CSC586B Course Project Report

A Comparison Study of Block Scheduling Algorithms in P2P-VoD Systems

Le CHANG
Uvic Number: V00687101
Department of Computer Science,
University of Victoria, Victoria, BC, Canada V8W 3P6
Email: lechang@uvic.ca
Abstract

Video-on-Demand streaming on P2P networks has become increasingly popular in recent years. This report presents a comparison study of block scheduling algorithms in P2P VoD networking. Global rarest-first and a promising technique named network coding are reviewed and simulated. The influence of the size of blocks to the performance is also covered in the simulation. The results confirm that without any knowledge of global block distribution, network coding can achieve at least the same performance with global rarest-first, which was considered to be the optimal solution in block scheduling but required global information that was unknowable in many P2P systems. And with smaller pieces, if ignoring overheads, both of Global rarest-first and network coding will achieve better performance.

Index Terms

P2P VoD, block scheduling, network coding, simulation.

I. INTRODUCTION

Peer-to-Peer (P2P) video streaming applications have become popular in recent years since Coolstreaming [1] appeared as the first live P2P video streaming system. A tendency can be easily found is that the P2P video stream data tend to dominate the network nowadays, which arouses research interests in this area.

Motivated by the success of P2P content distribution systems like BitTorrent [2] and Voice-of-IP systems like Skype [3], P2P networking technique is applied in the area of video streaming. With the improvement of network bandwidth, multimedia services such as IPTV [4] (i.e. an online service providing TV video programs), was proposed. As IP multicast [5] has not been widely deployed because of some practical problems in implementation, researchers turn their eyes onto P2P networks [6], in which multicast is implemented simply in end systems above application level. This leads to intensive research work in P2P live streaming to propose a number of P2P live streaming systems [1] [7] [8] [9] [10]. And many are deployed with high viewing quality, lower server burden, and high scalability and reliability. Some representative P2P live streaming systems are Coolstreaming [11], PPLive [12], PPstream [13], UUsee [14], Anysee [15], and Joost [16] etc. On August 2005, when PPLive was carrying "Super Girl Finals" in China, a number of 500,000 people watched online at the same time are recorded, setting a
milestone of P2P live streaming systems.

Very recently, P2P VoD (Video-on-demand) becomes the new interest of researchers and program developers. Different from P2P Live Streaming which provide on line live video programs, this new kind of service allow users to choose different videos, or different parts of a certain video to watch. Some even allow VCR operations on video programs. This brings to large extent flexibility to users. As pointed out in [17] that P2P-VoD could significantly save server loadings, many researchers are motivated to work on this topic. Many such systems or the VoD versions of original Live Streaming applications are developed and have a large population of users. Typical systems are PPLive [12], Joost [16], GridCast [18], PFSVoD [12], PPStream [13], UUSee [14] etc.

Different issues are emphasized on these two kinds of applications respectively due to the different services they provided. In a live streaming application, a live video program is disseminated to all peers in real time. Intuitively, this requires that the video playbacks on all users are synchronized or at least approximately synchronized. In this case, the buffer size on each peer may not be very large (typically 2 minutes) because it only needs to buffer a small part of the video stream to maintain a stable playback rate.

On the contrary, in P2P VoD, a huge size of buffer is needed in order to satisfy the diversified request from peers on different kinds of video programs. However, there is no constriction on synchronization of every node. Instead, the system should allow users to arrive at arbitrary times to watch arbitrary parts of arbitrary video programs, while providing low start up delays for users. The fact that different users may be watching different video parts at any time can greatly impact the efficiency of original P2P streaming protocol. The lack of synchronization among users causes the block sharing among users to be scarce while the complexity of the block scheduling algorithms increases.

Block scheduling is one of the kernel issues of in P2P VoD that aims at providing users with high-quality service while maintain a high utilization of system resources. In [19], Huang et al discusses about block scheduling in PPLive (the VoD version) referring to piece selection and transmission strategy. It is said in the paper that PPLive applies a mixed strategy for block scheduling, giving first priority to sequential and then rarest first. Annapureddy et al [20]
propose another in depth study of block scheduling algorithms and demonstrate through extensive simulation experiments that combining with sequential selection of segments, network coding can perform better than global rarest first. Concerning to block size, Huang, et al [19] refers to multiple layers of segmentation from a perspective of implementation of practical systems. And Annapureddy et al [20] applied fixed size to blocks and segments.

In this report, a comparison study of block scheduling algorithms is proposed. A simulator using python 2.3 is developed to test performance of these algorithms. Different sizes of blocks are also considered in the simulation to gain insights of its influence on the system.

