P2P 환경하에서 VOD 시스템을 위한 시간 기반 캐싱 기법

계이기, 최황규*
강원대학교 컴퓨터정보통신공학과
e-mail: gigi_1368@hotmail, hkchoi@kangwon.ac.kr

Time Based Caching Scheme for Video on Demand in P2P Environment

Yi-Qi Gui, Hwang-Kyu Choi*
Dept. of Computer Science and Engineering, Kangwon National University

요 약

본 논문은 P2P 네트워크 환경에서 대규모 VOD 시스템을 위한 새로운 시간 기반 캐싱 기법을 제안한다. 제안된 캐싱 기법은 각 피어들이 요구 시작 시간을 기준으로 비디오의 서로 다른 부분을 분산 저장 관리함으로써 피어의 저장 용량을 최소화하고, 피어간 데이터 요구에 대한 캐시 적중률을 향상시켜 서버의 부하를 최소화 한다. 또한 요구 시간을 기준으로 인접한 피어들을 하나의 클러스터로 그룹핑하여 피어의 탐색 범위를 최소화하고, 이에 따른 네트워크 트래픽을 최소화 한다. 시뮬레이션을 통한 성능 평가에서 피어의 참여와 탈퇴 증가에 따른 서버 부하의 증가가 기존의 P2VOD와 비교하여 크게 감소함을 보인다.

1. Introduction

As with the large adoption of high speed Internet, video on demand (VoD) is increasingly much more popular on the Internet, which gives Internet users greater choice and more control than live streaming or file downloading. Streaming a video to remote peer takes a significant amount of communication bandwidth, which is much more than traditional text based messaging. In the traditional client/server solutions, every demand is handled by a centralized server, requiring a powerful server and large bandwidth. Due to the server or network I/O bottleneck, the traditional solutions can only serve very limited number of concurrent demands.

Peer to peer (P2P) solves the bottleneck on the server under the centralized client/server architecture. The advantage of P2P is to share resources between peers and to utilize all the available resources on the Internet, where peers benefit from one other. Another advantage of P2P is low cost. P2P is an application-layer solution, which does not need upgrade to an existing network. It utilizes there sources of peers, which greatly reduces there requirements on the capability of a server. Recently P2P technologies[1,3] are being used for file sharing and application-level multicast (ALM) and more. For video distribution over today's Internet, where the deployment of IP multicast has been slow and especially the receiving ends are in vastly different network domain, while P2P video sharing is to allow hosts to share their videos directly. In a P2P video system, a host can be served by any other host that has the video it requests. Later this host can supply the video data it caches, if any, to serve future requests. This service model is different from ALM in taking advantage of peer computing resources. In ALM, a peer forwards an on-going video stream to serve other peers. Besides the high bandwidth requirement, the peer can only contribute during the time when it is downloading a video itself. After playing back a video, the client does not help further in distributing this video.

In contrast, P2P video services amplify the serving capacity of a video server by caching its videos on its peers. When a peer downloads a video from a server, the peer can cache the video and serve the whole community, just like the original server of this video. Thus, a peer does not have to forward its incoming video stream, while downloading it, in order to contribute in video services. The strength of a P2P video system relies on the effective aggregation of communication bandwidth and disk space contributed by its participating hosts. Ideally, after a host downloads and plays back a video, it caches the whole video and becomes a supplier of this video. In reality, however, very few hosts are willing to retain a complete video and supply it back to the community. This is not just because a video is usually very large in size, but also because serving a video request takes a significant amount of communication bandwidth, which seems to be the major concern of most users. Apparently, a P2P video system cannot simply rely on the few hosts that cache videos in their whole to serve all video requests. Otherwise, it will create the server bottleneck problem like in a central server architecture. Although proxy caching solution[2,5] can deal with the server side bottleneck, it is expensive and not very scalable. To materialize the advantage of P2P computing, a host should be allowed to participate in video services as long as it caches some amount of video data, instead of a whole video.

