マルチメディア通信と分散処理 95－11
（１９９９．１１．１９）

Lazy-group replication における ビュー最新度制御分散アルゴリズム

山下 高生

日本電信電話株式会社

東京都武蔵野市緑町 3-9-11

E-mail: yamasita@slab.ntt.co.jp

本論文では、lazy-group replication を用いて管理されるデータに、クライアントが参照する時、クライアントの要求した最新度以上のデータを提供するためにアクセスしなければならない、最小数の複製ノードの計算を行う分散アルゴリズムを提案する。本方法では、複製ノードが木構造に配置され、データに対する更新が、木構造を伝搬することによって行われる。本アルゴリズムでは、まず、各複製ノードが範囲内発信ノードと呼ばれるノードを管理する。範囲内発信ノードとは、各複製ノードが、範囲内発信ノードから開始された更新について、クライアントの要求した最新度以上のデータをもつことを保証するノードである。次に、各複製ノードは、範囲内発信ノードを木構造上で隣接する複製ノードとの間で交換・比較することによってクライアントの要求した最新度以上のデータを提供するために必要な複製ノードの計算を行うことを特徴とする。本アルゴリズムについては、検証を目的として実装による評価も行った。

キーワード: データ複製, 弱一貫性, 最新度, 遅延, コーラム・コンセンサス

A distributed algorithm for view divergence control of read data freshness in lazy-group replication

Takao Yamashita

NTT Software Laboratories

9-11, Midori-Cho 3-Chome Musashino-Shi, Tokyo 180 Japan

E-mail: yamasita@slab.ntt.co.jp

In this paper, we propose a distributed algorithm to calculate a minimum read replica set to control view divergence of replicated data in lazy-group replication, which has been implemented for verification. In our method, replicas are connected using a tree structure and updates for replicated data are propagated through the tree. In our algorithm, each replica manages a set of range originators from which updates can reach it with freshness required by clients. Each replica exchanges information about the sets of range originators with neighboring replicas. Then, each replica determines if it is a read replica for view divergence control by comparing sets of range originators received from neighboring replicas.

Keywords: data replication, weak consistency, freshness, delay, quorum consensus

1 Introduction

With the growth of the Internet, it has become a tool in various field, not only in research but also in electronic commerce, advertisement, personal communication fields and so on. Applications used in various field handle data that has different properties. Requests for accessing these data should be processed with scalability and reliability in the Internet, because it contains numerous hosts and links.

Data replication is a relatively effective method to achieve scalable and reliable processing of the requests. Data replication methods are divided into four types in terms of two parameters: the number of replica with privilege to update a data object and update timing [1]. In the first terms, the method is divided into *master* or *group* type. In the master type, only one replica can originate update for a data object. Group type can originate updates to a data object. The data replication method is divided into *eager* or *lazy* in the second term. The eager replication updates data synchronously. The lazy replication asynchronously updates data.

The eager replication can achieve ACID serializability [2] but it has higher overhead to update data on multiple replicas synchronously. In lazy-master replication, ACID serializability is achieved but its scalability depends on performance of master replica. Lazy-group replication has a lower consistency than ACID serializability but it has advantages for scalability. Some applications, in telecommunication, data warehouse systems, and so on, can work with a lower consistency.

Lazy-group replication has higher scalability lead by asynchronous updates originated at multiple replicas and aggregation of updates and so on. However, freshness of data read by clients varies depending on a replica accessed by a client. When a client access to a replica that is near from a replica that originates the latest update, a client can obtain the latest data with a higher possibility. However, the accessed replica is far from replicas that originates recent updates, data obtained by a client is older. In some applications, the freshness of data is essential, for example of the tracking location of mobile host.

We have previously proposed an algorithm to control view divergence of replicated data managed using lazy-group replication method [3][4]. This algorithm is centralized. However, because database of multiple locations and multiple organizations in distributed systems, the algorithm is hoped to be decentralized. In this paper, we propose a distributed algorithm to control view divergence of replicated data. We implement our algorithm to verify it. This paper is organized as follows. In Sec. 2, we first describe a summary of view divergence control of replicated data that has previously been proposed. We then describe a summary of a centralized algorithm to calculate replicas read by a client. In Sec. 3, We state a distributed algorithm for view divergence control.

