Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

BulkTree: an overlay network architecture for live media streaming*

GONG An†1, DING Gui-guang2, DAI Qiong-hai2, LIN Chuang1
(1Department of Computer Science & Technology, Tsinghua University, Beijing 100084, China)
(2Broadband Networks & Digital Media Laboratory, Department of Automation, Tsinghua University, Beijing 100084, China)
†E-mail: ga04@mails.tsinghua.edu.cn

Received Dec. 15, 2005; revision accepted Feb. 19, 2006

Abstract: Peer-to-peer (P2P) systems are now very popular. Current P2P systems are broadly of two kinds, structured and unstructured. The tree structured P2P systems used technologies such as distributed hash tables (DHT) and hierarchical clustering can search the required target quickly, however, in a tree, the internal node has a higher load and its leave or crash often causes a large population of its offspring's problems, so that in the highly dynamic Internet environment the tree structure may still suffer frequent breaks. On the other hand, most widely used unstructured P2P networks rely on central directory servers or massive message flooding, clearly not scalable. So, we consider both of the above systems' advantages and disadvantages and realize that in the P2P systems one node may fail easily, but that when a number of nodes organized as a set, which we call "super node", the set is robust. Super nodes can be created and updated aware of topology-aware, and used with simple protocol such as flooding or "servers" to exchange information. Furthermore the entire robust super node can be organized into exquisite tree structure. By using this overlay network architecture, P2P systems are robust, efficient, scalable and secure. The simulation results demonstrated that our architecture greatly reduces the alteration time of the structure while decreasing the average delay time, compared to the common tree structure.

Key words: Peer-to-peer (P2P), Overlay networks, Scalability, Live media, Distributed hash tables (DHT), Hierarchical clustering
doi:10.1631/jzus.2006.AS0125 **Document code:** A **CLC number:** TN919.8

INTRODUCTION

Peer-to-peer (P2P) systems are now attracting more attention both from the scientific and commercial domains. In contrast to Client/Server (C/S) Systems, P2P systems (Schollmeier, 2001) yield advantages of scalability and lower cost by aggregating all the peers' resources to provide services. Due to the single peer's lack of global knowledge about suitable sources and the unpredictable dynamics of overlay network, how to find appropriate service peers at query time in P2P systems is a challenging problem.

In this paper, we propose a new overlay network architecture for live media streaming, called BulkTree, which is a tree structure built on top of Super-Node, which is an unstructured set of joined nodes. The main design idea is simple. In tree structure one node has limited resource and may fail easily, whereas a number of these Weak-Nodes can be organized into a strong super node to provide service. Furthermore the entire robust super node can be constructed as an exquisite tree structure. By using this overlay network architecture, P2P systems have three salient features: (1) robustness, as super node conceals the single node's joining and disruption; (2) efficiency, as the tree structure just needs to manage the super nodes, which limits the communication cost and system overload of the total system, and what is more, the super node can be organized to be topology-aware and the protocol is simple but efficient; (3) scalability and security, as the join/departure procedure of the node influences the tree structure only little; and by using some reasonable policies we can control the effect of the only malicious node in the super node.

* Project (Nos. 60502014 and 60432030) supported by the National Natural Science Foundation of China

RELATED WORK

P2P systems can be broadly classified into two categories: structured and unstructured. In this part we briefly overview the existing P2P paradigms.

Structured P2P paradigms

Many overlay systems construct and maintain an efficient distribution tree to stream media data, aiming to be as close as to the tree of IP multicast.

