A Data Storage Mechanism for P2P VoD Based on Multi-channel Overlay*

Xiaofei Liao, Hao Wang, Song Wu, and Hai Jin

Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, 430074, China
{xfliao,wusong,hjin}@hust.edu.cn

Abstract. It is a big challenge to provide Video-on-Demand streaming services over Internet in a scalable way. Currently, many researchers use a single channel overlay to implement the scalability of on-demand streaming services. However, in a real application environment, various channels in a P2P VOD system have different popularities, which probably cause the imbalance of data storage-capability of the whole system. It results in a problem that a mass of unpopular channels' caching capability can not be used to satisfy the data requirements of the whole system. In order to solve the problem, this paper proposes a new data-storage mechanism, which constructs a multi-channel overlay to optimize the whole system's caching-capability and greatly improves unpopular channel's caching efficiency. The experimental results show that this mechanism can achieve significant effects.

1 Introduction

When designing a P2P streaming media system, the basic principle is to organize the nodes watching or serving the same program as a single channel overlay, no matter it is tree [1, 2, 3, 4] or mesh topology [5, 6, 7, 8]. Nodes in a single channel overlay store media data to construct P2P network storage. Nodes request and gain media from neighbor peers while they are playing media, in order to reduce the pressure of source server. When the scale of a single channel overlay grows up to a certain size, the P2P network can store most of data to meet all the requirements, minimize direct data requests to the source server.

Based on the analysis of existing P2P Video-on-Demand system, called GridCast [9], the study found that different channels' network scale meet the Zipf distribution. Some popular channels can assemble a large amount of nodes, and P2P network's data storage capability can meet the data request. But most P2P network channels'

* This work was supported in part by China National Natural Science Foundation (NSFC) grants No.60703050, No.60673174, No.60433040, No.60731160630, the Research Fund for the Doctoral Program of Higher Education grants No.20050487040, Wuhan Chengguang Plan with No.200850731350 and Program for New Century Excellent Talents in University under Grant NCET-07-0334.

J. Cao et al. (Eds.): NPC 2008, LNCS 5245, pp. 127–137, 2008.

sizes are small, the single channel overlay only store part of the channel's media, this channel still causes a lot of data sources requests. To effectively improve the load capacity of the entire P2P VoD system, to solve the insufficient data storage capability problem of most unpopular P2P network channels, is particularly important to reduce the pressure on the data source server.

Based on the above analysis, this paper presents a data storage mechanism based on multi-channel overlay, improving unpopular channels' data storage when popular channels' nodes joining unpopular ones. The simulation result proves that this approach can greatly improve the system load capacity.

The rest of paper is organized as follows. Section 2 describes the system architecture. In section 3, how to organize the multi-channel overlay is presented. Section 4 gives the experiments and results. In section 5, related works are described. Section 6 concludes this paper.

2 System Overview

Just like other P2P content distribution systems, GridCast uses a set of source servers to release media files to participating peers, who asynchronously are playing the files while exchanging data among themselves. Unlike file downloading and live streaming, a node is more *selfish* in the sense that it only cares about contents after its current playing position, which is often different from other nodes. Most of the time, a node's downloading targets are those whose playback positions are ahead, and it can only help those that are behind. However, a node can also change its playing position at any time. These characteristics make a VoD system harder to optimize, rendering globally optimal strategies such as *rarest first* employed in BitTorrent [14] inapplicable.

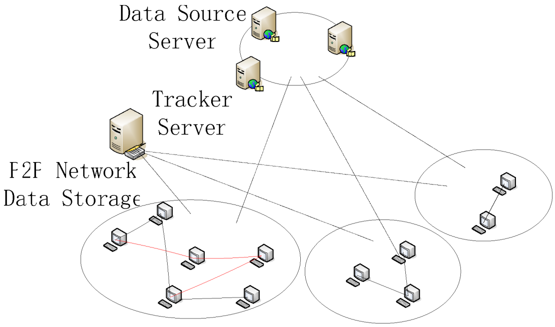

Fig. 1. Architecture overview

To cope with the above problem, the node of GridCast maintains a routing table, which consists of nodes placed in a set of concentric rings with power law distribution distanced using relative playback positions, and uses gossips to keep the routing table up-to-date. This architecture allows a node to find a new group of position-close partners in logarithmic steps after it seeks to a new playing position. The tracker can be considered as a stationary node whose playback position stays fixed at time zero.

