Heterogeneity Aware P2P Algorithm by Using Mobile nodeID

Kyungbaek Kim and Daeyeon Park

Department of Electrical Engineering & Computer Science,
Division of Electrical Engineering,
Korea Advanced Institute of Science and Technology (KAIST),
373-1 Guseong-dong, Yuseong-gu, Daejeon, 305-701, Republic of Korea
kbkim@sslab.kaist.ac.kr, daeyeon@ee.kaist.ac.kr

Abstract. The peer-to-peer systems have become an extremely popular platform for large-scale content sharing. A lot of research papers discussed the Distributed Hash Table (DHT) based p2p algorithms to promise that idle resources may be efficiently harvested. However, p2p systems are composed of components with extremely heterogeneous availabilities and for nodes which join/leave the system frequently, the system will generate a lot of information maintenance traffic such as routing information update traffic and data copy traffic to keep the efficiency of the DHT based p2p algorithms.

In this paper, we suggest the mobile nodeID based p2p algorithm to reduce the overhead by exploiting the heterogeneity of participant nodes efficiently. Unlike the DHT based p2p algorithms, the nodeID of a node changes according to its characteristic to support the p2p system efficiency and each nodes takes the different responsibility in accordance with its nodeID. We classify nodes into the two types according to the characteristics of nodes : the reliable nodes and the leaf nodes. The reliable node which is the more stable and more reliable node acts as the more important role of the routing and the replication. The leaf node which joins/leaves very frequently acts as the simple role to minimize the information maintenance traffic. The reliable node has the load-balanced ID to balance the loads and the leaf node has the load-free ID to reduce the responsibility.

We examine the efficiency of our p2p algorithm via a event driven simulation and show that the information maintenance traffic reduces and the routing process is more efficient.

Keywords: peer-to-peer, algorithm, mobile nodeID, heterogeneity.

1 Introduction

In these days, peer-to-peer systems have become an extremely popular platform for large-scale content sharing. Unlike client/server model based storage systems, which centralized the management of data in a few highly reliable servers, peer-to-peer storage systems distribute the burden of data storage and communications among tens of thousands of clients. The wide-spread attraction of this model arises from the promise that idle resources may be efficiently harvested to provide scalable storage services. A lot of research papers discussed the Distributed Hash Table (DHT) based p2p routing algorithms (Chord, Pastry, Tapestry and CAN) [2][3][4][5].

In contrast to traditional systems, peer-to-peer systems are composed of components with extremely heterogeneous availabilities - individually administered host PCs may be turned on and off, join and leave the system and have intermittent connectivity, and are constructed from low-cost low reliability components. For example, one recent study[6] of a popular peer-to-peer file sharing system found that the majority of peers had application-level availability rates of under 20 percent and only 20 percent nodes have server-like profiles. In such an environment, failure is no longer an exceptional event, but is a pervasive condition. At any point in time the majority of hosts in the system are unavailable and those hosts that are available may soon stop servicing requests.

A big issue in current DHT based p2p algorithms is the high overhead of maintaining DHT routing data structure and the stored data. When a node joins/leaves the system, the affected routing data structure on some existing nodes must be updated accordingly to reflect the change. Moreover, most p2p systems employ some form of the data redundancy to cope with failure and when the membership of nodes changes, these systems generate huge overhead of compulsory copies for the data availability. Especially, for nodes which join/leave the systems frequently, the p2p system will generate a lot of routing information update traffic and data copy traffic. It does not only increase the consumption of the network bandwidth, but also affects the efficiency of DHT based routing algorithms. Until now, DHT algorithms are not widely used in commercial systems yet, most p2p file sharing systems are still using non structured p2p mechanisms.

In this paper, we suggest the mobile nodeID based p2p algorithm to reduce the information maintenance overhead by exploiting the heterogeneity of participant nodes efficiently. Unlike the DHT based p2p algorithms, the nodeID of a node changes according to its characteristic to support the p2p system efficiency and each node takes the different responsibility in accordance with its nodeID. We classify nodes into the two types according to the characteristics of nodes : the reliable nodes and the leaf nodes. The reliable node which is the more stable and more reliable node acts as the more important role of the routing and the replication. The leaf node which joins/leaves very frequently acts as the simple role to minimize the information maintenance traffic. The reliable node has the load-balanced ID to balance the loads and the leaf nodes has the load-free ID to reduce the responsibility.

