CIGMA: Active Inventory Service in Global E-Market Based on Efficient Catalog Management**

Su Myeon KIM††, Student Member, Seungwoo KANG†, Heung-Kyu LEE†, and Junehwa SONG††∗, Nonmembers

SUMMARY A fully Internet-connected business environment is subject to frequent changes. To ordinary customers, online shopping under such a dynamic environment can be frustrating. We propose a new E-commerce service called the CIGMA to assist online customers under such an environment. The CIGMA provides catalog comparison and purchase mediation services over multiple shopping sites for ordinary online customers. The service is based on up-to-date information by reflecting the frequent changes in catalog information instantaneously. It also matches the desire of online customers for fast response. This paper describes the CIGMA along with its architecture and the implementation of a working prototype.

key words: electronic commerce, information aggregation, heterogeneous information, dynamic content, intelligent cache

1. Introduction

The Internet is significantly altering the way business is conducted, opening up many challenges and opportunities to both merchants and customers. One of the notable characteristics of a fully Internet-connected business world is that many aspects of a business environment will be subject to frequent changes. Merchants are actively adapting to this highly dynamic environment. However, to ordinary customers, these dynamics are a serious source of inconvenience; they do not feel comfortable shopping in a market where the status of an item, e.g., inventory, delivery condition, price, etc., may change several times a day. Customers will want services to overcome such difficulties and to benefit from the situation. However, existing service models and systems do not effectively reflect such environments and assist ordinary customers.

To discover potential inconvenience in online shopping, let’s consider the source of such dynamics from the ordinary customers’ point of view. First, information on sales items may undergo constant changes. The case of Cisco presents a good example. By effectively adapting their business processes to the Internet environment, Cisco has shortened 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. 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 may fall in a situation where making shopping decision becomes very difficult. Second, there could be a number of similar shopping sites on the Internet. For example, there are 6,514 registered sites in the Yahoo bookstores’ directory***. Thus, it is difficult for online customers to be sure if the site they have chosen is the best one. Thus, they may wander around to look for other shopping sites with better sales conditions. To make matters worse, online merchants compete to provide better sales conditions than others. Such competition may cause a chain reaction among similar shopping sites within a short time****. After all, online shopping in the highly dynamic business environment can be described in a sentence: “NO ONE KNOWS HOW TO PURCHASE GOODS WITH THE BEST CONDITION.”

To address the problem of online shopping under such a dynamic e-commerce environment, we have proposed a new EC service called “the aCtive Inventory service for Global eMArket” (CIGMA) [27]. The CIGMA provides catalog comparison services with one-stop shopping mechanism over multiple shopping sites for ordinary customers. An important characteristic of the CIGMA is that the service is based on up-to-date information by reflecting frequent changes in catalog information instantaneously. In addition, the service matches the desire of the online customers for fast response. This is possible since the service uses a high performance catalog caching system.

From a system’s perspective, the core of the CIGMA system***** is the catalog caching system. It collects and maintains catalog data from different merchants, and retrieves appropriate ones upon customers’ requests. Also, the performance of the CIGMA is largely dependent on that of


****Price War is the term used in economics to indicate such a chain reaction in price changes [17].

*****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.

Manuscript revised June 28, 2005.

††The authors were with the Department of EECS [CS Div.], Korea Advanced Institute of Science and Technology, 373–1 Kusongdong Yusong-gu, Taejon, 305–701, Korea.

†††The author was with IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, N.Y. 10598.

∗Presently, with Korea Advanced Institute of Science and Technology, Korea.

∗∗A short version of this paper has been published in ACM SAC 2003 [25].

a) E-mail: smkim@nclab.kaist.ac.kr

DOI: 10.1093/ietisy/e88–d.12.2651
the catalog cache. In [27], 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]–[5] can partly address the problems arising from the situation where too many similar shopping sites coexist. They gather price information on specific items from many shopping sites. Then, they provide customers with the comparison information. However, they cannot guarantee the correctness of the comparison information because they generally update 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 [6]. This approach may provide up-to-date price comparison information. However, it may accompany a long delay, which is intolerable to most online customers.

