Design and Implementation of Database Schema Evolution for Service Continuity of Web-based Internet Applications

Yi-Chang Chen*, Jeu-Yih Jeng*, Teh-Sheng Huang*, and Phone Lin+
*Telecommunication Laboratories Chunghwa Telecom Co., Ltd., Taiwan, R.O.C.
+Department of Computer Science and Information Engineering, National Taiwan University, Taiwan, R.O.C.
Email: {lupinchen, jyjeng, tshuang}@cht.com.tw, plin@csie.ntu.edu.tw

Abstract—**Today, Internet services are utilized as major service platforms to provide business services. Web-based Internet applications that follow the multi-tier architecture are a well-accepted computation model when providing Internet services. Such a model suffers from interruption caused by software failures, hardware errors or software maintenance. The interruption may incur significant financial loss to Internet service providers. Thus, Service Continuity is one of the main challenges for Web-based Internet applications. Although several service continuity technologies have been developed, many of them focus on a single tier of the applications and do not guarantee end-to-end service continuity. In this paper, we describe the design, implementation, and performance evaluation of our proposed service continuity technology. This technology resolves the long time interruption problem caused by database schema evolution, handles software components upgrade, and achieves end-to-end Service Continuity. We propose a mechanism that contains multiple modules running interactively between each module. This approach only requires minimal amount of code rewriting and a small storage space. Comparing with existing approaches, our approach is more efficient with the least overhead.**

Keywords- Web-based Internet Applications; Service Continuity; Database Schema Evolution

I. Introduction

Web-based Internet applications suffer from interruption caused by software failure, hardware error or software maintenance. As indicated in the Oracle white paper [1], the annual average of Internet service interruption is 4.38 hours, and the average loss per hour is 1.4 million dollars. This amounts to 6.1 million-dollar loss per year for Internet service providers. Thus, Service Continuity is one of the main challenges for Web-based Internet applications.

Service Continuity [2][3] is proposed to prevent an Internet service from being interrupted or terminated due to software maintenance. Software maintenance can include software components upgrade or database schema evolution. Previous studies propose mechanisms to prevent a session from being interrupted by software components upgrade. For example, the trap and recover mechanism [4] traps the session information before software components upgrade, and then recovers the sessions after software components upgrade. However, the trap and recover mechanism does not support database schema evolution. In this paper, we focus on end-to-end (i.e., client-end to the database tier) Service Continuity for database schema evolution.

Database schema evolution is the process for modifying the database schema and the contents. The database schema describes the format of the tables. Traditionally, in the beginning of database schema evolution, the Database Management System (DBMS) [5][6] stops accepting the read/write operations, which interrupts the Internet service. After the database schema evolution is completed, the web server tier and the application server tier replace the new version of the software components and then the DBMS restarts to accept the read/write operations. Some of the existing relational database systems offer simple transformation functionality of database schema. However, the existing systems do not support complex transformations, such as modifying the data type of the attribute. Thus, the database must go off-line to modify the data type.

A log-based framework, presented in [7][8], is based on log redo. In the beginning of database schema evolution, the framework starts to record the information of current data access (e.g., transaction ID, database name, table name, and so on) to produce the log. After modifying the new database, the database server stops the write operation to redo the log. After all logs are applied to the new database, the new database replaces the original database as the primary database. However, the framework cannot redo the log directly for some complex transformation.

In this paper, we focus on the database schema evolution for Service Continuity of Web-based Internet applications. We propose a solution for database schema evolution that addresses software component upgrade and achieves end-to-end Service Continuity. To process database schema evolution, we propose some modules to record the updated tuples, manage temporary data, and map/convert tuples to tables after database schema evolution. We use the concept of trap and recover mechanism to preserve the session information during the software component upgrade. In addition, we propose a module to control the processing of database schema evolution and software components upgrade. This module also determines the timing to switch software components. After switching software components, the sessions recover on the new software component, and the user will not be asked to

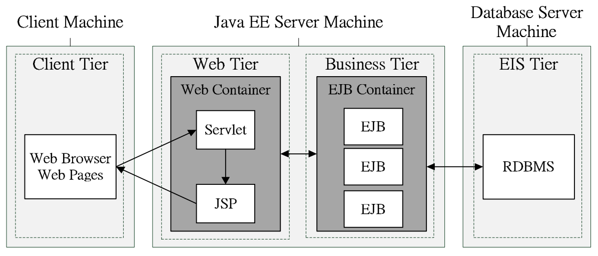

Figure 1: Java EE Distributed Multi-tier Architecture

restart an ongoing service. In other words, throughout the process the users are not aware of any interruptions with the Web-based Internet application.

