# Towards Heterogeneous Grid Database Replication

*Kemian Dang*

Master of Science
Computer Science
School of Informatics
University of Edinburgh
2008

# Abstract

Heterogeneous database replication in the Grid context is a very important and highly demanding issue, but little work has been done on this field. Most of the existing Grid replication tools merely deal with the read-only files, on the other hand, conventional database replication software is mainly designed for the homogeneous computing network environment. To address such issues, a Grid Database Replication Model (GDRM) has been developed in [1], which combined the conventional database replication mechanisms with the Grid replication mechanism, and IBM DB2 SQL-Replication has been successfully integrated into an implementation of this model.

To extend Chens work, we integrated the Oracles replication into the GDRM. We used the latest OGSA-DAI 3.0 to provide Grid facilities, and Oracles Materialized View replication mechanism to provide the flexible database replication.

To solve the cross-domain problem, we proposed an intermediate table solution. We use an intermediate site constructed by OGSA-DAI which could access both master and replica site, to propagate the caught changes from master to the replica.

Furthermore, we have tried to extract the redo log from the Oracle database and translate it into standard SQL language, which can be applied to other database systems later.

By these methods, we are able to provide a grid service to replicate the data objects in a heterogeneous, highly distributed computational Grid environment.

# Acknowledgements

I would like to thank my supervisor **Dr. Dave Berry** for his constant guidance during the course of the project, and **Yin Chen**, for her great help and technical support.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Kemian Dang*)

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Introduction

Replication is important for database systems as it can improve system availability, data consolidation and data distribution. Many scientific applications often need to process large amount of data which is stored in database management systems(DBMS). To store and process large amount of data, they may choose to use the Grid technologies, which can provide enormous CPU power and huge storage space by sharing a variety of computational devices from different organisations.

However, a Grid networking environment is often large in scale, heterogeneous and highly distributed. Existing Grid replication tools merely deal with the read-only files, and most of the relational database replication technologies are not able to cope with the Grid environment[1]. With the demands continue increasing, there is a common need for a Grid database replication mechanism.

To provide the Grid database replication facilities, two challenges arise: one is to replicate data in the Grid environment, and another is to copy among heterogeneous database systems.

Although there are some existing models for replication, they mainly focus on a single aspect of the problem:

- Replicating read-only files in the Grid system.
  These systems support large files copying among the heterogeneous operation systems. However, they were mainly design for managing read-only files and do not deal with database replication.

  Globus Data Replication [4], EGEE Database Management System [5], Lightweight

Data Replicator [6], Storage Resource Broker [7] are such systems.

- Vendor-specific database replication.
  Though DBMS producers have implemented replication for their own systems, usually they cannot replicate to virtual organization in a Grid environment or cross-domain environment.

  For example, MySql [8], PostgreSQl [9] DB2 [10] [11] and SQL Server[12] can not provide a cross-domain grid solutions on replication.

To cope with the shortcoming of above works, a Grid Database Replication Model (GDRM) has been developed in [1], which combined the conventional database replication mechanisms with the Grid replication mechanism, and IBM DB2 SQL-Replication has been successfully integrated into an implementation of this model. This dissertation follows Chen's work and tries to extend it.

The following is the description of components providing Grid facilities and Replication facilities:

**Grid facilities:** We used OGSA-DAI/Globus Toolkit to provide Grid services for database replication.

OGSA-DAI is a data access and integration tool built on Globus Toolkit. It provides Grid services like security, authentication, authorization, and database control services through JDBC drivers. It uses Java(which is OS independent) as the development language and web-services as the OS independent communication method.

**Replication facilities:** We are reusing existing DBMS replication mechanisms when replicating data between homogeneous database systems.

A database company has the full knowledge of its product, and their way of controlling the database is usually the most efficient one. Although their system do not have Grid support, almost every DBMS has a mature mechanism of doing replication in a common internet environment. Therefore, re-using this powerful replication mechanism will maintain high efficiency while keeping system complexity low.

Some DBMSs like Oracle, Sybase and IBM DB2 offer the functionality to replicate themselves to other databases in a common network environment. A Grid

interface can be developed to control these mechanisms, to control replication among homogeneous databases in a Grid environment.

For those DBMSs which need replication but do not provide a proper mechanism, we propose to extract redo logs, which capture the database changes in the form of the SQL commands. These SQL commands can be converted into the standard SQL language. Thereafter, these SQL commands can be passed to the replication target database, and the updates of the source database thus can be applied to the replica.

As there already exists one DBMS in the implementation of GDRM, we can show that the GDRM can adopt different databases by adding another: Oracle. This will provide us with an opportunity to study how we can control the replication of two different DBMSs in the existing model.

After implementing the Oracle replication, we carried out experiments to see whether the replication through Grid would degrade the performance of direct replication. This experiment is designed to calculate two important steps of the replication. One is the creation of initial replica and the other is synchronization.

Before doing this we first collect the pure OGSA-DAI invoking time, which indicates an average expected performance. For creation, we compare the time for creating through OGSA-DAI and using the create method directly in the Oracle database prompt. The expected result is that the time difference is similar to empty invoking time. For synchronization, we test the synchronization after DELETE command. We compare running delete command directly in the Oracle database prompt, running synchronization on the materialized view in Oracle database prompt and invoking synchronization through OGSA-DAI. The expected result is that first two times are similar and differences between the last two experiments are similar to the empty invoking time.

## 1.2 Paper structure

Chapter 2 describes the background about Grid, replication and related work; Chapter 3 shows the architectural design of our work; Chapter 4 gives the details of implementation; Chapter 5 describes the result of experiments which test the performance of our implementation; and the last chapter gives a conclusion of our work and what could be done in the future. The appendices contain further information on how to configure

OGSA-DAI with Oracle, and how to create the implementation environment.

# Chapter 2

# Background

In this chapter, two fundamental background concepts about Grid and replication will be introduced. After introducing these concepts we discuss two related works, one of which is what we are trying to follow and extend.

## 2.1 Fundamental Concepts

### 2.1.1 Grid

The requirement on large computing resources continue to increase. For example, the Large Hadron Collider(LHC) in CERN will produce 1 Peta-Byte of data every year, which must be stored and analyzed in a reasonable period. It is hard for any single organization to meet this needs of such a vast amount of computing power, storage space and human power. Meanwhile with the continue development of the internet, people try to cope with this difficulty by sharing computer resources among different organizations, and usually using these resources to achieve a similar goal.

Although there is no single accepted definition of Grid, usually it is understood to mean a computing resources sharing network. Inside the network, a group of people, who work for different organisations but collaborate on a given project are taken as virtual organizations, and the network itself is a virtual group. This group has a similar target, members authorize each other to use their resources so that all the CPU time could be used better efficiency.

In the data resource view, Grid is decentralized, because all the participants provide their resource for the whole Grid. It is also geographically distributed, because the participants are typically locate in different parts of the world. Its scale is very large as

many real organizations may join one virtual group. It is dynamic, as participants can join or quit the virtual group as they wish. Heterogeneity is an integral characteristic, as the users always have different computing environments and net bandwidth. And lastly, it is mission-oriented. Without a similar goal, different organizations would lack the motivation to connect together and share their resources with others.

Figure 2.1.1 shows a Grid used by several virtual organizations.

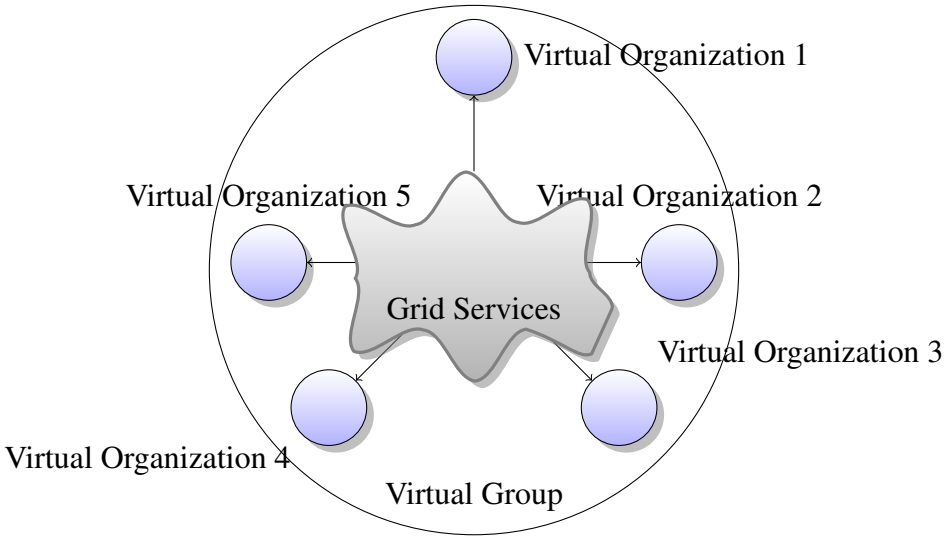



Figure 2.1: A Virtual Group view of Grid

But Grid is not a simple connection of computers over network, it has its specific management policies. The sharing of resources means a very important consideration is security. Members require a framework in which they may provide their resources and share them with others in a secure way. So, between user software and base operating systems, a middle-ware is needed to handle the aspects of authentication, authorization and security, as well as connecting each other freely. The Globus Toolkit [13] is one such kind of middle-ware, and it has been widely used.

### 2.1.2 Replication

Replication is a way to distribute data resources into remote systems and keep them synchronized with the master site on a particular level. Distributed data is usually processed by these remote sites, and sometimes the changes need to be propagated back to the origin site.

To make this idea easier to understand, replication can be taken as similar to data

backup, but it is different in its aim and constraints.

**Aim:** Backup has a single goal which is increasing data safety. By backing up another proper copy of data, the probability of losing that data will reduce by 50%. Although the backup is updated at specific times, to reduce complexity, a typical network backup is not designed to be used in the remote site, which means it is not read or written at the remote site. It only keeps a valid copy, and can be restored in the case that data in the master site is lost.

On the contrast, replication is designed to increase data availability, data consolation and data distribution. The data in a replica is designed to be accessed and even processed in its local site. By allowing manipulation of data at the remote site, replication gives a way of sharing computing storage and power resources in the Grid.

**Constraints:** Backup's constraints depends on user requirements. Usually it is only a time constraint on the time gap between latest the backup and the current data. It needs the time-stamp of the backed-up data being in a specified range. A hardware RAID system could provide fully synchronized backup, while some network backup is executed daily or monthly.

Replication's constraints also depend on user requirements, but the constraints are more complex, because there are usually many replica sites, and every site should follow certain synchronization rules with respect to the master site. If the replica is designed to be updatable, the change on the replica needs to be propagated back to the master. Further more, there can be a replication group, in which every member replicates itself to the group and the group coordinates every member to be synchronized. This is quite different from the backup scenario.

## 2.2 Related Work

This field is quite new and little work has been done before. There are two important papers, from the National e-Science Center, Edinburgh and CERN, Geneva respectively.

### 2.2.1 Grid Database Replication Model

In [1], Chen, Berry and Dantressangle have developed the Grid Database Replication Model (GDRM) to do Grid database replication. Its work flow is expressed in Figure 2.2.

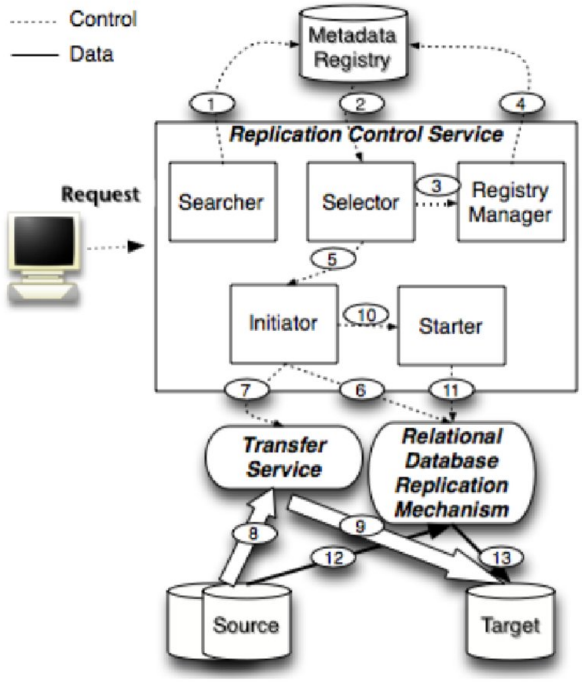



Figure 2.2: The work flow of the database replication of GDRM [1]

There are 4 Grid services in this figure:

1. Meta Data Registry:

   A service stores meta information, provides resource discovery;

2. Transfer service:

   A service uses a grid transfer tool like GridFTP to create a new replica database;

3. Relational Database Replication Mechanism:

   A service which controls the database to do replication through Grid – this is what we are going to focus on and try to extend;

4. Replication Control Service:

   A service that coordinates the above three services to complete the whole procedure.

This model does not indicate the detailed replication method but used abstract classes, and left the specific database method to be implemented. So it is compatible with most modern relational database management systems and hides the complexity of doing replication below the relational database replication mechanism service.

In GDRM, there are 5 abstract classes:

- *ConnectDB:*

  Creating the link from local to the remote database.