With catalog data cached on a high performance cache, the CIGMA effectively supports customers in shopping online even under a highly dynamic e-commerce environment. Customers can easily choose the best sales condition without exhaustive surfing over shopping sites. The CIGMA also provides purchase transaction mediation. By using this mechanism, online customers can buy goods offered by different shopping sites at the CIGMA site without visiting original shopping sites. We believe that customers can save both time and money.

For merchants, the CIGMA provides a chance to increase their business profit with little efforts. The CIGMA is considered as a kind of a sales agency that sells merchants’ items on behalf of them. The CIGMA is an e-commerce service that will attain high visibility to many customers, and thus provides merchants with an opportunity to be exposed to a large number of customers. In addition, by joining the CIGMA, merchants can be free from the duty of update management. Currently, in many shopping agent services, consistency management is up to individual merchants. Although some consistency management tools are provided, human intervention is indispensable in most cases. Automating the update process, the CIGMA allows merchants to focus on their original business and react quickly to the market’s needs in a timely manner.

1.1 High Performance Catalog Caching System

For an advanced service like the CIGMA, ordinary cache architectures such as proxies or reverse proxies are not appropriate. They are initially designed to cache static content such as HTML files and images and are not suitable for caching dynamic contents like item catalogs. Existing solutions for dynamic data caching [16, 18, 20, 21, 29, 34, 36] are not appropriate, either. These techniques have been proposed mainly as scalability solutions for ordinary Web services, noting that the generation of dynamic data becomes a major bottleneck. The situation in the CIGMA is different in that it focuses on the provision of new, value-added services, i.e., cross-organizational data services, based on cached information. Also, the aforementioned techniques can be considered as reverse proxies that are used within the contexts of specific servers, whereas the catalog cache in the CIGMA is closer to a proxy that operates with a number of multiple merchant sites. Below, we describe the novel characteristics of the proposed catalog caching system. We believe that the proposed architecture can serve as a reference for an Internet cache for advanced services.

First, it is a high performance cache. Since the CIGMA confronts ordinary customers directly, the performance level of the cache should be very high to process a high rate of requests from lots of customers. In addition, it should be capable of managing a lot of interactions with merchant servers to keep the freshness of cached catalogs. To satisfy such requirements, it caches e-catalogs in the form of source data. That is, the cache stores catalog data in unit of database fields as the data is saved in merchants’ databases. With source data caching, keeping cached data up-to-date can be done efficiently. Also, to manage a large volume of catalog data efficiently, the cache uses main memory as a primary storage. In addition, we try to further improve the system performance via crafted design and implementation.

Second, the cache supports cross-organizational services. To gather and manage different kinds of catalogs from heterogeneous shopping sites, a merchant wrapper is provided. Via the wrapper, the cache is able to deal with each shopping site in a uniform manner. In addition, an integrated schema is constructed to represent product catalogs from different shopping sites. We also identify a set of core tasks, such as catalog browse, update, comparison, which are essential to the CIGMA service. The application level service that is visible to customers is composed of these core tasks.

Third, the cache is equipped with instant update capability. To keep catalog information up-to-date, any modification to catalog data at the merchant DB should be propagated to the cache as soon as possible. Our update scheme is based on server invalidation. In the scheme, upon modification, the server initiates invalidation and modification of the cached data in the cache. In this way, the delay to the cache update can be very short.

This paper is organized as follows. In Sect. 2, we briefly overview the design of the CIGMA. The high performance catalog caching system is described in detail in Sect. 3. In Sect. 4, the implementation of the CIGMA prototype is described. Based on the prototype, system performance is discussed in Sect. 5. Section 6 discusses related work in two aspects: EC service model and dynamic data caching techniques. We conclude our work in Sect. 7. Other technical aspects of the CIGMA can be found in [26].
2. Overview of System Design

2.1 System Architecture

The CIGMA system has a modular structure. It consists of five server components and one remote component. The remote one, called the Merchant Wrapper (MW), runs on each merchant server. The server components are the Client Manager (CM), the Merchant Manager (MM), the Transaction Manager (TM), the Catalog Cache (CC), and the Catalog Cache Manager (CCM). Figure 1 shows the CIGMA system. The function of each component will be described in Sect. 2.3.