* 교신저자

In this paper, we proposed a unstructured P2P VoD streaming system to present a fully distributed video management using a new time based caching scheme (TBC). Each client in our system have a variable size buffer to cache a different part of the video in its local buffer to serve other clients who request the same video. The key idea of our technique is the concept of cluster, which is defined to be a group of hosts which together can supply a complete video. Each cluster is created dynamically and managed individually. The advantages of our technique are twofold. First, a client requesting a video can locate a complete set of video pieces from its nearest host that caches some part of the video. Thus, the search scope is dramatically reduced. Second, caching video data can be coordinated at the cluster level causes very minimal communication overhead because our scheme limits cluster's size, ie the number of its member peers.

The remainder of this paper is organized as follows. We present the architecture and algorithms of TBC system in Section 2. In Section 3, we present our simulation study. Finally, we give our concluding remarks in Section 4.

2. Architecture of TBC

2.1 Preliminary

In our system, video files are segmented on time rather than space. In a traditional file downloading system such as Bit-Torrent [4], files are segmented on space. Systems like Bit-Torrent do not provide any coherent way for users to interact with files during downloading. As long as downloading takes some noticeable amount of time, a VoD system must overlap user interaction and downloading. User seeks are based on time. Videos are partitioned into chunks of uniform time to make the file addressable on time. Each client has a buffer, whose maximum size is worth of M_{bf} to cache the content data to server other clients.

The unit amount of the buffer storage at one time used by a client X is denoted as U_{bf}. By caching in the same cluster, each clients caches exclusively different portion of the video stream. Any client receives the stream from other clients as long as the parts of the stream in available in the client cluster. The missing part of video streaming can be received from other clusters or Server. If the join time of a client X is X_{jt}, then X can storage the unit size from X_{jt} to $[X_{jt}+U_{bf}]$ at the first caching time. Clients are group into a cluster when $|I_{jt}\text{-}(I-1)_{jt}|<=U_{bf}$. Clusters are also numbered, starting from Clu1 as the oldest cluster to Clu(N) as the youngest generation. To keep minimal communication overhead in the cluster, the maximum number of clients can be joined the cluster is denoted as GS. A cluster can be closed, when the number of clients is increasing to GS or long time gap (Threshold) no client joins the system.

Figure 1 captures cluster structure of TBC at time t_{33}. Assuming that each request arrives at time $t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, t_{31}, t_{32}, t_{33}$, respectively, 3 clusters were be created, such as Clu1, Clu2 and Clu3 at time t_{11}, t_{21}, t_{31}. Then, $p_{12}, p_{13}, p_{22}, p_{32}, p_{33}$ are participated into their group as a member of the groups. Clu1 and Clu2 are Close state and Clu3 is Open state. And Clu3 is youngest cluster in the system now. p_{13},and p_{22} are tail of the Clusters. It can join into The Clu3 if a new client wants to join the system.

2.2 Data caching and Cluster

we formally introduce the cluster concept and our novel caching algorithm in this section. In TBC, a cluster is defined as a group of the clients who join the system in closing time as $|I_{jt}\text{-}(I-1)_{jt}|<=U_{bf}|$. For example , U_{bf} =2min, Client X join the system at time 00:12:30 (hh/mm/ss), and client Y join the system in 00:13:50 , so X and Y join the same Cluster, while the number of the Cluster is small to GS.

