On a construction and a highly effective update method for the consistency in structured p2p systems

Hong Minh Nguyen[1], Xuan Huy Nguyen[2], Ha Huy Cuong Nguyen[3]
[1] People's Security University, Vietnam
[2]Institute of information technology, Vietnam academy of science and technology, Vietnam
[3]Quang Nam University, Vietnam

Abstract: *Many replicas are distributed of an object sharing be processed by the independent users (replica nodes), which leads to inconsistency. The authors propose a new efficient solution to construct a tree and an update propagation method for each data sharing on distributed applications in the structured p2p substrate. In these trees, the replica nodes are linked logically as close together to increase communication efficiency and performance of optimal replication base on the update rate, thus enhancing the availability of data, so reducing of latency update and increasing the ratio of a successful update of generated replicas. The proposed method has been experimented by using the OverSim simulation to prove the effectiveness and comparing with existing solutions.*

Keyword: *structured p2p substrate; data consistency; replica; replica node; update tree.*

I. INTRODUCTION

Sharing a dynamic distributed data, in the sense that the data can change by frequent updates or even simultaneous processing interacted by many users increasingly interested in research, such as p2p WiKi [1], Social Networking [2], p2p collaborative workspace [3]…. The distributed systems are built on top of the p2p substrate. Besides, peers link logically to create an overlay network on the physical underlay network (such as Pastry [4], Tapestry [5], CAN [6]…). P2P is suitable for construct a large-scale and complex applications due to provide distributed platform, fault tolerance and ensuring high availability of data sharing. Main features are that peers are self-organizing, heterogeneous about processing capabilities, join/leave or update rate, delay of communication and bandwidth usage.

A peer which holds a copy of object sharing is named as a replica node. Therefore, the replicas are processed by replica nodes, which are independent and autonomous, so over time replicas can not guarantee consistency [7], this leads to misleading results. Therefore assurance of consistency is a very important requirement. However, it is also a difficult and major challenge because of the complexity requirements of distributed applications (scalability, requirements consistency …) and characteristics of p2p.

Any replica node updates on the local replica, so the change must be propagated to other replica nodes in the system, called the consistency maintenance scheme. The scheme includes a solution for the structure and a method of update propagation.

An auxiliary tree (update tree) is built on top of the overlay structure (example Figure 2) for sending updates proved to be a suitable and effective solution for distributed applications building on the structured p2p substrate. Such as ensuring high reliability, reducing the number of messages and using bandwidth usage due to redundancy, duplicate… However, it also poses many difficulties, complications in research and development, that need to solve as an overload of replica nodes, bottleneck, imbalanced… Consequently, the scheme which using of an updated tree in previous studies still has the following limitations:

The first, limited on a method of building tree

Because the update tree is built in chronological order into the node's system, then interconnected

nodes can be distant in physically or logically, so increases latency, number of messages and bandwidth usage when exchanging messages or spreading updates. The updated tree is built the base on calculating the ability of the node (stability, processing speed…) that will be very difficult and costly, especially when node's join/leave high rate. An updated tree dynamically only show effects when node's update low rate because of no maintenance cost of update tree.

The second, limited on a method of update propagation The method of update propagation for all node online is high cost, redundant and not suitable for large-scale applications or the users require to use a replica with different time. The method only propagates updates to a subscribed nodes that overcome above limitations but it needs to solve difficult problems as the bottleneck, imbalanced… The case of update propagation from the root to a subscribed nodes (below, even a leaf nodes), so high-cost a lot of time (latency), a number messages, bandwidth…

In order to overcome the above limitations, the researchers propose a new solution that will allow assuring sequential consistency [7], nodes link in tree base on node's ID (logical value) and assign the identifier space to which it is responsible. So that nodes are linked logically as close together to have a high-efficient communication and processing node's join/leave high-rate. Moreover, the solution calculates to perform an optimal replication base on the subscribed rate thus enhancing the availability of data and it only updates to a subscribed nodes, so that reducing latency and increasing the ratio of a successful update of a generated replicas. During, some researcher have been proposed method based on the update rate of nodes and to solve the problem of load balancing, or application on the consistency maintenance scheme use a dynamic tree, make updates for all nodes and is only effective when a node has low update frequency. The results of the research have new contributions are as follows:

A proposal for construction and maintenance of the update tree base on node's ID.

