CIGMA: aCtive Inventory service in Global e-MArket based on high performance catalog DB caching

Su Myeon Kim
Korea Advanced Institute of Sci. and Tech.
373-1 Kusung-Dong, Yoosung-Ku, Taejon, 305-701, Korea
smkim@nclab.kaist.ac.kr

Seungwoo Kang
Korea Advanced Institute of Sci. and Tech.
373-1 Kusung-Dong, Yoosung-Ku, Taejon, 305-701, Korea
swkang@nclab.kaist.ac.kr

Heung-Kyu Lee
Korea Advanced Institute of Science and Technology
373-1 Kusung-Dong, Yoosung-Ku, Taejon, 305-701, Korea
hklee@casaturn.kaist.ac.kr

Junehwa Song
IBM T. J. Watson Research Center
P.O. Box 218, Yorktown Heights N.Y.,10598, USA
junesong@us.ibm.com

ABSTRACT

A fully connected Internet business environment will introduce a high level of dynamics to the business process. It may result in very frequent changes in business decision, and thus, information of various items may undergo constant changes. In addition, there could be a flood of similar shopping sites. In such a highly dynamic environment, ordinary online customers may feel that online shopping is not comfortable. Existing service models or systems cannot effectively reflect such a dynamic environment and support ordinary online customers. This paper describes a new e-commerce service called the CIGMA. The CIGMA provides catalog comparison and purchase mediation services over multiple shopping sites for ordinary online customers. A key characteristic of the CIGMA is that the service is provided based on high performance catalog database caching. So, the CIGMA service can match the desire of the online customers for the fast response as well as provide high throughput to process internet-scale customers. This paper presents the system architecture and related issues; real-time update scheme and catalog conversion mechanism. Performance of the CIGMA is also evaluated based on a working prototype system

Keywords

e-commerce, catalog comparison, dynamic content, cache, real-time update

1. INTRODUCTION

Rapid expansion of the Internet introduces a high level of dynamics to business environment. Such an environment may open up an opportunity for merchants to increase their benefits. However, to ordinary customers, it may introduce a high degree of inconvenience due to the increased complexity in their shopping.

The source of such dynamics and complexity and thus potential inconvenience in on-line shopping can be observed in two aspects. First, information on sales items may undergo constant changes. This is so since, in an Internet connected environment, business processes can be promptly summarized and reported, and decisions can instantaneously be reflected. Such a situation may result in very frequent changes in sales conditions. An example can be seen from Cisco's case. By effectively adapting their business processes to the Internet environment, Cisco has shorten the term of their business closing to every six hours. Currently, they are trying to further shorten it to three hours. This means that they can accurately estimate the cost of their products, and hence change the sales conditions, every three hours. However, supposing that the sales conditions for popular goods are changed several times a day, customers may not be sure whether the conditions they saw at the merchant site ten minutes ago are correct. Thus, they can fall in a situation in which making shopping decision becomes very difficult. Second, there could be a number of similar shopping sites on the Internet. For example, there are 7,513 registered sites in the Yahoo booksellers' directory[1]. Thus, it is difficult for online customers to be sure if the sites they have chosen are the best ones. Thus, they may wander around to look for other shopping sites with better sales conditions. To make matters worse, on-line merchants compete to provide better sales conditions than others. Such competition may cause a chain reaction among similar shopping sites within a short time[2]. After all, online shopping under highly dynamic business environment can be described in a sentence: "NO ONE KNOWS HOW TO PURCHASE GOODS UNDER THE BEST CONDITION"

To overcome such inconvenience in online shopping, customers need proper support. However, existing service mod-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC 2003 Melbourne, Florida, USA
Copyright 2003 ACM 1-58113-624-2/03/03 ...$5.00.

[1]From Yahoo Web site(www.yahoo.com) on Mar. 14th, 2002
[2]Price War is the term used in economics to indicate such a chain reaction in price changes [15]

els or systems cannot effectively reflect such a highly dynamic environment and support ordinary online customers. To address the problem, we propose a new EC service called "the aCtive Inventory service for Global eMarket" (CIGMA) [23, 24]. The CIGMA provides catalog comparison services over multiple shopping sites for ordinary online customers. In addition, the CIGMA provides a one-stop shopping opportunity for online customers by mediating purchase transactions between the customers and merchants. A key characteristic of the CIGMA is that the services is based on up-to-date information by reflecting frequent changes in catalog information in real-time. In addition, the service matches the desire of the online customers for the fast response. This is possible since the service is based on the data cached in a high performance catalog caching system.

From a system's perspective, in the core of the CIGMA system[3] is the catalog caching system. It collects and maintains the catalog data from different merchants, and retrieves appropriate ones upon customers' requests. Also, the performance of the CIGMA is largely dependent on that of the catalog cache. In [23], we described the CIGMA focusing on the merchant-side interface including service setup and deployment procedures. In this paper, we further describe the CIGMA along with its high performance caching system.

Price comparison services [1, 2, 3, 4] can partly address the problems arising from the situation where too many similar shopping sites co-exist. It gathers price information for an item from many shopping sites. Then, it provides customers with the comparison information. However, it cannot guarantee the correctness of the comparison information because it generally updates the gathered price information periodically. So, the comparison information may not be up-to-date at the moment of access. In the end, customers need to visit the original sites to check the validity of the given prices. There also exists an approach based on instant gathering of price data [5]. This approach may provide up-to-date price comparison information. However, it may incur a long delay, which most online customers cannot endure.

