이기종 시스템에서 다층 구조를 통한 DBMS 대용량 데이터 로딩의 분석 및 평가

진기원* · 임효택**

Analysis and Evaluation of DBMS Bulk Data Loading Through Multi-tiered Architecture for Heterogeneous Systems

Hee-Yuan Tan* · Hyotaek Lim**

요 약

여러 과정을 통해 만들어진 수많은 데이터는 효율적으로 관리되기 위해서 DBMS의 도움이 절대적으로 필요하다. 네트워크 내부 또는 외부에서 오는 이러한 데이터는 실시간 또는 일괄적으로 데이터베이스에 삽입(insertion)이 된다. 다량의 데이터 삽입은 흔히 개별적인 DBMS에서 지원하는 특정 대용량 로딩 또는 삽입기능을 통해 수행된다. 본 논문에서는 다층 구조로 구성되어 있는 이기종 시스템의 대용량 데이터 로딩을 분석하고 평가하며 프로그램 삽입에서의 DBMS 대용량 로더의 결과와 비교한다. 또한 대용량 로딩의 성능을 쉽게 향상 시킬 수 있는 staging 데이터베이스를 사용한 하이브리드 방법을 제안한다.

ABSTRACT

Managing the growing number of data generated through various processes requires the aid of Database Management System (DBMS) to efficiently handle the huge amount of data. These data can be inserted into database in real time or in batch, that come from multiple sources, including those that are coming from inside and outside of a network. The insertion of large amount of data is commonly done through specific bulk loading or insertion function supplied by each individual DBMS. In this paper, we analyze and evaluate on handling data bulk loading for heterogeneous systems that is organised as multi-tiered architecture and compare the result of DBMS bulk loader against program insertion from a software development perspective. We propose a hybrid solution using staging database that can be easily deployed for enhancing bulk loading performance compared to insertion by application.

키워드

데이터베이스 관리 시스템 (DBMS), 대용량 로딩, 이기종 시스템, 다층 구조

Key word

Database Management System (DBMS), Bulk Loading, Heterogeneous Systems, Multi-tiered Architecture

* 동서대학교 일반대학원 석사과정
** 동서대학교 컴퓨터정보공학부 교수(교신저자)

접수일자 : 2009. 10. 05
심사완료일자 : 2009. 12. 08

Ⅰ. Introduction

With the growing number of work done using computerized method in an organization, the data generated from various processes increase significantly. For storing data on a smaller scale, a normal flat file or using spreadsheet might be sufficient for a number of applications, but once the data grows larger and data need to be shared between computers, file based storage does not scale well, and often requires leveraging on database management system (DBMS) for better handling of data storage and maintenance. Data in the initial stage come mainly from user input, and newer data in later stage are generated from different operations including computation, extrapolation, aggregation and manipulation of existing data. These newer generated data can be saved into another data store for further analysis without needing a data re-entry from users via bulk data insertion or bulk loading.

Bulk data loading into database can be carried out using various approaches, and each approach has its advantages and disadvantages. Since choice of bulk loading technique affects the performance of an application that relies on transferring huge amount of data, various considerations need to be taken into account for a software development project. In many client/server applications for example, users do not commonly access the data repository directly, but often the data flow through multiple intermediate layers or multi-tiered system architecture [1] before reaching to the end user. Figure 1 shows an example of multi-tiered architecture commonly found for a web site. Multi-tiered architecture does not limit to the hardware side of the system, but also can be defined on the functional level, like multi-tiered data architecture [2].

Servers that form a multi-tiered architecture do not necessary be of the same kind (homogeneous) but also can be comprising of different (heterogeneous) systems. Having different systems in the whole architecture requires a proper way to facilitate data communication between them, making the task of bulk data loading a challenge.

