request additional reactive than the distinct replicas. Anti-entropy [3] approach that chose the replicas of the up-to-date versions of all keys. It enhanced with prefix trees to create evaluations more rapidly. Experimentations matched with identical random selection with a greedy method. It is an effective system, more accessibility, and high reliability and partition tolerance. J. Paul Walters et al. offered to localize replica at some remote node with randomly relatively than storage it close to the major location. The objective is to develop replicas at random, subject to constraints [6]. The implementation of a datacenter is frequently issued by overworked data nodes due to extremely weighted loads. Some of the existing systems depend on consistent hashing and virtual servers are needed to migrate imbalance of workloads. One key restriction of these outcomes is their lack of capability to apply the dynamic workload variations. Data replication is theoretically uncomplicated and, therefore, often used by KVS providers

for load balancing. Nevertheless, challenges in data replication contain defining the suitable level of replication and maintaining consistency among replicas of KVS.

Unlike the several studies, the cost-effective consistent replica selection algorithm improves the consistent hashing to increase the performance of quorum replication of KVS. By allowing for the position and the read and write admission period of replicas with fine-tuning the read and write consistency level [10], these algorithms choose the suitable replicas for client contact.

III. BACKGROUND THEORY

Several consistent replica selection methods are mentioned for the KVS storage system. The explanation of these algorithms is described as follows.

A. Dynamic Snitching

In Apache Cassandra, a replica is selected by applying dynamic snitching (DS) by default. Completely the data nodes within a Cassandra cluster work in a ring. Each replica within the cluster holds any records. A record is estimated with the combination of more measurements. It determines how secure a replica can obey the application. Therefore, a replica with the most reliable record supports the succeeding request in the least latency. In the state of Cassandra, it utilizes a compound of requested latency and considers the weight of replicas. It handles standardize eased latency and the standardized measurement of replicas for arranging. To determine a score, the execution from the read requests of several replicas is observed completed period, and the most reliable replica is decided on up-to-date execution records.

B. Consistent Hashing

It is a KVS form distributed data store method. Data is placed on replicas of around space. Each replica handles data in the regions on the ring. To reduce the data cost, data is replicated by each virtual node on the right-hand. P2P systems such as Cassandra, with hashing-based distribution, take a great part in a KVS [4]. The output area of a hash function is executed as a specified circular area or "ring" such as the largest hash value wraps around to the least hash value with a consistent hashing. A random value in this space that defines its "position" on the ring allocates an individual node in the system. All data item named with a key is defined to a node with hashing the data object's key to generate its location on the ring. And then it is managed the clockwise of the ring to get the primary node by a position larger than the item's status. Therefore, every node is to be enough for the sector on the ring within it and its predecessor node on this ring. The policy benefit of consistent hashing is that the removal or addition of a node only affects its critical neighbors and different nodes stay uniform. The original consistent hashing method offers some difficulties. The random location placement of every node on the ring serves to non-uniform data and administration of load.

C. Replication Cost

Meaning for replication cost stands the number of replicas. Nevertheless, this meaning cannot decide how long a replica is required at a position in a dynamic replica situation. The cost description studies the usage cost [5].

Cost=

$$\sum_{i=1}(\infty * usage_i + \beta * create_i + \gamma * teardown_i) \quad (1)$$

In the equation, $usage_i$ means the total time, $create_i$ is the number of replicas, and $teardown_i$ is the teardown cost. ∞, β, γ are variables to define the cost per unit, creation cost, and downtime cost. Let these variables be zero, this function depends on the number of replicas.

D. Probability of Bounded Staleness

Probabilistically Bounded Staleness (PBS) [9], which supports the consistency granted by existing eventually consistent data storage is explained. According to PBS <k, t> staleness, PBS<k,t>-staleness is performed in a quorum system consistency with probability $1 - p_{skt}$. It means that as a minimum one value in a read quorum will be acknowledged within k versions and t units of time.

$$P_{skt} = \frac{(N-W)}{\binom{N}{R}} + \sum_{c\in(W,N)} \frac{\binom{N-c}{N}}{\binom{N}{R}}.[Pw(c+1,t) - Pw(c,t)k] \quad (2)$$

IV. CONSISTENT REPLICA SELECTION SYSTEM

The following factors are considered for the design of the proposed system.

