A Cluster-Based Active Router Architecture Supporting Video/Audio Stream Transcoding Service

Jiani Guo, Fang Chen and Laxmi Bhuyan
Computer Science and Engineering
University of California, Riverside, CA 92521
{*jiani,fchen,bhuyan*}*@cs.ucr.edu*

Raj Kumar
Hewlett-Packard Laboratory
raj_kumar@hp.com

Abstract

Active routers allow computation to be performed within the network by processing packets when they pass through the routers. We design and implement a cluster-based active router system that provides multimedia stream transcoding service. The performance of the system is evaluated with three different load balancing schemes. We evaluate the out-of-order phenomenon and analyze the tradeoff between this phenomenon and the processing speed. We present a stream-based round robin algorithm for the transcoding service offered in the router and demonstrate its superiority over the conventional round-robin scheme. The main design criteria are to minimize the total transcoding time and maintain the order of media units for each outgoing stream.

1. Introduction

The advent of World Wide Web has resulted in increasingly large-scale deployment of parallel and distributed computing systems. While limited network bandwidth is still the foremost cause of service degradation, the growing use of multimedia and e-commerce applications presents scalability concerns for processing requirements as well. Performing such kind of computation exclusively at the server end is difficult because of the overload on servers. It is also difficult to perform the necessary computation at the terminal end point, since it usually consists of less-expensive, specialized network appliances and thin-client terminal devices. The proxy model used by several commercial service providers only succeeds in moving the problem of scale from the server to the proxy. The active-networking offers a viable solution to this problem through parallel and distributed-computation models.

Active network architectures permit a massive increase of sophisticated computation within the network by allowing their users to inject customized programs into the nodes of the network. Broadly, there are two approaches to active networking [14]. One is that the network nodes are fully programmable and active packets carry all of the code that should be executed on them. The other approach is that network nodes are dynamically configured to provide different customizable services, and active packets use these services.

The transmission of multimedia information through networks has long been a research topic, and it is claimed that multimedia application is becoming one of the killer applications in this century. The ability to accommodate video/audio streams in the network extends the reach of TV type broadcasts into the broadband environment, and facilitates video/audio conferencing for collaboration and remote instruction. In most cases, the video/audio streams need to be transcoded into a form that satisfies the playback/recording requirements of various devices or users with different preferences before they reach the destination. Generally there are two reasons for this. First, the service provided may not suit users' need. For example, when a media server streams a high-bit-rate video/audio to low-bit-rate mobile clients, the video/audio should be transcoded into low-bit-rate to match the client's requirements. Second, the media stream should adapt to the variation of available bandwidth of the transmission channel, and transcoding can provide this kind of adaptation. However, transcoding is computation-intensive and needs to be executed fast. An algorithm for fast down-scale transcoding of compressed video is implemented on the new Itanium processor family [10]. It was shown that 30%-40% of computing power is saved while the quality of transcoding is maintained. In the active network, the task of transcoding video/audio streams is distributed over a number of processors, and allows better resource management both within the network and at the end hosts.

In the Journey network model [14], a cluster-based active router architecture, called Clara, is adopted to collocate routing and computational functionality. This model supports scalability by distributing computation among a sequence of active routers, but does not guarantee that the processing will be accomplished before the packet reaches the destination, which may introduce such problems as high packet out-of-order rate and high unprocessed packet rate.

In this paper, we present a cluster-based active router implementation that provides video/audio transcoding service. We assume that there is only one active router in the path of media stream, where transcoding is performed. There may be other passive routers on the path, as shown in Figure 1. In our model, the incoming video/audio streams are captured and transcoded before they leave the router. The active router consists of a routing node and a few computing nodes. The routing node and computing nodes in the cluster are organized as a distributed computing platform connected over a high-performance network. The routing node acts as the cluster manager as well as a normal IP router.