The rest of the paper is organized as follows. Section 2 illustrates the Web-based Internet application architecture to understand the issues caused by database schema evolution. In section 3, we describe each module of our mechanism. Section 4 presents the system prototype and performance evaluation. We conclude this paper in Section 5.

II. Web-based Internet Application Architecture

In this section, we present a Web-based Internet application architecture and then illustrate an Internet service as a session.

A. System model

The Java Enterprise Edition (EE) platform [9], which uses the distributed multi-tier [10] architecture, is a popular system model for the Internet service provider to build their Web-based Internet applications. It also follows the concept of multi-tier architecture and client-server architecture. Figure 1 shows the Java EE distributed multi-tier architecture. It consists of four tiers: client tier, web tier, business tier and Enterprise Information System (EIS) tier. Internet services process through the components of each tier to be completed.

B. Internet Service

We model the behavior of Internet service as a session. In the beginning, a user at the client-end connects to the Web-based Internet application by web browser. When the user sends the first request, the session starts. The servlet will parse the request to get the request content, and then invocate the corresponding EJB methods to execute business logic. After the EJB executing the business functionalities, it may issue one or more data access requests to write the execution results into the database and returns the results to the servlet. Upon receipt of the return value, the servlet will prepare a JSP to show the result of the request for the user. At this time, the first request is completed. After the user gets the result page, the user spends a period of time reading the results and thinking about the next operation, namely the user thinking time. Note that the request and the user thinking time may repeat several times in a session. When the final request finishes, the session of the Web-based Internet application completes.

C. Database Schema Evolution

In general, the behaviors of database schema evolution for relational database contains adding indices, adding table attributes, deleting table attributes, modifying attribute, merging tables, and splitting tables. Some simple database schema evolution can be achieved by the tools provided by the RDBMS, such as adding indices, adding/deleting table attributes. However, complex schema evolution affects the original data values. Complex schema evolution requires reprocessing the data tables for every value pair or for each data value, so is usually more time-consuming. Take for instance modify attributes, merging tables, and splitting tables. When we change the data type of an attribute, all values of this attribute will be converted from one data type to another. Moreover, the software components that use this attribute will also need to be modified to match the new data type. Complex schema evolution is more difficult to complete in a short period of time and there is no support tool. Hence, the database has to stop its service during database schema evolution.

III. The Database Schema Evolution Mechanism of Web-based Internet Application

This section describes our database schema evolution mechanism. Figure 2 shows all the mechanism modules for database schema evolution.

A. The Data Replication Module (DRM)

The DRM consists of the Trigger Unit (TU) and the Judge Unit (JU). TU is like a listener that listens to the original table. The JU judges whether to add or update this tuple information to the temporary data table. First, if the operation type is adding a tuple, the temporary data table does not have the record of this tuple, so the JU notifies the TDPM to add a record. If the operation type is deleting tuple, and then, adding a record and annotate the operation type. When the operation type is updating tuple, the JU judges whether the record of this updated tuple exist in the temporary table. If the updated record exists, the JU notifies the TDPM to update the record, or otherwise add a new record.