Based on catalog data cached on a high performance catalog cache, the CIGMA effectively supports customers to shop online even under a highly dynamic e-commerce environment. Customers can easily choose the best sales conditions and do not have to undergo exhaustive surfing over shopping sites. The CIGMA also provides purchase transaction mediation. By using this mechanism, online customers can buy goods from different shopping sites at the CIGMA site without visiting original shopping sites. We believe that customers can save time as well as money. For the merchants, the CIGMA provides them with a chance to increase their business profit. The CIGMA can be considered as a kind of a sales agency that sells merchants' items on behalf of them. The CIGMA is a system that will have high visibility to many customers, and thus provides merchants with an opportunity to be exposed to a large number of customers.

This paper is organized as follows. In Section 2, we briefly overview the design of the CIGMA. The high performance catalog caching system is described in detail in Section 3. In Section 4, the implementation of the CIGMA prototype is described. Based on the implementation, system performance is also discussed. Section 5 discusses related work. in two aspects: EC service model and dynamic data caching techniques. We conclude our work in Section 6. Other technical aspects of the CIGMA can be found in [23, 24].

[3]When required, we distinguish the CIGMA system from the service it provides, i.e., the CIGMA service. Otherwise, we interchangeably use the term "the CIGMA" to mean either the service or the system when there is no confusion.

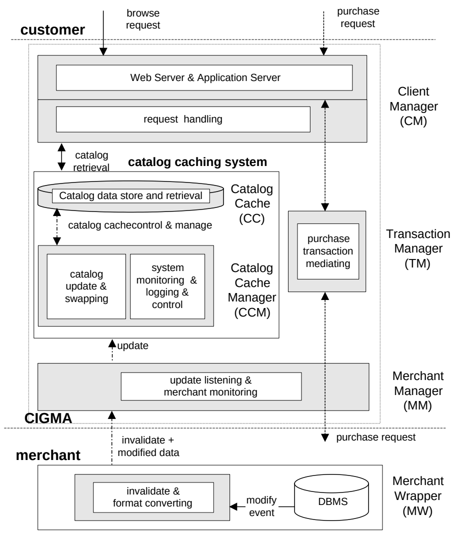

customer
browse request
purchase request
Web Server & Application Server
request handling
Client Manager (CM)
catalog retrieval
catalog caching system
Catalog data store and retrieval
Catalog Cache (CC)
catalog cachecontrol & manage
purchase transaction mediating
Transaction Manager (TM)
catalog update & swapping
system monitoring & logging & control
Catalog Cache Manager (CCM)
update
update listening & merchant monitoring
Merchant Manager (MM)
CIGMA
invalidate + modified data
purchase request
merchant
invalidate & format converting
modify event
DBMS
Merchant Wrapper (MW)

Figure 1: The CIGMA system architecture with service processing flows

2. OVERVIEW OF SYSTEM DESIGN

The CIGMA system has a modular structure. It consists of five server components and one remote component. The remote one, called Merchant Wrapper (MW), runs on each merchant server. The server components are Client Manager (CM), Merchant Manager (MM), Transaction Manager (TM), Catalog Cache (CC), and Catalog Cache manager (CCM). Figure 1 shows the CIGMA system along with a merchant's and a customer's. The function of each component will be described in Section 2.2.

2.1 Internal Service Flows

There are three kinds of external services in the CIGMA: catalog browse, item purchase, and catalog update. The first two are for customers and the last, for merchants with each request being handled differently. The internal service flows for these requests are shown in Figure 1.

The catalog browse request is the most frequent one. It is first received by the Web server, which is a part of the CM. The CM then parses the request and constructs an equivalent query string. It sends the query string to the CC to retrieve the requested information such as a catalog comparison table, an entire catalog, an item category list, etc. Lastly, it replies to the request with an HTML page which is dynamically generated with the query result.

To handle a purchase request, a safety mechanism is required because it generally includes important information such as a credit card number, address, phone number, etc. To secure the purchase transaction processing, the CM communicates with customers using the secure HTTP protocol. Then, it forwards the received data to the TM. The TM also forwards the data to a proper merchant server at once. The result of the transaction is delivered from the merchant server to the customer in the opposite direction.

In the case of an update request, immediate processing is important. Each modification of source data at the merchant's DBMS is detected by the MW module. Then, the module constructs and sends an update request message to the MM at once. The MM forwards the message to the CCM after verifying the integrity of the message. Then, the CCM actually updates the cached data of the CC by composing a query string based on the message. After the CC update, the return value of the update operation (i.e., OK or NOT_OK) is forwarded to the MW in the reverse order.

2.2 Major Components

2.2.1 Client Manager (CM)

The CM takes charge of every communication with customers. It includes a Web server and a Web application server. The Web application server is required to generate responses with dynamic contents[4]. Via these servers occurs each interaction with customers. The CM receives both catalog browse requests and purchase requests from customers.

2.2.2 Merchant Manager (MM)

The MM manages most interactions with merchants. Upon a catalog modification, the MM receives an invalidation message from a merchant server. Then, the MM immediately notifies the CCM of the event to request the update of the CC.