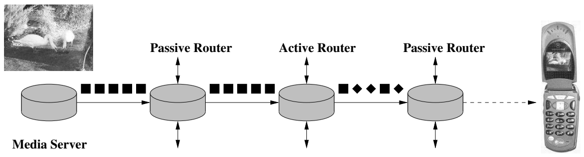

Figure 1. An active router in network

Three kinds of load sharing strategies, round robin, adaptive load sharing and stream-based round robin, are implemented in the system. Notice that although a plethora of adaptive load balancing strategies have been suggested in the literature [7], they are rarely used in practice. It is because of the difficulty in their implementation, the high overhead of measuring the parameters, and evaluating optimization functions. As a result, most of the implementations are based on simple round-robin schemes. In this paper, we identify and implement a particular adaptive load balancing scheme [6] for our active router. We also propose a new stream-based round-robin policy that is simple but greatly reduces the out-of-order degree of the media units in a simple unit-based round-robin scheme.

We did experiments to test the system performance of our implementation by varying the number of the Computing PCs, incoming multimedia streams, and load balancing policies. The main performance metrics are out-of-order rate of the out-flowing media data, departure rate of these media streams and scalability of the cluster. The paper makes the following contributions:

1. We implement an active transcoding router using five linux-based PCs connected over a Gigabit Ethernet.
2. We implement a RED-like algorithm for admission control, and two load balancing policies, round-robin and adaptive load sharing, for distributing tasks to the computing nodes. We also develop a new stream-based round robin algorithm and shows its superiority over the traditional unit-based round robin policy.
3. We carry out extensive performance evaluation of the active router through measurements. The results give insights into out-of-order properties of the video outputs and scalability of the parallel computation.

The paper is organized as follows: In section 2, we briefly discuss the related work. The framework of the cluster-based active router and some implementation issues are introduced in section 3. In section 4, we present various load balancing schemes. Experimental results are presented in section 5. Finally, section 6 concludes this paper.

2. Related work

In the active network research domain, the NetScript project [16] at Columbia University, the ANTS system [15] from MIT, and SwitchWare [1] from the University of Pennsylvania all aim toward the model where routers are fully programmable and active packets carry executable codes. For security reason, the execution context usually resides in a virtual machine, which makes it questionable for high performance.

The MeGa project [2] of the University of California, Berkeley, and the Journey network model [14] at the NEC-USA, fall into the second category, where routers provide customizable services according to packet requests. Ralph Keller et al. proposed an active router architecture [5], where video scaling algorithms are deployed to improve video performance. The difference between this model and the journey network model is that the former requires that all the scaling computation should be accomplished within the same active router, while the latter focuses on distributing computation to different routers even when the packets belong to the same stream.

Admission control is a crucial part in router queue management to enforce QoS policies, so that ill-behaved flows will experience higher packet drop rate. In contrast to static algorithms like Drop-Tail, Drop-Head, etc., active algorithms such as RED [3] pro-actively drop a packet with a variable probability when the average queue length reaches a threshold.

Load balancing is widely used in parallel and distributed systems. A detailed survey of general load balancing algorithms is provided in [12]. In the network domain, load balancing schemes are particularly adopted to split network service requests among a bunch of servers such as web servers, or distributed cache servers. When the concept of flow is involved, one of the most important properties of any load balancing policies is that the packets belonging to the same flow should be kept in order as much as possible. For example, in TCP-based flows, out-of-order packets may trigger retransmission or even worse congestion control, and thus degrade the throughput; in UDP-based flows such as video/audio transmission, limited receiving buffers may not accommodate the out-of-order packets, which results in higher drop rate and thus affects the quality of service.

In practice, simple static policies, such as random distribution policy [11] or modulus-based round robin policy [4], can achieve satisfactory results. However, the random distribution cannot preserve packet order within a flow if per-flow information is not maintained. Modulus-based round robin policy also has the drawback that all flows are re-mapped if the number of computing nodes is changed. On the other hand, adaptive load balancing policies are usually complicated and require prediction of computation time for any incoming requests [17]. The hash based highest random weight (HRW) algorithm was first proposed by Thaler [13], and then developed by Ross [9]. It is reported to provide good performance and low overhead when the request identifier space is evenly distributed. Lukas Kencl et al. HRW is extended with a feedback mechanism, which allows adjustment to the weights with minimum flow re-mapping[6], to cope with request identifier space locality. We incorporate this adaptive load balancing technique in our active router.