The tracker's job is to keep track of its membership view, which bootstraps any new nodes.

3 Multi-channel Overlay

3.1 Network Scale Distribution

In P2P VoD System (Gridcast [9]) based on single channel overlay, each channel's data storage consists of two parts: the P2P network data storage and data server. After one VoD node joins P2P network, it connects with other P2P nodes which are watching the same program to construct a single channel overlay, and exchanges stored data information using Gossip [11] protocol. When the node requests media data, it checks if other nodes in the single channel have stored the data first. If so, it directs request data from the P2P network, and stores data in local cache. Assume that the total media playing time is *T*, each node stores data with length of time *t* in local cache. A single channel overlay at least needs *T/t* nodes to completely stores media data.

The popularities of different programs meet the Zipf distribution [12]. Assuming that the system has *n* program channels, the probability of user options listed *S1, S2,... Sn*, its choice of probability *pi= P{X=Si} (i = 1, 2,... n)*, *{p1, p2,.... pn}* with Zipf distribution, $P_i = \frac{1}{i^{1-\theta}} \Big/ \sum_{j=1}^{n} (\frac{1}{j^{1-\theta}})$ (θ is the Zipf constant). The collected log data of GridCast shows that θ closes to 0.25. During normal system runtime, the system distributed more than 50% of overall nodes in 20% of the most popular channels, but most channels' network scale fail to achieve stable size. In popular channels, because of the large scale P2P networks, program data stored in the P2P networks are more than data needs. Correspondingly, there are fewer nodes in the unpopular channels; the data storage capability is unable to meet the data needs. There still are a lot of data requests to the data source server.

3.2 Data Storage Status Maintenance

GridCast system uses tracker server to maintain the data storage status of whole P2P network. It keeps track on all of the nodes currently joined GridCast system. A node has been represented as an item that holds nodes GUID, address, port, bandwidth, playing time and so on. In order to maintain the information of all nodes, the tracker needs to update the playing position of each node. Each node will send one UDP message to synchronize its buffer status in every minute.

Tracker server uses a hash table to index one channel's storage status. The length of this hash table is the duration of this channel. Every element of this hash table is a double-link list and it maintains information of each node who has stored the corresponding media data. Figure 2 gives a sketch map of the tracker server. When a P2P node requests data status from tracker server, it sends the ID of requested channel and the playing time of the requested data. Tracker server searches the channel's corresponding hash table and acquires the related nodes' information.

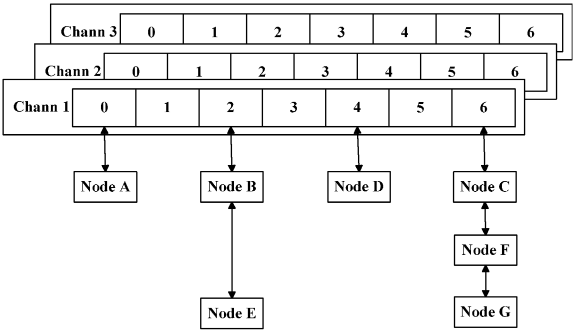

Fig. 2. Data structure of tracker server

P2P node uses a host-list to maintain data storage status of known nodes. The host-list is divided into two levels: neighbors list and nodes list. The neighbors list maintains information of neighbor nodes which are connected with local nodes, and neighbor nodes use directional gossip protocol to exchange information of data storage status. The basic idea of directional gossip protocol is that every node just forwards gossip messages to the neighbors which can retrieve data from source node or send data to source node. Suppose the playing time of one message from some sources is t_{source} and the current playing time of traversed nodes is $t_{forward}$, then we have the following formula: $t_{source}-m \leq t_{forward} \leq t_{source}+m$, here m represents the total time length in caches.