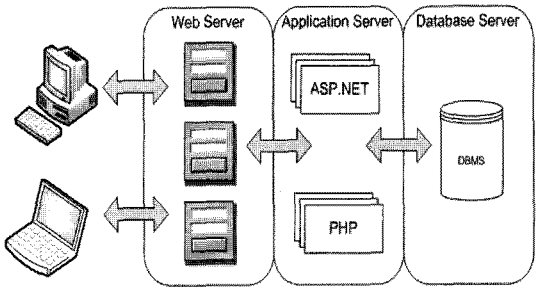

그림 1. 웹 기반의 다층구조
Fig. 1 Web based multi-tiered architecture

In this paper, we analyze and evaluate difference approaches of bulk loading from a software development perspective. The remaining of the paper is organised as follow: Section Ⅱ will discuss on the background and issue pertaining data loading, alongside with analysis of bulk loading in its simplest form. We evaluate on performance of bulk loading between different systems in Section Ⅲ together with explaining on the function of a staging database in data loading and how it can aid data loading for multi-tiered system. In Section Ⅳ, we explain on how to improve performance of bulk loading using hybrid technique through a prototype bulk data loader that we have developed, which can be easily applied in heterogeneous system environment. Experimental result using our proposed solution will be compared and evaluated before we conclude our paper and outline our future work in Section Ⅴ.

Ⅱ. Background

The task of loading large amount of data is commonly under the purview of database administrator, but as system architecture becomes more complex, software application

developer is getting more involved in data bulk loading, especially for accessing and interfacing between different systems. Applications that insert large amount of data from an input source to a targeted database in its simplest form can be summarized as follow:

1. Open input data file
2. Open database connection
3. For each record in data file
4. Insert data into target database
5. Next
6. Close database connection
7. Close input data file

Most widely available DBMS have utility function or command that supports data bulk loading. Some DBMS like MySQL for example provides a simple command LOAD DATA INFILE [3] to achieve bulk insertion by running it as part of a SQL syntax. Apart from bulk load operator found in PL/SQL, Oracle contains a utility function SQL*Loader [4] for loading data using a control file, while SQL Server bulk loading can be done using BULK INSERT syntax in Transact-SQL, bcp utility or as part of database extract, transform, and load (ETL) utility, such as those provided by SQL Server Integration Service [5].

The bulk loading function and utility require the execution to be carried out within the context of the respective database and some DBMS vendors do not made available such utility available in multiple systems that are running on different platform and architecture. External system in particular does not have access to the bulk loading function directly for security and manageability reasons.

DBMS specific bulk loading utilities are optimized for inserting large amount of data and generally performs better than a regular INSERT syntax as summarized in Table 1 from a series of incremental tests using program insertion and bulk loading utility. The test data used in the experiment are taken from IMDb Movie List [6]. The movie list file contains information about the title of the movie, the year of the movie being released and other additional information. A total of up to 1,475,891 e takes are available within the character delimited data file. Three different database systems were configured as outlined in Table 2.

A C# bulk loading application program as shown in Figure 2 using .NET Framework 3.5 library for SQL Server, MySQL Connector/Net 6.1.3 and Oracle ODP.NET 2.0 11.1.0.7.20 database drivers supplied by each database vendor was created to evaluate on the execution performance. Database bulk loading utility performs significantly faster in our experiment. Oracle SQL*Loader able to insert a single record in 0.15ms on average, which is almost six times as fast from inserting through the application program. SQL Server and MySQL bulk show on average 28 times as fast in 0.05ms/record and 10 times as fast in 0.06ms/record respectively for bulk inserting through utility and function compared to direct insertion via application.

표 1. 대용량 로딩 기능과 직접 프로그램 삽입 수행시간

Table 1. Execution time for bulk loading function and direct program insertion

Record Size	Oracle 11g		SQL Server 2008		MySQL 5.4.3	
	DB Util.	Program	DB Util.	Program	DB Util.	Program
500	0.25s	0.48s	0.06s	0.70s	0.04s	0.33s
1,000	0.37s	0.89s	0.11s	1.44s	0.18s	0.72s
5,000	0.54s	4.38s	0.19s	7.34s	0.25s	2.98s
10,000	0.84s	8.38s	0.34s	14.31s	0.42s	5.99s
50,000	2.96s	42.55s	1.31s	72.33s	1.86s	30.50s
100,000	5.47s	83.42s	2.55s	144.38s	3.62s	71.77s
500,000	24.00s	421.78s	16.66s	722.77s	18.46s	350.36s
1,000,000	51.96s	815.56s	32.69s	1464.33s	39.11s	618.09s
1,475,891	66.07s	1185.80s	46.89s	2175.92s	43.00s	860.07s
Average	0.1467ms	0.8577ms	0.0508ms	1.4465ms	0.0596ms	0.6435ms