The reliable nodes are more stable and more reliable nodes and these nodes act as more important roles such as routing and replication. The reliable nodes have Load Balanced ID (LBID) which is evenly distributed and balances the workload of each reliable node. This LBID is dynamically assigned and the LBID routing table which helps for routing to any reliable nodes is also organized when the LBID is assigned. The leaf nodes join/leave very frequently on the system and the majority of the participant nodes are these leaf nodes. These nodes act as simple roles such as servicing the request and helping the reliable nodes. The leaf nodes have Load Free ID (LFID) which makes the ID region of a leaf node as small as possible and reduces the effect of the dynamic membership change which increases the information maintenance overhead. According to these basic behaviors, because the frequently joining/leaving of nodes occurs as the leaf nodes which make little overhead, we can reduce the overhead of the whole p2p system and achieve more efficient routing without the frequent updates.

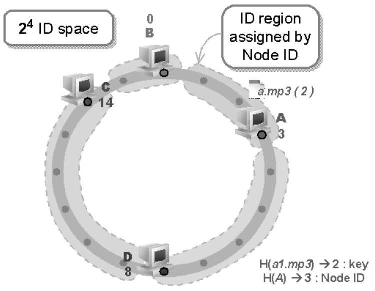


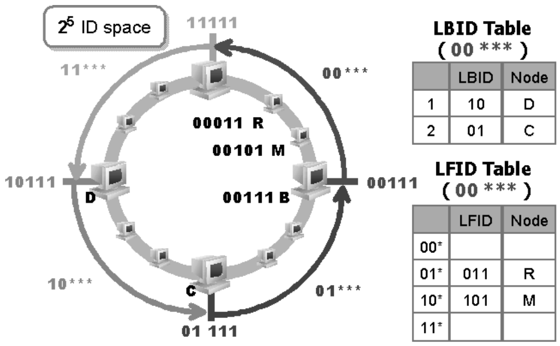
Fig. 1. Overview for the general DHT based P2P algorithm

This paper is organized as follow. In section 2, we describe the DHT based p2p algorithm and its problem. Section 3 introduces the detail of the mobile nodeID based p2p algorithm. The simulation environment and performance evaluation are given in section 4. Finally conclude in section 5.

2 Background

There are many DHT based p2p algorithms such as Chord, Pastry, Tapestry and CAN [2][3][4][5]. Each node has a DHT which is a small routing table and any node can be reached in about $O(logN)$ routing hops where the N is the total number of nodes in the system. To achieve this efficient and bounded routing, there are some rules for the organizing the participant nodes. First of all, each node has a unique nodeID which is taken by hashing any identifier of a node, and according to its nodeID it maps on the ID space where the nodes and the objects are co-located with the nodeIDs or the keys which are the hashed values of the nodes or the objects. In the figure 1, the node id of node A is 3 and it maps on the position for 3. After the mapping of the node id, each node knows its ID region from the next position of its previous nodeID to its nodeID and each node should store and service the objects for its ID region. In the figure 1, node A takes its ID region from 1 to 3 because its node id is 3 and its previous node B locates on 0 and when a node wants to get a.mp3 whose key is 2, node A gets the request for it.

Though these well-organized rules make the routing of the p2p system efficient and bounded, a big issue in current DHT based p2p algorithms is the high information maintenance overhead of maintaining DHT routing data structure and the stored data. Because its node id is already given by the hashing function and its position on ID space is already fixed, when a node joins/leaves the p2p system, the ID region of its neighbor nodes changes and the stored data should be copies for the new ID region to service the right and reliable object, and the update of the routing table is also needed. In figure 1, if node A leaves, the ID region of node D changes and the object from 1 to 3 should be copied from node A to node D. Moreover, the affected routing table which has the entry with node A must be updated. In this case, one recent research[6] of a popular p2p file sharing system found that 80 percent of total nodes of a p2p system join/leave very



LBID Table
(00 ***)

	LBID	Node
1	10	D
2	01	C

LFID Table
(00 ***)

	LFID	Node
00*		
01*	011	R
10*	101	M
11*		

Fig. 2. Overview of Mobile nodeID based P2P Algorithm

frequently and the majority of nodes have the application-level availability rate of under 20 percent. In such an environment, the information maintenance overhead is getting worse and this overhead discourages that the DHT based p2p systems are deployed to the real world.

