A Flexible Replication Mechanism with Extended Database Connection Layers

Nobutatsu Nakamura[1,2], Ken'ichiro Fujiyama[1], Eiji Kawai[2] and Hideki Sunahara[2]
[1]NEC Corporation, [2]Nara Institute of Science and Technology
n-nakamura@ab.jp.nec.com

Abstract

It is vital to achieve a disaster recovery system that allows a backup site to take over a primary site's IT services while the primary site is down. We propose a flexible replication mechanism based on service requirements such as system performance, recovery time objective (RTO), and recovery point objective (RPO). For high flexibility, the mechanism controls the replication schedule by monitoring the application's database accesses in the database connection library and matching the accesses with previously registered access patterns. In our experiments, we confirmed that the proposed mechanism outperforms other existing mechanisms, especially in situations with network delays and packet losses.

1. Introduction

Now that IT systems have become indispensable for business, it is vital to prepare a backup site located where it can continue the same IT services if the primary site stops because of a disaster, subversive activity, etc. System managers usually create a backup on another media such as a tape, but setting up a new host and restoring data from the backup will take time and effort. There are several proposed approaches for quick recovery in order to continue IT services at backup sites after the primary site stops. They usually employ concurrent replication and synchronization between the primary and backup sites.

Existing data replication techniques are roughly divided into two categories: Primary Copy and Update Everywhere [24]. In Primary Copy, data at the primary site are updated first, and then the data at the backup sites are updated. From the viewpoint of application programs, this approach makes no differences in their processing flows. Therefore, no source code modifications for the application programs are necessary to implement the replication features. In addition, since communication with the backup sites can be scheduled independently of updating data at the primary site, network performance is not a critical issue. On the other hand, synchronization between the primary and backup sites is difficult, and thus disaster tolerance is limited.

In Update Everywhere, an update query is simultaneously sent to the primary and backup sites. When hardware and software are not reliable and service often halts, this method is effective because both sites always have the same process status, and the backup site can simply replace the primary site. In other words, Update Everywhere has better disaster tolerance than Primary Copy. However, this method offers lower performance because replication one by one requires more time and cost.

In this paper, we propose a novel mechanism for replication with high flexibility in the balance between the system performance and reliability. Although our approach is based on Update Everywhere, it does not adopt to the one-by-one updating approach. Our idea is to control the replication schedule, which is determined by the semantics of the updating queries. This is reasonable, since not all queries require synchronization of the updates at both sites. There is usually a trade-off between system performance and recovery requirements such as the recovery time objective (RTO) and the recovery point objective (RPO), which can be optimized in our proposed mechanism by controlling the replication schedule. For replication schedule control, we extend the database connection library in the application layer, where the examination of each query is easy and therefore queries can be replicated easily for both the primary and backup sites.

Today, many system managers are greatly concerned about cost efficiency [4], and they often prefer the Internet to leased lines. For this reason, network delays and packet losses should be considered within the system design. In addition, although commodity PCs and storages are often employed because of their cost efficiency, hardware failure should be considered. Our proposed system can cut both hardware and network costs because the replication data size is reduced by controlling the replication schedule based on the semantics of each updating query.

The remaining part of this paper is organized as follows. In Section 2, we describe existing data replication approaches and discuss their pros and cons.

We then propose our replication system in Section 3 and give detailed descriptions about the replication and the recovery in Section 4 and Section 5, respectively. We conducted experiments for performance evaluation and the results are shown in Section 6. We offer a short discussion on related works in Section 7 and finally outline our conclusions in Section 8.

2. Location of Replication

When recovering the halted primary site's service at the backup site, the backup site must have the same data as the primary site. Quick recovery with continuous service depends on how crucial data are synchronized between the primary and backup sites. There are many approaches to synchronization by replicating data from the primary site to the backup sites. In this section, we describe each approach and discuss the pros and cons.

To replicate data while continuing services, the data or processing command must be replicated at a certain point in the course of the processing flow.