The merchant server monitoring should be noted. There could be a certain situation such as network partitioning, merchant server failure and network congestion, in which the MM cannot receive invalidation messages. This may result in consistency problem in cached catalogs. To avoid this problem, the MM monitors heart beat messages from merchants to check the status of servers. If a merchant sends no messages during a predefined period, the MM notifies the CCM of the situation. Then, the CCM will initiate a suitable action. For example, the CCM may invalidate all the catalogs from the unreachable merchant server until the server responds again. A merchant is forced to send an empty message at the end of the predefined interval if there is no update.

2.2.3 Transaction manager (TM)

The TM mediates purchase transactions. It does not process the transaction on behalf of merchants, but simply relays all the purchase-related information, i.e., order form, purchase order data, and transaction result, etc., between a customer and a merchant site. This approach is taken to avoid the complication and overhead which may be incurred by related business issues.

The TM mediates purchase transactions in the order of time. An assurance for ordered processing is important when many requests arrive within a short period for a few popular goods with insufficient supply. A fair ordering is guaranteed by tagging messages with its original arrival time at the CM. A purchase request is processed according to the tag until the processing is completed.

2.2.4 Catalog Caching System (CCS)

Catalog Caching System is the most important and complex component in the CIGMA system. It is composed of Catalog Cache (CC) and Catalog Cache Manager (CCM). The CC stores and retrieves cached catalog data. Generally, the performance of storage module is critical to the overall caching performance. The CIGMA is expected to process a much higher number of update requests as well as customer requests than general shopping sites. Therefore, the performance of the CC, which store and manage cached catalogs, is even more critical to our catalog caching system. The CCM has three main functionalities: updating cache data, monitoring and controlling the system, and swapping catalog. The Catalog Caching System is described in detail in Section3.

2.2.5 Merchant Wrapper (MW)

The MW covers all the required works for a merchant to interact with the CIGMA. As shown in Figure 1, the MW is deployed on a merchant server for the merchant to join the CIGMA service. Two main functions of the MW are the update propagation and the heart beat message transmission.

The update propagation is performed according to the real-time update scheme, which is described in section 3.1. The MW also periodically sends out heart beat messages to the CIGMA, if there is no update event during a predefined interval. Although the heart beat message carries no information, the message itself is important because it notifies the CIGMA of the fact that the merchant server is alive. If there are neither update messages nor heart beat messages, the CIGMA regards the merchant as having some critical problems. Thus, the CIGMA can invalidate all catalog data from the merchant to avoid providing catalog data that may be stale.

3. CATALOG CACHING SYSTEM

As the CCS is a source data cache, it stores the source data in unit of database fields such as price, model, stock status, etc., as are saved in merchants' databases. From these source data, a catalog page is generated by Web applications upon each customer's request. Other approaches exist which cache catalog data in the form of a HTML/XML pages or fragments [16, 18, 19, 22] or a query result [27, 29, 30]. However, the source data caching is much more efficient especially in maintaining cached data up-to-date. For instance, the CC can invalidate only the modified fields when a catalog is changed at a merchant site. If other approaches were taken, it would have to invalidate all the related objects including the modified fields. As a matter of fact, source level caching is rather indispensable for the CIGMA and its performance. This is because field level operations over cached catalogs, like sorting according to a specific database field, are essential in providing the CIGMA service.

To manage a large volume of catalog data efficiently for the CIGMA service, a catalog cache requires some basic functionality of DBMS. However, conventional disk based databases are too heavy and complex to provide a high throughput required for an Internet cache. Thus, we designed the CC to be highly light-weighted. We use main memory as the primary storage. The use of main memory significantly improves the performance because, in general, a disk I/O is a fairly heavy operation. Thus, the CC can be considered to be a kind of a main memory DB. In addition, the following characteristics have been added to further im-

[4]Popular web application servers are JAVA servlet, common gateway interface (CGI), active server page (ASP), etc.

prove the performance.

First, to be further light-weighted, the CC provides only some core functions such as storing, retrieval, indexing, etc, which are required for catalog caching, and does not support many complex functions common to full-fledged DB's. This is possible because the CC is designed to be used as a cache rather than a permanent storage. For example, it does not have the persistency-related functions such as support for the durability property, failure recovery, logging and roll-back operation, etc. This is not a problem in the CIGMA because the data are replications that have a persistent copy in the original merchant servers. Therefore, when an error occurs in a cached catalog, the catalog may be recovered by retrieving it again from merchant servers.

Second, the CC is further optimized for the situation where there are a lot of update requests. Merchants may frequently evaluate their business situations and continuously change their prices and other sales conditions. This is highly possible in the globally connected Internet business environments as shown in the motivating example and the CISCO's case. Thus, the CIGMA will encounter very frequent update requests from a lot of merchants. The CC assigns higher priorities to update requests so that the request dispatcher gives higher preferences to them than the others in the request queue. In addition, once started, an update transaction is processed exclusively to its completion and is not blocked by any other operations. Therefore, very fast processing is assured for update requests. Also, the CC is designed to utilize its resources efficiently by avoiding unnecessary operations such as context switches.

The CCM takes charge of updating cache data. Upon update requests from MM, the CCM composes a proper query string based on the incoming request. Then, it updates cache data by sending the query to CC. After the update, the return value of the update operation (i.e., OK or NOT_OK) is forwarded to the MW through the MM.

