PAPER *Special Issue on Content Delivery Networks*

Stream Caching Using Hierarchically Distributed Proxies with Adaptive Segments Assignment

Zhou SU[†a)], ***Student Member***, **Jiro KATTO**[†], ***Regular Member***, **Takayuki NISHIKAWA**[†], **Munetsugu MURAKAMI**[†], ***Student Members***, ***and*** **Yasuhiko YASUDA**[†], ***Honorary Member***

SUMMARY With the advance of high-speed network technologies, availability and popularity of streaming media contents over the Internet has grown rapidly in recent years. Because of their distinct statistical properties and user viewing patterns, traditional delivery and caching schemes for normal web objects such as HTML files or images can not be efficiently applied to streaming media such as audio and video. In this paper, we therefore propose an integrated caching scheme for streaming media with segment-based caching and hierarchically distributed proxies. Firstly, each stream is divided into segments and their caching algorithms are considered to determine how to distribute the segments into different level proxies efficiently. Secondly, by introducing two kinds of segment priorities, segment replacing algorithms are proposed to determine which stream and which segments should be replaced when the cache is full. Finally, a Web-friendly caching scheme is proposed to integrate the streaming caching with the conventional caching of normal web objects. Performance of the proposed algorithms is verified by carrying out simulations.

key words: *stream caching, web caching, content delivery networks, hierarchically distributed proxies*

1. Introduction

With the growth in popularity of the Internet and the wide availability of high-speed networks, an increasing number of streaming media objects are being distributed over the Internet.

How to efficiently distribute stored information has become a major concern in the Internet community. More recently, some content distribution networks companies [3], [4] have emerged and they work directly with content providers to cache and replicate the providers' content close to the end users. However, web caching and delivery mechanisms of web objects such as HTML files or images can't be efficiently used for streaming media such as video and audio.

Streaming media has several inherent properties: Firstly, the size of streaming media is usually larger than non-streaming files by orders of magnitude [5]. Secondly, user access behavior shows different characteristics. For example, clients sometimes stop one stream without watching all of the parts [6]. Thirdly, in contrast to other web objects, streaming media do not require to be delivered at once. Instead, the server usually pipelines the data to clients through the network.

In this paper, we therefore propose an integrated delivery and caching system for streaming media, where each stream is divided into segments and these segments are distributed among hierarchically distributed cache servers.

Firstly, because storing the entire stream in a single proxy cache is inefficient or even impossible due to its large size, different segment-based caching algorithms are proposed and compared. A part of the requested stream is cached in a local cache, and the remainder of the stream will be cached in an upper proxy cache. When another client requests the stream again, the system will send the segments in the local cache and request the remaining segments from the upper proxies (maybe from the origin server).

Secondly, because each different stream has a different popularity and each segment has different access patterns, two different segment replacement algorithms are also proposed and compared. By convention, the same popularity was assigned to the whole stream when the classical replacing algorithm was carried out. In this paper, however, two priorities for each segment are introduced: one reflects its access property, and the other represents its position information in the stream. Two kinds of replacing algorithms are compared to decide which segments of which streams should be removed when the cache exceeds its limit. One of the two algorithms keeps the same relative length of each stream in the cache, while the other keeps the most accessed segments in the cache.

Thirdly, how to coordinate the streaming caching with the current caching scheme for normal web objects such as HTML files or images is considered. By introducing the EWMA (Exponential Weighted Moving Average) estimator, a Web-friendly caching scheme, which can improve the performance of the whole caching system, is proposed.

This paper is organized as follows: In Sect. 2, related works with regard to stream caching algorithms are reviewed. In Sect. 3, an overview of the proposed system architecture is provided, that can be applied to state of the art of content delivery networks. Section 4 presents our proposed algorithms for segment

Manuscript received September 9, 2002.
Manuscript revised February 18, 2003.

[†]The authors are with the School of Science and Engineering, Waseda University, Tokyo, 169-0072 Japan.
a) E-mail: suzhou@yasuda.comm.waseda.ac.jp

caching, segment replacing and Web-friendly caching, respectively. Simulation results are given in Sect. 5 and conclusions are shown in Sect. 6.

2. Previous Work

There are numerous works on replacing algorithms for normal web objects such as HTML files and images [7]–[13]. However, detailed behavior or simulation results of these algorithms have not been studied or shown for streaming media such as audio and video.

How to efficiently distribute content has atracted much research. In peer to peer networks, users can determine from where different files can be downloaded with the help of a directory service [7], [11]. Similar ideas are expanded as the Overlay network [25], where each connection in the overlay is mapped onto a path in the underlying physical network.

Content delivery networks (CDNs) appeared recently and are deploying quite rapidly [1]–[4]. Load balancing by request routing, efficient content delivery by locating edge servers near to clients and information exchange protocols among different CDN sites are developed. However, their concern is mainly placed on efficient delivery of static content, i.e. HTML files and images. Some CDN companies advocate their streaming caching support, but their technical details are not yet clarified nor verified.

There have been some published studies on stream caching. Sen et al. [17] showed that caching a prefix (i.e., the initial part) of a stream at the proxy can hide the potentially large initial start-up delay of the work-ahead transmission schedule from the client. Lee et al. [18] proposed a scheme that provides users with the video summary (i.e. a number of key-frame images) before they download the stream files. Recently, Kangasharju et al. [16] and R. Rejaie et al. [28] studied distributing layered encoded video through caches. M. Sasabe et al. [24] also discussed how to cache MPEG-2 video with a goal of video quality adjustment. However, the first two are immature from the viewpoint of complete systems. The others are also limited in a sense that a video frame has to be encoded into multiple layers instead of a group of video frames, i.e. segments, which characterizes our work. Note also that the key frame approach can be incorporated into our approaches.

We previously proposed a novel stream caching method using hierarchically distributed proxies [22]. We also proposed a "Web-friendly" stream caching scheme in which each stream is divided into segments and each segment is efficiently cached and replaced in order not to remove popular Web (static) contents [23], [29]. This paper is therefore characterized as an extended version of these papers, in which segment handling strategy (i.e. segment caching and segment replacement) is improved.

3. System Architecture

In this section, we will introduce an overview of our system architecture, which can be also applied to content delivery networks. Figure 1 depicts the system structure of hierarchically distributed and cooperative cache servers.

