行政院國家科學委員會專題研究計畫 成果報告

寬頻網際網路之多媒體代理伺服器的快取及取代控制方法
和機制

計畫類別：個別型計畫
計畫編號： NSC93-2213-E-006-013-
執行期間： 93 年 08 月 01 日至 94 年 07 月 31 日
執行單位： 國立成功大學資訊工程學系（所）

計畫主持人： 黃崇明
計畫參與人員： 許子衡 趙志文 林進瑋 蔡競賢

報告類型： 完整報告
處理方式： 本計畫可公開查詢

中華民國94年07月22日
一、中文摘要

為了解決網路頻寬的不足所造成的多媒體服務品質問題，我們提出了一個以媒體物件價值為考量觀點的快取演算法GCC-DS-F-DA。此演算法除了考慮傳統網路環境中影響資料暫存的因子，如物件大小及傳輸延遲，還考慮了多媒體物件價值及其對多媒體內容服務提供者的整體效益。在本演算法中，我們設計了一個快取利益計算公式，利用此公式來評估快取一多媒體物件所能得到的回饋。根據得到的利益，來決定物件是否暫存於快取空間之中。GCC-DS-F-DA演算法的主要貢獻有二：(1) 藉由以媒體物件價值為考量觀點的快取利益計算公式，增加行動通訊服務提供者的整體效益；(2) 減少使用者存取多媒體服務的等待時間。

關鍵詞： Multimedia caching, data prefetching, Peer-to-Peer, wireless and mobile networks, file sharing

Introduction

With the explosive growth of 3G cellular/WLAN networks, supporting multimedia services is considered as a promising trend in the wireless network environment. The increasing use of wireless multimedia service invokes several problems that need to be resolved. A typical example is that the overloading of the media transmission, which increases multimedia retrieval latency, makes the infrastructure's backbone be the bottleneck. The limitation of wireless network bandwidth between a client and a remote server results in a major obstacle on the performance of multimedia presentations. By equipping some multimedia caching and sharing schemes for multimedia services and through the collaboration of multimedia proxies or P2P file sharing networks, the utilization of media transmission can be enhanced, and the requirement of real-time multimedia presentations can be satisfied. The multimedia caching and sharing schemes not only enables the quality adaptation more effectively, but also maximizes the delivered quality.

To speed up the transmission of multimedia objects in wireless network environments, we propose a price-based caching algorithm named GCC-DS-F-DA (GCC-DS with Frequency and Dynamic Aging). Unlike other existing caching replacement algorithms considering only the object size and transmission delay, the GCC-DS-F-DA caching algorithm considers both the requirement of (i) service and content providers and (ii) mobile users. In the GCC-DS-F-DA caching algorithm, a caching profit formula is derived to estimate the reward for service providers to cache a multimedia object. By caching only those objects that have a higher gold content ratio and larger response time than others, service and content providers can get better reward and users can get better service quality.

Peer-to-peer (P2P) computing is another approach to generalize the proxy functionalities into every client. In P2P overlay networks, computers can act as proxies. Such a P2P paradigm allows users to contribute their storages for sharing multimedia objects and reduce transmission latency among participants. We propose a novel P2P architecture named Shoran for interconnecting heterogeneous P2P resource sharing networks to enable connectivity and universal access of multimedia objects. Heterogeneous P2P file sharing networks need a way to collaborate and communicate with each other. Based on the approach of
interconnecting heterogeneous P2P resource sharing networks, users on one P2P resource sharing network can share and search multimedia objects with other P2P resource sharing networks.

**Existing Caching Algorithms**

In the past years, some caching algorithms have been proposed for web caching. In [ASAWF95], Abrams et al. proposed the LRU-Threshold and Log(Size) + LRU algorithms. The LRU-Threshold algorithm is the same as LRU, but does not cache documents larger than a certain threshold size. The Log(Size) + LRU algorithm replaces the document which has the largest log(size) and is the least recently used. In [WASAF96], Williams et al. proposed the SIZE and Hyper-G algorithms. The SIZE algorithm is for replacing the largest document from cache. The SIZE algorithm tries to minimize the miss ratio by replacing the largest document rather than a lot of small documents. These works concentrated on the web document size and its' impact is on the caching hit ratio.

