Performance Measurement and Bottleneck Analysis for Streaming Media Servers

Yan He[1], Haocheng Huang[2] and Jinyao Yan[3]

Abstract. With the rapid development of Internet and the popularity of media contents, the performance of stream media server is becoming an important factor for stream-media systems. By measuring and analyzing the performance constraints of the media server, we find the way to improve the performance of stream media server, and then improve media service. In this paper, we discuss the performance metrics and constraints of stream media server based on the analysis of media streaming system and service. To evaluate the performance of stream media servers, we change the following conditions as the performance metric: the number of connections, the number of concurrent active media files, the format of the media and the bit-rate of the stream. We measure the CPU idle time, IO wait, memory usage, network bandwidth as metrics to find the bottleneck and constraints for streaming server. By increase the capacity of identified bottleneck, we may further improve the performance of steam media servers.

Keywords: Streaming media servers • Performance measurement • Bottleneck

1 Introduction

Traditionally, the quality of service for streaming media systems depends on the network bandwidth, network delay, efficiency of media coding compression and

[1] Yan He (✉)
School of Information and Engineering, Communication University of China
e-mail: hey4tc@gmail.com

[2] Haocheng Huang
Computer and Network Center, Communication University of China
e-mail: mickypc@cuc.edu.cn

[3] Jinyao Yan
Computer and Network Center, Communication University of China
e-mail: jyan@cuc.edu.cn

decoding in client-side, and so on. In the past, the network transmission capacity is one of the key factors to constrain the development of streaming media. With the development of broadband network, the quality of service for streaming systems is more constrained by the performance of streaming servers. In addition, streaming media application have been paid more attention and became one of the main applications of high-speed networks. Typical streaming media applications include VOD, video conference, remote education, digital library and so on.

With the increase in bandwidth and the use of load balancing and CDN, the performance of streaming media server has become an important factor restricting streaming media application. Finding out the bottleneck of media server is one of the prerequisites of further optimizing streaming systems and it is also the theoretical basis to evaluate the performance of streaming media servers. To date, the research knows little about how to find out the bottleneck of the media server and to estimate the performance [2, 3, 4, 5]. Therefore, we, in the paper, evaluate the performance of streaming servers and attempt to find out an experimental method to analyze the bottleneck of streaming servers.

The rest of the paper is organized as follows. In Section 2, we introduce the model of media service and performance metric of media server. In Section 3, we describe the experiment settings and the methodology. In Section 4, we analyse the result of the experiment. In Section 5, we conclude this paper.

2 The model of media service and performance metric

Streaming media server is the basic functional unit to provide service to users. Its performance directly affects the service ability of streaming media system. In the measurement of streaming media server, the most important indicators are throughput capacity of media and the concurrent request quantity. Let's introduce the process of streaming media server:

1. When a request arrives, the media server reads the media contents from the hard disk and stores them in memory;
2. Before media files are send to the network, they should be processed by CPU as copying, segmentation, being packed based on the protocol;
3. The contents are packetized in memory, and then sent to the NIC;
4. The contents are sent to the network.

Through the above process, there are four key factors affecting the performance of streaming media server: CPU, memory, disk reading ability and the network throughput rate.

Therefore, we observe CPU utilization, I/O wait, memory free size, reading and writing bit rate of disk and network flow rate in our test to get the server bottleneck.

3 Experiment settings and the methodology

3.1 Experiment settings

As shown in table 1, we use Linux redhat enterprise 5 as our operating system; *Live555* and *lighttpd* to provide the RTSP [1] and http service respectively; *Systat* to collect performance data ,i.e. CPU, hard disk, memory data. Furthermore we use *oprofile* to dig out the reason for the performance issue.