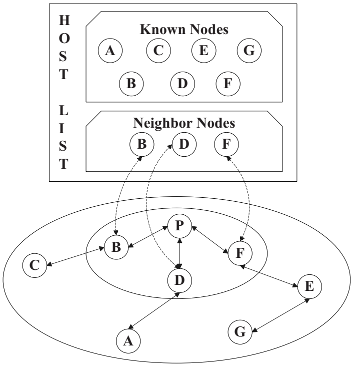

Fig. 3. Host list

3.3 Multi-channel overlay

This paper presents the data storage mechanism from the two areas: 1) constructing a multi-channel overlay network to balance data storage capability between the different kinds of channels; 2) improving utility efficiency of each node's data cache and using idle data storage capacity to raise unpopular channels' data storage capability.

The multi-channel overlay network (Figure 4) is constructed on single channel mesh topology. Compared with the traditional single channel overlay, node can join two kinds of channels in multi-channel overlay: the main channel and the service channel. One node in the main channel plays media, stores data and serves other nodes, and in service channel one node only stores data and serves other nodes. Each node has only one main channel and can choose several service channels. In Figure 4, node *K*'s main channel is channel *A*, and service channels are channel *B* and channel *C*. Node *M*'s main channel is channel *A* and service channel is channel *B*. Node *L*'s main channel is channel *B*, and have no service channel.

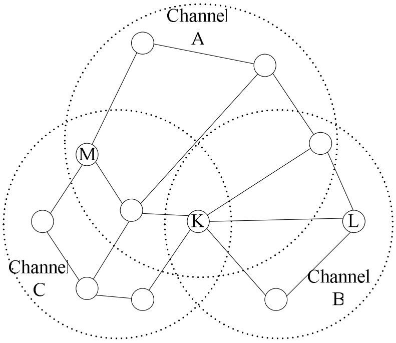

Fig. 4. Multi-channel overlay

Media data will be divided into *L* data blocks according to granularity τ. Assuming that the total program's duration time is *T*, The value of *L* is $\lceil T/\tau \rceil$. Data of playing-time T_i will be classified into $\lceil T_i/\tau \rceil$ data block, and uses a serial number $\lceil T_i/\tau \rceil$ to identify the data blocks. The data storage capability of a node can be described as the ability to store several pieces of data blocks: $N = \lfloor D/(k \times \tau) \rfloor$ (*D* is the node physical storage capability; *k* is the encoding rate of media file).

When nodes join the system, in accordance with its own storage capabilities and channel's network scale. The node will decide the number of data blocks stored of the main channel. If it still has idle data blocks, the newly joined node chooses channels that cause the most data source server request, and joins them as its own service channels. The node provides data storage capacity and stores service channels' program data. Throughout the process of accession, node use caching optimization strategy (see section 3.4) to determine the number of data blocks stored in the main channel and service channels.

After joining into the network, the node maintains its data store and elutriation according to program playing process or data storage state of the whole channel's P2P network. The data storage state of one node in one channel can be described as a state of the attribute set *Sp*=(*Cid, Pc, Np*). *Cid* is the channel's identifier. *Pc* is the initial location of data blocks. *Np* is the number of data blocks stored locally. Nodes in its main channels choose the starting block *Pc*=$\lceil pos/\tau \rceil$. *Pos* is the program playing time. The node in its service channels uses neighborhood nodes' data storage state to maintain its own data in local cache. Nodes exchange data storage state information through gossip protocol. The node can gather data storage information of neighborhood nodes as a set $\pi(P^1, P^2, P^3, \dots P^N)$, calculate amount N_1 of nodes which have store data block Pc and amount N_2 of nodes which have store data block $Pc+Np$. If $N_1>N_2$, it means the data stored of other nodes in P2P network are sliding forward. So the node must slide forward its own data blocks.

3.4 Optimization strategy

In a stable P2P VoD system, after nodes join the P2P network, it can not change the data block number of its own data storage. Otherwise, when node's data block number decreases, it will cause the whole P2P networks' data loss. So each node in the main channel of P2P network should determine its number of data blocks according to the node status and channel's data storage state, and optimize the number of data blocks in the storage of the node to optimize whole channel's data storage capability.

The data storage status of a channel can be described as a set $\Omega(C_1, C_2, C_3, \dots C_L)$. L means total data blocks of the program. C_i (i=1, 2, 3……L) means the number of nodes which store data block i.

