The Application and Improvement Discussion of Gossip in NoSQL Databases: Taking Cassandra as an Analysis Example

Yichen Qian

1Ocean University of China, Qingdao, Shandong, China
*Corresponding author’s e-mail: chunchenxiao@cas-harbour.org

Abstract. The application of gossip in NoSQL has a promising future and has significant academic value. By using gossip, a large storage system can be more scalable and stable with better fault-tolerance. This study takes Cassandra as an example to study the advantages of applying the gossip protocol in NoSQL by comparing with the conditions of using other protocols, such as Flooding algorithm and Structured Broadcast algorithm. To improve the efficiency and scalability when the size of individual gossip populations is limited, the author proposes a reconciliation mechanism named weighting. This study reveals that the gossip protocol in Cassandra contributes to linear scalability with limited individual gossip populations by implementing a simulation of gossip transmission rounds.

1. Introduction

The technological requirements for databases including “high concurrent of reading and writing with low latency”, “high scalability and high availability”, and “efficient data storage” give birth to NoSQL [1]. A representative NoSQL called Cassandra provides a simple data model that enables clients to control the data dynamically [2]. Cassandra was developed by Facebook to support the constant growth of the network system and to deal with high write throughput by providing a simple data model. There are several characteristics that distinguish Cassandra from other NoSQL databases: 1) Cassandra is a decentralized distributed storage system which runs on a cluster of hundreds of nodes; 2) Adding or deleting data is very convenient in Cassandra [1]; 3) Cassandra is highly scalable and supports linear expansion; 4) Failure or start of a single node will not influence the whole cluster. Cassandra is widely used in various areas, including product catalogs, playlists, sensor data, Internet of Things, messaging, social networking, recommendation, personalization, fraud detection, etc. Among them, Instream and GitHub are one of the most typical examples of the broad application of Cassandra [3].

Cassandra has achieved several goals, including scalability, high performance, high availability, and applicability [2]. The gossip protocol contributes to realizing these features in Cassandra. Two participants may exchange the information in a gossip round through a variety of approaches. In the push model, the node only sends information to the selected peer. However, adopting a push-only approach may easily lead to irreversible partitioning of the set of nodes [4]. Consequently, a push-pull gossip model is proposed to improve the situation, where the node and selected peer exchange descriptors. Push-pull gossip applied in Cassandra enhances system efficiency and robustness.

If two participants, p and q, gossip, only a subset of gossip messages stored in a participant can be sent under constraints of the system. Two different reconciliation mechanisms are proposed successively to solve this problem, precise reconciliation and scuttlebutt reconciliation. In precise reconciliation, the two participants exchange all recent information. However, the problem is that a considerable amount of data need to be transmitted.
A more efficient reconciliation mechanism called Scuttlebutt Reconciliation is proposed for dealing with the problem mentioned above. Instead of exchanging version numbers of all data, it maintains a unitary version state of every participant and makes a list of digests consisting of these unitary versions when gossip being implemented. Only if the unitary versions are inconsistent, precise reconciliation will be applied. Compared to precise reconciliation, scuttlebutt reconciliation is able to keep up in some of these heavy load cases and recovers quickly [5]. However, Cassandra can organize such an extensive system hierarchically, despite the limited size of digests; eventually, the propagation time will be increased [6]. In this situation, Cassandra uses a reconciliation mechanism, which selects a subset of pseudo-random participants of the required maximum size into gossip message [5].

This study analyzes how the gossip protocol model works in Cassandra and uncovers the advantages of using gossip compared to other protocol in this circumstance. The author also illustrates how it contributes to the excellent features of Cassandra. When the size of digests is limited, the author proposes a weighting reconciliation mechanism. In order to find the influence of digests’ size on the system scalability and robustness as well as comparing the efficiency of two reconciliation mechanism, when the system scales up, a simulation is presented to study the performance of random reconciliation mechanism and weighting reconciliation mechanism with varying size of limited digests.

2. Gossip Protocol Model in Cassandra
Cassandra is a distributed storage system, aiming to run on top of an infrastructure of hundreds of nodes. Within the Cassandra system, gossip protocol is used for cluster membership and failure detection by disseminating respective positions and controlling states of the nodes [7]. The gossip protocol runs every second to ensure that all nodes know the states of other nodes in the cluster.