The key issue of these systems is how to organize structures for both control and data delivering. Some systems such as CoopNet (Padmanabhan *et al.*, 2002) choose the centralized algorithm, using the source server to build and maintain the tree. The server should have full knowledge of the information in all the nodes so that this model is efficient but need a powerful server. Another kind of system utilizes client nodes with server node to structure architecture like Narada (Chu *et al.*, 2000), NICE (Banerjee *et al.*, 2002) and ZIGZAG (Tran *et al.*, 2003). Narada maintains and optimizes a mesh that interconnects peers, and is constrained to only small scale level applications. On the other hand, NICE and ZIAZAG adopt hierarchical clustering principles and can scale to large number of peers. Our system, BulkTree, is similar to this kind in the tree structure face. Distributed hash table (DHT) is a third category of tree structure builder. Lookup (or routing) protocols employ substrates such as CAN (Ratnasamy *et al.*, 2001), Pastry (Rowstron and Druschel, 2001), Tapstry (Zhao *et al.*, 2004) and locate the requested for object within a logarithmic number of steps by distributed hash functions. However DHT masks the peer heterogeneity as well as network topology and conditions (except that Pastry exploits some network locality properties). DHT also can be used in our system to hash the Super-Node ID to build tree.

P2P systems are highly dynamic but delicate tree structures with difficulties to adjust correspondingly. What is more, the load is unbalanced in that an internal node's load is higher than that of a leaf node in the tree structure.

Unstructured P2P paradigms

Though structured P2P systems can efficiently find the requested object, building and maintaining an exquisite structure is really big work. Unstructured P2P systems do not rely on any structure. There are also three solutions to multicast message for unstructured P2P systems' lookup (or routing) algorithms. The first kind of systems such as Napster (www.Napster.com) and Maze (Chen *et al.*, 2004) use central servers to maintain the information on shared files stored on peers. Clearly, more users need more powerful servers. Our system adopts a simple algorithm similar to this to be used inside the unstructured Super-Node. Another kind of lookup algorithm in use is to blindly "flood" a query to the network among peers (such as in Gnutella) (www.Gnutella.com) or among super nodes (such as in KazaA) (www.KaZaA.com). Unfortunately the flood behavior restrains greatly the bandwidth of the net. Gossip (or epidemic) (Ganesh *et al.*, 2003; Eugster *et al.*, 2004) algorithms have recently become popular approaches to distribute messages in P2P systems. For example, DONet (Zhang *et al.*, 2005) employs a gossiping protocol for membership management. Gossip algorithms can enormously reduce the query load. Compared with structured systems, unstructured P2P systems are simple, but the delay to find the sought for object is large.

DESIGN AND OPTIMIZATION

The basic idea of BulkTree protocol is to organize close nodes into robust and uniform Super-Nodes and build the multicast tree on top of these nodes. The nodes inside the Super-Node are unstructured while the Super-Node tree is structured. In this paper we use end-to-end latency as the distance metric to find the close node. In the rest of this section, we describe how to construct BulkTree tree structure and Super-Nodes, policies to be taken to adjust node join/departure and optimization at the end of this part.

Multicast tree

As shown in Fig.1, Super-Nodes are used to replace the single Weak-Node as the element to build Multicast Tree.

Super-Node

The following properties hold for Super-Node:

(1) Super-Node is close-by Weak-Nodes set and works as the "node buffer".

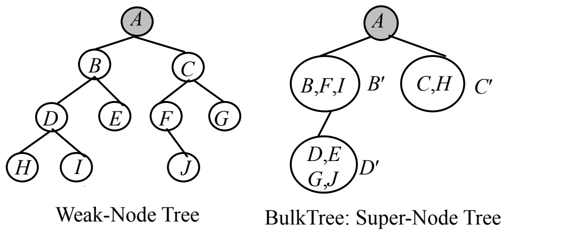

Fig.1 Weak-Node tree and BulkTree (node *A* is the server)

(2) Each internal Super-Node has its size bounded between k and $3k-1$, where k is a constant (for example, also as the internal node's children number's lower limit).

(3) Each leaf Super-Node has its size bounded between 1 and $3k-1$.

(4) Each Super-Node has a leader node and a vice leader. The leader is chosen by all the nodes inside and the vice leader is assigned by the leader as the backup.

The leader node acts as the server (the character like the server in Napster). It collects detailed information on parent Super-Node, other Weak-Nodes in the same Super-Node and its children Super-Node's leaders. The vice leader node has backup. Other Weak-Nodes only store the leader nodes' address information of parent and children. In BulkTree System the media data are delivered in pull model. In multicast the Super-Node first schedules data inside and then queries its parent Super-Node if needed. By this way the query message load is largely restrained. If the parent has the data, the leader of parent chooses k good nodes to send sequential $1/k$ data to the receivers. Obviously this multi-to-multi scheme can greatly improve the server ability in contrast to the pure tree's one-to-multi scheme.