Table 1: Software Configuration

Item	*Description*
OS	Linux redhat Enterprise 5
File System	ext3/tmpfs
Media Server	Live555/ lighttpd
Imitation Client	openRTSP/wget/VLC
Performance collection tools	Systat
Server analysis tool	oprofile

Streaming server hardware are shown in Table 2:

Table 2: Hardware Configuration

Item	*Description*
CPU	Intel(R)Xeon(R)CPU X5460@3.16GHz
Disk	Sata 7200rpm/FC
Memory	2G
Network Card	1000mbps/100mbps

3.2 Experiment methodology

On the server, we run the media server program and start a Perl script to call *Systat* tools to collect server performance data. On the other hand, in the client we use the *openRTSP* to simulate connecting to server with different concurrent connec-

tions, flow rate, the number of access files and so on. At last, we can get the collected performance statistics data on server. For analyzing the bottleneck of server, we export the data exported to excel to make a chart for observation.

3.3 The choice of sampling interval

As the minimum sampling interval of *Sysstat* is one second, we choose the sampling interval based on the following principle:

1. Sampling interval should be as small as possible, so that the data obtained will be more accurate and conducive for our bottleneck analysis;
2. The impact on the server performance by running the tool should be as small as possible which should not affect the results of the experiment.

Therefore, we run the sampling tool without running the media server program, and observe the performance of the server from which to select the most appropriate sampling interval. The sampling both in the intervals of 1s and 5s consume almost the similar CPU resource, while 1 second is the minimum interval. Therefore we select 1 second as the sampling interval.

4 Performance analysis

In this section, we present the experiment results of two representative cases and our analysis to find the bottleneck of streaming server. In the results, *idle* means the percentage of the time interval when the CPU was idle; *iowait* is time that the processor/processors are waiting (i.e. is in an idle state and does nothing), during which there in fact was outstanding disk I/O requests.

4.1 Experiment case 1

Experimental conditions are as follows. The number of concurrent request is 30; the flow rate is 5 MB/s, the media format is MPEG2, and we change the number of accessing media files from one to thirty by ten.

We only show the result which the number of accessing media files are 1 in Figure 1 and 30 in Figure 2.

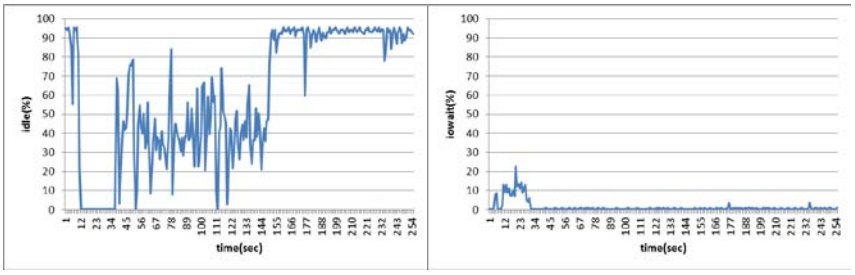

Fig 1: The CPU idle (left) and the IOwait (right) when the access file quantity is 1

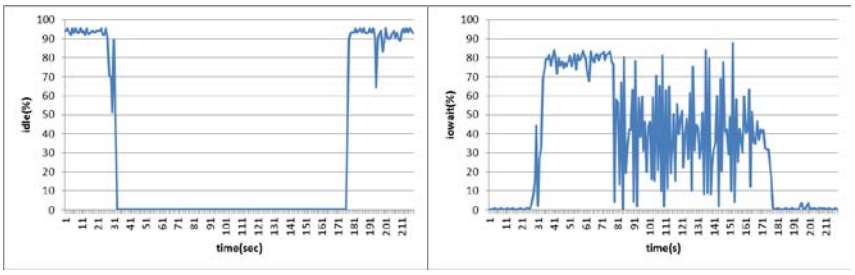

Fig 2: The CPU idle (left) and the IOwait (right) when the access file quantity is 30

We make the following observation from Figure 1 and Fig 2. When the client has not connected the server yet, idle and IO wait almost are 100 and 0 respectively. When the client starts visiting, I/O wait increases quickly and therewith CPU idle reduce. When the I/O operation declines, CPU idle recovers very slowly. When the visits complete, the CPU idle and IO wait restore to the level before visiting.

Figure 1 and 2 also show that the load of CPU is different in different stages of the streaming service. In the initial stage of connections, in addition to the file replication, segmentation and protocol packaging, there will be interactive process need to establish connections which lead to more resource-intensive than the smooth connection stage.

