Flash Crowd Handling in P2P Live Video Streaming Systems

Anurag Dwivedi, Sateesh Awasthi, Ashutosh Singh, Y. N. Singh
Electrical Engineering, IIT Kanpur

***Abstract* – An interesting and challenging phenomenon that has been observed in P2P live video streaming systems is flash crowd – the arrival of hundreds of thousands of peers in a short time span. Experiments have shown that the system can scale only up to a limit during flash crowd. This is limited both by the available surplus bandwidth as well as the intense competition among the peers for scarce resources. Various population control measures have been suggested for both mesh-based and tree-based systems. The key idea is to organize the peers and reduce competition among them. The main focus area of this paper is on tree-based systems. Such systems have seen some algorithms with centralized control. In this paper, we propose a distributed algorithm to handle flash crowd in tree-based P2P live video streaming systems by organizing the peers into hierarchical positions and only the peers at the top of the hierarchy directly access the available resources.**

1. INTRODUCTION

P2P systems have greatly enhanced live streaming experience by creating efficient and highly scalable streaming overlays where bandwidth capabilities of all peers are utilized. Measurement studies on existing systems [3] [4] have shown that the system performance can be maintained at high level once a sufficient system scale has been achieved. However, its realization is greatly challenged by the phenomenon of flash crowd – the arrival of hundreds of thousands of peers in a very short span of time. Such situations may typically arise at the beginning of live streaming events. The newly arriving peers compete for the limited system resources; drastically reducing the quality of service. Flash crowd impacts live streaming systems even more because due to the stringent time constraints, many impatient peers may leave the system unsatisfied.

The most intuitive way to handle flash crowd is by deployment of additional resources – servers in CoolStreaming+ [2] or content delivery network in SkyNet [5]. However, this method is not cost effective. Moreover, when the flash crowd subsides, the system can be maintained by only a few resources, making the additional resources useless.

In recent years, more rigorous analysis of the dynamics of flash crowd has been carried for both mesh-based [7] [8] and tree-based systems [9] [10]. Our main focus is on tree-based systems. Some proposals [9] [10] have been suggested to handle flash crowd in tree-based live streaming systems, but they adopt a centralized algorithm. Such solutions suffer from scalability and fault tolerance issues. In this paper, we propose a distributed algorithm that can arrange the newly arrived peers in streaming trees without requiring any centralized control.

The rest of the paper is organized as follows. Section 2 presents the existing research done on handling flash crowds. Section 3 covers the preliminaries whereas the proposed method is discussed in Section 4. Section 5 contains the experimental results. Finally, Section 6 is conclusion.

2. RELATED WORKS

Liu et al. [6] [7] were the first to quantify the dynamics of flash crowd and propose a fundamental time-scale relationship in mesh-based live streaming systems. They showed that with stringent time constraints as in the case of live streaming systems, the network can scale only up to a limit during flash crowd. This is because only that many peers can be satisfied in the first attempt as is the surplus bandwidth capacity of the system. This fraction of satisfied peers is furthermore reduced in systems that rely on gossip protocols to exchange data chunks because of the intense competition among the peers for limited resources and incomplete knowledge of the system [6]. Hence, collaboration instead of competition among the peers can lead to better system scale.

Based on the above principle, admission control measures have been developed. For mesh-based systems, Liu et al. [7] suggested a simple population control framework that trades initial peer start-up delay to achieve better system scale. Chen et al. [8] have devised population control measures where the peer waiting time increases logarithmically with system scale.

Flash crowd handling in tree-based systems have attracted considerable less attention. Chung et al. [9] constructs a tree completely out of newly arrived peers and then connects it to the existing system. Wu et al. [10] takes this scheme further and proposes to construct not one but multiple trees based on the available surplus bandwidth of the system. The underlying principle in these approaches is to isolate the newly arrived peers and arrange them in a topology before connecting them to the existing network.

The first step involves detection of flash crowd by the tracking server and calculation of number of trees to be constructed. Then, the merit rank of each peer is calculated and positioned in different layers of the tree(s) accordingly. Finally, the peers are connected to peers in a layer above it and the root node of the tree(s) is connected to existing peer(s). An essential role is played by the tracking server as it is responsible for performing all of the above responsibilities. This burdens the tracking server and makes it a potential single point of failure.

We propose a new methodology that has two vital aspects – (1) A distributed algorithm to construct the streaming tree freeing the bootstrapping server of excessive load. (2) Our design doesn't need any special flash crowd detection mechanism. Even in the absence of flash crowd, it can keep on working without necessitating any change. Thus, we propose a distributed, scalable and simplistic solution to handle flash crowds in tree-based P2P live streaming systems.