A Number of total replicas: Defining the total number of replicas is important to support the good performance of the system. Because the system maintenance's cost will be higher when the number of replicas is high. Moreover, it can lead to unnecessary resources of the system as too many replicas may not provide the availability of the system.

Replicas placement: Arrangement of placing the replicas is also important because it is hard to choose the suitable locations for the balance of the workload system. Therefore, the replicas should be placed for minimizing the job completion time and latency.

Replicas selection: Defining replicas selection is essential because the replicas should be chosen to get the necessary item for job completion. One of the important constraints in selecting replica is response time.

Defining the consistent replicas: Defining the consistent replicas is important to access the items from the consistent replicas. Consistent replicas are based on the execution time and latency cost of the read / write operations.

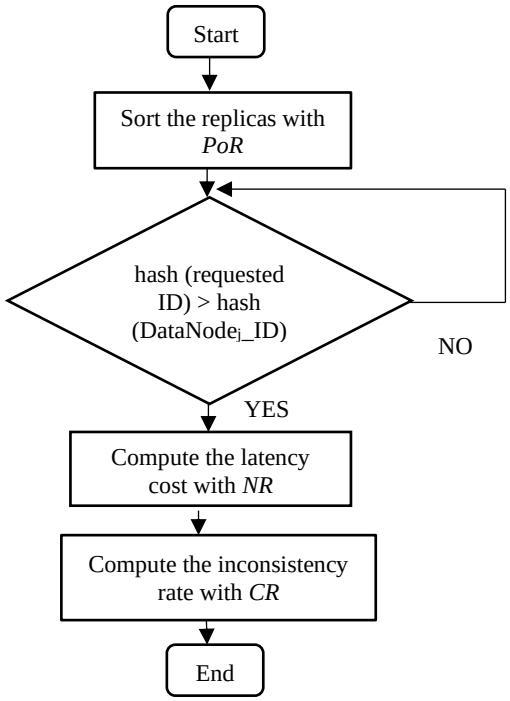

Fig. 1. System Flow for Proposed Architecture

The processes of the consistent replica selection approach for distributed KVS are stated as 3 phases. Firstly, the replicas are ordered in a cluster with the ascending order of the *PoR*. In the second stage, the latency will be measured among several replicas with *NR*. In the last stage, the consistency rate is estimated with *CR*.

A. PoR, NR, and CR algorithms

The proposed *PoR*, *NR*, and *CR* are as follows.

The definition and notation of the proposed algorithms are listed in Table I.

TABLE I. THE DEFINITION AND NOTATION OF THESE ALGORITHMS

Symbols	Definition
ß	Dataset
$DN_1, DN_2 \ldots, DN_n$	Data Nodes defined on the Write Consistency Level
$NR_1, NR_2 \ldots NR_n$	Nearest Replicas
$CR_1, CR_2 \ldots CR_n$	Consistent Replicas
RCL	Read Consistency Level
WCL	Write Consistency Level

Placement of Replica (*PoR*) Algorithm

Input: *ß* (YCSB Workloads), $DN_1, DN_2 \ldots, DN_n$ (Data nodes)
Output: $DN_1, DN_2 \ldots, DN_j$ (no of replicas for *WCL* in ring)
Begin
Define *WCL=n*/2+1
for i from 1 to *n*
nodeHashVal←MD5(DN_i)
*sorted*Hash*Val*← sort(nodeHashVal (DN_i))
end for
for j from 1 to *WCL*
ring←(*sorted*Hash*Val*(DN_j), *ß*, timestamp, version)
end for
return *ring*.get (DN_j)
End

A write request is assigned to the Placement of Replica (*PoR*); the user chooses the most neighbouring node according to the range among adjacent group features in the ring. That most adjacent node is specified as a coordinator node and addresses the request to the other replica nodes. By hashing an IP address, every single data node as a replica has a distinct ID to hold a value. These distinct IDs are assigned to a ring. The hash contents of replicas are arranged into the KVS. A key is assigned on each data and located to the ring and walks left to right nearby the ring from that location until obtaining the initial node ID.