표 2. 테스트 환경구성
Table 2. Test environment specification

Parameter	Value
Processor	Intel Pentium 4 3.20 GHz
RAM	512MB
Operating System	CentOS 5 Linux Kernel 2.6.18
DBMS Server	MySQL 5.4.3-Beta

Parameter	Value
Processor	Intel Pentium 4 2.00 GHz
RAM	1GB
Operating System	Windows XP Service Pack 3
DBMS Server	SQL Server 2008

Parameter	Value
Processor	Intel Core 2 Duo 2.00 GHz
RAM	2GB
Operating System	Windows Vista Service Pack 2
DBMS Server	Oracle 11g Release 1

Ⅲ. Heterogeneous Bulk Loading

Deploying business applications in an organization often requires interfacing with multiple type of systems for a seamless end-to-end solution. An external third party or a different internal system might need to connect and access the data and supply new data from its system, often requires data interfacing or translation. The interfacing process can be carried out using a proprietary middleware solution or DBMS specific utility to integrate two different systems together. For software development that needs implementation within the application, DBMS driver can be used either using provider specific driver or through ODBC connection. In this section, we examine on the performance of application approach to bulk loading using the same data set outlined in section Ⅱ, and explain how a staging database can help to improve overall application performance.

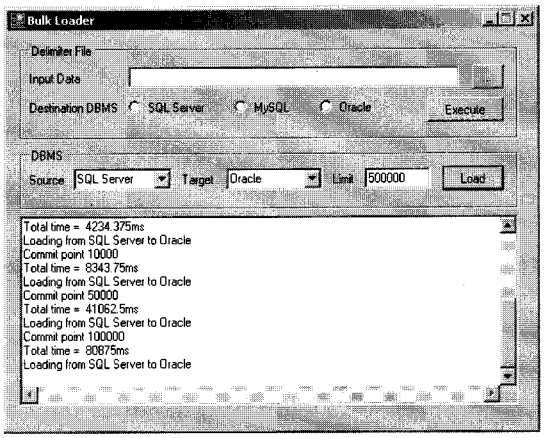

그림 2. 데이터베이스 드라이버를 사용한 대용량 로딩 프로그램
Fig. 2 Bulk loading program using database driver

3.1 Application Program Loading

Bulk loading application that we have created as shown in Figure 2 contains a feature to fetch data from a source DBMS and insert to another target DBMS using DBMS specific driver for better compatibility and performance. We call this *fetch-insert* approach where a data is fetch from a source and subsequently inserted to another target destination database. Bulk data loading was executed using transactional insertion stored procedure. Figure 3 depicts result of four out of nine more significant test dataset in line chart format using nine different combinations of DBMS mix.

표 3. Fetch-insert 방법을 사용한 레코드당 평균 수행시간
Table 3. Average execution time per record across different DBMS using fetch-insert method

From \ To	Oracle 11g	MS SQL 2008	MySQL 5.4.3
Oracle 11g	1.0065ms	1.5363ms	0.8507ms
MS SQL 2008	0.8509ms	0.5995ms	0.7979ms
MySQL 5.4.3	0.8552ms	1.4729ms	0.8467ms