To maintain consistency between the primary and backup sites, the processing order of the request queries at both sites must be identical. If the order of request queries is different at a site, the storage will probably have a different status. Herein, if multiple request queries are issued concurrently at the upper layer, the lower layer data may be indefinite and depends on various factors that affect the processing order of the queries such as operating system implementation, system load, and network condition. However, from the viewpoint of service, in order to continue the service, it is enough to assure a logically identical status at the primary and backup sites.

The processing order is usually not guaranteed on the upper layer of the service. For example, consider a case where there are two sets of application servers (and therefore two sets of a database server) operating separately in the primary and backup sites, and the client requests that the application servers at the primary site are duplicated and also sent to the servers at the backup site. In this situation, it is difficult to ensure identical processing orders between the two servers. Since a multiple application server setup is commonly employed for load balancing, requests are distributed among them and processed in parallel.

The processing order can usually be assured on the storage and database layers. In the storage layer, there is no process that interrupts storage access, and therefore it is the safest way to replicate data synchronously. Actually, storage layer replication is widely used not only in database systems but in other IT systems. However, it is difficult to achieve storage layer replication from a distant location. This is because storage layer I/O throughput is much higher than that of the wide area networks. Even when the transaction data in the database layer is small enough, the I/O data in the storage layer can be huge.

Storage layer replication presents another problem – restoration reliability during a disaster. For IT service continuity, consistency between the data and the program must be guaranteed. In other words, to resume a stopped program, both saved data and program status are necessary. To save the program status, we need some other mechanisms in addition to the storage layer replication.

There are two types of database layer replication [8]: replication inside the database and replication outside the database. Several database products have provided the former as a replication feature in which vendor specific transactions are replicated to the backup database server. However, until several years ago, such replication features in database systems had been provided only in high-end versions of commercial systems [14, 10].

Recently, a replication feature has also been provided in several free database systems, however, some restrictions still exist. MySQL Cluster [18] and Postgres-R [13] add a replication feature. To introduce this feature, modification of the program source codes is required. PGCluster [25] can add a replication feature with no modification, but it becomes difficult to manage the database engine and the distributed data because PGCluster changes the original PosgreSQL database engine.

For replication inside database as described above, there is another problem – replication increases the load of the database system. Typical IT systems that consist with a two-layer structure (or a three layer including a web layer) often employ multiple application servers for load balancing and a single database system for consistency. Therefore, the database server can easily become a performance bottleneck, meaning the replication approach inside a database is often impractical.

Pgpool [11] and C-JDBC [3] are well known techniques as outside database replication method. For example, pgpool achieves a replication feature by adding a proxy to PostgreSQL and replicating the database accesses. However, there are problems in that the proxy is a new single point of failure and may become a performance bottleneck. Adding a new serial node like a proxy to the system means great disadvantages in terms of reliability and availability. C-JDBC also needs a C-JDBC controller module as an extra serial node.

We can conclude that a database outside replication is the best approach for disaster recovery for the following reasons: 1) the processing order can be easily assured; 2) the query semantics can be examined; 3) backup can be saved with the appropriate program status; 4) the impact on the database server load is low; and 5) modification of the application program source codes is unnecessary. However, we do not adopt the proxy method because it adds a new serial node. Our approach involves extending the database connection library and selectively replicating the query requests.

3. Replication System

3.1. Extension of Database Connection Library

A database connection library is commonly placed between a database system and an application. The library provides unified APIs that provide for database-independent accesses by virtualizing specific database connection procedures. ODBC [7], JDBC [16], and ADO.NET [19] are well-known database connection libraries. After the application (servelet) issues a database-independent request, the database connection library changes it into a database-specific request and sends it to the database. Then, it receives a response from the database and changes it into a database-independent response.

Figure 1 shows how the database connection library is extended in our system. The database connection library sends the application's access request to multiple databases, receives their responses, and returns one response to the application.

We implemented the extended library and conducted preliminary experiments with various databases and benchmark applications [6]. In our experiments, when the Internet was used for long distant communication, the system performance was poor because of communication delays or packet losses.

