Content Aggregation Middleware (CAM) for Fast Development of High Performance Web Syndication Services

Su Myeon Kim, Jinwon Lee, SungJae Jo and Junehwa Song

Dept. of EECS., Korea Advanced Institute of Science and Technology,
371-1 Guseong-Dong, Yuseong-Gu, Daejon, 305-701, Korea
{smkim, jcircle, stanleylab, junesong}@nclab.kaist.ac.kr

Abstract. The rapid expansion of the Internet accompanies a serious side effect. Since there are too many information providers, it is very difficult to obtain the contents best fitting to customers' needs. Web Syndication Services (WSS) are emerging as solutions to the information flooding problem. However, even with its practical importance, WSS has not been much studied yet. In this paper, we propose the Content Aggregation Middleware (CAM). It provides a WSS with a content gathering substratum effective in gathering and processing data from many different source sites. Using the CAM, WSS provider can build up a new service without involving the details of complicated content aggregation procedures, and thus concentrate on developing the service logic. We describe the design, implementation, and performance of the CAM.

1 Introduction

The Internet has been growing exponentially for the past decades, and has already become the major source of information. The estimated number of Internet hosts reached 72 million in February 2000, and is expected to reach 1 billion by 2008[1]. However, such a rapid expansion accompanies a serious side effect. Although users can easily access the Internet, it is very difficult to obtain the contents best fitting to their needs since there are too many information providers (Web hosts). This problem is usually called information flooding.

Various Web Syndication Services (WSS) (See Figure 1) are emerging as solutions to the information flooding problem. WSS is a new kind of Internet service which spans over distributed Web sites. It provides value-added information by processing (e.g., integrating, comparing, filtering, etc) contents gathered from other Web sites. Price comparison service such as Shopping.com [2] and travel consolidator service like Expedia.com [3] can be considered as examples of the WSSs. To ordinary clients who are not familiar with a specific domain, a WSS targeting the domain would be of a great help to overcome the information flooding.

Providing a WSS is technically challenging. It is much more complicated than providing an ordinary service. However, even with its practical importance, WSS has not been much studied yet. A system providing a WSS can be seen into two parts; the

WSS service logic and the content aggregation subsystem. The content aggregation is the common core of many WSS's while the service logic is service specific and differs from service to service. It receives requests from clients and interacts with source sites to process the requests. In this paper, we propose the Content Aggregation Middleware (CAM). The CAM is an efficient content aggregation system designed to be a base for many WSS's. Using the CAM, a service provider can easily develop and deploy a high performance WSS system supporting a large number of clients and source sites.

We identify several requirements for a WSS site. First, a WSS site should support a high level of performance. The performance requirement in a WSS site is a lot higher than in ordinary Web sites. It should manage much larger number of requests from clients spread over the Internet. Additionally, it should handle a huge number of source sites and interactions with them. Second, it should support high dynamics of Internet environment. In a fully Internet-connected environment, real world events can be quickly reflected and propagated to systems. Once generated, the information will go through frequent changes. Third, a WSS site should deal with many source sites, which are highly heterogeneous.

The CAM has been designed to meet the above requirements of a WSS site. It provides a WSS with a content gathering substratum effective in gathering and processing data from many different source sites. Using the CAM, WSS provider can build up a new service without involving the details of complicated content aggregation procedures, and thus concentrate on developing the service logic. The CAM simplifies the complex procedure of interacting with content providers through a formalized service contract (SC). Also, it effectively masks the high level of heterogeneity among different source sites. In addition, it is a high performance system much relaxing the burden of performance concerns in system development. Below, we describe the novel characteristics of the proposed content aggregation system.

First, the CAM is a source data caching system along with basic data processing capabilities. It caches data in the form of source data, e.g., the unit of database fields as stored in content providers' databases. For value-added service, fine-grained control on the cached contents gathered is required. Source data caching makes such fine-grained control possible. For data processing, basic functions such as content conversion, filtering, and query processing, are provided.