Let \( F \) be a set of all nodes in the cluster. \( F = F_p + F_q \), \( F_p = \{ p, \ldots \} \) is the set of live nodes, while \( F_q = \{ q, \ldots \} \) is the set of unavailable nodes. Let \( F_s = \{ s, \ldots \} \) be a set of seed nodes, \( F_s \subseteq F_p \), and seed nodes chosen from all live nodes in the cluster are organized as an initial host list and reserved locally. The set of seed nodes has no particular purpose except for storing enough information for recovery.

It can be learned from the source code of Cassandra that Class Gossiper manages the gossip between nodes in the cluster and the gossip protocol works every second by running a timed task GossipTask. Consequently, it is reasonable to conclude that the gossip exchange between peer nodes can be generated through analyzing GossipTask.

In every round each participant \( p \) from \( F_p \) accompanies the following rules:
1) Always gossip to a node randomly chosen from \( F_p \) \( - \{ p \} \).
2) Probably gossip to a node randomly chosen from \( F_q \).
3) Under some circumstances, \( f \) probably gossips to a node randomly chosen from \( F_s \). For example, the size of \( F_p \) is smaller than that of \( F_s \).

3. The Advantages of Applying Gossip Protocol in Cassandra
Gossip algorithm contributes to the characteristics of Cassandra, and it is decentralization that distinguishes it from other NoSQL. Moreover, compared to Flooding algorithm and Structured Broadcast algorithm, Gossip algorithm is faster with fault tolerance and scalability in Cassandra.

In this section, the advantages of gossip used in Cassandra are demonstrated by comparing them to those of previously known protocols. The author takes Flooding and Structured Broadcast Algorithm as examples for the following reasons: (1) Flooding and Structured Broadcast Algorithm are both widely applied in a distributed network such as gossip; (2) Both these two algorithms are simple to implement.

In Simple flooding algorithm, each node forwards each message it receives, in order to ensure that all messages reach its destination. Under the worst circumstances, \( n(n-1)/2 \) messages will be transmitted in the cluster during the whole progress. Thus, it does not suit Cassandra with hundreds of nodes.

The reverse path forwarding routing algorithm works in the way that: each node keeps the state of the network [8]. Some nodes will be functioned as providers to transmit messages from a node to the other node. When flooding, a node will broadcast message to neighbors except the provider. Some
problems will occur if reverse path forwarding routing algorithm is used in Cassandra: (1) To make simplified operations of writing and reading, each node calculates routes based on the shortest paths and uses these to determine the providers for nodes. The reverse path forwarding algorithm is incorrect if the cost of links can depend upon the node using that link [8]. (2) In Cassandra, nodes getting up and down are treated as a normal situation. Under this circumstance, some providers for nodes need to be calculated again, which makes the algorithm less robust.

The structured broadcast algorithm uses BFS to build a spanning tree of nodes [9]. How much time it consumes depends on the depth of the tree. O(kn) rounds are needed to transmit all messages, and n is the number of nodes. Some problems will occur if the structured broadcast algorithm – is used in Cassandra.

First of all, if a node goes down, the whole subtree cannot get messages. It is assumed that a root of the subtree is connected top nodes. If one of p nodes goes down, nearly \( \frac{1}{p} \) nodes cannot get messages. Under this circumstance, a single fault node will lead to the construction of the spanning tree. The depth of the tree will be enlarged; additionally, if the chosen root node is not suitable, the time consumed will be increased.

The root node of the spanning tree acts as a central controller which exercises control over the communication of the whole cluster and backups of data, so the computation power and energy resources may be limited [10]. When new nodes are added to the system to suit the raised load, the spanning tree will be rebuilt, which takes extra computation (about O(n)), and imposes restrictions on the extension of the system.

Assuming k messages needed to be transmitted in the cluster of n nodes, then gossip algorithm costs O(k) rounds to make each node gets all messages. Obviously, it is faster than flooding and structured broadcast. Steps are illustrated as the following:

**Phase 1:** Less than half of the nodes have been informed. Probability [message is helpful] \( =1 - \frac{k}{n-1} \geq 0.5 \). Thus, at least 1.5k messages need to be sent.