In the proposed system, we assume that each streaming media is divided into segments based on the result of shot boundary detection. In our definition, the segment represents a sequence of frames that can be decoded by itself. For the MPEG compressed data, for example, it consists of a group-of-pictures (GOP) or multiple GOPs. By definition, the GOP starts from an I-picture followed by P-pictures and B-pictures. This structure guarantees self-decodability of the segment and the I-pictures are also utilized as a key frame for video indexing. We also assume that each segment (GOPs) has the same size although this is not an actual case (due to variable rate characteristics of video compression and the fact that an intelligent encoder may detect scene change and generate a new I-frame as a GOP boundary). However, average GOP sizes are almost constant and the effectiveness of our segment-based caching still holds.

Recent studies [14], [15] have shown that a number of proxies cooperating and sharing contents with one another can improve cache hit ratio. Therefore, distribution of the segments of streaming media among multiple proxies is also expected to result in performance improvement of stream caching as long as cache sharing and cooperative caching are done efficiently.

Our proposed caching system consists of an origin server, upper proxies and local proxies. As shown in Fig. 1, local proxies are directly connected with clients (or are located most near to clients). When a client sends a request to the local proxy and the requested data can't be provided, the local proxy sends an ICP-like query message to other local proxies to ask whether

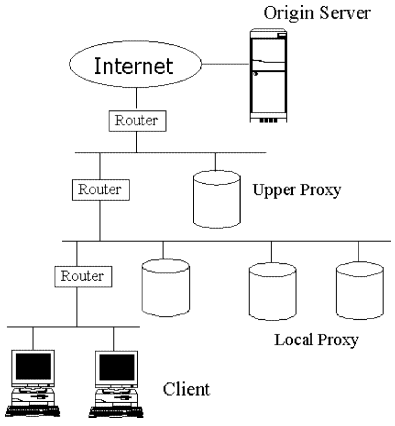


Fig. 1 Distributed cooperative caching system.

they have the requested contents.If the requested content can't be found in any local proxies, this request will be sent to upper proxies. If the requested content is found in the upper proxies, they return it to the client. If the requested content is not yet in the upper proxies, the request is finally sent to the originating server and the requested data is returned.

Note that our proposed architecture can be directly applied to the content delivery networks. Local proxies are "edge servers" Some of the upper proxies and local proxies can be grouped and a CDN site is composed. The original server may be expanded to multiple origin servers (i.e. original site). Request routing methods, such as DNS redirection and URL rewriting, are also effective in our system. We want to emphasize that our formulation described below is also effective to any type of CDNs.

4. Proposed Algorithms

To improve the performance of stream caching, stream sequence segmentation and hierarchical distribution of cooperative cache servers are introduced in this paper. Problems to be solved are as follows.

1. Conventional methods have to store the entire data of each stream in a local proxy or in a higher level proxy (maybe in an origin server). As the length of the streaming data is very long, careful consideration has to be given about what kind of segments are defined and how they are distributed among different level proxies.

2. As cache capacity is limited, efficient use of the cache is very important. As we have already discussed, different streams have very different popularities. Furthermore, different parts (positions) inside a stream have different requested times. Therefore, how to select clients' favorite segments from all segments of all streams has to be discussed to optimize cache performance.

The former is discussed in Sect. 4.1 entitled "Segment Caching," and the latter is discussed in Sect. 4.2 entitled "Segment Replacing" In addition, "Web-friendly" stream caching scheme is referred to in Sect. 4.3.

4.1 Segment Caching

In conventional methods, the entire data of one stream is either completely cached in a local cache or not. There are some disadvantages in these approaches. Firstly, the cache has to purge some currently cached data to make room for the incoming stream to satisfy current request. Because of its large size, many streams or normal web objects in the cache will be removed even some of them are frequently requested by many clients. Secondly, sometimes not all the requested streams will be completely observed by clients. Caching the entire stream is not efficient in this case.

Therefore, by taking account of advantage of cooperative caches, we suggest only caching a part of the requested stream in a local cache and putting the rest of the stream in a higher-level proxy (upper cache). A problem is how to decide the caching quantity in a local cache. For this purpose, three schemes are proposed as follows.

Let M be the number of streams in an origin server, and L_i be the length of the stream i ($i = 1, 2, \ldots, M$). Let U_{seg} denote the size of one segment, and B_i be the number of segments of the stream i. For simplicity, we assume that every segment has the same size.

We also introduce a vector of cache indicators $\boldsymbol{C} = (C_1, C_2, \ldots, C_M)$, with $0 \leq C_i \leq B_i$. Each C_i is set to n if n segments of the stream i is cached in a cache. Note that $C_i = 0$ means that there is no segment of the stream i in a cache. With this notation, cache space occupied by cached segments, $G(\boldsymbol{C})$, is given by

$$G(\boldsymbol{C}) = \sum_{i=1}^{M} C_i \times U_{seg} \tag{1}$$

- *Last Watch*: Assume that the total number of the requests for stream i is Q during a fixed period, and each time (the q-th time, $q \in (1, Q)$) the client stopped watching stream i after the playback of segment $S_{q,i}$ ($i \in (1, M)$, $S_{q,i} \in (1, B_i)$), where $S_{q,i}$ means that the client finished watching total $S_{q,i}$ segments of stream i when he requested this stream at the q-th time. In this approach, each local cache keeps the parameter $S_{Q,i}$, which is a value of $S_{q,i}$ for the last access (the Q-th access). When a new request for this stream i occurs next time ($(Q+1)$-th time), the Last Watch approach caches only the segments before the position ($S_{Q,i}$) in a local cache. The remaining segments are not cached by the local cache and will stay in an upper cache.
- *Average Watch*: Different from the Last Watch, the Average Watch method keeps a series of the parameters $S_{q,i}$ for the past Q accesses. When a new request for the stream i occurs at the $(Q+1)$-th times, the average watch approach caches only the segments before the position ($\frac{1}{Q}\sum_{q=1}^{Q} S_{q,i}$) in a local cache. The remaining segments are cached in an upper cache.
- *Judging*: Different from the above two approaches that only use the position information of the requested segment itself, we define an algorithm that will compare the requested stream with other streams in the cache to judge whether caching a new segment is effective for improvement of the cache performance or not. Here, $SP_{request}$ represents a priority of the segment of a requested stream, and SP_{remove} does a priority of the segment that needs to be removed from a local cache. If $SP_{request}$ is higher than SP_{remove}, the segment of the requested stream is cached. If

not, this segment is not cached on the contrary. The details of how to determine and assign these priority values will be discussed in Sect. 4.2.1.