Although system requirements such as system performance, recovery time objectives (RTO), and recovery point objectives (RPO) depend on services and applications, system performance is sometimes favored over reliability since the disaster risk is very low. In this case, exact synchronization is not expected; service may not be recovered immediately, and it may even restart without the most recent data.

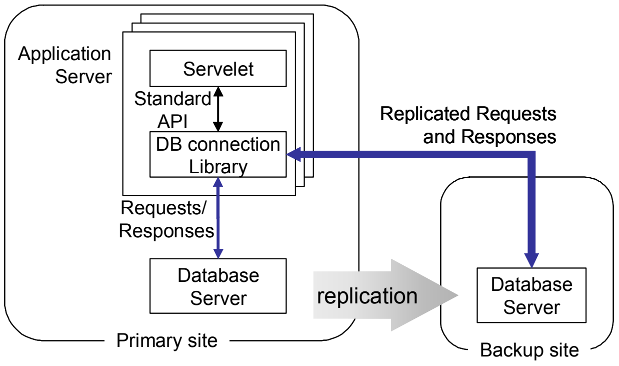

Figure 1: Data Replication Using an Extended Database Connection Library

The goal of this study is to achieve a system that can flexibly control trade-offs between performance and reliability. In our proposed system, some database accesses are exactly synchronized for higher reliability and some are only assured of their processing order for higher performance. We will describe the details of the control mechanism in Section 4.

3.2. System Overview

Figure 2 shows the basic system configuration. The system consists of the primary and backup sites. Both sites have an application server and a database server. The application server in the primary site consists of application processes that receive requests and send responses, a database connection library extended by a site-to-site communication module, and a database driver for the local database. The communication module utilizes a database driver to access the remote (backup) database and maintains the access log storage.

The application server in the backup site consists of a remote manager for communications with the primary site, logging storage, application processes, a database connection library, and a database driver. The application processes, the database connection library, and the database driver are usually inactive; they are only activated when a disaster occurs at the primary site. Last, both database servers consist of a database, an OS (file system) and storage module.

4. Data Replication

4.1. Replication Control Mechanism

In the application server at the primary site, the standard database connection library is replaced with our extended one. Our database connection library has compatible APIs with standard ones, although it multiplies a database connection. One of the multiplied database connections is connected to the primary database through the database driver. Another database connection is linked to the communication module. Basically, the connection is bound to the backup database through the database driver. Some requests sent to the primary database are simultaneously sent to the backup databases in order to make the logical status of both databases consistent.

Access logs are used for database restoration by resubmitting the query. In the communication module, access logs are saved at the primary site and are transferred through the remote manager. Access logs are also saved at the backup site. We will discuss how the access log is used in Section 5.2.