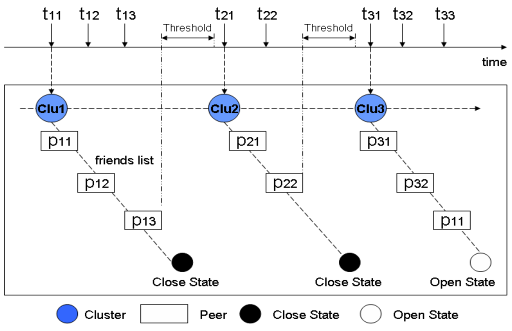

Figure 1. clusters structure of TBC at time t_{33}

```
Caching Schedule:
void Cache_Scheduler()
// update the list of cluster member
    if  the cluster Clu(N) is open // the youngest cluster Clu(N)
      request Client I to catch video data in [$I_{jt}$, $I_{jt}+U_{bf}$]
        If (CN==GS)// check the number of client in the cluster
          set cluster Clu(N) be closed and
          gossip message with all the clients to check
          Cache_scheduling in Clu(N).
          $\triangle g=I_{jt}+U_{bf}$
        end
      end
    else // the cluste Clu(N) is closed
        while(client I  is Caching the data <=$I_{jt}+U_{bf}+\triangle g*K$)
          // the last client I join in the cluslter
           for each clients in the Clu(N) client I: 1 to CN
            // client I in the Kth times cache streaming
              request client I cache video streaming data in
               [$I_{jt}+\triangle g*K$, $I_{jt}+U_{bf}+\triangle g*K$]
           end
          K++
        end
    end
  end
```

Figure 2 illustrates how peers C_1, C_2, C_3, C_4 and C_5 of cluster Clu(N-1) apply the caching algorithm. As an example client C_5at time t_5 , due to the caching schedule, C_5 can cache a unit amount of buffer size U_{bf} video streaming data in [hash(t_5), U_{bf}+hash(t_5)]. C_6 join into the system at t_6, he/she cannot join into Clu(N-1) because of | t_6-t_5 | $>U_{bf}$. Hence, the cluster Clu(N-1) will be closed at t_5+U_{bf}. A new cluster Clu(N) and a new video session are created at t_6, and C_6 is the first member of the cluster. For the remaining of the paper, Two peers are called friends if they are belong to the same cluster.

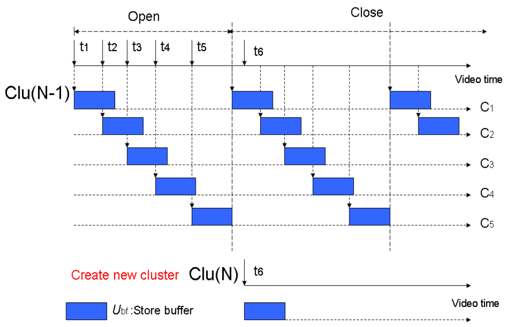

Figure 2. caching strategy for peers in the same cluster.

2.3 Peer Management

This section describes the how TBC works for peers sharing in terms of peer management and chunk fetch. Our TBC system is comprised of Web entry, a track server(tracker), one or more source server(sources), and peers. Figure 3 illustrates these components and their interactions. In arrow1, the user on peer C contacts the web portal to browse the catalog and select a video file. The portal returns a video file ID to the peer, with arrow 2. To form connections with others for sharing, a peer contacts the tracker, sending the tracker a description of its state. The tracker uses this information to construct a list of candidate peers, returned in arrow 4. Chunks are fetched from peers or sources. A fetch from source server β is shown with arrows 7 and 8 and a fetch from peer D with arrows 5 and 6. And A joined the Clu(N) with its friends (D ,E, F) and starts its caching schedule(details in the section2.2).

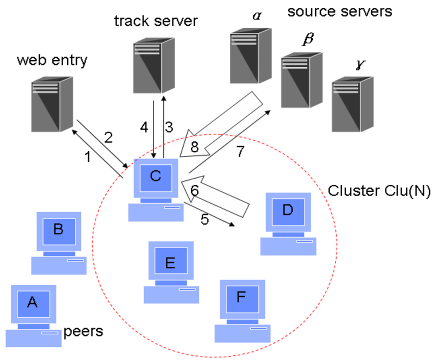

Figure 3. basic architecture of our system.