Calculating of an optimal replication base on the subscriber's update rate and only an update to a subscribed node, so that reducing latency and increasing the ratio of the successful update of a generated replicas.

The experiment of the proposal by using the OverSim simulation [8]; comparing and demonstrating of the effectiveness by the proposal to the solutions by Nakashima [9], Yi [10].

The rest of the paper is organized as follows: the related work is reviewed in section II. Gives an overview of our proposal, followed by a detailed description of its design in section III, the performance of proposal is evaluated in section IV. The paper is concluded in section V.

II. RELATED WORK

Li [11] propose to construct an updated tree dynamically including a supernode (high reliability and capacity). The maximum α of low-capacity nodes are liked to each super node with close proximity to receive updates. This solution only shows effects when node's update low rate, because of no maintenance cost of update tree. However, when a node has an update high rate so the cost of construction tree is very high, which has difficulty to determine the ability and proximity of nodes. Moreover, all updates are sent to all nodes online by root so that increasing the cost and be not practical because of each node requires use the replica with different time.

Xin [12] propose SCOPE for construction of an updated tree statically by using splitting identifier space into equal partitions and selecting one representative node in each partition which records the replica locations within that partition. Only leaf nodes store a replica. Then the update can be sent layer by layer from the root to leaf nodes. Nodes may not be interested in the object sharing are in the object's update tree, which adds the scale of the tree is very large so that it is the unnecessary overhead of maintaining the tree from node's join/leave and increases the delay of update propagation…Moreover, nodes may be easily overloaded when sending updates, because of its participating all update tree. Especially, updates are sent from root to leaf nodes so that inefficient latency and problems as the bottleneck, imbalanced…

Nakashima [9] propose a construction of an updated tree statically only from replica nodes. It connects to the tree by computation of the balance of the number of subtree nodes. When new updates are

received, the root propagates to all nodes below, then other new updates arrive in the root that will be discarded. Because of node join/leave can be a single node or a subtree, therefore the reconstruction of such the tree will be ineffective when the node's join/leave high rates, since it both has cost calculation and may be the imbalance of the tree (height), it also increases the average delay of update propagation for all nodes. Since each joint/leave node can be single ỏ a subtree, the reconstruction …

Yi [10] propose BCoM for construction of an updated tree statically from replica nodes which are ordered by their arrival times and balance of the number of subtree nodes. Each replica node (except leaf nodes) is given a buffer of size k for buffering updates. The root receives an update, then sequentially sends it to all its children. Such operation is repeated until the leaf nodes received. The sliding window size k is critical for balancing the consistency strictness, object availability, and latency. So BCoM proposed a class that each node requires a maximum and sequences of generated updates, balanced among the parameter above. The drawbacks are that a tree is built ordered by their arrival times and balanced tree, thus the tree is not optimal. Moreover, using the buffer easily leads to a bottleneck. Although, Yi proposes a swapping for preventing from blocking by the master node to a higher level besides, degrade the bottleneck node to the lower level. However, this is very difficult and costly, especially when in both cases, node's join/leave or update high- rate.

III. DESCRIPTION OF A PROPOSAL

III.1. Overview

1.1. The researchers use an overlay p2p Pastry to illustrate the new proposal. However, it can also be applied for other structured p2p such as Tapestry, CAN… Pastry use a distributed hash function to uniquely identify (ID) of each node and object sharing in the same identifier space 128 bit ([0, $2^{128}-1$]). Node i's ID (denoted by ID_i hash from address IP and object f's ID (denoted by ID_f) hash from the name of the file. Node linked logically in Pastry as Figure 1 and exchanging messages by the overlay routing.

Any node I need an object sharing, it sends a request to a root by the overlay routing. The root call algorithm ID_LINK to link the node into the tree. After linking, the new node subscribers a replica from its parent (or any node subscribes an update), so just register them all the corresponding location in vector updated. The operation can be repeated until it is sent to a replicated node. This node has the latest update, then the nodes send the update to a subscribed nodes by using an information in vector updated.

