Secure and Efficient Data Replay in Distributed eHealthcare Information System

Ken Qing Yang, Fellow of IEEE
Dept. of Electrical, Computer, and Biomedical Engineering
University of Rhode Island, Kingston, RI, 02881, USA

Abstract---This paper presents a new distributed data storage architecture that facilitates secure and efficient data replay in eHelathcare information system. The new architecture uses secured iSCSI protocol and records the parity of every data change to the eHealthcare information system performed by authorized health care providers. The recording and transmission of encrypted parities are done in background without requirement of explicit involvement of users. Together with either backup data or real-time production data, one can easily replay Electronic Health Record (EHR) as it was at any point-in-time in the past using simple reverse parity computations. Because parities are substantially smaller than original data, the new system provides much higher online performance and fast encryption time. Experimental results on our prototype system have shown the clear advantages of the new system.

I. Introduction

Rapid advances in computer technologies have made eHealthcare information systems ubiquitous. In such eHealthcare information systems, patient data are stored in a distributed environment allowing healthcare providers at different locations to share and access easily a variety of electronic health records (EHR). Such EHR will become essential source of information for future healthcare providers and rapidly take over the role of the paper-based medical records. Therefore, a reliable, secure, and efficient data storage infrastructure is critical to future healthcare systems. However, there are several technical challenges including reliability, security, and adequate online performance that make the design and implementation of such distributed data storage difficult.

The first technical challenge regarding data reliability has to do with the importance of having EHR data available to authorized healthcare providers once they have been created and recorded for a patient. Such EHR should not only be playable (viewable) in real time as it is currently but also be re-playable (reviewable) as it was at any point-in-time in the past [1,2,3]. Replay/review of the history of patient data is necessary because of situations of medical audit, law suits, quality control and self-assessment. The requirement of being able to replay EHR data makes the design of the eHealthcare information system challenging because of the fundamental differences between paper records and electronic records. With paper records, one can easily review the history by following the "paper trails" of the records. With electronic records, on the other hand, existing data storage designs do not have the "paper trails". Any change to a piece of data in the data storage is destructive because a data write operation will overwrite/destroy previous data in the same file or record. For example, any time when one saves or writes a changed word document or a spreadsheet file, the previous version of the file is overwritten and replaced. Similarly, a database transaction will also overwrite previous record of the same table. Even the meta data that records the time of last change or last access are also changed in a destructive way.

Realizing the importance of replaying history data, there has been extensive research in data storage and database systems in terms of data protection and recovery [4], file versioning [5,6,7], and database testing [8]. Data protection and recovery technologies periodically make backups or snapshots of data so that data can be recovered to a point-in-time in the past in case of failures or disastrous events. The granularity of backups/snapshots varies depending on the reliability requirement and cost. Continuous data protection (CDP) makes a copy of old data upon each write operation. CDP provides the finest granularity for data recovery at the cost of huge amount of data storage that is several orders of magnitude larger than the amount of normal real time data. File versioning systems keep different versions of files when file changes occur. The number of versions and the frequency of making file versions can be specified by users. In addition to the negative performance impacts, file versioning is file system dependent and requires users to be familiar with the file system. Database replays were originally designed for the purpose of database testing of production database systems that need upgrade or changes. By storing real transactions happening in production systems in a separate storage system, database replay makes testing of new database installation more realistic. Again, such database replays require users to explicitly define when and for how long to capture transactions in the production system. The major issue is that it is practically infeasible to enable SQL tracing on the entire database system because of high overheads [8,9].

The second technical challenge is data security and privacy of EHR system. Because data in an EHR

are stored and transmitted in a distributed environment over a network, data encryption and access authentication are very important to protect privacy of patient data. As it is well known, data encryption and decryption are very time consuming process especially for large amount of patient data. Supporting replay of EHR data aggregates this problem even further because the amount of data transmitted and stored to enable data replay is several orders of magnitude larger than production data due to repetitive overwrites [4]. As a result, online performance of such EHR system will be dragged down dramatically by storage systems supporting data replay and data security.