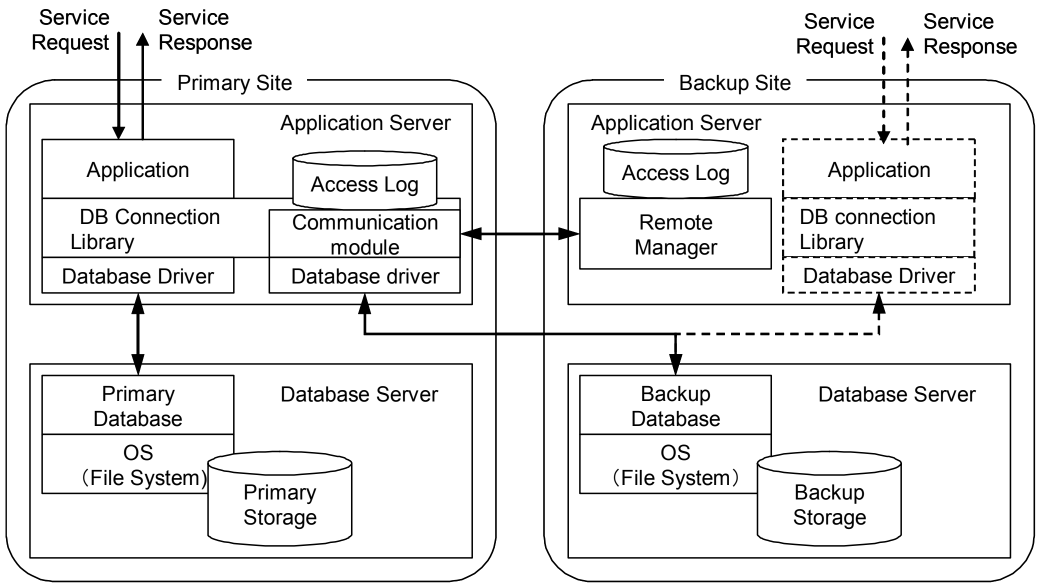

Figure 2: Proposed System Overview

Next, we give the details of the replication mechanism for synchronization of the primary and backup databases. In general, to utilize a database system, first the database has to be connected, and then transactions such as "execute" and "commit" are performed. Finally, the connection to the database is closed. Each transaction often involves a large number of database accesses whose replications and process order assurances are necessary for strict synchronization. However, some accesses may be skipped according to the access pattern, which is a series of several accesses. In our proposal, accesses are classified into four types: "normal", "synchronized", "order-assured", and "skipped". Each access type provides different replication processes.

Figure 3 depicts a sequence chart of the database access types. Accesses A, B, C, and D are "normal", "skipped", "order-assured", and "synchronized", respectively.

When there is a request from the application in the "normal" access A, our library accesses both the primary and backup databases. After it receives a response from the primary database, it returns the response to the application. Although the library receives a response from the backup database, no special action is required.

In the "skipped" access B, an access is performed only at the primary database, and the library does not transfer it to the backup database.

In the "order-assured" access C, the database connection library puts a marker (a unique and sequential number) in the access as identification. This unique and sequential number can be easily obtained by several means. An example is utilizing a server that generates a sequential number. The marker shows where the application proceeds and will be used in the restoration operation for disaster recovery. After one response is received from the primary database, it returns a response to the application.

In the "synchronized" access D, the database connection library also adds a marker as an identified database access with a unique number. After the responses are received from both the primary and backup databases, responses are returned to the application.

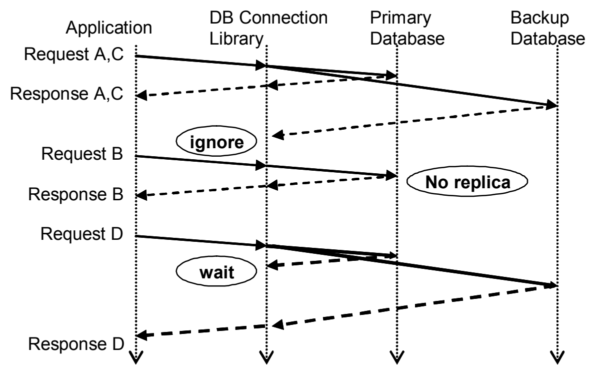

Figure 3: Database Access Sequences

4.2. Database Access Patterns

In this subsection, we give a detailed description of possible inconsistencies between the databases. Table 1 shows the status combination of the primary database, the backup database, and the application when a disaster occurs while an access from the application is being processed on each database through the database connection library.

The simplest cases are cases 1 and 6. In those situations, services can be continued without any special processes because there is no inconsistency among the primary database, the backup database, and the application.

	Primary DB	Backup DB	Application
Case 1	**completed**	**completed**	**completed**
Case 2	**completed**	not yet	**completed**
Case 3	**completed**	not yet	not yet
Case 4	**completed**	**completed**	not yet
Case 5	not yet	**completed**	not yet
Case 6	not yet	not yet	not yet

Table 1: Status of Primary/Backup Databases and an Application

In case 2, the access is completed at the primary database and the application, but is not completed at the backup database. Therefore, it is necessary for the access to be reissued at the backup database. Here, there is an exception; if the access does not affect the system status, it can be ignored, and the services can continue.

In case 3, only the primary database completes the access. In this scenario, even if the primary database is lost in the disaster, consistency between the application and the backup database is maintained, and services can continue.