The CCM monitors the status of the system and controls system components. For example, the CCM monitors the frequency of the requests from customers and merchants and collects its statistics. Based on the collected data, it controls major facilities of the components to improve the performance.

3.1 Real-time update mechanism

It is important to keep the data in the CC up-to-date in the CIGMA service. Thus, any modifications in catalog data in merchant databases should be promptly reflected to those in the CC. In addition, the update mechanism should be efficient since a high number of update requests are expected.

The update scheme is based on server invalidation. The merchant server instantly identifies any modification in the database, and initiates an update in the cache by sending out an invalidation message. Thus, an update is propagated to the CIGMA with very small delay.

The instant identification of update is done based on trigger mechanisms in merchant databases. Using a trigger mechanism, the update process can be done very efficiently. It is so since, via the mechanism, changes can be detected in the unit of a field and thus, fine-grained invalidation is possible. Trigger mechanisms are provided in most popular database systems such as Oracle [6], DB2 [7], Sybase [8], MySql [9], PostgresSQL [10], etc.

Currently, time-to-live (TTL) based schemes are most popularly used as cache consistency mechanisms in the Internet [25]. However, TTL-based schemes are not proper for the CIGMA service since they cannot quickly propagate updates to a cache. Prompt propagation of updates may be achieved if a cache frequently polls changes in servers in a very small interval. However, this will incur excessive overhead to the cache. On the contrary, server-push style approaches can more quickly reflect changes in original data. However, it has been reported that the update schemes of server-push style experience heavy server-side load [17]. This load comes from the fact that a server should keep track of all the caches which hold copies of its data per data item basis. That is, the server has to handle a very high number of caches. However, the CIGMA can avoid this burden. As the CIGMA service requires an explicit permission from a merchant, the merchant server can control the number of contracted caches.

Figure 2 shows the detailed structure of the MW and the whole process from DB modification to sending an invalidation message. When an update event occurs at the merchant DB (1), the Update Trigger [5] is automatically called by the merchant's DBMS and the Update Trigger sends the modified data to the Event Reporter (2). The Event Reporter sends the modification information to the MW that is running on the merchant server (3). The modification information includes the table ID, the primary key, the field name, and the modified field value. The Event Listener within the MW receives the information. Catalog Converter converts the category as well as the format of the modification information, if needed (4) referring Catalog Mapping Table (5). The Catalog Converter composes an invalidation message (6). Lastly, the Communication Module sends the message to the CIGMA (7). The MM forwards the received message, comprising the invalidation message and modified data, to the CCM (8). Lastly, The CCM take charge of constructing an proper query message based on the received message and committing the update transaction on the CC (9).

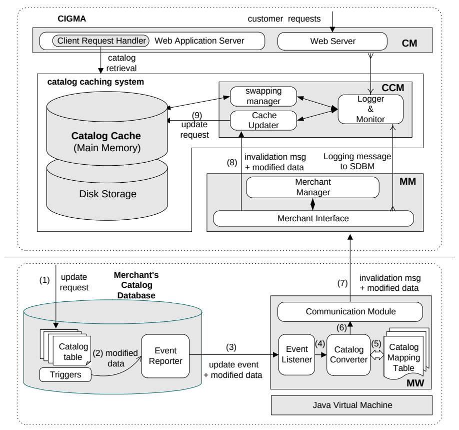


Figure 2: Update process and the structure of the catalog caching system of the CIGMA including the Merchant Wrapper.

When sending an invalidation message, we piggyback the message with the modified field and value. Thus, an update

[5]The Update Trigger is a kind of DB trigger that is constructed by using SQL statement. It is installed in the merchant DB in advance

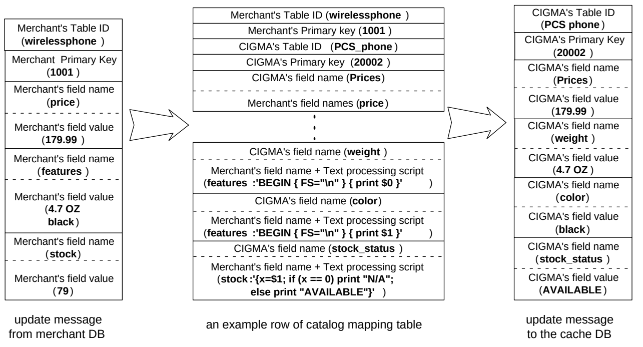
can be completed with one message containing both invalidation and modification information. Most of modifications will occur in the fields of small sizes, such as item prices and inventory information. Therefore, the piggybacking effectively reduces the update delay and improves processing efficiency.

The update mechanism can further enable merchants to control the the level of visibility for their catalogs to customers. This visibility control is essential for merchants, since complete transparency about their business activities may not be preferred by merchants. The CIGMA also allows merchants to specify the update interval of their catalogs. For example, merchants can have update events to be propagated to the CIGMA immediately, or in every hour or day. Merchants can also set update conditions for their catalogs, so that a modification of a catalog is propagated to the CIGMA only when the update condition is satisfied. The catalog conversion capability described in Section 3.2 may be used to control the visibility level too.

3.2 Catalog Conversion

The CC should integrate catalogs from different merchant sites. Merchants generally use different catalog formats and item categories from each other. The CIGMA also has its own catalog format and category. Therefore, merchants' catalogs have to be re-categorized and their formats, converted to match those of the CIGMA.