In [WA97], Wooster et al. proposed the Latency Estimation (LAT) algorithm and Hybrid (HYB) algorithm. The LAT algorithm estimates the time required to download a document, and then replace the document with the smallest download time. It is aimed to minimize the average latency. The HYB algorithm is a hybrid of several factors, which considers not only download time but also the number of references to a document and the document size. These works concentrated on the retrieval latency of web documents rather than caching hit ratio. In [Cao-Irani97], Cao and Irani proposed the Greedy-Dual-Size (GD-Size) algorithm. The GD-Size algorithm deals with the case when documents in a cache have variable size and cost. By setting the cost of bringing the document and size of the document, the GD-Size can have better performance than LRU, SIZE, and Hybrid for different metrics. In [C98], Cherkasova proposed the Greedy-Dual-Size-Frequency (GD-Size-Frequency) caching policy. The GD-Size-Frequency policy is an improvement of GD-Size by incorporating the idea of frequency count with an aging mechanism. In [Yingjie2001], Li et al. proposed a CNP caching algorithm and extended its support of Web access for wireline networks into the wireless GPRS environment.

Most of the above cache algorithms focus on the network performance metrics, e.g., hit ratio, byte hit ratio, and latency reduction. Since commercial service providers expand their Internet services and need a way to investigate how caching will affect pricing and charging policies, pricing and charging issues will become more and more important for the coming years. Therefore, a caching algorithm that can (1) maximize the the reward of service and content providers and (2) reduce the transmission delay of request objects is needed. By caching those objects that have higher price and larger response time than others, service and content providers can get better reward and users can get better service quality.

**Price-based Caching Algorithm**

This section presents a caching algorithm that is based on the content’s price and users’ usages. The detailed consideration and design of the proposed price-based caching algorithm is depicted and discussed.

**Gold Content Control Policy (GCC)**

The Gold Content Control (GCC) policy aims at maximizing the reward of content providers by caching only those objects that have a higher gold content ratio than others. GCC associates every object $i$ with a gold content ratio $G_i$, which correlates the content price $P_i$ in retrieving the object $i$ with the size $S_i$ (measured in bytes) of the object. The value of gold content ratio $G_i$ is defined as follows:

$$G_i = \frac{P_i}{S_i}$$

The gold content ratio $G_i$ reflects the general idea that the object of higher content price should be putted in cache for revenue concern. When two or more objects have the same price, it is required to make a choice among them and the choice should effectively
increase the utilization of cache storage. Therefore, removing the one with a larger size will free larger amount of cache storage. The cache table is sorted in the ascending order of the gold content ratio. When the cache storage is full and a new object needs to be cached, the object with the lowest gold content ratio is chosen for replacement.

**Gold Content Control Policy with Delay Sensitive (GCC-DS)**

In the GCC-DS policy, we take consideration of Gold Content Control Policy with the object’s transmission delay. The GCC-DS policy aims at (1) maximizing the reward of content providers and (2) reducing the transmission delays of request objects. By caching only those objects that have a higher gold content ratio and larger response time than others, content providers can get better reward and users can get better service quality. GCC-DS associates every object with the average transmission time Ti (measured in seconds/byte) of retrieving object i from server. The value of profit Profiti is defined as follows:

\[ Profit_i = \frac{G_i}{G_{\text{max}}} + \frac{T_i}{T_{\text{max}}} \]  \hspace{1cm} (2)

where Gi is the gold content ratio of object i and Gmax is the maximum value of gold content ratios of all objects stored in the cache, Ti is the transmission time of object i and Tmax is the maximum value of transmission time of all objects stored in the cache. The profit parameter can help us to determine which object is worthy to store in the cache. Comparing with other objects, the object with a higher gold content ratio and higher transmission time will have better chance to stay in cache.

A service provider may want to increase the revenue rather than to reduce the transmission delay. Thus, the priority of gold content ratio must be higher than the priority of transmission delay. However, Equation (2) can’t satisfy this kind of requirement because the priority of gold content ratio and transmission delay is equal. Therefore, we modified Equation (2) to accommodate such kind of requirement. The value of profit Profiti is changed to:

\[ Profit_i = \left( \frac{G_i}{G_{\text{max}}} \right)^{W_i} + \left( \frac{T_i}{T_{\text{max}}} \right) (1 - W_i) \]  \hspace{1cm} (3)

where Wi is a weight value that determines the weight of gold content ratio Gi. The value of 1- Wi determines the weight of transmission delay. By giving different weight values to the gold content ratio and transmission delay, the GCC-DS policy can satisfy different requirements based on the chosen weight values.