3 Mobile nodeID Based P2P Algorithm

3.1 Overview

Previous DHT based p2p algorithms lack the explicit methods for exploiting the heterogeneous characteristics of participant nodes. The main reason of this lack is the static nodeID which makes the location of a node fixed on the ID space, and the system with the static nodeID is not flexible. We address this problem with the mobile nodeID which changes according to the characteristics of a node. We classify the participant nodes into two types : reliable nodes and leaf nodes. The reliable nodes are more reliable and more stable nodes and the leaf nodes join/leave very frequently. The nodeID of a reliable node is well distributed on the ID space and makes that the each reliable node gets fair ID region and balanced loads. The leaf node gets the node id which makes its ID region as small as possible to minimize the information maintenance overhead for joining/leaving of it.

Figure 2 shows the overview of the p2p system that uses the mobile nodeID based p2p algorithm. The participant nodes are on the 2^5 ID space and the number of bits for a nodeID is 5. The nodeID is consist of the *Load-Balanced ID (LBID)* and the *Load-Free ID (LFID)* and ,in this example, the first 2 bits of a nodeID mean the LBID and the other 3 bits are for the LFID. In this figure, the large sized computer means the reliable node and the small sized computer means the leaf node. To distribute the participant nodes efficiently, we divide the ID space into many *sub-regions* which are the balanced ID regions. Each sub-region has one reliable node which represents this region and many leaf nodes which assist the reliable node. That is, the reliable node is mainly responsible for the objects for the sub-region and the leaf nodes service the objects for the small ID region which is assigned by both of their node id and the LFID

```
If No Reliable Node
    $LBID_{new}$ = set all bits of LBID to 1
    Level = 1
Else
    If( Join ){
        If( Level < Levelthres ){
            // Accept Join
            $LBID_{new}$ = set exclusive bit of $Level_{th}$ bit of $LBID_{target}$
            LBID RT entry Update
            $Level_{th}$ RT entry = $LBID_{new}$
            Level++
        }
    }
    Else{
        If( Temporal Routing Entry )
            Forward Join request to this entry
        Else If( There is no temporal routing entry )
            If( sending node != last RT entry )
                Forward Join request to next RT entry of sending node
            Else
                // finalize
                Sending Notification of Finalization to all of RT entries
    }
```

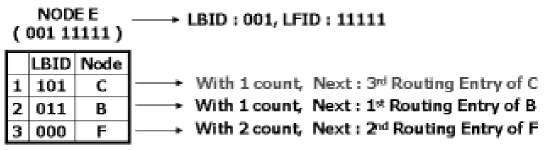
Fig. 3. Basic algorithm of the LBID assignment

table on behalf of the reliable node. For example, when a node wants to get an object whose key is 00001, the reliable node B takes the request, however when a node tries to get an object with 00010, the leaf node R takes the request to assist the reliable node, because the ID region of node R is from 00010, the start of the LFID slot to 00011, its nodeID. All nodes on the same sub-region have the same LBID and they are identified by the LFID. The all bits of LFID for the reliable node are set to 1 and the LFIDs of other leaf nodes change according to the behaviors of them.

The LBID table and the LFID table are used to lookup the location of a node or an object. When a node joins, we get its static nodeID by hashing its identifier. The first thing is to route to the right sub-region according to the LBID table. In this case, a node which gets a join request forwards it to the next node which is the node of the most prefix matched entry of the LBID table. After finding the sub-region, the LFID table assigns the right LFID to the new node. This join process is only for the leaf nodes and in the next section, we show the detail of the whole join process. Moreover, when a node tries to lookup an object, it sends a lookup request with the object key which is its hashed value to any other participant node. Like the case of the join process, it forwards to the right sub-region by the LBID tables and find the node whose ID region is responsible for the key by the LFID tables. Basically, we use the hashed values of both of nodes and objects, but they are used only for finding the location of them. When the nodes join to the p2p system, their ids are newly assigned by the p2p system and change according to its characteristics for the nodes to be locate on fit places.

3.2 Mobile nodeID

Load Balanced ID is the identifier for the reliable node. Each reliable node is assigned this LBID and it is responsible for storing and servicing the request for the fair and balance ID region. For this, the LBID is well distributed and evenly divided. Moreover, because there is no routing information, we need LBID routing table which helps routing to any reliable nodes.