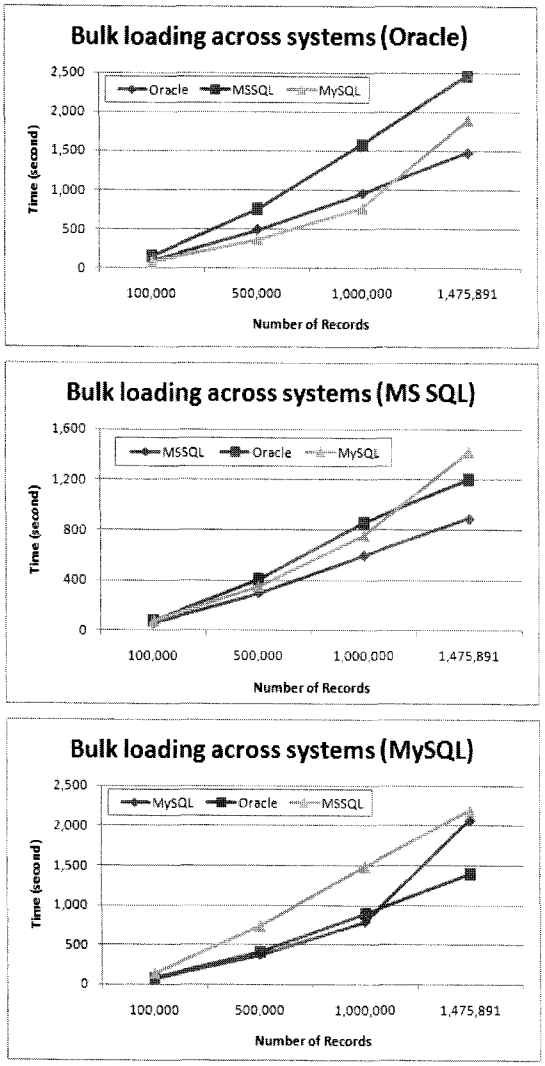

그림 3. Fetch-insert 방법을 사용한 대용량 데이터 로딩
Fig. 3 Data bulk loading using fetch-insert method across different database systems

Bulk loading to target database using SQL Server 2008 requires more time than the other two DBMS in our test. Another test of running approximately 3.1 million records to each DBMS is shown in Table 3 giving similar outcome. We have included another test for loading data to the same kind of DBMS but located on a separate system from the source system for comparison as displayed in italics in the above table. SQL Server 2008 scores best in our test for data loading using the same DBMS with about slightly more than half a millisecond to insert a record.

표 4. SQL 서버를 통한 직접삽입의 수행시간
Table 4. Execution time for direct insertion through SQL Server 2008 linked server feature

Data Size	Oracle 11g		MySQL 5.4.3		SQL Server 2008	
	Total (ms)	Avg (ms)	Total (ms)	Avg (ms)	Total (ms)	Avg (ms)
500	5,572	11.144	299	0.054	959	1.918
1,000	10,990	10.990	559	0.051	1,892	1.892
5,000	55,144	11.029	2,919	0.053	9,416	1.883
10,000	109,887	10.989	5,924	0.054	18,640	1.864
50,000	551,492	11.030	31,786	0.058	93,455	1.869
100,000	1,098,190	10.982	58,736	0.053	186,631	1.866
500,000	2,394,487	4.789	275,045	0.115	941,967	1.884
1,000,000	5,219,529	5.220	575,678	0.110	1,884,954	1.885
1,475,891	7,744,708	5.247	800,485	0.103	2,778,416	1.883

3.2 Database-to-Database Loading

Commercial DBMS like Oracle and SQL Server support linking to another DBMS for accessing remote database within a same session as the current connection. Oracle database link via heterogeneous service and SQL Server linked server allow connecting directly to a remote DBM.S, not necessary from a same kind, and performs SQL query for Data Manipulation Language (DML) and Data Definition Language (DDL) depending on the rights granted for remote user. Linked database provides a standard avenue for us to access distributed database using SQL statement from the source server without having to handle the underlying detail of interfacing with diverse data source of the destination server. This serves the need of both heterogeneous system and multi-tier architecture bulk data loading.

Performance test for linked database was carried out with result of using SQL Server 2008 linked server is shown in Table 4. Oracle shows a higher execution time

with an average of 9.05ms/record. This is much slower compared to result of fetch-insert method. MySQL 0.58ms/record fairs better with linked database method compared to fetch-insert 0.80ms/record.

3.3 Role of Staging Database in Bulk Loading

In developing software application that requires data bulk loading for heterogeneous system or multi-tiered system, deploying an intermediate staging database can be helpful not only to increase performance, but also able to mitigate compatibility and system configuration issues. This can be observed from the following scenarios:

① *Data Type Compatibility*

DBMS with special data type or different data type size, for example *image* data type of SQL Server, and different maximum value for data type *varchar*. (MySQL supports up to 65,535 [3], SQL Server 2008 supports up to 8,000 or 2^{31} - 1 bytes [5]), the sending end might not have compatible data type. By staging the input data, we can perform a conversion or invoking function that the intermediate system can support to achieve the data loading.