In cases 4 and 5, the access has already been processed at the backup database, but the application has not yet discovered the processing status. For rigorous database transaction processing, a rollback should be made based on the application side status. However, for most commercial database products, once a “commit” access is issued, access is fixed even if an “ack” response does not reach the application. At this time, services can continue.

Case 2 must be avoided when the access affects the system status. In our system, such accesses have been previously registered as access patterns. If an access matches one of the registered access patterns, the “synchronized” access described in Section 4.1 is applied.

In this system, the system operator registers the access patterns beforehand, and the “normal”, “skipped”, “order-assured”, and “synchronized” accesses are processed according to the matching results with the access patterns. The default is “normal” access processing.

For example, a “skipped” access pattern corresponds to retrieving accesses. An “order-assured” access pattern corresponds to executing accesses with an exclusive lock in each transaction. A “synchronized” access pattern corresponds to committing accesses.

A flexible configuration among the system performance, RTO and RPO can be achieved by modifying the registered access patterns. When “synchronized” accesses occur frequently, the total waiting time for the backup database replies is increased, causing low performance. Therefore, depending on the system reliability requirements, access patterns are selectively registered.

For example, in a seat reservation service system that keeps 10 spare seats, synchronization can be made every 10 access requests because at most 10 spare seats can be assigned to the unsynchronized seat reservations in the recovery procedures. Another example is a commercial site that has agreed to deliver a product within 24 hours. If manual processing for each access takes 10 minutes, synchronization every 144 access requests meets the requirements.

5. Disaster Recovery

5.1. Failover

When a disaster occurs at the primary site, application processes are activated at the backup site using the data in the backup database to continue the service. However, as we mentioned, the backup database is not completely synchronized with the primary database. Therefore, a failover operation, i.e., a restoring process is necessary.

First, rollbacks are made until the latest point where processing is guaranteed in the backup database. In this system, since a marker is inserted in some accesses, the rollback point equals the marker in the latest “synchronized” or “order-assured” access in the log. At this point, the application starts up and service is resumed on the backup site.

5.2. Resynchronization

In some several cases where either a disaster occurs in the backup site, the network between the primary and backup sites fails, or the system is updated for hardware/software maintenance, the backup database has a different status from that of the primary site. In this case, a resynchronization process is necessary.

In slight fault cases, such as a temporal database disconnection, the recovery procedure is conducted in the following way. First, rollbacks are made to the latest marker’s point. The remote manager puts the accesses after the marker into the backup database. These accesses are found in the log. While they are being processed at the backup site, service continues at the primary site, and new access logs are accumulated. If the performance of the primary database is better than that of the backup database, the backup database processing point may not be able to catch up to the primary database process. In this situation, the primary database process rate must be limited. Finally, when the backup data process catches up to the primary database process, service at the primary site momentarily pauses and the backup database is connected via the database connection layer for normal operation.

When data are lost by disk faults and so on, database restructuring is necessary. In this case, the file system and the database must be restored using some full disk backup tools such as snapshot. After the database starts up, the remote manager connects to the database and puts the recorded accesses. From this point, the processing flow is identical to slight fault cases.

6. Experiments

6.1. Implementation

We implemented our disaster recovery mechanism in JDBC, a database connection library in Java. In this experimental system, a replication controlling mechanism described in Section 4.1 and database access control according to the data access patterns in Section 4.2 were implemented. In this section, we give a brief description of the system implementation.

JDBC is a standard database connection library used by most Java applications and servelets. It consists of a JDBC driver and a JDBC driver manager. The application calls a JDBC driver manager who provides standard APIs. The JDBC driver depends on the database, and therefore, it is usually provided by the database vendor. We extended the part that loads the JDBC driver into the JDBC driver manager. Our extensions for various functions such as database access replication, loading two JDBC drivers, and two databases connections are achieved in the extended JDBC driver manager. We call this extended library a "JDBC gateway."