NODE E (001 11111) → LBID : 001, LFID : 11111

	LBID	Node	
1	101	C	→ With 1 count, Next : 3rd Routing Entry of C
2	011	B	→ With 1 count, Next : 1st Routing Entry of B
3	000	F	→ With 2 count, Next : 2nd Routing Entry of F

Fig. 4. LBID Routing Table of node E and finalization rule

This LBID is assigned after a node finds the any reliable nodes. If there are not enough reliable nodes, a new node is assigned the LBID and acts as the reliable node without any relation of its real characteristics. A new reliable node is assigned the right LBID and create the LBID routing table according to its LBID. Each reliable node has the state information such as Join, Level, Full and Leaf. When the Join bit sets to 1, this node can process the join request and create new LBID for the new node. The Level bit is the depth value which means how many join requests is processed in this node, that is, how many routing entries are filled. The Full bit sets to 1 after the enough reliable nodes join the p2p system and they are ready to get the leaf nodes. The Leaf bit means the number of leaf nodes which is connected to a reliable node. According to these state information, LBID is assigned automatically and correctly.

The basic algorithm for the LBID assignment is in figure 3. When a node joins to the system and there is no reliable node, the new node has the new LBID whose all bits set to 1. Otherwise, when any reliable node gets a join request, it creates new LBID based on the two information which are its LBID and the Level bit. That is, the $level_{th}$ bit of LBID sets to the exclusive bit and this is the new LBID. This simple rule makes the difference of LBID of any two closest reliable nodes even and each reliable node gets the balanced and fair ID region. The LBID routing table which is used for routing to any reliable nodes is also organized when the new LBID is created. The basic rule is the N_{th} entry of the routing table has the node information whose N_{th} bit of LBID is exclusive to the owner's LBID. In figure 4, the node whose LBID is 001 has the LBID routing table whose 1st entry has the information for the node C whose LBID is 101 and 2nd entry has the information for the node B whose LBID is 011. These bit-wise exclusive entries make the LBID routing table and any node can reach any other nodes. This LBID routing table has $logN$ of routing entries and the maximum routing hops are limited to $logN$, where N is the number of LBID bits. When there is not proper node information for a routing entry, we set the temporal routing entry. This temporal routing entry has the node information which does not matched but closest to the right node information. When the node which has the temporal routing entry gets a join request, it forwards the join request to the temporal node which is ready to process join request. After this forwarding process, the new node replaces the temporal routing entry with right routing entry and the LBID routing table is composed completely.

After the enough number of reliable nodes join, every reliable nodes should set the Full bit to 1 and be ready to get leaf nodes. When a join request routes to the reliable node by the LBID routing table and every node can not process the join request, the last requested node knows that the reliable nodes are assigned fully and the finalize mechanism should start. In this case, each node does not know the whole of the reliable

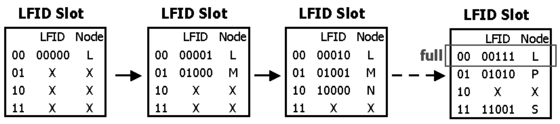


Fig. 5. LFID slot and its change according to the lifetime

nodes, but only knows the $logN$ routing entries. According to this, one node can not notify to all node and needs the efficient and systematic notification. To achieve this notification, a node sends the notification with TTL count value to every nodes which is on the routing table. Like figure 4, the notification to the N_{th} routing entry has $N - 1$ TTL count value, except the 1st entry whose notification has 1 TTL value. The target node which gets the notification reduces the TTL value by 1 and sends the notification to the $M - 1_{th}$ routing entry, where M is the position of the target node on the routing entry of the sending node. However, if the target node is 1st routing entry of the sending node, it sends the last notification message to its last routing entry.

