COUPLING DATABASES AND ADVANCED ANALYTICAL TOOLS (R)

IT4BI MSc Thesis

Student: SEDEM SEAKOMO
Advisor: ALBERTO ABELLÓ

Master in Information Technologies for Business Intelligence
Universitat Politècnica de Catalunya

Barcelona
JULY, 2014

A thesis presented by SEDEM SEAKOMO
in partial fulfillment of the requirements for the MSc degree in
Information Technologies for Business Intelligence

DEDICATION

I dedicate this thesis to my loving family

Acknowledgements

Commit to the LORD whatever you do, and your plans will succeed (NIV, Proverbs 16:3). I am very grateful to God Almighty for giving me the strength, knowledge, wisdom and understanding to carry out the study. Many were challenging times during my studies but He kept and sustained me through them all.

I am also grateful to my outstanding thesis supervisor, Professor Alberto Abelló, for his unique and distinguished role of constructive analysis and assessment of this work, as well as advice and guidance on progressing throughout the various stages of this work.

Finally, I would like to thank my family for their immense support, well wishes and prayers. I especially wish to thank my immediate family for their immense sacrifice, understanding and support throughout my studies away from home. In all, I really do appreciate every supportive and constructive roles played by all in line with the successful completion of my studies.

Thank you all.

Abstract

Today, several contemporary organizations collect various kinds of data, creating large data repositories. But the capacity to perform advanced analytics over these large amount of data stored in databases remains a significant challenge to statistical software (R, S, SAS, SPSS, etc) and data management systems (DBMSs). This is because while statistical software provide comprehensive analytics and modelling functionalities, they can only handle limited amounts of data. The data management systems in contrast have capacity to handle large amount of data but lack adequate analytical facilities. The need to draw on the strengths of both camps gave rise to the idea of coupling databases and advanced analytical or statistical tools which seems very promising and is gaining a lot of grounds.

This work studied the level of development of integration of a rising popular advanced analytical tool (R) with database systems (PostgreSQL, Oracle, DB2, SQL Server) and investigated the analytic performance of such coupling vis-à-vis the performance of stand-alone implementation of (R). The results showed that the overall performance of coupling databases and R is about two (2) times faster than performance of stand-alone R. In the case of some individual benchmarks, the coupled systems (R+DBMS) performance is more than ten (10) times faster. However, there remain the challenges of efficient retrieval and passing of data to analytic functions, code portability, indistinguishable or flat analytics performance on small datasets and integration configuration snags with some of the well-known DBMSs. Although, stand-alone R performs competitively well compared to DBMSs coupled with R in cases of very small datasets analytics, the issue of data security still lingers.

Our conclusion is that coupling databases with advanced analytical tools (R) is a good concept and technique which yields considerable performance gains for advanced analytics on substantial datasets provided retrieval and passing of data to the analytical functions are efficiently done. Thus, we confirm the initial assertion or hypothesis but on the condition that significant amount of data is involved in the process and the data is efficiently retrieved and passed to analytic functions. Overall, we recommend an integration which synergizes the robust DBMSs' data management capabilities and the rich statistical functionalities of advanced analytical tools for complex analytics in-situ databases in all situations for faster performance and data security.

Keywords: *Advanced analytics, in-database analytics, R, data management and analytics*

Contents

List of Figures

List of Tables

Listings

List of Acronyms

CIA	Confidentiality, Integrity and Availability
CLR	Common Language Runtime
CRAN	Comprehensive R Archive Network
DBMS	Database Management System
DLL	Dynamic Link Library
DSN	Data Source Name
HDFS	Hadoop Distributed File System
JDBC	Java Database Connectivity
MPP	Massively Parallel Processing
NoSQL	Not only SQL
ODBC	Open Database Connectivity
ORE	Oracle R Enterprise
RDBMS	Relational Database Management System
RJDBC	Java Database Connectivity for R
RODBC	Open Database Connectivity for R
SQL	Structured Query Language

Chapter 1

INTRODUCTION

This introduction chapter examines the scientific background of the subject matter, looking at the development and the need for advanced analytics, the motivation and significance of the study. The chapter also touches on the research questions and presents rationales that hold or support the theory of the study by formulation and declaration of the thesis statement or hypothesis. Also, it clearly précis the scope of the study in terms of de-limitations and limitations.