Each component is designed as a separate process. For communication among the components, a message-passing-based mechanism is used. Other inter-process communication mechanisms, e.g., shared memory or pipe, may provide better performance when the system is constructed in a single node. However, a message-passing-based approach is more appropriate in terms of scalability and flexibility. For instance, the system can easily be deployed in a multi-node environment without any modifications since message-passing-based mechanisms support inter-node communications. In addition, each component can be modified or upgraded independently as long as the external interfaces are not changed. Providing scalability is very important in the CIGMA service as well as in Internet services in general because such services have to handle a large number of customers and their requests.

Achieving the high performance required in the CIGMA service is a very important objective of our system design. Thus, we define and use simplistic message formats to keep the communication overhead with merchants as low as possible. That is, currently, we do not choose other complex message formats such as an XML-based one, for the sake of performance. Such message formats may be advantageous in terms of flexibility. However, they require additional processing for message parsing and interpretation. Since it is expected that there are very frequent interactions between the CIGMA and merchants, additional costs in the communication are very critical to the overall system performance. For the same reason, our system works directly with TCP/IP for its efficiency. High-layer protocols like HTTP are considered less appropriate. However, our system can easily be extended to incorporate such alternative methods for message formats and communication protocols.

2.2 Internal Service Flows

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

Catalog retrieval requests are the most frequent one. The request 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 by returning a dynamically generated HTML page containing the query result.

To handle purchase requests, a safety mechanism is required since such requests generally include sensitive information such as credit card numbers, addresses, phone numbers, 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 immediately forwards the data to a proper merchant server. The result of the transaction is delivered from the merchant server to the customer in the opposite direction.

For update requests, immediate processing is important. Each modification of source data at the merchant’s DBMS is detected by the MW module. The module then constructs and sends an update request message to the MM immediately. The MM forwards the message to the CCM after verifying the integrity of the message, the CCM then 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.

Fig. 1 The CIGMA system architecture with service processing flows.
2.3 Major Components

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\(^1\). Each interaction with customers occurs via these servers. The CM receives both catalog retrieval requests and purchase requests from customers.

Merchant Manager (MM)

The MM manages most of the interactions with merchants. Upon 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.

Merchant server monitoring should be further considered. There are certain situations such as network partitioning, merchant server failure, and network congestion, in which the MM cannot receive invalidation messages. This may result in consistency problems in cached catalogs. To avoid such problems, the MM monitors heartbeat 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. The CCM then initiates an appropriate 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.

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. Assuring 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.

Catalog Caching System (CCS)

The Catalog Caching System is the most important and complex component in the CIGMA system. It is composed of the Catalog Cache (CC) and the 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 stores and manages cached catalogs, is even more critical to our catalog caching system. The CCM has three main functionalities: updating cached catalogs, monitoring and controlling the system, and swapping the cached data. The Catalog Caching System is described in detail in Sect. 3.

Merchant Wrapper (MW)

Deployed on a merchant server (see Fig. 1), the MW covers all the required works for a merchant to interact with the CIGMA. It provides a uniform interface between the two, resolving the heterogeneity of merchants. The main functions of the MW are update propagation and catalog conversion. Update propagation is performed according to the instant update scheme, which is described in Sect. 3.1. The MW also sends out a heartbeat message to the CIGMA if no update event occurs during a predefined timeout interval. Although the heartbeat message carries no information, the message itself is important because it notifies the CIGMA that the merchant server is alive. Catalog conversion is described in Sect. 3.2. Note that our design to have the catalog conversion performed in the MW, and hence, in the merchant server, is for the performance of the CIGMA, which can be easily heavy-loaded. In this way, a new merchant can join the CIGMA without incurring any additional cost to the CIGMA.