III.2. Construction of an update tree

For each object sharing f, the solution construct a update tree statically d_ary (node has maximum $d = 2^b$ child nodes) includes the primary node of a key (f) in the identifier space is the root (denoted by R) and replica nodes. The method of construction is as follows:

Node R is responsible for all identifier space as [0, 2^{128}-1]. Node i in tree is responsible for the identifier space, denoted by ws_i and consecutive d-equal partitions denoted by ws_i^1, ws_i^2…ws_i^d. N_i^n denotes as a children of node i at the n^{th} position ($n = \overline{1,d}$).

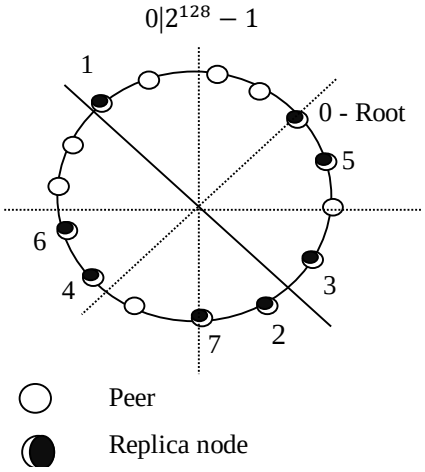

Figure 1: Peers in Pastry

Node j sends a request joining to the root R. It check $ID_j \in ws_R^n$ ($n = \overline{1,d}$). If R has not yet its children N_R^n, then j is linked to become N_R^n and $ws_j := ws_R^n$. Otherwise, R sends a request to N_R^n and this child which received the forwarded message conducts the same operation, and such a forwarding is repeated until the parent of j is determined. After linking into the update tree, a new node sends a request to the its parent for a data. The tree construction algorithm in pseudocode ID_LINK is shown as follows:

ID_LINK *d-Ary* Construction(R,j)

Input: node j which send request_join to R

Output: the parent of j

Begin

//at the root check

$ID_j \in ws_R^n$ // $n = \overline{1,d}$

If *has not yet its children* N_R^n **Then**

Return R

ws_j: = ws_R^n

Else if

R sends a request to N_R^n and the child which received the forwarded message conducts the same operation, and such a forwarding is repeated until the parent q of j is determined

//q *has not yet its children* N_q^n

//$ID_j \in ws_q^n$, $n = \overline{1,d}$

ws_j: = ws_q^n

Return N_q^n

End

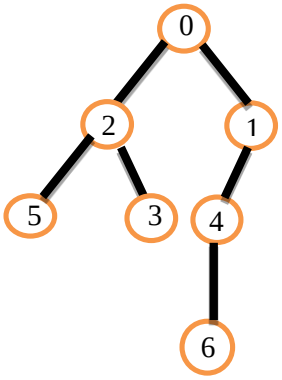

Figure 2: Dissemination tree illustration 2-Ary

The nodes denoted by 1,2, ... 6 are ordered by their arrival times in update tree 2-Ary (illustrated in Figure 2). At the beginning, node 1 is joined. Then node 1 sends a request to node 0 (root), it check $ID_1 \in ws_0^2$ and it has not yet N_0^2, so node 1 were assigned by node 0 as its children N_0^2 and of course ws_1: = ws_0^2. The same as node 2 is joined, the root check $ID_2 \in ws_0^1$ and it has not yet N_0^1. So that node 2 were assigned by node 0 as its children N_0^1 and ws_2: = ws_0^1. When node 3 joined, the root check $ID_3 \in ws_0^1$, but it has a node N_0^1 (node 2). Then the root sends a request to node 2. It check $ID_3 \in ws_2^2$ and has not yet N_2^2, so node 3 were assigned by node 2 as its children N_2^2 and ws_3: = ws_2^2.

Lemma: the cost of communcation or update propagation (following called propagation cost, measured in hop) between node i_l (node i at level l) and node j_k (or opposite) as:

$$cost_{i_l}^{j_k} = |l-k| * \log N - \frac{b*(l-1)^2}{2} + \frac{b*k^2}{2}$$

Where N is the number of peer in overlay network.