In this mechanism, access patterns are specified beforehand. Then, the JDBC gateway matches the database accesses with the access patterns and determines the access type ("normal", "order-assured", "synchronized", or "skipped"). In this experiment, we specified several JDBC methods as access patterns. *Statement.executeQuery()* and *Statement.executeUpdate()* are "order-assured" access patterns and *Connection.commit()* is a "synchronized" access pattern.

6.2. Experimental System

The configuration of the experimental system is shown in Figure 4. The server OS was Linux and the database was PostgreSQL 8.0. We used an IBM OLTP benchmark toolkit V2.0 based on TPC-C [22]. NIST Net [2] emulated the Internet with delays and packet losses. In the experiments, the benchmark application ran while changing the NIST Net parameters of delay time and packet loss rate. We compared the performances of the following cases: remote access without replication, remote access via NFS without cache, replication using PGCluster [25] (as replication inside database), replication using pgpool [11] (as replication outside database), replication using the proposed method without access control (JDBC-GW), and replication using the proposed method with access control (Managed JDBC-GW). In the pgpool system configuration, a proxy server was located at the application server and connected to the benchmark application by local socket communication.

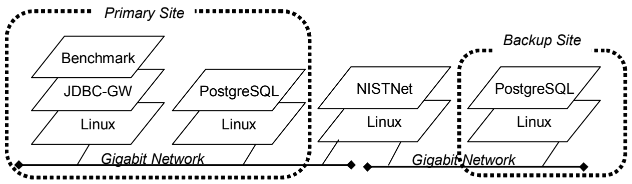

Figure 4: Experimental System

6.3. Experimental Results

Results are shown in Figure 5. In the experiments, the round-trip delay time in NIST Net is set at 0, 1, and 10 ms, and packet loss rate was set at 0 %, 0.1 %, and 1 %.

The vertical axis of the graph is the database throughput in the number of transactions per second. The throughput in the local database access with no replication case was 11.4 transactions per second. In the remote database access with no replication cases, the delay and drop parameters greatly affected the performance.

In the NFS case, the performance was very low. It seems that there were many file system accesses inside the database and they causeed a bottleneck.

In the PGCluster and pgpool cases, each performance was less than 50% of the remote access with no replication. In the JDBC-GW case, the performance was 70%-90% of the remote access with no replication. On the other hand, the managed JDBC-GW was proved to have a great tolerance to network delays.

The behaviors at a failure for each cases are as outlined below. In the PGCluster, pgpool and managed JDBC-GW cases, the benchmark program stopped once after a emulated primary database failure. However, the benchmarks could be restarted using the duplicated backup database server data after the benchmarks connected to the backup database server again. In the JDBC-GW (without access control) case, the backup database server data was updated and the benchmarks continued without stopping. That means the benchmarks were not influenced at all when the primary database server access stopped. All queries including reference had to be duplicated for non-stop service.

Replication Method	delay (ms)	drop (%)	Tx/sec	Replication Method	delay (ms)	drop (%)	Tx/sec
Remote Access (no replication)	0	0	11.42	pgpool	0	0	5.07
		0.1	10.51			0.1	4.29
		1	4.78			1	1.67
	1	0	8.66		1	0	4.01
		0.1	8.08			0.1	3.68
		1	4.31			1	1.3
	10	0	2.85		10	0	1.33
		0.1	2.76			0.1	1.35
		1	2.16			1	0.86
Remote Access NFS (noac)	0	0	1.13	prposed method without access control	0	0	7.61
		0.1	1.07			0.1	7.16
		1	0.76			1	3.93
	1	0	1.1		1	0	6.41
		0.1	0.97			0.1	6.06
		1	0.78			1	3.65
	10	0	0.74		10	0	2.6
		0.1	0.66			0.1	2.53
		1	0.59			1	1.94
PGCluster	0	0	4.1	prposed method without access control	0	0	8.15
		0.1	3.64			0.1	8.48
		1	2.5			1	5.76
	1	0	3.62		1	0	7.18
		0.1	3.48			0.1	7.07
		1	2.04			1	4.78
	10	0	1.8		10	0	3.89
		0.1	1.72			0.1	3.76
		1	1.45			1	2.9