② *Security & Licensing Constraint*

Some target DBMS has licensing or security issue that do not permit more connections or direct connections to the DBMS. Staging data in the middle between source and target allows isolation of target database from direct interfacing with source. Number of connections to target database can also be controlled through staging database while still allowing connection from source to staging database.

③ *Session Timeout*

Execution of database query and command might require a certain amount of processing time. If the waiting period is longer than the configured timeout session, the process will be timed out before completion. This happens especialpenfor web request where response need to be prompt so that the client session will not timed out. Using a staging database, a response or an acknowledgement out. return while the staging database process with the actual processing.

④ *Derived Data. Lookup and Mapping*

Insertion of data at target database might require value derived from existing table or a lookup translation. Reference code used at the source for example, can be different from those used in target table. Staging input data can allow us to perform a mapping operation prior to final insertion into target database and at the same time filter off unnecessary data for target database.

⑤ *Staging as Flow Control*

In certain implementation of DBMS a subroutine using script or a small program can be invoked to carry out preprocessing of data. Loading high volume of data in a single transaction might overwhelm to subroutine causing error and inconsistent data. Using staging database, we can control the insertion to the target time in a orderly manner, while still permitting source data from coming in.

Ⅳ. Proposed Solution and Analysis

Although insertion using fetch-insert method in application is simple in implementation and works fairly well for small input data, it can be generally slow for large input data and does not scale well for deployment that relies heavily on data insertion such as application program that does content aggregation. Instead of fetching from one source and inserting immediately to the target, we propose using a staging database as an intermediary to leverage additional advantages of staging database as explain in previous section.

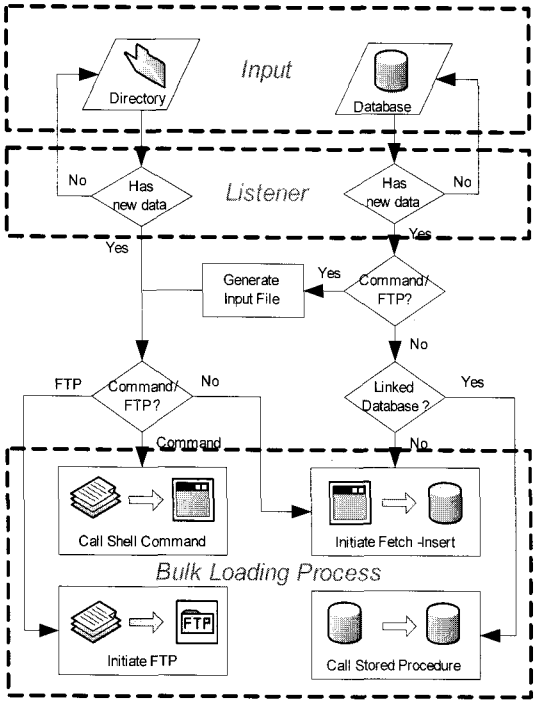

그림 4. 제안된 방법의 개념도
Fig 4. Conceptual diagram of proposed solution

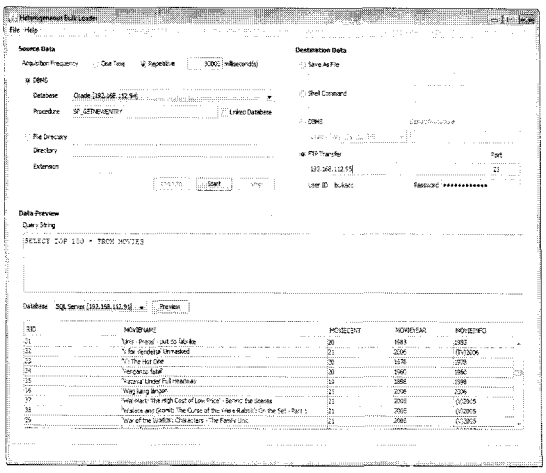

그림 5. 제안된 방법의 Java 프로토타입 구현
Fig. 5 Prototype Java implementation of proposed solution

4.1 Design and Implementation

Figure 4 shows the high level concept of our proposed solution together with a Java prototype implementation of bulk data loader for heterogeneous and multi-tiered system shown in Figure 5. Using our proposed solution, an instance of the program is executed on the source and the intermediate staging system. The data output from one end of the program to input to another as shown in Figure 6. This creates a continuous end-to-end data flow from source to target destination via intermediate staging system.