This initial replica node is assigned for the client requested data. Every node is required to take the same value of data to improve the system execution. *PoR* returns the number of replicas for the write process until the Write Consistency Level (*WCL*). *WCL* is initially defined (*N*/2) +1 for the quorum replication factor.

Nearest Replica (*NR*) Algorithm

Input: $DN_1, DN_2 \ldots, DN_j$ (no of replicas for *WCL* in ring)
Output: list of nearest replicas ($NR_1, NR_2 \ldots NR_j$)
Begin
HashVal= MD5(requestedID) // client requested_ID
for i from 1 to j
if (MD5(DN_i) < HashVal) **then**
$NR \leftarrow DN_i$
HashVal← MD5 (DN_i)
else
i++
HashVal← MD5 (DN_i)
end for
compute *RTT* time in *NR*
sortedNodes←sort(*RTT*(NR_j))
return *sortedNodes*.get (NR_j)

End

According to *NR*, a read request is sent by a client to one of the replica nodes in the ring to receive the content. Initially, it finds the nearest node of a coordinator node by comparing the hash values and total Round Trip Time (RTT). The coordinator node selects the most adjacent group of the arranged replica lists with ascending order until getting the initial node ID. *NR* returns the list of the nearest nodes on the ring.

Consistent Replica (*CR*) Algorithm

Input: list of nearest replicas ($NR_1, NR_2 \ldots NR_j$)
Output: list of consistent replicas ($CR_1, CR_2 \ldots CR_j$)
Begin
RCL← 0
for i from 1 to j
Compute consistencyRate of NR_i with PBS
if(consistencyRate== maxConsistencyRate)
then
$CR \leftarrow NR_i$
RCL ++
end if
end for
return CR_j

End

Consistent Replica (*CR*) Algorithm manages a replica selection method to choose a consistent replica of the read request for the quorum replication $R+W > N$.

In the current consistent hashing method [3], a random node is chosen on the MD5 hash of the major key. One of the main differentiation is that the data node is taken associated with the hashing of the demanded key. Therefore, the proposed algorithms could minimize the capability of replica and latency cost. This influences the effect of the consistency rate.

V. Performance Evaluations

In this section, experimental setups for performance evaluations in distributed KVS are described. The proposed *PoR*, *NR*, and *CR* are combined with Apache Cassandra, which is a popular KVS. Yahoo Cloud Serving Benchmark (YCSB) [2], a standardized benchmark is tested to examine the effect of the proposed algorithms.

A. Experimental Setting

The proposed *PoR*, *NR*, and *CR* are evaluated by using an Apache Cassandra cluster on Ubuntu 14.04 LTS i386. Among the five YCSB core workloads such as workload A, workload B, workload C, workload D, and workload E are evaluated for the performance of the proposed algorithms. Table II shows the environmental setting for performance evaluations.

TABLE II. Environmental Setting for Performance Evaluations

Testing Environments	Testing Parameters
Commodity Linux VMs Cluster	Ubuntu 14.04 LTS i386
Hosts Specification	Intel (R) Core (TM) i7-6700 CPU @ 3.40 GHz 8.00 GB RAM 2TB Hard Disk
Software Components	Processing System Apache Cassandra 3.11.4

In Fig 1, 8 replicas (R1, *R*2, …) and clients are setups on VMware to estimate the performance of the cost-effective algorithms.