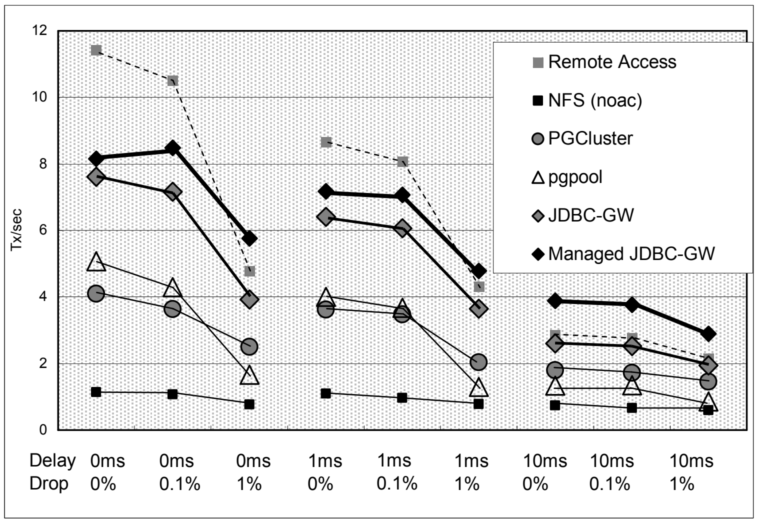

Figure 5: Experimental Results

7. Related Work

As a data replication technique for system reliability, redundant arrays of inexpensive disks (RAID) [15] is well-known. Although RAID is applied in one computer system, a network-extended mechanism, called RADD, is also proposed [20]. Although RADD is applicable only to a local area network, it can be a basis for creating a fault tolerant system by preserving data among computers. Even if one computer in the system stops, data can be recovered. However, the performance of RADD over wide area networks has not been examined.

Next, from the viewpoint of data process flow, we can classify existing replication techniques by their processing layers. First, as network layer replication, a replication technique is proposed that captures the network communications between a NFS server and clients and duplicates them for a backup server [21]. This method has no influence on system performance if it utilizes a different monitoring host and different networks between the monitoring host and the backup NFS server. However, consistency is difficult to maintain because the primary NFS server and the clients do not know about errors in the backup server. In addition, packet losses make the situation more complicated, and therefore it is difficult to use in a system where high reliability is required.

For storage layer replication, iSCSI based replication [4] has recently become an active research area. This is because they can use the Internet to reduce network costs instead of exclusive lines such as lease lines. However, since small data chunks are frequently exchanged between the primary and backup sites, network delays greatly impact on system performance. We cannot expect practical performance between distant sites.

They can, however, be used in a data center where all servers are consolidated and connected via high performance networks. However, there are performance problems while connected to slower or more distant networks.

Log file system techniques such as Spiralog [12] and ReiserFS [17] are related to file system replication and system reliability. In these systems, file access logs are preserved and used to recover a file when it is broken because of some system fault. File access logs are redundant data that can be regarded as replication metadata. Although log file systems have enough reliability to recover files, extra processes are necessary to recover the status of the application and the database system. Recently, a snapshot feature is provided by several file systems such as VxFS [23]. This feature is also available in our proposed mechanism.

For database layer replication, there are many research literatures published in a distributed database area. Database replication techniques [1, 25] improve availability and fault tolerance. However, stable networks among distributed databases are assumed, and therefore they are not designed to be applied to the Internet. Meanwhile, in the grid technology, widely distributed computers are assumed. In the distributed parallel storage system (DPSS) [5] and the CMS data grid system [9], fast and reliable data copy can be achieved, but system recovery requirements such as RTO and RPO improvements are impossible or difficult to define.

8. Conclusion

In this article, we proposed a flexible disaster recovery mechanism. By extending a common database connection library, backup database accesses for replication are controlled by matching the accesses with registered access patterns. By changing the registered access patterns, the backup database is sometimes updated synchronously or sometimes updated with processing order assurance. Thus, system managers can configure systems to fulfill the RTO, and RPO requirements of their system performance.