Note that the *Judging* itself does nothing about how to select a segment to be removed from the cache, which is done by the replacing algorithm discussed later. The *Judging* only compares $SP_{request}$ with SP_{remove} of which values are shared with the replacing algorithm, then decides whether to keep it in the cache or not. Therefore, it is a caching algorithm which is comparable with the other two ones, *Last Watch*, and *Average Watch*. The performance of these caching algorithms will be shown in Sect. 5.2.1.

4.2 Segment Replacing

In this subsection, some segment priorities are firstly defined in Sect. 4.2.1, and then the proposed segment replacing algorithms are introduced in Sect. 4.2.2.

When a new object comes in and the cache exceeds its limit, one or more objects in the cache must be removed in order to make room for the newly accessed object. The replacing algorithm is to decide which object should be removed.

By convention, if one stream is selected to be removed, it will be entirely purged from the cache. For large streaming objects, algorithms for replacing streams segment by segment in a careful manner have to be defined.

4.2.1 Segment Priority

Well-known conventional methods to decide object priority for replacing algorithms are *LRU* and *LFU*.

LRU is the most commonly used priority in replacing algorithms nowadays. The one least recently used will be removed from the cache when the cache is full. However, researches showed that one of the main weaknesses of this *LRU* is that the cache would be flooded by objects referenced only once, flushing out documents with higher probability of being reused.

LFU is another commonly used one, which removes the object with the lowest request times. However, as the popularity changes over time, the object owning high request times in a past period is not absolutely popular now.

Besides, most research concerns normal web objects. How to efficiently define priority for streaming media has not been proposed.

(1) Priority Index of Access (*PIA*):
Through theoretical analysis, we firstly propose a priority index of access (*PIA*) for streaming media. Assume that stream i has been requested by Q times within a fixed period and each time the client stopped watching stream i after the playback of segment $S_{q,i}(q \in (1,Q),\ i \in (1,M),\ S \in (1,B_i))$ as shown before. Priority Index of the j-th segment of stream i is then defined as follows:

$$PIA(i,j) = |\{S_{q,i}\,|j \le S_{q,i}\}|\, v_i/(T - T_{c,i}) \qquad (2)$$

PIA in Eq. (2) is defined as a priority index to dynamically reflect the access property of the segment, while other conventional methods are only related to the whole stream. Besides, it also takes both access times and access period into consideration resulting in keeping up with the current access pattern, which avoids storing a lost-popularity-one in the cache for too long time.
Equation (2) is a method to calculate PIA fast. Here $|.|$ denotes the number of elements of a set. T and $T_{c,i}$ are respectively the present time and the time when stream i is cached in a cache. v_i denotes the request times between $T_{c,i}$ and T. From Eq. (2), by using $S_{q,i}(q \in (1,Q))$, v_i and $T_{c,i}$, PIA can be obtained without keeping the numerous request records for all segments of all streams. The details will be introduced in Appendix A.
(2) Priority Index of Relative Length (*PIRL*):
As we mentioned before, not only do different streaming media have different popularities, but also access varies by position of each segment inside the stream. By taking this fact into account, another parallel priority for each segment is defined. We firstly define a relative length l_{rela}, which is a ratio of the serial number of this segment to the number of total segments in this stream. Then, assume that the total number of segments of stream i is B_i and the current vector of cache indicators $C = (C_1, C_2, \ldots, C_i, \ldots, C_M)$ as before, which means that there are C_i segments of stream i in a cache. Let the serial number of the beginning segment be one, then, the last segment of this stream has a relative length:

$$l_{rela}(i,j) = C_i/B_i \qquad (3)$$

We call this $l_{rela}$the priority index of relative length ($PIRL$).

4.2.2 Replacing Algorithms

Based on the two segment priorities defined before, two kinds of replacing algorithms are proposed:

(1) Popularity Policy: Firstly, the cache selects the segments with the lowest *PIA*. Then, if more than one segment have the same lowest *PIA*s, the one with the highest *PIRL* will be removed.
(2) Balance Policy: Firstly, the cache selects the segments with the highest *PIRL*. Then, if more than one segment have the same *PIRL*s, the one with the lowest *PIA* will be removed.

Here, we explain more about the Balance Policy algorithm in detail. Research [17] showed that keeping

the first few segments of one stream in a cache could reduce start-up latency. The reason is: if the beginning part of one stream is available, the remainder of the stream can be sent to the client during the playback time of the beginning part. Because the Balance Policy tends to keep the same relative length of different streams in a cache, it will clearly be good for cache performance. Furthermore, if the end segments of two streams have the same l_{rela}, the one with the largest stream size will be removed from the cache. This makes more cache space available for an incoming object.

A disadvantage of the Balance Policy is a case where the beginning segment of one stream will remain in a cache forever even if it has not been requested for a long time. To solve this problem, we introduce a 'zero check' strategy, which means that streams whose recent request frequency is zero are removed from a cache.

4.3 Web-Friendly Streaming Caching Scheme

It can be expected that an increasing number of streaming media will be distributed over the Internet with its further development. Since more cache space is likely to be occupied by the streaming media, a cooperative caching method with normal web objects such as images and HTML files should be discussed (otherwise, large streaming data causes removal of popular Web (static) contents). Large-scale CDNs may prepare multiple servers for Web objects (HTTP server) and streaming ones (e.g. RTSP server) separately, it costs much and a single server approach sharing a cache space is still preferable for small-scale CDNs and P2Ps. Therefore, we introduce here novel "Web-friendly" caching (WFC) schemes.

In our proposal, a portion of cache capacity is prepared for normal web objects according to clients' access conditions. Firstly, the proxy cache detects clients' access conditions and calculates a proportion of cache capacity being taken by streams (E_{stream}). Secondly, if the next request for a streaming object comes and E_{stream} exceeds a fixed threshold (Th), the cache will remove streaming media instead of normal web objects in order to make room for the newly requested object.