When a node joins its main channel, it requests channel's data storage status Ω, and calculates the data blocks number Np that should be stored, according to the data storage status Ω, node's maximum data storage capability N and program's playing time P. The algorithm is as follow:

```
Input: Data storage status $\Omega$, maximum data storage capability $N$, playing time $P$
Output: Data blocks number $Np$
for $i = 0$ to $N$ do
  $CurrentPos = N+P$;
  $SelectPos=i+P$;
  if $CCurrentPos < CSelectPos$
then $Np = i$;
  end if;
end for $i$;
```

After a node joins its main channel, if it finds still has idle data blocks, the node distributes the idle data blocks to unpopular channels, according to the data requests of data source server. Distribution conforms to the following principles: a) priority to store data blocks that requesting more data at data source server to reduce system load; b) to reduce the possibility of the same data storage duplication between nodes.

The requesting status of one data block can be described as attribute set Q=(Cid, Pos, Req), Cid is the channel identifier, Pos is the location of data block, Req is the current data request number at data source server.

The distribution process of node P is as follow:

Step1: Node *P* acquires systems current requesting data blocks set $\Psi(Q^1, Q^2, Q^3, \ldots Q^M)$ at data source server from tracker server;

Step2: Node *P* sorts set Ψ according to the requesting number *Req*, and get a new set Ψ';

Step3: In order to avoid conflict between nodes choice, node *P* sets a selection probability α to choose data block. Sequence checks data blocks in set Ψ', and uses selection probability α to choose whether select or not. When a data block is selected, go to next step;

Step4: Choose channel according to the information of selected data block. Set the selected data block as node *P*'s initialize position of data storage, archive channel's data storage status *S* from tracker server;

Step5: Calculate data block number *Np* using the same method as above;

Step6: Node *P* joins the selected channel, and sets it at node *P*'s service channel.

4 Performance Measurement and Analysis

4.1 Simulation Environment

Simulation programs use GT-ITM [13] topology generator to create a network based on transit-stub model. The network consists of 5 transit domains, each with 20 transit nodes and one transit node connects to 10 stub domains, each with 10 stub nodes. Each stub node offers 35MB physical data storage capability. Set program video's encoding rate to 480kbps. Stub node is able to store 10 minutes of program data. Set system total channel count as 100, with each channel duration 90 minutes. Set granularity size as one minute. We divide program data into 90 data blocks. According to Zipf distribution, set the Zipf constant of the network scale distribution of channels as 0.25. The simulation program test duration is 300 minutes, the start time of nodes in the same channel in accordance with the Poisson distribution. By comparing the simulation based on single channel overlay data storage and multi-channel overlay data storage, we analyze the network scale difference and directly data source requests of the two data storage models, and analyze new multi-channel overlay data storage mechanism performance.

4.2 Network Scale Difference

As showed in Figure 5, in single channel overlay, more than 80% of the total channels' nodes number is below 100. We classify channels of this type as unpopular channels, and classify the opposite 20% channels as popular channels. By building multi-channel overlay network, the network scale of most unpopular channels is upgraded. The node number of smallest channel is 74 nodes. 80% channels' node number of entire system are more than 100, 70% channels' node number of entire system are between 100 and 200. We can see that by building a multi-channel overlay network, unpopular channels' network scale are greatly upgraded, the entire system's nodes distribution is more balanced than single channel overlay.