To compare the results, it is observed that, with the increase in the quantity of access files, IO wait value increases gradually. When the quantity is 30, IO wait reached 80%, a large proportion of occupied CPU. CPU is not fully used for processing, as the most of the time is waiting for I/O reading and writing. As the CPU does not reach bottleneck, and I/O ability become the server performance bottlenecks.

With the number of access files rising, the remaining amount of memory decreases. How the remaining memory is finally stable at 50,000KB, which is not shown in the figures. As Memory is not exhausted, it is not a bottleneck. We consider the I/O performance as an important bottleneck.

To further verify that the bottleneck is the I/O read-write operation, the program will be ported to 64-bit system, which can handle more memory space. We

compare server performance between *ext3* and *tmpfs*. *ext3* is build on *sata* while *tmpfs* use RAM or swap for storage. In order to ensure *tmpfs* using the real memory, we specially use "*swapoff -a*" to close off swap partition. First we use *dd* program to test, we can conclude that the I/O performance of the *tmpfs* is ten times better than ext3. Then, we make the test of 30 access files again and find out the I/O bottleneck is almost eliminated in *tmpfs*.

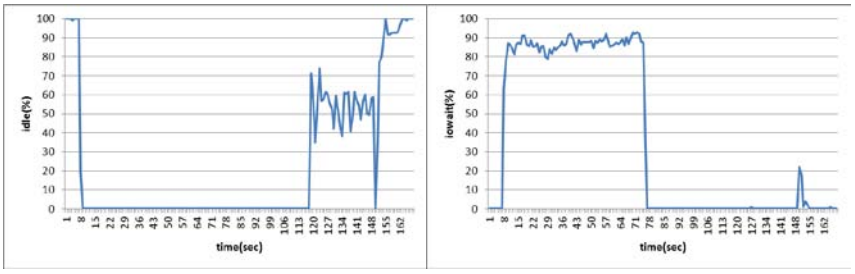

Fig 3: the cpu idle (left) and the iowait (right) running respectively on ext3

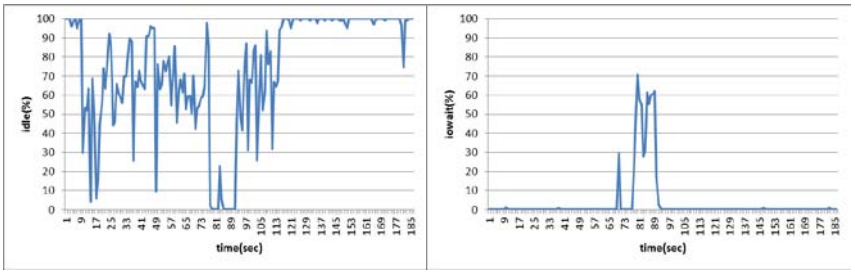

Fig 4: the cpu idle (left) and the iowait (right) running respectively on tmpfs

Figure 3 shows the server performance in Sata hard disk which is similar when running on 32-bit system. When the number of access files reaches 30. IO wait maintained at about 80%, idle remains 0. We conclude the I/O read-write operation is the bottleneck.

Figure 4 shows the result of server running in tmpfs. Due to the use of RAM memory, IO wait almost 0, that only appears a peak in the 85s. As CPU idle is about 70%, server performance is significantly improved. Therefore, disk I/O capacity is the main bottleneck of the server performance in this case.

4.2 Experiment case 2

We list the experimental conditions below: the number of access file is one; the flow rate is 5 MB/s, the media format is MPEG2, and we change the concurrent request number from one to thirty.