Proof: suppose $0 \le k < l$, the propagation cost between a nodes with level l and $l-1$ as $\log \frac{N}{d^{l-1}}$, so the propagation cost between node i_l and node j_k as:

$$\begin{aligned} cost_{i_l}^{j_k} &= \log\frac{N}{d^{l-1}} + \log\frac{N}{d^{l-2}} + \cdots + \log\frac{N}{d^{k}} \\ &= \log N - b*(l-1) + \log N - \\ &\quad b*(l-2) + \cdots \log N - b*k \\ &= (l-k)*\log N - \frac{b*(l-1-k)*(l-1+k)}{2} \\ &= (l-k)*\log N - \frac{b*(l-1)^2}{2} + \frac{b*k^2}{2} \end{aligned}$$

Denoted by p_l probability of a new node liked as at level l ($0 < l \le L$, $L = \log_d NR$ as height of update tree and NR is a number of tree nodes).

$$p_l = \frac{d^l}{NR}, \sum_{l=1}^{L} p_l = 1$$

Then, the cost of joining a new node can be expressed as follows:

$$cost_{join} = \sum_{l=1}^{l=L} p_l * cost_{i_{l-1}}^{R_0}$$

III.3. Basic Operations

a. The new update request

Each node uses a vector updated to record subscribers and to send to its parent. If a node is a replicated node (method of replication will be presented below), it forwards updates to the subscribers directly, elsewhere it sets the corresponding bit 1 in a vector updated (d bit) and continues to forward subscriber requests to the node at the next upper-level. This operation proceeds until it reaches the replicated node (or the root). It towards the updates to all subscribers using the information of vector updated.

The operation of subscription request of node i_l can be finished in hops as:

$$averager\_sub(i_l) = \frac{\sum_{k=l-1}^{k=0} cost_{i_l}^{j_k}}{l}$$

$$\begin{aligned}\sum_{k=l-1}^{k=0} cost_{i_l}^{j_k} &= \sum_{k=l-1}^{k=0} (l-k)*\log N - \frac{b*(l-1)^2}{2} + \frac{b}{2}*k^2 \\ &= \frac{l*(l+1)}{2}*\log N - \frac{b*l*(l-1)^2}{2} \\ &\quad + \frac{b*(l-1)*l*(2l-1)}{6} \\ &= \frac{l*(l+1)}{2}*\log N - \frac{b*l*(l-1)*(l-2)}{12}\end{aligned}$$

Therefore,

$$averager\_sub(i_l) = \frac{(l+1)}{2}*\log N - \frac{b*(l-1)*(l-2)}{12}$$

b. The update propagation

Any replica node can update on local replica but it should be submitted to the root. This is responsible for update propagation (or replication) to a childs and repeated until the subscibers received.

The update propagation performance in two ways as follows:

The first, the replication to a nodes which satisfy condition.

The second, each replicated node (include root) check the information in vector updated for knowledge about an subscribers for sending the child nodes and its continue repeating until the subscibers received. Note that, then the nodes receive also know the information of the replicated node. During any update is being replicated by the root, then new update that arrives to the root will be discarded.

Suppose the number of updates request (subscribers) at node i_l in period τ as a Poisson process, denoted by $n_{ru}^{i_l}$. Denoted by N_i as the number of nodes in subtree rooted with i; $h_i = \log_d N_i$ as height of subtree rooted with i.

Denoted by pu_m is that a probability of any node x_m $(l < m \le l + h_i \le L)$ subscribes to node i_l.

$$pu_m = \frac{d^{m-l}}{N_i}, \sum_{m=l+1}^{m=l+h_i} pu_m = 1$$

The average cost of updating for each subscriber among $n_{ru}^{i_l}$ as:

$$\text{averager\_update\_to}(i_l) + \sum_{m=l+1}^{l+h_i} pu_m * cost_{i_l}^{x_m}$$

At which easy to see:

$$\text{averager\_update\_to}(i_l) = averager\_sub(i_l)$$

$$= \frac{(l+1)}{2}*\log N - \frac{b*(l-1)*(l-2)}{12}$$

III.4. The optimal replication

Data sharing is replicated for high availability to reduce a latency of update, the tree balance, handing over possibility of bottleneck or overload of node. However, it also increases other costs as storage, a number of messages and bandwidth usage. The proposal calculates all costs based on the update rate at each node for deciding of replicating for the node. So that, the optimal replication reduces a latency without increasing the costs (even reduced).