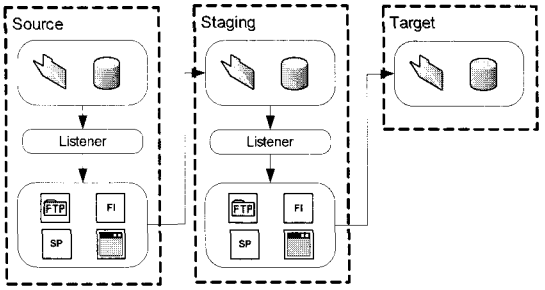

그림 6. 각 경우의 입력에 대한 응용결과
Fig. 6 Application output as input to another instance

Input data for bulk loading can be either coming from a file or result queried from a database. We have included a listening process into our proposed solution to allow continuous processing of new incoming data using a defined interval. For listener to process only newly arrived data, a status flag or information about processed data in kept to ensure no duplication of records exist. In addition to fetch-insert method to load data through application, we proposed a combine or hybrid of linked database, command execution for calling DBMS bulk loading utility and FTP transfer of data.

For scenario where linked database is supported, we can invoke a stored procedure to facilitate bulk loading from database to database, without involving application fetch and insert. From our analysis shown in previous section, database linked method is efficient for loading data from

표 5. 여러단계에서 제안된 방법의 수행시간 비교
Table 5. Source to target database via staging database using proposed hybrid solution

Record Size	Fetch-Insert Approach (ms)				Proposed Solution (ms)				Difference (ms)	
	Source-Staging	Staging-Target	Total	Average	Source-Staging	Staging-Target	Total	Average	Total	Average
500	1,158	750	1,908	3.816	106	406	512	1.024	1,396	2.792
1,00	2,129	1,032	3,161	3.161	217	750	967	0.967	2,194	2.194
5,000	10,190	3,500	13,690	2.738	490	3,219	3,709	0.742	9,981	1.996
10,000	20,420	6,797	27,217	2.722	698	6,437	7,135	0.714	20,082	2.008
50,000	100,960	33,640	134,600	2.692	3,042	30,563	33,605	0.672	100,995	2.020
100,000	202,118	67,750	269,868	2.699	6,018	60,422	66,440	0.664	203,428	2.034
500,000	1,019,200	344,672	1,363,872	2.728	29,362	300,875	330,237	0.660	1,033,635	2.067
1,000,000	2,094,867	703,438	2,798,305	2.798	65,715	603,422	669,137	0.669	2,129,168	2.129
1,475,891	3,124,606	1,204,250	4,328,856	2.933	99,907	914,953	1,014,860	0.688	3,313,996	2.245

SQL Server to MySQL.

The optimized DBMS bulk loading utility has significant advantage over other bulk loading method. We have included in our proposed solution to allow user to invoke bulk loading utility through shell command execution. For input data coming from database, an additional process of generating input file is included.

DBMS bulk loading utility can be used hand in hand with FTP data transfer where DBMS utility is not supported on a certain platform. Input file can be uploaded to an intermediate system that can execute the utility program. For cases where bulk loading utility that cannot support local file in remote system, such as SQL Server bulk loading function does not allowed non-UNC remote file, FTP allows us to transfer the input file to a staging location where the utility can support the input file. Using FTP also allow us to leverage on the benefits of transferring large file such as file resume.

4.2 Experimental Simulation and Result

We have conducted an experiment to evaluate on the performance of bulk loading using our prototype implementation in Java. We compared our result against fetch-insert method based on the test environment outlined in Table 2. Source DBMS is running Oracle 11g, staging DBMS is running SQL Server 2008 and our target DBMS is running MySQL5.4.3 Beta. Staging system is installed with Filezilla FTP Server 0.9.30 Beta.