To tackle the technical challenges discussed above, we propose a new secured data storage architecture supporting replay/review of EHR data as it was at any point-in-time in the past by any authorized healthcare provider. The new system is referred to as REAPIT for **R**eplay **E**HR at **A**ny-**P**oint-**I**n-**T**ime . The main idea of REAPIT is to calculate and store parity as result of any data change at block storage level. As a data block is being changed, a log of parities are computed and stored. When a replay is requested for a specific point-in-time in the past, the corresponding parities are retrieved. Using either current real-time production data or a previous backup data, a simple reverse parity computation will generate the exact data as it was at the specified point-in-time. The clear advantages of REAPIT are three folds. First of all, the parity as a result of data change is substantially smaller than the data itself being changed. Capturing these parities can be done very efficiently. Transmission of these parities over the distributed network can also be done very quickly. Secondly, transmitting and storing parity add another level of security because wire tappers cannot easily interpret the meaning of parity without original data. Furthermore, encrypting parity is much faster than encrypting original data because of much smaller size of parities than that of original data. Finally, the new REAPIT system is user friendly because parity capture and transfer are done in background at storage level without requiring users' explicit involvement. The easy to use user interface allows a user to choose any point-in-time to replay EHR data.

To show the advantages of the newly proposed REAPIT system, we have carried out experiments on a prototype REAPIT system using Secured iSCSI protocol. Under the Microsoft Windows environment, we measured the performance of REAPIT as compared to traditional storage systems providing the same functionality. Standard database and file system benchmarks are used to drive the experimental system. Numerical results show that REAPIT is very efficient providing significant performance improvements compared to traditional storage systems. The executable program of the prototype REAPIT is available online at www.ele.uri.edu/hpcl for general public to repeat our experiments.

The paper is organized as follows. Next section introduces REAPIIT architecture and its design. Section 3 presents our experimental setup for performance evaluations. Numerical results and performance comparison are discussed in Section 4. Section 5 concludes the paper.

II. REAPIT System Design

Consider a distributed EHR system as shown in Figure 1. Multiple clients are connected to shared storage servers through the IPSEC network. Each of the clients represents an authorized healthcare provider such as doctors, radiologists, cardiologist, pathologist, dentists, dermatologist, neurologist, etc. Any client can access, modify, add, or delete information in the shared storages. Since IPSEC runs over the Internet, clients and storage servers can be located in any geographical location allowing true patient data sharing among healthcare providers. As shown in Figure 1, providing real time data sharing and data access is fairly straightforward among multiple health providers. The question is how to capture the history of data accesses and data changes securely in such a distributed information system.

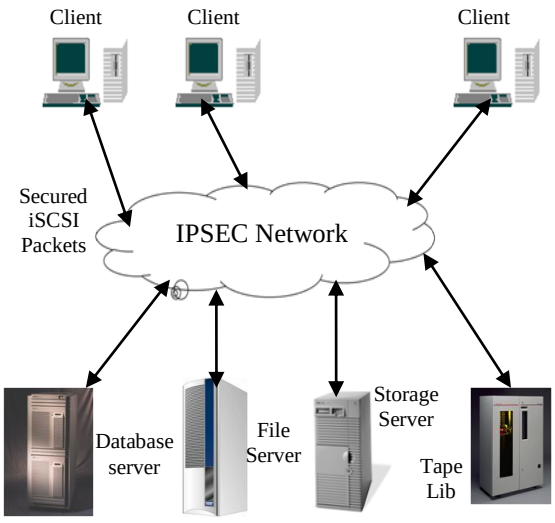

Figure 1. A distributed information system using iSCSI protocol.

REAPIT is a pair of software modules that are inserted to each client and shared storages, respectively. The software module inserted in a client is called iSCSI initiator and the module inserted in a shared storage system is called iSCSI target. Both modules work at block device level beneath the file

system or databases. Users do not need to know the existence of these modules. All IO operations generated at the upper application layer go through file system or database down to the block level device and become data block reads or writes. The iSCSI initiator representing client applications communicates directly with the corresponding iSCSI target through secured IP network by exchanging data blocks for real time operations. In order to support replay of history data in addition to real time data accesses, all data access and data changes need to be recorded. For example, if a 64KB (typical data block size) data block is changed, both the newly changed block and the original block before the change should be kept in order to replay data in the future. As data changes occur, the amount of data that need to be stored and transmitted over the network keeps increasing dramatically.