3. The framework

3.1. Active router cluster architecture

Figure 2 demonstrates the computation model of our active router system. When media units pass through the active router in the network, they are processed before their departure as long as the active router can provide enough computational resources. The proposed active router consists of a cluster of generic PCs. One PC, called routing PC, acts as the coordinator, as well as performs all router functionality. Other PCs, called computing PCs, provide customizable computation-intensive services. In this model, the computing PCs and the routing PC are connected through a Gigabit Ethernet. The routing PC is also connected to a remote PC, which continuously sends several media streams to it. A media stream is a complete video/audio segment, like a movie or a conference recording. The remote PC can be regarded as a simplified media server in the practical network.

We assume that the media stream data can be divided into a sequence of media units that are ready for independent transcoding. A media unit can be a group of pictures (GOP) of MPEG streams or a FRAME of AVI streams. The routing PC receives these media units from the remote PC, and forwards them to the computing PCs for transcoding. Each computing PC independently processes the media units using the local computing resource and does not require any global stream state. We do not transcode any units in the routing PC because the routing function will usually keep this PC busy. Since our experiments emphasize the parallel computing service offered in the cluster, we did not include layer 3 functions such as routing-table lookup in our cluster-based active router. Because of the limited resources available in the active router cluster, some packets in the media stream will be sent out without being processed. The aims of our implementation are to minimize the processing time for each media unit in the active router, and to preserve the unit order of outgoing media streams as much as possible. Many factors need to be considered under this circumstance.

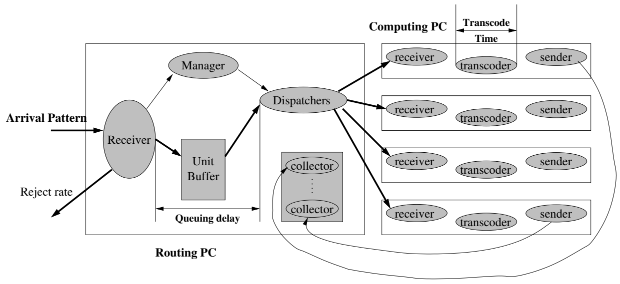

Figure 2. Computation Model of Active Router Cluster

- The first is the arrival pattern with which multiple media streams reach the routing PC from the remote PC. How to define and guarantee the fairness among streams needs to be addressed.
- The second is the admission policy that determines which packets from which streams should be admitted to ensure the fairness and QoS.
- The third issue is the load balancing policy that determines how to distribute the units to different computing PCs for transcoding. Since the order of output media stream units is largely affected by the dispatching policy, whether the units of the same media stream should be dispatched to different computing PCs has to be decided.

In the user-level multiprocess/multithreaded implementation of the active router, shown in Figure 2, media units are continuously sent from the remote PC to the routing PC according to the arrival pattern discussed in section 3.3. On the routing PC, four kinds of threads, namely, receiver, dispatcher, collector and manager, are running concurrently. Three kinds of processes, receiver, transcoder and sender, reside in the computing PC.

Routing PC

The *receiver* puts unprocessed units into the unit buffer according to the admission control policy described in section 3.2. Once a packet is admitted into the unit buffer, all packets of the same unit should be accepted and assembled before they can be distributed to a computing PC. When the admission control rejects the first packet of a unit, all packets of this unit are rejected, i.e., sent out to the outgoing path without being processed. The unit buffer adopts a simple FIFO policy for assembled units since the media streams simulated in our experiment are concurrent and treated with equal priority. The *dispatcher* fetches units from the unit buffer and dispatches them to computing PCs according to the load balancing policy. When all computing PCs are very busy, the accepted units will stay in the buffer until computing resources are available.

The *manager* collects the load statistics information and does load balancing only when feedback information is required by a load balancing policy. The load balancing strategies are further described in section 4.

We have a *collector* per computing PC to collect all the processed units from the computing PC.

Computing PC

Each computing PC has one *receiver*, multiple *transcoders* and one *sender*, which are running in pipeline. The unprocessed units received by the *receiver* are first stored in the buffer, and are then fetched and transcoded by one of the *transcoders*. Finally, the *sender* sends the processed units back to the router. The processing time per unit at the transcoding stage is much higher than that of the other two stages, hence multiple *transcoders* are adopted, but the number of *transcoder* processes should be optimized.