**Phase 2:** More than half of nodes have been informed. For each node, probability [none of nodes sends to me] \(=(1 - \frac{1}{n-1})^k \approx 0.25 \). t is set as the number of rounds needed, \( 0.75^t = n \), \( t=\log(n) \), \( O(n\log n)=1 + 2 + 4 + \ldots + \frac{n}{2} = 2\log_2 n - 2 = o(\log n) \).

Cassandra is a decentralized structured storage system, which means that each node takes the same part in communication coordination and data storage [7]. It is the application of gossip that makes it possible. In Cassandra, the gossip protocol works every second and every round, and each participant node gossip to one to three random peers. Thus, information of all nodes kept in each node is updated every second, which results in decentration. Without a centralized node of management, the system is more reliable and easier to scale up. By applying gossip protocol, only adding nodes can achieve the goal of scalability in Cassandra [1]. If the number of nodes can be increased by m times to extend the system, the rounds needed is \( O(k \log mn) \) / \( O(k \log n) \) < m. The increase of resources that manages the dissemination is linear and less than m times. Thus, the efficiency of the resource utilization is improved, when load of the system grows, it is necessary to scale the system up.

Under the action of some Cassandra mechanism, if a node gets UP, the list of seed nodes is provided to the new node to avoid gossip storm. Accordingly, the new node can choose one of them and gossip immediately. After a round of gossip, it will know all the information of other member nodes. In Cassandra, each data item is replicated at N hosts, where N is the replication factor configured “per-instance” [7]. When a node goes down suddenly, neither re-calculating nor repairing needs to be carried out since the outage usually is temporary, and the operations of reading and writing are hardly influenced. In a word, gossip algorithm provides better fault tolerance for communication in Cassandra, as well as increasing system robustness and scale.

4. Improvement of Efficiency when Scaling up
4.1. Limited size of digests in Cassandra

In Cassandra, information to gossip is key-value pairs being wrapped in ApplicationState object and HeartbeatState. ApplicationState and HeartbeatState consist of generation number and version number. EndPointState for certain nodes is composed of ApplicationStates and a HeartbeatState. But the version of these states can only grow. Once a state is updated, the version should be set to a number larger than the maximum version that participant q has for p. Gossiper in every node has internal structure called EndPointStateMap that comprises EndPointState for all nodes (including itself) that it has heard about [11].

Cassandra uses a mechanism called Scuttlebutt Reconciliation to eliminate differences between two participants in gossip [5]. Let P= \{p, q, \ldots\} be a set of all nodes in the cluster. In each round of the initial phase, only one live peer being selected by a node to gossip. In this circumstances, p randomly chooses a node q from, = P - \{p\} - \{unavailable nodes\}, and then p can be exchanged by sending a list of gossip digests called GossipDigestSynMessage. Instead of sending data (key, value, version) of all ApplicationStates in EndPointStateMap, GossipDigestSynMessage contains only the maximum generation and version of HeartbeatState and ApplicationStates. q will receive the message and reply a message called GossipDigestAckMessage. It involves two parts, gossip digest list of older information and endpoint state list of newer information by comparing generation and version with the list of digests from p. After receiving GossipDigestAckMessage, p will send a message similar to it, yet in the opposite direction called GossipDigestAck2Message.

4.2. Reconciliation mechanism

To exchange digests in a large system where the size of digests is limited, participants of gossip need to use reconciliation mechanism. At present, it is quite simple for Cassandra to generate pseudo-random numbers and choose corresponding version numbers. However, if the system scaled up by adding nodes, it will cost more propagation time with less stability.

This study proposes a more efficient mechanism to select nodes into the list of gossip digests in the size-limited situation, making it faster to achieve data consistency. Let w_q be the number of ApplicationStates and HeartbeatState of every participant q’s EndPointState stored in a participant p. \(\sum_i w_q\) is a total of w_q in participant p. Each time when selecting a participant’s EndPointState stored in a participant p to put into the list of digest, the probability of choosing q’s EndPointState stored in participant p is \(\frac{w_q}{\sum_i w_q}\). The author names the reconciliation mechanism weighting.