In detail, the EWMA (Exponential Weighted Moving Average) estimator is introduced as follows:

$$E_{\text{avr}}(t) = \omega \times E_{\text{stream}} + (1 - \omega) \times E_{\text{avr}}(t-1) \quad (4)$$

Where E_{stream} is the proportion of the cache capacity being taken by streams and $E_{\text{avr}}(t)$ is the averaged proportion at time t, and ω is a smoothing constant, respectively. When a cache is full and the cached data need to be removed to make room for the newly requested object, if $E_{\text{avr}}(t)$ of Eq. (4) exceeds the threshold (Th), the cache will select streaming data to be removed according to the algorithm in Sect. 4.2.

In addition, note that this scheme can control the portion parameter adaptively according to clients' preferences for normal web objects and streaming objects.

5. Evaluation of Algorithms

In this section numerical results will be presented by simulation experiments to validate various proposed algorithms.

5.1 Simulation Conditions

In simulation experiments, we assume following conditions. There are 1000 different streams with the rate of 64 kbps. The length of each stream is uniformly distributed between one minute and ten minutes [5] and the size of one segment is 48 kbyte. The position where a client stops watching a stream is decided at random. The request distribution yields to a Zipf distribution with a Zipf parameter of 0.8 [19]–[21]. The distributed cooperative system is made up of one origin server, one upper proxy and four local caches. The capacity of each local cache is 3% of the total size of all streams in the origin server. The capacity of the upper cache is double the capacity of the local cache. The total request times in the simulations are 20000 and the buffer size of a client is 240 kbyte. We should note that, as results of auxiliary experiments, different parameters did not drastically affect the performance orders described later.

Hit-ratio and Byte hit-ratio [10] have been popularly used to evaluate the cache performance by most researchers. Here, we define a Segment Hit-ratio as the ratio of the data size of segments directly satisfied by a cache to the total size of segments requested by clients.

Since the size of streaming objects is very large and researches showed that keeping the initial part of the stream in a cache is helpful to reduce delay, we define another hit ratio called Partial Hit-ratio. This Partial Hit-ratio is defined as the number of times when the size of the cached segments of the requested stream are more than client's buffer size divided by the number of total requests.

5.2 Simulation Results

This subsection is organized as follows. We firstly study the performance of caching algorithms in Sect. 5.2.1. Then, illustrative numerical results and comparisons for different replacing algorithms are provided in Sect. 5.2.2. Finally, Web-Friendly Streaming Caching Schemes are tested in Sect. 5.2.3.

5.2.1 Caching Algorithms Evaluation

In this subsection, performance of caching algorithms discussed in Sect. 4.1, *Last Watch*, *Average Watch* and *Judging,* are compared with the conventional algorithm

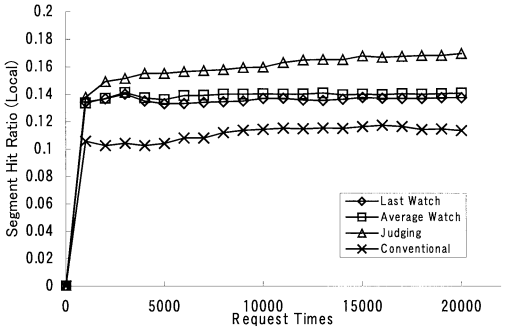


Fig. 2 Comparison of segment/byte hit-ratios of local proxy caches with different caching algorithms.

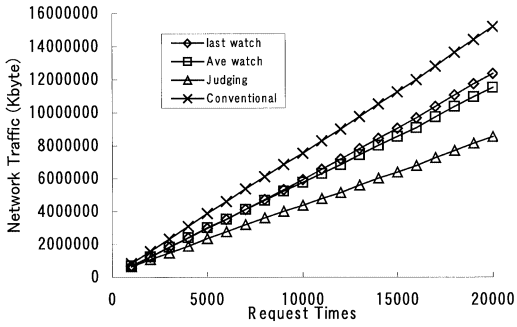


Fig. 3 Comparison of network traffics between an upper proxy and local proxies with different caching algorithms.

without any segmentation. In the *Judging* algorithm, whether or not an incoming segment should be cached in a local cache is decided by the priority of this segment. Here *PIA* defined in the Appendix A is used to decide segment priority. To study these caching algorithms under the same conditions, the same replacing algorithm (Popularity Policy in Sect. 4.2.2) is selected as the corresponding replacing algorithm in our simulation. The conventional one is the one that the stream is not segmented and the priority is given based on the unit of one stream instead of one segment, which means that the whole stream has the same priority regardless of segments. Also, the priority for one stream is decided by *LRU,* which is the most commonly used.

Figure 2 shows a comparison result of the Segment Hit-ratio of the local caches with respect to request times. For the conventional one without stream segmentation, Byte hit-ratio is used to evaluate its performance in Fig. 2. Note that the Byte hit-ratio is comparable with our proposal's Segment hit-ratio, which is also a kind of hit ratio of data size because we assume that each segment has the same size. Therefore, we compare the segment hit-ratio of our proposal with the conventional one's Byte hit-ratio. All of the three proposed algorithms outperform the conventional one in Fig. 2. The *Last Watch* and *Average Watch* provide a modest improvement while the *Judging* achieves the best result. The reason is that whether an incoming segment should be cached in a local cache is decided by the value of *PIA* based on the unit of segments according to the *Judging* policy. It can keep the clients' most accessed segments in a cache so that the Segment Hit-ratio is improved.

Figure 3 evaluates the network traffic between the upper proxy and local proxies. From this figure, we can find that the *Judging* algorithm performs best and can reduce the network traffic most since this algorithm only caches the segment with the high priority in a local cache. It also verifies that algorithms with high Segment Hit-ratios lead to reduction of network traffics.

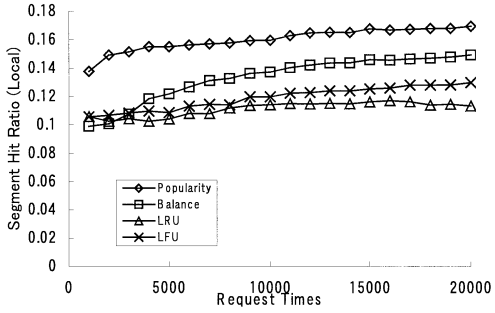


Fig. 4 Comparison of segment/byte hit-ratios of local proxy caches with different replacing algorithms.

5.2.2 Replacing Algorithms Evaluation

In this second step, performances of different replacing algorithms discussed in Sect. 4.2 are compared. There are four replacing policies we will study.