**GCCDS with Frequency and Dynamic Aging**

The GCC-DS policy can be improved to reflect the object access pattern by incorporating a frequency count Fi. The frequency of request parameter Fi is associated with each object i stored in a cache. Incorporating Fi with the GCC-DS policy results in the GCC-DS-Frequency policy, the GCC-DS policy with frequency factor (GCC-DS-Frequency) maintains the following parameters in the cache table for each of the cached objects:

- Gi: the gold content ratio of object i
- Ti: the average transmission delay to retrieve object i from the server
- Fi: the frequency count associated with object i

These parameters are combined into a priority function Pr(i) which is defined as follows:

\[ Pr(i) = Fi \times \left( \frac{G_i}{G_{\text{max}}} \right)^{W_i} + \left( \frac{T_i}{T_{\text{max}}} \right) (1 - W_i) \]  \hspace{1cm} (4)

Object i is inserted into a cache table with the priority value that is computed from Equation (4). GCC-DS-Frequency selects the least used object for replacement when the cache capacity is exceeded.

Considering that a new object i of size Si arrives and needs to be cached. If the free cache space is greater than Si, then it causes no problem. If the available space is less than Si, then some objects are required to be removed from cache. GCC-DS-Frequency sorts the entries in the cache table in the ascending order of priority and selects the objects in that order for eviction. Thus, least priority objects are replaced first. A high priority value indicates a high revenue in
fetching the object and that the object was frequently requested. GCC-DS-Frequency also implies that objects with high Ti are given preference over objects with low Ti for caching, which ultimately reduces the transmission delay.

A drawback of the GCC-DS-Frequency algorithm is that some objects are brought into the cache and may never be requested again. It is also known as the pollution problem. The pollution problem is caused by objects with a short period of sudden popularity increase and occupy the cache spaces. These stale objects may never be accessed again and still persist in the cache for a long time because their priorities remain high. LFU with Dynamic Aging (LFUDA) uses dynamic aging to accommodate shifts in the set of popular objects. The dynamic aging policy simply increments the cache age when evicting objects from the cache – setting it to the key value of the evicted object. LFUDA has the property that the cache age is always less than or equal to the minimum key value in the cache. This prevents the need for parameterization of the policy, which LFU-Aging requires. By adding the cache age factor, the influence of previously popular objects from polluting the cache is limited. The GCC-DS-Frequency with the dynamic aging policy is depicted as follows:

\[ Pr(i) = F_i \times (\frac{G_i}{t_{max}} \cdot W_i + \frac{T_i}{t_{max}} \cdot (1-W_i)) + L, 0 < W < 1 \]  

(5)

Parameter L is the age factor that starts at 0 and is updated when evicting objects from the cache. Parameter L is set to the priority value of the evicted object. Parameter L is defined as follows:

\[ L = Pr(\text{evicted}) \cdot \min_{i \in \text{cache}} Pr(i) \]  

(6)

Every time an object i is evicted from the cache, the value of parameter L is calculated. L is the same as the priority value of the evicted object i. When a new object j is put into the cache, the priority of new object j that replaces the object i is calculated from Equation (5) with the parameter L. In this way, the priority of new object j is bigger than parameter L. Every time an object is evicted from the cache, the parameter L as well as the priority of new object j are increased. At some point, the priority of new object j gets high enough than these ”long time no accessed” objects. Therefore, the stale objects with high priority value get replaced if they are not accessed again. In this way, the pollution problem can be solved because these objects will not persist in the cache. The ”aging” mechanism prevents proxy cache from pollution. Using the aging technique, our caching policy can avoid the pollution problem. Figure 1 shows the detail algorithm of our price-based caching policy.

The operation of the caching policy is as follows:

1. At time t, if the client sends a request to acquire the desired object, then the proxy examines the cache (priority queue) to identify the requested data.
2. If the requested data is already stored in the cache, then the proxy re-calculates the priority of this data. The proxy will update the priority of this data according to the GCC-DSFrequency with the dynamic aging policy. The counter of hit is plus one. At the same time, the proxy sends back the requested data to the client.
3. If the requested data are not in the cache, then the proxy sends a request to the server. After the proxy gets the requested data, it calculates the priority of this data and then puts it into the cache according to this priority. The counter of miss is plus one. At the same time, the proxy sends back the requested data to the client.