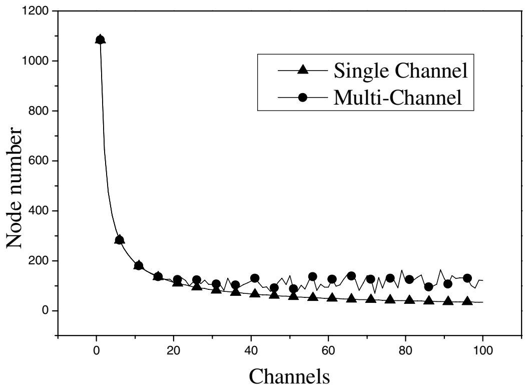

Fig. 5. Node number per channel

4.3 System Performance

By observing the requests number of data source server during the test period, we analyze the performance difference of single channel overlays and multi-channel network. Figure 6 gives each channel's data source server request number during test period. In single channel network, the direct data source server request number of unpopular are more than 600 in general, the average number of data source server request is 890, and the maximum is 1236. The average number of popular channels' data source server request is 442.4. In multi-channel overlay network, the average direct data source server request number of unpopular is 476.1, a decrease of 46.5%, the maximum is 1236, a decrease of 37.5%. The average number of popular channels' data source server request is 403.8, a decrease of 8.7%. We can see that providing some popular channels' idle data storage ability to store unpopular channels' program data can substantially reduce direct requests to data source server. Meanwhile, the reasonably decrease of popular channels' data storage capability does not cause the increase of direct data source server request, but cause the decrease of direct data source server request by decreasing duplicate data request to source server.

Figure 7 gives source server request per minute during test period. We can observe that in the initial period, compared with single channel overlay networks. The source server request pressure is increasing because the number of nodes joining unpopular channels in multi-channel overlay network is larger than that of single channel overlay. After the multi-channel overlay network is built stably, the unpopular channels in multi-channel overlay have more nodes than that of single channel overlay. The data storage capability is improved, and direct data source server request is substantially reduced. From 0 minute to 50 minute, data request per minute to data source server in single-channel P2P network is 206.5, and in multi-channel overlay network is 212.8, request number in multi-channel overlay increases 3.1% than that in single channel overlay. From 51 minute to 300 minute, data request per minute to data source server in single-channel P2P network is 278.3, and in multi-channel overlay network is 174.3, request number in multi-channel overlay decreases 37.3% than that in single channel overlay. In the whole test period, data request per minute to data source server in single-channel P2P network is 266.3, and in multi-channel overlay network is 180.9, request number in multi-channel overlay decreases 32.1% than that in single channel overlay. Although in the early stage of building a multi-channel overlay network will increase data source server requests to a certain extent, but if the entire system's P2P overlay is built stably, multi-channel overlay network can substantially reduce data request to data source server, and impressive upgrade the entire system performance.

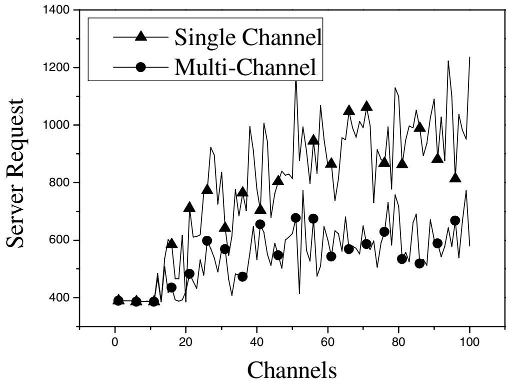

Fig. 6. Server request per channel

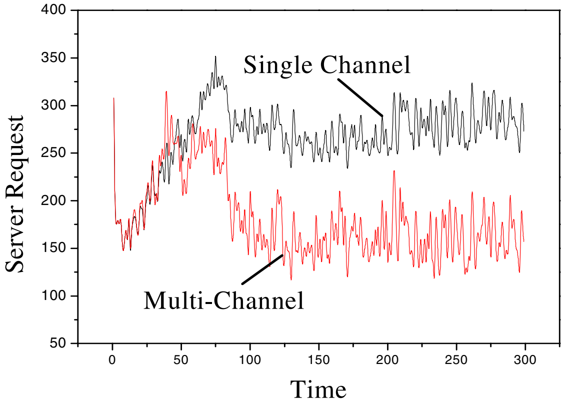

Fig. 7. Server request per minute

5 Related Work

In order to provide a large-scale on-demand streaming service over Internet, several techniques have been proposed to increase the scalability of on-demand streaming by adopting peer-to-peer methods. However, most of them try to use a tree-based overlay to build their logical topology, such as P2Cast [1], P2VoD [2], DirectStream [3], MetaStream [4]. Compared with the traditional methods, i.e. CDN, proxy, and patching, they achieve better scalability. But for these systems, the greatest challenge is to accommodate the dynamic change and to mask the impact of node joining or leaving frequently. Their drawbacks include the following respects. On the one hand, tree maintenance is always very complicated in order to avoid the impact because of the silent departure of several key parent nodes. On the other hand, each peer depends on only one data supplier. This will lead to inefficient resource utilization and increase the load of the central source server. There are other kinds of streaming systems based on unstructured overlay, such as SplitStream [6], CoolStreaming [7], GNUStream [10], PROMISE [8]. However, all of them focus on the streaming overlay constructions for single channel, not for multiple channels.