3. PRELIMINARIES

We use Distributed Hash Table (DHT) based protocol to provide the overlay over which streaming tree can be constructed. Chord [11] is our preferred choice but other DHT based protocols such as Tapestry [16] or Pastry [17] can also be used.

3.1 Chord

Chord is a scalable P2P look up protocol designed to construct an overlay network based on DHT. The node-id and resource-id space in Chord coincides and each node maintains the resources lying in its region of id space. By constructing and maintaining efficient routing structures, a resource can be located definitely in a maximum of $log_2(N)$ hops. On bootstrapping, each peer contacts the bootstrapping server and receives a list of existing peers. Then it connects itself in the chord overlay and populates its routing tables by exchanging information with these peers. The bootstrapping server keeps tracks of number of requests received. Chord supports multiple joins and can bootstrap large-scale DHT networks [12].

3.2 System Model

The network consists of homogenous nodes with same uplink bandwidth, all connected in the chord overlay. To distribute the video-stream, all nodes are connected in a tree topology and a list of feed-forwarders is maintained for all the levels of the tree in a distributed manner. Each list is identified by a unique key of the form *genX_levelY*, where *X* and *Y* represent the generation and level number respectively. More about generations can be found in Section 4.4. The node maintaining this list can be reached by hashing the key and routing it over the chord overlay. This node is responsible for registering peers in the list and replying with latest feed-forwarder list on requests. A new arriving peer first connects itself in the Chord overlay and then runs our distributed algorithm to join the streaming tree. We adopt the terminology used in [8] to denote the peers who are connected and able to forward streams as stable peers and the new arriving peers as start-up peers.

4. PROPOSED METHOD

4.1 Measurement of Flash Crowd

A quantity called *numLevel* is used quantify the intensity of flash crowd. It denotes the number of level that needs to be maintained to accommodate the start-up peers. The value of *numLevel* will depend on the size of flash crowd *M*, feed-forwarding capacity of individual peer *k* and the surplus feed-forwarding capacity of the system *U* and is given by $\lceil \log_k(1+M(k-1)/U) \rceil$. *U* peers would be accommodated in first level, *kU* peers in second and so on.

The value of *numLevel* is maintained and updated by the bootstrapping server. The value is also cached at local nodes for fast retrieval by start-up peers. Whenever, *numLevel* changes or at regular intervals, controlled update propagation [13] is used to push the latest value to the interested peers in the network.

4.2 Construction of levels

A start-up peer on joining the network will enquire about the value of *numLevel*. Now, each peer will run the algorithm given below on their local machines to select a level and then register itself as a future feed forwarder at that level. Levels are selected such that the probability of selecting a level is equal to the fraction of peers that can be accommodated in that level. Let P_i be the probability of selecting level *i*. Then,

$$P_i = \frac{\text{nodes that can accomodate in level } i}{\text{total number of arriving nodes}} \qquad (1)$$

If individual feed-forwarding capacity is *k* and number of levels maintained is *numLevel*, we get

$$P_i = \frac{(k^{i-1})(k-1)}{k^{numLevel}-1} \qquad (2)$$

Due to the randomized nature of level selection, the levels may not be proportionately populated. Hence level balancing schemes needs to be adopted. Section 5.2 discusses and evaluates some such schemes.

After successfully registering at a level, start-up peer requests the list of feed-forwarders registered at a level above it and selects a node randomly from the list to connect to.

4.3 Global feed-forwarder list

The global feed-forwarder list contains server nodes as well as all those stable peers who have unused feed-forwarding capacity. These peers comprise the surplus feed-forwarding capacity of the system *U* and help the nodes in the highest level (level 1) of the tree to get feed.

```
Level Selection Algorithm
// the algorithm will run independently by each peer
for ( i = 1 to numLevel ) {
    calculate $P_i$ according to (2)
}
choose a random number $x \sim U(0,1)$
myLevel = 1  // represents level selected by peer
commProb = $P_1$  // represents sum of probabilities
while (1) {
    if ( $x \leq commProb$ ) {
        send request to level myLevel
        break
    } else {
        myLevel = myLevel + 1
        commProb = commProb + $P_{level}$
    }
}
```

4.4 Generations

After the flash crowd subsides, only the lower level will have nodes with unused feed-forwarding capacity. So if a new batch of peers arrives, some of them may unsuccessfully try to register at higher levels and incur delay. Hence, the start-up peers are compartmentalized into generations. The peers belonging to the same generation would interact among themselves to create the overlay. Once a generation is over, the peers with unused feed-forwarding capacity would register themselves in the global feed-forwarder list and the process would start again with afresh calculation of *numLevel*.

5. EXPERIMENTAL RESULTS

5.1 Simulation Settings and Parameters

PeerSim [14] [15], a java based event driven simulation engine, is used to run the simulation. The preferred protocol for providing DHT overlay is Chord.