Second, it is a high performance system. As mentioned, a WSS should handle a high rate of requests from lots of clients. In addition, it should be capable of managing a lot of interactions with source sites to keep the freshness of cached data. With a source data caching, keeping cached data up-to-date can be done efficiently. Also, to manage a large volume of data efficiently, it uses main memory as a primary storage.

Third, the CAM is equipped with real-time update capability. To keep the freshness of cached contents, any modification on the data at source sites is propagated to the CAM as soon as possible. The update mechanism is based on server invalidation scheme; upon modification, the source site initiates invalidation and modification of the cached data in the CAM. In this way, the delay to the data update can be shortened.

Fourth, a wrapper is used to deal with the heterogeneity of the source sites. The CAM gathers contents from many different source sites. So, handling the different

sites in a uniform way is critical to the CAM system. By deploying a wrapper module to each source server, the CAM can handle different source sites in a uniform way.

In this paper, we present the design and implementation of the CAM. We also show some measurement results to demonstrate the performance of the system. The current version of our system is designed for WSS interacting with typical Web sites. Thus, Web sites adopting new technologies such as XML and Web Services, are not considered in this paper. We believe that such emerging Web technologies can be easily incorporated to our system.

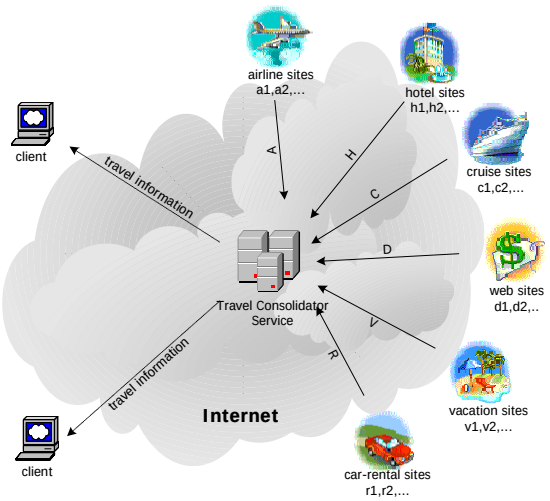


Fig. 1. An Example of Web Syndication Services – Travel Consolidator Service

This paper is organized as follows. The CAM architecture is described in Section 2. A few challenging issues are discussed in Section 3. In Section 4, system performance is discussed. We discuss related work in Section 5. Finally, we conclude our work in Section 6.

2 Content Aggregation Middleware (CAM) Architecture

A WSS can be constructed with a front-end WSS logic and a back-end CAM (See Fig. 2). The WSS logic implements service specific application logic. It is usually implemented as Web applications using JSP, Servlet, etc. It interacts with clients via Web server or application server to receive requests and deliver results. It also interacts with the CAM to request or to receive data required to construct result pages.

The CAM has a modular structure, which consists of four components: Content Provider Wrapper (CPW), Content Provider Manager (CPM), Memory Cache Manager (MCM), and Memory Cache (MC). CPW runs on content provider sites and enables the CAM to access different content providers in an identical way. The other components are on the WSS site. CPM communicates with content providers and receives contents. MCM manages MC, which stores and manages the retrieved data.

2.1 Processing flow

The CAM mainly deals with two kinds of requests: content update request and content access request. The request processing flows are shown in Fig. 2.