Node join and departure

In P2P systems, nodes can join and departure arbitrarily. And in live streaming many just join the session to see what program is going and then leave after a few seconds or minutes. Each join or departure may cause the tree structure to be unbalanced or broken. To avoid this, splitting and merging nodes algorithms are taken into the tree structure. However that ability is somehow not equal to that of highly dynamic P2P systems. So we proposed the approach "node buffer" called Super-Node to alleviate the influence of frequent join/departure.

Node join

Whenever a new node which joins the multicast tree must not violate the rules of Super-Node and tree balance. We propose the join algorithm below.

In a join scene, the newly joined node H first contacts the server and the server redirects newly joined node to its children. The rest of the steps are listed below. $D(X, Y)$ denotes the currently end-to-end delay from X to Y, $Super(S)$ means the Super-Node set containing S, $Leader(S)$ denotes the last received leaders set from S, $Addable(T)$ denotes that after a new node joins the boundary of Super-Node, its size is not exceeded.

```
1. Select a node $X$ from $Leader(S)$: $D(H, X)$ is min;
2. If $Super(X)$ is a leaf
3.     Add $H$ to the $Super(X)$;
4. Else
5.     If $Addable(X)$
6.         Add $H$ to the $Super(X)$;
7.     Else if another node $Y$ in $Leader(S)$ is addable
8.             Add $H$ to the $Super(Y)$;
9.         Else send $Leader(X)$ to $H$ for contact;
```

The join overhead is $O(\log N)$ in terms of the number of peers to contact. If the join procedure terminates in Step 3 at the leaf Super-Node X, two states are distinguished. If the new size of Super-Node is still in $[k, 3k)$, no further work is needed. Otherwise the split and balance algorithm is taken:

(1) Divide the Super-Node X into sets U and V ($|U|, |V| \in [k, 3k)$), and the leader and vice leader of X become the leaders of U and V respectively. Then the two leaders assign their vice leaders as backup.

(2) If the parent Y of Super-Node X's children count is no more than k, take U and V as Y's children. Otherwise, make V as the child of U.

(3) If the tree structure is balanced, nothing is done. Otherwise balance approach is to be done. The balance algorithm is the same as that of the tree structure, which is well known, so we do not show it here (one approach presented coupled with split and merging in later "Optimization" subsection).

We take an example to explain the benefit from using Super-Node concept. Current conditions are as shown in Fig.1. Nodes K, L, M, N, O, …, and T (close to B, next to D, I) join the session in sequence.

Fig.2 and Fig.3 show the difference. That is the first structure change caused by the node K joining Weak-Nod Tree and caused by the node T joining BulkTree.

Node departure

To obey the rules of Super-Node and tree balance, the algorithms should be designed to handle a node departure. In BulkTree, the departure can be detected by leader or vice leader (if both are down, by all the peers together) whenever the node leaves either gracefully or accidentally due to crash.

Consider a node H who departs. If the peer is the leader, then the vice leader becomes the leader to schedule and assign a new vice leader. The leader then detects its Super-Node size. If the condition $[k, 3k)$ is satisfied, then no further work is needed. Otherwise the merging is taken. The merging has the following steps.

(1) If the Super-Node X is a leaf node, then do the rest of the steps. Otherwise X selects a leaf Super-Node Y from its offspring. If $|U|+|V|\in[k, 3k)$, merge X and Y into a new node Z. If not, take away k nodes from Y and merge these nodes to X.

(2) If the tree structure is balanced, nothing is done. Otherwise balance approach is to be done.

The departure example is given below. Current conditions are as shown in Fig.1. Peers B, C, D, E and F leave the session in sequence.

Fig.4 and Fig.5 show the difference. We notice that the first structure change caused by the node B leaving Weak-Nod Tree while caused by another structure change is the node F leaving BulkTree. Our system has more resilience especially to those peers who just join for a while.