The idea of our new REAPIT is very simple. Instead of keeping all versions of a data block as it is being changed by write operations, we keep a log of parities as a result of each write on the block. Every time when a write operation happens, the iSCSI initiator calculates the parity and transmits the parity to the iSCSI target that store parities in a log structure. The parity logs are accessible to all clients for replay purpose. During a replay, the iSCSI initiator makes a replay request to the target with a specified time point. The target then finds the corresponding parity in the parity logs to compute back the data as it was at the specified time point. Since all parties of write operations are stored, our approach can replay EHR as it was at any point-in-time by parity computation. Figure 2 shows the basic design of the REAPIT architecture in a typical RAID storage system. Suppose that at time point i, the client writes into a data block with logic block address a_s that belongs to a data stripe $a=(a_1, a_2 \ldots a_s, \ldots a_n)$. The RAID controller performs the following operation to update its parity disk:

$$P_i(a)=F_i(a_s) \oplus F_{i-1}(a_s) \oplus P_{i-1}(a), \quad (1)$$

where $P_i(a)$ is the new parity for the corresponding stripe, $F_i(a_s)$ is the new data for data block a_s, $F_{i-1}(\alpha_s)$ is the old data of data block α_s, and $P_{i-1}(\alpha)$ is the old parity of the stripe. Leveraging this computation, REAPIT appends the first part of the above equation, i.e. $P_i'(\alpha) = F_i(\alpha_s) \oplus F_{i-1}(\alpha_s)$, to the parity log stored in the REAPIT disk after a simple encoding box, as shown in Figure 2. Parity computation can also be done easily for non RAID storage systems as discussed in our prior research [4].

Consider the parity log corresponding to a data block, α, after a series of write operations, the log contains ($P'_1(\alpha)$, $P'_2(\alpha)$ ……, $P'_{i-1}(\alpha)$, $P'_i(\alpha)$,……) with time points $1,2, \ldots,i-1,$ and i associated with the parities. Suppose that we only have the data image at time point r ($1 \le r \le i$) and all parities, and we would like to replay data backward or forward. To do a forward replay to time point s ($s>r$), for example, we perform the following computation for each data block α:

$$F_s(a)=F_r(a) \oplus P'_{r+1}(a) \oplus \ldots \oplus P'_{s-1}(a) \oplus P'_s(a), \quad (2)$$

where $F_s(\alpha)$ denotes the data value of block α at time point s and $F_r(\alpha)$ denotes the data value of α at time point r. Note that

$$P'_l(\alpha) \oplus F_{l-1}(\alpha)=F_l(\alpha) \oplus F_{l-1}(\alpha) \oplus F_{l-1}(\alpha)=F_l(\alpha), \quad (3)$$

for all $l=1,2, \ldots i$. Therefore, Equation (2) gives $F_s(\alpha)$ correctly assuming that the data value, $F_r(\alpha)$, exists.

The above process represents a typical redo replay process while earlier data is available. A backward process is also possible with the parity log if the newest data is available by doing the following computation instead of Equation (2):

$$F_s(a)= F_r(a) \oplus P'_r(a) \oplus P'_{r-1}(a) \oplus \ldots \oplus P'_{s+1}(a), \quad (4)$$

where $s<r$. This is a typical undo process by using the newest data that is available. In order to replay data in either direction, only one reference image is needed along time dimension because of the commutative property of XOR computation. This reference image could be original data image in a backup, real-time fresh data image, or any data image in the middle.

Besides being able to replay data in two directions, the important feature of REAPIT is its space efficiency. Our extensive experiments have demonstrated a very strong content locality that exists in real world applications and have shown that only 5% to 20% of bits inside a data block actually change on a write operation. The parity, $P_i'(\alpha)$, reflects the exact changes at bit level of the new write operation on the existing block. As a result, this parity block contains mostly zeros with a very small portion of bit stream that is nonzero. Therefore, it can be easily encoded to a small

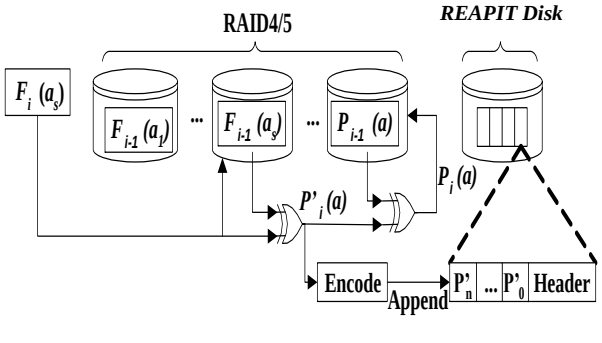

Figure 2. Block diagram of REAPIT Design.

size parity block to be appended to the parity log reducing the parity transmission time and the amount of storage space required to keep track of the history of writes.