4
Figure 1: The Price-based Caching Policy

Symbol definition:
CacheQueue: the priority queue that stores the cache data
RequestData: the data that are requested by users
L: the aging factor
End definition

while a request arrives do
// Checking cache is empty or not. If cache is empty, retrieves // the requested data from server and then stores the requested // data into cache.
if CacheQueue.empty() then
Server.Retrieve(RequestData);
RequestData.priority= CalculateProfit(RequestData);
CacheQueue.push(RequestData);
Miss++;
else
if CacheQueue.find(RequestData) then
TotalHitByte=TotalHitByte+RequestData.bytes;
TotalHitReward=TotalHitReward+RequestData.price;
RequestData.priority= CalculateProfit(RequestData);
CacheQueue.update(RequestData);
Hit++;
else
Server.Retrieve(RequestData);
while StorageSize <= CalculateCacheQueueSize() do
if EnableAging==true then
L=(RequestData.top()).priority;
end if
CacheQueue.pop();
end while
RequestData.priority= CalculateProfit(RequestData);
CacheQueue.push(RequestData);
Miss++;
end if
end if
end if
end while

Figure 2: Shoran’s super-peer components.

Figure 3: The abstract diagram of the lookup routing module.

module that provides resource retrieval service for heterogeneous P2P file sharing networks, (4) a result cache module that provides the cache service for query/response message, and (5) a XML-based uniform message format (UMF) defining the communication protocol for heterogeneous P2P file sharing networks. Figure 2 depicts the abstract components of Shoran’s super-peer. The lookup routing module is responsible for routing the messages that are based on the uniform message format (UMF) to/from different super-peer. The message exchange module is responsible for converting network dependent query messages to/from the UMF-formatted messages that the super-peer network can understand. UMF is used to define the network independent messages for finding resources that are stored in heterogeneous P2P file sharing networks. A standardized message format can simplify the complexity of managing multiple communication message format. By using the XML technique, each super-peer can convert network dependent query messages to/from UMF-formatted messages. In other words, it will be much easier for developers to add new P2P file sharing protocols using UMF.

Lookup Routing Module

The lookup routing module consists of (i) a message queue that stores incoming messages, (ii) a message queue manager that controls and maintains state information of the message queue, (iii) a forward engine that performs message forwarding according to
the message routing table, and (iv) a message routing table that holds super-peers' addresses. The super-peers' addresses are referenced by a set of indexes, which determine the corresponding address mapping from incoming message to outgoing message. The message routing table can be configured statically or dynamically. Figure 3 depicts the abstract architecture of the proposed lookup routing module. The lookup routing module is responsible for routing messages in super-peer network. Two main functions of the lookup routing module are (i) managing incoming messages and (ii) forwarding messages to the corresponding super-peer. When one super-peer sends a UMF-formatted query/response message to another super-peer, the lookup routing module scans the routing table for an address of other super-peer and forwards the message to the selected super-peer. When a selected super-peer receives a UMF-formatted message, it then converts the UMF-formatted messages into the heterogeneous P2P file sharing networks by using the message exchange module.

In order to avoid message flooding problem, the message routing module uses similar techniques of Gnutella protocol to prevent loops and indefinite propagation in super-peer network. If the lookup routing modules receives a query message, the forward engine forwards the incoming message to all super-peers that are connected with it except the super-peer from which it receives the message. By using the Time To Live (TTL) counter to limit the range of message forwarding, the indefinite propagation can be avoid. By using a

Globally Unique Identifiers (GUIDs), a super-peer will not receive same message twice.

**Message Exchange Module**

The message exchange module consists (i) a message converter that performs the message conversion, (ii) a message state table that records the state information of the query message, and (iii) a message translation table that stores transition information for converting messages. These tables are used for processing super-peers' queries and responses. Two main functions of the message exchange module are (i) converting each underlying P2P file sharing network’s message format to/from the uniform message format (UMF) and (ii) providing protocol interoperability among various P2P file-sharing applications. The message exchange module handles all messages of its underlying P2P file sharing network. When the message exchange module receives query messages from its underlying P2P file sharing network, it converts these messages to the uniform message format (UMF). These UMF-formatted messages are routed through the lookup routing module, and then reach the destination super-peer. The destination super-peer then converts these UMF-formatted messages into the message format that is understood by the destined P2P file sharing network. Figure 4 shows the abstract diagram of message exchange module.