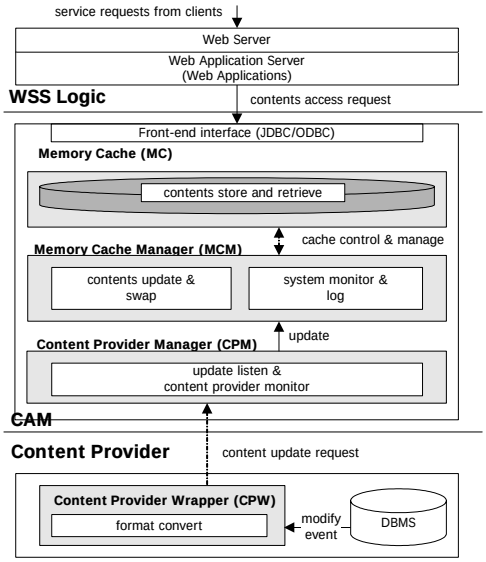


Fig. 2. CAM Architecture

The update process is initiated when data is modified in the content provider's database. The content provider detects and notifies the update event, including table ID, field names, and modified values, to CPW. CPW receives the notification message. Then, it converts the data according to the converting information and sends the converted result to CPM. CPM receives the update message and forwards it to MCM. Finally, MCM replaces the data in the MC with the new data in the update message.

The content access process is initiated when clients request a service. Web applications implementing WSS logic retrieve data from MC and generate a service result. The Web applications access MC via a popular database interface such as JDBC or ODBC

2.2 Deployment of WSS and the CAM

In order to start a WSS with the CAM, content providers as well as the WSS provider need to participate in service deployment process. First, both the WSS provider and participating content providers should agree on how to interact with each other. Second, content providers need to install and configure a CPW. We use Service Contract (SC) to simplify the configuration process. The SC represents a collection of well-defined and externally visible rules which both human and machine can understand [4]. It is used as an enforcement mechanism for proper interactions between the CAM and content providers. The structure of the SC is shown in Fig. 3.

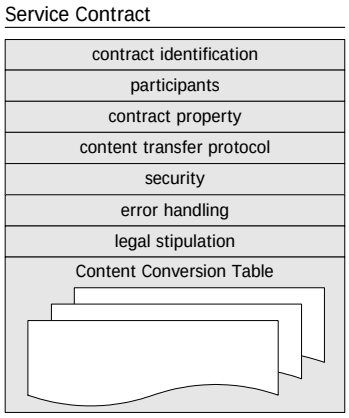


Fig. 3. Structure of the Service Contract

After the SC is filled up and a CPW is installed in a content provider server, the content provider and the CAM configure their systems according to the SC. Since the SC contains the specification for all the interaction rules, the configuration is simply done by feeding the SC into the systems. At the content provider site, the CPW first parses the SC and then sets up related components such as the communication interfaces.The CAM also parses the SC. Then, it notifies CPM's monitoring module and update-listening module of the new content provider. If needed, it also forwards configuration information such as valid actions, protocols, and addresses, to each module. Based on this information, the modules prepare themselves for the new content provider. Note that, in the proposed architecture, the re-configuration is easily done dynamically by feeding a new SC into the CAM and the contracted content provider.

3 Design Challenges

3.1 Instant update mechanism

It is important to keep the data in the CAM up-to-date. Thus, any modifications of contents in the content provider's database should be promptly reflected to those in the CAM. In addition, the update mechanism should be efficient since a high number of update requests are expected.

The update scheme is based on server-push. The content provider server instantly identifies any modification in the database, and initiates an update in the CAM by sending out an invalidation message. Thus, an update is propagated to the CAM with very small delay. When sending an invalidation message, we piggyback the message with the modified field and value. Thus, an update can be completed with one message.

The instant identification of content modification is done based on a trigger mechanism in the content provider's database. Using a trigger mechanism, the update process can be done very efficiently. It is so since fine grained invalidation is possible due to the use of the mechanism where changes can be detected in the unit of a field. Trigger mechanisms are provided in many popular DBMS's such as Oracle, DB2, and MySql .