The remainder of this report is organized as follows: Section 2 discusses block scheduling algorithms in P2P VoD. In section 3 we focus on our simulation work. We define the problem, scenarios and measuring metrics to be covered in this project and describe how we implement our simulator. We also present our simulation results and related discussions. Finally we mention future work in section 4 and conclude this project in section 5.

II. BLOCK SCHEDULING ALGORITHMS IN P2P VO D

A. Concepts and Principles

The most important issue in designing a P2P VoD system is block scheduling, which focuses on dealing with dispatching and transmitting blocks within a system efficiently and optimally.

The object of block scheduling is, (1) to satisfy the playback rate of the peer itself; (2) to contribute to diversity of global chunk bitmap to gain global optimization as well as cut down redundant data transmission. Normally, compromised approaches are adopted to solve the problem.

For P2P Live Streaming systems, this topic appears comparatively simpler. Because as all users are watching the same part of the same video, there will be more sources for a single user. And the size of content each peer need to cache is small which reduce the difficulty for block scheduling. As to P2P VoD, the desynchronization of interests among users causes the number of object blocks sharing among users to be small that a balance should be achieved between local interests and global diversity to ensure requests for certain content could be satisfied while local playback rate will not be largely influenced.
To schedule blocks in VoD, several techniques are involved to meet the requirement of this particular service.

1) **Larger amount of Buffering Storage:** To overcome the problem of lack of synchronization among peers, the system requires a peer to cache video blocks on local hardware. This is a notable difference from P2P-live-streaming systems that P2P VoD usually requires users to contribute larger amount of storage. With this large buffer size, requests for blocks are likely to hit these cached entries. In [19], Huang et al. declare this to be 1 GB in usual in PPLive. In fact after a user installs such version of PPLive and run the system for the first time, he or she will discover an unknown kind of file of 1 GB residing in the hard disk. Annapureddy et al address this issue in their work [20] by assuming that each peer has unlimited size of storage to contribute.

2) **Segmentation:** Segmentation refers to how to divide a video into multiple pieces. This will affect the scheduling algorithm and overlay topology constructing. The server, which also behaves as the source of video content, conducts this work by dividing a video program into small pieces and then sends them to requesters. Note that there is a tradeoff that such pieces could be very small to be easy to schedule and to improve content diversity; or to be bigger to reduce control overhead. Usually, a mechanism of multiple-level segmentation is applied in order to combine advantages of different block scheduling algorithms. That is, after dividing a video program into small pieces, we divide a piece again into smaller pieces, or named subpieces. According to the particular use and the level it belongs to, block may be named ‘chunk’, ‘segment’, ‘piece’, or ‘subpiece’. In PPLive [19], the smallest unit is called subpiece which is used for transmission. Several subpieces build up a piece, which is used for playback. Chunks that contain multiple pieces are used for advertisement. And a movie consists of multiple chunks. This approach combines some advantages of big and small chunks/blocks together. In [20], a movie is divided into segments and blocks where a segment consists of many blocks. Different algorithms are applied on scheduling these two levels of pieces, which is quite similar to PPLive. In our project, we followed [20] to divide a movie into 2 levels of blocks where we call bigger ones ‘segments’ and smaller ones ‘blocks’, respectively.

3) **Piece Selection:** This is to determine the order of blocks to be downloaded. There are many methods that the sequential algorithm conforms to watching behaviors of humans while rarest
first benefits exchanging proficiency. And we easily know that the simple approach sequential will be a naive approach as it would remove the diversity of the whole global chunk bitmaps. This is also our major focus in this project and we specifically call this issue as ‘block scheduling’.

4) Overlay Building: This refers to from which peers to download? As a block may be cached by many peers, how to efficiently make use of these resources available in the system from the global network to facilitate building an optimized Overlay will largely influence the performance of the system. There could be some considerations based on geographical information, the same target, or quality of service. In PPLive, if there were no enough neighbors assisting a peer to download a video, then the server will supplement the need [19]. [20] proposes a mixed approach based on the same target and quality of service. In our simulation, we don’t consider much about this issue and simply use a random method to assign neighbors to peers.

B. Existing Block Scheduling Algorithms

A peer downloads blocks from other peers using a pull method. That is, a peer demanding contents send requests to other peers to ask for data. There are several considerations for selecting which piece to download first:

1) sequential: This intuitive method is to select pieces according to the closest order, which conforms to our watch behaviors. If a block is the immediate interest of a peer then it is sent. The problem this method tend to raise the probability of content under the scenario of flash crowd.