The MW is designed to be dynamically customizable. That is, it works as a generic wrapper template and is used for every merchant. Each wrapper module on different merchant sites configures itself according to a service contract (SC) (See Sect. 2.4), and will act as specified in the SC. Note that the SC is signed by the merchants as well as by the CIGMA. This design of the MW much helps reduce the safety concerns of merchant systems; one-time certification of the safety of the template code will suffice for every merchant.

The MW is implemented in Java. Java-based implementation is advantageous in several ways. First, the module can be installed in any computing environment running the Java Virtual Machine (JVM). Second, faults in the MW do not affect the reliability of merchant’s server system. The faults are propagated only to the virtual machine. Third, by using the powerful access control mechanism of Java, merchants can prevent the MW from accessing their resources. This can be done by having a property file to include only the network ports used by the MM in the CIGMA. The communication of the wrapper with external devices is limited to the set of ports specified in the property file.

---

\(^1\)Popular web applications include Java servlets, common gateway interface (CGI), active server page (ASP), etc.
2.4 Service Deployment

To participate in the CIGMA, merchants need to install and configure the MW. We use Service Contract (SC) to simplify the configuration process. The SC represents a collection of well-defined and externally visible rules which both humans and machines can understand [15]. It is used as an enforcement mechanism for proper interactions between the CIGMA and the merchant. An example of the SC is shown in Fig. 2.

After the SC is filled up and the MW is installed on the merchant’s server, both the merchant and the CIGMA needs to configure their systems according to the SC. Since the SC contains the specifications for all interaction rules in a machine readable form (XML), the configuration is simply done by feeding the SC into both systems. While the CIGMA is running, there could be some changes in the service configuration; a new merchant may join, an existing merchant might cancel the contract or provide new content, etc. This re-configuration is also easily done by feeding a new SC into both the CIGMA and the contracted merchant.

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 are caching catalog data in the form of HTML/XML pages or fragments [18], [20], [21], [24] or a query result [32], [34], [36]. However, 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 service 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 DBMS functionality. However, conventional disk based databases are too heavy and complex to provide the high throughput required for an Internet cache. Thus, we designed the CC to be highly light-weight. We use main memory as primary storage. The use of main memory significantly improves performance because, in general, a disk I/O is a fairly heavy operation. Thus, the CC can be considered as a kind of main memory DB. In addition, the following characteristics have been added to further improve the performance.

First, to be further light-weight, the CC provides only several core functions such as storing, indexing, and query processing, which are indispensable for catalog caching. It does not support many complex functions common to full-fledged DBMS. This is possible because the CC is designed as a cache rather than as a permanent storage. For example, it does not have persistency-related functions such as support for the durability property, failure recovery, and logging and rollback operations. 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 probable in 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 preference to them over the others in the request queue. In addition, once started, an update transaction is exclusively processed 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 the MM, the CCM composes a proper query string based on the incoming request. Then it updates cached data by sending the query to the CC. After the update, the result 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 and collects the frequency of the requests from customers.
and merchants. Based on the collected data, it controls major facilities of the components to improve the performance.

3.1 Instant Update Mechanism

It is important to keep the data in the CC up-to-date in the CIGMA service. Thus, any modifications of catalog data in merchant databases should be promptly reflected 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 a very short delay.

The instant identification of updates is done based on trigger mechanisms in merchant databases. Using a trigger mechanism, the update process can be done very efficiently. Since changes can be detected in the unit of a field via this mechanism, fine-grained invalidation is possible. Trigger mechanisms are provided in most popular database systems such as Oracle [7], DB2 [8], Sybase [9], MySql [10], PostgreSQL [11], etc.

Currently, time-to-live (TTL) based schemes are most popularly used as cache consistency mechanisms in the Internet [28]. 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 at very short time intervals. However, this will incur excessive overhead to the cache. On the contrary, server-push style approaches can more quickly reflect changes in the original data. However, it has been reported that the update schemes of server-push style experience heavy server-side loads [19]. This load results from the fact that a server must keep track of all the caches which hold copies of its data per data item basis. That is, the server has to handle a large number of caches. However, the CIGMA can avoid this burden. As the CIGMA service requires explicit permission from a merchant, the merchant server can control the number of contracted caches.

