ATL-
23 July

Evaluation of the Relational Implementation of the Conditions Database Interface

Author: A.Amorim, N.Barros, D.Klose, J.Lima, C.Oliveira, L.Pedro
Date: February 2003

Abstract

The ConditionsDB interface is a C++ class library, built on top of a database management system, to enable storage and retrieval of detector-related information with an associated period of validity.

This document describes a first comparison study between the Oracle v0.4.1.6 and the MySQL v0.2.6b implementations of the ConditionsDB interface. A set of performance tests were executed both running locally in the database server machine and using clients in remote machines.

It also addresses the issues related to the Large Scale Tests performed using the MySQL prototype.

The necessary steps to build and install both implementations are also described.

Contents

List of Figures

1 Part I - Comparison between MySQL and Oracle implementations

1.1 Introduction

The purpose of this work is to evaluate the performance of the Oracle and MySQL implementations of Conditions Database (ConditionsDB) as they are implemented at the present moment and to suggest paths for improvement for both implementations.

The comparison was carried out using both the examples distributed with the Oracle implementation, that were also used, with the appropriate modifications, on the MySQL implementation and implementing wew intensive usage examples that were created and applied to both implementations. These new applications, developed to evaluate the intensive usage of the ConditionsDB include not only creating complex foldersets and folder based structures but also storing and retrieving large number of objects.

The time spent running each one of this examples was measured and provided the source for the comparison results.

1.2 Setting up the system

Both implementations can be found at standard CERN AFS locations and both bring documentation on how to correctly set up the system in order to achieve compilation without errors.

An Oracle or a MySQL server must be available to support the databases and the client API's libraries must also be installed in order to correctly build the implementations. Both setup procedures are described in detail in Appendix A. Installing the ORACLE version the Oracle 9i, Release 2 on Suse Linux was rather straighforeward once the respective Suse rpm was used. This is not the case for release 1 where several ad hoc operations have to be performed. Information on how to instal this previous release is available from the authors.

1.3 Running examples

All the tests were performed using a database server in a Linux PC with the following configuration:

- Hardware:
  - Intel Pentium 4, 1,6GHZ
  - 756Mb DDR RAM PC133MHz
  - 30 Gb HDD, ATA100, 7200RPM

- Operating Sytem:
  - Suse 8.0, linux kernel 2.4.18-64GB-SMP
- Database Servers:
  - Oracle: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production With the Partitioning, OLAP and Oracle Data Mining options JServer Release 9.2.0.1.0 - Production
  - MySQL: Distrib 3.23.48

Below is the list of all the examples used on the tests, as well as a brief description of the different steps involved. Some benchmarks were adapted from the examples distributed in the 0.4.1.6 distribution of the Oracle implementation that were also migrated to the MySQL implementation. Due to the abstract interface design of the ConditionsDb interface, the migration to MySQL is easily accomplished with minor changes in the code.

From the common code:

#include <ConditionsDB/CondDBXXXXMgrFactory.h>
...
ICondDBMgr CondDBmgr = CondDBXXXX MgrFactory::createCondDBMgr();*
...
CondDBmgr->init();
...
CondDBXXXXMgrFactory::destroyCondDBMgr(CondDBmgr);

XXXX must be changed for the specific implementation. For MySQL it must be changed to MySQL and for Oracle to OracleDB.

1.3.1 Description of the tests

- *createFolders* - Connects to the database server. Verifyes if a folderset/folder structure *(cal/temp)* exists. Should not exist, creates it.
- *exampleObject* and *storeData* - Stores an object under a given folder with defined characteristics, such as *time validity, insertionTime, layer* and *data*. *exampleObject* creates one object and *storeData* creates three objects with different time validity.
- *exampleObjectRead* and *readData* - Reads an object already stored and prints out all it's properties.
- *comprehensiveTest* - Extends the functionality of all the procedures (*NOTE* - this test was not migrated to MySQL).

To measure the time spent by each test, the time Unix function was used by preceeding the command by "time". We have recorded the *real time*, the *user time* and *system time*. From the manual page of the function comes the definitions: *real time - the elapsed real time between invocation and termination, user time - the user CPU time, system time - the system CPU time.*

The real time for each test is listed bellow. The chosen value includes the time spent on network transport. All tests were done in dedicated machines that were performing only this task. Appendix B contains the detailed test results for real, user and system time.

- **MySQL Results**

	MySQL (local)	*MySQL (remote)*
createFolders	real 0m0.134s	real 0m0.133s
storeData	real 0m0.065s	real 0m0.065s
readData	real 0m0.023s	real 0m0.046s
exampleObject	real 0m0.038s	real 0m0.060s
exampleObjectRead	real 0m0.021s	real 0m0.053s

Figure 1: *time* values for the MySQL implementation of ConditionsDB (local and remote)

- **Oracle Results**

	Oracle (local)	*Oracle (remote)*
CreateFolders	real 0m11.101s	real 0m23.086s
storeData	real 0m0.427s	real 0m0.633s
readData	real 0m0.130s	real 0m0.228s
exampleObject	real 0m0.233s	real 0m0.293s
exampleObjectRead	real 0m0.131s	real 0m0.248s
comprehensiveTest	real 6m55.043s	real 7m42.640s

Figure 2: *time* values for the Oracle implementation of ConditionsDB (local and remote)

1.4 Intensive Usage

The following tests were implemented in order to test the different implementations in it's functionalities and to see it's behaviour under heavy load, creating the database structures and performing intensive storage and reading procedures. The tests are based on the examples distributed with the Oracle implementation.

- *createFolderx* - Connects to a given database in the database server. If none exists, creates a defined structure associated with the three levels of foldersets and a folder.
- *storeDatax* - Connects to the database server. Verifies the existence of a defined folder. If it does not exist, it is created. The user is prompted for the number of objects to be stored. All objects will be stored using the same *time validity, insertionTime*, and *data*, but each will be in a different *layer*.
- *readDatax* - Connects to the database server. Looks into a defined folder and sees if it's not empty. Gets all the objects stored in that folder and iterates over all, showing their values.

For the *storeDatax* test, diferent quantities of objects were stored in order to study the relation between objects stored vs. time spent storing and between objects stored vs. time spent reading.