- *Configure:*

  Configuring the replication detail like replica name, start time, synchronization time and so on.

- *Create initial replica:*

  Creating the initial replica, which will be synchronized with the source later.

- *Synchronization:*

  Synchronizing replica with source under configuration rules.

- *Monit or:*

  Checking the information of the replication.

There is already a prototype implementation which replicates IBM DB2 database. It is implemented on top of OGSA-DAI middle-ware version 2.2 using Globus Toolkit 4.0.3 and JDK1.4.

Our work will continue and extend Chen's, focusing on the "Replication Database Replication Mechanism" and "Replication Control Service" by connecting to an Oracle database in the model and exploring how to extend the method for heterogeneous database replication.

### 2.2.2 Relaxed Consistency Replication

This work[2] mainly discusses the consistency aspect of data replication, which we also need to take account of in our work. It introduces a replication model following a relaxed consistency(asynchronous) style of replication. It points out some important aspects in the replication:

- The logical consistency can be met if $\sigma_{d,i}(r_i) = \sigma_{d,j}(r_j)$. In which $\sigma_{d,i}$ is the $i-th$ replica of dataset $d$.
- Physical constraints like data size, distance between sites and communication failure should be considered.
- Using SQL commands instead of data file transfer to achieve heterogeneous replication: obtaining SQL command from the source, using a translator component to change it to the target SQL commands.
- The paper takes fault tolerance as an important aspect. The module receiving user commands and managing replicas is also used to take care of fault tolerance.

Like the GDRM, this work has defined services for replication control and transfer. It uses a two-level replication with a replication catalogue to control replication. Global services are used to coordinate all local replication services, and local services are used to connect the replication source and client. A Log Watcher in the local service is used to capture the changes on the master table and apply the other sites' changes to itself.

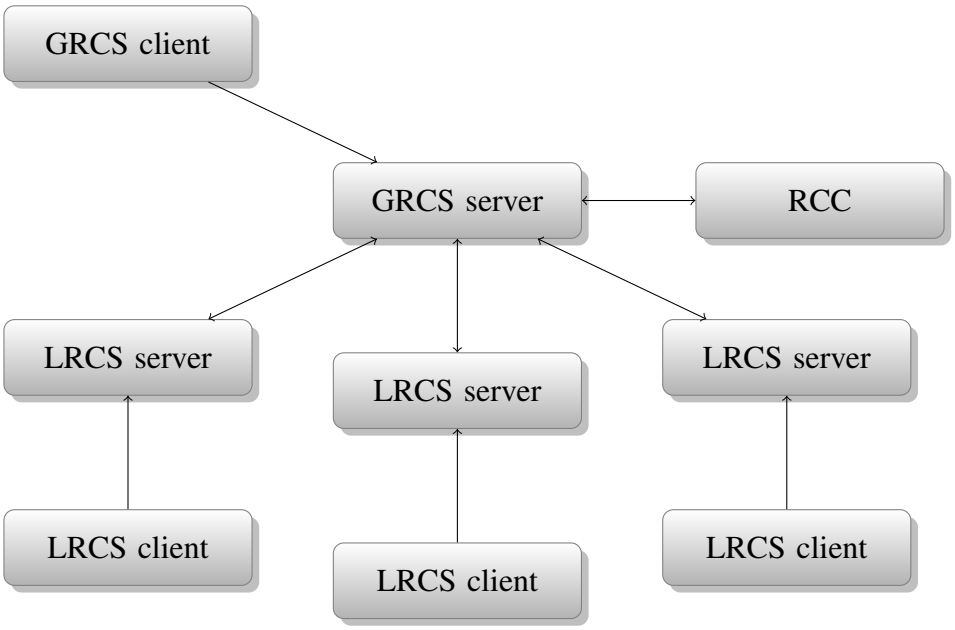



Figure 2.3: Architecture of consistency service from CERN[2].

Figure 2.2.2 shows the architecture of this work. GRCS means Global Replication Consistency Service, LRCS means Local Replication Consistency Service and RCC means Replication Consistency Catalogue.

One possible shortcoming of this work is that all its operations are implemented in its own scope and do not work on any Grid middle-ware, which means there will be a lot of portability work before any practical use. Our work avoids this problem by using OGSA-DAI Grid middle-ware.

# Chapter 3

# Design

In this chapter, we first introduce the concepts and tools we use including incremental serialized consistency, the materialized view of Oracle and OGSA-DAI. After that, we describe our design solution for replicating a homogeneous database through Grid, cross-domain replication and replication of heterogeneous database.

## 3.1 Concepts and Tools

### 3.1.1 Incremental serialized consistency in Grid replication

Consistency means that every part of the system is kept at a certain level of synchronization, with the differences between remote sites are in a defined range.

The measure of the synchronization is denied as the size of differences between all the replicas. But the actual differences cannot be captured precisely in a short time and with limited computing, available during the replication. That is because identifying the difference between two sites is very expensive in terms of network and system load. Site A would need to send all its content to another, which would cost a lot of network resources; and site B would calculate all the differences, which would cost a lot in CPU load. Additional computing on the origin site could reduce some network cost, but on the other hand it will increase CPU load.

As an alternative to measure differences, one solution is to use “time” to indicate difference. The difference range is defined by the period between the last synchronization with the master site, and the current time. For example, if one replica was synchronized every 1 hour, we say the difference between the replica and master is at most 1 hour.

One extreme consistency of replication is full synchronization. This means every change would be synchronized at no time. By "no time," I mean the master and all the replicas are locked to receive synchronization updates until the last one finishes. Replicas are synchronized whenever there is a change, or some changes in the master site. The capture mechanism in the master site is triggered whenever there is a change, and replicates this change to the replica. For example, a database could be configured to synchronize the replica whenever there is a commit in the source database.

But besides being complicated to implemented, full synchronization will also consume a large amount of system resources on locks, and increase the probability of dead-lock.

On the other hand, too loose a policy would make replication useless – if the difference is too big, all the results from the replica will not be trustable.

So how to choose a appropriate level of replication is very important. Two criteria for choosing a level are: what the user/system requirements are and how much resource can be used for the replication. Of these two aspects, the former is more important, because failure to meet the requirement means the system is useless. If there is a very high requirement and the system resource is not enough, we then have to add resources to use some other method to further optimize the resource.

Besides the time consistency level requirements, different environments replication will have different policies. We can design different policies according to the points below:

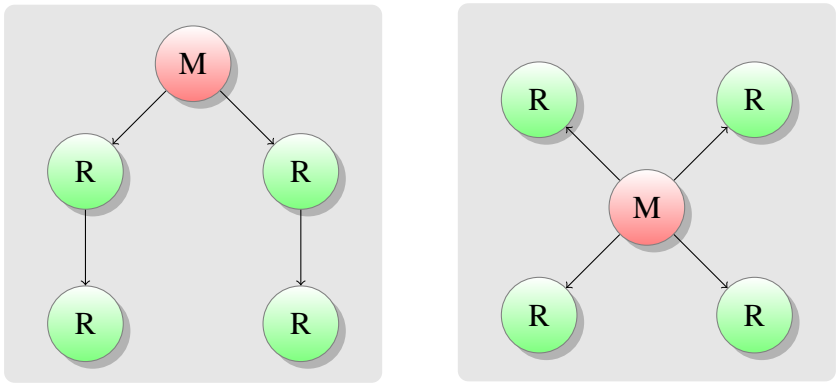
- Whether the replication system has multiple tiers.

  A scenario in which a site is a slave replica to a higher tier and in the same time a master to lower tier is multi-tiered replication environment. The master is the site which produces the data and usually changes the data; a slave is the site which will be synchronized with the master. Figure 3.1 shows the comparison between the architecture of multiple and single tiers.

  Although multi-tier is a way to reduce network costs, it can also make the whole procedure of replication use more times compared to a single tier architecture. Whether to choose multiple-tier depends on the number of replicas and network performance.

- Whether replication system has multiple masters.

  The scenario in which data is updated in multiple sites is a multiple masters




Multiple Tiers Replication Single Tier Replication

Figure 3.1: Comparison of multi-tiered system and single-tiered system

environment. These sites could replicate to a separate replica or replicate to each other.

This means there may be conflict when two master sites are updated differently in two synchronizations. One way of solving this conflict is to merge/delete according to the time-stamp of the change occurring in each site. To get a time-stamp in a Grid system, sites need establish a global order of all the events in the system. This depends on a logical clock, which is negotiated among all the system participants.

The multiple-masters scenario costs a lot of CPU time on solving conflicts and incurs a high communication cost on keeping logical order.

In the simplest scenario, we have $n$ replicas and a single master in a single tier environment. We can see that the time cost of a full replication is:

$T_{total} = T_m + max(\frac{D_r}{B_r} + T_r)$

In this equation $T_{total}$ is the total time, $T_m$ is the computing time for the master to produce a replica, $D_r$ is the replication data for one replica, $B_r$ is the bandwidth of one replica, and $T_r$ is the applying and responding time of one replica. As we have replicas replicating concurrently, so the total time of one successful replication includes the master's processing time and the time needed to get a response from slowest replica.

In a strict consistency system, all the sites of the system need to be locked throughout the whole procedure. If the data or the communication time is very large or the computing resources are not enough, the lock procedure will be so long that there will be a long queue to process while the system is nearly constantly locking.

Addressing such a problem, we proposed an Incremental Sequential Consistency policy for Grid database replication.

This policy is based on the fact that: 1, most Grid database replication scenarios do not need a strict consistency policy; 2, data change is less frequent than a business database; 3, the data is very large and 4, the data processing model can be partitioning a big request into pieces, which means every site keeping data synchronized with master and doing its own work.

In such a scenario, a large datasets costs too much to be distributed to replicas for every change. A better way is to use *incremental database replication*. A database is a structured system controlled by SQL commands. This characteristic suggests that the change to the data could be captured by processing the structured change of the data – which is expressed by SQL. This is easier than a non-structured system as it does not need to parse all the data – the structured information could indicate which part is changed.

There are already SQL standards that every database vendor follows. So a standard SQL command can run on most Relational Database Management Systems (RDBMSs). Thus a change could be expressed as a standard SQL, captured by the master and then applied to the replica. This solution reduces the load on the network. After the initial replication copy, the replica site does not need to receive a full copy of data anymore. It receives SQL commands such as UPDATE and DELETE that details the changes , and focuses on the data that needs to change. Of course INSERT is still carrying data.

*Sequential* is a causal order policy which keeps the order of every change only inside replica site. That is:

If in the master site $D_i$ happens before $D_j$, then in every replica $D_i$ is applied before $D_j$.

This order does not require all the replicas to be locked and then wait for the slowest one to finish, but only to lock the one which is doing the replication. It keeps every replica consistent after they have applied the same change; while every replica may be different in the same physical time.

A time-stamp of the change from the master will be recorded in the replica to indicate its current state. A catalog service could also be used to record the time-stamp and to provide a query service. It would need to be locked when doing replication.

In such a manner, the order of all the changes in one replica site follow the production order of the master site.

### 3.1.2 Re-using RDBMS replication mechanism

One way to reduce replication complexity is by *re-using* the replication mechanism from the database vendor.

Nearly every RDBMS has its own way of doing replication, they usually capture the changes and apply them to the remote database. Though it cannot be used in the Grid environment, it is ready for use in a local network environment. We want to re-use these mechanisms in the Grid environment by using Grid middle-ware to control and process it. Before processing by OGSA-DAI and integrating into GDRM, we firstly examine the replication mechanism.

We focus on the Oracle relational database, and its materialized view replication mechanism.

Unlike a normal view which does not store any data but only an expression in the database, a materialized view is a physical storage of data chosen from the master tables according to the expression. This means data in the materialized view can be accessed directly, without retrieving data from the master table first. The materialized view is actually a table, but has a constraint with a master table, which could be in a remote site or the local site.

Materialized view is taken as an advanced replication[3] mechanism in Oracle and it is very flexible: the replications data's synchronization type, period, data row/column and many other aspects could be configured to meet the user's needs. So we choose the materialized view as our replication mechanism when doing Oracle to Oracle replication in the Grid environment.

The characteristics of the materialized view include:

- Besides read-only, materialized view in Oracle can support the updatable model.

  Read-only and updatable materialized view replications are expressed in figure 3.2 and 3.3, respectively.

- It does not need a dedicated link from replica to master.

  The replica can be configured as an on-demand refresh model, thus it does not need a dedicated link to the master, or even a persistent link to the master. Whenever the replica is ready to receive data, it can get online and refresh its data as needed.

- It supports a replicating a subset of the table by indicating a proper SQL command, which could be more efficient and enhance security.