- *LRU Policy*
- *LFU Policy*
- *Popularity Policy*
- *Balance Policy*

The former two are conventional ones, which we have introduced by assigning *LRU* and *LFU* to one whole stream (not segments). The last two are introduced in Sect. 4.2.2. The proposed priorities, Priority Index of Access *(PIA)* and Priority Index of Relative Length (*PIRL*), are used in these two algorithms. Since there are 1000 streams in the server, according to Zipf distribution, it can be expected that the every top-20%-stream will be requested at least once when the total number of web requests reaches 2000 [20]. That is to say: if one stream has not been requested even once after 2000 web requests, it must be an unpopular one. Therefore, the 'zero check' strategy in the Balance Policy is carried out every 2000 requests in the simulation. Because the simulations in Fig. 2 and Fig. 3 showed that the *Judging* policy performs best, we will compare the replacing algorithms under the same situation where the *Judging* policy is applied as a caching algorithm.

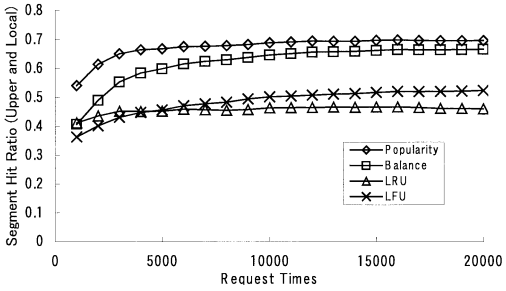


Fig. 5 Comparison of segment/byte hit-ratios in upper and local proxy caches with different replacing algorithms.

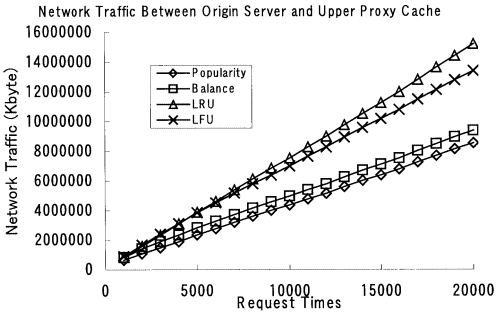


Fig. 6 Comparison of network traffics between an origin server and an upper proxy cache with different replacing algorithms.

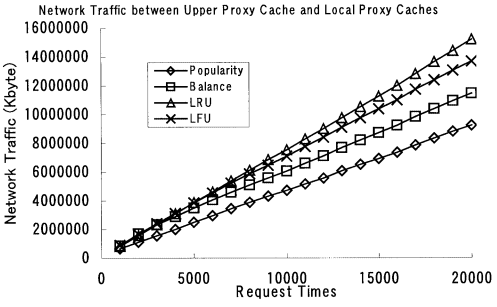


Fig. 7 Comparison of network traffics between an upper proxy cache and local proxy caches with different replacing algorithms.

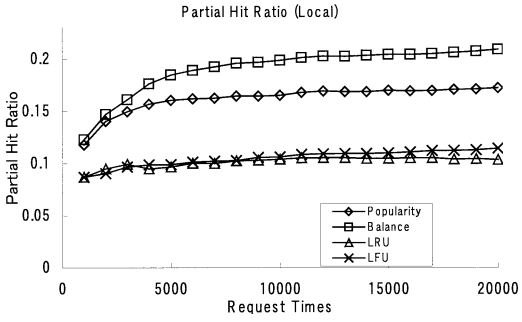


Fig. 8 Comparison of partial hit-ratios of local proxy caches with different replacing algorithms.

In Fig. 4 we show a comparison of the results of the Segment Hit-ratio in local caches as a function of request times. Among four algorithms, It can be found that the Popularity policy gets the best result. Let's refer to the Appendix A for explanation. *PIA* is related to clients' access records, accumulative access times and the period of recent accesses. As a result, the cache can cache the objects according to recent client access patterns so that it can improve the Segment Hit-ratio. Both *LRU* and *LFU* obtain approximate results. And the Balance policy shows poorer performance than the Popularity policy. The reason is that it is likely to keep the initial parts of unpopular streams in the cache resulting in wasted cache resources.

Note that the above result is only for the Segment Hit-ratio in local caches. We then go on to study the Segment Hit-ratio including both an upper proxy cache and local proxy caches. The result in Fig. 5 shows that the Popularity policy can still get the highest hit ratio. And the Balance policy obtains the second highest hit ratio, which reflects that both of the proposed algorithms are able to improve the cache performance of the whole system compared with the conventional methods.

Network traffic is also evaluated and divided by two parts respectively shown in Fig. 6 and Fig. 7. Figure 6 shows the traffic between an origin server and an upper proxy, and Fig. 7 does the traffic between an upper proxy and local. The results shown in in Fig. 6 and Fig. 7 are similar to those of Segment Hit-ratio, where the the Popularity policy outperforms all. Besides, it can be found that both the proposed algorithms substantially reduces network traffic in both Fig. 6 and Fig. 7.

If the first few segments of one stream are available in the client's buffer, the remainder of the stream can be sent to the client during the playback period of the beginning part in buffer. In Fig. 8 we plot the Partial Hit-ratio for all algorithms mentioned previously. Both the Popularity policy and the Balance policy get better results than the other conventional ones. Another point is that the Balance policy obtains the highest Partial Hit-ratio while the Popularity policy showed the best performance in Segment Hit-ratio and network traffic before. The reason is that the Balance policy tends to keep the same relative length of different streams in a cache. That is to say, large-size streams are likely to be removed to make room for other streams. As a result, more kinds of streams can be stored in a cache at the same time and the Partial Hit-ratio becomes higher.

5.2.3 Web Friendly Caching

In the previous simulations, we assumed that all of the requested objects by clients were streaming data. In this subsection, we show how our proposal cooperates with the caching of normal web objects such as HTML files and images.

The scheme introduced in Sect. 4.3 will be carried out with parameters ω=0.002 and Th =0.7. The details

about how to select these parameters are introduced in Appendix B. We assume that a server contains 1000 different streams and 10000 normal web objects. Streams have the same characteristics as mentioned before. The size of a normal object is 60 kbyte.

At the beginning of the simulation, the request arrival rate of streaming objects is 5% of normal web objects'. Then it is continually increased during the time of the first 50000 requests until both streaming data and normal web objects have the same request arrival rate.