Intensive Usage		***MySQL (local)***	***MySQL (remote)***
createFolderx		real 0m0.034s	real 0m0.072s
storeDatax	*10 objs*	real 0m1.447s	real 0m1.127s
	100 objs	real 0m1.368s	real 0m2.345s
	1.000 objs	real 0m4.056s	real 0m6.341s
	10.000 objs	real 0m23.175s	real 0m56.929s
	50.000 objs	real 1m40.040s	real 3m53.565s
	100.000 objs	real 3m49.184s	real 8m16.510s
readDatax	*10 objs*	real 0m0.028s	real 0m0.082s
	100 objs	real 0m0.121s	real 0m0.146s
	1.000 objs	real 0m0.312s	real 0m1.084s
	10.000 objs	real 0m3.379s	real 0m11.021s
	50.000 objs	real 0m16.691s	real 0m49.572s
	100.000 objs	real 0m34.258s	real 1m45.335s

Figure 3: *time* values for the MySQL implementation of ConditionsDB with intensive usage tests (local and remote)

Intensive Usage		*Oracle (local)*	*Oracle (remote)*
createFolderx		real 0m15.173s	real 0m11.857s
storeDatax	*10 objs*	real 0m4.973s	real 0m5.434s
	100 objs	real 0m9.749s	real 0m15.820s
	1.000 objs	real 1m2.375s	real 1m2.850s
	10.000 objs	real 9m22.103s	real 11m12.554s
	50.000 objs	real 53m48.330s	real 52m49.003s
	100.000 objs	real 109m40.878s	real 104m13.283s
readDatax	*10 objs*	real 0m0.513s	real 0m0.311s
	100 objs	real 0m1.075s	real 0m1.531s
	1.000 objs	real 0m6.441s	real 0m13.571s
	10.000 objs	real 1m8.759s	real 2m19.243s
	50.000 objs	real 6m57.175s	real 13m12.387s
	100.000 objs	real 12m18.258s	real 30m14.354s

Figure 4: *time* values for the Oracle implementation of ConditionsDB with intensive usage tests (local and remote)

Graphical Representation

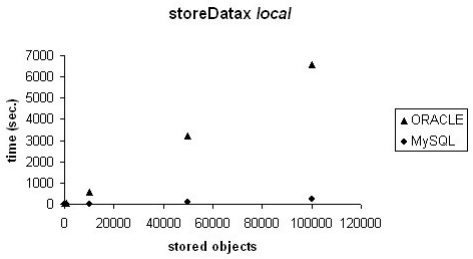

Figure 5: objects stored vs. time spent storing for MySQL and Oracle implementations (local)

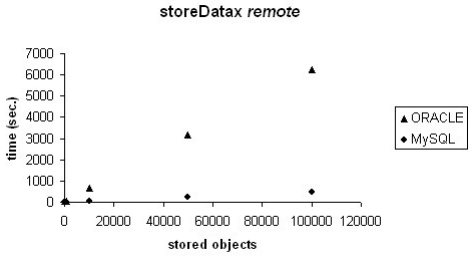

Figure 6: objects stored vs. time spent storing for MySQL and Oracle implementations (remote)

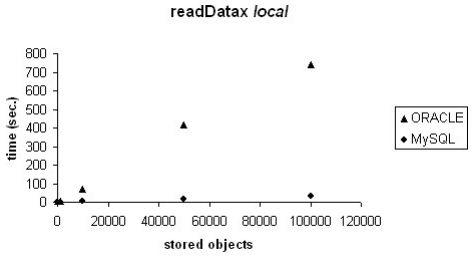

Figure 7: objects stored vs. time spent reading for MySQL and Oracle implementations (local)

Figure 8: objects stored vs. time spent reading for MySQL and Oracle implementations (remote)

2 Part II - Large Scale Tests

2.1 Introduction

For the Large Scale Tests up to 100 PC's were used as clients for the MySQL server using the MySQL ConditionsDB API. The Online Software infraestructure was used to allow synchronisation of the client processes. Each client runs as a controller for the Online Software that activates a certain process on that client everytime a state transition occurs using RunControl. The type of test and the number of objects to be managed were preconfigured on a database server. On the Load to Configure transition, the test configuration occurs connecting to the database server in order to get its parameters such as the type of test (Store or Read test), database names and number of objects. The Configure to Run transition executes the test.

The tests are the same as described in the Intensive Usage chapter. Tests were performed by using different values of objects to be stored and read from the database and different numbers of clients accessing it. Due to implementation limitations, each client stores the objects in one database. At the moment it is not possible for several clients to store the objects on one database at the same time. Although it is possible for all the clients to read the objects from the same database.

Each client gets its start and end time for its test process. The time results presented are the average value of its duration.

2.2 System Setup

- **Client Details:**
  - Intel PIII 1GHz dual processor
  - Linux 2.4.18-18.7.x.cernsmp

  - 376 MByte RAM
  - On-Line Software release 00-18-01
  - gcc 2.96 compiler
  - Gigabit network between the clients

- **Server Details**
  - Intel PIV 2GHz
  - Linux 2.4.18
  - 1 GByte RAM
  - cc 2.95.4 compiler
  - MySQL Distrib 3.23

2.3 Large Scale Tests Results

- **Storing objects on diferent databases**

objects	10 controllers	20 controllers	50 controllers	80 controllers	100 controllers
100	0:00:03	0:00:06	0:00:16	0:00:37	0:00:49
1000	0:00:28	0:01:02	0:03:22	0:08:47	0:15:06
10000	0:04:43	0:11:21		1:50:44	3:04:40
100000	0:49:01	1:55:15			

Figure 9: Objects stored vs. time spent - fixating the number of controllers

controllers	100 objects	1.000 objects	10.000 objects	100.000 objects
10	0:00:03	0:00:28	0:04:43	0:49:01
20	0:00:06	0:01:02	0:11:21	1:55:15
50	0:00:16	0:03:22		
80	0:00:37	0:08:47	1:50:44	
100	0:00:49	0:15:06	3:04:40	

Figure 10: Number of controllers vs. time spent - fixating the number of objects stored

- **Reading objects from diferent databases**

objects	10 controllers	20 controllers	50 controllers	80 controllers	100 controllers
100	0:00:00	0:00:01	0:00:03	0:00:05	0:00:07
1000	0:00:06	0:00:12	0:00:30	0:00:47	0:00:59
10000	0:00:55	0:02:01		0:08:01	0:10:03
100000	0:09:11	0:19:25			

Figure 11: Objects stored vs. time spent - fixating the number of controllers

controllers	100 objects	1.000 objects	10.000 objects	100.000 objects
10	0:00:00	0:00:06	0:00:55	0:09:11
20	0:00:01	0:00:12	0:02:01	0:19:25
50	0:00:03	0:00:30		
80	0:00:05	0:00:47	0:08:01	
100	0:00:07	0:00:59	0:10:03	

Figure 12: Number of controllers vs. time spent - fixating the number of objects stored

- **Reading objects from same database**

objects	10 controllers	50 controllers
100	0:00:00	0:00:03
1000	0:00:06	0:00:31
10000	0:00:54	0:05:02
100000	0:09:13	0:51:17

Figure 13: Objects stored vs. time spent - fixating the number of controllers

-

controllers	100 objects	1.000 objects	10.000 objects	100.000 objects
10	0:00:00	0:00:06	0:00:54	0:09:13
50	0:00:03	0:00:31	0:05:02	0:51:17

Figure 14: Number of controllers vs. time spent - fixating the number of objects stored

3 Conclusions

At the present moment MySQL implementation shows better performance in all tests.