B. The Temporary Data Processing Module (TDPM)

The TDPM manages the data that were added or changed in the original table during database schema evolution. It consists of the Table Access Unit (TA), the Table Monitor Unit (TM), and a temporary data table, named Temp Table. The TA is responsible for handling all table access requests of the Temp Table from other modules. The TM monitors the Temp Table to judge when the MCM starts to update data from the Temp Table to the new table. And it continues to monitor and calculate the temporary data to determine the time point for switching the software components. The purpose of the Temp Table is to log the changed records between the original table and the new table. In order to redo the behavior, we specify the attribute "Operation_Type" to record the Data Manipulation Language (DML) operation type. In addition, we specify another attribute "Status_Check" used to mark each tuple's status of updating to the new table.

In this paper, we assume that the growth rate of the unfinished data is slower than the MCM processing speed. We propose a formula to obtain the time points for software components upgrade. The maximum length $LSQL\_temp$ represents a single SQL statement for the MCM to access the Temp Table; the maximum length $LSQL\_new$ is a single SQL statement for the MCM to access the new table. The maximum execution time of one tuple in the MCM is E, and it is a changing value. The number of unfinished tuples N represents

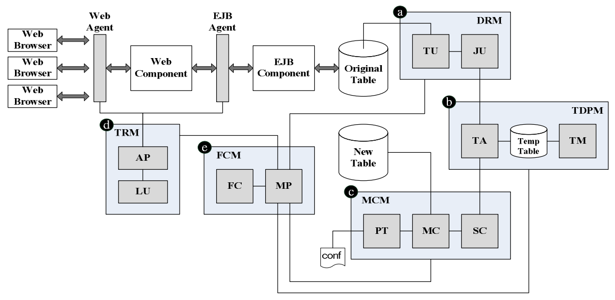

Figure 2: Database Schema Evolution Mechanism

the tuples in Temp Table that the MCM needs to execute. The data transfer speed of the server machine is defined as *P*. The shortest period of time for the Java EE server machine to replace the software components is defined as *T*. Hence, the total execution time of one tuple from the Temp Table to the new table $X = (LSQL\_temp + LSQL\_new) / P + E$. Then, TM continues to achieve the *N* to confirm that the $N * X < T$ condition is satisfied. When this condition is satisfied, all unfinished tuples in the Temp Table will be processed and the new table will be updated completely during switching the software components.

C. The Map and Convert Module (MCM)

The MCM is responsible for processing the temporary data in the Temp Table to match the database schema of the new table, and to update the new table. Since there are different database schemata between the new table and the original table, we implement the Policy Transfer Unit (PT) to analyze the conversion policy. It parsers the configuration file to establish a mapping and conversion rule. When the MCM starts running, the Status Check Unit (SC) will check the Status_Check attribute of the Temp Table to find out which tuples need to be processed. The Map and Convert Unit (MC) determines the operation based on the Operation_Type value and processes data transformation by following the rule established by the PT.

D. The Trap and Recover Module (TRM)

The TRM consists of the Web Agent, the EJB Agent, the Agent Processing Unit (AP), and the Logger Unit (LU). The web agent is set before the web component before; and the EJB agent is set as a Proxy [11] between the web component and the EJB component. At the trap state, the Web/EJB Agent duplicates HTTP requests and EJB invocation to send to the AP. The LU is responsible for collecting the information that the AP analyzed. At recover state, the AP receives the information from the LU to restructure, and then it sends to the Web/ EJB Agent for execution.

E. The Flow Control Module (FCM)

The FCM plays the communication role between other modules, and also manages the work flow of the database schema evolution mechanism. The Message Processing Unit (MP) is the processing element that sends and receives messages. The MP sends the message to the DRM to notify the DRM to start to duplicate the tuples. After database schema evolution is completed, the MP sends the message to the MCM to start processing the temporary data. When the number of the unfinished temporary data meets the condition (i.e., the $N * X < T$ condition) of the TM, the MP communicates with the

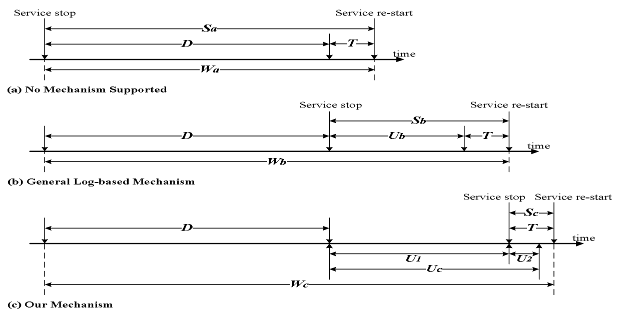

Figure 3: Timelines Diagram for Systems

TRM to switch to the recover state. All of the state transitions are handed over by the Flow Control Unit (FC) to decide when and under what conditions to move on to the next workflow stage. The FC also decides whether to automatically restart the mechanism workflow, if the mechanism error occurs in certain circumstances.