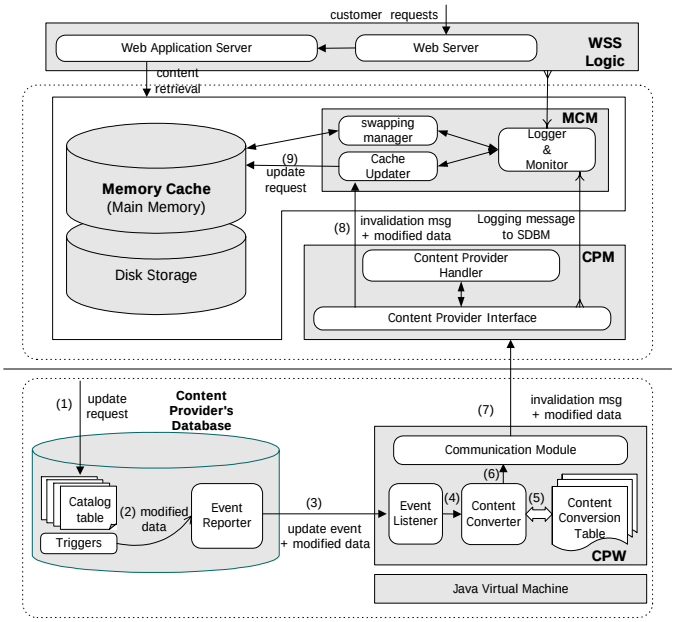


Fig. 4. Instant Update Mechanism and Its Procedure

Currently, time-to-live (TTL) based schemes are most popularly used as a cache consistency mechanism in the Internet [8]. However, TTL-based schemes are not proper for the CAM 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.

Fig. 4 shows the detailed structure of CPW and the whole update process from database modification at content provider server to actual update at the CAM. When an update occurs at the content provider's database (1), the trigger routine activates a trigger, here we named it as Event Reporter. (2). The Event Reporter sends the modified information to CPW (3). The update information is received by the Event Listener module in CPW. Then, Content Converter converts the schema and format of the received information (4) by referring to Content Conversion Table, if needed (5). The Content Converter makes an update message with the converted information (6). Then, the Communication Module sends the message to the CAM (7). As soon as CPM receives the update message, it forwards the message to MCM (8). Lastly,

MCM constructs a proper query message based on the received message and commits the update transaction on MC, logging this event if required (9).

3.2 Template-based, safe wrapper mechanism

In CPW design, safety should be importantly considered since it runs in foreign (i.e., content providers) servers. It may access confidential data, crash, or generate an error disturbing the server system.

For safety, we propose a dynamically customizable wrapper. In our approach, a generic wrapper template is composed and used for every content provider. Since there is only one template, certifying the safety of the wrapper becomes easy. For instance, the safety can be certified by the third part agency. Once certified, the safety of the wrapper is assured for every content provider. Each wrapper instance is generated from the template along with an SC. The instance will act as specified in the SC. Note that the SC is signed by the WSS provider as well as the content provider.

We implement the wrapper using Java. Java-based implementation is advantageous in several ways. First, the module can be installed in any computing environment running Java Virtual Machine (JVM). Second, faults in a wrapper module do not affect the reliability of a content provider system. Faults in the wrapper module propagate only to the virtual machine. Third, by using the powerful access control mechanism of JAVA, content providers can prevent a wrapper from accessing their resources.

4 Performance Evaluation

The performance of the CAM prototype is evaluated using a prototype. For high performance, the CAM prototype was implemented mostly in C++ on Linux platform. Most of the MCM's functions, including system monitoring and logging, have been implemented. In the current prototype, the number of requests for each content object and that of messages from each content provider are monitored and logged. The current version of MC has been implemented by customizing the third party main memory database, ”Altibase” [9]. This helps us quickly implement the prototype. To help content providers set up database 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

4.1 Experimental environment

We assume that the CAM is deployed on a single node. The performance will increase when the CAM is deployed on multiple nodes. For the simplicity of measurement, clients and content providers are connected to the CAM via 100M local area networks. Each node has a Pentium III 1Ghz CPU and 512MB main memory except the node for the CAM which has 2GB main memory. Red Hat Linux 7.2 is used as

the operating system, and Sun JAVA 1.3 is used as the JVM. Apache 1.3.20 and Tomcat 3.2.3 is used for the Web server and the application server, respectively. In the rest of this section, we assume that all the cached contents fit in the main memory.