Figure 3 shows the detailed structure of the MW and the whole process from modifying a DB to sending an invalidation message. When an update event occurs at the merchant DB (1), the Update Trigger 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 in the MW receives the information. The Catalog Converter converts the category as well as the format of the modification information (4) referring to the Catalog Mapping Table, if necessary (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 to the CCM (8). Lastly, The CCM takes charge of both constructing a proper query message based on the received message and also committing the update transaction on the CC (9).

When sending an invalidation message, the MW piggybacks the message with the modified field and value. Thus, an update can be completed with one message containing both invalidation and modification information. Most modifications will occur in small size fields 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 level of visibility of 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 propagated to the CIGMA immediately, or every hour or every day. Merchants can also set update conditions for their catalogs, so that a modification in a catalog is propagated to the CIGMA only when the update condition is satisfied. Note that the catalog conversion capability described in Sect. 3.2 may be used to control the visibility level as well.

3.2 Catalog Conversion

The CC integrates 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 item category. Therefore, merchants’ catalogs have to be re-categorized and formatted to match those of the CIGMA. Such conversions become significant burdens to the CIGMA since they occur upon every update request from merchants. To alleviate the burden to the CIGMA, we design that the conversion is performed at each merchant server not on the CIGMA system. Thus, a new merchant can join the CIGMA without incurring additional overhead to the CIGMA system. In addition, as we cache the catalog data in the source level, the conversion is performed to each updated source data without any unnecessary conversion.

For the conversion, the CIGMA uses the Catalog Mapping Table (CMT). The CMT is a table which contains all of the descriptions for catalog conversion and re-categorization. That is, the exact specification of the conversion is done based on the CMT before starting a service. Once a service is started, the conversion is automatically processed. The CMT is used when a merchant server converts its catalogs to those of the CIGMA. It is also used when the CIGMA needs to retrieve a field from the merchant DB.

The CMT includes table names and primary key of a merchant DB as well as those of the CC. It then specifies how field names and values in a merchant catalog database are changed to those of CC using an AWK-like script language. Figure 4 shows a row of a CMT with an incoming update message from a merchant DB and an out-going update message to the cache. The item “wirelessphone” is re-categorized 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 is transformed into a semantically different type in the out-going message. This semantic transformation is especially important since it enables merchants to control the external appearance of their catalogs.

4. Implementation

We have implemented a prototype of the CIGMA system and two sample shopping sites. Figure 5 shows a snapshot of the CIGMA Web site. Details about an example service including sample shopping sites are described in [27]. The CIGMA and merchant systems have been developed and are operating on Linux. The GNU C++ compiler has been used to implement most of the components of the CIGMA, and Java servlets and Java Server Pages (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.

The current implementation of the CM uses Apache Tomcat as the Web and application servers and its Coyote HTTP connector to support the HTTP protocol. It also includes several handlers for each request type, such as category list, catalog browse, catalog comparison, item purchase.

In many cases, an HTML page includes embedded data

---

1AWK 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.

11You can view the CIGMA and the two sample merchant sites. The URL of the CIGMA is http://rosebud.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/.
such as images, sounds, and videos. Web pages with such embedded data require careful handling because they have a significant impact on performance. Such media data generally require a lot of memory space in the cache. Also, under 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 container HTML file.

For 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 main memory and transferred to a customer upon 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 have embedded data retrieved from the original merchant server. 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 throughputs different from each other. Thus, a decision on whether media data is cached or not needs to be made in consideration of the resource situation of both the CIGMA and the relevant 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 only been partially implemented. 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” [12]. This helped 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 DBMS’s. For the time being, those for the Oracle DBMS are provided.