**Protocol Adaptation Module**

The protocol adaptation module consists a set of protocol adaptors which offer data
transfer services for the file-sharing applications among heterogeneous P2P file sharing networks. The protocol adaptor provides the adaptation of one resource retrieval protocol to another resource retrieval protocol so that resource can be retrieved among different resource-sharing applications. In the current environment of P2P file sharing networks, there is a need to integrate various protocols for different P2P resource-sharing applications. For example, the Gnutella uses HTTP as its data download protocol, but the Napster uses its application-dependent protocol as its data download protocol. Without protocol adaptation, file-sharing applications cannot share data with each other. Shoran supports the protocol adaptation among heterogeneous P2P file sharing networks. The protocol adaptor acts as a proxy module, which includes a temporal cache for storing the resource data. Upon receiving a request from a peer, the message exchange module passes it to a selected protocol adaptor. If the resource provider (peer) is ready, the protocol adaptor sends a request to retrieve the data and then stores the data into a temporal cache. After finishing the data retrieval, the protocol adaptor sends the data back to the peer that sends the request. The developers can write their own adaptors to integrate various protocols such as HTTP, FTP, TCP and application-dependent protocols. All adaptors need to provide interfaces for exposing their parameters. The function enables the message exchange module to choose a suitable protocol adaptor for transforming data. Through the use of the protocol adaptor, different P2P resource-sharing applications are able to share resources among each other. Figure 5 depicts steps involved in protocol adaptation among heterogeneous P2P file sharing networks.

**Result Cache Module**

In Shoran, every super-peer maintains two data structures: (i) a query result cache that contains recently query/response information and (ii) a data cache for processing data retrieval request. The query result cache is a list containing an entry for every unique query request received and processed. Each entry contains the following fields: a string stores the original query message, a query message identifier, a source identifier (requester’s id), a destination identifier (resource provider’s id), the time of a query message was last used for searching resource, and number of times a query message is found in result cache. A query result entry which has not been used for more than QueryLifeTime seconds is deleted. The data cache is a list containing an entry for every unique data request received and processed. Each entry contains the following fields: a source identifier (requester’s id), a destination identifier (resource provider’s id), a session number of the data request, a source protocol description, a destination protocol description, a set of super-peers that cooperate to deliver the data, and a record stores status of the data request.

**Project Evaluation**

By incorporating with the web caching concept, users can have a better presentation quality for web services. In this project, we proposed a price-based cache algorithm for mobile services provider, in which services are charged. The main contribution of the price-based caching algorithm is twofold: (1) to provide a price-based caching mechanism and thus can, potentially, increase the reward of media objects to service and content provider and (2) to provide a mechanism that can reduce media transmission delay based on the pricing concept.

Unlike other existing caching replacement algorithms, the GCCDS-F-DA caching policy considers both the requirement of (i) service and content providers and (ii) cache users. By caching only those objects that have a higher gold content ratio and larger response time than others, service and content providers can get better reward and users can get better service quality. We also shows that the performance of GCC-DS-F-DA is better than the performance of LRU and LFU caching replacement algorithms in the cache hit ratio concert.
With the rapid development of P2P techniques, many applications/systems are proposed to share resource via the P2P overlay networks. The lack of a message exchanging mechanism makes peers cannot share resources among heterogeneous P2P overlay networks. Therefore, how to make resource available among heterogeneous P2P overlay networks becomes an important issue. In this project, we have proposed an architecture to handle message exchanging and resource sharing among heterogeneous P2P overlay networks. Our message exchanging scheme are based on the XML technique and RDF description. We proposed a uniform message format to ease the communication among heterogeneous P2P overlay networks. With the uniform message format (UMF), peers can exchange messages for finding and sharing resources. In this way, users can share resources without the interconnect limitation and developers can be free from hardship in converting messages among different P2P overlay networks. Comparing with related works proposed in the literature, our work focuses on the resources retrieval in heterogeneous P2P overlay networks environment. Different P2P overlay networks may use different resource retrieval protocols. Therefore, peers need a mechanism to retrieve resources among heterogeneous P2P overlay networks. A resource retrieval scheme is proposed in our project. The peers can retrieve resources with the help of super-peers. The super-peers consists a set of protocol adaptors, which provide the adaptation of one resource retrieval protocol to the other resource retrieval protocol. Our future work includes the in depth analysis of the relationships of heterogeneous P2P overlay networks, and the study of other more advanced and useful resource discovery mechanism to fast up the speed of finding desired resources.

References