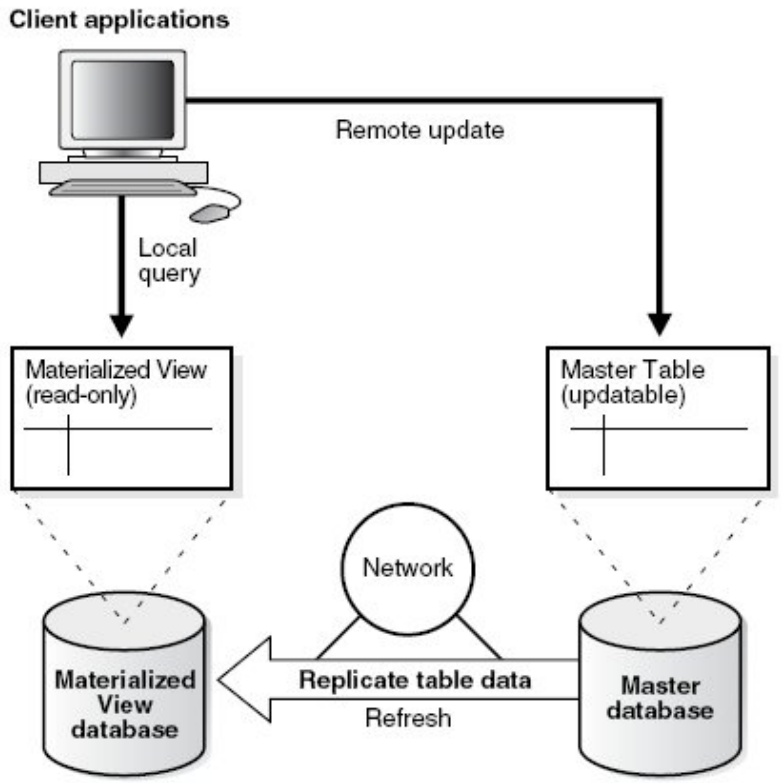



Figure 3.2: Read-only replication of Materialized View [3]

The data of the materialized view is selected from the master table using SQL commands. For example:

CREATE MATERIALIZED VIEW mv as SELECT name, UID from student where DOB $<$ '1983'

would create a materialized view which only contains the name and UID column and only the rows that date of birth is before 1983. This could be used to hide some data even if they are in the master table.

- The materialized view itself can also be a master site, thus the materialized view could be used to construct a multiple tier replication model.

### 3.1.3 OGSA-DAI

OGSA-DAI is the abbreviation of Open Grid Services Architecture Data Access and Integration. As its name suggests, it is a data access and integration middleware for

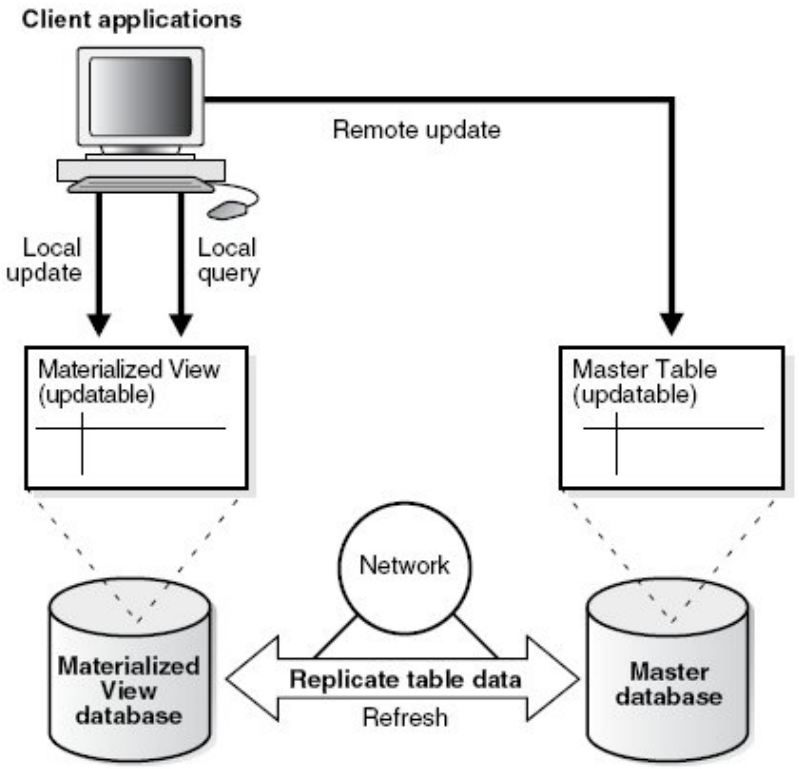



Figure 3.3: Updatable replication of Materialized View [3]

Grid services using the Open Grid Services Architecture. We are using version 3.0 of OGSA-DAI, which is the latest.

The characteristics of OGSA-DAI include[14]:

1. It provides a controlled exposure of physical data resources to the Grid.

2. It provides access to heterogeneous data resources in a unified way.

3. It uses security, management and accounting services from the Grid infrastructure.

The heterogeneous physical resources usually have different inherent structures, associated content, capability meta-data, properties and access mechanisms. All of these make the exposure and access of the data more difficult compared to the homogeneous resources. Here OGSA-DAI creates views of the data from different kinds of sources, it receives compatible requests, and manages the queries to the corresponding sites and gets data required for the answer.

OGSA-DAI is written in Java and could be deployed to systems that can support a Java virtual machine. It can manipulate relational databases, XML databases and file systems. It uses web-services as its communication method, thus can easily integrated with different network environments.

It does not implement any specific data manipulation method; instead it uses existing methods to do it. For example, it has a class which can invoke GridFTP to transfer large data through a Grid environment and it uses JDBC for relational database control and eXist to control XML databases.

OGSA-DAI can use Globus Toolkit or Tomcat as its container, to provide remote access. We are using Tomcat as the container. Each remote command goes through Tomcat to access OGSA-DAI and invokes the Java class program, runs the activities and finally returns the results through Tomcat to the client's program, which then transfers them in the format requested by the user.

Here "activity" means a specific function or program that can be invoked remotely through OGSA-DAI to achieve a specific goal. For example, a SQLQuery activity runs a SQL query to a specific database and submitting the result to the client.

Server activity needs to be invoked by the client activity toolkit, which is also a Java program, but inherits from base class which has already implement communication services, to reduce complexity for the user. The OGSA-DAI team has already developed toolkits for many useful server activities. Users only needs to write a client program, using the toolkit to pass arguments to the server, and do some translation, such as from a SQL dataset to string rows. This work flow is shown in figure 3.4.

Data resource is another important concept for OGSA-DAI. Every data resource has a resource ID, which indicates the informations that needed to connect to this resource, and a list of activities that are allowed to be used on this resource. By using different Resource IDs, there can be different activities doing similar things but with different privileges for different purposes in one resource, which provides flexibility and security.

OGSA-DAI also provides means to add new activity as a developer needs. In this way, OGSA-DAI itself can be extended to have more functions in the framework. We have made our replication implementation as an activity in the OGSA-DAI framework and have further developed the client toolkit.

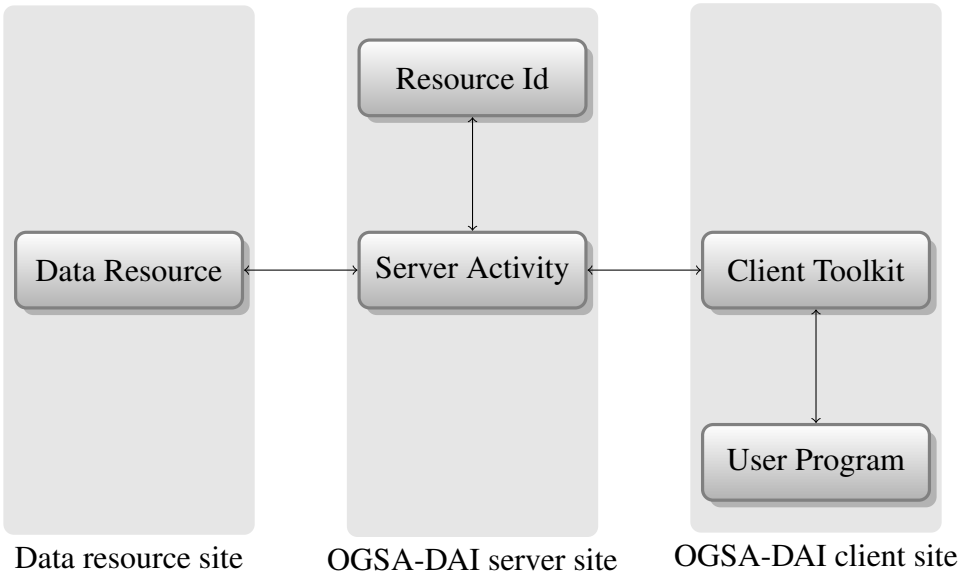



Figure 3.4: The architecture of an OGSA-DAI activity.

## 3.2 Architecture of the Work

### 3.2.1 Replicating Oracle databases in the Grid environment

As described before, we are using OGSA-DAI as Grid middle-ware and reusing Oracle's materialized view mechanism to do the replication.

Chen has already proposed this model in her work, and has implemented IBM DB2 replication. She proposed five common actions that every replication needs[1] - (1) connecting to remote database, (2) configuring replication, (3) transferring initial replica, (4) synchronizing replication system and (5) monitoring the replication process. In order to be compatible with more databases, we need to design all the activities in an abstract manner such that these common actions can be used by most relational databases. A new concrete activity implementation is needed for every new database to reflect the characteristics of the replication.

A database replication between Oracle through Grid by OGSA-DAI can be seen in figure 3.5.

We need to implement replication program for Oracle database as this figure shows.

Before using this program user needs to know two things:

In our project, we use Tomcat as an OGSA-DAI container. So, the user needs to know the URL of the OGSA-DAI server first, then to know the services it provides before accessing any data. There are built-in services ready for use. As we are going to

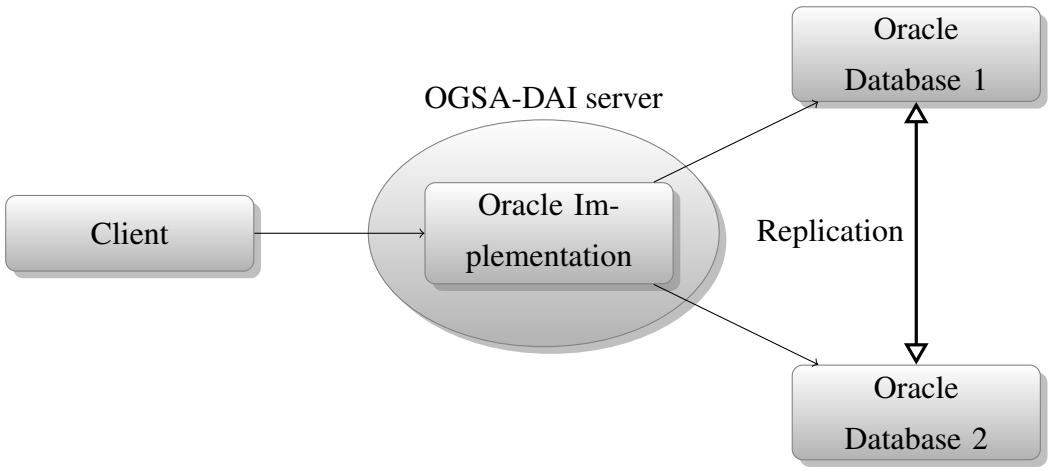



Figure 3.5: The architecture of Oracle replication through OGSA-DAI

manipulate relational database, we use Data Request Execution Service. It acts as an executor: receiving the pipelined activity, executing every object in it and returning the result. Using a pipeline can bind several activities together and provide the advantage of using output from previous activity as the input of the current one.

OGSA-DAI uses particular web pages for the user to explore which activities the server supports, and which activities the server's resources support. There are also web pages providing details for the client to package communication packets for web services.

Another important thing the user needs to know about is data resource. Security information like username, password and location of a database are hidden behind the resource ID. Administrators only need to use the OGSA-DAI server IP, Port and data resource ID, with which a user can operate the database.

This data resource can be located in the OGSA-DAI server site, or at another machine. The only requirement is that OGSA-DAI server can access it.

In the client side, the important thing is the 5 client toolkits for the user to use. To avoid showing the user the design detail of the system, an activity toolkit is usually provided by the server developer. Users will develop their program to use these toolkits to control the activities.

Finally, the user must import the proper classes, chain all the necessary activities, tell Data Request Execution Service to execute them, and get the result.

The details of the five Oracle activity implementations are described below:

**CreateDB** Creating a database link from master database.

This is achieved by using “CREATE DATABASE LINK” command from Oracle. This command uses a particular configuration style to represent a connection from a local to a remote Oracle database (it can be the same database if needed). There are 6 parameters: database link name, host, port, username, password and SID. SID is a specific parameter that identifies the database. The user needs to set up these parameters to the activity to create the link.

**Configure** Deciding the synchronization type and synchronize action time.

As described in the materialized view section, this configure action is going to set the behaviour of materialized view.

Configuration options include whether to use an incremental (Oracle calls it fast refresh) replication, or complete replication; whether to replicate when the master database is running a “commit” command; whether to replicate in a fixed period; and whether to replicate on-demand. (replicate when replica executes a “refresh”(synchronize) command) All these information are stored in the replica site identified by materialized view name.

**CreateReplica** Creating the initial replica at replica site.

Master table needs a materialized view log to do the incremental replication in Oracle materialized view replication environment. This log is used to identify the changes that have been made on the master table.

Flexibility of the replication comes from how the replica content is indicated. Materialized view uses SQL “SELECT” expression, which makes this replication very flexible. For example, materialized view with a SQL command “SELECT field FROM table WHERE condition” will only replicate the “field” columns in rows which meet the “condition”. Furthermore, it can use JOIN, UNION, MINUS and nested SQL to make more complex replication data pattern.