Both implementations show time values that grow linearly with the number of objects stored or retrieved.

On possible factor that will be investigated in the future is the use of more indexed fields in the MySQL implementation. This feature can bring even better results while selecting and reading objects but might have a negative impact on the time spent while storing the data.

Setting up the system for the MySQL implementation seems much simpler and has less hardware requirements. The Oracle DBMS on the other hand has many powerful features and includes many assistants that should help to bring the most benefit from the whole system.

Large Scale Tests

The time spent results for the *readDatax* test with one database per client fixating the number of objects shows a exponential growth in function of the number of clients making it unbearable to support for clients above 100. The clients access should be of more concern on the implementation future development.

For all other combinantions a linear growth is showned.

4 Appendix A - Installing, setting up and running the MySQL and ORACLE servers, compiling and using the ConditionsDB API on Linux

4.1 Using MySQL

The MySQL source code can be obtained at http://www.mysql.com as well as documentation on how to install and use MySQL. The MySQL_max server should be installed since it includes extended funcionalties. The *mysql_install_db* script must be invoked in order to create the necessary structure to start MySQL. To protect the MySQL root user with a password, issue the command:

mysqladmin -u root password <password>

The next step is creating a new user:

mysql -u root -p mysql
Enter password:
mysql>*grant all privileges on *.* to <username> @'localhost' identified by '<password>' with grant option;*
mysql>*grant all privileges on *.* to <username> @"%" identified by '<password>' with grant option;*

This will create a very privileged user who can connect to the database server either locally or from remote machines.

To enter in the MySQL command line client environment one can use the following client application:

mysql -u <user> -p <password> -h <hostname>

The following command lists all databases inside MySQL:

mysql>*show databases;*

To use a specific database one can:

mysql>*use <database name>;*

To navigate trough the database structure one can use SQL commands like *select * from <table name>;*

Another usefull command that allows erasing a database:

mysql>*drop database <database name>;*

- *Using the ConditionsDB MySQL implementation:*

The abstract interface of the ConditionsDB allows the user code to become almost independent of the implementation. The only modifications are in the init() method that is used to establish a connection to the MySQL server. The input string, in this case, takes the form:

$condDBmgr- > init(" < host >:< database\_name >:< user >:< password > ")$;

4.2 Using ORACLE

Oracle 9i, Release 2, was used. Due to some dependencies on the specific linux platform, the instalation of Oracle 9i R1 in Suse Linux 8.0 is not trivial. However, once installed, it works correctly together with the ConditionsDB implementation.

The installation problems in Oracle 9i R1 distribution were fixed in R2 release and for this reason this last version was used to perform the tests. The installation under Suse 8.0 follows the standard ORACLE installation procedure but a package released by Suse support on Oracle, *orarun9i.rpm* had to be installed to set up the correct kernel parameters and several environment variables that are later used by the instalation procedure.

During installation, the following installation options must be made:

- Available Products -> Oracle 9i Database 9.2.0.1.0
- Installation Type -> Enterprise Edition
- Database Configuration -> General Purpose

All this options enable the Partitioning feature that is required for the process of creating the support database structure for ConditionsDB.

To start using a database, the database administrator must perform this some steps. It's necessary to start up and mount the database that will be used. Perform:

user@machine >export ORACLE_SID=<database name that will be used>
user@machine>oraenv
ORACLE_SID = [database name] ? (press Enter or insert database name again)
user@machine>sqlplus /nolog
*SQL*Plus>connect system as sysdba*
password:
*SQL*Plus>startup*

For unmounting and shuting down a database, do *shutdown* instead of *startup*.

In order to use the ConditionsDB implementation with Oracle, a user must be set. This user must have certain privileges. To create a user perform:

user@machine>sqlplus /nolog
*SQL*Plus>connect system as sysdba*
*SQL*Plus>password:*
*SQL*Plus>create user <username> identified by <password>;*
*SQL*Plus>grant connect to <username>*
*SQL*Plus>grant ALL PRIVILEGES to <username>*

To have access and navigate through all the structure created by ConditionsDB, a SQL*Plus console can be used. Perform:

user@machine>sqlplus <user>/<password> @ <database name>

From SQP*Plus console it's possible to navigate through all the tables issuing SQL commands.

Dropping the ConditionsDB is not at the moment a code feature. It must be performed using an external SQL script that uses all the necessary SQL commands to automatically erase all the structure. This script named *dropCondDB.sql* comes with the Oracle implementation under the path *implementationOracle/sql*. To launch it, perform:

user@machine>sqlplus <username> / <password>@<database name>

*SQL*Plus>start /<path>/dropCondDB.sql*

The script asks for two values. The first one is the username and the second is the ConditionsDB database name that was passed on the init() method. This operation is irreversible and permanently deletes all the structure and data stored in the ConditionsDB database chosen.

To see the ConditionsDB database names that are already created, perform:

*SQL*Plus>select * from condition_dbs;*

Another way of dropping the ConditionsDB database is to change the value of the *STATUS* field for the chosen database from 0 to 1 on the *condition_dbs* table. This is not the same as erasing the database with the *dropCondDB* script. Changing this value only makes the implementation rewrite the structure like no one was created. For changing the value, perform:

*SQL*Plus>update condition_dbs set status=1 where status=0;*

To prepare the server in order to accept connections from clients, it's necessary to start a *listener*. This is a process that runs on the server side and whose function is to listen for incoming client connection requests and manage the traffic to the server. To configure a listener the *Oracle Net Manager* tool may be used. Launch the tool by issuing *netmgr*. Choose *Oracle Net Configuration -¿ Local -¿ Listeners* and then *Edit -¿ Create*. Give the *listener* a name. Press *Add Address* and in the *Listening Locations* verify if the default values for *Protocol, Host* and *Port* are correct. Under *Database Services* press *Add Database* and then modify the values for *Global Database Name, Oracle Home Directory* and *SID*. Those values will identify the databases available for the remote connections and the *Global Database Name* must be passed on the SQL*Plus connect command in the client as it was defined. To start the *listener* perform:

oracle@machine>lsnrctl start

For stopping the *listener* use *stop* instead of *start*.