3.2. Admission control

Admission control determines whether an unprocessed media unit should be admitted into the unit buffer. Usually the algorithm will affect the fairness and quality of service.

One possible policy is that each unit carries a tolerable delay bound. The active router, by checking its own status, evaluates whether the bound can be satisfied. If possible, the video packet and all following packets in the same unit are accepted for transcoding; otherwise, they are rejected and hopefully processed by other active routers in the ongoing path. However, this requires that the router accurately predict the process time. In addition, the router has to assume that all packets of the same unit should arrive in time, or the interval does not exceed the delay bound. In our experiment, we only care about how many streams can be supported with a reasonable delay under the system configuration assuming all the streams have the same priority.

We define fairness as giving each stream similar acceptance rate (or rejection rate). Keeping this in mind, we propose a RED-like algorithm to achieve fairness over media streams. In the algorithm, we set two threshold values to the unit buffer. Whenever the average queue length exceeds the lower threshold, the incoming units are randomly rejected with a variable probability p that is proportional to the average queue length. If the average queue length exceeds the upper threshold, any incoming units are rejected; otherwise, all incoming units are accepted. In our experiment, the lower threshold and the upper threshold are set as 0.8 and 0.99 respectively. The maximum queue size is an adjustable parameter.

The RED idea adopted here is different from that of a regular router, where RED is used to drop packets and notify senders [3]. In our case, the packets are not dropped but rejected, i.e., sent out without processing. However, we expect that the future video streaming servers will have congestion-control response mechanism to count unprocessed packet rate. Another difference arises in that, all packets of a unit should be rejected once the first packet of the unit is rejected in our scheme, while RED drops packets without considering the correlations among packets.

3.3. Hardware Setup

We implement a simplified media server on the remote PC, which continuously sends media stream data to the active router using UDP. The media streams are short movies encoded in MPEG-1 format with the bit rate of 1441Kbps. Each GOP of the media stream consists of 15 frames for the playback time of 0.5 second, since the normal playback rate is 30 frames per second (fps). The average GOP size is around 90k. The media server sends streams in round robin fashion. For each stream, the media server splits its GOP into a series of packets, and sends all these packets within 0.5 seconds. Such a round robin scheme reflects the correct scenario where the active router receives video packets from multiple media severs simultaneously. Although the simple policy does not consider multi-layer encoding or stream error correction encoding used in some commercial applications, it is a reasonable assumption for our measurement purpose.

The transcoding service, provided by each computing PC, is derived from a powerful Linux video stream-processing tool implemented by Thomas Ostreich [8]. It can change video compression formats, change the playback bit-rate and even adjust the frame resolution by chopping off some frame regions. Its current implementation is based on a multi-process and multi-thread model. Because the whole program runs in the user space, invoking the transcoding process consists of many context switches and thus affects the performance. In addition, the interface to this service is through files that involves disk I/O operations.

The remote PC, routing PC and computing PCs are equipped with Athlon 1.4G CPU and 1GB memory; the remote PC connects with the routing PC through 100M Ethernet, while the routing PC and computing PCs are connected through a Gigabit switch. The operating systems is Linux Mandrake 7.2.

4. Load balancing

A critical issue in implementing this cluster system is the load balancing strategy, i.e., how to distribute multiple media streams among different computing PCs to achieve the best utilization. A unit based round robin scheduling is adopted in [14]. Using this scheme, we observe that the performance is poor in terms of out-of-order degree for processed units. Hence, we propose to implement another two algorithms and evaluate them in the next section.

4.1. Round Robin Strategy

With round robin, the *dispatcher* searches for an available computing PC in fixed order. When multiple *dispatchers* are used, concurrently sending different streams to different computing PCs is allowed. In this case, once a *dispatcher* starts to send a media unit to a computing PC, the computing PC will be labeled "unavailable"; and when the data communication is completed, this computing PC will be labeled "available" again and can be chosen by another dispatcher. Notice that in this policy, the units of the same flow are most likely to be distributed to different computing nodes, and thus the order is not preserved. However this scheme is efficient when only processing speed is concerned.