**Synchronize** Refreshing the replica site to get synchronized with master.

A refresh procedure of Oracle’s materialized view is shown by figure 3.6

This part is not difficult as we do not need to implement but re-use the replication mechanism from Oracle. A time-stamp is needed to keep in the replica site to indicate the time of the last synchronization.

**Monitoring** Displaying the content of replication configuration and time-stamp of last synchronization.

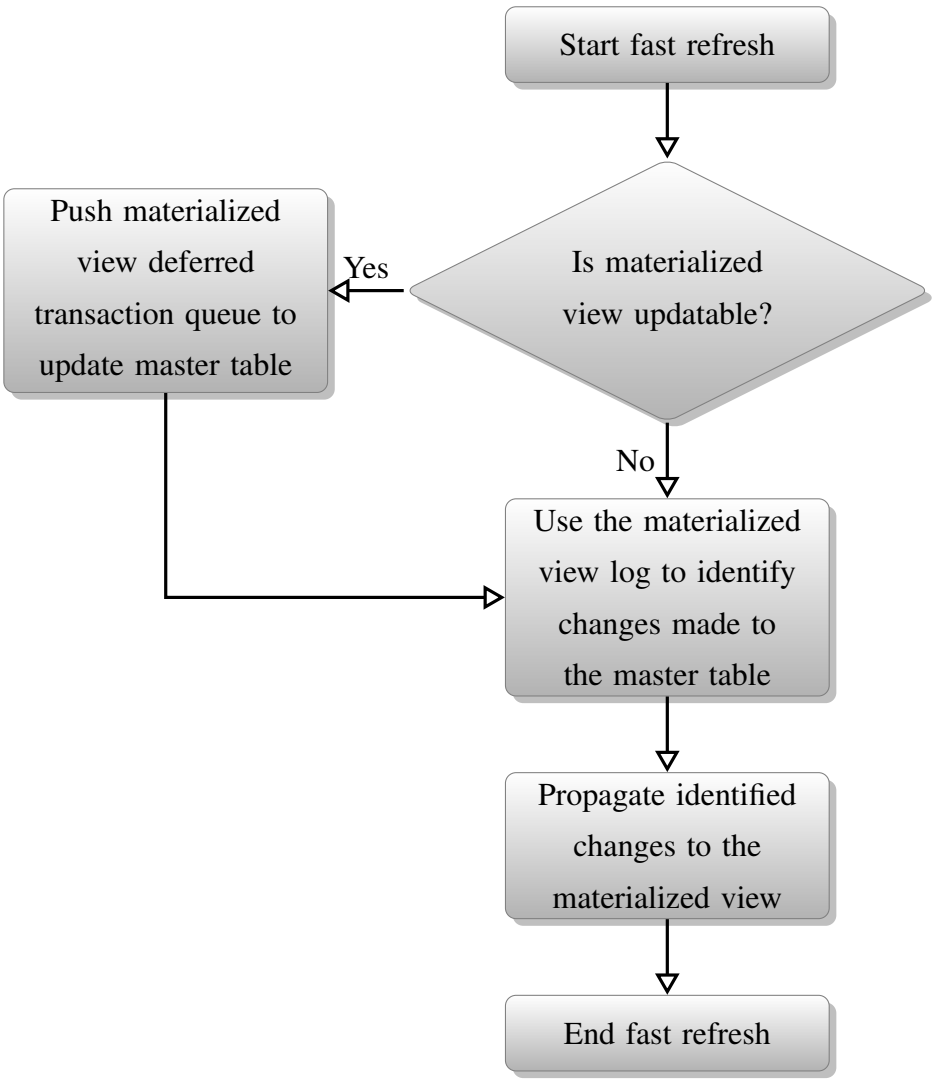



Figure 3.6: The procedure of refreshing a replica.[3]

In this part, client reads data in the replica to get relevant informations.

### 3.2.2 A common framework for database replication

After stated single database replication, we go on to discuss how to integrate more databases into GDRM by using abstract actions as figure 3.7 showed.

From the knowledge of IBM DB2 and Oracle, we know actions needed for a database replication are all database commands. Some of them are common SQL commands like “SELECT”; on the contrast, others are vendor specific commands, like“CREATE DATABASE LINK” in the Oracle. But all of them can be executed in their own JDBC drivers, which are provided by vendors for public use. So we are creating abstract

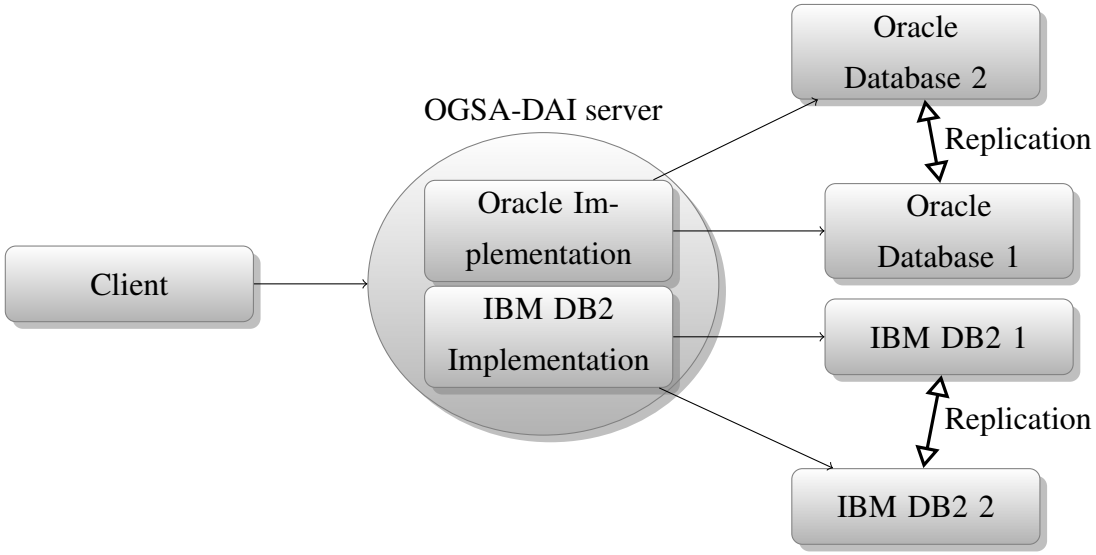



Figure 3.7: Abstracted classes represent heterogeneous databases

classes in Java, which receive replication commands from all databases. But the implementation of every database executes replication commands through their own JDBC drivers by indicating the ResourceID. OGSA-DAI here maintains the relationships between ResourceIDs and JDBC drivers.

In such a way, we hide as much as details from the user while leaving more configuration options. To be more specific, we force the user to override the abstract "inputParameters" method for every database implementation, which packages replication commands. There are also matched toolkits in the client side, which is also abstracted actions and need to be overridden.

When later a new database needed to be integrated into this framework, Developers only need to implement the server side activity and client side activity toolkit according to the requirements for the replication.

### 3.2.3 Replicate between different domains

In the Grid environment, cross domain is very common. Because every organization has its own network environment, operating systems and privilege control systems. Though Grid middle-ware provides the facility to connect virtual organizations, sometimes it is still difficult or even impossible for one organization to access another organization's network and database directly because everything is hidden behind the firewall and the Grid middle-ware. This will cause our original solution to fail. For example, the

database link can not be established without network connection.

To address such a problem, an intermediate solution can be used. A table is put in a site that both organizations can have direct network connections. Firstly, data from the master table is replicated into the intermediate table, then replicated from the intermediate table to the replica table. All the other replication parts are the same as the parts in a direct replication environment. Here we are still *re-using* database's replication mechanism on every single transfer. The intermediate site is a little complex because it takes a role of master table as well as a replica. Figure 3.8 shows the intermediate solution for the Grid database replication.

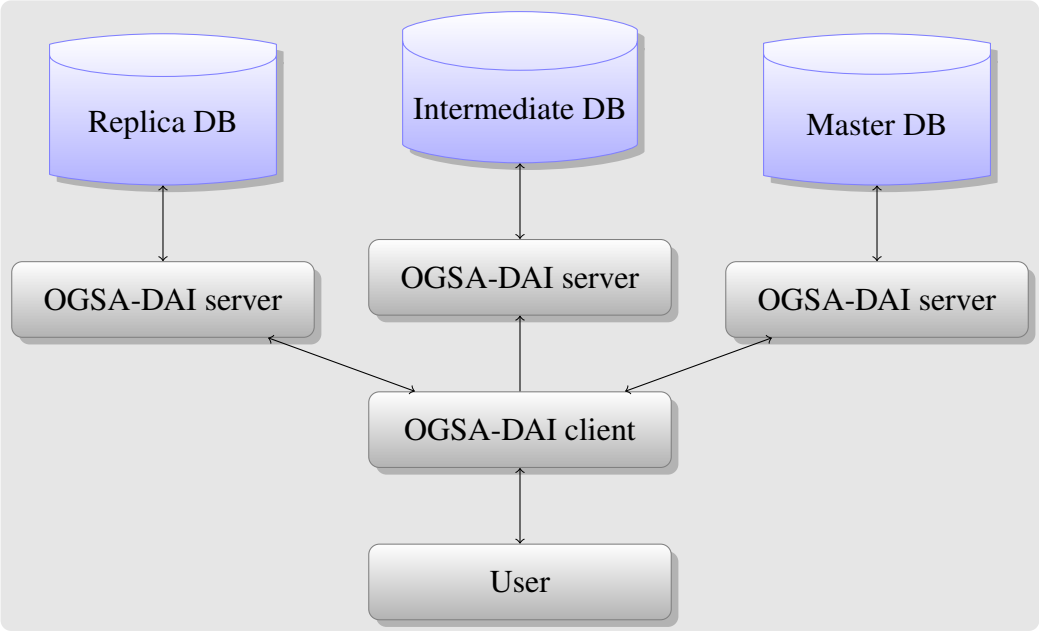



Figure 3.8: Intermediate solution scenario

In this way, the intermediate table keeps serialized consistency with the master table and the replica table keeps serialized consistency with the intermediate table. So the replica still keeps serialized consistency with the master table.

For Oracle replication, materialized view is enough to take the role of the intermediate table, because it is able to be a master while being a replica. Cross-domain part has not been implemented in the current project.

### 3.2.4 Replicating between heterogeneous databases

Heterogeneous replication between different databases means the changes that happened on database A need to be happened on database B, while A and B are from different vendors. Figure 3.9 shows a way of replicating between heterogeneous databases.

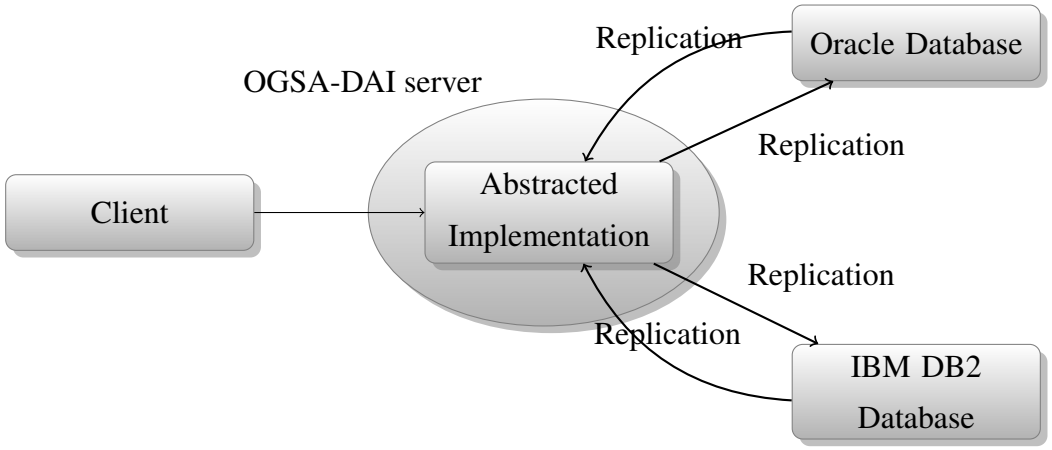



Figure 3.9: Replication between heterogeneous databases

In this figure, OGSA-DAI controls both master and replica databases. It gets changes from the Oracle database and applies them to the IBM DB2 database.

One of the critical problems is, though different vendors follow the instructions of SQL standard, they have lots of differences on implementation of their specific extensions. Replication is a good example, there is no information on how to replicate in the standard Database Language SQL 1992[15]. So every vendor has its own commands.

Fortunately, commands which manipulate data are almost standard SQL commands. So if we can map all the data manipulation commands to standard SQL commands(discarding the nonstandard ones), and then apply them to the destination database, we can achieve a heterogeneous replication.

There are 3 important parts for a heterogeneous replication:

**Capturing:** This part finds the changes in the master table and extracts them out from the database management system.

As mentioned before, to realize incremental replication, changes are described in SQL language. Commands for changing table content include INSERT, DELETE and UPDATE, this kind of commands is called Data Manipulation Language(DML). There are also commands which change the table structure, they are called Data

Definition Language(DDL). These two parts are very different in definition, and also in the way of affecting replication.