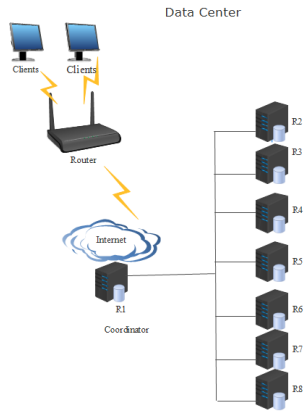

Fig 1. Experimental Setups for a Replica Selection Approach with Apache Cassandra Cluster

```
--  Address         Load       Tokens       Owns    Host ID
                    Rack
DN  192.168.88.248  27.79 MiB  256          ?       23653963-2424-4b59-bb93-540c
18a04cc2  rack1
UN  192.168.88.249  24.69 MiB  256          ?       b3b265c0-71ee-4978-a253-4c41
18a3de8c  rack1
UN  192.168.88.234  8.72 MiB   256          ?       c31885e7-2804-40c9-bf8b-d253
f1efe74b  rack1
DN  192.168.88.235  9.57 MiB   256          ?       f238a499-1a8e-443d-a7a7-264c
ce09dd39  rack1
DN  192.168.88.251  10.33 MiB  256          ?       4dfc56de-260d-44eb-b117-77f1
ee8dbe9a  rack1
UN  192.168.88.236  9.97 MiB   256          ?       d5fbd162-21d8-4b2b-b123-14fa
ef5ad3cc  rack1
UN  192.168.88.241  68.28 KiB  256          ?       bab8e6f4-7f65-4981-9cb8-07da
85bfaa34  rack1
UN  192.168.88.242  749.8 KiB  256          ?       191b585e-972e-432e-a329-92dc
44206afc  rack1
UN  192.168.88.243  2.2 MiB    256          ?       0c3d67ff-1402-46eb-8cb3-440b
c90665f3  rack1
```

Fig 2. Status of Datanodes by using nodetool

In Fig 2, there are eight data nodes in one data center, and they are all “up”. The up/down status of a node is independently determined by every node in the cluster so that the nodetool status is used on multiple nodes in a cluster to see the full view.

B. Experimental Results

As experimental results, the performance of write execution time, latency cost, and throughput, replication cost, memory cost, and the number of consistent replicas are mentioned.

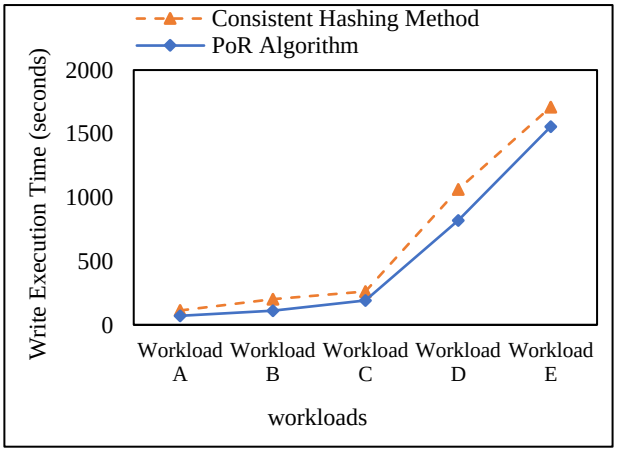

Fig 3. Comparison of Write Execution Time between PoR, and Consistent Hashing Algorithm

Fig 3 displays the evaluation of the write execution time for different replica nodes of the quorum replication with the *PoR* replica placement algorithm and the consistent hashing method. According to the *PoR*, the write execution time difference of each replica node depends on the replica nodes failure. When the failures of replica nodes increase, the system will have the more write execution time. The number of replicas and replica positions are defined on the quorum replication according to the replica arrangement policy with the *PoR*.

In a consistent hashing method, the replicas are randomly assigned to each data node. A great benefit of the consistent hashing is that the data layout does not transform significantly when the number of data nodes changes. However, the alteration to these data nodes cannot be arranged according to the system load. According to the load situation of each replica node, the intended *PoR* achieves a

dynamic replica placing approach to share the data on each node as evenly as possible. In *PoR*, each key as IP address is hashed and ordered in ascending order. And these nodes are assigned according to the write consistency level N/2+1. Therefore, the proposed *PoR* achieves write execution time by reducing the replicas for the write request and is better than the consistent hashing method in performance. In summary of experimental results, *PoR* achieves nearly 51.8% of write execution time more than the consistent hashing algorithm.