Suppose the number of updates arrive at root in period τ as a Poisson process, denoted by $n_{ud}^{R_0}$. It is periodically collected and sent to all node to help with the optimal replication. The number of subscribers at i_l in period τ as $n_{ru}^{i_l}$. Moreover, node i_l know that node j_k is the replicated node (presented above).

The costs of update for $n_{ru}^{i_l}$ subscribers in case of replicating from j_k to i_l include: the cost of propagation $n_{ud}^{R_0}$ updates from j_k to i_l and the average cost of propagation from i_l to $n_{ru}^{i_l}$ subscribers, so that:

$$cost\_replica_{i_l}^{j_k} = n_{ud}^{R_0} * cost_{i_l}^{j_k} + n_{ru}^{i_l} * \sum_{m=l+1}^{m=l+h_i} pu_m * cost_{i_l}^{x_m}$$

The costs of update for $n_{ru}^{i_l}$ subscribers in case of not replicating from j_k to i_l include: the cost of sending of $n_{ru}^{i_l}$ subscribers from i_l to j_k and sending of $n_{ru}^{i_l}$ update from j_k to i_l, so that:

$$cost\_propagation_{i_l}^{j_k} = 2 * n_{ru}^{i_l} * cost_{i_l}^{j_k} + n_{ru}^{i_l} * \sum_{m=l+1}^{l+h_i} pu_m * cost_{i_l}^{x_m}$$

Note that in the two expressions above, we don't calculate the cost of sending for $n_{ru}^{i_l}$ subscribers from the subscribers to i_l. Because, in both cases this same cost.

Node i_l performs to compare two expressions above, if

$$cost\_replica_{i_l}^{j_k} < cost\_propagation_{i_l}^{j_k}$$

$$\Leftrightarrow n_{ud}^{R_0} < 2 * n_{ru}^{i_l}$$

Node i_l requets to replicate from node j_k. After that, node j_k is responsible to replicate for node i_l. Then the inequality above is not satisfied, so node i_l require to stop replicating.

We note that a values $n_{ud}^{R_0}$ and $n_{ru}^{i_l}$ are dependent of the updates rate and the total number of tree nodes (or subtree nodes). Then the solution of optimal replication above can be evaluated effectively when theses parameters are changed.

III.5. Maintenance for tree structure

Each node has an information of its parent and grandfather which helps to robust against frequent node churn and failures. In which, departure of node can be detected by through periodical message exchanging. For example, after a certain number of messages are sent without feedback.

When there is a change, relevant nodes will be passed parameters as the new identifier space that it is responsible. The proposal only requires very little structural change, so it is effective in the case of node's join/leave high-rate. Two cases of node departure as:

a. The actively leave of nodes

When a node is not interested in data sharing anymore, it is active-leave. If node *i* is a leaf node, then just notify to the parent *j* for this node to acknowledge and know the identifier space ws_j^n currently has no node responsible, else node *i* select a leaf node of the subtree rooted with i to substitute itself.

b. Node churn

If node churn is a leaf node, the parent will detect the abandonment and performance of the same operation as above. Else each child independent discovers this and they send a request of maintenance at the same time to them grandfather by proposing the substitution nodes (leaf nodes of each subtree). The grandfather chooses one of the proposed nodes to replace the node leaving. Especially, if the root is leaving, the overlay network is responsible for finding the new node instead.

IV. PERFORMANCE EVALUATION

The reseachers evaluate the performance of new proposal by using the OverSim simulation about the latency and the ratio of successful update by the impact of three parameters include the update rate, the total number of tree nodes and the churn rate respectively.

- The latency: the average delay propagation for all subscribed nodes to receive an update.
- The ratio of successful update: the ratio of the number of successful update to the total number of generated replicas.