Each peer organizes peers it knows from the tracker or gossip messages into four lists: members, neighbors and partners for overlay network and friends for caching scheme in the same cluster. All the partners are neighbors and all neighbors are members. And all the friends are members. A members is a peer with a known IP address, its join time(JT) and cache map(*cm*), cluster number. The member list is the most inclusive. It is updated when the peer synchronizes with the tracker or receives gossip messages from other peers. A friend is a member in a cluster Clu(N) for caching chunks as its caching schedule for sharing them with peers. For the friends, they are request to periodically exchange control message with their friends in order to keep the friends list up-to-date. we can choose to update the list on-demand. That is the update is initiated either by a new node joining the generation or by a node leaving the system intentionally. A neighbor is a member that has been promoted based on *cm* proximity. Neighbors persistent TCP connections. A partner is a neighbor that has been promoted based on play-cache proximity. Chunks are only shared between partners. Partners share data and neighbors share meta data.

To limit the overhead of peer management, each peer constrains the number of its members, neighbors, and partners and friends. User operations can quickly change the potential for sharing between peers. The local scheduler needs fresh information to find chunks. To keep minimal the overhead of finding chunks, we take account into the cluster number as the first priority when partners are selected from neighbors. For example, peer A and Peer F are the Neighbors of Peer H, and Peer F and Peer H in the same cluster Clu(N), but Peer in the Clu(N-1). So Peer H chooses Peer F firstly. To keep the partner list relevant for chunk sharing, every 30s the peer recalculates the content proximities of its members, neighbors and partners, then promotes or demotes based on this calculation. JT and *cm* proximity promotes members to neighbors and neighbors to partners.

2.4 New client admission

The tracker has the list of peers of the youngest cluster as Clu(N) for each video session. To keep the first sorts of chunks(Ex:0~2min) can be cached in peer X in this cluster till the last peer join into the Clu(N) and finish to cache its part of video chunks, we assume that keep the maximum number of the cluster as GS and keep each peer can storage the chunk sizes are limited (<=2min) at one time. peers in the same cluster can share the chunks only if they are overlapped. Hence, peers can be into the same cluster if their join time are so closed(as I_{jt}-$(I-1)_{jt}$|<=U_{bf}).

case1: if the youngest cluster closed, a new client X will get information from the tracker and a new video session is created, and X is the first number of the youngest cluster Clu(N). At the same time, X updates the friends list and gossip with other peers and the tracker. Then X contracts the Clu(N-1)(or other clusters and source server) to fetch chunks and start its caching schedule, meanwhile finds neighbors to and make partners to structure overlay network.

case2: For the case where there is the youngest cluster Clu(N) is open, a new Client X can get the information from the tracker and join in the Clu(N). update the friends list and gossip with other peers and the tracker. Then X contracts the Clu(N) to find neighbors to fetch chunks and start its caching schedule, meanwhile finds neighbors and selects partners to structure overlay network.

3. Performance Evaluation

We study the performance of our system using GT-ITM[6] topology generator to greate the underlying network topology of 1000 peer nodes based on the transit-stub model. The network consists of 3 transit domains, each with 5 transit nodes and a transit node is connected to 6 stub domains, each with 12 stub nodes. In this set of experiments, peers can be located on any stub nodes in the topology. We randomly choose 1000 stub nodes as peer clients and place the source server that stores all of media contents on a transit node. The bandwidth settings between two transit nodes, a transit node and a stub node are 100Mbps and 10Mbps, respectively, and the out-bound bandwidths of stub nodes are heterogeneous. In addition, we choose a movie with 60 KB/s streaming rate and 2 hours content as our testing stream. Some other important parameters are given in Table I. Both TBC and P2VoD cache recently watched media data in local buffer to relay to other peers, so they are similar with respect to buffer management. We compare the performance of TBC and P2VoD in terms of server stress and quality of streaming. In our experiments, P2VoD uses Smallest Delay Selection as its parent selection scheme. Additionally, the maximum number of clients allowed in the first generation of each session is 8 and the buffer window size is 600 seconds.