DML can be used to make an efficient incremental replication, whereas DDL is more complex as it does not only concern data, but also changes the data type by changing the structure of how data storing. DDL sometimes includes vendor specific changes to the database, this kind of changes make replication very difficult to realize since they are hard to map to standard commands. Considering that DDL may greatly change the data object and potentially change vendor specific data structure, it is better to do a complete replication when DDL is met in the replication. However, a low frequency of using DDL limits its effect.

**Transforming:** This part is not the most difficult one, but it is usually very complex and needs a lot of work. Because a transformation is a mapping from one vendor's database to another vendor's. There are many things in database management systems, and every two databases need a checking: finding differences and making schema mappings of them. Even the commands from different database versions of the same vendor may be different when doing the same thing.

Another problem is that some of the commands are hard to map to any other vendors' databases. In this situation, there needs to be a complete synchronization.

But once a mapping is done, it will be a big advantage to the heterogeneous database replication.

**Applying:** This part is the easiest one. After mapping, all the SQL commands are ready for use and compatible with the replica databases.

The procedure of capturing, transforming and applying are shown in Figure 3.10.

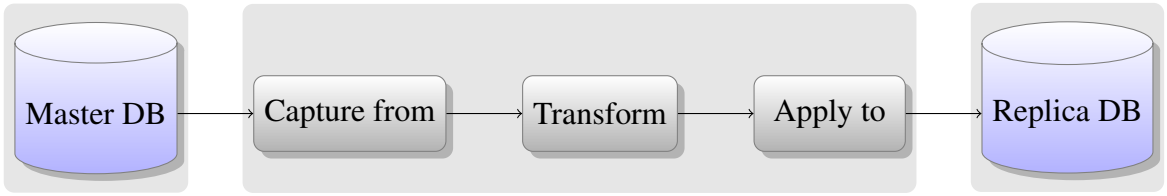



Figure 3.10: Actions needed for a heterogeneous replication.

An actual implementation needs to extract SQL commands from database management systems. Some databases have already provided a way to extract history SQL

commands. For example, Oracle has a tool called Log Miner, which is used to extract SQL languages.

However, most databases do not provide a direct way to extract SQL history. But because DBMS has to maintain atomicity, consistency, isolation, durability (ACID) itself, it has to record all the SQL commands that happened in the history. These records are usually stored in *redo* and *undo* logs, which are used in database management systems to recover from a crash. Because they contain all the SQL commands in the history, analysing them will produce commands we need to carry out replication. Such a task is not easy, as not all the databases have an open guide on their log system, and every vendor has its own way of doing this.

Another way of implementing heterogeneous replication is again re-using database vendor's replication mechanisms.

Some large DBMSs have their own way of doing heterogeneous replications in a non-Grid environment. For example Oracle 11g's stream replication method support heterogeneous replication; Sybase's Replication Agent can control replication between MS SQL Server, Oracle and IBM DB2 to Sybase database; and IBM DB2 can use non-DB2 databases as replication source. Re-using such mechanisms with OGSA-DAI middle-ware makes it possible to perform heterogeneous replication through Grid environment.

The mapping replication method is a wide-compatible way, and it can do a 2-way replication between all the databases that already have a mapped schema; on the other hand, the re-using method is much easier and more efficient, but this method is more specific in that master databases are limited to some specific vendors' database versions, which need to have the heterogeneous replication mechanism. If using mapping method in a cross domain environment, we only need to add a transformation action in the intermediate site. Though highly difficult, but if supported, the mapping method will surely be an easier way of doing heterogeneous Grid database replication.

Heterogeneous replication and schema mapping between different databases are not implemented in this project.

# Chapter 4

# Implementation

This chapter will describe how do we implement the Oracle database replication through Grid environment using OGSA-DAI middle-ware.

## 4.1 Implementation of the server activities

Figure 4.1 gives a view of the Java class connections of the server side activities for heterogeneous replication architecture.

Because all the actions regarding to the replication are mostly SQL commands, I created a SQLBase class in the uk.org.ogsadai.activity.replication package. This is an abstract class that extends from MatchedIterativeActivity and implements ResourceActivity. MatchedIterativeActivity is one of the base activities to be extended by others. It is the mostly common used one as it meets most requirements like iterative input and match-checking of the input. ResourceActivity is an interface that provides resource access ability [16].

Methods defined in the ResourceActivity are implemented in SQLBase, including functions for getting data input; making connection to the database resource, executing SQL commands and cleaning up. Among all of these, only the methods processing input data are abstract and are required to be implemented in the sub classes, because every action uses this method to package commands.

The parameters and corresponding SQL commands are listed below:

- **ConnectDB**

  CREATE DATABASE LINK *DLName*
  CONNECT to *username* IDENTIFIED BY *password*

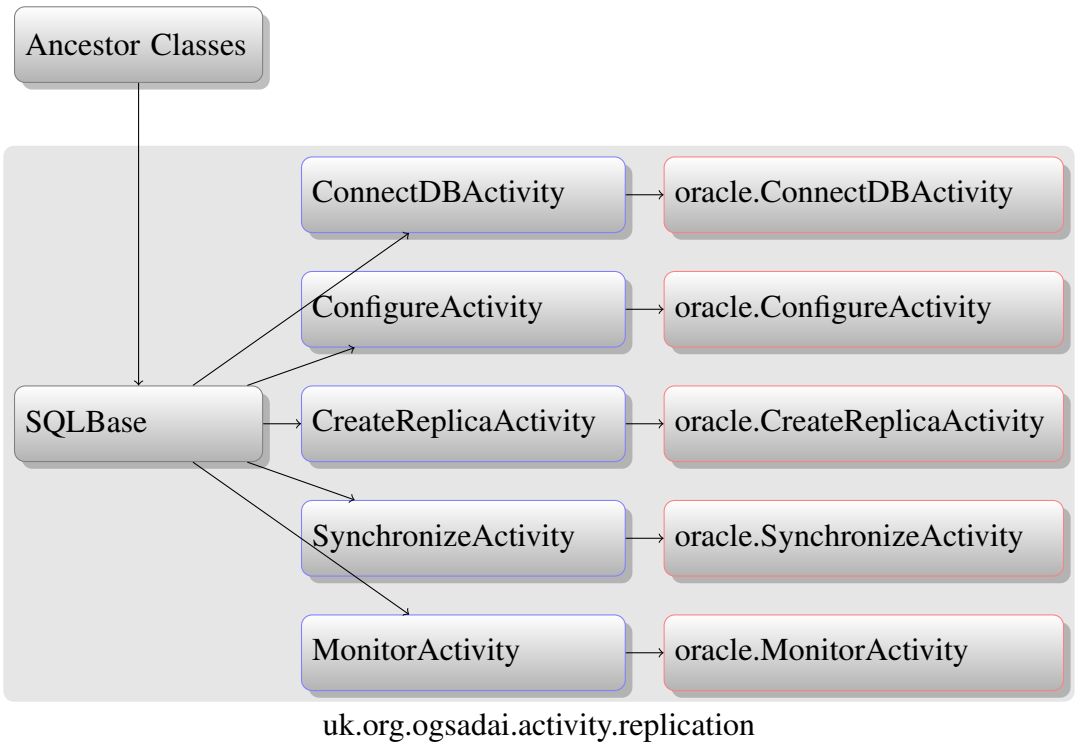



Figure 4.1: The hierarchical structure of the server side activities.

USING '( DESCRIPTION =
(ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)
(HOST = *host*)(PORT = *port*)))
(CONNECT_DATA = (SID = *sid*)))';

- **Configure**

  Storing *DLName*, *masterTable*, *MVName*, *isIncremental*, *isOnDemand*, *syncPeriod* in the replica's property file "user.home/RepConfig/MVName.property".

- **CreateReplica**

  CREATE MATERIALIZED VIEW LOG on *masterTable*;

  CREATE MATERIALIZED VIEW *MVName*
  REFRESH *FAST or COMPLETE or*
  *ON [DEMAND or COMMIT] or NEXT syncPeriod*
  AS SELECT * FROM *masterTable@DLName*;

  This command is created with the parameters from the previous step. A creation time-stamp is also recorded in the replica's property file.

- **Synchronize**

  BEGIN DBMS_MVIEW.REFRESH(*MVName*) END;

  This SQL command is used only for the on-demand refresh. The time at which the command is executed is recorded in the property file.

- **Monitor**

  Display *DLName*, *masterTable*, *MVName*, *isIncremental*, *isOnDemand*, *syncPeriod* from replica's property file. Period refresh time-stamp can be known by calculating start time and synchronization period.

As all the actions need to receive parameters from the client, we have implemented a method in SQLBase providing a uniform way to receive parameters for all the sub classes. This is realized by:

1. Instead of using multiple parameters, we use one parameter in the communication;

2. All the parameters for the action are re-structured in the client toolkit – adding them together in one string, separated by comma;

3. Implementing the "processIteration" method in the activities inherited from the SQLBase class, which processes parameters and uses the result to form a SQL command.

In detail, the processInteration method splits the string into an array by taking comma as separator; then following the predefined order, parameters are packaged together with the skeleton already present in the sub class to create a SQL command. This is the actual command to be executed in the database. The real execution method is again in the SQLBase, which is used by all the actions.

In this way, the number of parameters in sub classes can be changed easily. For example when the database system is upgraded and a extra parameter to be processed, the developer only needs to change the predefined style of parameters both in the server activity and the client toolkits.

This view is shown in figure 4.2

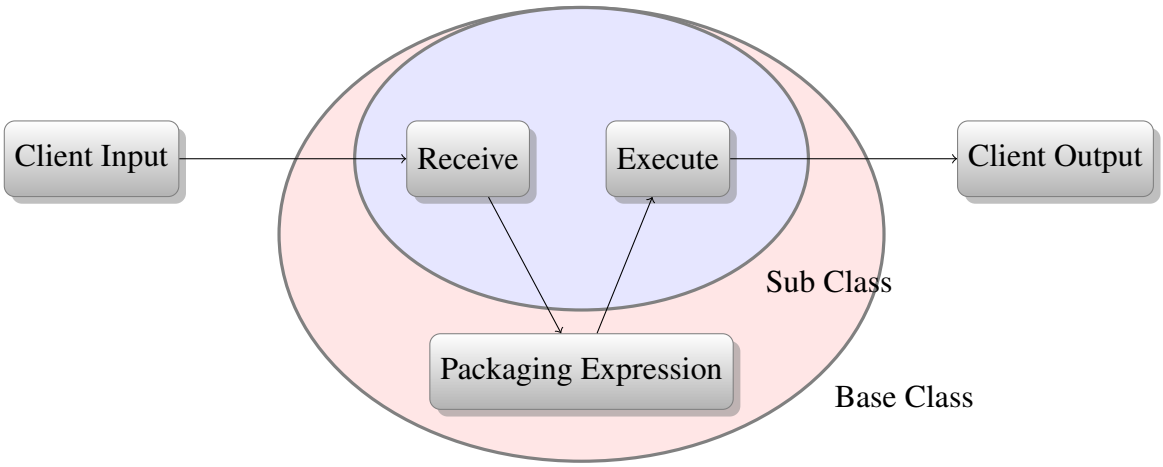



Figure 4.2: Using override and call back to achieve a tidy execution.

## 4.2 Implementation of the client toolkits

The aim of toolkits is to provide a convenient way of accessing server activities, in order to avoid user trapping into OGSA-DAI detail. It acts like a bridge, which sends the input from user programs to server activities and gets the output from server activities.

Because of the abstract design in the server side, we can simply use one toolkit for all our replication actions. But to make developing end programs easier and to follow the style of inheritance in the server, we developed matching client toolkit for every action.

Figure 4.3 shows the inheritance structure of the client toolkits.

Base class extends from uk.org.ogsadai.client.toolkit.activity.BaseResourceActivity, and implements uk.org.ogsadai.client.toolkit.ResourceActivity. They provide access to resources, communication with servers and other base services. The base classes wait for parameters from the user program, send packaged input, wait for the result from the server and provide methods to show the result. All the concrete action classes only need to pass ActivityName to the Base class.

This ActivityName is not a real name of the class in the server side. Data resource deploying in the OGSA-DAI server makes every activity class has a unique virtual ActivityName, for example uk.org.ogsadai.ConnectDB is the activity name of the class uk.org.ogsadai.activity.replication.ConnectDBActivity in our implementation. In this point of view, we can take the client toolkit as a presentation layer of server activities.