IV. Performance Evaluation

In this section, we present the system prototype and performance evaluation for the prototype.

A. System Prototype

To demonstrate the concept of database schema evolution mechanism and evaluate its performance, we implement the mechanism on a Web-based Internet application and conduct numerous experiments. The Web-based Internet application consists of multiple tiers as shown in Figure 1. In this prototype, we target JAVA EE framework in which the JAVA EE server supports Servlets, JSP, EJB and Java Database Connectivity (JDBC). At the database server side, we used MySQL as our RDBMS, which is a popular database in Web-based Internet applications.

B. Service Interruption Time

Figure 3 illustrates the timeline diagram for systems. The timeline *a* represents that the system undergoing the database schema evolution without any mechanism supporting; *b* represents the system using the general log-based mechanism; *c* represents using our mechanism. The *D* period represents the time of the database schema evolution, and the *T* period represents the time of software component upgrade. The *D* and *T* periods is the same at timeline a, b, and c. The working time *W* contains the time of database schema evolution, software component upgrade, and the data update time, if necessary. The *S* period represents real service interruption, that is, during this period of time the system is unable to perform any requirement. When the system does not have any supporting mechanism, its working time *Wa* and service interruption time *Sa* is the same as the sum of the *D* and *T*. In the general log-based mechanism, its working time *Wb* is the sum of the *D*, *T*, and *Ub*. The service interruption time *Sb* is the sum of the *T* and *Ub*. In general, the period *D* takes a longer time, which makes it also more time-consuming to update the amount of data logged during *D*.

Compared with the general log-based mechanism, our mechanism may experience longer data update time *Uc* because we still accept new requests when the update starts.

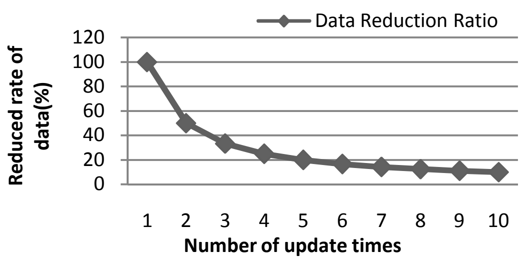

Figure 4: Data Reduction Ratio for Systems

Our data update time consists of two periods. The *U1* period represents the time from when the update starts to when the software component upgrade starts; the *U2* period represents the time from when the software component upgrade starts to the update completed point. Since the *U2* period and the *T* period are parallel processing periods, our actual service interruption time *Sc* is the same as the *T* period, and much shorter than the service interruption time *Sb*. As a result, we can use the TRM to keep the sessions alive during the *T* period, while the users remain unaware that the system had to stop the Internet service.

C. *The Temporary Data and Update Time Evaluation*

The amount of temporary data directly affects the data update time and the working time of the mechanism. Therefore, effectively reducing the amount of temporary data shortens the data update time. During the insert or delete operations, the amount of temporary data in our mechanism is similar to the amount of data in the general log-based mechanism. However, for the update operation, our mechanisms significantly decreases data amount. The data reduction ratio compared with the general log-based mechanism is shown in Figure 4. The X-axis is the number of update times for one tuple; and the Y-axis is the ratio of the data amount in our mechanism relative to the general log-based mechanism. When the repeated tuple updates twice, the amount of temporary data is reduced to 50% and by 1/*x* ratio (i.e., *x* represents the number of update times). In general, during the Internet service, the tuple may be updated several times. Thus, our mechanism delivers good performance based on the amount of temporary data and promises shorter working time (i.e., the *Wc* period) than that of the general log-based mechanism (i.e., the *Wb* period).