In addition, the reseachers also compared to the results obtained to demonstrate the effectiveness of the new proposal with the proposed solution by Nakashima and Yi. The reason for choosing these solutions is because they both also use an update tree on structured p2p substrate and highly effective in latency (Nakashima) or the ratio of successful update (Yi).

IV.1. Simulation setting

The configuration parameters of Pastry include: network consisting of 5000 peers. The routing table of each node has 40 levels, each level consists of 15 entries, and the leaf set of each node has 32 entries.

The configuration parameters of distributed systems and simulation include: the length of each simulation is fixed to 1000 unit times. The number of data sharing ranges from 10^2 to 10^4 follows a Zipf's distribution [13]. The number of replica nodes of each data sharing follows the Zipf's distribution with parameter $s = 1, NR = 5000$. The heterogeneity of node capacities follows a Pareto distribution [14]. We set the shape parameter $a = 1$ and scale parameter $b = 5000$ to get 5000 different node capacities. The update rate, arrival and departure of nodes follows the Poisson distribution with default value $\lambda = 0{,}05$, $\lambda = 0{,}1$ respectively. The results points in the figures are the average values of 10 trials.

IV.2. The effectively latency

In Figure 3, the results show that the impact of parameter d on latency (100 replica nodes). When d has a small values, so the height of the tree is high then it will cost to propagate through many levels (as in the expression $cost_{i_l}^{j_k}$ above). Oppositely, the tree will have a small height which reduces the cost of propagation among levels but increasing the cost of propagation at the same level (as $\log \frac{NR}{d^l}$ at level l). So we see that when $d = 16$, the latency is smallest at 9.5

unit times. That is why, the researchers used the value d = 16 in later experiments.

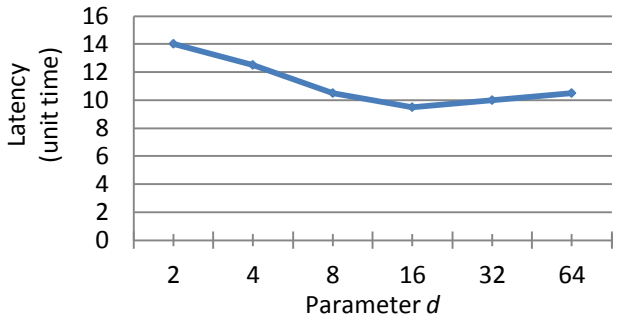

Figure 3: impact of parameter *d* on latency

The results as shown in Figure 4 when increasing the number of replica nodes (scalability), the proposal compares to the solutions by Nakashima and Yi to significantly reducing the delay of updates propagation as 16; 18,5 và 40,5 unit time ($NR = 1000$), respectively. There are a reasons as follows:

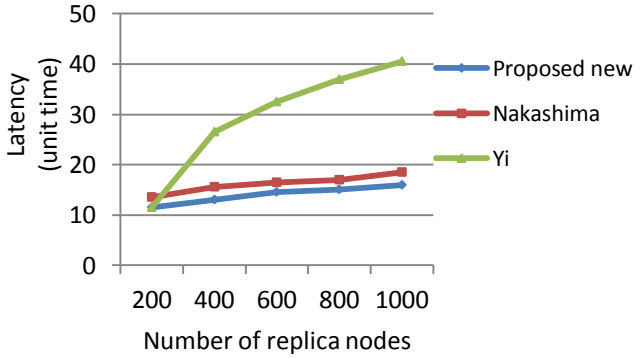

Figure 4: impact of the number of tree nodes on latency

Nakashima and Yi's solutions, the both use a method of update propagation from the root to all nodes (not just list of subscribed nodes). So that, when scalability there will have two causes for increasing latency. The first, this leads to increase the height tree, so the update propagation from the root is low-efficient. The second, of course the sending to all node also increases the latency. Besides, the both has a limitations in the mothods of construction and communication (presented above). Furthermore Yi's propagation also depends on the update rate of the all nodes, so Yi's solution is less effective in latency. For the proposal, the optimal replication that puts the replicas at the nodes can respond to the update requests faster than the root. This also includes the efficiency of communication because the nodes are linked logically as close by the method construction of the update tree and the solution only update the subscribed nodes, so reducing the number of nodes that need updating, also reducing the latency for other solutions.