The Web-friendly caching (*WFC*) is firstly carried into the conventional stream caching scheme, where the whole data of the stream is cached in the cache. Two scenarios are compared when the Web-friendly caching (*WFC*) is carried out (With *WFC*) or not (No *WFC*). From Fig. 9, we can recognize that, by keeping an adaptive proportion for normal web objects, the Segment Hit-ratio of the streaming media decrease about 8% while the hit ratio of normal web objects increase to around 80%.

Then, *WFC* is applied into the proposed stream caching scheme, where both segment caching and segment replacing have been considered. Compared with the result in Fig. 9, the improvement of normal web objects gets lower in Fig. 10. This is because the stream data is adaptively cached by segmentation in our proposal, which makes more room for web objects at the same time. However, it can be found that the Segment Hit-ratio of the streaming media decrease about 7% while the Segment Hit-ratio of normal web objects can still be increased to around 23% in Fig. 10. In one word, our WFC algorithm leads to "balanced" Segment Hit-ratio between web caching and stream caching. Therefore, it is confirmed that performance of the whole caching system is improved by using WFC.

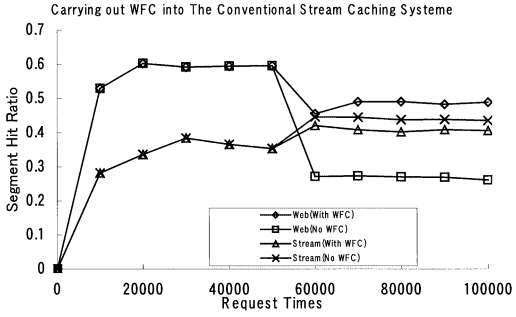


Fig. 9 Comparison of segment hit-ratios (of an upper proxy cache and local proxy caches) with/without carrying out WFC into the conventional stream caching scheme.

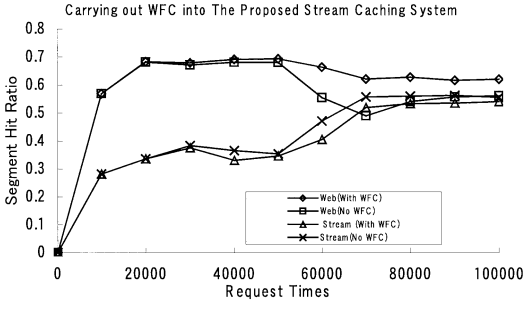


Fig. 10 Comparison of segment hit-ratios (of an upper proxy cache and local proxy caches) with/without carrying out WFC into the proposed stream caching scheme.

6. Conclusion

In this paper, novel mechanisms to deliver and cache streaming media over the Internet were discussed. Hierarchically distributed and cooperative caching systems were designed to spread the segmented streaming media.

Caching algorithms, where different priorities are assigned based on the segment units, were discussed, and three different caching algorithms were then proposed in order to cache a part of one stream into a local cache in an efficient manner. Among the proposed algorithms, simulation results showed that the *Judging* achieved the best performance.

Next, two priorities for streaming (priority index of access (*PIA*) and priority index of relative length (*PIRL*)) were proposed, and by combining them, two replacing algorithms were proposed, where both the proposed algorithms outperform the conventional ones. Simulation results showed that the Popularity policy showed better performance in Segment Hit-ratio and network traffic while the Balance policy obtained higher Partial Hit-ratio. That is to say, the Popularity policy is better to deal with network congestion while the Balance policy is helpful to achieve shorter response time.

Finally, a Web-friendly caching (*WFC*) scheme was proposed to cooperate streaming caching with normal web caching. Simulation results showed that the proposed approach guaranteed caching performance for normal objects without degrading the Segment Hit-ratio of streaming media.

There are a number of works to be done as further researches. Implementation onto actual systems is ongoing. Theoretical analysis should be expanded to be applicable to general cases.

References

[1] J. Kangasharju and K.W. Ross, "Performance evaluation of redirection schemes in content distribution networks," The 5th International Web Caching and Content Delivery Workshop, May 2000.
[2] A. Beck and M. Hofmann, "Enabling the Internet to delivery content-oriented services," Proc. 6th International Web Caching and Content Distribution, Boston, MA, June 2001.
[3] Adero, ⟨URL:http:www.adero.com⟩
[4] Akamai, ⟨URL:http:www.akamai.com⟩
[5] M. Chesire, A. Wolman, G.M. Voelker, and H.M. Levy, "Measurement and analysis of a stream media workload," USITIS'01, San Francisco, CA, March 2001.