All the actions listed for local server are also valid for the client. Even though it's a client installation, on the *Available Products* window, the *Oracle 9i Database 9.2.0.1.0* option must be chosen and not

Oracle 9i Client 9.2.0.1.0 because this option does not install the all Oracle files that are necessary for the implementation to compile. In *Database Configuration* window, *Software Only* option may be chosen since no database is needed on the client. The *listener* doesn't need to be configured and started since the client don't receive connections for services but a *Net Service Name* does. This is necessary in order for the client to identify the Oracle service to access on the server. To establish this configuration, the *Oracle Net Configuration Assistant* tool may be used. Launch the tool by issuing *netca*. On the first window choose *Local Net Service Name Configuration*. Next window choose *Add a Net Service Name*. Next Window choose for which Oracle version it should be. On the next window provide the service name you want to access.

Normally it should be the *Global Database Name* for the database that will be accessed in the form *database-name.domain*. Next choose the appropriate protocol and finally the hostname of the server and if the standard port 1521 should be used or another. Give the *Net Service Name* a name and finish the Assistant. This tool writes a *tnsnames.ora* simple text file under the path stored on *$TNS_ADMIN* variable (usually */opt/oracle/product/901/network/admin* if the default options were used). This file stores all the information given and again can alternatively be modified to our convenience instead of using the *Oracle Net Configuration Assistant* tool.

The server can then be accessed using SQL*Plus:

user@machine>sqlplus <username>/<password>@<Global Database Name>

- *Using the ConditionsDB Oracle implementation:*

In the implementation code, the init() method is used in order to establish a connection to the Oracle server. The syntax is the form:

condDBmgr->init("user=<username>,passwd=<passwd>,db=<Oracle_database_name>,cond

The *db* input string should be the Oracle database name (specified in *$ORACLE_SID* variable) for the local connection and the *Global Database Name* for the remote connection. *cond_db* input string is a general name for the ConditionsDB structure.

5 Appendix B - Detailed test results for the MySQL and Oracle implementation of ConditionsDB

5.1 MySQL

	MySQL (local)	*MySQL (remote)*
createFolders	real 0m0.134s user 0m0.010s sys 0m0.010s	real 0m0.133s user 0m0.020s sys 0m0.000s
storeData	real 0m0.065s user 0m0.010s sys 0m0.000s	real 0m0.065s user 0m0.020s sys 0m0.000s
readData	real 0m0.023s user 0m0.020s sys 0m0.000s	real 0m0.046s user 0m0.020s sys 0m0.000s
exampleObject	real 0m0.038s user 0m0.020s sys 0m0.000s	real 0m0.060s user 0m0.020s sys 0m0.000s
exampleObjectRead	real 0m0.021s user 0m0.010s sys 0m0.010s	real 0m0.053s user 0m0.010s sys 0m0.000s

Figure 15: MySQL implementation: local / remote

IntensiveUsage		*MySQL (local)*	*MySQL (remote)*
createFolderx		real 0m0.034s user 0m0.010s sys 0m0.010s	real 0m0.072s user 0m0.020s sys 0m0.000s
storeDatax	*10 objs*	real 0m1.447s user 0m0.010s sys 0m0.010s	real 0m1.127s user 0m0.020s sys 0m0.010s
	100 objs	real 0m1.368s user 0m0.050s sys 0m0.050s	real 0m2.345s user 0m0.030s sys 0m0.040s
	1.000 objs	real 0m4.056s user 0m0.320s sys 0m0.260s	real 0m6.341s user 0m0.170s sys 0m0.150s
	10.000 objs	real 0m23.175s user 0m2.390s sys 0m2.270s	real 0m56.929s user 0m1.910s sys 0m1.800s
	50.000 objs	real 1m40.040s user 0m12.700s sys 0m11.480s	real 3m53.565s user 0m9.150s sys 0m8.020s
	100.000 objs	real 3m49.184s user 0m26.970s sys 0m24.240s	real 8m16.510s user 0m17.880s sys 0m16.760s

Figure 16: MySQL implementation: local / remote - Intensive Usage - *storeDatax*

IntensiveUsage		*MySQL(local)*	*MySQL(remote)*
readDatax	*10 objs*	real 0m0.028s	real 0m0.082s
		user 0m0.020s	user 0m0.010s
		sys 0m0.000s	sys 0m0.010s
	100 objs	real 0m0.121s	real 0m0.146s
		user 0m0.020s	user 0m0.030s
		sys 0m0.010s	sys 0m0.010s
	1.000 objs	real 0m0.312s	real 0m1.084s
		user 0m0.080s	user 0m0.170s
		sys 0m0.070s	sys 0m0.050s
	10.000 objs	real 0m3.379s	real 0m11.021s
		user 0m0.760s	user 0m1.110s
		sys 0m0.680s	sys 0m0.430s
	50.000 objs	real 0m16.691s	real 0m49.572s
		user 0m3.870s	user 0m5.420s
		sys 0m2.840s	sys 0m1.810s
	100.000 objs	real 0m34.258s	real 1m45.335s
		user 0m7.670s	user 0m10.130s
		sys 0m5.710s	sys 0m3.270s

Figure 17: MySQL implementation: local / remote - Intensive Usage - *readDatax*

5.2 Oracle

	Oracle (local)	*Oracle (remote)*
createFolders	real 0m11.101s user 0m0.060s sys 0m0.020s	real 0m23.086s user 0m0.090s sys 0m0.110s
storeData	real 0m0.427s user 0m0.030s sys 0m0.010s	real 0m0.633s user 0m0.030s sys 0m0.030s
readData	real 0m0.130s user 0m0.010s sys 0m0.010s	real 0m0.228s user 0m0.050s sys 0m0.010s
exampleObject	real 0m0.233s user 0m0.030s sys 0m0.000s	real 0m0.293s user 0m0.010s sys 0m0.030s
exampleObjectRead	real 0m0.131s user 0m0.030s sys 0m0.020s	real 0m0.248s user 0m0.040s sys 0m0.030s
comprehensiveTest	real 6m55.043s user 0m11.550s sys 0m1.910s	real 7m42.640s user 0m12.640s sys 0m2.280s