4.2. Stream-based Round Robin Policy

To preserve the order of computation among media units, as well as keep the simplicity of round robin, we propose and implement a stream-based round robin algorithm. The unit is mapped to a computing PC according to the following function:

$$f(C) = C \bmod N \tag{1}$$

where, C is the stream number to which the unit belongs; and N is the number of computing PCs in the cluster. Therefore, all the units belonging to one stream will be sent to the same computing PC. Concurrently running dispatchers are allowed to dispatch different streams to different computing PCs simultaneously. Once a dispatcher thread fetches one unit from the unit buffer, it first checks if the mapped computing PC is available and decides what to do. The unit will be dispatched immediately if the PC is available; otherwise, the dispatcher will go back to the unit buffer to fetch another unit.

4.3. Adaptive Load Sharing Policy

Although a number of adaptive load sharing policies are proposed in the literature[12], we found that the extended HRW technique[6] is suitable for network applications containing a number of flows. Hence, we implemented it in our system. Since the studies [6] are theoretical in nature, we have to carefully assign the parameters.

According to the adaptive load sharing policy proposed in [6], a packet can be mapped to a particular computing PC according to the function $f(\vec{v}) = j$, which is defined as

$$x_j \; g(\vec{v}, j) = \max_{k \in \{1, \ldots, N\}} x_k \; g(\vec{v}, k) \tag{2}$$

where, v is the identifier vector of the packet; j is the computing PC to which the packet will be mapped for processing; $g(\vec{v}, j)$ is a peudo-random function which produces random variables in (0,1) with uniform distribution; and $x_0, x_1, ...x_N$ are the weights for all the N computing PCs. Concerning the weights which describe the processing capacity of each computing PC, [6] also proposed a dynamic adaptation through feedback. The routing PC gathers information about the utilization of the computing PCs. If an adaptation threshhold is exceeded, the routing PC adjusts the weights. A smoothed, low-pass filtered processor utilization measure of the following form is used to calculate the utilization of each computing PC $\bar{\rho}_j(t)$ by gathering the load statistics information $\rho_j(t)$ at fixed time interval.

$$\bar{\rho}_j(t) \;=\; \frac{1}{r}\,\rho_j(t) + \frac{r-1}{r}\,\bar{\rho}_j(t - \Delta t) \tag{3}$$

The total system utilization is measured as $\bar{\rho}(t) = \frac{1}{r}\,\rho(t) + \frac{r-1}{r}\,\bar{\rho}(t - \Delta t)$. The adaptation algorithm consists of triggering policy and adaptation policy. Once the triggering condition is reached, adaptation will be taken to the weights of involved computing PCs.

To implement the algorithm, [6] suggests to implement function $g(\vec{v}, j)$ by using the hash function $h_{\phi^{-1}}(y) = (\phi^{-1}y) \bmod 1$, which is based on the Fibonacci golden ratio multiplier $\phi^{-1} = (\sqrt{5} - 1)/2$, such that,

$$g(\vec{v}, j) = h_{\phi^{-1}}(\vec{v} \text{ XOR } h_{\phi^{-1}}(j)) \tag{4}$$

Another open implementation issue is how to actually measure the load of each processor.

In our experiments, we adopt the above function $g(\vec{v}, j)$, and define the load indicator $\rho_j(t)$ as

$$\rho_j(t) = ttask_j / \Delta t \tag{5}$$

where, $ttask_j$ is the CPU time spent by the transcoding services during the polling interval Δt. $\rho(t)$ is defined as

$$\rho(t) = (\sum_{i=0}^{N} ttask_i)/N\Delta t = \frac{1}{N}\sum_{i=0}^{N}\rho_i(t) \tag{6}$$

The identifier vector v is chosen to be the stream number to which the units belong. So, once a dispatcher fetches a unit from the unit buffer, the function f is calculated to determine a specific computing PC to which the unit should be dispatched.

In summary, with the adaptive load sharing strategy, the system works as follows: all the Computing PCs are assigned the same weights at the beginning. Then, the *manager* polls all the computing PCs to collect load statistics information at a fixed time interval of 3 seconds. The collected load statistics information is calculated to determine if adaptation of the weights is triggered. If adaptation is necessary, the weights of adapted computing PCs are modified and so the packet to computing PC mapping may change to better balance the loads.