In the experiments, our proposed mechanism was proved to have a large tolerance to both network delays and packet losses. With 10 ms of RTT and 0.1 % of packet loss ratio, our proposed mechanism outperforms the conventional mechanisms 2.2 times (ours: 3.76 transactions/s, PGCluster: 1.72 transactions/s).

For future work, we plan to develop an advanced order-assurance method for a system in which multiple application servers are connected via a load balancer.

References

[1] Carey, M. J., and Livny, M., "Distributed concurrency control performance: A study of algorithm, distribution, and replication", Proc. 14th VLDB Conf., pp. 13-15, (1998)
[2] Carson, M. and Santay, D, "NIST Net: A Linux-based Network Emulation Tool", ACM SIGCOMM Computer Communications Review, Vol. 33, No.3, pp. 111–126, (2003)
[3] Cecchet, E., Marguerite, J., and Zwaenepoel, W., "C-JDBC Documentation", http://c-jdbc.objectweb.org/doc/index.html
[4] Chang, F., Ji, M., Leung, S. A., MacCormick, J., Perl, S., and Zhang, L., "Myriad: cost-effective disaster tolerance", Conference on File and Storage Technologies, (2002)
[5] Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., and Tuecke, S., "The data grid: Towards architecture for the distributed management and analysis of large scientific datasets, (1999)
[6] Fujiyama, K., Nakamura, N., and Hiraike, R., "High-Availabile Database Transaction Management Framework on Cluster" SWoPP 2005, (2005), (in Japanese)
[7] Geiger, K., "Inside ODBC", Ascii, (1996)
[8] Gray, J., Shasha, D. et al., "The Dangers of Replication and a Solution", Proceedings of the 1996, ACM SIGMOD International, (1996)
[9] Holtman, K., "Cms data grid system overview and requirements", CMS collaboration, (2001)
[10] IBM, "DB2: Replication Guide and Reference", (1999)
[11] Ishi, T., pgpool page, http://pgpool.projects.postgresql.org/, (2005)
[12] Johnson, J. E. and Laing, W. A., Overview of the Spiralog file system, Digital Technical Journal,
Vol. 8, No. 2, pp. 5–14, (1996)
[13] Kemme, B. and Alonso, G., "Don't be lazy, be consistent: a new way to implement Database Replication", Proceedings of the 26th International Conference on Very Large Databases, (2000)
[14] Oracle, "Oracle9i Advanced Replication", Oracle white paper, (2002)
[15] Patterson, D. A., Gibson, G., and Katz, R. H. "A case for redundant arrays of inexpensive disks (RAID)", Proceedings of the 1988 ACMSIGMOD international conference on Management of data, pp. 109–116, (1988)
[16] Reese, G., "Database programming with JDBC and Java", O'Reilly & Associates Inc, (2000)
[17] Reiser, H., ReiserFS whitepaper, http://www.namesys.com/, (2001)
[18] Ronstrom, M. and Thalmann, L., "MySQL Cluster Architecture Overview", MySQL Technical White Paper, (2004)
[19] Sceppa, D., "Programming Microsoft ADO.NET", NikkeiBP Press, (2002)
[20] Schloss, S. M., "Distributed raid - a new multiple copy algorithm", 6th Intl. IEEE Conf. on Data Eng IEEE Press, pp. 430-437, (1990)
[21] Tanemura, M., Shinjo, Y., Chiba, S., and Itano, K., "An Implementation of the Backup System Using Network Monitoring Technology", IPSJ Computer System Symposium, Vol. 2001, No. 16, pp. 57-64, (2001), (in Japanese)
[22] Transaction Processing Performance Council(TPC) benchmark C, standard secification revision 5.2, (2003)
[23] Veritas Software, http://www.veritas.com/,
[24] Wolfson, O. and Milo, A., "The multicast policy and its relationship to replicated data placement", ACM Transaction on Database Systems, vol. 16, no. 1, pp. 181-205, (1991)
[25] PGCluster, http://pgcluster.projects.postgresql.org/