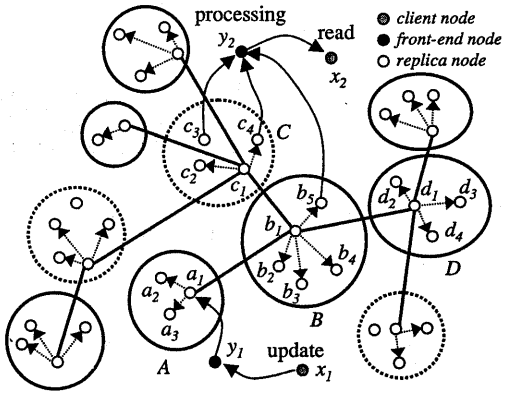

Figure 1: System architecture used in our method of controlling the view divergence of replicated data

2 Summary of view divergence control of replicated data

In this section, we first describe a summary of view divergence control that has previously been proposed [3][4]. Then, we describe a centralized algorithm that has also been proposed previously [3][4].

2.1 view divergence control

In our method, there are three types of node: client, front-end, and replica nodes as shown in Fig. 1. Replica nodes are divided into multiple groups called *replica groups*. Data on replicas in one group are synchronously updated by eager replication. The transactions requested by clients are read and update. The update transaction requested by a client is sent to a front-end node. The front-end node that receives the update transaction sends it to a replica group. Then, the update is propagated into all the replica groups as a refresh transaction. A read transaction requested by a client is also sent to a front-end node. A front-end node sends a read transaction to multiple replica groups. After a front-end node receives the data and, if necessary, recent logs of updates received by each replica group as a reply message to the transaction, the front-end node calculates the data that reflects all the updates received by the accessed replica groups using this information and the timestamp associated with the data. In this calculation, a front-end node processes updates as a replica group does. Our method controls the view divergence by using this calculation process, which depends on the applications or the definition of consistency. For example, in the case of the tracking location

of a mobile host, this calculation is very simple. In this application, a replica group process updates by replacing old data with a newer update based on the timestamp associated with the data and update. Therefore, the calculation of data that reflects all the updates received by the accessed replica groups is done by selecting the latest data among obtained data from replica groups based on their timestamp.

There are a number of combinations of replica groups that meet the view divergence required by a client. Since the response time increases as the number of replica groups a front-end node has to access increases, the number of replica groups should be minimized. In addition, since there are $2^n - 1$ combinations of replica groups, where n is the number of replicas, an effective algorithm to select the minimum number of replicas is necessary. We have previously proposed an effective algorithm that is described in the next section [3][4]. In this algorithm, we use the probabilistic lateness of updates reflected in acquired data, called read data freshness (RDF). The degree of RDF represents a client's requirement about the view divergence. The formal definition of RDF T_p is $T_p = t_c - t$ if all the updates invoked before time t somewhere in a network are statistically estimated to be reflected in data acquired by a client with probability p, where t_c is the present time. This means that if the last update reflected in the acquired data was invoked at $t_l < t$, no updates are invoked between t_l and t. In other words, any updates invoked between t_l and t will not come to any replicas in the future with probability p. We represent data with the degree of RDF T_p as data within T_p.

2.2 System architecture

To select replica groups so that the acquired data by a client meets the degree of RDF, we have to know how update propagation delay occurs. It depends on the topology of update propagation paths. Figure 1 shows the system architecture used in our method. Replica groups are connected by logical links. The topology of the graph comprised of the logical links and the replica groups is a tree. An update sent by a front-end node propagates in the tree through the links as a refresh transaction. When a replica group receives an update, the group forwards it to all the adjacent groups except the replica group from which the updates came. The update delay is caused by communication delay, protocol overhead to record logs to fulfill recovery [5], time for aggregating some updates in one refresh transaction [6], and so on.

In our method, we assume that clocks of client, front-end, and replica nodes are synchronized using various clock synchronization methods [7][8]. In addition, we assume that update propagation delay can statistically be estimated from past update propagation delay. These assumptions are discussed in [3][4].

$A := \{i \mid i \text{ is a replica node }\}$;
$R := \{\emptyset\}, V := \{\emptyset\}$;
$S := A$;
while $(\forall i \in A \setminus V : O_i \neq A \setminus V)$; ··· (1)
  for all $i \in S$, $O_i := O_i \setminus V$; ··· (2)
  $C := \{i \mid i \in S \wedge \forall j \in S : O_i \not\subset O_j\}$; ··· (3)
  for all $i, j \in C$ $(i \neq j)$:
    if $O_i = O_j$ then $C := C \setminus \{j\}$; ··· (4)
  $M := \{i \mid \exists j \in A \setminus V : \forall k \in C\ (k \neq i) :$
    $j \in O_i \wedge j \notin O_k\}$; ··· (5)
  $R := R \cup M$; ··· (6)
  $V := V \cup \{i \mid i \in O_j \wedge j \in M\}$; ··· (7)
  $T_s := constructMinimumTree(A \setminus V)$; ··· (8)
  $S :=$ a set of replicas in T_s; ··· (9)
$R := R \cup \{i\}$, where $O_i = A \setminus V$; ··· (10)

Figure 2: Algorithm to calculate a minimum read replica set.