III. Experimental Settings

In order to see how REAPIT performs in a real distributed eHealthcare information system, we have set up a cluster of four PCs connected through an Ethernet switch in our lab. Two Windows based PCs act as clients, one PC acts as an application server, and the 4th PC acts as a storage server. Our iSCSI initiator module based on the Windows standard iSCSI initiator is installed on the client PCs and application server PC while the iSCSI target module is installed on the storage server. The iSCSI initiators communicate with the iSCSI target on the storage server using IPSEC protocol. All encoded parities are stored at the iSCSI target in a log format with time stamps to allow replay of history data by any authorized client. Table 1 shows the hardware and software settings in our evaluation experiments.

PC 1-3	P4 2.8GHz/256M RAM/80G+10G Hard Disks
PC 4	P4 2.4GHz/2GB RAM/200G+10G Hard Disks
OS	Windows XP Professional SP2
	Fedora 4 (Linux Kernel 2.6.9)
Databases	Oracle 10g for Microsoft Windows (32-bit)
	MySQL 5.0 for Microsoft Windows
iSCSI	UNH iSCSI Initiator/Target 1.6
	Microsoft iSCSI Initiator 2.0
Benchmarks	TPC-C for Oracle (Hammerora)
	TPC-W Java Implementation
	File system micro-benchmarks
Network	Intel NetStructure 470T Switch
	Intel PRO/1000 XT Server Adapter (NIC)

Table 1. Hardware/Software settings of ouw experiemtns.

Because of privacy requirement of real patient data, we are not able to obtain realistic patient data for our experiments. In order to make our performance evaluation close to realistic situations, we have chosen a set of standard database benchmarks as well as real file system benchmarks.

The benchmarks we selected for database evaluations include TPC-C and TPC-W. TPC-C is a well-known benchmark used to model the operational end of real-time transactions [10]. TPC-C simulates the execution of a set of distributed and on-line transactions (OLTP) for a period of between two and eight hours. TPC-C incorporates five types of transactions with different complexity for online and deferred execution on a database system. These transactions perform the basic operations on databases such as inserts, deletes, updates and so on. At the block storage level, these transactions will generate reads and writes that will change data blocks on disks. For Oracle Database, we use one of the TPC-C implementations developed by Hammerora Project [11]. We build 5 data tables with 25 users issuing transactional workloads to the Oracle database following the TPC-C specification. The installation of the database including all tables takes totally 3GB storage.

TPC-W is a transactional web benchmark developed by Transaction Processing Performance Council that models an on-line bookstore [12]. The benchmark comprises a set of operations on a web server and a backend database system. It simulates a typical on-line/E-commerce application environment. Typical operations include web browsing, shopping, and order processing. Tomcat 4.1 is used as an application server and MySQL 5.0 as a backend database. The configured workload includes 30 emulated browsers and 10,000 items in the ITEM TABLE.

Benchmark	Brief Description
tar	Run 5 times randomly on ext2
gcc	Compile Postgres 7.1.2 source code on ext2
zip	Compress an image directory on ext2
Latex	Make DVI and PDF files with latex source files on ext2
cp/rm/mv	Execute basic file operations (cp, rm and mv) on ext2
Linux Install	Install Redhat 8.0 on VMWare 5.0 virtual machine
XP Install	Install Windows XP system on VMWare 5.0 virtual machine
App Install	MS Office2000 and VC++ on Windows
VC++ 6.0	Compile our *REAPIT* implementation codes

Table 2. File system benchmarks.

Besides benchmarks operating on databases, we have also formulated file system micro-benchmarks as listed in Table 2. The first micro-benchmark, *tar*, chooses five directories randomly on ext2 file system and creates an archive file using *tar* command. We run the *tar* command five times. Each time before the *tar* command is run, files in the directories are randomly selected and randomly changed. Similarly, we run *zip*, *latex*, and basic file operations *cp/rm/mv* on five directories randomly chosen for 5 times with random file changes and operations on the directories. The actions in these commands and the file changes generate block level write requests. Two compiler applications, *gcc* and V*C++6.0*, compile Postgress source code and our *REAPIT* implementation codes, respectively. *Linux Install*, *XP Install*, and *App Install* are actual software installations on VMWare Workstation that allows multiple OSs to run simultaneously on a single PC. The installations include *Redhat 8.0*, *Windows XP*, *Office 2000*, and

Visual C++ for Windows.