[6] S. Acharya, B. Smith, and P. Parnes, "Characterizing user access to videos on the videos on the World Wide Web," SPIE/ACM MMCN 2000, San Jose, CA, Jan. 2000.
[7] M. Arlitt, R. Friedrich, and T. Jin, "Performance evaluation of Web proxy cache replacement policies," Lect. Notes Computer Science, vol.1469, pp.193–206 1998.
[8] Y. Yasuda, T. Yasuno, F. Katayama, T. Toida, and H. Sakata, "Image database system featuring graceful oblivion," IEICE Trans. Commun., vol.E79-B, no.8, pp.1015–1021, Aug. 1996.
[9] Z. Su, T. Washizawa, J. Katto, and Y. Yasuda, "A new and robust replacement algorithm for proxy caching with hierarchical image coding," IMPS2001, Nov. 2001.
[10] A. Ortega, F. Carignano, S. Ayer, and M. Vetterli, "Soft caching: Web cache management techniques for images," IEEE Signal Society Workshop on MSP, Princeton, NJ, June 1997.
[11] T. Ishikawa, W.B. Hui, H. Ohsawa, and Y. Yasuda, "A method of improving response time in still database based on CD-ROM changers by graceful caching," IIEEJ, vol.28, no.5, 1999.
[12] Z. Su, T. Washizawa, J. Katto, and Y. Yasuda, "A new prefetching algorithm for graceful caching system," Proc. IEICE Gen. Conf. 2001, March 2001.
[13] Z. Su, T. Washizawa, J. Katto, and Y. Yasuda, "Performance improvement of graceful caching by request frequency based prefetching algorithm," IEEE Tencon, Singapore, Aug. 2001.
[14] P. Rodriguez, C. Spanner, and E.W. Biersack, "Web caching architectures: Hierarchical and distributed caching," The 4th International Web Caching Workshop, March 1999.
[15] S. Paul and Z. Fei, "Distributed caching with centralized control," The 5th International Web Caching and Delivery Workshop, May 2000.
[16] J. Kangasharju, F. Hartanto, M. Reisslein, and K.W. Ross, "Distributing layered encoded video through caches," IEEE Trans. Comput., vol.51, no.6, pp.622–636, June 2002.
[17] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," IEEE INFOCOM99, New York, NY, March 1999.
[18] S.-J. Lee, W.-Y. Ma, and B. Shen, "An interactive video delivery and caching system using video summarization," WCW 2001, Boston, MA, June 2001.
[19] V.A.F. Almeida, M.G. Cesario, R.C. Fonseca, W.M. Jr, and C.D. Murta, "Analyzing the behavior of a proxy server in the light of regional and cultural issues," Proc. 3rd International WWW Caching Workshop, 1998.
[20] L. Breslao, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zip-like distributions: Evidence and implications," Proc. IEEE INFOCOM'99, New York, April 1999.
[21] J. Gwertzman, "Autonomous replication in wide-area networks," Technical Report 17-95, Harvard University, 1995.
[22] K. Ebisawa, J. Katto, and Y. Yasuda, "A study on distributed caching for multimedia streams," IEICE Autumn Conference, B-7-84, Sept. 2000.
[23] K. Ebisawa, J. Katto, and Y. Yasuda, "A Web-friendly streaming caching scheme," IEICE Spring Conference, B-7-206, March 2001.
[24] M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara, "Proxy caching mechanisms with video quality adjustment," SPIE ITCom, Feb. 2001.
[25] J. Byers, J. Considine, and M. Mitzenmacher, "Informed content delivery across adaptive overlay networks," SIGCOMM 2002, Pittsburgh, PA, Aug. 2002.
[26] Gnutella, ⟨http://www.gnutella.co.uk⟩
[27] Napster, ⟨http://www.napster.com⟩
[28] R. Rejaie, H. Yu, M. Handley, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet," IEEE INFOCOM 2000, March 2000.
[29] Z. Su, J. Katto, T. Nishikawa, M. Murakami, T. Washizawa, and Y. Yasuda, "An integrated scheme to distribute segmented streaming media over hierarchical caches," ICITA2002, Bathurst, Australia, Nov. 2002.

Appendix A

In this appendix, we give a mathematical formulation of the problem we are concerned with. Performance evaluation of cache is given by the value of Segment Hit-ratio. If a miss hit happens, the clients need to wait for the cache to fetch the absent data from the server. It will also often incur network transmission cost and delay. Therefore, our goal is to selectively cache different segments of different streams so as to maximize the total Segment Hit-ratio.

In the case when the cache keeps storing the first k segments of stream i, if a user requested this stream and stopped watching it after the playback of j segments, the probabilistic form of the Segment Hit-ratio is given by the following equation:

$$P_{s\_hit} = \sum_{i,j,k} P(j \le k \,|j,i)p(j,i) \qquad (\text{A}\cdot 1)$$

Where $P(j \le k|j,i)$ is the conditional probability distribution, or cpdf for short, that means the user is satisfied after watching the playback of total j segments already stored in the cache. $P(j,i)$ is the pdf that a user requests the j segments of stream i. And $P(j,i)$ is decomposed into $P(j|i)$ and $P(i)$ and Eq. (1) is rewritten as

$$P_{s\_hit} = \sum_{i,j,k} P(j \le k \,|j,i)P(j\,|i)P(i) \qquad (\text{A}\cdot 2)$$

Improvement of the Segment Hit-ratio P_{s-hit} can be obtained to increase only $P(j \le k|j,i)$ because $P(j,i)$ almost completely depends upon users' preferences, therefore this parameter is not controllable by the proxy cache.

With an infinite amount of storage, $P(j \le k|j,i) = 1$ when the first j segments of stream i with the pdf $P(j|i) = 1$ are stored in the cache.

In practice, however, cache capacity is finite and we must determine which segments of which streams to be stored. Therefore, if the correct $P(j,i)$ of the top-ranked segments of the streams can be estimated and then stored in the cache, more clients' requests will be satisfied and then high $P(j \le k|j,i)$ and a high Segment Hit-ratio can be obtained.

If a client's requested data is not stored in the cache and the cache is full, replacement is required to make room for incoming data.

Let the indices of the removed segments of the streams from the cache be $\Lambda 3$. If the k-th segment of the stream a is removed, $\Lambda 3 = \{(k, a)\}$. Besides, the new coming segments are $k+1$ to l segments of the stream m. We denote the set of indices $\{(k+1, m), (k+2, m), \ldots, (l, m)\}$ as $\Lambda 2$.

Then the Segment Hit-ratio after the replacement can be written as follows:

$$\begin{aligned} P_{s\_hit} = & \sum P(j \le k/j, i)P(j, i) \\ & + \sum P(r \le k/l, m)P(r, m) \\ & - \sum P(s \le k/r, n)P(s, n) \end{aligned} \quad (\text{A}\cdot 3)$$

Where (r, m) belongs to $\Lambda 2$ and (s, n) belongs to $\Lambda 3$. Since only the third term is controllable by the cache server in the above equation, the maximization of P_{s-hit} is equivalent to the minimization of the third term. Therefore, $\Lambda 3$ is the set of indices of the layers of the images with the lowest pdf $P(s, n)$.

Therefore, the segment of one stream with the lowest $P(j|i)P(i)$ should be removed first. This is the rule of the replacement.

How to efficiently calculate $P(j|i)P(i)$ is also a problem considered by us. As there are numberless segments of different streams, keeping the request records for all segments of all streams is not realizable. Here, we introduce a method to obtain $P(j|i)$ by just keeping the stopping viewing point of stream i within several past requests.

Assume that stream i has been requested by Q times within a fixed period and each time the client stopped watching stream i after the playback of segment $S_{q,i} (q \in (1, Q),\ i \in (1, M),\ S \in (1, B_i))$. $P(j|i)$ can be then computed as follows:

$$P(j\,|i) = |\{S_{q,i}\,|j \le S_{q,i}\}|\,/Q \quad (\text{A}\cdot 4)$$

where $|.|$ denotes the number of elements of a set.

On the other hand, $P(i)$ is the request frequency of stream i. It can be obtained by calculating the accumulative request times within one testing period. However, as the popularity may change over time, a stream with high access frequency in the last period may not be popular currently. Therefore, how to decide the testing period is very important.