Optimization

We now discuss a few optimizations to enhance BulkTree system performance. The first optimization is motivated by the observation that a newly joined node always received the media data after some time. The reason is that the node should firstly find its proper Super-Node to join before receiving data. To accelerate the node's play process, two ways are designed. One way is to create virtual Super-Node. The server is the leader of it. Every newly joined node can be a member of the virtual Super-Node and exchange data before the node joins its right Super-Node. The other way is to utilize the leaf Super-Node to broadcast data to new node. The server can collect some up-to-date leaf Super-Nodes information and assigns one to the newly joined node as its temporary Super-Node.

The second optimization seeks to make the distribution more quickly. As stated before, the distribution scheme in BulkTree is multi-to-multi. For a receiver there are many candidate senders. However the overlay links are not actually physical links, some overlay paths may share the same physical path. To perform better, the leader node can use tools such as traceroute to build the physical topology and select the "good" senders.

To avoid the overhead of splitting and for conve-

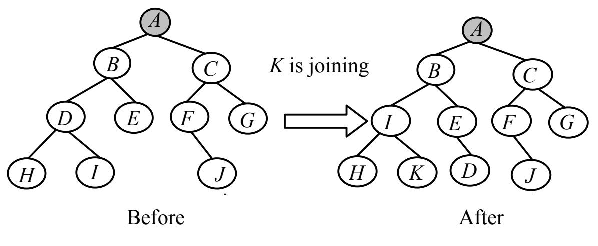

Fig.2 Join (Weak-Node Tree): First structure change (k=2)

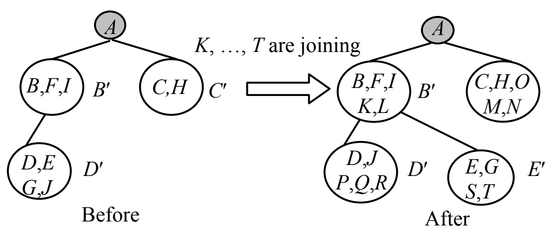

Fig.3 Join (BulkTree): First structure change (k=2)

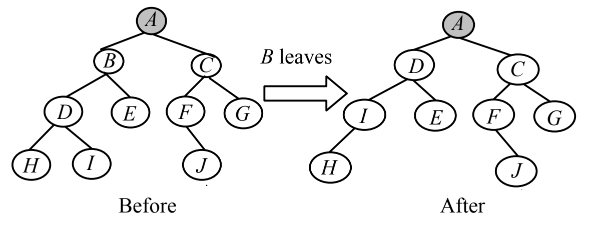

Fig.4 Departure (Weak-Node Tree): First structure change (k=2)

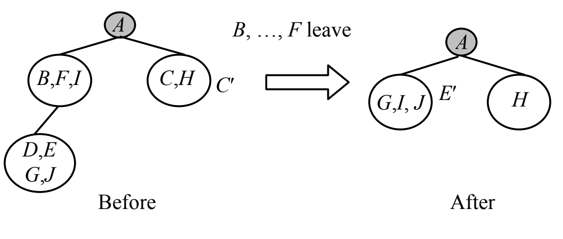

Fig.5 Departure (BulkTree): First structure change (k=2)

nience,we can optimize the split and balance algorithm below:

(1) Divide the Super-Node X into sets U and V ($|U|$, $|V|\in[k, 3k)$), and the leader and vice leader of X become the leaders of U and V respectively. Then the two leaders assign their vice leaders as backup.

(2) Make U and V as Y's children (Y is the parent Super-Node of X).

(3) If the parent Y's children count is no more than 2k (a Super-Node can afford to feed $2k$ children), nothing is done. Otherwise spilt the Super-Node Y.

And the merging and balance algorithm changes to:

(1) If the Super-Node X is a leaf node, then do the rest of the steps. Otherwise X selects a leaf Super-Node Y from its offspring. If $|U|+|V|\in[k, 3k)$, merge X and Y into a new node Z. If not, take away k nodes from Y and merge these nodes to X.

(2) If the parent Super-Node T of the leaf Super-node X notices its children count is below k, then asks its parent to schedule one child from its brothers as its child. Otherwise if brothers have no extra child, T makes all its children join his brothers. And T breaks from its parent and joins one of its brothers' subtree.