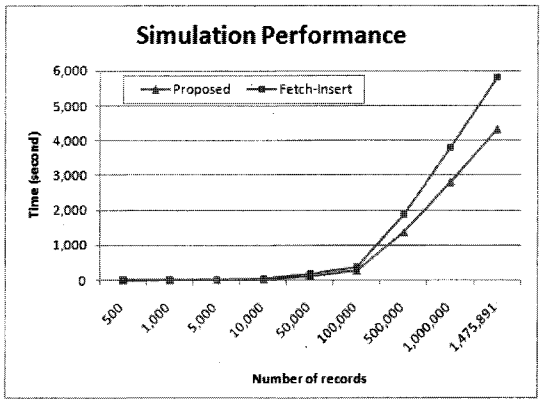

그림 7. fetch-insert 방법과 제안된 방법의 수행시간 비교
Fig. 7 Comparison of execution time for proposed solution against fetch-insert method

The same data set used in previous sections was selected as the data to pre-load in our Oracle source database. Using our prototype program in source computer, we have selected input from database and the result is transferred over to staging system using FTP. As for staging computer, file directory is selected as the source for input data listening and the data is loaded into target database using linked database.

Results of the experimental simulation is shown in Table 5 with a summary in line chart shown in Figure 7. Using hybrid method in bulk loading data from source to destination, we are able to achieve a loading rate of 0.756ms/record, compared to 2.921ms/record using fetch-insert method.

V. Conclusion & Future Work

In this paper, we have discussed issue on bulk data loading in a multi-tiered architecture environment using heterogeneous database systems. We have analyzed on the performance of multiple se syusing different bulk loading method including application loading, se sybulk loading utility, linked database loading, and fetch-insert loading. Also, we have seen how data staging can be useful to increase performance ogenevercome some issue ofybulk loading, especiiply for heterogeneous or multi-tiered system. We have proposed a hybrid solution that leverage on staging database, which has shown significant performance increment in bulk loading in a multi-tiered environment.

More recently, XML formatted file is getting more common as the standard for modeling data for heterogeneous system for its platform neutrality and extensibility, it has garnered a lot of attention in research circle. As technology that is based on XML is increasing in popularity, such as XML web service and a steady stream of DBMS that provides native support for XML, we have planned our future work to focus on XML as the choice for data bulk loading for heterogeneous system.

References

[1] B.M. Subraya, "Integrated Approach to Web Performance Testing: A Practitioner's Guide," IRM Press, 2006.

[2] G. Lord, "The Importance of Data Architecture in a Client/Server Environment", in *Designing a Total Data Solution: Technology, Implementation, and Deployment*, R. E. Burkey and C. V. Breakfield, e.d., pp. 69-80, CRC Press, 2001.

[3] MySQL 5.4 Reference Manual, Sun Microsystems, 2009. http://dev.mysql.com/doc/ refman/5.4/en/index.html

[4] K. Rich et. al, "Part II : SQL*Loader," in *Oracle Database Utilities, 11g Release 1 (11.1)*, Oracle, Sept. 2007.

[5] SQL Server 2008 Books Online, Microsoft, October 2009. http://msdn.microsoft.com/en-us/library/ms130214.aspx

[6] The Internet Movie Database (IMDb). Retrieved 24 September 2009. Available from : http://www.imdb.com/interfaces

저자소개

진기원 (Tan Hee Yuan)

2000년 말레이시아 멀티미디어 대학교 졸업 (학사)
2008년 ~ 현재 동서대학교 디자인&IT전문대학원 유비쿼터스 IT학과 석사과정

2004년~2008년 말레이시아 근무
※ 관심분야 : 웹서비스, 데이터 처리, 모바일 애플리케이션

임효택 (Hyotaek Lim)

1988년 홍익대학교 전자계산학과 졸업 (이학사)
1992년 포항공과대학원 전자계산학과 졸업 (공학석사)
1997년 연세대학교 컴퓨터과학과 졸업 (공학박사)
1988년~1994년 한국전자통신연구소 연구원
2000년~2002년 Univ. of Minnesota(미) 컴퓨터공학과 연구교수
1994년~현재 동서대학교 컴퓨터공학과 교수
※관심분야 : 컴퓨터네트워크, 프로토콜공학, 스토리지 네트워킹, IPv6, 모바일 애플리케이션