2.3 Summary of centralized algorithm

2.3.1 Terminology

To explain our algorithm, we define four terms, which are read replica (set), range originator, classified replica, and mandatory replica. A *read replica* is a replica group to which a front-end node sends a read transaction to obtain the value of a data object. We denote a set of read replica groups to obtain the data with the degree of RDF required by a client as a *read replica set*. A *range originator* of replica group r is a replica group that originates a refresh transaction and from which the transaction can reach r within time T_p with probability p when the degree of RDF required by a client is T_p. This means that the upper confidence limit of the delay from a range originator to replica group r with probability p is T_p. Let O_i be a set of range originators of replica group i. Then, we say that replica group j is *covered* and *uncovered* by replica group i if $j \in O_i$ and $j \notin O_i$, respectively. A *classified replica* is a replica group whose set of range originators is not a subset of a set of range originators of any other replica groups. A *mandatory replica* is a classified replica that has one or more range originators which are not included in any other classified replicas' range originators.

2.3.2 Calculation of a minimum read replica set

In this section, we describe an algorithm to select a minimum read replica set when an update is propagated into replicas connected by a tree. This algorithm utilizes two properties that caused by statistical delay distribution of update propagation.

Let d_{xy} be the upper confidence limit of the delay time from replica group x to y with probability p. The first property is that if $d_{ik} \leq T_p$, then $d_{ij} \leq T_p$ and $d_{jk} \leq T_p$ for all replica group j that are on the path from replica group i to k. The second property is that if $d_{ij} > T_p$, then $d_{ik} > T_p$ for all replica groups k that satisfy the condition that replica group j is on the path from replica group i to k.

Our algorithm incrementally constructs a minimum read replica set R comprised of only classified replicas. Figure 2 shows our algorithm to calculate a minimum read replica set. T_o and T_s are the original tree for update propagation and a subtree of T_o constructed in our algorithm. V, S, C, and M are the sets of replica groups covered by $i \in R$, replica groups in T_s, classified replicas for T_s, and mandatory replicas in each iteration, respectively.

In step (2), we remove replica groups covered by replica groups in R from O_i. In step (3), we calculate the classified replicas for T_s, which is initially T_o. Step (4) removes the classified replicas that have the same range originators. Step (5) calculates the mandatory replicas. In step (6), all the mandatory replicas are added to R, since a mandatory replica has at least one replica group that is not included in the range originators of any classified replicas. In step (8), we construct the minimum subtree T_s including all the uncovered replica groups, assuming the weight of all the edges is 1. The procedure from step (2) to (9) is iterated unless there is at least one replica group that covers all the replica groups uncovered by any replica group $i \in R$ in T_s. Step (10) adds one of the replica groups that covers all the replica groups uncovered by any replica group $i \in R$ in T_s to R and then the algorithm terminates.

3 A distributed algorithm

The centralized algorithm determines classified replicas at first and then determines mandatory replicas from classified replicas. These are done by comparing range originators of each replica. To decentralized this algorithm, it is necessary to decentralized these determinations.

Theorem 1 *If replica groups i and j are adjacent and replica group k is a range originator of i but is not that of j, then replica group m does not cover k for all m, where the path from replica group i to m includes j.*

Proof: Assume that the path from i to k include j. Because replica i covers k, replica j also covers k. This is contradiction. Therefore, if replica j does not cover k, replica m does not cover k. □

Our distributed algorithm determine if each group is a classified replica by comparing sets of range originators between neighboring replicas. The following is the procedures of our decentralized algorithm.

1. Fladding node information to all the other replicas.
2. Detect termination of the above fladding.
3. Exchange a set of range originators between neighboring replica groups.
4. Determine if each group is a classified replica.
5. Determine if each group is a mandatory replica.
6. Fladding information about replica groups covered by mandatory replicas.
7. Construct subtree using remaining replica groups.