5. Performance Evaluation

We evaluated the performance of the CIGMA in four different categories. First, to show the performance as seen to merchants, we measured the throughput when there are only update requests. We repeated the experiment varying the number of merchants which are simultaneously sending update requests, to identify the effect of the number of contracted merchants on the performance. We also measured update delays under several network environments. Second, we show the performance of the CIGMA as seen to customers by measuring the throughput when there are only catalog retrieval requests. We measured the throughput varying the size of catalog cache. The measurements were performed on catalog browse requests and catalog comparison requests separately. Third, we measured the throughput when both catalog retrieval and update requests are issued together to know the performance in a real situation where both customers and merchants are in operation at the same time. Lastly, we measured the throughput of a merchant database to show the performance effect of each merchant side component i.e., update trigger, Event Reporter, and the MW, on a merchant server.

All the reported measurements were performed on a single-node deployment. That is, all the major components of the CIGMA resided 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 than that reported here. The node has a Pentium III 1 Ghz CPU and 2 GB of main memory. Red Hat Linux 7.3 was used as the operating system. Tomcat 4.1 was used as the Web and application server. Sun Java SDK 3.1 was used as the Java virtual machine. As the CIGMA’s catalog cache, FastDB version 2.37 was used. To test the performance under customer’s catalog retrieval requests, we used Httperf [23]. However, to measure update performance, we built a simple utility called an update request generator.

5.1 Catalog Update

The throughput under catalog update requests is measured
as a function of concurrently updating merchants which represents merchants simultaneously sending update messages. Figure 6 shows the throughput when the number of merchants sending update requests concurrently is 1, 5, 10, 50, and 100. It shows that the CIGMA can process nearly 550 update requests per second when the size of update messages is less than 1 Kbyte. The performance degrades to 500 as the message size increases to 10 Kbyte. However, we believe that the throughput of 550 is more realistic since updates usually occur in small-sized fields, e.g., changes in stock status or prices.

Figure 6 also shows that the number of merchants sending update messages concurrently has little effect on update performance as long as the number is not larger than 100. The throughput varies little as the number of merchants sending update requests concurrently increases up to 100. The result that CIGMA can process more than 500 req/s with 100 updating merchants means that CIGMA can handle the situation where 10 k merchants send update requests three times a minute. The reason why the throughput with 1 merchants shows low throughput is that 1 merchant can not fully utilize the CIGMA system. This is because a merchant sends a new request only after receiving a response for the previous one. The figure also shows that the throughput decreases very slowly with the increase of update message size. (Note that the horizontal axis is in log scale.) We conjecture that the throughput is affected more seriously by the overhead of TCP connection, thus the throughput will be significantly increased if persistent connections are used for updates. (This was partially confirmed by other experiments which we do not report here.)

We next measured the update delay under various situations. First, we measured the delay in a local area network (LAN) environment. In this case, the CIGMA server and the systems for merchants or customers were connected via a 100 Mb local Ethernet. Second, using Dummynet [31], we emulated three different wide area network (WAN) environments: WAN I - 20 ms round trip time (RTT), WAN II - 50 ms RTT and 1% packet loss rate (PLR) †, and WAN III - 200 ms RTT and 2% PLR.

Figure 7 shows the average delay as a function of update message size. The average delay is no longer than 800 ms even under the WAN III environment. We believe this tells that the instant update mechanism is valid in WAN environments. Note that the average delay of 5 Kbyte message is shorter than that of smaller-size message when there is packet loss (WAN II or III). Our analysis, which is not reported here, shows that this is due to TCP’s retransmission mechanism ††.

Figure 8 (a) and (b) show the distribution of update delays under the WAN II and III environments, respectively. The maximum update delay is 13 seconds under WAN II, and 28 seconds under WAN III. The figures show that more than 99.9% of updates are finished within 10 seconds in both cases. Note that even the rare cases with long update delays such as the case with the delay of 28 seconds are identified within a predefined time-out interval by heartbeat messages.

5.2 Catalog Retrieval

Figure 9 (a) shows the throughput of the CIGMA under catalog browse requests as a function of response HTML page size when there are 100k, 500k, and 1 M items in the catalog cache. With a 100k item cache, the CIGMA can process 245 requests per second when the response HTML page is 1 Kbyte. This rate decreases to 205 and 170 when the size increases to 4 and 8 Kbyte, respectively. Even when there are 1 M items in the cache, the throughput is 210 for 1 Kbyte response.