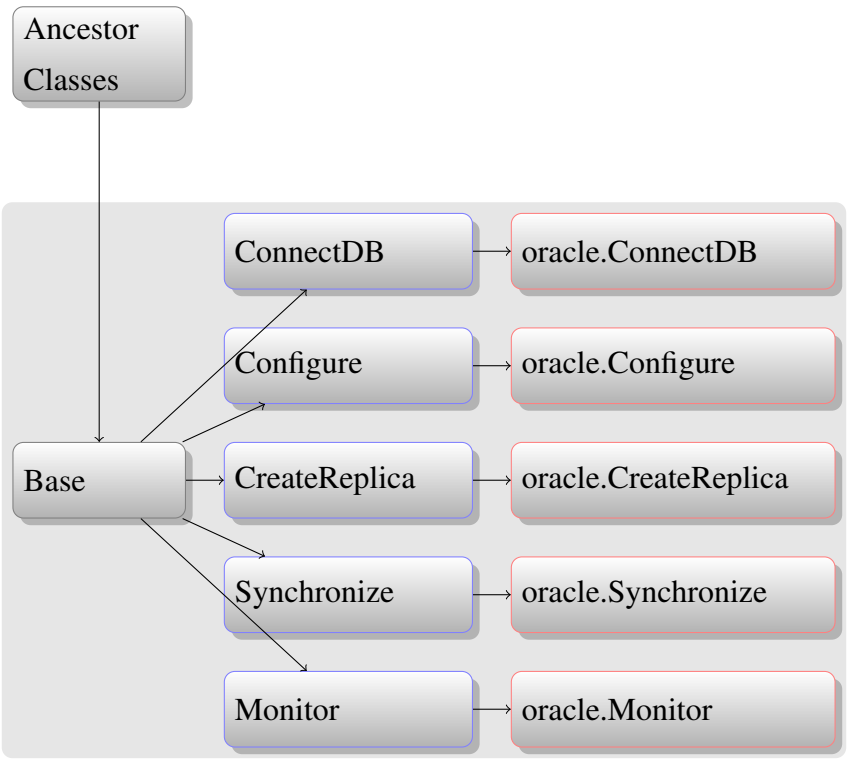



Figure 4.3: The hierarchical structure of the client side toolkits.

## 4.3 Implementation of the client end program

Toolkits alone are not enough to control the replication, thus the user program organizes all the things, including connecting to the server, preparing the activities, sending parameters and displaying the returned data. This is shown in Figure 4.4.

Client connects to the proper OGSA-DAI server by setting URL in the server proxy; then it gets the data request execution services by setting the execution service's resource ID. After that, it puts all the activities needed in a pipeline work flow. These activities have already been configured with proper parameters. This pipeline is sent to be executed in the server's execution service, and finally the results are shown to the user.

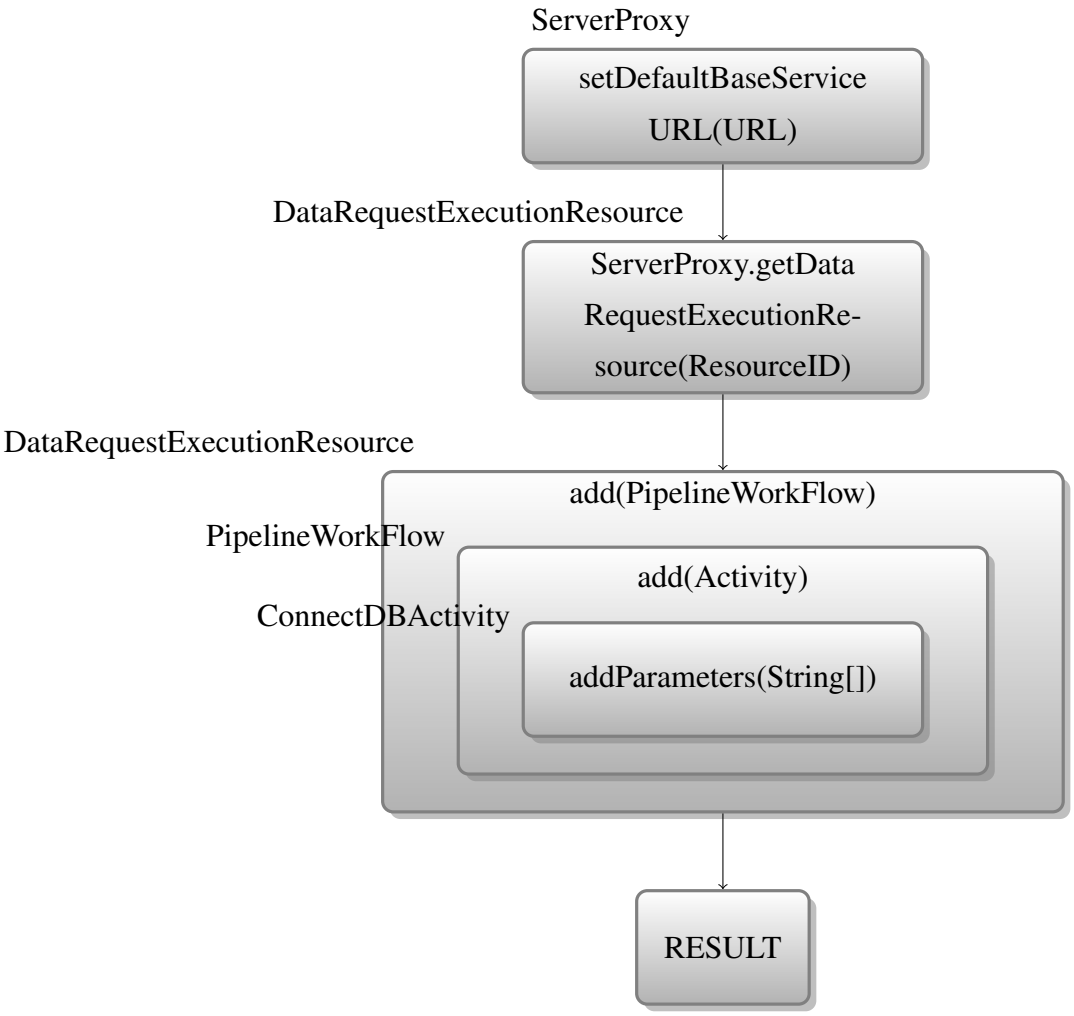



Figure 4.4: The procedure of invoking server activity in user program.

# Chapter 5

# Experiments and Evaluation

This chapter presents how we carried out the experiments on our implementation and compares performances of replication through OGSA-DAI and replication directly. After that, we evaluate OGSA-DAI 3.0 and discuss the differences between IBM DB2 replication and Oracle replication.

## 5.1 Experiment results

The experiment includes 3 parts: OGSA-DAI server invoking time, materialized view creation time, and synchronization time. These three parts together not only indicate expected behaviour of our program, but also show adequate performance and functionality of OGSA-DAI and materialized view.

### 5.1.1 Experiment environment

OGSA-DAI with our activities has been deployed to 3 test boxes, their configuration details are listed in Table 5.1.

A successful deployment shows that OGSA-DAI can work on different operating systems. Tomcat 5 and 6 branch's latest version can both take the role of container properly. All the testing boxes worked smoothly during the experiments.

OGSA-DAI and Oracle XE have been deployed on both Windows and Linux boxes. There is only OGSA-DAI on FreeBSD box. Windows box is taken as the master database, Linux as the replica database and FreeBSD box as the OGSA-DAI client.

Thus FreeBSD box's OGSA-DAI client accesses Linux box's OGSA-DAI server to create database link, create materialized view and keep synchronized with the database

| Item | Box 1 | Box 2 | Box3 |
|---|---|---|---|
| Operation System: | Windows XP | Scientific Linux 5 | FreeBSD 7 |
| Java Version: | 1.5.0_06-b05 | 1.5.0_16-b02 | 1.5.0_14-p8 |
| OGSA-DAI Version: | OGSA-DAI GT 3.0 | | |
| Globus Version: | Globus Toolkit 4.0.5 | | |
| Tomcat Version: | 5.5.26 | 6.0.16 | 6.0.16 |
| CPU: | Pentium 4 1.80GHz | Pentium 4 2.4GHz | Turion64x2 1.8GHz |
| Memory: | 1GB | 1GB | 2.5GB |
| Network: | 100M LAN | 100M LAN | 54M WLAN |
| Oracle Version: | Oracle XE | Oracle XE | N/A |

Table 5.1: The experiment environment.

in Windows.

The network topology is shown in Figure 5.1.

### 5.1.2 OGSA-DAI invoking time

In this part, I will use an empty test activity to show how long it takes to invoke an empty call to the OGSA-DAI server. By doing this we can have an overview of architecture performance of the OGSA-DAI.

This test takes Linux as the OGSA-DAI server and FreeBSD as the OGSA-DAI client. As FreeBSD is in a wireless environment, the result is expected to have a large random difference. This test will run for 10 times to get an average time. To make sure it can indicate the performance in a production environment, tests run while the Oracle was working.

The test results are shown in Table 5.2. From the table we can see that OGSA-DAI needs about 3.8 seconds to finish one empty call. The *time* utility is used to record time, a shell script is used to run the test for ten times and make sure that every round has a fixed interval.

A strange behaviour of OGSA-DAI was noticed. If the OGSA-DAI client and server are in the same box and request from client calls server very frequently, i.e. run another round immediately after the current round, the time cost is very high. For example, using 1 second as the interval makes invoking time increasing to more than 6 seconds and sometimes the time can reach up to even 14 seconds, which is 2 – 4 times of the

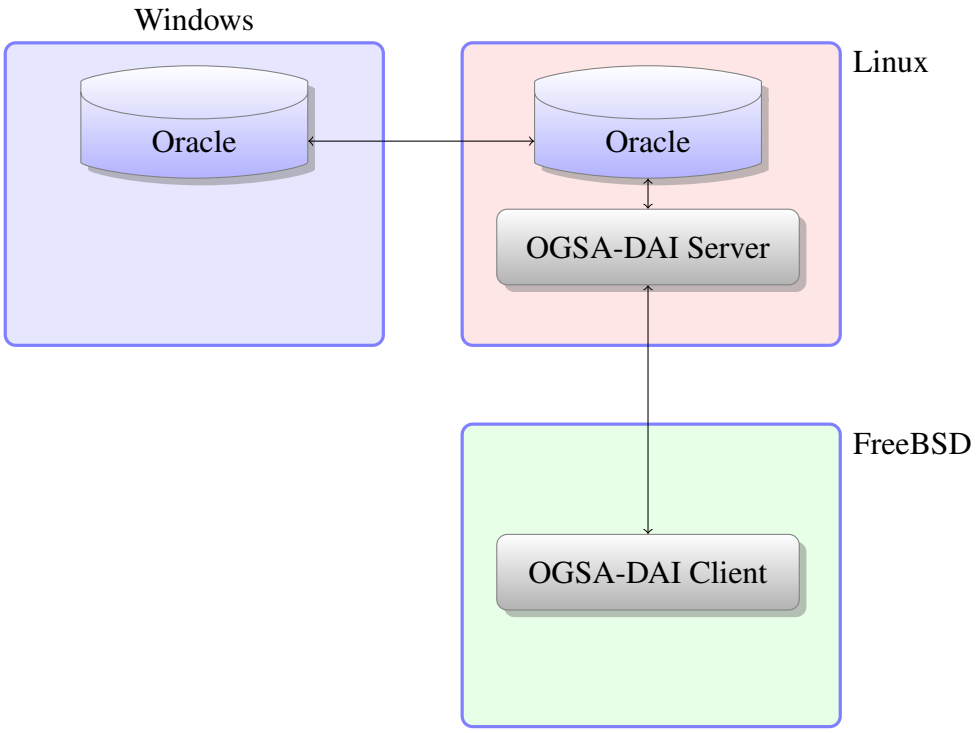



Figure 5.1: The network topology of the testing system.

normal time. Furthermore, nearly 50% of CPU load is occupied. But running the test at a 60 seconds interval keeps the server in a normal load when the client and server are on the same machine. It costs even less time than using a remote client. On the other hand, invoking by remote client gets a high performance, no matter how big the interval is.

From the observations above, we can tell that this performance problem is caused by using the OGSA-DAI server and client on the same machine, but which part of software produces such a problem is still unclear.

So there will be a dramatic performance drop down of the OGSA-DAI system if both the server is busy and it is used as the client. However, the replication actions usually do not need to be invoked very frequently. Usually the ConnectDB, Configure and CreateReplica actions only need to be run once in the life time of one replication. Synchronization may run a little more frequency than others, but it usually has a much larger interval than one minute. If the synchronization model is chosen to be run in a automatically fixed time interval style, the OGSA-DAI invoking time cost on synchronization will be zero. Monitoring is also a infrequent action. And usually people will not use the server as a client. Thus, this kind of performance drop down will not affect

| | Time cost(seconds) |
|---|---|
| 1 | 4.37 |
| 2 | 3.69 |
| 3 | 3.84 |
| 4 | 3.69 |
| 5 | 3.79 |
| 6 | 3.75 |
| 7 | 3.90 |
| 8 | 3.71 |
| 9 | 3.84 |
| 10 | 3.86 |
| Average: | 3.84 |

Table 5.2: Time cost of empty calls to local OGSA-DAI server.

replication too much.

### 5.1.3 Materialized view creation time

In this part, we will use CreateReplica activity to create an initial replica table from the master database. We will compare the replication between creating through OGSA-DAI and creating directly using Oracle. By doing this, we can see the SQL execution performance of OGSA-DAI. Because we are re-using the replication mechanism of Oracle, the comparison between them should not have much difference, and time incremental trends on OGSA-DAI side should be similar to the incremental trends on direct Oracle side with the increase of the rows.

We create the test master table using class dbcreate.CreateTestOracleDB, which will produce a contact book table containing random data. This test table contains 4 columns as shown in Table 5.3.