5. Experimental results

5.1. Performance metrics

To measure the performance of our cluster-based active router, several performance metrics are defined below:

Metric 1: Out-of-order (OFO) *degree* is used to compare the output order of media units in a given video stream with respect to their input order. To describe the OFO degree, we examine the statistical distribution of out-of-order units in each stream. The out-of-order is caused by several reasons: first, multiple *dispatchers* may fetch the units of the same stream out of order; second, different media units consume different computation time; third, several concurrently running *transcoders* may transcode the units of the same stream out of order; lastly, when round robin policy is adopted, the units belonging to the same stream are not guaranteed to be dispatched to the same computing PC.

Metric 2: Output time interval among successive media units of a media stream (OTI per stream) is defined to describe how fast the transcoded media stream can be output by the router. It is different from the average processing time per unit in that it describes the actual output time interval between two successive units belonging to the same stream.

Metric 3: Total processing time is defined as the total time spent in the active router from receiving the first packet of all the streams to collecting the last packet of all the streams from the computing nodes.

Metric 4: Average processing time per unit is calculated by dividing the total processing time by the total number of the media units transcoded in the active router. It depicts how fast the active router can process one media unit.

5.2. OFO Degree and OTI per Stream

We start the measurements with OFO degree of the media units. The in-order property is the most valuable performance measure to preserve the video quality. There are two kinds of out-of-order situations: "n preceding" means a media unit jumps n units ahead of its original order; "n lagging" means a media unit lags n units behind its original order.

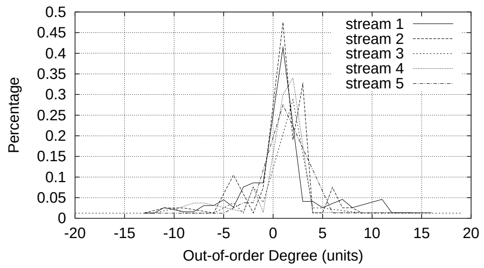

Figure 3. OFO Degree (round robin)

Figure 3 depicts the OFO degree when the experiment uses 3 computing PCs, 5 media streams, 700 units, buffer of 200 units, and *round robin policy*. The X-axis indicates the OFO degree: *-n* means "n lagging", and *n* means "n preceding". Specifically, 0 means that the unit is in order. The percentage of in-order units is not shown in the figure, since we only consider out-of-order units here. Using round robin policy, all streams have OFO degree between -15 and 15. Most of the OFO units fall between -5 and 5. As we can see, the OFO degrees with the highest percentage of all 5 streams are 1 (42%), 1 (47%), 2 (34%), 1 (27%) and 2 (28%), respectively.

Figure 4 demonstrates the statistical distribution of OTI per stream. The graph shows that *n*% (Y-axis) of the units of the stream are output within *m* (X-axis) seconds successively. All the units are classified into 5 categories: the OTI per stream is 0.5 second, 1 second, 1.5 second, 2 second, and 2.5 second, respectively. As shown in Figure 4, three streams have most of their units processed with the OTI of 0.5 second; while the other two streams have most of their units processed with the OTI of 1 second.

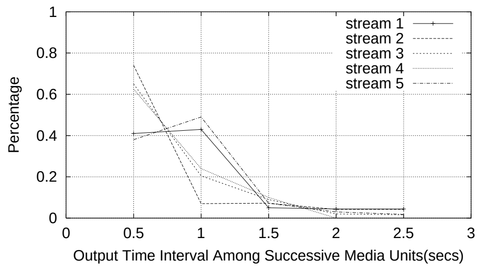

Figure 4. OTI per Stream (round robin)

Figure 5 demonstrates the OFO degree for the same experiment with *adaptive load sharing policy*. The performance is much better than the round robin. Three of the streams barely have out-of-order units. For the other two streams, at most 14% of their units have the OFO degree of 1; and only 1% of their units have the OFO degree greater