Figure 18: Oracle implementation: local / remote

IntensiveUsage		*Oracle (local)*	*Oracle (remote)*
createFolderx		real 0m15.173s user 0m0.020s sys 0m0.020s	real 0m11.857s user 0m0.060s sys 0m0.040s
storeDatax	*10 objs*	real 0m4.973s user 0m0.030s sys 0m0.000s	real 0m5.434s user 0m0.060s sys 0m0.030s
	100 objs	real 0m9.749s user 0m0.140s sys 0m0.030s	real 0m15.820s user 0m0.130s sys 0m0.050s
	1.000 objs	real 1m2.375s user 0m0.830s sys 0m0.120s	real 1m2.850s user 0m1.150s sys 0m0.170s
	10.000 objs	real 9m22.103s user 0m8.470s sys 0m1.510s	real 11m12.554s user 0m10.110s sys 0m1.630s
	50.000 objs	real 53m48.330s user 0m41.140s sys 0m5.960s	real 52m49.003s user 0m48.950s sys 0m7.220s
	100.000 objs	real 109m40.878s user 1m26.100s sys 0m12.610s	real 104m13.283s user 1m35.490s sys 0m15.380s

Figure 19: Oracle implementation: local / remote - Intensive Usage - *storeDatax*

IntensiveUsage		*Oracle (local)*	*Oracle (remote)*
readDatax	*10 objs*	real 0m0.513s	real 0m0.311s
		user 0m0.030s	user 0m0.070s
		sys 0m0.000s	sys 0m0.030s
	100 objs	real 0m1.075s	real 0m1.531s
		user 0m0.280s	user 0m0.270s
		sys 0m0.040s	sys 0m0.070s
	1.000 objs	real 0m6.441s	real 0m13.571s
		user 0m1.890s	user 0m2.050s
		sys 0m0.370s	sys 0m0.380s
	10.000 objs	real 1m8.759s	real 2m19.243s
		user 0m17.040s	user 0m21.570s
		sys 0m2.710s	sys 0m3.710s
	50.000 objs	real 6m57.175s	real 13m12.387s
		user 1m40.250s	user 1m34.300s
		sys 0m19.150s	sys 0m14.500s
	100.000 objs	real 12m18.258s	real 30m14.354s
		user 3m26.630s	user 3m39.840s
		sys 0m38.670s	sys 0m37.030s