We define T and $T_{c,i}$ respectively as the present time and the time when stream i is cached in the cache. Let v_i denote the request times between $T_{c,i}$ and T. Then, we can get:

$$P(i) = g(i)/(\textstyle\sum_j g(j)) \quad (\text{A}\cdot 5)$$

$$g(i) = v_i/(T - T_{c,i}) \quad (\text{A}\cdot 6)$$

In the above equation, both access times (v_i) and access period ($T_{c,i}$) are considered. We think it is better than just adopting a constant period to calculate request frequency.

Hence, the priority index of segment can be decided by

$$\begin{aligned} P(j\,|i)P(i) = & \left(|\{S_{q,i}\,|j \le S_{q,i}\}|\,/Q\right)g(i) \\ & \Big/\Big(\textstyle\sum_j (g(j)\Big) \end{aligned} \quad (\text{A}\cdot 7)$$

As Q and the summation of $g(j)$ over j are identical for all streams in the above Quation, for convenience, we define *PIA* (priority index of access) as follows:

$$PIA = |\{S_{q,i}\,|j \le S_{q,i}\}|\,v_i/(T - T_{c,i}) \quad (\text{A}\cdot 8)$$

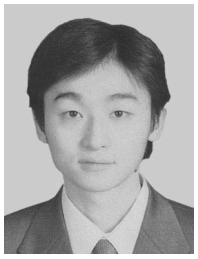
Appendix B

In this appendix, we induce how to decide the parameters for the EWMA estimator.

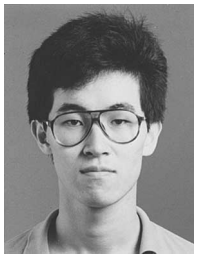
In the EWMA estimator, a smoothing constant $w = 2/(1 + \text{Pe})$ is introduced, where Pe = number of periods for EWMA. 100,000 accesses are assumed in our simulation and the EWMA estimation is executed per every 100 accesses. Accordingly, parameter values are set to Pe = 1000 and w = 0.002.

We assume that a server contains 10000 normal web objects with the size of 60 kbyte. So the data size of all normal objects amounts to 10000 * 60 = 600,000 kbyte. On the other hand, there are also 1000 different streaming objects with the rate of 64 kbps in the same server. The length of each stream is assumed to be uniformly distributed from one minute to ten minutes. So the data size of all streams are 1000 * (64/8) kbyte/second * [(1+10)/2]minute * 60second/minute = 4.4 * 600,000 kbyte. On the assumption that the stream is very long and the position where a client stops watching a stream is decided at random, the possibility that a segment is watched becomes 1/2, so we can set an average size of the watched streams to be 4.4 * 600,000 * 0.5 = 2.2 * 600,000 kbyte. Therefore, the ratio of the stream size to normal object size is (2.2: 1) resulting in $Th = 2.2/(2.2+1)$, which is 0.7.

That is to say, the scheme introduced in Sect. 4.3 will be carried out with parameters ω =0.002 and Th = 0.7.



Zhou Su is currently working for the Ph.D. degree at the Graduate School of Science and Engineering, Waseda University, after receiving his B.S. and M.S. degree from Xi'an Jiaotong University, Xi'an, China in 1997 and 2000, respectively. He was an exchange student between Waseda and Xi'an Jiaotong University from 1999 to 2000. From 2001 he has been a research associate at Waseda University. His research interests include multimedia communication, web performance and network traffic. He received the SIEMENS Prize in 1998, and received Rockwell Automation Master of Science Award in 1999. He is a student member of IEEE and IEE.



Jiro Katto was born in Tokyo, Japan, in 1964. He received the B.S., M.E. and Ph.D. degrees in electrical engineering from University of Tokyo in 1987, 1989 and 1992, respectively. He worked for NEC Corporation from 1992 to 1999. He was also a visiting scholar at Princeton University, NJ, USA, from 1996 to 1997. He then joined Waseda University in 1999, where he has been an associate professor at the Department of Electronics, Information and Communication Engineering, School of Science and Engineering. His research interest is in the field of multimedia signal processing and multimedia communication systems such as the Internet and mobile networks. He received the Best Student Paper Award at SPIE's conference of Visual Communication and Image Processing in 1991, and received the Young Investigator Award of IEICE in 1995. He is a member of the IEEE and the IPSJ.

Takayuki Nishikawa was born in Chiba, Japan in 1978. He received the B.E. degree in Electronic Information and Communication Engineering from Wasada University, Tokyo, Japan in 2002. His current research interests include the Contents Delivery Network and Network Traffic. He is currently working toward the M.E. degree at Waseda University.

Munetsugu Murakami was born in Yamaguchi, Japan in 1978. He received the B.E. degree in Electronic Information and Communication Engineering from Wasada University, Tokyo, Japan in 2002. His current research interests include the Contents Delivery Network and Network Traffic. He is currently working toward the M.E. degree at Waseda University.

Yasuhiko Yasuda was born in Tokyo on July 7, 1935. He received B.E. and M.E. degrees in Electrical Engineering, and D.E. degree in Electronic Engineering from the University of Tokyo, in 1958, 1960 and 1963, respectively. In 1963 he joined the Institute of Industrial Science, the University of Tokyo as an assistant professor and was promoted to a full professor in 1977. Since retiring from the University of Tokyo in August of 1992, Dr. Yasuda has been professor at the Department of Electronics, Information and Communication Engineering, School of Science and Engineering, Waseda University. During his long career of research, he produced numerous useful achievements including invention of delta sigma modulation, development of 3-level VSB high speed transmission scheme for newspaper fax machines corresponding to the CCITT G2 fax standard, proposal of coding methods for digital facsimile images, proposal of hierarchical image coding and processing, etc. His current interest is in the fields of image coding and processing, mobile communications and satellite communications. He is a past President of Institute of Electronics, Information and Communication Engineers (IEICE) and a past President of the Institute of Image Electronics Engineers of Japan. He served acting chairman of Telecommunication Technology Council (MPT), a member of Broadcast Technology Council (NHK) and a member of board of directors, Engineering Academy of Japan. He is now serving President of Mobile Computing Promotion Consortium, Chairperson of Japan Approvals Institute for Telecommunications Equipment, President of the Telecommunication Technology Committee. Dr. Yasuda has been awarded numerous prizes from various academic organizations including the IEICE Outstanding Achievement Award.