The node's join/leave rate is the Poisson process that can be calculated by the ratio of time of that node online to a simulation cycle. When increasing this value, the solutions by Nakashima and Yi take more time to build or maintain the update tree due to the balance of the number of nodes in a subtrees and the ineffective of the communication. Moreover, Yi's solution also added time to transmit updates in the buffer. The proposal uses the identifier space to be responsible for each node which provides a solution that requires minimal alteration of the tree structure in case of node's join/leave. Then as a result shown in Figure 5, the proposal has a smaller latency. In particular, when the node's join/leave rate is high (≥ 0,4), it does not spike as in the solutions by Nakashima and Yi. Example 29; 48 and 45.5 units time in case of the node's join/leave rate at 0,5 by proposal, Yi and Nakashima respectively.

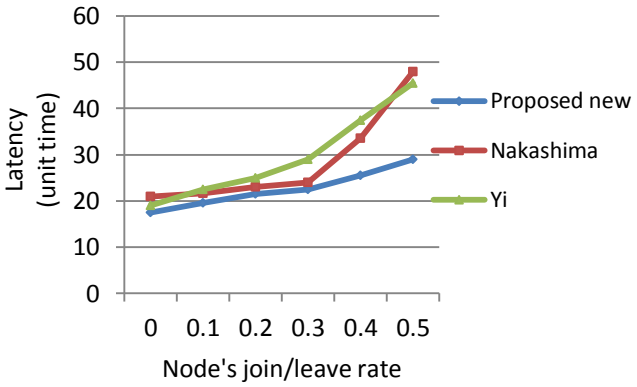

Figure 5: impact of the node's join/leave rate on latency

IV.3. The effectiveness of successful update

Although the latency is always greatest (in previous results), however the results from Figure 6 show that Yi's solution has a ratio of successful update of approximately 90%. The reason is that it uses buffer ($k = 20$ in the experiment) at each node (except leaf node). Only in case the root's buffer is full then the new update will be discarded. But the large values of k are only weakly consistent. Therefore, Yi's solution is applicable in the event of a need to update up to the

maximum number of generated replicas and to meet the balance between latency and consistency.

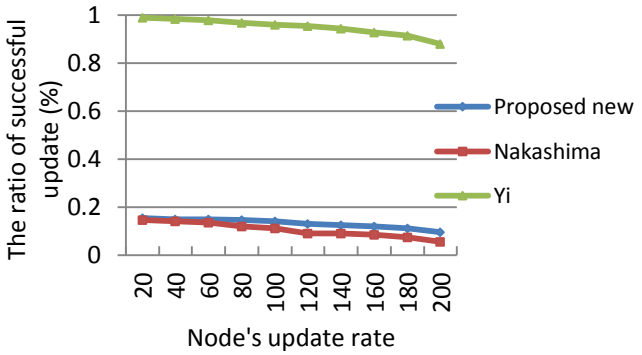

Figure 6: impact of the update rate on the ratio of successful update

The proposal has a higher ratio of successful update of Nakashima (both without a buffer) and when the higher the update rate, so the better the result (9.5% compare to 5.5% in the case of the update rate of 200). Because, the proposal always has smaller latency and more stable. Furthermore, Nakashima's solution updates from the root to all nodes. So the root will remove the new update until the previous update has propagated to all nodes. While with the proposal, the root only needs to propagate the new update to the replicated nodes.

V. CONCLUSION

This paper presents the efficient proposal which ensures consistent data sharing on structured p2p. It proposes a new way to build, maintain the update tree and perform the optimal replication based on the update rates, moreover, it uses the effective method of update propagation. The results of the experiments show that as increasing the node's join/leave rate, the update rate, and the replica nodes, so the proposal is more stable and effective than by Nakashima and Yi about the latency and the ratio of the successful update.