Figure 20: Oracle implementation: local / remote - Intensive Usage - *readDatax*

6 Appendix C - Graphical representation of the results for the Large Scale Tests

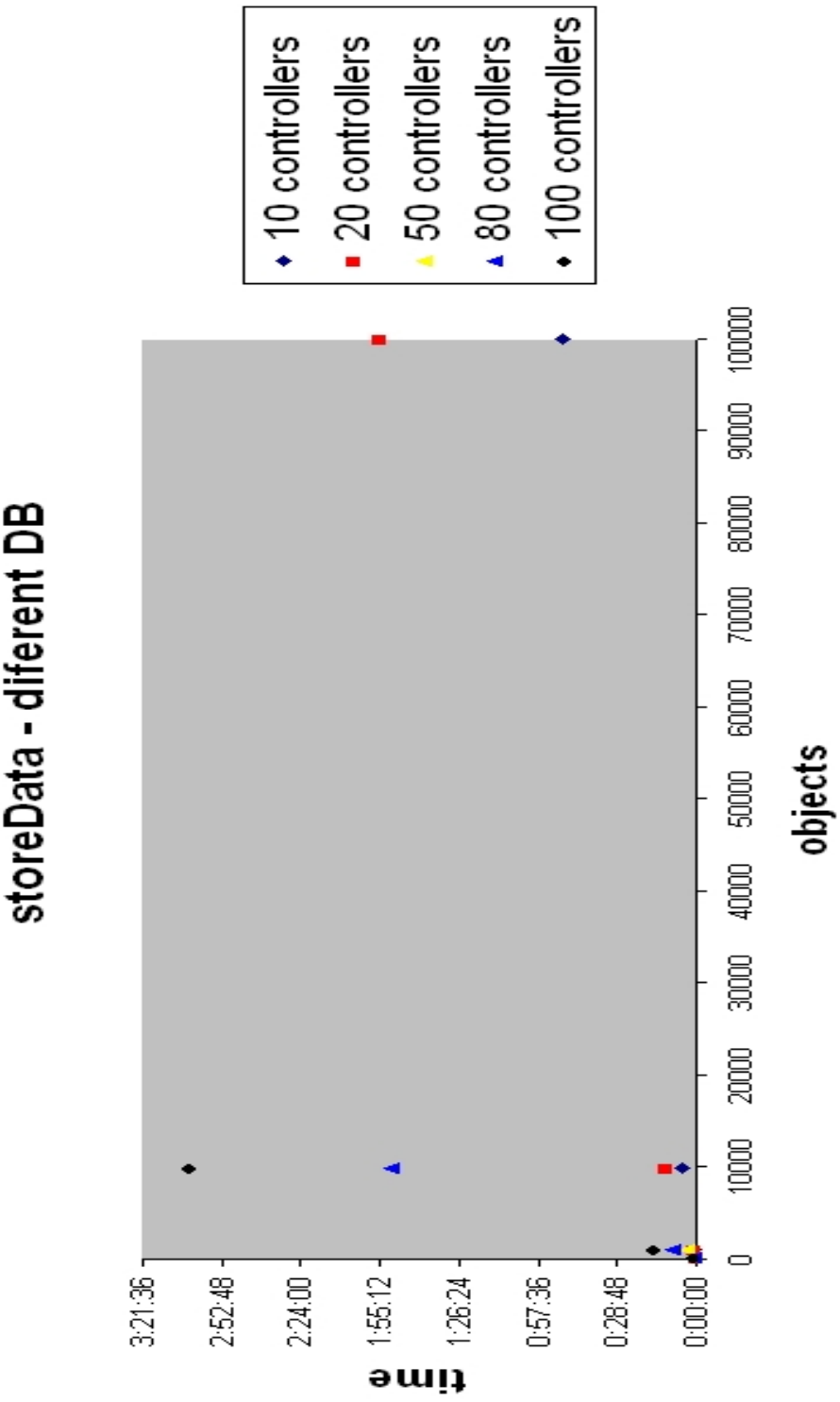

Figure 21: Objects stored vs. time spent - fixating the number of controllers

6 Appendix C - Graphical representation of the results for the Large Scale Tests

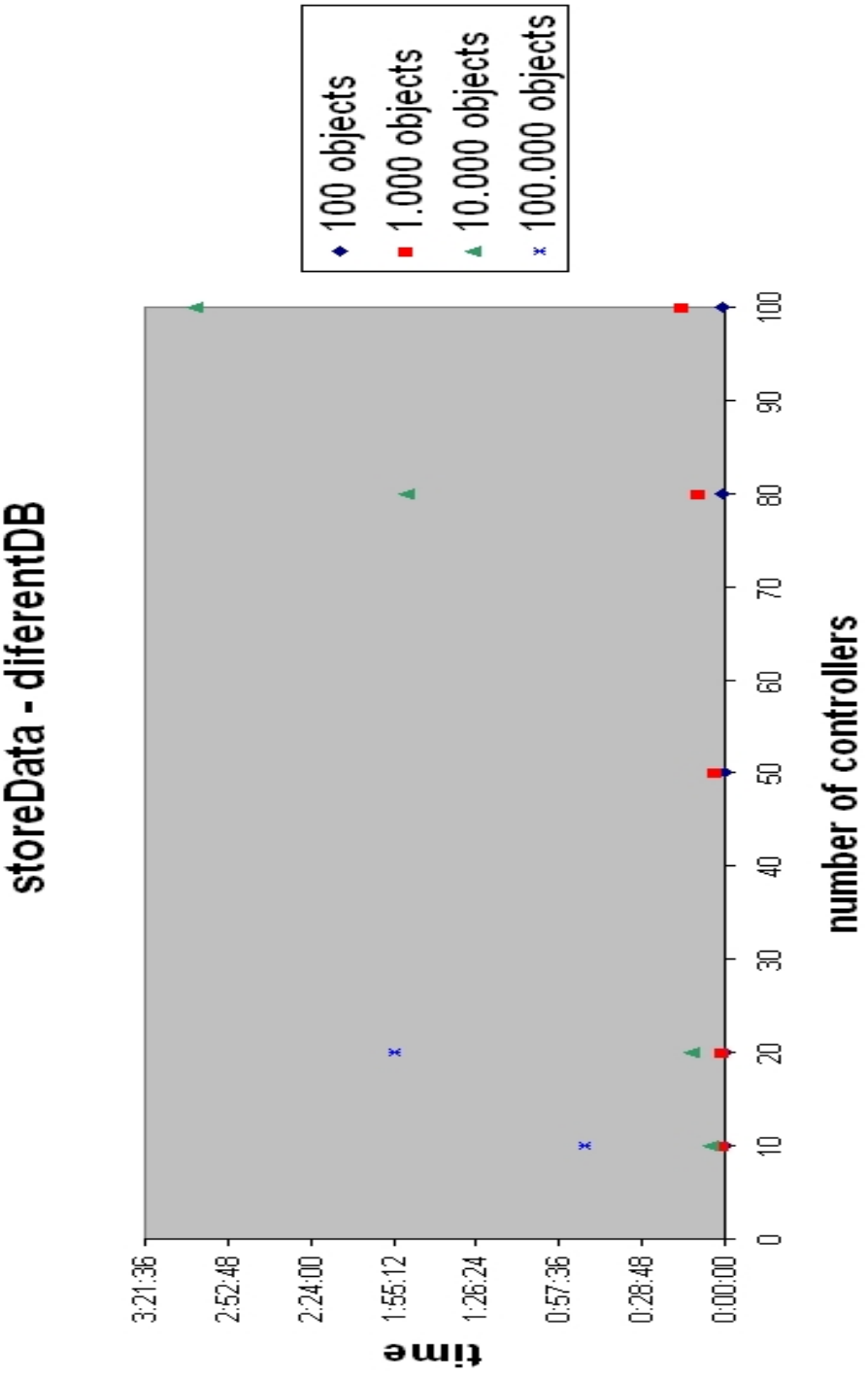

Figure 22: Number of controllers vs. time spent - fixating the number of objects stored