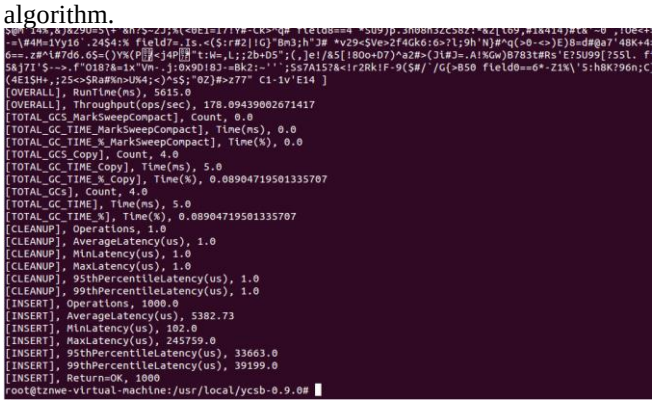

Fig 4. Performance Testing of Dynamic Snitching Method by ycsb Tool

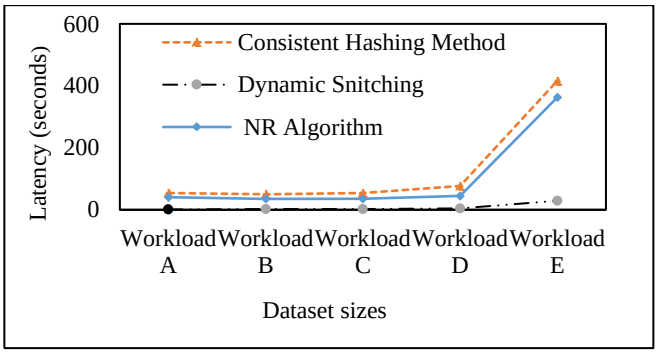

Fig 5. Comparison of Latency Cost between *NR*, Consistent Hashing, and Dynamic Snitching

Fig 5 shows the 99th percentile latency with various workloads. It indicates the read latency over various workloads by applying the *PoR* and *NR* and consistent hashing method. The *PoR* and *NR* enhance the performance of latency over the measured metrics, namely, 99th percentile latencies. The impacts of latency are examined by changing various workloads. The 99th percentile latency means that one client request has an almost ten percent chance of being affected by the slow response. In this situation, these latencies are noted at the multiple workloads. Firstly, the latency cost of the *NR* is matched with the consistent hashing method. For choosing the best replica, there are considered with N=3, R=1, and W=1. In default, dynamic snitching is used in Cassandra. It cannot get consistent replicas although it has a low latency cost because of the default replication factor. Fig 4 shows the performance testing of the Dynamic Snitching method by the ycsb tool. When the size of the workload rises in the Apache Cassandra database, latencies are more expensive because the system needs more time to complete these entire workloads. *NR*'s latency cost more improved by nearly 13.6 % than consistent hashing.

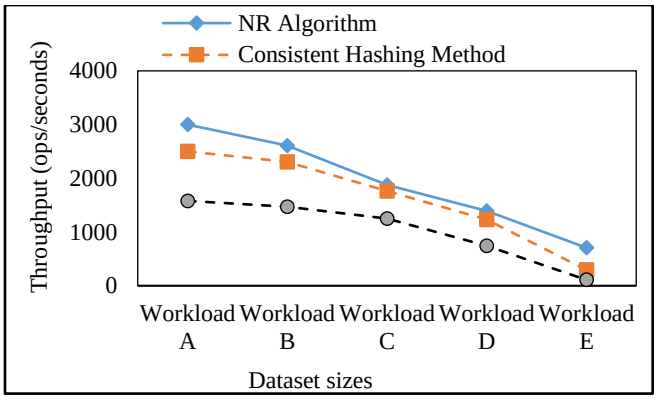

Fig 6. Comparison of Throughput Cost between *NR* and Consistent Hashing Method and Dynamic Snitching

As the various workloads increase, the throughput becomes slowly and finally, it decreases. It is a cause when the different loads get the latency, one replica or some replicas may cause failure. Therefore, it has a strong impact on the entire system's throughput. According to Fig 6, the lowest throughput can be found in strong consistency because the coordinator needs to wait for these replica nodes to acknowledge the result. This means that it also wastes resources and waiting. According to a similar reason, eventually, consistency provides the highest throughput. *NR* applies generally well in different workloads. *NR*'s throughput more enhanced nearly 30 % than a consistent hashing algorithm and nearly 30 % better than Dynamic Snitching.