The throughput under catalog comparison requests is also measured as a function of response table size. Figure 9 (b) shows that the throughput ranges from 180 to 100 for 100k item cache when the size of the comparison table varies from 10 to 40. The throughput varies from 127 to 85 for a 1 M item cache.

Considering that the response HTML file size of cata-

†The packet loss rate is applied simultaneously to both directions.

††In TCP, a sender retransmits a packet promptly after three duplicated ACKs. Interested readers are referred to [35].
log comparison requests is smaller than that of the catalog browse requests (When the comparison table size is 10, the size of HTML response containing the comparison table is about 2 Kbyte), the throughput under catalog comparison requests is lower than that under catalog browse requests. We conjecture that the performance difference arises mainly from the difference in the overhead to access the catalog cache. Generally, to construct a comparison table, a separate cache access is required for each row. From our informal observation, the overheads incurred by the Web server and the application server were similar in both cases.

5.3 Catalog Retrieval and Update

In a real situation, the CIGMA processes both catalog retrieval and update requests simultaneously. To see the performance in such a situation, we measured the update performance varying the number of catalog retrieval requests which the CIGMA process at the same time. Since both workloads share the CCS, there might be resource contention and thus performance degradation. We assume that the size of response HTML page for a catalog browse request is 1 Kbyte and the size of a catalog comparison table is 10.

Figure 10 shows the result when the number of items in the cache is 10k and five merchants continuously send 100 byte update messages. The two dashed lines show the
expected throughputs and calculated using the performance result of individual workloads described in Sect. 5.1 and 5.2. These two lines represent the throughput under the ideal situation where both workloads share the CIGMA without any contention. The measured throughputs are shown as solid lines. The measured throughputs are very close to the expected ones, which shows that the CIGMA is shared with little overhead.

Table 1 The performance effect on merchant servers.

<table>
<thead>
<tr>
<th>Experimental Condition</th>
<th>Throughput (updates/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>no active component</td>
<td>206 (100%)</td>
</tr>
<tr>
<td>w/ trigger</td>
<td>192 (93%)</td>
</tr>
<tr>
<td>w/ trigger and event reporter</td>
<td>118 (57%)</td>
</tr>
<tr>
<td>w/ trigger, event reporter, and MW</td>
<td>110 (54%)</td>
</tr>
</tbody>
</table>

5.4 Update Trigger and Content Conversion

To identify the performance effect of instant update mechanism on a merchant system, we measure the update throughput of a merchant database under four different conditions: (1) with no active component, (2) with an update trigger in operation, (3) with an update trigger and an event reporter in operation, and lastly (4) with all of an update trigger, an event reporter, and the MW in operation. For the simplicity of the experiments, we configure the MW just to convert the format of an incoming update message without delivering it to the CIGMA. The merchant system is equipped with a Pentium III 1 GHz CPU and 512 M RAM. Red Hat Linux 7.3 and Oracle 8.1.7 are used as the O/S and the database, respectively.

To measure the update performance of the merchant server, we built a small utility similar to the update request generator. It receives as arguments the number and the size of an update query, generates queries, and sends them to the server. Upon receiving each update query, the trigger in the server calls the event reporter. The event reporter is implemented as a Java stored procedure.

Table 1 shows the update throughput of the merchant’s item database. When all three components are active, the performance degrades to 54% of that measured with no active components. The overhead of activating the trigger is 7%, while that of activating both the event reporter and the trigger is 43%. The cost of the event reporter takes almost 80% of the whole update overhead. We believe that this is due to the high cost of communication between the JVM where the event report sits and the database.

6. Related Work

6.1 EC Service Models

In the view of the EC service model, price comparison services [1]–[5] 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 instant update scheme. Many services update their data manually or use schemes based on periodic or aperiodic polling. Compared to the CIGMA’s case, we may say that those services help online customers in a best-effort style.

AddAll [6] provides a price comparison service based on instant catalog searching and gathering upon customers’ requests. It can thus provide fresh information about an item. However, it has a serious problem; the response time may be too long because it has to visit many shopping sites. In addition, it issues multiple Web requests per each customer request and thus may cause heavy traffic on the Internet.