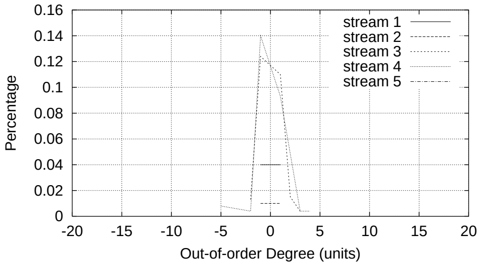

Figure 5. OFO Degree (adaptive load sharing)

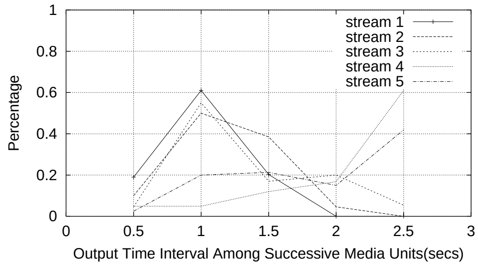

Figure 8. OTI per stream (Stream-based round robin)

than 1. The measurement of OTI per stream is described in Figure 6. On average, only roughly 20% units can be output within 0.5 second following each other. Compared with round robin, adaptive load sharing gives much better output order of media units per stream, but its overhead of collecting load statistic information causes the OTI per stream to degrade.

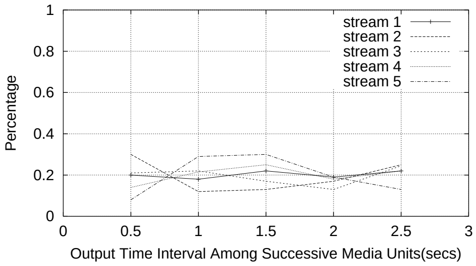

Figure 6. OTI per stream(adaptive load sharing)

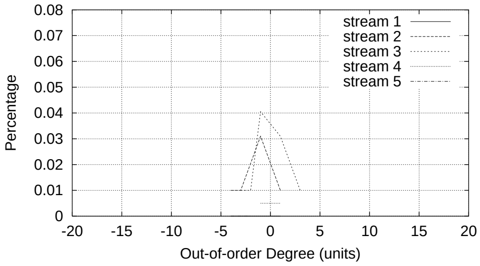

Figure 7. OFO degree (Stream-based round robin)

To solve this problem, we propose the stream-based round robin described in section 4.2. The results are shown in Figures 7 and 8. It can be observed that the order is preserved well and the output interval has been improved compared with adaptive load sharing. This is because the stream-based round robin has less computation overhead and no communication overhead for load balancing.

5.3. Processing Time

To evaluate the total processing time, we use 5 media streams, 50 units per stream, buffer of 200 units. With the RED-like admission policy, about 10% of the total units are rejected when only one computing PC is used in the cluster, because the units cannot be processed on the computing PC fast enough and thus the number of units waiting in the queue passes the threshold. As shown in Figure 9 , with round robin(RR), the total processing time scales well when the number of computing PCs increases from 1 to 4. On the other hand, both adaptive load sharing(ALS) and stream-based round robin(SRR) consume more processing time than RR because of their overhead to maintain the flow consistency. The ALS needs to collect load statistics and calculate the triggering and adaptation function,while the SRR needs to calculate the mapping function. Besides, they both distribute incoming traffic according to streams, which makes the system less efficient when the stream number is not a multiple of the cluster size.

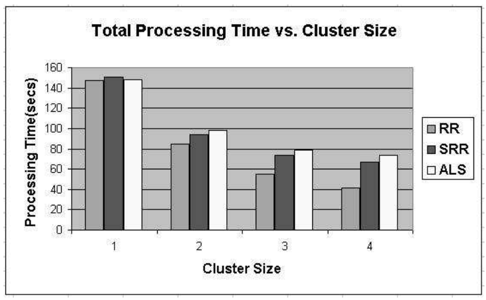

Figure 9. Total Processing Time

Finally, we use 8 media streams and plot the results in Figure 10. The average per unit processing time scales best with RR, and then SRR, with ALS ranking the last. SRR scales worse than RR when the cluster size increases from 2 to 3; but performs as well as RR when the cluster further goes up to 4. With ALS, the system shows the same scalability trend as SRR, except that the overhead costs ALS more and more time when the cluster size becomes bigger.