The following parameters are used in simulation. Flash crowd scale *M*, system surplus capacity *U* and peer feed-forwarding capacity *k*.

5.2 Level Balancing scheme

As the peers select the levels independently and with certain probabilities, some levels may receive more or less registration requests than their capacity. This would create an imbalance. Hence, level balancing schemes are required. We propose and compare 2 different level balancing schemes. In the first scheme, if a peer is unable to register at a particular level *Y*, it tries to register at level *Y+1* and then *numLevel*. In scheme *S2*, it tries at *Y-1* and then *numLevel*. If the peer is unsuccessful even after this, the registration would be counted as failed. Scheme *S0* refers to no level balancing scheme.

We compare the schemes across 3 metrics:

(i). System Imbalance *I*

$$I = 100\left[1 - \frac{\sum_{i=1}^{numLevel-1} (\text{Peers Registered in level } i)}{\sum_{i=1}^{numLevel-1} (\text{Capacity of level } i)}\right]$$

(ii). Extra Message Cost Per Peer *C*

$$C = \frac{\text{Messages generated for balancing levels}}{\text{Total number of peers registered}}$$

(iii). Percentage of failed registrations *F*

Three different cases have been considered for peer arrival pattern – (1) Extreme Flash Crowd: All 4090 peers arrive simultaneously (2) Uniform: Peers arrive with rate $\lambda=2000/s$ for a period of 2 seconds. (3) Poisson: Peers arrive with rate $\lambda=2000/s$ for a period of 2 seconds. The parameters for simulation are $M=3800$, $U=2$ and $k=2$. As we can see from Table 1, 2 and 3, adopting a level balancing scheme significantly reduces system imbalance. Although both schemes perform comparably, *S1* has a slight advantage in terms of cost. For the rest of the paper, we used *S1* for all simulations.

5.3 Peer Start-up delay and System Scale

We compare the performance of our method with a basic mechanism with no population

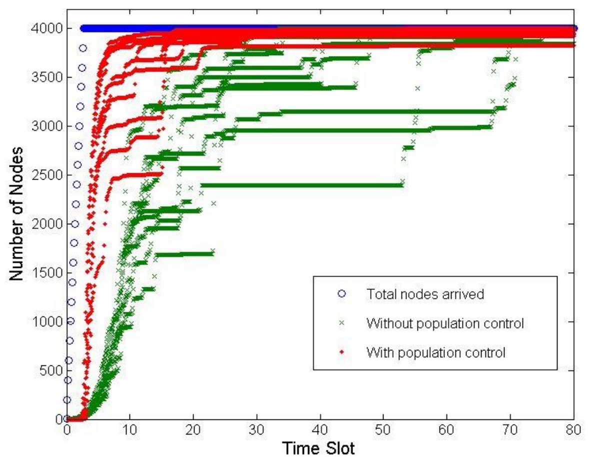

Fig. 1. System scale with and without population control measures.

control. In this, the network contains a single feed-forwarder list. New peer on joining the system fetches this list and sends connection requests to the existing peers. If successful, they then register themselves on the list. If unsuccessful, the peer will fetch the updated list and start again.

The time scale is divided into time slots where each time slot denotes the time taken by a peer to get the feed in adopting the basic mechanism described above under no flash crowd condition. The simulation is run for one generation with $M=4000$, $U=2$ and $k=2$. The peers arrive with a uniform rate $\lambda=2000/s$ for a period of 2 seconds.

We can see in Fig. 1 that by adopting population control measures, the system scale can be greatly enhanced. After the initial delay incurred in placing the start-up peers in levels, the system scales very fast. However, the system scale is not 100% in this case because

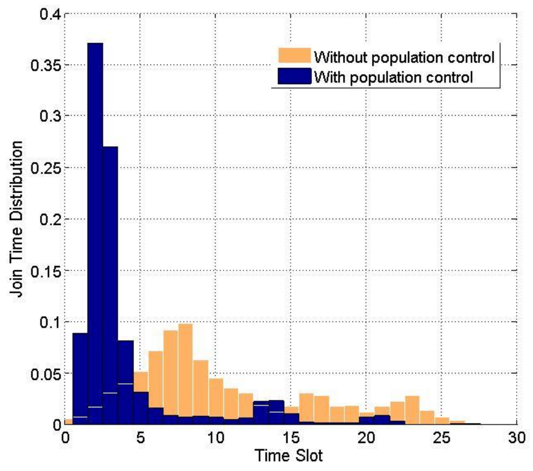

Fig. 2. Peer Startup distribution with and without population control measures

some nodes are unable to register themselves in proper levels as seen by the results of Section 5.2. We are continuously working on it to further modify our algorithm.