To shorten the update time, we must reduce the average processing time for each data. Figure 5 shows the average processing time for different number of temporary data that the MCM processes each time. The result shows that batch processing delivers shorter processing time on average than a single data processing.

V. CONCLUSION

In this paper, we present the design, prototype, and performance evaluation of an end-to-end Service Continuity mechanism for database schema evolution of the Web-based Internet applications. This mechanism can work during long periods of database schema evolution and software component upgrade. Our mechanism can only duplicate the changed tuples, and then, through the specially designed temporary table, decide how to record the temporary data. When database schema evolution is completed, our mechanism maps and converts the temporary data to match the new schema. Therefore, the mechanism traps the user sessions before software component upgrade, ensuring that the temporary data processed are completed and that the session will not be terminated. The users will not be aware of the software maintenance. The performance evaluation results show that the service interruption time is shorter and the update time experiences a significant enhancement.

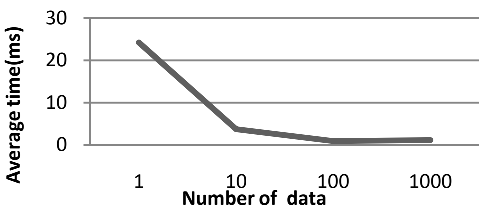

Figure 5: Average Processing Time for the MCM

ACKNOWLEDGEMENT

P. Lin's work was supported in part by NSC 101-2219-E-002-022, NSC 100-2221-E-002-176, NSC 100-2221-E-002-184, NSC 100-2219-E-002-015, NSC 101-2219-E-002-002, NSC 100-2219-E-007-010, National Science Council, National Taiwan University and Intel Corporation under Grants NSC 100-2911-I-002-001 and 101R7501, Chunghwa Telecom, and ICL/ITRI.

REFERENCES

[1] R. Delval, P. Bhat, F. Castro, and B. Cortright. (2007) An Oracle White Paper, October. [Online]. http://www.oracle.com/technology/products /ias/index.html

[2] Florin Sultan, Aniruddha Bohra, and Liviu Iftode, "Service Continuations: An Operating System Mechanism for Dynamic Migration of Internet Service Sessions," *Reliable Distributed Systems, IEEE Symposium on*, vol. 0, p. 177, 2003.

[3] Florin Sultan, Kiran Srinivasan, Deepa Iyer, and Liviu Iftode, "Migratory TCP: Connection Migration for Service Continuity in the Internet," , 2002, pp. 469--.

[4] Hung-Siang Chang et al., "Seamless On-Line Service Upgrade for Transaction-Based Internet Services," in *IEEE International Symposium on Parallel and Distributed Processing with Applications*.

[5] Raghu Ramakrishnan and Johannes Gehrke, *Database Management Systems*.: McGraw-Hill, Inc., 1999.

[6] Abraham Silberschatz, Henry F Korth, and S Sudarshan, *Database Systems Concepts*.: McGraw-Hill Higher Education, 2001.

[7] G H Sockut, T A Beavin, and C Chang, *IBM Systems Journal, title=A method for on-line reorganization of a database*, vol. 36, pp. 411 -436, 1997.

[8] Jørgen Løland and Svein-Olaf Hvasshovd, "Online, Non-blocking Relational Schema Changes," in *Advances in Database Technology*, 2006, p. 405.

[9] Ian Evans, Devika Gollapudi, Kim Haase, William Markito Oliveira, Chinmayee Srivathsa Eric Jendrock. (2012, Apr.) The Java EE 6 Tutorial. http://docs.oracle.com/javaee/6/tutorial/doc/index.html

[10] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi, "Analytic modeling of multitier Internet applications," *ACM Trans. Web*, vol. 1, no. 1, May 2007.

[11] Richard Helm, Ralph Johnson, John M. Vlissides Erich Gamma, Design Patterns: Elements of Reusable Object-Oriented Software, 1994, Addison-Wesley Professional.