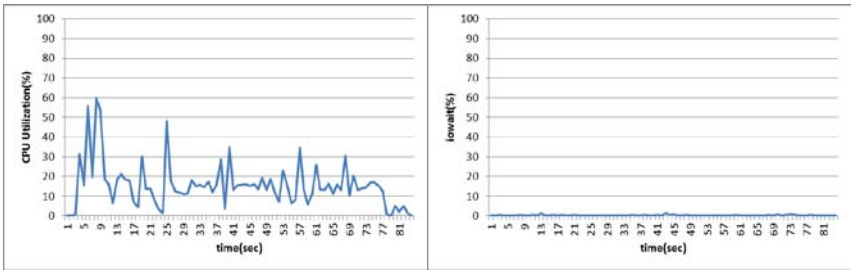

Fig 5: The CPU utilization (left) and the IO wait (right) when the connection quantity is 10

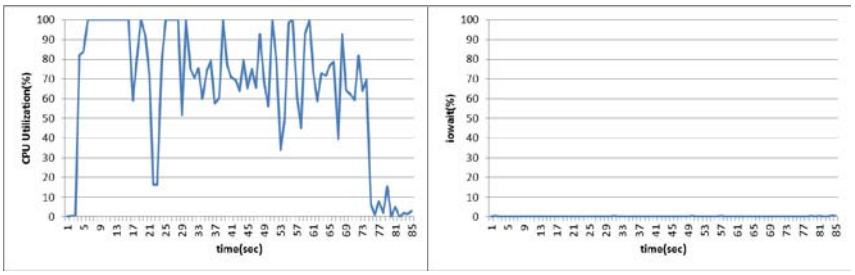

Fig 6: The CPU utilization (left) and the IO wait (right) when the connection quantity is 30

As it can be seen from Figure 5 and 6, when the concurrent requests gradually increase, the utilization of server's CPU raise. When concurrent requests increase to 30, CPU utilization reaches about 80%, CPU becomes the server bottleneck. In Figure 6, the number of access file is just one, IO Wait is almost zero, so the CPU is not waiting for the disk I/O request.

To find out what the CPU is working at, we use oprofile to collect the information of CPU usage. We show results in Figure 7:

```
CPU: CPU with timer interrupt, speed 0 MHz (estimated)
Profiling through timer interrupt
samples   %          symbol name
71        62.2807    MPEG1or2VideoStreamParser::parseSlice()
4          3.5088    MPEGProgramStreamParser::parsePackHeader()
3          2.6316    BasicTaskScheduler::SingleStep(unsigned int)
3          2.6316    StreamParser::saveParserState()
2          1.7544    AlarmHandler::handleTimeout()
2          1.7544    MPEGProgramStreamParser::parsePESPacket()
1          0.8772    BasicHashTable::Iterator::Iterator(BasicHashTable&)
1          0.8772    BasicHashTable::keyMatches(char const*, char const*) const
1          0.8772    DelayQueueEntry::handleTimeout()
1          0.8772    Groupsock::output(UsageEnvironment&, unsigned char, unsigned char*,
unsigned int, DirectedNetInterface*)
1          0.8772    HandlerIterator::next()
1          0.8772    MPEG1or2AudioStreamParser::parse(unsigned int&)
1          0.8772    MPEG1or2Demux::getNextFrame(unsigned char, unsigned char*,
unsigned int, void (*)(void*, unsigned int, unsigned int, timeval, unsigned int), void*, void
(*)(void*), void*)
1          0.8772    MPEG1or2Demux::registerReadInterest(unsigned char, unsigned char*,
unsigned int, void (*)(void*, unsigned int, unsigned int, timeval, unsigned int), void*, void
(*)(void*), void*)
1          0.8772    MPEG1or2Demux::useSavedData(unsigned char, unsigned char*,
unsigned int, void (*)(void*, unsigned int, unsigned int, timeval, unsigned int), void*)
1          0.8772    MPEG1or2DemuxedElementaryStream::doGetNextFrame()
... ...
```

Fig 7: The result of Oprofile

As it can be seen, most CPU time resources is consumed in MPEG1or2VideoStreamParser :: parseSlice (). This function is mainly used for the slice parsing. We can conclude that when disk I/O request is not the bottleneck, CPU capacity is the main bottleneck and CPU is mainly used for media data analysis and packaging in this case.

To prove our conclusion, we make experiments in *tmpfs* to eliminate the bottleneck of disk I/O requests. We gradually increase the concurrent connections and the number of access file simultaneously to 50.