From Fig. 2, we can see that more than 80% of the peers are able to get the feed within 6 time slots. This percentage is very less without flash crowd control.

Thus, as evident from the experimental results, the proposed method performs significantly well in handling flash crowd.

6. CONCLUSION

By proposing a distributed population control algorithm to handle flash crowd, we were not only able to reduce peer start-up delay but also able to provide a better system scale as compared to system with no population control. The newly arrived peers instead of competing with each other for scarce system resources

Table 1: Extreme Flash Crowd

Param	***I***	***C***	***F***
S1	2.432	0.014	1.121
S2	2.023	0.025	1.536
S0	2.055	0	1.200

Table 2: Uniform Arrival

Param	***I***	***C***	***F***
S1	0.488	0.638	1.019
S2	0.103	0.707	0.598
S0	6.290	0	33.61

Table 3: Poisson Arrival

Param	***I***	***C***	***F***
S1	0.004	0.632	0.594
S2	0.202	0.706	0.917
S0	5.32	0	30.85

were arranged in different levels of the tree and connected to the network as a whole. Flash crowds in P2P live streaming systems can have a degrading effect on the QoS and population control measures can help counter it effectively.

7. REFERENCES

[1] M. Castro, P. Druschel, A. M. Kermarrec et al., "SplitStream: high-bandwidth multicast in cooperative environments." In *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 298-313. ACM, 2003.

[2] B. Li, S. Xie, G.Y Keung, J. Liu, I. Stoica, H. Zhang, and X. Zhang, "An Empirical Study of the Coolstreaming+ System," *Selected Areas in Comm., IEEE Journal on* , vol. 25, no. 9, pp. 1627-1639, Dec. 2007

[3] B. Li, S. Xie, Y. Qu, G.Y Keung et. al, , "Inside the New Coolstreaming: Principles, Measurements and Performance Implications," *Proc. INFOCOM,* April 2008.

[4] X. Hei, C. Liang, J. Liang, Y. Liu, K. Ross, “A MeasurementStudy of a Large-Scale P2P IPTV System,” *IEEE Trans. Multimedia*,vol.9, no.8, pp. 1672-1687, Dec. 2007.

[5] H. Yin, X. Liu, T. Zhan, V. Sekar, et. al, “Design and Deployment of a Hybrid CDN-P2P System for Live Video Streaming: Experiences with LiveSky,” *Proc. 17th ACM Int'l Conf. Multimedia,* Oct. 2009

[6] F. Liu, B. Li, L. Zhong, B. Li, and D. Niu, “How P2P streaming systems scale over time under a flash crowd?,” *Proc. USENIX IPTPS,* p. 5, Apr. 2009.

[7] F. Liu, B. Li, L. Zhong, B. Li, H. Jin, and X. Liao, “Flash crowd in P2P live streaming systems: Fundamental characteristics and design implications,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 7, pp. 1227-1239, Jul. 2012

[8] Y. Chen, B. Zhang, and C. Chen, “Modeling and performance analysis of P2P live streaming systems under flash crowds”, *Proc. ICC 2011, Kyoto, Japan,* p. 1-5, Jun. 2011.

[9] T. Chung, O. Lin, “A batch join scheme for flash crowd reduction in IPTV systems,” *Proc. ICPADS 2011, Tainan,* p. 823-828, Dec. 2011.

[10] H. Wu, K. Xu, M. Zhou; A.K Wong, J. Li, and Z. Li, "Multiple-tree topology construction scheme for P2P live streaming systems under flash crowds," *WCNC, 2013 IEEE,* pp. 4576-4581, Apr. 2013

[11] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” *ACM SIGCOMM 2001*

[12] J.W. Lee, H. Schulzrinne, W. Kellerer, Z. Despotovic, "0 to 10k in 20 Seconds: Bootstrapping Large-Scale DHT Networks," *Communications (ICC), 2011 IEEE Int'l Conf. On* , pp. 1-6, 5-9 June 2011

[13] M. Roussopoulos, and M. Baker, “CUP: Controlled update propagation in peer-to-peer networks,” *Proceedings of the 2003 USENIX Annual Technical Conference,* 2003

[14] A. Montresor, and M. Jelasity, “PeerSim: A scalable P2P simulator,” *Proc. of the 9th Int. Conf. on P2P*, pp. 99-100, Sept. 2009.

[15] peersim.sourceforge.net

[16] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, “Tapestry: An infrastructure for Fault-tolerant Wide-area Location and Routing,” Tech. Rep.UCB/CSD-01-1141, U.C.Berkeley, Apr. 01

[17] A. Rowstron, and P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems,” in *MiddleWare,* Nov. 2001