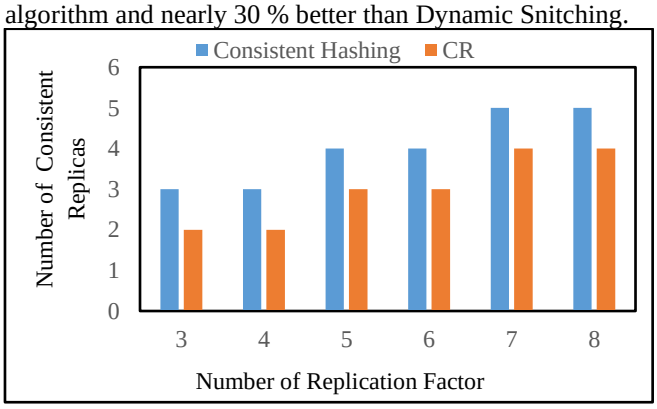

Fig 7. Comparison of Number of Consistent Replicas between Consistent Hashing, and *CR*

And determining consistent replicas becomes a problem as replicating data files to data nodes. Therefore, it is important to propose a dynamic replica selection for eventual consistency is critical [8]. It is needed to support the consistency algorithms dynamically adapt the consistency level for the read and write transactions at runtime to support a balance between consistency and the value of service. In Fig 7, the consistent replicas for the read requests are described on the total number of replication factor N=8. Maintaining a suitable number of consistent replicas for the read transactions is one of the central difficulties in data replication methods for completing the execution features. Therefore, this suitable number of replicas should be defined with consistency and performance concern, because the threshold value for several consistent replicas persists in a major role in performing accessibility for both clients and service providers.

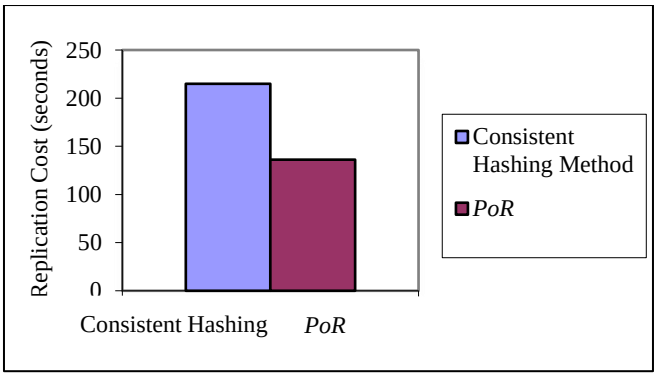

Fig 8. Comparison of Replication Cost between Consistent Hashing Method and Proposed Algorithm

From Fig 8, a fewer number of replicas can be accessed by applying the *CR* with equation 2. It intends to get the consistent replica of every request correlated to a consistent hashing for quorum replication. Therefore, the proposed system does not need to get more replicas for the choice of consistent replicas. If the experimental results complete with four write consistent replicas and three read consistent replicas, it can be defined as the best result of consistency rate. In Fig 8, the proposed algorithms get the minimum replication cost by the enhanced consistent hashing method and RTT time. It increases data accessibility and system reliability. Replica location contains classifying the best probable node to replicate data depends on network latency and client request. Replica selection means choosing the best replica position to get the data for job completion in the KVS. According to equation 1, it can basically be adapted by adapting a replica creation cost. If the time required to build a replica is 8 minutes, the replica creation cost can be increased by 8 minutes*replica. It can be simply shown that resolving the difficulty with the new replica establishment cost if immediate replica making is equivalent to resolving the initial challenge with non-zero replica construction delay.