3.1 Determination of classified replicas

Let i be each replica group, j be a neighboring replica group and O_k be a set of range originators of replica group k. When $\forall j$: $O_i \not\subseteq O_j$, replica i is a classified replica. When $\neg\ (\forall j$: $O_i \not\subseteq O_j)$, or when $\exists j$: $O_i \subseteq O_j$, the relationship of O_i and O_j at each replica group i is as follows.

1. $O_i \not\subseteq O_j$, $O_i \subset O_k$
2. $O_i = O_j$, $O_i \subset O_k$
3. $O_i \subset O_j$
4. $O_i = O_j$, $O_i \not\subseteq O_k$, $O_i \subset O_l$
5. $O_i = O_j$
6. $O_i = O_j$, $O_i \not\subseteq O_k$

When the relationship is one of 1, 2, 3, or 4, replica group i is not a classified replica. When the relationship is one of 5 or 6, it is impossible for each replica group to determine if it is a classified replica by comparing sets of range originators of neighboring replica groups.

Theorem 2 *If replica group k is a range originator of replica groups i and j, then replica group k is a range originator of replica group l for all replica group l along the path between replica groups i and j.*

Proof: Assume that replica group m is along the path between replica groups i and j and k is not a range originator of replica group m. Then update originated by replica group k reaches replica group m before it reaches replica groups i and/or j. this means that replica k is not a range originator of replicas i and/or j. It is contradiction. □

Replica groups with the same set of range originators are adjacent or are connected via replica groups whose set of range originators are a superset of it from the Theorem 2 as shown in Fig. 3. In (a) of Fig. 3, replica groups 2, 3, 4, 6, and, 9 have the same set of range originators. In (b) of Fig. 3, replica groups 2, 3, 4, and, 9 have the same set of range originators. When the set of range originators of replica group 6 is not the same as those replica groups, the range originator set of 6 is a superset of that of those replica groups. When the range originator set of those replica groups is a subset of that of any other replica groups, at least one of those replica groups can detect it. Otherwise, any those replicas cannot detect if it

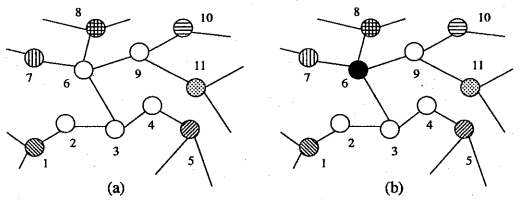

Figure 3: Range originator sets of neighboring replicas.

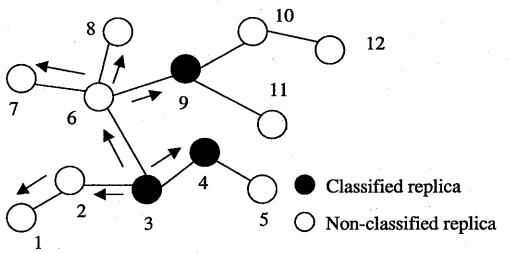

Figure 4: Determination of mandatory replicas using diffusing computation.

is a classified replica. In this case, one of them should be a classified replica.

When there exists replica group i, where the relationship between O_i and O_j is 5 or 6, our algorithm determines if each replica group is a classified replica as follows. When replica group j exists, where $O_i \subset O_j$, replica group i can detect it. Therefore, if replica group j exists, at least one replica group can detect and notify it to adjacent nodes with the same set of range originators. On the contrary, if any replica cannot detect if there is a replica group whose set of range originators is a superset of it, it is not covered by any replica. Hence, in our algorithm, each replica group sends the determined result to neighboring replicas with the same set of range originators. In addition, when any replicas with the same set of range originators cannot determine if it is a classified replica, one of them is selected as a classified replica.

3.2 Determination of mandatory replicas

To determine if each replica group is a mandatory replica, we use Theorem 1. A computation model for mandatory replica determination is diffusing computation [9] and we use the termination detection in the diffusing computation [9]. Each classified replica first sends messages including its set of range originators to neighboring replicas. When a non-classified replica receives the message, it is forwarded based on diffusing computation model after removing range originators that are not those of the receiving replica. At this time, if the range originator set of the message is empty and/or it does not have any neighboring replicas to forward the message, the receiving replica returns a termination message based on the termination detection method in diffusing computation. When a classified replica receives the message, it returns a termination message including common range originators between originating and receiving classified replicas. When a classified replicas receives termination messages from all the neighboring replicas and there is at least one range originators that are not included in any termination message, it is a mandatory replica. For example of Fig. 4, classified replica 3 originates a message to determine if it is a mandatory replica. This message is forwarded to replicas 1, 2, 4, 6, 7, 8, and, 9. Termination messages for this message are forwarded through reverse path to replica 3.