IV. Experimental Results

Based on the experimental settings discussed above, we measured total execution time needed to encrypt and transmit history data (EHR) to storage servers. Our intent is to compare the performance of REAPIT with the traditional storage system that supports replay of patient data at any point-in-time. As discussed previously, REAPIT captures the parity of data change while the traditional storage keeps both old copy and new copy of the data block upon a write operation to the block.

Our first experiment is running TPC-C benchmark on Oracle database. Continuous database transactions are performed for 1 hour period following the TPC-C specification. As results of these transactions, many data blocks are changed at storage level. The changed data are then encrypted and transmitted to the storage servers. Figure 3 shows the measured total execution time of data encryptions and transmissions over the IPSEC network to the shared storages. Five execution times are shown for 5 different data block sizes from 4KB to 64KB. Recall that the block is the basic data unit based on which data transmissions are performed at storage level (Windows default block size is 64KB). The execution times shown in the figure are extra overheads in order to support data replay for any point-in-time of patient history data. As shown in Figure 3, the traditional storage system takes prohibitively long time to support data replay with overheads ranging from about 177 seconds to 581 seconds for one hour's database transactions. Our REAPIT system, on the other hand, takes well under 20 seconds extra time to encrypt and transmit parities for 1 hour of intensive database transactions. In other words, REAPIT can support replay of patient data as it was at any point-in-time with no noticeable performance slowdown from users point of view.

Our next experiment is to measure the total execution time of TPC-W on Microsoft SQL databases. A fixed number of transactions are performed on the SQL database using both traditional storage system and REAPIT storage system by varying block size from 4KB to 64KB. The measured results are shown in Figure 4. Similar to TPC-C, we observed again an order of magnitude performance improvement of REAPIT system over the traditional storage system to support data replay of patient history data.

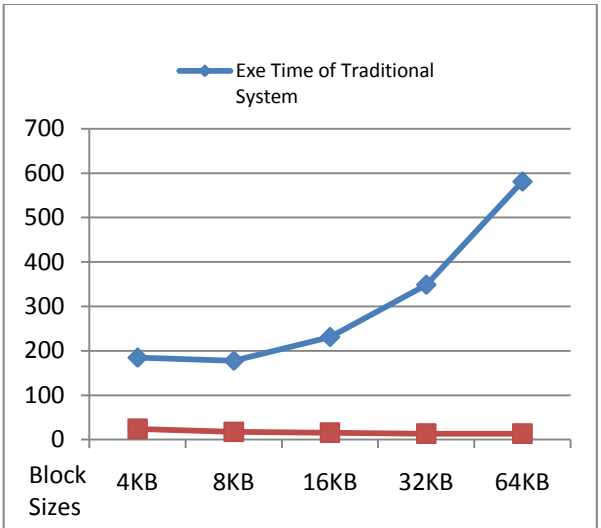

Figure 3. Comparison of total execution times for encryption and transmission of history data of 2 hours TPC-C transactions on Oracle databases in terms of seconds.

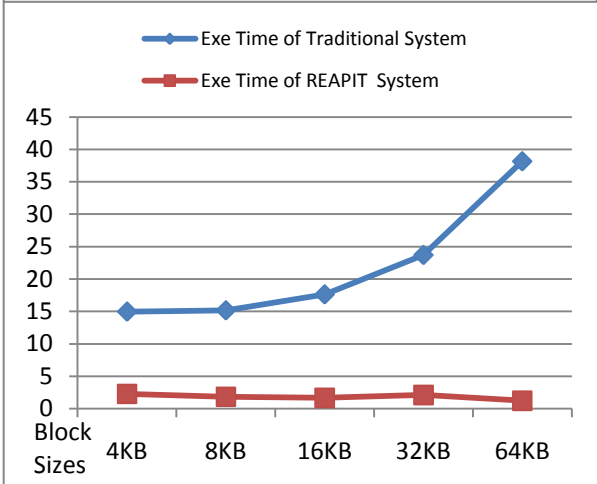

Figure 4. Comparison of total execution times for encryption and transmission of history data of 3,000 TPC-W transactions on Microsoft SQL databases in terms of seconds.

In addition to database applications, many eHealthcare information systems use file systems. In order to see how REAPIT performs on file systems, we carried out experiment on file system benchmarks as described in the previous section. We run these file system micro benchmarks and measure the extra execution time needed to encrypt and transmit history data. Figure 5 plots the measured results for different benchmarks. As shown in this figure, REAPIT substantially reduces the execution time. For all benchmarks considered, we observed 2 orders of magnitude performance improvement indicating the effectiveness of using parity logs to capture history data as opposed to keeping data themselves.