For this conversion, the CIGMA uses the Catalog Mapping Table (CMT). The CMT is a table which contains all description for catalog conversion and re-categorization. That is, the exact specification of the conversion is done on the CMT before starting a service and once a service is started, the conversion is automatically processed. The usage of the CMT is not only for a merchant server to convert its catalogs to those of the CIGMA. In converse, the CIGMA also uses the table when it needs to retrieve a field from the merchant DB.


Merchant's Table ID (wirelessphone)
Merchant Primary Key (1001)
Merchant's field name (price)
Merchant's field value (179.99)
Merchant's field name (features)
Merchant's field value (4.7 OZ black)
Merchant's field name (stock)
Merchant's field value (79)
Merchant's Table ID (wirelessphone)
Merchant's Primary key (1001)
CIGMA's Table ID (PCS_phone)
CIGMA's Primary key (20002)
CIGMA's field name (Prices)
Merchant's field names (price)
CIGMA's field name (weight)
Merchant's field name + Text processing script (features :'BEGIN { FS="\n" } { print $0 }')
CIGMA's field name (color)
Merchant's field name + Text processing script (features :'BEGIN { FS="\n" } { print $1 }')
CIGMA's field name (stock_status)
Merchant's field name + Text processing script (stock :'{x=$1; if (x == 0) print "N/A"; else print "AVAILABLE"}')
CIGMA's Table ID (PCS phone)
CIGMA's Primary Key (20002)
CIGMA's field name (Prices)
CIGMA's field value (179.99)
CIGMA's field name (weight)
CIGMA's field value (4.7 OZ)
CIGMA's field name (color)
CIGMA's field value (black)
CIGMA's field name (stock_status)
CIGMA's field value (AVAILABLE)
update message from merchant DB
an example row of catalog mapping table
update message to the cache DB

An example of catalog conversion and re-categorization - table at the center is a CMT's row : (bold type characters means that the value of that field) - note that merchant's *features* field is splited into two fields and *stock* field is transformed into a different type at the message to CIGMA . It also should be noted that re-categorization is occurred implicitly by giving a new table id and a primary key

Figure 3: An example of Catalog Mapping Table

The CMT includes a merchant DB's table name and primary key as well as those of the CC. Then, it specifies, using an AWK-like[6] script language, how field names and values

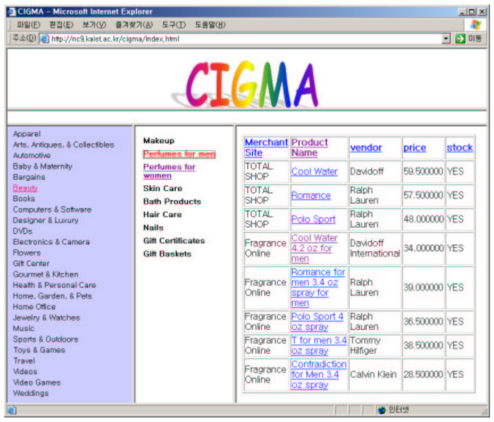
[6]AWK is a popular utility in UNIX environment. It enables us to specify what kind of data you are interested in and the operations to be performed when that data is found.

in a merchant catalog database is changed to those of CC. Figure 3 shows a row of a CMT example with an incoming update message from a merchant DB and an out-going update message to the cache, respectively. It shows that the item "wirelessphone" is recategorized into the item, "PCS phone". In the figure, the field *features* in the merchant DB is split into two fields: *weight* and *color*. It also should be noted that the type of field *stock* in the incoming message can be transformed into a semantically different type in an out-going message. This semantic transformation is especially important since it enable merchants to control the external appearance of their catalogs.

4. IMPLEMENTATION & PERFORMANCE EVALUATION

4.1 Implementation

We have implemented a prototype of the CIGMA system and two sample shopping sites.[7] Figure 4 shows an snapshot of the CIGMA Web site. Details about an example service including sample shopping sites can be seen in [23]. The CIGMA and merchant systems have been developed and are operating on Linux. GNU C++ compiler have been used to implement most of the components of the CIGMA, and JAVA servlet and JAVA Server Page (JSP) have been used for Web interface programming. Most of the core components have been implemented except for a few elements such as error handling and security and authentication.



Merchant Site	Product Name	vendor	price	stock
TOTAL SHOP	Cool Water	Davidoff	59.500000	YES
TOTAL SHOP	Romance	Ralph Lauren	57.500000	YES
TOTAL SHOP	Polo Sport	Ralph Lauren	48.000000	YES
Fragrance Online	Cool Water 4.2 oz for men	Davidoff International	34.000000	YES
Fragrance Online	Romance for men 3.4 oz spray for men	Ralph Lauren	39.000000	YES
Fragrance Online	Polo Sport 4 oz spray	Ralph Lauren	36.500000	YES
Fragrance Online	T for men 3.4 oz spray	Tommy Hilfiger	38.500000	YES
Fragrance Online	Contradiction for Men 3.4 oz spray	Calvin Klein	28.500000	YES

Figure 4: Snapshot of the CIMGA Web site.

The current implementation of the CM uses Apache Web server and Tomcat as the Web and application servers. It also includes several handlers for each request type, such as category list, catalog browse, catalog comparison, item purchase, etc.