We create masters table which contains 1000, 10000, 100000, 250000, 500000, 750000 and 1000000 rows of data, respectively. These tables did not have a primary key which was required by materialized view log, so after creating them, we put a primary key on the ID column using “ALTER TABLE master ADD PRIMARY KEY(id)”, and put materialized view log on them. Then we used the CreateReplica activity to produce replica table.

| Name | Type | Length |
|---|---|---|
| ID | NUMBER | 38 |
| NAME | VARCHAR | 64 |
| ADDRESS | VARCHAR | 128 |
| PHONE | VARCHAR | 20 |

Table 5.3: The structure of test table.

| # of Rows | Create Through OGSA-DAI | | | | Create directly | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | ave. | 1 | 2 | 3 | ave. |
| 1000 | 4.83 | 4.59 | 4.74 | 4.72 | 0.325 | 0.443 | 0.232 | 0.333 |
| 10000 | 4.50 | 4.61 | 4.76 | 4.62 | 0.656 | 1.368 | 0.335 | 0.786 |
| 100000 | 6.17 | 6.30 | 5.97 | 6.14 | 2.919 | 1.751 | 1.723 | 2.131 |
| 250000 | 7.76 | 7.73 | 7.78 | 7.76 | 4.156 | 3.795 | 3.684 | 3.878 |
| 500000 | 12.08 | 11.14 | 11.43 | 11.55 | 6.908 | 6.520 | 6.565 | 6.664 |
| 750000 | 14.30 | 13.55 | 13.97 | 13.94 | 10.520 | 10.483 | 10.126 | 10.376 |
| 1000000 | 19.55 | 17.75 | 17.33 | 18.21 | 14.035 | 14.031 | 14.384 | 14.150 |

Table 5.4: Replica creation comparison between OGSA-DAI and direct

Test of every row number ran for three times to get an average time. If we had enough time, we would use at least five times to get a more accurate average time. The result of this test is shown below in table 5.4:

From the table we can see the time cost on direct creation is very steady, especially when the number of rows is large. This is because the two Oracle databases are in a LAN environment, in which connection is very stable. But the time of creating replica through OGSA-DAI's is affected by the network communication of the OGSA-DAI client and OGSA-DAI server, so it's a little less steady, though not too much. If using OGSA-DAI client in a more unstable environment, when behind an ADSL router, to access OGSA-DAI server, the cost is very fluctuating.

Because the initial replica table is a complete copy of the original table, we can see the time increment of two methods are both approximately linear.

Figure 5.2 shows the comparison of the two methods.

From the figure we can see that the difference between the two methods for each number of rows is almost same as the empty invoking time.

A interesting thing is the time cost on making replica through OGSA-DAI on 1000

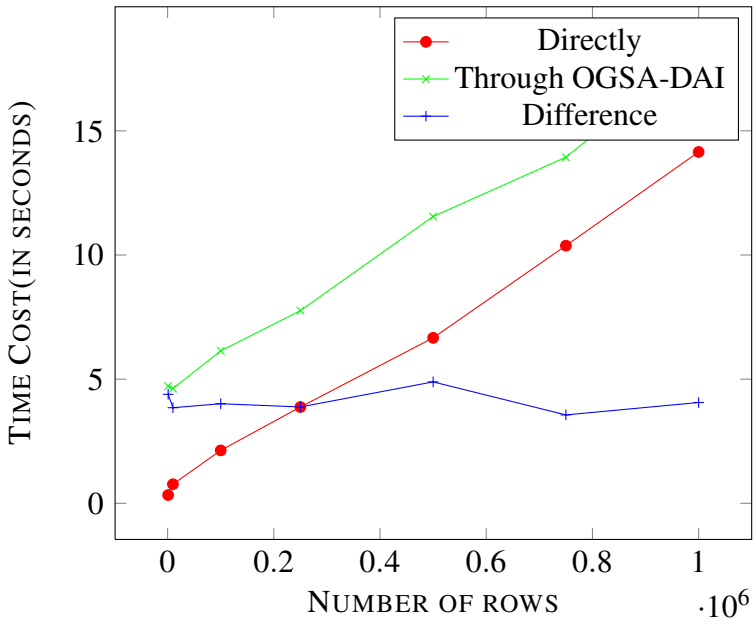



Figure 5.2: Comparison of initial replica creation through OGSA-DAI and directly.

and 10000 rows. Replication 10000 rows cost less time than replicating 1000 rows. This may be because their difference on direct creations are on the level of $10^{-1}$ seconds, while the invoking time is about 4 seconds, so the little difference between replicating 1000 rows and 10000 rows can be ignored.

### 5.1.4 Materialized view synchronization time

It is a similar experiment as creating replica. We are going to use Synchronize action to synchronize the replica table with the master table. Moreover, we are going to delete data whose id is larger than 1, which means almost every data is changed and only the row whose id is 1 is kept. In such a situation, if the replication is a complete synchronization, then the time cost will be similar to the creation cost of table containing only one row. In theory, this cost will be very small. But if the replication is using SQL command to do an incremental replication, the time cost is similar to the cost of running SQL command directly on the database.

We are using the replica tables that have already been created in the previous step, but only use tables having less than and equal to 250000 rows, as they are enough to indicates the incremental synchronization changes.

It is an important point to note that in Oracle, commands concerning the remote database need a COMMIT command to make sure all the affiliates get the notice of the

| Item | | 1000 | 10000 | 100000 | 250000 |
|---|---|---|---|---|---|
| Synchronize | 1 | 5.17 | 10.81 | 64.08 | 164.13 |
| through | 2 | 5.04 | 9.27 | 66.47 | 166.93 |
| OGSA-DAI | ave. | 5.11 | 10.04 | 65.28 | 165.53 |
| Synchronize | 1 | 0.644 | 5.841 | 59.135 | 163.782 |
| directly | 2 | 0.662 | 5.853 | 60.954 | 164.729 |
| | ave. | 0.653 | 5.847 | 60.045 | 164.256 |
| Delete | 1 | 0.719 | 6.937 | 71.453 | 186.968 |
| directly | 2 | 0.734 | 7.250 | 70.594 | 183.859 |
| | ave. | 0.724 | 7.093 | 71.024 | 185.413 |

Table 5.5: Replica synchronization comparison between OGSA-DAI and direct

changes.

We compare the time cost of deleting in the database prompt, directly synchronizing in the database prompt and synchronizing through OGSA-DAI. All the tests are run for 2 times to reduce random errors. If we had time, we would have run the tests for at least 5 times to reduce random errors.

Table 5.5 shows the test results of Synchronization.

From the table we can see that time costs of all the methods have a linear increase with the increase of the number of rows. We noticed that direct synchronizations have a lower time cost than delete actions in the master site. This is because both of them execute only one SQL command. But since the replica site's machine has a better hardware(the CPU is faster), the cost on computing overcomes the cost on 100M LAN network. We can also see that the synchronization through OGSA-DAI keeps the functionality of the original replication during the experiment.

We use Figure 5.3 to give a more clear comparison of the three methods.

## 5.2 Evaluation

### 5.2.1 Differences between OGSA-DAI 2.2 and OGSA-DAI 3.0

According to the above experiments, we find that our implementation of GDRM Oracle Grid replication is suitable for replicating Oracle database through Grid environment constructed by OGSA-DAI 3.0 on top of Globus toolkit. In this work, we use the latest

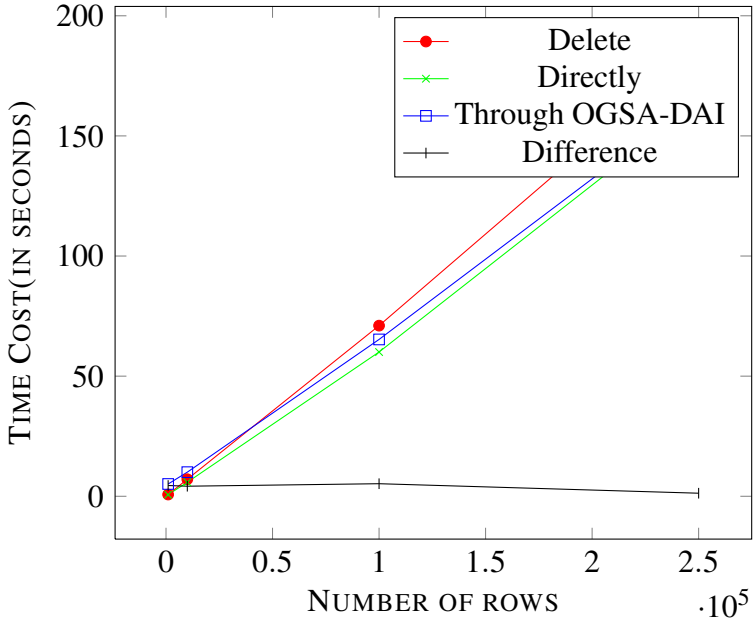



Figure 5.3: Comparison of synchronization through OGSA-DAI and directly

version of OGSA-DAI – 3.0, while the original work from Chen was based on OGSA-DAI 2.2.

Though lots of the OGSA-DAI details are changed, its main functions remain. All the actions are composed by activities, services and resources. Resources expose data that need to be manipulated, services expose the activities that can provide functions to solve the problem.

But the new version is more powerful and hides more details from users. For example, developers do not need to write XML files to describe activities anymore, and tools like Ant are used to take care of deploying activities automatically.

During the using of OGSA-DAI 3.0, I found some possible improvements.

- OGSA-DAI's activity name space does not operate in a natural way.

  Developers write classes in Java name space, but deploy it using another name space – activity ID, and then a third name space is used to put them into a resource file and this one is used in the actual work – activity name. So one activity gets three unique labels which have some relations among them. This may cause some confusions.

  In my opinion, the Java name space rule is enough since it is designed to reduce the probability of having two different class with the same name.

- I noticed a performance drop down when using OGSA-DAI client program doing frequent invoking in the same machine as the OGSA-DAI server. Nearly half of the CPU load is used by the client. There is a more detailed description in the section 5.1.2.

- Another question is that users cannot easily get to know what's wrong with their program when encountering a problem. Because the exceptions emitted from the remote site are not transfered back to the user, if the user cannot get to the server and check the log of the container, there is no way to know the problems.

  I think this could be improved by catching the exception and implementing a transfer activity between the client toolkit and server activity to show the exceptions directly to the user. And this can be configured by the administrator to allow only some of the logs being accessed.

### 5.2.2 Differences between IBM DB2 and Oracle's replication

IBM DB2 and Oracle both have more than one way of doing replication. IBM DB2 has DataPropagator, SQL Replication and Queue replication; while Oracle has materialized view replication and stream replication.

As for IBM DB2, DataPropagator is replaced by SQL Replication and Queue replication in the newer version of WebSphere. In fact, SQL replication is a powerful replacement of DataPropagator. SQL replication has the functions that DataPropagator has, and also has a new monitor function that DataPropagator do not have. Queue replication uses parallel applied method to apply changes in the queue to the replica. It tries to provide a method of low latency and high throughput replication.

As for Oracle, stream replication is newer than materialized view replication. It has newer functions which support heterogeneous replications and uses an asynchronized way to capture changes while materialized view uses a synchronized way. But since materialized view is very mature, easy to develop and easy to use, it has already been used in lots of places. The free Oracle XE edition supports materialized view replication while only the commercial edition supports stream replication.

Chen has used SQL replication to replicate IBM DB2 database while we are using materialized view to replicate Oracle database.

The differences between them are: SQL replication from IBM DB2 needs to construct a intermediate table to store the changes from the master and then apply them to the replica. While materialized view from Oracle does not, it needs a database link on

replica database to the master database. It also needs a materialized view log on the master table to indicate changes that have happened on the master.

However they have similar ways of doing replications in concept, both of them use incremental replication method. They capture the changes, and then use different ways to apply them on the replica table. Such a difference comes from the implementation detail, but not the concept.

# Chapter 6

# Conclusion

## 6.1 Conclusion of project work

This work was intended to implement a database replication through the Grid environment, and investigate the ways of doing heterogeneous replication. It is based on the work from Chen, who proposed Grid Database Replication Model, which we trying to extend.

We have discussed a serialized incremental consistency that is suitable for Grid replication. It is a relaxed consistency that only locks the replica that is doing the replication, and needs all the incremental changes to be applied on different replica in a same order – the order of producing them. We also have looked into incremental replication methods, which capture changes and apply them on the replica sites.

We implemented our replication method by (1) re-using database's mature replication mechanism, and (2) using standard Grid middle-ware to provide Grid facilities. We have chosen Oracle's materialized view as the replication mechanism to re-use, and OGSA-DAI as the Grid middle-ware. We chose materialized view because it provides an easy and flexible way to do the replication. It supports incremental replication, multiple masters replication and subset replication. OGSA-DAI is a standard middle-ware to do data access and integration.

In order to adopt more databases in the future, we designed the abstract replication classes by making them inherit from five common actions used in the replication. They are ConnectDB, Configure, CreateReplica, Synchronize and Monitor. The Oracle materialized view implementations are inherited from these five actions and having a prefix of oracle.