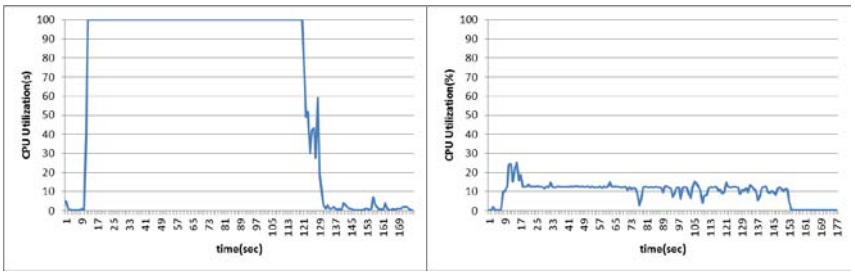

Fig 8: CPU utilization under the original CPU capacity (left) and 8 times of it (right) when the numbers of concurrent connections and of accessed files are both 50

When both the concurrent connections and the quantity of access file are 50, CPU usage change to 100%, and the quality of video observed by VLC player is not smooth. We consider that the server reaches its bottleneck. Then we increase the CPU capacity by 8 times, CPU utilization percent is only 20% and the viewing quality has improved significantly. Thus, we prove the conjecture.

In addition to the above experiments, we also did other experiments and further make the following conclusions:

1. When the concurrent connection is 20, along with the increase of flow rate (2 MBPS, 5 MBPS, 10 MBPS), the throughput increase linearly, and server processes more and more data. When the flow rate reach 10 Mbps, CPU idle is always 0 and becomes the bottleneck of the server.
2. Consumption of CPU is relatively lower and server performance is better when playing H.264 files than MPEG 2 files: CPU resources remain 80% available when playing H.264 file while MPEG 2 remain about 60%, both with 20 concurrent connections, 5MBPS flow rate, and 1 access file.
3. We make experiments to find out the difference between HTTP and RTSP. We use *lighttpd* as HTTP server. Compared with the RTSP, HTTP consumes less CPU resources. CPU Idle remain about 90% by HTTP while CPU run exhausted by RTSP in the same condition. According to experiment case 2, RTSP server needs to use CPU for data analysis and packaging beside transmission while HTTP mainly focus on transmission, so the performance of HTTP is better.

5 Summary

To find out the real bottlenecks of streaming media server, we changed the capacity of disk IO, CPU, flow rate and the bandwidth respectively. Experiment results and conclusions are summarized as follows:

1. When the connection is in the establishing state, CPU handles the establishing process and reads the media data into memory, and then analyses and packetizes the media data;
2. Disk I/O read-write capacity is the main bottleneck of streaming media server in many cases;
3. After improving the capacity of reading and writing, CPU processing capacity become the bottleneck;
4. When the bandwidth is limited, the quantity of concurrent connections is limited;
5. Efficient media coding method improves performance of streaming media server;
6. HTTP is simpler and faster than RTSP.

6 Acknowledgements

This work was supported partly by NSFC (National Natural Science Foundation of China) under grant No. 60970127 and the Program for New Century Excellent Talents in University (NCET-09-0709).

7 References

1 H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. "RTP: a transport protocol for real-time applications". RFC 1889, Internet Engineering Task Force, Jan. 1996.

2 Jinyao Yan; Muhlbauer, W.; Plattner, B., "Analytical Framework for Improving the Quality of Streaming Over TCP," Multimedia, IEEE Transactions on , vol.14, no.6, pp.1579,1590, Dec. 2012.

3 Jinyao Yan ; Katrinis, K. ; May, M. ; Plattner. Media- and TCP-friendly congestion control for scalable video streams. Multimedia, IEEE Transactions on , 2006:196- 206 .

4 Lee Y, Min O, Kim H. Performance evaluation technique of the RTSP based streaming server[C]. Computer and Information Science, 2005. Fourth Annual ACIS International Conference on , IEEE, 2005:414 - 417 .

5 M Chesire, A Wolman, G M Voelker, and H M Levy. "Measurement and Analysis of a Streaming-Media Workload". In USITS, 2001.