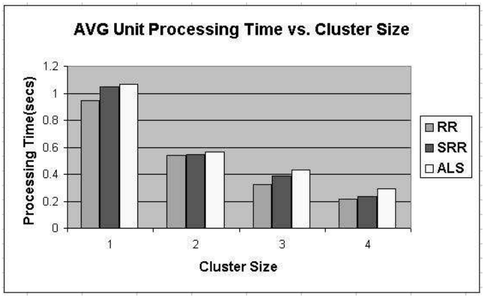

Figure 10. AVG Unit Processing Time

6. Conclusion

Active network provides a new direction for processing network traffic, which makes it possible to distribute expensive computation among a number of active routers instead of executing it on the server or client. This is especially useful in processing multimedia traffic like video or audio streams. However, this model does not guarantee that the media unit will be processed before it reaches its destination, and the computation distribution has an inherent out-of-order problem, which in return will affect the overall performance and QoS.

In this paper, we designed and implemented a cluster based active router architecture and evaluated its performance through real MPEG stream data. We found the architecture scales well with the cluster size when accompanied with adequate admission control and load distribution policies. To keep better unit order of the outgoing media streams, we proposed to adopt a stream-based round robin instead of the unit-based round robin load balancing scheme. We also implemented an adaptive load sharing scheme from the literature. Experimental results showed that the stream-based round robin outperforms the adaptive load sharing by reducing the overhead to maintain flow consistency. However, the inherent out-of-order problem can still severely affect the QoS of the received stream, and might not be solved without coordination among successive active routers. In our future research, we plan to address the problems above and try to find performance optimizations to this router architecture.

References

[1] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The switchware active network architecture. *IEEE Network*, vol. 12, May/June 1998.

[2] E. Amir, S. McCanne, and R. Katz. An active service framework and its application to real-time multimedia transcoding. *ACM SIGCOMM Symp.*, September 1998.

[3] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACMTransactions on Networking*, 1(4), August 1993.

[4] E. Katz, M. Butler, and R. McGrath. A scalable http server: The ncsa prototype. *Computer Networks and ISDN systems*, 27:155–164, 1994.

[5] R. Keller, S. Choi, M. Dasen, D. Decasper, G. Fankhauser, and B. Platter. An active router architecture for multicast video distribution. *IEEE INFOCOM*, 2000.

[6] L. Kencl and J. Y. L. Boudec. Adaptive load sharing for network processors. *IEEE INFOCOM*, 2002.

[7] N. Ni and L. N. Bhuyan. Fair scheduling and buffer management in internet routers. *Proc. INFOCOM*, June 2002.

[8] T. Ostreich. Linux video stream processing. http://www.theorie.physik.uni-goettingen.de/˜ostreich/transcode/.

[9] K. W. Ross. Hash routing for collections of shared web caches. *IEEE Network*, 11(6), November-December 1997.

[10] S. Roy and B. Shen. Implementation of an algorithm for fast down-scale transcoding of compressed video on the itanium. *Proceeding of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2002.

[11] M. Satyanarayanan. Scalable, secure, and highly available distributed file access. *IEEE Computer*, May 1990.

[12] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. Scheduling and load balancing in parallel and distributed systems. *IEEE CS Press*, 1995.

[13] D. G. Thaler and C. C. Ravishankar. Using name-based mappings to increase hit rates. *IEEE/ACM Transactions on networking*, vol. 6:10–27, November-December 1997.

[14] G. Welling, M. Ott, and S. Mathur. A cluster-based active router architecture. *IEEE Micro*, 21(1), January/February 2001.

[15] D. J. Wetherall, J. V. Guttag, and D. L. Tennenhouse. Ants: A toolkit for building aned dynamically deploying network protocols. San Francisco, CA, April 1998.

[16] Y. Yemini and S. da Silva. Towards programmable networks. L'Asquila, Italy, October 1996.

[17] H. Zhu, T. Yang, Q. Zheng, D. Watson, O. Ibarra, and T. Smith. Adaptive load sharing for clustered digital library servers. *Proceedings of the seventh International Symposium on High Performance Distributed Computing*, pages 235–242, 1998.