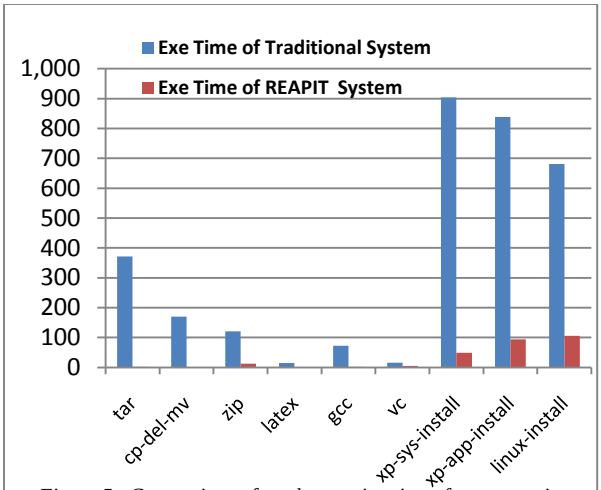

Figure 5. Comparison of total execution times for encryption and transmission of history data of file system micro benchmarks in terms of seconds.

V. Conclusions

This paper presents a new data storage architecture supporting data **R**eplay of **E**HR (Electronic Health Records) as it was at **A**ny **P**oint **I**n **T**ime of patient history data, referred to as REAPIT. By capturing parities resulting from data changes, REAPIT allows a user to replay patient data using simple parity computations. Storing and transmitting parities provide additional security and efficiency compared to data itself. Experimental results have shown orders of magnitude improvements over traditional storages.

Acknowledgements

The author thanks Mr. Jin Ren and Dr. Weijun Xiao for their help in prototype implementation and evaluation experiments. This research is supported in part by National Science Foundation under Grants CPS-0931820 and CCF-0811333. It is also partly supported by Natural Science Foundation of China under grant NSFC-60736013. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] A. R. Bakker, Access to EHR and access control at a moment in the past: a discussion of the need and an exploration of the consequences, *Int J Med Inform 73 (2004).: 267-270.*

[2] A. R. Bakker, "The need to know the history of the use of digital patient data, in particular the EHR," *Int' Journal of Medical Informatics 76 (2007), pp. 438-441.*

[3] Kudawashe Dube, "The Motion Picture Paradigm: Record, Play and Re-play in Data/Information," *SEAT, Massey University, available online at: http://www.massey.ac.nz/~kdube/main/wp-content/uploads/2009/11/mpp-problem-massey1.pdf*

[4] Weijun Xiao and Qing Yang, "A Case for Continuous Data Protection at Block Level in Disk Array Storages" *IEEE Transactions on Parallel and Distributed Systems*, Volume 20, Issue 6 (June 2009), Pages 898-911.

[5] K. Muniswamy-Reddy, C. P. Wright, A. Himmer, and E. Zadok, "A versatile and user-oriented versioning file system," In *Proc. of the 3rd USENIX Conference on File and Storage Technologies*, San Francisco, CA, 2004.

[6] Z. Peterson and R. C. Burns, "Ext3cow: A Time-Shifting File System for Regulatory Compliance", *ACM Transactions on Storage*, Vol.1, No.2, pp. 190-212, 2005.

[7] S. Quinlan and S. Dorward, "Venti: a new approach to archival storage," In *Proc of the 2002 Conference on File and Storage Technologies*, Monterey, CA, Jan. 2002, pp. 89-101.

[8] Wang, Y., Buranawatanachoke, S., Colle, R., Dias, K., Galanis, L., Papadomanolakis, S., and Shaft, U. 2009. Real application testing with database replay. *In Proceedings of the Second international Workshop on Testing Database Systems* (Providence, Rhode Island, June 29 - 29, 2009). 1-6.

[9] SQL server profiler, RDBMS documentation, available online: msdn.microsoft.com.

[10] Transaction Processing Performance Council, "TPC BenchmarkTM C Standard Specification," 2005, http://tpc.org/tpcc.

[11] S.Shaw, "Hammerora: Load Testing Oracle Databases with Open Source Tools," 2004, http://hammerora.sourceforge.net.

[12] H.W. Cain, R. Rajwar, M. Marden and M.H. Lipasti, "An Architectural Evaluation of Java TPC-W," *HPCA 2001*, Nuevo Leone, Mexico, Jan. 2001.