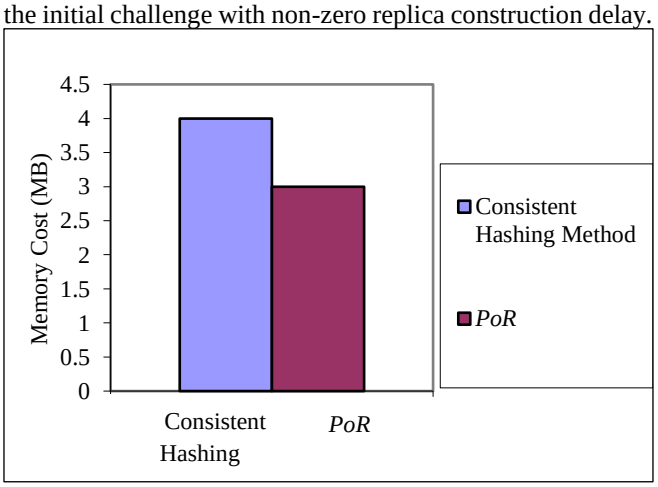

Fig 9. Comparison of Runtime Memory Cost between Consistent Hashing Method and Proposed Algorithm

Fig 9 shows the performance of memory cost between the consistent hashing method and the proposed algorithm. By applying the algorithm, the storage space, and the processing time of the user request are improved. To identify the suitable position to replicate and to decide the respective number of replicas, the total number of replica factor, and the number of replicas for the read and write requests. In the *PoR* algorithm, the number of replicas for the write request (*N*/2) +1 is initially defined according to the quorum replication factor. And the nearest or routed replicas are needed to find according to the *NR* algorithm. For the consistent read requests, the closest replicas are routed by client-ID and RTT time. Finally, the number of consistent replicas is defined by the PBS model. Therefore, the memory cost of the proposed algorithms is cheaper than the existing system used in a consistent hashing method.

VI. CONCLUSION

The problems of the fixed replication are studied and the *PoR*, *NR*, and *CR* are proposed for choosing the consistent replica in distributed KVS. The proposed algorithms allow eventual consistency which means the implementation of the KVS with replica choosing. The effect is improved the whole completion of the KVS for a large-scale variation of various workloads. These algorithms measure the read/write latency cost, throughput, replication factor cost, replication cost, memory cost and take the suitable consistency level for the read and write performance in real-time by using the enhanced consistent hashing method, and RTT. Yahoo Cloud Servicing Benchmark (YCSB) datasets are used to evaluate the proposed algorithms. *PoR* achieves write execution time more than the consistent hashing algorithm. *NR*'s latency cost is more improved than consistent hashing method. *NR*'s throughput is more enhanced than the consistent hashing algorithm and better than Dynamic Snitching. *CR* provides the consistency rate better than the consistent hashing method.

REFERENCES

[1] A. Feinberg, "Project Voldemort: reliable distributed storage", Proceedings of the 10th IEEE International Conference on Data Engineering, 2011.

[2] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in SoCC, 2010.

[3] B. Bengfort, K. Xirogiannopoulos, P. Keleher, " Anti-Entropy Bandits for Geo-Replicated Consistency, 2018 IEEE 38th International Conference on Distributed Computing Systems.

[4] D. Karger, E. Lehman, T. Leighton, R. Panigraphy, M. Levine, and D. Lewin, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," ACM symposium on Theory of computing, 1997, pp. 654-663.

[5] H. Yu, A. Vahdat, "Minimal Replication Cost for Availability", June 2002 DOI: 10.1145/571825.571839.

[6] J. Paul Walters, V. Chaudhary, "A Scalable Asynchronous Replication-Based Strategy for Fault Tolerant MPI Applications", High Performance Computing - HiPC 2007, Goa, India, December 18-21, 2007, Proceedings.

[7] L. Suresh, M. Canini, S. Schmid, and A. Feldmann, "C3: Cutting tail latency in Cloud data stores via adaptive replica selection," in NSDI, 2015.

[8] M. Diogo, B. Cabral and J. Bernardino, "Consistency Models of NoSQL Databases", Future Internet 2019, doi: 10.3390/fi11020043.

[9] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica, "Probabilistically bounded staleness for practical partial quorums," Proc. VLDB Endow., vol. 5, no. 8, Apr. 2012, pp. 776-787.

[10] P. Malik, A. Lakshman, "Cassandra — A Decentralized Structured Storage System", April 2010 ACM SIGOPS Operating Systems Review 44(2):35-40.