In many cases, an HTML page includes embedded data such as images, sounds, and video. Web pages with such embedded data require careful handling because it has a significant impact on performance. Such media data generally requires a lot of memory space in the cache. Also, under

[7]You can view the CIGMA and the two sample sites. The URL of the CIGMA is http://nc9.kaist.ac.kr/cigma/. The two sample merchant sites can be accessed via http://nc2.kaist.ac.kr/merchant/ and http://nc2.kaist.ac.kr/merchant2/.

the current HTTP protocol, each embedded URL requires separate processing (e.g., a separate access to a storage, and even a separate TCP connection in the case of HTTP 1.0) and cannot be transferred along with the including HTML file.

For an efficient retrieval of embedded media data, a separate multimedia data handler has been implemented. The multimedia data handler prefetches the embedded media files when the container HTML data is retrieved from the CC. That is, the CIGMA retrieves all data required to construct the requested catalog via a single access to the CC upon the initial request. Then, the pre-fetched media data is temporarily stored in the main memory and transferred to a customer at the succeeding requests to the media files. In this way, the number of CC accessing operations is reduced. Also, the response time to a customer's request is shortened.

A simpler way to handle embedded media data is to avoid caching them and to have embedded data retrieved from the original merchant servers. This will incur little overhead to the CIGMA and greatly save its main memory space. However, clients may experience fluctuation in response times since merchant servers may have throughput different from each other. Thus, a decision on whether or not caching media data needs to be made by considering the resource situation of the CIGMA and a merchant.

Most of the CCM's functions, including system monitoring and logging, catalog swapping and update buffering, have been implemented. However, monitoring and logging and catalog swapping have been partially implemented. Although a full implementation of the functions is not difficult, we currently monitor only the number of client requests and update requests to determine the popularity of each catalog. Based on this information, the current prototype uses a Least Frequently Used (LFU) based cache replacement algorithm for catalog swapping. We believe that further study is required to optimize the algorithm. The current version of the CC has been implemented by customizing a third-party main memory database,"FastDB" [11]. This helps us quickly implement the prototype. However, it leaves room for much improvement in performance. We plan to further improve the CC in the next version of the system. We used JAVA technology to implement the MW module to address the heterogeneity of the running platforms of merchants. To help merchants set up DB triggers, we plan to provide templates and samples of the triggers for different DB's. For the time being, those for the Oracle DBMS are provided.

4.2 Performance Evaluation

We are currently making various performance measurements for the CIGMA system. We plan to describe the details of the system performance in a follow-up report. In this paper, we give a brief report on the throughput of the CIGMA system just to assist in a high level understanding of the system performance. We report the throughput of the system under catalog update requests and that under catalog browse requests. We think that another measurement that is important in understanding the performance of the CIGMA is the update delay. Measurement of the update delay will be included in the follow-up performance report.

The reported measurement was done on a single-node deployment. That is, all components of the CIGMA reside in a single machine. This is mainly to give an understanding of the basic performance through a brief presentation here. Under multi-node deployment, we can expect much higher performance numbers than those reported here. The node has a Pentium III 1Ghz CPU and 2GB of main memory. The Red Hat Linux 7.2 was used as the operating system. Apache Web server 1.3.20 and Tomcat 3.2.3 were used as the Web and application servers. Sun JAVA SDK 3.1 was used as the Java virtual machine. As the CIGMA's catalog cache, FastDB version 2.37 was used. The CIGMA server and other systems used to emulate merchants or customers were connected via a 100MB local Ethernet. To test the performance under customer's browse requests, we used the Httperf [21]. However, to measure the update performance, we built, on our own, a simple utility called an *update request generator*.

Catalog Update

The throughput under catalog update requests is measured as a function of update message size. The throughput is also measured while the number of concurrent threads is changed. Figure 5 shows the throughput when the number of concurrent threads is 1, 5, 10, 50, and 100. It shows that the CIGMA can process nearly 600 update requests per second when the size of update message is less than 1 Kbytes. The performance degrades to 500/s as the message size increases to 10 Kbytes. However, we think that the throughput of 600/s is quite realistic since updates usually occur in small sizes, e.g., changes to stock status or prices.

It is shown in Figure 5 that the number of concurrent merchants has little effect on update performance when the number is five or more. However, when a single thread is used, the throughput is smaller than in the other cases. This is because a thread sends a new update request only after receiving a response for the previous one. Also shown in the Figure is that the system throughput decreases very slowly as update message size increases linearly. (Note that the horizontal axis is in log scale.) We conjecture that the throughput is affected more seriously by the TCP connection overheads. If this is the actual case, the update throughput can be distinctively increased if persistent connections are used for updates. (This was partially conformed by other experiments that we don't report here.)