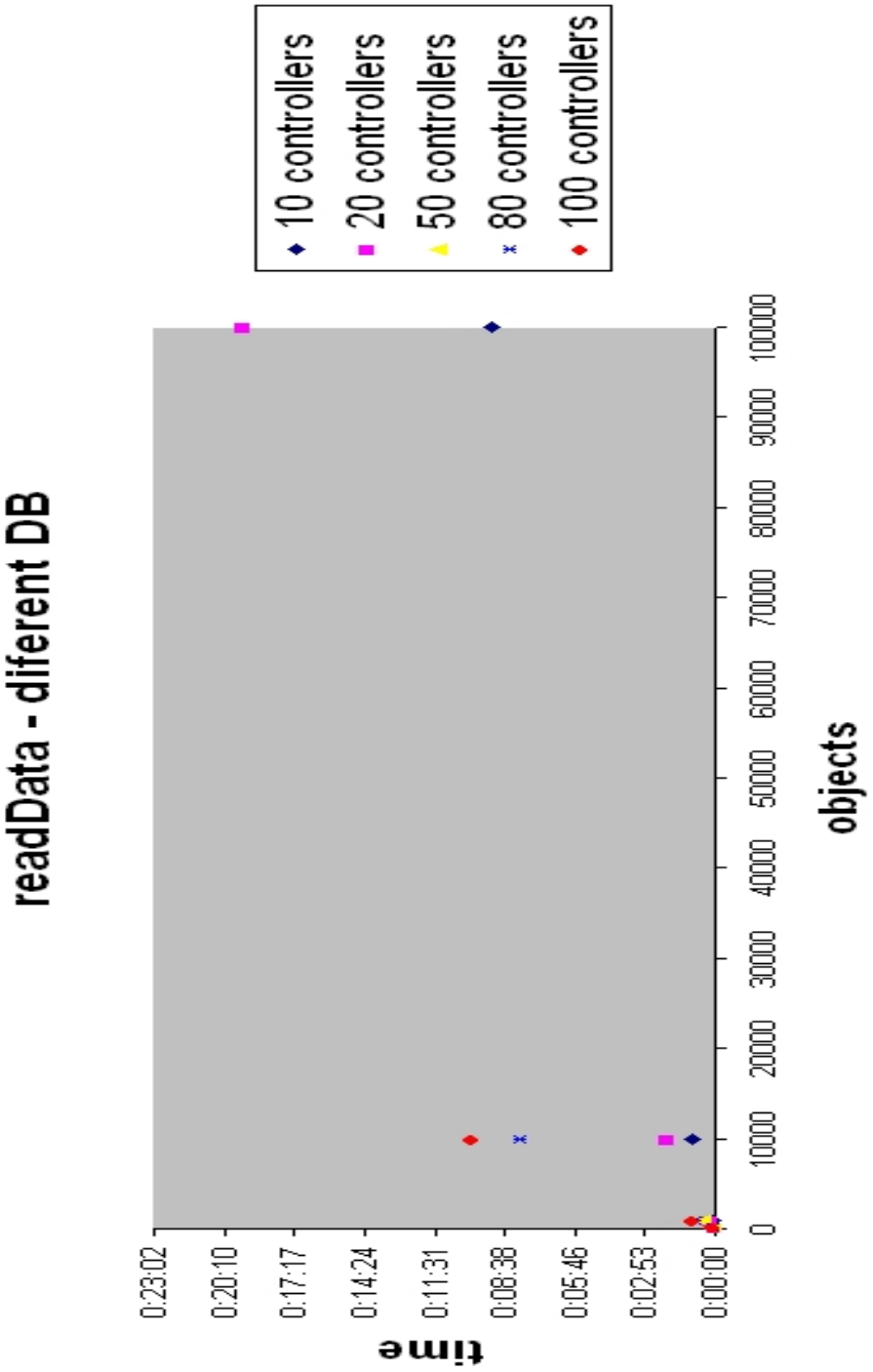

Figure 23: Objects stored vs. time spent - fixating the number of controllers

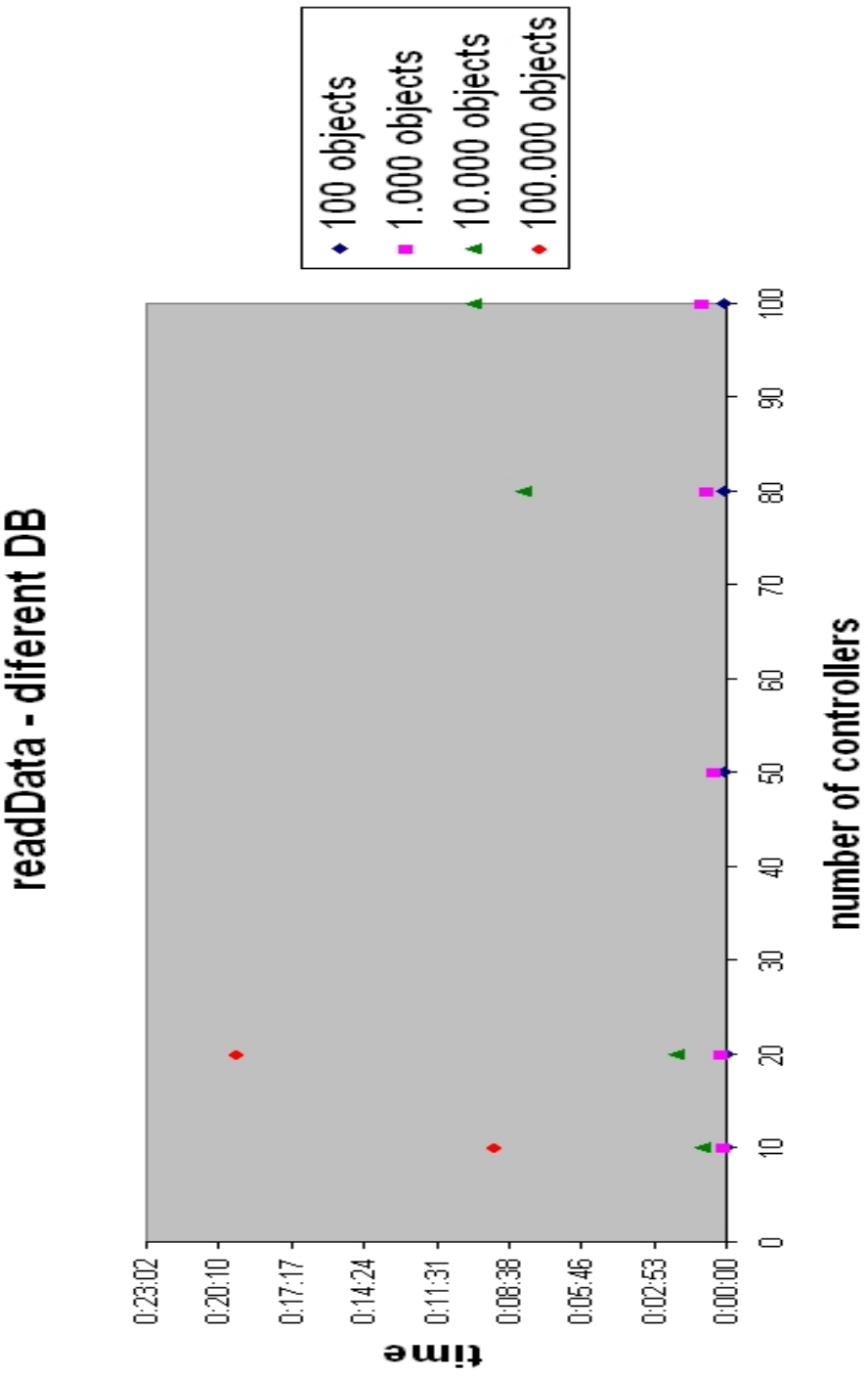

Figure 24: Number of controllers vs. time spent - fixating the number of objects stored