5. Simulation

A simulation has been implemented so as to model the gossip protocol in a large storage system. Let m be the limited size of digest list. The author changes m and runs 50 times for each chosen nodes number to quantify the number of rounds needed in varying scale cluster in order to reach data consistency. The results of the simulation are plotted in Figure 1 and Figure 2.

It is noticeable from Figure 1 that there is a nearly linear growth in average rounds, which makes the system scalable by adding nodes. Additionally, with the increase of nodes number in the cluster, the limited size of the digest in Gossip can be neglected. Furthermore, the author observes that the average rounds nearly decreases linearly with the increase of the number of randomly chosen nodes. Moreover, it can be observed from Figure 2 that if the limited size of the digest is small, variance becomes large when the system is expanded. Thus, there is a chance that when some information of a node changes, it takes a long time for the cluster to reach data consistency. The increasing size of the digest in the hierarchically organized database enables to facilitate the stability of the system.
Further experiments have been carried out to compare the efficiency of random reconciliation mechanism and weighting reconciliation mechanism. The results of the experiments are plotted in Figure 3 and Figure 4. When the system scales up from 2 nodes to 100 nodes, weighting reconciliation costs relatively fewer average rounds compared to random reconciliation, especially in the case where $m$ is large.

In conclusion, weighting reconciliation is more efficient, while the limited size of the digests listed in GossipDigestSynMessage is large. Moreover, the increase of average rounds by weighting reconciliation is gentler, which makes the system more extensible. However, the problem with this is that sample standard variance of rounds to reach data consistency is larger when $m$ is small. It represents that the time it costs is less than the average time to reach data consistency by weighting reconciliation occasionally.

6. Conclusion
Applying gossip protocol in NoSQL database meets the needs of scalability, robustness as well as fault-tolerance. In this study, the author firstly compares gossip with two widely used algorithms in distribute network. It is found that the gossip protocol contributes to efficiency and scalability of Cassandra, meaning that rounds needed to reach data consistency are fewer by using gossip when compared to two other algorithms. Furthermore, there is a linear increase in resources when the network system scales, if the gossip protocol is applied for information dissemination. The simulation results show that weighting reconciliation is relatively more efficient, especially when the limited size is large.
In further study, the author will explore reconciliation mechanisms and try to improve the “weighting” reconciliation mechanism to avoid the situation that time costs of gossip occasionally become relatively large when limited size is small.

References
[1] Jing Han, E Haihong, Guan Le, Jian Du, “Survey on NoSQL Database”, 6th International Conference on Pervasive Computing and Applications, 2011.
[2] Lakshman, Avinash, and P. Malik. “Cassandra: structured storage system on a P2P network.” SPAA 2009: Proceedings of the 21st Annual ACM Symposium on Parallelism in Algorithms and Architectures, Calgary, Alberta, Canada, August 11-13, 2009 ACM, 2009.
[3] Chebotko, A., A. Kashlev, and S. Lu. “A Big Data Modeling Methodology for Apache Cassandra.” IEEE International Congress on Big Data IEEE Computer Society, 2015.
[4] Márk Jelasity, et al. “Gossip-based peer sampling.” Acm Transactions on Computer Systems 25.3 (2007):8.
[5] Renesse, Robbert Van, et al. “Efficient reconciliation and flow control for anti-entropy protocols.” Workshop on Large-scale Distributed Systems & Middleware ACM, 2008.
[6] Van Renesse, Robbert, K. P. Birman, and W. Vogels. “Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining.” Acm Transactions on Computer Systems 21.2 (2003):164-206.
[7] Lakshman, A. “Cassandra-a decentralized structured storage system.” 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware (LADIS 09), Oct.ACM, 2009.
[8] Bolton, Christie, and G. Lowe. “Analyses of the Reverse Path Forwarding Routing Algorithm.” International Conference on Dependable Systems & Networks IEEE Computer Society, 2004.
[9] El-Ansary, Sameh, et al. “Efficient Broadcast in Structured P2P Networks.” International Workshop on Peer-to-peer Systems Springer, Berlin, Heidelberg, 2003.
[10] Boyd, S., et al. “Randomized gossip algorithms.” IEEE Transactions on Information Theory 52.6 (2006):2508-2530.
[11] https://wiki.apache.org/cassandra/ArchitectureGossip