4.2 Workload and measures

The performance of the CAM prototype is evaluated via three different measures: (1) browse, (2) update, (3) mixed throughputs. The browse throughput is measured when requests are only from clients, while the update throughput is measured when requests are only from content providers. The mixed throughput is measured when the two types of requests are issued.

To measure the performance of browse request processing, we use a transactional Web benchmark: TPC Benchmark™ W (TPC-W) [13]. It is commonly used to measure the performance of a database-backed web serving system. We slightly modified the TPC-W benchmark. Originally, there are two kinds of interactions in the TPC-W specification: browsing and ordering. We use only browse interactions in our experiment since the browsing interaction is composed of database retrieval operations. Note that *database scale factor* is used to specify the scale of the measured web serving system. To measure the update throughput, we made our own utility called update request generator. It generates and sends multiple update requests simultaneously, emulating the situation where several content providers update their contents at the same time.

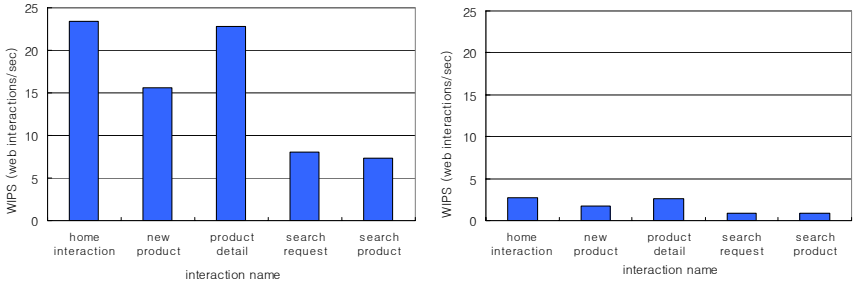
4.3 Performance evaluation

We measure the throughput and response time when database scale is 10k or 100k. Fig. 5 shows the throughput of five browsing interactions. Throughput is represented in WIPS - the number of interactions processed per second. The total WIPS, i.e., the summation of the WIPS for five interactions, is 77 and 8 when database scale is 10k or 100k, respectively. The response time measured from the same experiments shows that all requests are processed in 0.23 and 1 second when database scale is 10k or 100k, respectively.

Fig. 6 (a) shows the throughputs as the number of threads in the update request generator increases from one to ten; the number of threads represents the number of content providers sending updates simultaneously. We run the experiments when the update message size is 64 and 256 bytes. Although the update size would be arbitrary, we assume that the sizes of frequently changed database fields are not large. The number 64 is chosen since it is the smallest power of two larger than 38, which is the maximum digit of numeric variable in Oracle database. Similarly, 256 is the closest number to 255 which is the default size of char type in Oracle. The figure shows that the CAM processes about 400 requests per second. The number of active content providers or update message size has a negligible effect on the performance.

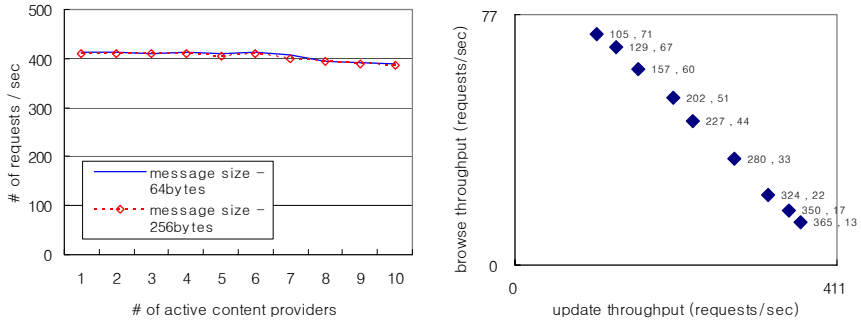
To measure the mixed throughput, we kept sending a fixed number of update requests per second via the update request generator, and then measured the browse throughput via TCP-W. Fig. 6 (b) shows the results. For simplicity, the throughput is

represented as the total WIPS. Note that from the previous experiments, the browse only throughput, i.e., browse throughput without any update, is 77 and the update only throughput is 411.