6 Conclusions

In order to solve the data storage unbalance problem among channels in P2P VoD system, this paper presents a data storage mechanism based on multi-channel overlay. In the proposed overlay, nodes of popular channels will join unpopular channels' P2P overlays to construct a multi-channel overlay, and use data storage optimization strategy to improve node's data store ability. This mechanism can effective balance node distribution among different channels, and improve the storage capability of the entire system. The experiment results prove the idea.

References

[1] Guo, Y., Suh, K., Kurose, J.: P2Cast: Peer-to-peer Patching Scheme for VoD Service. In: Proceedings of the 12th World Wide Web Conference (WWW 2003), Budapest, Hungary (May 2003)

[2] Do, T., Hua, K.A., Tantaoui, M.: P2VoD: providing fault tolerant video-on-demand streaming in peer-to-peer environment. In: Proceedings of IEEE ICC 2004, Paris, France (June 2004)

[3] Guo, Y., Suh, K., Kurose, J., Towsley, D.: A Peer-to-Peer On-demand Streaming Service and Its Performance Evaluation. In: Proceedings of 2003 IEEE International Conference on Multimedia & Expo (ICME 2003), Baltimore, MD (July 2003)

[4] Zhang, R.M., Butt, A.R., Hu, Y.C.: Topology-Aware Peer-to-Peer On-demand streaming. In: Proceedings of 2005 IFIP Networking Conference (Networking 2005), Waterloo, Canada (May 2005)

[5] Chu, Y.-H., Rao, S.G., Zhang, H.: A case for end system multicast. In: Proceedings of SIGMETRICS 2000, Santa Clara, CA, USA (June 2000)

[6] Castro, M., Druschel, P., Kermarrec, A.M., Nandi, A., Rowstron, A., Singh, A.: SplitStream: High-Bandwidth Content Distribution in Cooperative Environments. In: Proceedings of ACM SOSP 2003 (October 2003)

[7] Zhang, X., Liu, J., Li, B., Yum, T.-S.P.: Data-Driven Overlay Streaming: Design, Implementation, and Experience. In: Proceedings of IEEE INFOCOM 2005, Miami, USA (2005)

[8] Heffeeda, M., Habib, A., Botev, B., Xu, D., Bhargava, B.: PROMISE: peer-to-peer media streaming using CollectCast. In: Proceedings of ACM Multimedia (MM 2003), Berkeley, CA (November 2003)

[9] Cheng, B., Stein, L., Jin, H., Zhang, Z.: GridCast: Providing Peer-to-Peer On-Demand Streaming Service Based On Unstructured Overlay. In: Proceedings of Eurosys. 2008 (2008)

[10] Jiang, X., Dong, Y., Xu, D., Bhargava, B.: GnuStream: a P2P Media Streaming System Prototype. In: Proceedings of International Conferences on Multimedia & Expo (ICME 2003), Maryland, USA (2003)

[11] Sun, Q., Sturman, D.: A Gossip-Based Reliable Multicast for Large-Scale High-Throughput Applications. In: Proceedings of the Int'l Conf Dependable Systems and Networks (DSN 2000) (2000)

[12] Hou, M., Lu, X., Zhou, X., Zhang, C.: Study on Replication in Unstructured P2P System. MINI-MICRO SYSTEMS 26(11), 1903–1906 (2005)

[13] Zegura, E., Calvert, K., Bhattachajee, S.: How to model an Internetwork. In: Proceedings of INFOCOM 1996 (1996)

[14] Guo, L., Chen, S., Xiao, Z., Tan, E., Ding, X., Zhang, X.: Measurements, Analysis, and Modeling of Bit Torrent-like Systems. In: Proceedings of ACM IMC 2005, Berkeley, CA, USA (October 2005)