TABLE I. Parameter List

Parameter	Value and Description
M_{bf}	600 seconds, buffer window size
TTL	5, maximum hop number for gossip message
t	50 seconds, gossip period
g	50, number of member
h	30, number of near neighbor
n	10, number of partners
w	20. number of friends
T_m	2min, checking period of partners in partner list
T_p	8min, checking period of members in member list

1) Server Stress

Figure 4 shows the source server stress caused by TBC and P2VoD with different numbers of nodes. The arrival rate of client is 1 per second. Note that the server stress of TBC remains at 6~7 streams when the number of nodes increases. In contrast, for P2VoD, the server stress increases almost linearly (from 8 to 23). This is mainly because that in P2VoD, each peer receives data from only one parent; and consequently, a peer's residual bandwidth will be wasted if it cannot support one more child. When a newly arriving client fails to find a peer capable of supporting a full stream, it has to create a new session from the source server, even though there may exist some peers whose aggregate bandwidth is greater than that of a full stream.

2) Quality of Streaming

We compare the reliability of TBC and P2VoD by "shutting down" some peers. Initially we start 1000 peers with an arrival rate of 2. After all peers have started and played for a while, we randomly stop some joined peers at a speed of 2 peers per minute, and then calculate the average times lot missing rate (TMR) for the remained peers. TMR is measured by the number of missed times lots divided by the total number of timeslots. We repeat this experiment ten times. Figure 5 presents the result of times lot missing rate with different percentages of node failure. We can see that TBC has a lower TMR than P2VoD for the same percentage of node failure. That means TBC achieves better reliability by using gossip protocol and retrieving data packets from multiple partners.

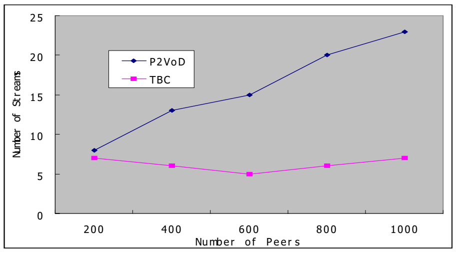

Figure 4 Server Stress (TBC vs. P2VoD)

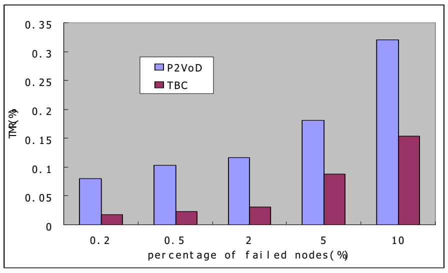

Figure 5. Quality of streaming (TBC vs. P2VoD).

4. Conclusions

In this paper, we have presented a new time_based caching scheme for VoD streaming in our unstructured P2P overlay network, called TBC. The key ideas in this paper are introduction of cluster, by which a peer can caching any part of video data dynamically, and our caching scheme. Since a client can always contact its nearest caching for a complete Video, the search scope of a video look up is minimized. Since the cluster size is small, it incurs little communication and computation overhead. The simulation results show that our system is superior to previous scheme in terms of server stress, and quality of streaming .

References

[1] D. Tran, K. Hua, and T. Do, “"Zigzag: An efficient peer-to-peer scheme for media streaming”", In *Proc. of IEEE INFOCOM’'03*, San Francisco, CA, April 2003.

[2] B. Wang *et al.*, Optimal Proxy Cache Allocation for Efficient Streaming Media Distribution, IEEE Infocom 2002,New York, USA.

[3] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H.Zhang, “"The feasibility of supporting large-scale live streaming applications with dynamic application endpoints”",In *Proc. of ACM SIGCOMM’'04*, Portland, USA, Aug. 2004.

[4] B. Cohen. Incentives Build Robustness in BitTorrent. In*P2PEcon*, June 2003.

[5] D.A. Tran, K.A. Hua, and S. Sheu, A New Caching Architecture for Efficient Video Services on the Internet, IEEE The 2003 International Symposium on Applications and the Internet, Orlando, Florida, 2003.

[6] E. Zegura, K. Calvert, and S. Bhattacharjee, “How to model an internetwork,” In Proceedings of IEEE INFOCOMM’'96, Mar. 1996