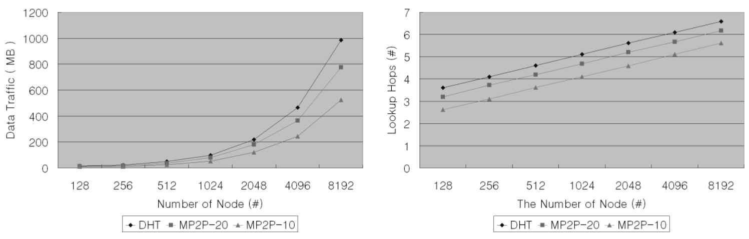
Load Free ID is the identifier of the leaf nodes. Because each leaf node is an unreliable node which joins/leaves very frequently, we should minimize the management cost for the effect of the dynamic membership change by assigning little load to leaf nodes. At first, LFID is close to 0 and when the access time of the leaf node increases, LFID also increases for the node to get more load and help the p2p system.

A new node routes to a reliable node by the LBID of the unique node key. When the Full bit of the reliable node is 1, this node processes the join request and increases the number of the Leaf bit by 1. To help assigning LFID, every reliable node has the LFID slot which divides the sub-region and each slot manages the leaf node information. The figure 5 shows the LFID slot and its change according to time. When node L joins, the first slot which is 00 slot assigns the LFID 00000 to node L. After time pass, the LFID of node L changes to 00001 and new node M gets 01000 for the second slot which is 01 slot, and so on. Each node is responsible for storing and requesting the data for the id space from the LFID 000 to the current LFID for each slot. When the LFID changes the data copy occurs, but this traffic is smaller than the management traffic of previous DHT, because these leaf nodes are free for data availability. Though each LFID increases according to the time, it can not increase more than the slot size.

4 Performance Evaluation

4.1 Simulation Setup

We make our p2p simulator which emulates the node behavior on the application layer. We implement the previous DHT based p2p algorithms such as pastry and chord and our mobile nodeID based p2p algorithm. We use 160 bit ID space to identify nodes and the number of LBID bits changes according to the number of the representative nodes which are assigned by the total number of the participant nodes. To make this dynamic characteristic, we use poison distribution whose average is 4, and to assign join/leave duration of a node, we use exponential distribution. According to this poison



(a) Bandwidth Usage (b) Lookup Hops

Fig. 6. Comparison of the bandwidth usage and the lookup hops

distribution, the lifetime of 80% of total nodes is below 60% of total simulation time, that is, only 20% of total nodes have the reliable server-like profile. Recent researches [6] measure the life distribution of the p2p nodes and show the similar distribution, and we can tell that this distribution is similar to the real world. This characteristics of nodes are assigned when the nodes are created and by using the exponential distribution with this characteristics, we can generate the on-time for which the nodes are on the p2p system and the off-time for which the nodes are off.

In the next results, DHT means the DHT based p2p algorithms and MP2P-N means the mobile nodeID based p2p algorithm in which the N percent nodes of total nodes act as representative nodes. That is, MP2P-20 can have the sub-region twice as many as MP2P-10. According to the number of the sub-region, the system makes up the bits of LBID and the bits of LFID.

4.2 Bandwidth Usage

The main problem of the current DHT p2p is the high management cost. In the figure 6(a), we show how the mobile nodeID based p2p algorithm reduces the management cost. To evaluate this cost, we assume that each node obtains same number of objects, that is, if the total number of nodes is 100 and the total number of objects is 10000, and if the total number of nodes is 200, the number of objects is 200000. In this case, the our p2p algorithm reduces the data management cost extremely. The main reason of this improvement is the behavior of leaf nodes. In DHT p2p, the frequent join/leave of leaf nodes cause the compulsory copies and update cost for routing information. However, in our p2p algorithm, the dynamic behavior of leaf nodes does not affect the data availability and the routing efficiency. According to these, the MP2P can reduce more management traffic. Moreover MP2P-10 reduces more traffic than MP2P-20. On the same node characteristics, the MP2P-20 needs more reliable nodes than MP2P-10, and the average availability of the reliable nodes of the MP2P-20 is less than the MP2P-10. In the MP2P-20, the transitions for the reliable nodes occurs more than the MP2P-10 and MP2P-20 exhausts more network bandwidth than MP2P-10. To prevent this side effect, we need the adaptive method which manages the number of reliable nodes according to the state of the nodes and this work is our ongoing job.