VI. REFERENCES

[1] G.Pierre, and M. V. Steen G. Urdaneta, *A decentralized wiki engine for collaborative wikipedia hosting*, WEBIST, pp.156-163, 2007.

[2] D. Schioberg, L.H.Vu, and A.Datta S. Buchegger, *Peerson: p2p social networking early experiences and insights*, Proceedings of the Second ACM EuroSys Workshop on Social Network Systems, ACM, pp.46-52, 2009.

[3] Jun, et al Wang, *Distributed collaborative filtering for peer-to-peer file sharing systems*, Proceedings of the 2006 ACM symposium on Applied computing, ACM, pp.1026-1030, 2006.

[4] A.Rowstron and P.Druschel, *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*, IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer Berlin Heidelberg, pp.329-350, 2001.

[5] Y.Zhao, J. D. Kubiatowicz, and A. D. Joseph, Tapestry: An infrastructure for fault-resilient wide-area location and routing, Technical Report UCB//CSD-01-1141, vol. 214, U. C. Berkeley, (April)2001.

[6] S. Ratnasamy, P . Francis, M. Handley, R. Karp, and S. Shenker, *A Scalable Content-Addressable Network*, Proc. of ACM SIGCOMM, vol. 31, no. 4, pp.161-172, (Aug)2001.

[7] David, and Jörn Kuhlenkamp. Bermbach, *Consistency in distributed storage systems*, Networked Systems, Springer Berlin Heidelberg, pp.175-189, 2013.

[8] Ingmar, Bernhard Heep, and Stephan Krause. Baumgart, *OverSim: A flexible overlay network simulation framework*, IEEE Global Internet Symposium, IEEE, pp.79-84, 2007.

[9] Nakashima, Taishi, and Satoshi Fujita, *Tree-Based Consistency Maintenance Scheme for Peer-to-Peer File Sharing Systems*, Computing and Networking (CANDAR), 2013 First International Symposium on, IEEE, pp.187-193, 2013.

[10]Yi, Hu, Laxmi N. Bhuyan, and Min Feng, *Maintaining data consistency in structured p2p systems*, IEEE Transactions on Parallel and Distributed Systems, pp.2125-2137, (23.11)2012.

[11]Zhenyu LI, Gaogang XIE, and Zhongcheng LI, *Efficient and scalable consistency maintenance for heterogeneous peer-to-peer systems*, IEEE Transactions on Parallel and Distributed Systems, pp.1695-1708, 19.12(2008).

[12]Xin, et al Chen, *SCOPE: Scalable consistency maintenance in structured p2p systems*, in 24th Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, pp.1502-1513, 2005.

[13]Lada A., and Bernardo A. Huberman. Adamic, *Zipf's law and the Internet, Glottometrics*, pp.143-150, 2002.

[14]Josef. Steindl, The Pareto Distribution, Palgrave Macmillan UK, Economic Papers, pp.321-327, 1990.

THE BRIEF'S AUTHORS

NGUYEN HONG MINH

Born in 1982.

Nguyen Hong Minh received the BS degree in Computer Science from People's Security Academy, Vietnam in 2005 and the MS degree in distributed systems and network from Université de Franche-Comté, France, in 2010. He is currently an teaching in the Department of mathematics and Information Technology at People's Security University. His research interests include distributed system and network.

Email: hongminhnguyen1982@gmail.com

NGUYEN XUAN HUY

Born in 1944

Nguyen Xuan Huy received the BS degree in mathematics from Leningrad University of Pedagogy, The Union of Soviet Socialist Republics, in 1973 and the in Ph.D. degrees Information Technology from Soviet Academy of Sciences, in 1990. His research interests include software technology, big data and distributed systems

Email: nxhuy564@gmail.com

NGUYEN HA HUY CUONG

Born in 1979

Ha Huy Cuong Nguyen received the M.S degrees Computer Science from Danang of University in 2010. From 2011 until 2016, he studied at the center DATIC, University of Science and Technology - The University of Da Nang. At the center of this research, he doctoral thesis "Studies deadlock prevention solutions in resource allocation for distributed virtual systems." He received the Ph.D. degrees Computer Science from Da Nang of University in 2017.