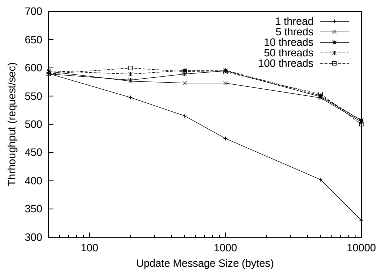
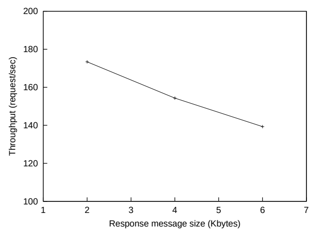


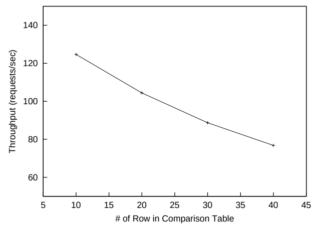
Figure 5: Update performance as the function of update message size

Catalog Browsing and Catalog Comparison

Figure 6 (a) shows the throughput of catalog browsing requests as a function of catalog size. The CIGMA can process 173 requests per second when the catalog size is 2 Kbytes. It degrades to 154 and 139 when the catalog size increases to 4 and 6 Kbytes, respectively. The throughput under catalog comparison requests is measured as a function of requested table size. Figure 6 (b) shows that the throughput is from 125 to 77 when the number of items in the comparison table varies from 10 to 40.



(a)



(b)

Figure 6: (a) Performance of item browse requests in the function of catalog size (b) Performance of catalog comparison table as the function of result table size

The throughput under catalog comparison requests is smaller than that under catalog browse requests. We conjecture that the performance difference comes mainly from the difference in the overhead to access databases. Generally, to construct a comparison table, a separate database access is required for each row. From our informal observation, the overhead incurred by the Web server and the application server was similar in the two cases. Also, the response HTML file size was smaller in the case of catalog comparison requests than that of catalog browse requests.

5. RELATED WORK

5.1 EC service models

In the view of the EC service model, price comparison services [1, 2, 3] are similar to our work. They provide catalogs for an item by gathering them from many merchant sites. However, they are not supported by an automated real-time update scheme. Many services update their data manually or use schemes based on periodic or aperiodic polling [2, 3, 1]. Compared to the CIGMA's case, we may say that those services help online customers in a best-effort style.

AddAll [5] provides price comparison service based on instant catalog searching and gathering upon customer's requests. Thus, it can provide fresh information about an item. However, it also has a serious problem: the response time may be very long because it has to visit many shopping sites in an on-demand fashion. In addition, it issues multiple Web requests per each customer request and thus may cause a heavy traffic on the Internet.

These price comparison services are not a real shopping service. Neither of them have purchase transaction mediation functionality. Thus, customers have to visit the original shopping site to buy a selected item.

The B2B marketplaces [12, 13] intermediates between customers and merchants, and provides a set of services to support on-line purchasing. But, contrary to the CIGMA, most B2B marketplace model is designed and available only for business customers and/or transactions of a large volume. In addition, since B2B marketplaces do not deal with ordinary customers directly, performance concern is less serious in their design.

5.2 Techniques to cache dynamic data

Recently, a number of researches have proposed techniques for dynamic data caching [16, 29, 18, 19, 26, 30, 14]. These techniques have been proposed mainly as the scalability solution for ordinary Web services, noting that the generation of dynamic data becomes a major bottleneck. The CCS is different from those because it focuses on the provision of new service, i.e., cross-organizational data services, based on the cached information. The CCS is also different from others in that other caches can be considered as reverse proxies that are used within the contexts of specific servers, whereas the CCS is closer to a proxy that operates along with a number of multiple merchant sites. Below, we classify related techniques in terms of caching units.

HTML page caching stores HTML pages generated upon client's requests [22, 16]. In the view of cache hit gain, i.e. cost saving upon a cache hit, this approach is most advantageous; it saves the cost of query processing as well as that of HTML page generation. However, this approach lacks flexibility; for instance, it is not useful in caching personalized Web pages. Hit ratios for personalized pages will be very low because only specific clients will access those pages. In addition, modification to a common part may result in update of numerous pages.

XML/HTML fragment caching stores XML or HTML fragments which are parts of generated HTML pages. The system proposed in [19] provides an algorithm for efficient update propagation to HTML fragments stored in cache. However, this system requires an administrator to map the relationships between the updates and the fragments affected by their updates. Fragment caching can provide more flexibility than HTML page caching.

Query result caching stores query results in caches. The advantage lies in that it removes the query processing step. Form-based Proxy Caching [28] and DBAccel[20] come under this category.

The caching schemes summarized above can be called *derived data caching*. On the contrary, the CCS is a *source data cache*. As mentioned, the source level caching is effective in providing advanced services, as shown in the CIGMA service, since fine-grained operation over the cached data is possible. More importantly, source data caching significantly reduces the cost of keeping data up-to-date. In the case of derived data caching, the semantic information of derived data may become different from that of the original data. Thus, it is difficult to automatically find derived data affected by data updates in the original servers. However, source level caching does not reduce the cost of the query processing, rather it moves the cost of the query processing from back-end servers to caches.

6. CONCLUSIONS

Despite many advantages of the EC, online shopping is an overloaded activity for most ordinary customers due to a highly dynamic e-commerce environment. For instance, they are not sure if a chosen sales condition is really a good one even after exhaustive Web surfing. The CIGMA addresses the problems in on-line shopping under such a dynamic environment. It suggests the best sales conditions over multiple shopping sites and provides a convenient shopping environment for customers. Thus, customers can save time and money by using the service. The CIGMA also helps merchants increase their business profit because the CIGMA can act as a sales agency.