(3) T's parent acts accordingly.

SIMULATION

In this section, we present the experiment results of our system. GI-ITM Generator (Zegura *et al*., 1996) was employed to create a 1000-node transit-sub graph as our underlying network topology. We used the following metrics to evaluate the systems.

(1) Structure change times: the structure breaks and recovers frequently due to the node's dynamic trait. We take the shift of the node's parent as one change.

(2) Delay time: the link delay time.

We studied two scenarios, the first focusing a stable network and the other investigating a dynamic network. For comparison, we also built normal tree structure systems. We set the value of k to 3, hence the children count of both systems is no less than 3 and the Super-Node's size of BulkTree is bounded to [3,9).

Stable network

In this scenario, 1000 nodes joined and after some time left the system sequentially.

Fig.6 shows the structure change times due to the node's joining and departure. The change time is measured as the number of node's parent shift. In BulkTree, if the Super-Node has size still in [k, 3k) after a node's join or departure there is no structure change counted. The Super-Node operates as "node buffer" to quench the influence of a single node. However in Weak-Node Tree, each join/departure makes the tree structure changed and adjusted and the failure of a single node may cause all of its offspring to be lost. Each node's delay time is shown in Fig.7. We notice that the average delay time of BulkTree is smaller than that of Weak-Node Tree. There are two reasons: the first is that the Super-Node which contains several nodes reduces the height of the tree. And on the other hand a receiver node can select several best nodes from parent Super-Node and current Super-Node to broadcast data, which also decreases the delay.

Dynamic network

Peer-to-peer systems are highly dynamic systems

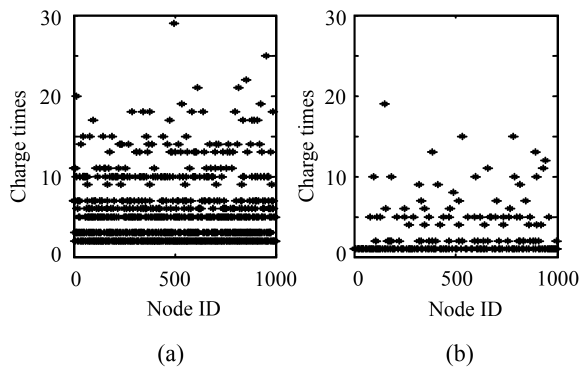

Fig.6 1000 joins and 1000 leaves: struture change time. (a) Weak-Node Tree (*max*=29, *avg*=4.92); (b) BulkTree (*max*=19, *avg*=0.58)

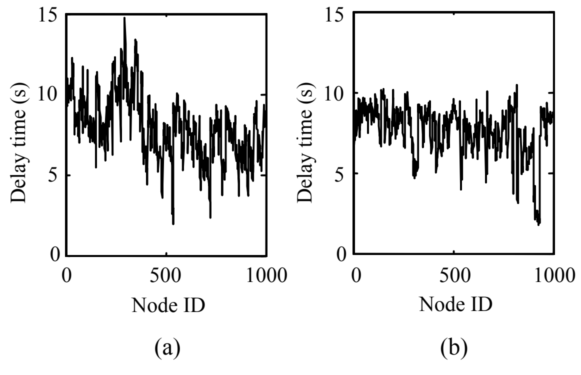

Fig.7 After 1000 Joins: delay time. (a) Weak-Node Tree (*avg*=7987.50); (b) BulkTree (*avg*=7601.86)

which the nodes join and leave arbitrarily. In this scenario, we started with the system consisting of 1000 nodes. Then we let the systems suffer nine fluctuations below: nodes (1, 399) leave, nodes (900, 999) leave, nodes (100, 300) join, nodes (700, 900) leave, nodes (301, 400) and nodes (901, 999) join, nodes (1, 100) and nodes (800, 900) join, nodes (401, 700) leave, nodes (500, 700) join, and nodes (400, 500) with nodes (700, 800) join. For each fluctuation we measured the structures change time and average delay time.