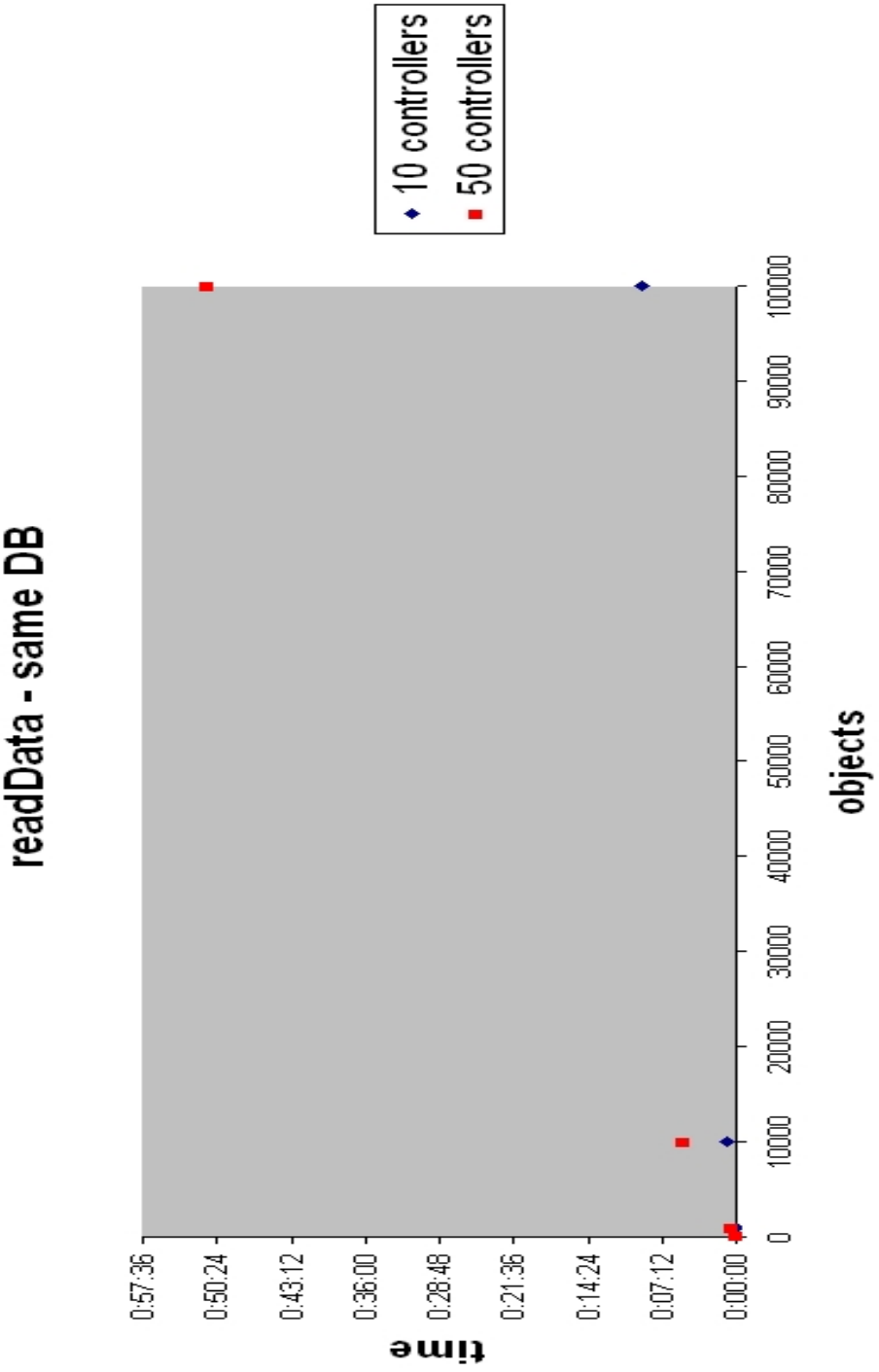

Figure 25: Objects stored vs. time spent - fixating the number of controllers

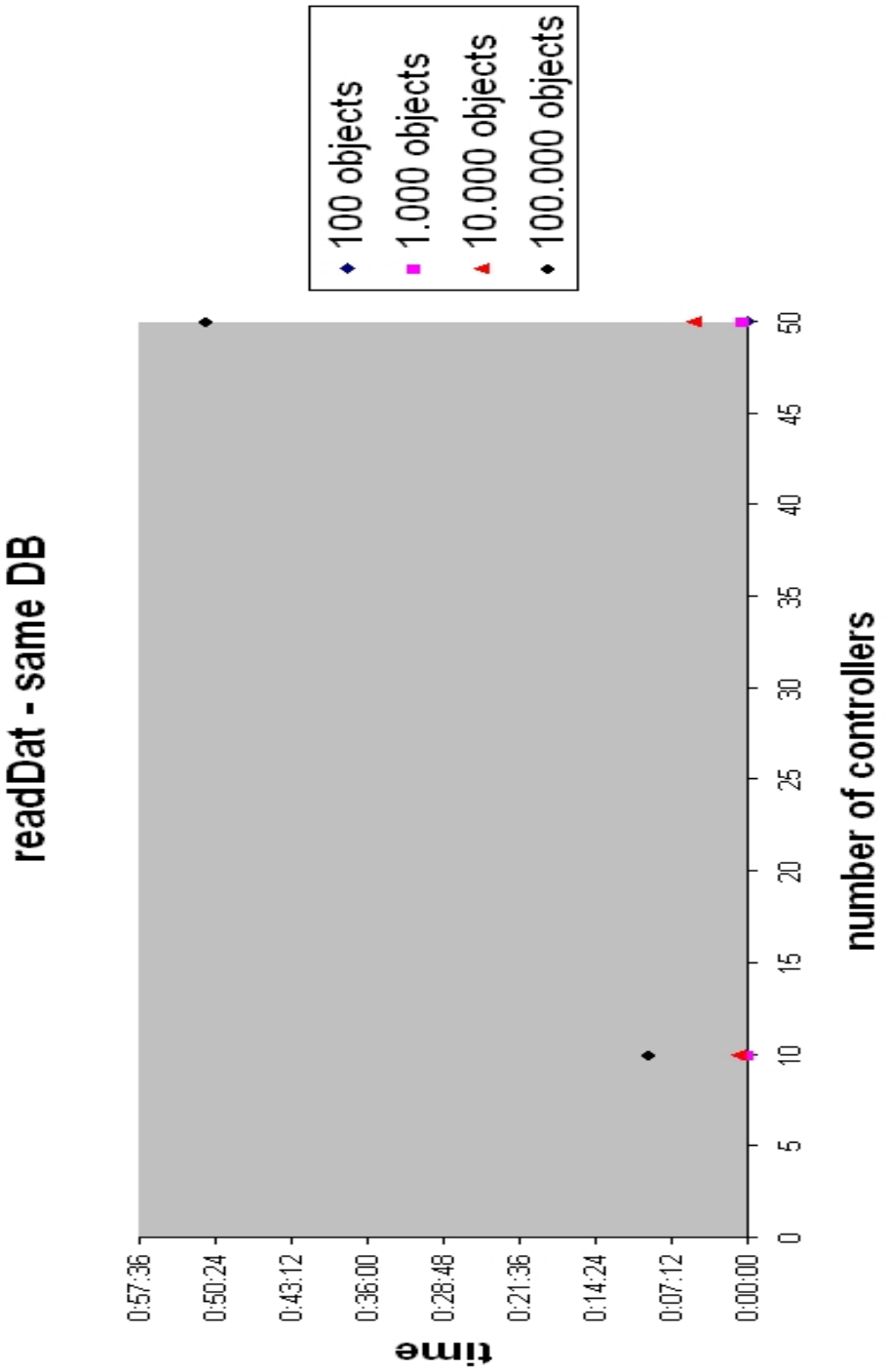

Figure 26: Number of controllers vs. time spent - fixating the number of objects stored