2) rarest first: Another algorithm taking global diversity of blocks into consideration is to select the piece that is the rarest in the system (newest). Selecting the rarest piece helps increase the diversity of pieces, therefore enhance the quality of service of the system and help it scale, which is clearly explained in [21]. Note that this requires a node to know global information, which may not be applicable in many network systems.

3) network coding: Network coding is applied for improving the throughput by making optimal use of bandwidth resources of a network for content distribution in [22] [23] [24]. Avalanche [24], [25] has proposed randomized network coding in content distribution to reduce downloading time. [26] and [20] have studied the effect of network coding in P2P live streaming
C. Network Coding in P2P VoD

An intuitive benefit of Network coding for block scheduling is that it needs no selecting algorithm for blocks. Any received block is useful with a high probability because these blocks are coded by senders. A downloader only needs to wait until enough number of blocks for decoding. The drawback is that waiting for all the blocks to arrive to start watching is not applicable in the context of VoD. This problem can be overcome by restricting coding within a limited length of segments that adopted by [26] and [20]. Fig. 1 [26] demonstrates how random linear network coding can be applied in VoD using this approach.

In linear random network coding, a movie is divided into multiple segments where each segment is further divided blocks. Assume that for a movie we have m segments and n blocks \([b_1, b_2, \cdots, b_n]\) within a segment. Each \(b_i\) has a fixed length of k bytes (i.e. block size). The process of coding can be demonstrated by Fig. 1 [20].

When sending a coded packet, the sender checks the blocks (from \(b_1\) to \(b_t\)) it already has within the corresponding segment and picks random coefficients \(c_1, c_2, \cdots, c_n\), and generates
Fig. 2: A brief description of network coding [20].

\[ E_1 = \sum_{k=1}^t c_i \cdot b_i. \] Note that the range of \( t \) can be up to \( n \) and all the coefficients are chosen from the operations done in a finite field. The sender then forwards the coefficient vector and \( E_1 \) to receivers.

From the perspective of a receiver, it is receiving linear equations with \( n \) unknown numbers. So it needs to wait for \( n \) equations if they are linear independent to each other to start extract \( b_1 \) to \( b_n \). And any equation within the segment is useful (i.e. linear independent with other received equations) with a high probability if the size of the finite field is as large as \( 2^{16} \), according to [27].

We now compute the overhead in transmitting a coded block. The coefficient vector is usually stored in the head of a coded block and is responsible for overhead. For the finite field of \( 2^{16} \), we need \( n \) coefficients, each of which has a size of 16 bits, namely 2 byte. Thus the space to store the coefficient vectors will be \( 2n \) bytes. When \( k = 1KB \) and \( n = 10 \), we have 20 byte overhead for each coded block of 1KB. So, to reduce overhead, we tend to have larger size of blocks.

In summary, network coding reduce the probability of sending redundant blocks within a network system and thus makes optimal use of bandwidth and maximizes system throughput.
III. SIMULATIONS AND RESULTS

In this section we describe our simulations and related results. We aim at testing the performance of sequential-rarest-first and sequential network coding for different size of blocks.

A. Simulation Model

In this simulation, we develop a simulator using python 2.3. Our simulator runs in discrete time slots, which we call 'rounds'. For each round, a series of events happen and complete, then the system moves to next round.

We assume there is one server (i.e. source) holding exactly one movie with service capacity of 5, which means within one round, the server is able to serve at most 5 peers for their downloading requests.

The movie is divided into several segments, and every segment is further divided into blocks. In our simulation, we have 10 segments and different numbers of blocks for the same movie. At the beginning only the server has the movie. The object is to distribute the movie to peers fast and fairly.

We define upload bandwidth and download bandwidth for each peer. We can easily have asymmetric upload and download capacity by regulating these parameters but we follow the setup by Annapureddy et al [20] to set these parameters to one, meaning that within each round, a peer can only download from at most one peer, and serve at most one peer.

When a new peer joins the system, it is assigned with random active neighbors of a fixed number within the system. We set this number to be 8, namely each peer has 8 neighbors. We define peer behaviors as follows:

- A peer starts to download the movie from the start, namely from the first segment, when it joins the system. It can send requests either to the server or its neighbors, and then starts downloading a block if its request is accepted and the object block is available on the object node. A peer can also serve other peers if it is in the neighbor list of the requesting peer. This is also restricted by the availability of the block.
- A peer will never leave the system. If it completes downloading the movie, it starts to behave like a server with service capacity of 1.
• A peer downloads segments sequentially. Within a segment, it applies certain algorithms in different scenarios, either global rarest-first or network coding, to request for blocks.