3.3 Constructing a subtree for the next iteration

We describe a method to determine if each replica is in a subtree for the next iteration. We call a message to determine if each replica is in a subtree of the next iteration as a subtree-sensing (SS) message. First each node manage state about a receiving and transmitting SS messages for each neighboring replica. The receiving message state for neighboring replica i is on when a replica receives a SS message from replica i. The transmitting message state is on when a replica transmits a SS message to replica i. A SS message includes a set of range originators covered by mandatory replicas.

A replica that has only one neighboring replica originate a SS message. This replica turns on the transmitting states for its neighboring replica. The message originated by this replica includes a set of range originators. The set is empty when it is not a mandatory replica. The set is the set of range originators of this originating replica, when it is a mandatory replica. A SS message is forwarded to a neighboring replica group i when each replica group has received SS messages from all neighboring replica groups except replica group i. When each replica group forwards a SS message, it modifies a set of range originators. When it is a mandatory replica, it add its range originator set to the message. Otherwise this message is only forwarded.

Finally, each replica group may determine if it is in a subtree for the next iteration, when it receives SS messages from all the neighboring replica groups. When each replica group is not covered by any mandatory replica, it is in a subtree of the next iteration. When each replica group finds more than one neighboring replica groups through which reach uncovered replica groups, it is in a subtree. Otherwise, each replica group is not in a subtree of the next iteration.

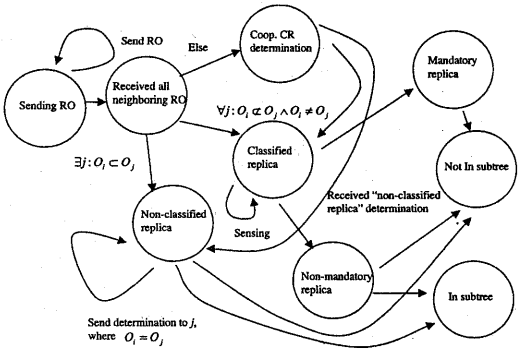

Figure 5: State transition diagram for distributed read replica set determination

3.4 State transition diagram

The primary part of this algorithm has 9 states at each replica group as listed below. The primary part is composed of procedures from 3 to 7. Transitions between these states are done based on the algorithms in the above description as shown in Fig. .

1. Sending RO
2. Received all neighboring RO
3. Non-classified replica
4. Cooperative CR determination
5. Classified replica
6. Mandatory replica
7. Non-mandatory replica
8. Not in subtree
9. In subtree

4 Conclusion

We have proposed a distributed algorithm to determine a minimum read replica set for view divergence control of replicated data. We have verify our algorithm by implementing it. We will evaluate our distributed algorithm in an environment that is closer to real world.

Acknowledgments

We thank Dr. Haruhisa Ichikawa for his continuous support. We also thank Mr. Hirohide Mikami for his support and helpful suggestions.

References

[1] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *Proc. ACM SIMOD International Conference on Management of Data*, pp. 173–182, 1996.

[2] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[3] T. Yamashita and S. Ono, "A view divergence control method for replicated data and its scale-dependency evaluation," tech. rep., DPS 99-1-1, IPSJ Technical Report, 1999.

[4] T. Yamashita and S. Ono, "View divergence control of replicated data using update delay estimation," in *Proc. 18th IEEE Symposium on Reliable Distributed Systems*, pp. 102–111, Oct. 1999.

[5] E. Pacitti, E. Simon, and R. Melo, "Improving data freshness in lazy master schemes," in *Proc. IEEE 18th International Conference on Distributed Computing Systems*, pp. 164–171, May 1998.

[6] K. P. Birman, *BUILDING SECURE AND RELIABLE NETWORK APPLICATIONS*. Manning Publications, 1996.

[7] D. L. Mills, "Precision synchronization of computer network clocks," *Computer Communication Review*, vol. 24, pp. 28–43, Apr. 1994.

[8] T. Yamashita and S. Ono, "A statistical method for time synchronization of computer clocks with precisely frequency-synchronized oscillators," in *Proc. IEEE 18th International Conference on Distributed Computing Systems*, pp. 32–39, 1998.

[9] N. A. Lynch, *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.