Then we investigated the way of replicating through cross-domain and replicating

between heterogeneous databases. Cross domain problem is possible to be solved by adding an intermediate table, which has connection to both the master and replica site. Heterogeneous replications can be achieved in two ways: 1, mapping specific command to standard SQL commands; 2, re-using heterogeneous replication mechanism with OGSA-DAI that provides Grid facilities. First one is difficult to implement especially on schema mapping, but is widely compatible while the second is easy but limited in the type and version of databases.

At last, we have tested the performance of our implementation. The result shows that all the actions behave as expected. The creation experiment indicates the implementation has an adequate performance, its a linear trend on time cost. The synchronization experiment shows the implementation does replication in an incremental way. We have also pointed out some problems in the experiment, and possible ways of solving them.

We have not implemented the cross-domain replication and heterogeneous schema mapping, leaving those for future work.

## 6.2 Future work

Future work can be focused on the below aspects:

1. Cross domain implementation.

   If two organizations do not have a direct network link, even if they both access to the Virtual Group's resource, the current work still can not make them replicate to each other. To cope with this problem, an intermediate table is a good solution.

   Such a solution is useful on the same vendor's replication, and also important to heterogeneous replication between different databases. Because besides transport, the intermediate site can also be a transform site to merge differences between different databases.

2. Schema mapping between databases.

   This part will need a lot of work, because the number of differences between databases are hard to recognize and there are many databases. It is difficult also because database vendors will not open their design architecture to the public.

   This part is one of core parts of heterogeneous replication. A successful schema mapping will make heterogeneous replication between databases possible.

3. Integration of other DBMSs into this model.

   More DBMSs integrated into the model will make this model more practical and useful.

   The criteria of choosing a proper database relies on whether this database has a way of replicating itself to its own kind. And a better candidate would be easier to extract redo logs from the database.

   Following the first criteria, modern databases like Sybase, Microsoft SQL Server, MySQL and PostgreSQL are all good candidates for replicating, because they all have the ability of homogeneous replication. But open source softwares like MySQL and PostgreSQL would be easier to extract redo log files as their source code is able to be checked.

# Appendix A

# Configuration OGSA-DAI

## A.1 Preparation

### A.1.1 OGSA-DAI 3.0 GT

Download from http://www.ogsadai.org. Registration is needed. Download the src package for later use. Get ogsadai-3.0-gt-4.0.5-src.zip file to $HOME.

### A.1.2 Globus Toolkit

Download from http://www.globus.org/toolkit/downloads/4.0.5/#wscore_bin
As OGSA-DAI document suggested, we use 4.05 version of Globus Tookit instead of the recommendation one from Globus website. Download the bin core file of Globus toolkit for later use. Get ws-core-4.0.5-bin.zip file to $HOME.

### A.1.3 JDK5

JDK5 is needed, though SUN has already move themselves to support JDK6.

### A.1.4 apache-ant

Download from apache website http://ant.apache.org/, get the newest version. Get apache-ant-1.7.0-bin.zip file to $HOME.

### A.1.5 Oracle XE

The eXpress Edition is the free edition of Oracle database management system. The latest edition is Oracle XE 10.2, which can be downloaded from
http://www.oracle.com/technology/products/database/xe/index.html
Registration is needed. Get oracle-xe-10.2.0.1-1.0.i386.rpm file to $HOME.

### A.1.6 Tomcat

Download latest version from http://tomcat.apache.org. It is used as a container for OGSA-DAI. Get apache-tomcat-6.0.16.zip file to $HOME.

## A.2 Installation

Unzip all the zip files to to their default directory in the $HOME directory, by execute: unzip x.zip. This will extract apache-ant, apache-tomcat, OGSA-DAI and Globus toolkit to the home directory. Assuming JDK has already installed in the system, if not, please refer to the SUN website.

### A.2.1 Installing Oracle

Execute "> rpm -i oracle-xe-10.2.0.1-1.0.i386.rpm", it will point you to configure when it finishes.
Execute "> /etc/init.d/oracle-xe configure" to configure Oracle, using the default value and set the password.
Execute "> source /usr/lib/oracle/xe/app/oracle/product/10.2.0/server/bin/oracle_env.(c)sh" to set the environment variables. It is suggested to put this line into your .bashrc|.cshrc file. Test the Oracle is installed successfully by executing "> sqlplus SYSTEM/password". It will shows a welcome information plus a "SQL>" prompt.

### A.2.2 Setting Apache Ant

Apache-Ant is in binary model and can be used directly. The path should be set properly, so the ant can be run directly. It is suggested to put it in .bashrc or .cshrc file.

### A.2.3 Setting Tomcat

To start Tomcat, cd to $HOME/apache-tomcat-6.0.16/bin, and type $>$ chmod +x *.sh to give the execution privileges to shell scripts. It is important to edit $HOME/apache-tomcat-6.0.16/conf/server.xml file, change the line: "$<$Connector port="8080" protocol="HTTP/1.1" ", using 8180 port instead, as 8080 is already used by Oracle. Then $HOME/apache-tomcat-6.0.16/bin/startup.sh to start the tomcat web server; shutdown.sh could shutdown the web server.

After these done, the procedure can be continued to deploying step.

## A.3 Deploying OGSA-DAI

### A.3.1 Building OGSA-DAI binary file

Go to $HOME/ogsadai-3.0-gt-4.0.5-src, run the command:
ant -Dgt.dir=$HOME/ws-core-4.0.5
This will take a some time, about 2 minutes on a 512M RAM Pentium 4 Linux. This build ogsa-dai to "build/ogsadai-3.0-gt-4.0.5-bin", and **all following work is done at this directory unless noted**.

### A.3.2 Deploying OGSA-DAI into Tomcat container

Set the GLOBUS_LOCATION environment to $HOME/ws-core-4.0.5 and go to $HOME/ws-core-4.0.5[1], run the command:
ant -f share/globus_wsrf_common/tomcat/tomcat.xml deployTomcat -Dtomcat.dir=$HOME/apache-tomcat-6.0.16
go back to $HOME/ogsadai-3.0-gt-4.0.5-src/build/ogsadai-3.0-gt-4.0.5-bin, run the command:
ant -Dgt.dir=$HOME/ws-core-4.0.5 -Dtomcat.dir=$HOME/apache-tomcat-6.0.16 buildAndDeployGARTomcat
This two command deploy OGSA-DAI into the tomcat.

[1] Set it in .bashrc file would make life easier

### A.3.3 Restarting Tomcat to make changes apply

As mentioned before in section A.2.3, use the script to shutdown the tomcat and start it again.

After this step, the OGSA-DAI server is ready to use, the following things are mostly about how to connect this server to a database.

## A.4 Configuring OGSA-DAI to connect Oracle

### A.4.1 Creating the Oracle test database using OGSA-DAI

- OGSA-DAI use JDBC to connect to the database, so JDBC from Oracle is needed. It can be downloaded from: http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/htdocs/jdbc_10201.html Put it in $HOME/ogsadai-3.0-gt-4.0.5-src/build/ogsadai-3.0-gt-4.0.5-bin/thirdparty/lib/

- Another thing is set the class path for the OGSA-DAI, go to $HOME/ogsadai-3.0-gt-4.0.5-src/build/ogsadai-3.0-gt-4.0.5-bin/, run $>$ source setenv.sh to set the necessary class path.

After these preparation, run “java uk.org.ogsadai.dbcreate.CreateTestOracleDB -database XE -username SYSTEM -password [password set when installing, see section A.1.5] ”. *NOTE, the manual use uk.or.ogsadai.client.dbcreate.CreateTestOracleDB, which is not right.* This command will create a table whose name is “littleblackbook” and add 10000 entries. Seeing “Test database created successfully!” means the OGSA-DAI can be used to connect to Oracle directly. [2] [3]

### A.4.2 Configuring Oracle Data Resource file for OGSA-DAI in Tomcat container

- First check the URL http://localhost:8180/wsrf/dai-manager.jsp in the browser, it should show somw information of OGSA-DAI, which means the OGSA-DAI is successfully deployed into tomcat.

[2] If the database is in another machine, or does not use the default value, the arguments should be changed, please refer to the OGSA-DAI user guide

[3] eXpress Edition ox the Oracle only has one database and its name is XE!

- **Create the data resource file:**
  Create a file to contain the Oracle database information, oracle.prop, which has the content as below:

```
dai.resource.id=OracleDataResource
dai.db.product=Oracle
dai.db.vendor=Oracle
dai.db.version=10
dai.db.uri=jdbc:oracle:thin:@localhost:1521:XE
dai.db.driver=oracle.jdbc.driver.OracleDriver
dai.resource.jar.dir=
dai.dn=ANY
dai.user=SYSTEM
dai.password=[password]
```

  The important part is the uri, which indicate host name and database name. dn is also a trouble, set ANY will suit most situation.

  After finish this property file, run "> ant -Dtomcat.dir=$HOME/apache-tomcat-6.0.16/ -Dgt.dir=$HOME/ws-core-4.0.5/ -propertyfile oracle.prop deployRelationalResource". With this command, a login.txt file is created in $HOME/apache-tomcat-6.0.16/wsrf/WEB-INF/etc/dai/, and a OracleDataResource file is created $HOME/apache-tomcat-6.0.16/wsrf/WEB-INF/etc/dai/resources/. These two make the OGSA-DAI can connect to database through tomcat.

- At last run the command to select some information from littleblackbook, which we have created already.
  java uk.org.ogsadai.toolkit.example.SQLClient -u http://localhost:8180/wsrf/services/dai -d OracleDataResource -q "select * from littleblackbook where id < 3"
  A successful execution would give information, while problem will cause a exception or show a failure notification.

## A.5 OGSA-DAI Client

- Use server proxy class.

- Use data request execution service/resource.

- Use pipeline, adding toolkit activity.

## A.6 Deploying a activity into OGSA-DAI

- Write a suitable activity program, there is no restriction on the name, but take ca re of the ”extends” and ”implements”.
- Make this activity and all its depend classes, excluding the one already in OGSA-DAI, to a jar file.
- Using ”ant deployActivity” to deploy the class into tomcat container. Make sure to indicate the full class name, like uk.org.ogsadai.activity.replication.ConnectDB.
- Edit the data resouce file in the Tomcat, making it belong to the ACTIVITIES showed in the file. This give resource the privileges to use this activity.
- Restart Tomcat, things done. Remember to put client toolkit in the path of the client.

## A.7 Useful Script when doing materialized view replication

```
java uk.org.ogsadai.dbcreate.CreateTestOracleDB -host nesc.homeunix.net -database X

alter table master add primary key(id);
create materialized view log on master;

create materialized view replica refresh fast as select * from master@nesc;

select systimestamp from dual;
delete master where id > 1;
select systimestamp from dual;
commit;

select systimestamp from dual;
execute dbms_mview.refresh('replica');
```

```
select systimestamp from dual;
drop materialized view replica;
```

# Bibliography


[1] Y. Chen, D. Berry, and P. Dantressangle, “Transaction-Based Grid Database Replication.”

[2] A. Domenici, F. Donno, G. Pucciani, and H. Stockinger, “Relaxed data consistency with CONStanza,” *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, vol. 1, pp. 5 pp.–, May 2006.

[3] R. Urbano, *Oracle Database Advanced Replication, 10g Release 2*, Oracle, Nov 2007.

[4] A.Chervenak, R. Schuler, C. Kesselman, S. Koranda, and B. Moe, “Wide Area Data Replication for Scientific Collaborations.” 6th IEEE/ACM International Workshop on Grid Computing, 2005.

[5] P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger, “Data Grid Replica Management with Reptor.” 5th international conference on parallel processing and applied mathmatics, 2003.

[6] L. Liming, *Large-Scale Data Replication for LIGO*, ClusterWorld, July 2005.

[7] C. Baru, R. Moore, A. Rajasekar, and M. Wan, “The SDSC Storage Resource Broker.” Toronto, Canada: CASCON’98 Conference, 1998.

[8] J. D. Zawodny and D. J. Balling, *High Performance MySQL: Optimization, Backups, Replication & Load Balancing*, 2004.

[9] M. Cavalleri, R. Prudentino, U. Pozzoli, and G. Reni, “A set of tools for building postgreSQL distributed databases in biomedical environment,” *Engineering in Medicine and Biology Society, 2000. Proceedings of the 22nd Annual International Conference of the IEEE*, vol. 1, pp. 540–544 vol.1, 2000.

[10] L. Gu, L. Budd, A. Cayci, C. Hendricks, M. Purnell, and C. Rigdon, *A Practical Guide to DB2 UDB Data Replication V8*. IBM RedBooks.

[11] A. J. Ciccone and B. Hamel, “The Next Generation: Q Replication,” *DB2 Magazine*, vol. 9, no. 3, quarter 4 2004.

[12] R. Sharma, *Miscrosoft SQL Server 2000: A Guide to Enhancements and new Features*, 2004.

[13] (2007, Nov.) The globus alliance. University of Chicago. [Online]. Available: http://www.globus.org/

[14] M. A. et al., “The Design and Implementation of Grid Database Services in OGSA-DAI,” *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 357–376, February 2005.

[15] “Database Language SQL,” International Organization for Standardization, Jul 1992.

[16] *OGSA-DAI 3.0 Documentation*, The University of Edinburgh, 2007. [Online]. Available: http://www.ogsadai.org/documentation/ogsadai3.0/ogsadai3.0-gt/