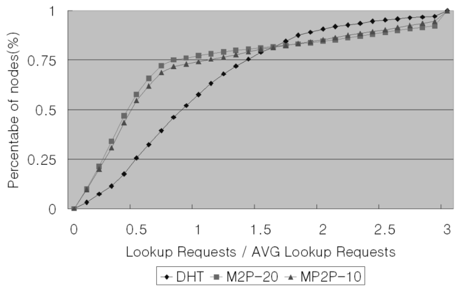


Fig. 7. Load distributions of whole participant nodes

4.3 Lookup Hops

In the p2p system, the lookup cost is also important parameter for the scalability because there are too many participants. Figure 6(b) shows the comparison of the lookup hops. For all algorithms, the lookup hops are proportion to the Log N, where N is the total number of nodes. The mobile nodeID based p2p algorithm performs more efficient lookup than normal DHT. The reason is that the our p2p algorithm uses the reliable nodes to route the lookup request and the number of these nodes are much less than the total nodes. These reliable nodes are more stable and more powerful than other nodes and they are durable nodes for the many routing requests. Additionally, the leaf nodes assist the reliable nodes to take the request for the ID region and the load of the reliable node are reasonable.

4.4 Load Balance

The figure 7 shows the load distribution for the total nodes. In this figure, we define the load of a node as the number of lookup requests of it divided by the average number of lookup requests of whole nodes. As the nature of the previous DHT based p2p algorithm, the load is distributed to the whole of nodes by the shape of the normal distribution and the average load of nodes is nearly 1. This behavior causes the heavy information maintenance overhead because the nodes which join/leave very frequently can be responsible for the big ID region. On the other hand, in our mobile nodeID based p2p algorithm, the load distribution can be classified into the representative nodes and the leaf nodes. About 75 percent of nodes have less load than other nodes because these nodes act as leaf nodes which join/leave frequently and they takes the responsible for small ID region which is assigned by the LFID. The average load of the leaf nodes are about 0.4 and these nodes are distributed uniformly. Otherwise, the representative nodes take much more load because they are alive for a long time and represent for the subregion. The average load of these nodes are about 2. This feature which classifies the load according to nodes is very useful for the p2p system on the heterogeneous network which is consist of the various nodes such as servers, workstations and PCs. Some p2p approaches need the server-like components to increase the efficiency, and our algorithm can exploit these components easily and efficiently because the server-like nodes locates for the reliable nodes automatically.

5 Conclusions

In this paper, we suggest the mobile nodeID based p2p algorithm to reduce the information maintenance overhead by exploiting the heterogeneity of participant nodes efficiently. Unlike the DHT based p2p algorithms, the nodeID of a node changes according to its characteristic to support the p2p system efficiency and each node takes the different responsibility in accordance with its nodeID. The reliable node which is the more stable and more reliable node acts as the more important role of the routing and the replication. The leaf node which joins/leaves very frequently acts as the simple role to reduce the information maintenance traffic. The reliable node has the load-balanced ID to balance the loads and the leaf nodes has the load-free ID to reduce the responsibility. This algorithm is very good for the p2p system on the heterogeneous environment which is consist of the various kinds of nodes such as servers, workstations and PCs, because it locates the server-like nodes at the positions for the reliable nodes and can exploit these nodes efficiently. However, our algorithm may over-provision for the reliable nodes and this may decreases the performance of our algorithm. The adaptive method for the whole state of nodes to keep the proper number of reliable nodes is our ongoing job.

References

1. K.Kim and D.Park. Efficient and Scalable Client Clustering For Web Proxy Cache. *IEICE Transaction on Information and Systems*, E86-D(9), September 2003.
2. I.Stoica, R.Morris, D.Karger, M.F.Kaashoek, and H.Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. *In Proceedings of ACM SIGCOMM 2001*, August 2001.
3. A.Rowstron and P.Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. *In Proceedings of the International Conference on Distributed Systems Platforms(Middleware)*, November 2001.
4. B.Y.Zhao, J.Kubiatowicz, and A.Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *UCB Technical Report UCB/CSD-01-114*, 2001.
5. S.Ratnasamy, P.Francis, M.Handley, R.Karp, and S.Shenker. A scalable content-addressable network. *In Proceedings of ACM SIGCOMM 2001*, 2001.
6. S. Saroiu et al. A measurement study of peer-to-peer file sharing systems. *In Proceedings of MMCN 2002*, 2002.
7. R. Bhagwan, K. Tati, Y. Cheng, S. Savage and G. M. Voelker. Total Recall: System Support for Automated Availability Management. *In Proceedings of NSDI 2004*, 2004.