Fig.8 shows the structure change times due to fluctuations. We can see that BulkTree clearly prevails over Weak-Node Tree. As shown in Fig.9, the average delay time of BulkTree is also shorter than that of Weak-Node Tree. Consequently we think our systems can efficiently get the data and are very suited to highly dynamic networks.

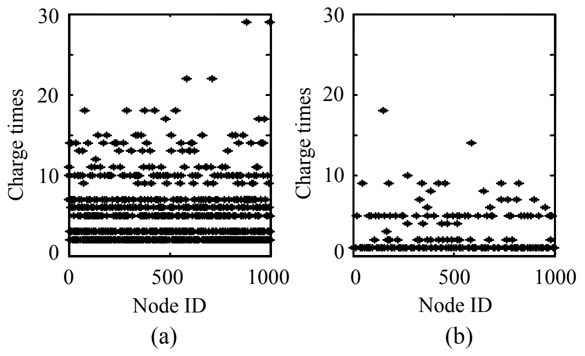

Fig.8 Nine fluctuation: structure change time. (a) Weak-Node Tree (*max*=29, *avg*=4.91); (b) BulkTree (*max*=18, *avg*=0.52)

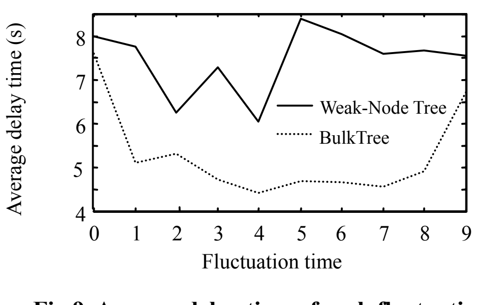

Fig.9 Average delay time of each fluctuation

CONCUSION AND FUTURE WORK

In this paper, we present BulkTree, the overlay network architecture, for live media streaming. BulkTree uses Super-Node as node buffer and employs tree structure aiming to make the structure more robust and efficient and combine the advantages of both structured and unstructured peer-to-peer systems. In later work we will deploy BulkTree in live media application and optimize algorithms.

References

Banerjee, S., Bhattacharjee, B., Kommareddy, C., 2002. Scalable Application Layer Multicast. ACM SIGCOMM. Pittsburgh, PA.

Chen, H., Yang, M., Han, J.Q., Deng, H.Q., Li, X.M., 2004. Maze: a Social Peer-to-Peer Network. Proceedings of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business (CEC-East'04).

Chu, Y.H., Rao, S.G., Zhang, H., 2000. A Case for End System Multicast. Proceedings of ACM SIGMETRICS. Santa Clara, p.1-12.

Eugster, P., Guerraoui, R., Kermarrec, A.M., Massoulie, L., 2004. From epidemics to distributed computing. *IEEE Computer*, **37**(5):60-67.

Ganesh, A.J., Kermarrec, A.M., Massoulie, L., 2003. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans. on Computers*, **52**(2):139-149. [doi:10.1109/TC.2003.1176982]

Padmanabhan, V.N., Wang, H.J., Chou, P.A., Sripanidkulchai, K., 2002. Distributing Streaming Media Content Using Cooperative Networking. ACM/IEEE NOSSDAV. Miami, FL, USA.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S., 2001. A Scalable Content-Addressable Network. Proc. of ACM SIGCOMM'01. San Diego, CA, USA.

Rowstron, A., Druschel, P., 2001. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). Heidelberg, Germany, p.329-350.

Schollmeier, R., 2001. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01). Linköping, Sweden.

Tran, D.A., Hua, K.A., Do, T.T., 2003. ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming. Proceedings of IEEE INFOCOM 2003. San Francisco, CA, USA.

Zegura, E.W., Calvert, K., Bhattacharjee, S., 1996. How to Model an Internetwork. IEEE Infocom. San Francisco, CA.

Zhang, X.Y., Liu, J.C., Li, B., Yum, T.S.P., 2005. CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming. Proceedings of Infocom2005.

Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A., Kubiatowicz, J., 2004. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, **22**(1):41-53. [doi:10.1109/JSAC.2003.818784]