(a) 10k database scale (b) 100k database scale

Fig. 5. Browsing Throughput



(a) (b)

Fig. 6. (a) Update Throughput, (b) Mixed Throughput

5 Related Work

Content aggregation tools such as Agentware [10], Active Data Exchange [11], and Enterprise Content Management Suite [12] help to retrieve and aggregate contents from multiple information sources. However, those tools are for an intra-organizational use, while the CAM is designed for inter-organizational use.

Recently, a number of researches have proposed techniques for dynamic data caching [5, 6, 7]. 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 CAM is different in that it focuses on the provision of a WSS, which is a new type of cross-organizational data services, based on the cached information. The CAM 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 CAM is closer to a proxy that operates along with a number of content providers.

6 Conclusion

A WSS system is composed of a WSS service logic and the content aggregation subsystem. The content aggregation is the common core of many WSS's, while the service logic is service specific and differs from a service to another. We proposed a high performance content aggregation middleware called the CAM. The CAM provides a WSS with a content gathering substratum effective in gathering and processing data from many different source sites. Using the CAM, WSS provider can build up a new service without involving the details of complicated content aggregation procedures, and thus concentrate on developing the service logic.

The CAM is a source data caching system and makes possible fine-grained control of gathered contents. It is a high performance system capable of handling a high rate of request from lots of clients and content providers. Also, it uses main memory as a primary storage to efficiently manage a large volume of data. The CAM is equipped with real-time update capability to keep the freshness of cached contents. It is equipped with a wrapper to deal with the heterogeneity of the source sites.

In this paper, we described the design and implementation of the CAM. We also showed the performance of the CAM prototype. We currently plan to further improve the performance of the system.

References

1. T. Rutkowski,http://www.ngi.org/trends/TrendsPR0002.txt, Bianual strategic note, Center for Next Generation Internet, February, 2000
2. http://www.shopping.com/, price comparison service site
3. http://www.expedia.com/, travel consolidator service
4. Asit Dan and Francis Parr, The Coyote approach for Network Centric Service Applications: Conversational Service Transactions, a Monitor and an Application, High Performance Transaction Processing (HPTS) Workshop, Sep, 1997
5. Khaled Yagoub and Daniela Florescu and Cezar Cristian Andrei and Valerie Issarny, Building and Customizing Data-intensive Web Site using Weave, Proc. of the Int. Conf. on Very Large Data Bases (VLDB), Cairo, Egypt, Sep, 2000
6. K. Selcuk Candan and Wen-Syan Li and Qiong Luo and Wang-Pin Hsiung and Divyakant Agrawal, Enabling Dynamic Content Caching for Database-Driven Web Sites, Proceedings of SIGMOD'2001,pages 532 – 543, California, USA, May, 2001
7. Qiong Luo and Jeffrey F. Naughton, Form-based proxy caching for database-backed web sites, Proc. of the Int. Conf. on Very Large Data Bases (VLDB), Roma, Italy , Sep, 2001
8. Balachaner Krishnamurthy and Jennifer Rexford,Web Protocols and Practice - HTTP/1.1, Networking Protocols, Caching and Traffic Measurement, Addison-Wesley, 2001
9. http://www.altibase.com, Altibase Main Memory Database
10. http://www.agentware.net/, AgentWare (integrating the internet) - AgentWare Syndicator
11. http://www.activedatax.com/, Active Data Exchange - active data Syndicator
12. http://www.reddot.com/, RedDot Solutions - Content Import Engine
13. http://www.tpc.org/tpcw/default.asp, TPC benchmark W specification version 1.4, Feb., 2001