These price comparison services are not shopping services. That is, none of them have purchase transaction mediation functionality. Thus, customers have to visit the original shopping site to buy a selected item.

The B2B marketplaces [13], [14] mediate between customers and merchants, and provide a set of services to support on-line purchasing. But, contrary to the CIGMA, most B2B marketplace models are designed and available only for business customers and/or a large volume of transactions. In addition, since B2B marketplaces do not deal with ordinary customers directly, performance is not seriously considered in their design.

An electronic catalog mediating system was proposed in [30]. Its goal is to mediate distributed catalog databases. It retrieves catalogs from multiple catalog DBs. However, it does not cache catalogs but visits all catalog DBs repeatedly upon customers’ requests. Thus, this system can be used to provide an instant search-based price comparison service.

6.2 Techniques to Cache Dynamic Data on the Internet

Recently, a number of researchers have proposed techniques for dynamic data caching [16], [18], [20], [21], [29], [34], [36]. While these techniques are mainly to improve the performance of individual Web sites, the CCS is to support the provision of cross-organizational data services. Below, we classify related techniques in terms of caching units.

**HTML page caching** stores HTML pages generated upon clients’ requests [18], [24]. In terms 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 updates in numerous pages.

**XML/HTML fragment caching** stores XML or HTML fragments which are parts of generated HTML pages. The system proposed in [21] 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 the fact that it removes the query processing step. Form-based Proxy Caching [33] and Database Accelerator [22] 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, 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.

7. 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 a really good one even after exhaustive Web surfing. The CIGMA addresses the problems in online shopping under such a dynamic environment. It suggests the best sales condition 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 is 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 an example of an Internet cache for advanced Internet services. For an advanced service such as the CIGMA, the ordinary caching architectures such as proxy or reverse proxy are not appropriate. To meet the high performance requirement of the CIGMA and other advanced services, it is designed as a light-weight main memory cache. To further improve performance and flexibility, it caches catalog data at the source level. It also provides a instant update mechanism.

We plan to extend the CIGMA in several ways. First, we plan to provide a failover or redundancy scheme for the TM. We also plan to develop an alternative to the database trigger mechanism to further lessen the performance overhead of merchants. One possible approach is to develop a method based on log monitoring. By continuously inspecting and analyzing DBMS’ log file, most database events can be caught. Finally, we would like to apply the proposed schemes to other types of e-commerce services, e.g., auction or reverse auction.

References

Su Myeon Kim is a graduate student at the Korea Institute of Science and Technology (KAIST), Daejeon, Korea. He received the B.S. degree in Mathematics from the Yonsei University, Seoul, Korea, in 1995, and M.S. degree in Computer Science from KAIST in 1997. His research interest lies in E-commerce, Distributed Computing, Internet Computing, and System and Application Design and Building.

Heung-Kyu Lee received the B.S. degree in electronics engineering from the Seoul National University, Seoul, Korea, in 1978, and M.S., Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), in 1981, and 1984, respectively. From 1984 to 1985 he served as a post-doc at the University of Michigan, Ann Arbor. Since 1986 he has been a professor in the Department of Computer Science, the Korea Advanced Institute of Science and Technology, Daejeon, Korea. His major interests are digital watermarking, real-time processing, and image processing.

Seungwoo Kang is a graduate student at the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. He received the B.S. degree in Electrical Engineering from the Yonsei University, Seoul, Korea, in 2000, and M.S. degree in Electrical Engineering from KAIST in 2002. His research interest lies in Internet Technologies, Internet-scale distributed computing systems.

Junehwa Song is an associate professor, Department of EECS, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. Before joining KAIST, he worked at IBM T.J. Watson Research Center, Yorktown Heights, NY as a Research Staff Member from 1997 to September 2000. He received his Ph.D. in Computer Science from University of Maryland at College Park in 1997. His research interest lies in Internet Technologies, such as intermediary devices, high performance Web serving, electronic commerce, etc., and distributed multimedia systems.