• A peer attempts to start decode if it gets the enough number of blocks in network coding. If a segment is divided into 10 blocks, when a peer receives more than 10 coded blocks, it tries to decode them. However, as these coded blocks may be linear dependent, the attempt may not be successful. This refers to decoding successful rate.

B. Scenarios

In the simulation, we are concerned about system performance under flash crowd. We expect the performance with benign arriving patterns to be better than that with flash crowd. To be more specific, we assume 20 peers arrive in the system at the same time. As the service capacity of the server is only 5, it is unable to serve all the peers. So peers are forced to exchange blocks in order to compensate the upload bandwidth of the system. We apply segmentation to divide a movie into segments and blocks. We use sequential scheduling for segments and network coding or global rarest-first for blocks.

Four scenarios are tested in our simulation:

• Sequential-network coding, 10 segments, 10 blocks (SNC 10×10)
• Sequential-network coding, 10 segments, 20 blocks (SNC 10×20)
• Sequential-global rarest-first, 10 segments, 10 blocks (SRF 10×10)
• Sequential-global rarest-first, 10 segments, 20 blocks (SRF 10×20)

Note that within a segment, if we have more blocks, namely we will have blocks of smaller size. As overhead is ignored in our simulation, the smaller blocks can be transmitted 2 times faster that in each round, we can transmit 2 blocks for each peer, given the bandwidth 1 of that peer. In sequential-global rarest first, we assume a node has some method to gain global information of block distribution.

C. Measuring Metrics

Three major measuring metrics are covered in this project.
Fig. 3: This hypothetical graph shows the calculation of sustainable playback rate, given the setup time. The y-axis shows the number of consecutive blocks, while the x-axis shows the time [20].

1) **Throughput:** The term throughput in our simulation refers to system throughput which means the total number of blocks transmitted within one round. This metric relates to the utilization of the system resources. For example, in a certain round, if there are $x$ blocks exchanged within the system, then the throughput is $x$. As there are 20 peers in our simulation, so the maximum number of throughput is 20.

2) **Goodput:** For a given setup time (i.e. amount of initial buffering), the sustainable playback rate is calculated as the maximum slope of a line that does not exceed the y-coordinate at any time of the curve of consecutive arrived blocks (see Fig. 3 [20]). The rate is called the goodput. This characterizes how fast the playback could be supported by other uploaders. An easy way to see goodput in our simulation is to compute the average finishing time of peers. Earlier the finishing time refers to better the goodput.

3) **Successful Decoding Rate:** As mentioned in Simulation Model, the attempt to decode may not be successful. We record all attempts to decode and successful attempts to decode and then compute the rate as successful attempts/total number of attempts. This reflects how efficient network coding is on avoiding transmitting duplicated blocks.
D. Implementation of Simulator

We use python 2.3 to develop our simulator. For network coding, we download and install two packages, scipy and numpy, to implement operations of matrix.

1) Simulating Events: We give an example of events happen in each round in the system:
   - Step1. Refresh system parameters for current round.
   - Step2. Shuffle the list of peers in order to implement random selection.
   - Step3. Choose five peers to download from the server.
   - Step4. Peers not chosen in step 3 download blocks from their neighbors.
   - Step5. Summarize and record statistical data for current round. For network coding, check if a peer can start decoding and if so, let it attempt to decode.
   - Step6. Move to next round.

2) Simulating Network Coding: In our simulation we simulate random linear network coding. For simplicity, we treat blocks and coefficients as integers. For every time of coding in SNC 10×10, we generate random numbers in the range of (1, 4) as coefficients. As we may do coding for approximate 10 times, the coefficients may be as high as 4^{10}. By doing so we simulate the finite field with size of 2^{20}. For SNC 10×20, the range of random number will be (1, 2) which also roughly covers a space of 2^{20}. When the server receives a downloading request, it simply codes every blocks within a corresponding segment by generate a linear equation with 10 or 20 unknown numbers. When a peer receives a request for downloading, it codes all the arrived coded blocks it has in the segment. Then a new coded block is sent to requesters. On the other side, the downloader waits to accumulate enough number of coded blocks, namely linear equations. For SNC 10×10, if 10 linear independent equations within the same segment are received by a peer, a decoding will take place and the segment is then ready to playback. We compute the rank of the coefficient matrix of the linear equations to determine whether the decoding is successful.