An architecture of a new catalog cache system, CCS, is proposed which comprises the core of the CIGMA system. Technical components of the CCS are discussed and its performance is evaluated based on a prototype system. We believe that the proposed architecture can serve as example of an Internet cache for advanced Internet services. For an advanced service such as the CIGMA, the ordinary architecture such as proxy or reverse proxy is not appropriate. To match the high performance required in the CIGMA and other advanced services, it is designed as a light-weight main memory cache. To further improve the performance and flexibility, it caches catalog data in source level. It also provides a real-time update mechanism.

7. ACKNOWLEDGMENTS

This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center(AITrc).

8. REFERENCES

[1] http://www.nextag.com/,NexTag - The Search Engine for Shoppers.

[2] http://www.mysimon.com/,mySimon - Compare products and prices from around the Web.

[3] http://www.dealtime.com/,DealTime - compare products, prices & stores.

[4] http://www.pricescan.com/,PriceSCAN - Comparison Shop for the Lowest Prices.

[5] http://www.addall.com/,AddALL - Book Search and Price Comparison.

[6] http://otn.oracle.com/doc/oracle8i_816/appdev.816/a76939/adg13trg.htm#376, Oracle8i Application Developer's Guide - Fundamentals Release 2 (8.1.6).

[7] http://www-4.ibm.com/software/data/db2/udb/ad/v7/adg/db2a0/frame3.htm#trigger,DB2 Application Development Guide: Using Triggers in an Actie DBMS.

[8] http://my.sybase.com/detail?id=1355,Sybase Manual: Triggers.

[9] http://www.mysql.com/doc/A/N/ANSI_diff_Triggers.html,MySQL manual: 1.7.4.4. Stored Procedures and Triggers.

[10] http://www.postgresql.org/idocs/index.php?plpgsql-trigger.html,PostgreSQL 7.1 Documentation: Chapter 24. PL/pqSQL-SQL Procedural Language.

[11] http://www.ispras.ru/ knizhnik/fastdb.html,FastDB - Main Memory Relational Database Management System.

[12] http://service.ariba.com/, Ariba Supplier and Buyer Network.

[13] http://www.biz2biz.com/Marketplace/, Biz2Biz B2B Marketplace.

[14] A. Brown, G. Kar, and A. Keller. An active approach to characterizing dynamic dependencies for problem determination in a distributed environment. In *Proc. of IFIP/IEEE International Symposium on Integrated Network Management*, pages 377 – 390, May 2001.

[15] L. M. B. Cabral. *Introduction to Industrial Organization* . MIT Press, 2001.

[16] K. S. Candan, W.-S. Li, Q. Luo, W.-P. Hsiung, and D. Agrawal. Enabling dynamic content caching for database-driven web sites. In *Proceedings of SIGMOD'2001*, pages 532 – 543, May 2001.

[17] P. Cao and C. Liu. Maintaining strong cache consistency in the World Wide Web. *Computers, IEEE Trans on*, 47(4):445–457, April 1998.

[18] J. Challenger, A. Iyengar, and P. Dantzig. A scalable system for consistently caching dynamic web data. In *Proceedings of INFOCOM '99*, pages 294–303, 1999.

[19] J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed. A publishing system for efficiently creating dynamic web content. In *Proceedings of INFOCOM '00*, pages 844–853, 2000.

[20] S. Choi, J. Lee, H. Cho, J. S. Junghoon Kim, and Y. Lee. *DBAccel: A Light-weight Internet Cache for Accelerating Database-backed Web Sites.* unpublished working paper.

[21] David Mosberger and Tai Jin. httperf: A Tool for Measuring Web Server Performance. *Performance Evaluation Review*, 26(3):31–37, December 1998.

[22] V. Holmedahl, B. Smith, and T. Yang. Cooperative caching of dynamic content on a distributed web server. In *Proc. of the 7th IEEE International Symposium on High Performance Distributed Computing*, pages 243–250, 1998.

[23] S. M. Kim, S. Kang, H.-K. Lee, and J. Song. CIGMA: aCtive Inventory service in Global e-Market for enabling one-stop shopping over Internet shopping sites. In *to appear in Proceedings of the 3rd International Symposium on Electronic Commerce (ISEC-02).*

[24] S. M. Kim, S. Kang, H.-K. Lee, and J. Song. CIGMA: aCtive Inventory service in Global e-MArket for enabling one-stop shopping over internet shopping sites. Technical Report CS-TR-2002-178, Korea Advanced Institute of Science and Technology (KAIST).

[25] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice - HTTP/1.1, Networking Protocols, Caching and Traffic Measurement.* Addison-Wesley, 2001.

[26] A. Labrinidis and N. Roussopoulos. Webview materialization. In *Proceedings of SIGMOD'2000*, pages 32 – 35, May 2000.

[27] Q. Luo, S. Krishnamurthy, C. Mohan, H. Pirahesh, H. Woo, B. G. Lindsay, and J. F. Naughton. Middle-tier database caching for e-business. In *Proc. of SIGMOD 2002*, June 2002.

[28] Q. Luo and J. F. Naughton. Form-based proxy caching for database-backed web sites. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, September 2001.

[29] Qiong Luo and Jeffrey F. Naughton and Rajasekar Krishnamurthy and Pei Cao and Yunrui Li. Active query caching for database web servers. In *Proceedings of WebDB'2000*, pages 32 – 35, 2000.

[30] K. Yagoub, D. Florescu, C. C. Andrei, and V. Issarny. Building and customizing data-intensive web site using weave. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, September 2000.