3) Simulating Global Rarest First: Global information can be simply recorded and looked up by using an array to store numbers of each block existing in the system. For example, if block b1 in segment s1 resides both in the server and 3 other peers, then number 4 is stored in corresponding position of the array. Every peer attempts to find in its neighbors the global rarest block within its requesting segment. To improve the performance of global rarest first, we
<table>
<thead>
<tr>
<th>Scheduling algorithm</th>
<th>Average finishing time</th>
<th>Average throughput</th>
<th>Successful decoding rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNC 10×10</td>
<td>116.1</td>
<td>17.9</td>
<td>66.6%</td>
</tr>
<tr>
<td>SNC 10×20</td>
<td>110.3</td>
<td>18.9</td>
<td>50.6%</td>
</tr>
<tr>
<td>SRF 10×10</td>
<td>115.3</td>
<td>17.0</td>
<td>-</td>
</tr>
<tr>
<td>SRF 10×20</td>
<td>108.8</td>
<td>18.2</td>
<td>-</td>
</tr>
</tbody>
</table>

slightly modify the algorithm by allowing a peer to request for the second global rarest block in its neighbors when the rarest one is not found, and for the third rarest one if the rarest two are not available. By doing so, we expect this algorithm to be the optimal solution in the context of our simulation.

E. Simulation Results and Discussions

For each scenario, we run our simulator for 150 rounds to ensure that the movie is successfully distributed to all the 20 peers. We repeat each experiment 100 times to compute the average performance.

Fig. 4 and Fig. 5 show the average goodput and throughput of SNC 10×10 and SNC 10×20 respectively. Fig. 6 and Fig. 7 shows the average performance of SRF 10×10 and SRF 10×20. Statistical figures are summarized in Table I.

From Fig. 4 to Fig. 7 and Table I we try to interpret the data and have some conclusions for our simulation results.

- Without global knowledge, network coding can achieve nearly the same performance of global rarest first, which is unfeasible in many network systems. So in the context of P2P VoD, network coding can be a practically feasible substitute for global rarest first.
- Ignoring overhead, if we have smaller pieces, then we have better performance of the system. This is because pieces are distributed more evenly among the system. To demonstrate this point, consider an extreme example that the movie has exactly one segment and one block. The performance in this case goes back to file downloading without segmentation in C/S
model.

- The successful rate of 66.6% in SNC 10×10 means each decoding will be successful in two attempts. On other words, on average 11 blocks are needed to ensure the success of decoding. The extra 1 block can also be considered as overhead. From this point of view, the overhead does not increase notably in SNC 10×20 because the successful rate of 50.6% also means that on average only one extra 1 block is needed for decoding. However, if we take overhead into consideration as discussed in section 2, the overhead will be at least doubled because the workload is halved while space to send coefficient vectors maintains unchanged.

- The successful rate decreases from 66.6% to 50.6% when the size of piece goes down. This is because in SNC 10×20, for each time we generate coefficients in the range of (1, 2). Although the whole space covered can be up to $2^{20}$ equal to $4^{10}$ of SNC 10×10, the range (1, 2) provide worse diversity when applying multiplication to 1 or 2, compared with (1, 4). As our simulation is only to approximately simulate the operation of coding and decoding, we still need to conduct further study on this issue.

IV. FUTURE WORK

In this section we discuss future work and possible improvement of our simulation project.
As mentioned in Section 3, we simulate network coding using arithmetic method. In real systems, encoding and decoding operations are over a Galois Field GF($2^{16}$). To gain more insights of network coding in P2P VoD, implementation and test of network coding in practical applications is a good choice.

Using our simulator, we are able to simulate many scenarios other than the 4 ones mentioned in this report. An interesting topic is to simulate distribution of multiple movies and different
interests towards these movies among peers, which refers to popularity of movies. Huang et al have presented statistical results concerning this issue in [19].

Another topic named ‘churn’, which refers to the highly dynamic network of peers, is to be implemented in our simulator. Reliability of a system under such scenarios is also a further work of this simulation project.

V. CONCLUSION

In this report, mainstream block scheduling algorithms in P2P VoD are discussed and studied. A simulation experiment is designed and conducted to test performance of network coding and global rarest-first algorithm P2P VoD. Simulation results show that based on a combination of sequential algorithm for segments and encoding for blocks, network coding can be used to support acceptable playback rate and high throughput in P2P VOD systems with limited server capacity and user bandwidth. And smaller size of blocks will lead to more diverse distribution of blocks thus provides better performance if overhead can be neglected.

REFERENCES