1.1 Scientific Background

SQL/relational DBMSs are powerful systems for managing, querying, and aggregating data. Nevertheless, performing complex analytics, drawing of inferences, making predictions, or fishing out subtle relationships in data suffer performance difficulties and/or are hard with SQL/relational DBMSs. In spite of this, several contemporary organizations presently collect various kinds of data at very detailed level possible, creating large data repositories in various DBMS and/or other data stores. Statistical software seem to provide answers for advanced analytics needs of these organizations even though they also have their limitations. Hence, the ability to carry out meaningful advanced analytics using various statistical analysis methods on data in such repositories has become very critical for effectiveness, efficiency and competitiveness.

However, the need to perform the advanced analytics over the large amount of data stored in databases remains a significant challenge to existing statistical software and data management systems. On the one hand, statistical software provide rich functionality for data analysis and modelling, but can handle only limited amounts of data. For instance, advanced analytical/statistical tools like R and SPSS operate entirely in main memory. On the other hand, data management

systems such as relational DBMS can store large amount of data, but provide insufficient analytical functionality. There is therefore a need to combine the data management capabilities of data management systems (e.g. relational model/DBMS) with the statistical functionalities provided by advanced analytical/statistical tools such as R in order to meet the challenge of delivering the ever-growing sophistication of analytics to businesses.

The idea of in-database analytics seems very promising and is gaining a lot of grounds. This is because taking algorithms to data rather than taking data to the algorithm in general has several performance benefits and data security advantages. Additionally, R as an advanced analytical tool has been receiving a lot attention in recent times, seeing major DBMSs providing means to work with it in-database. There was therefore a need for a carefully planned, painstaking, extensive and practical study into the analytic performance capabilities and possibilities of integrating advanced analytical tools such as R with DBMSs to obtain detail insights. In light of this, the research topic *Coupling Databases and Advanced Analytical Tools (R)* has been conceived for study.

Basically, most RDBMS are employing extended RDBMS features to power the embedded execution of R. This is to help them to overcome the obvious performance problems associated with the abusive use of SQL for advanced analytics (linear algebra operations). Facilities are being provided for user-defined functions inside the DBMS that either implement certain matrix operations or call R to perform the operations. However, these facilities come with varying limitations with respect to performance, completeness and scalability. So prudence requires that we first benchmark the performance, scalability and completeness of the solutions on offer before considering usage [42].

R is an open-source language and environment. R programming language is one important tool for computational statistics, visualization and data science [1]. It is used to interpret, interact and visualize data. It is used by big technology companies, such as LinkedIn, Google and Facebook, that thrives on analytics. Of late, R has been receiving a lot of attention and known database management systems (DBMS) such as Oracle, PostgreSQL, SAP-HANA, Sybase, etc are providing facilities for integrating with R. The question which then arises is: *How does this advanced analytical tool perform when coupled with the database systems?*

Some studies involving analytics and databases have been done in times past. These include *Database Analytics Acceleration using FPGAs* [44] for evaluating expensive analytics queries while saving CPU resources; *The MADlib Analytics Library or MAD skills, the SQL* [25] which introduces an open-source library of in-database analytic methods of SQL-based algorithms for machine learning, data mining and statistics inside database engine; and *Towards a Unified Architecture for in-RDBMS Analytics* [18] which presents a unified architecture for in-RDBMS analytics with emphasis on faster implementation of new statistical techniques in RDBMS. Ad-

ditionally, some performance benchmark tests and studies on R have also been carried out in the past by others including Stefan Steinways [41], Donald Knuth [29] and Philippe Grosjean [23][24]. These studies centred on comparing performance of versions of R implementations, R implementation with and without some packages and R as analytical tool compared with other analytical tools. Our work however, focuses on an aspect which has not received proper studies; the performance study of database integrated R implementations as against stand-alone R implementation.

1.2 Motivation for the study

The choice of the study of the thesis topic, *Coupling Databases and Advanced Analytical Tools (R)*, was inspired by two main considerations: the development of analytics as an industry and the increasing relevance of data mining which is to be driven by complex analytics.

In recent times, there has been an upsurge in the amount of data which organizations store and/or analyze for information in their decision making process. According to [28], the gleaning information and insights from these massive stores of data has become an industry in itself while presentation of the information in an intuitive and easily obtainable means has become increasingly challenging. Additionally, the important observations and conclusion by [42] cannot be swept under the carpet. According to these observations, relevance of complex analytics will increase dramatically along with increase in importance of data mining and other sophisticated operations on data. This development is expected to be driven by advanced analytics problems with remarkable economic importance such as focused segmentation of customers, proper placement of advertisements, and recommendation engines for various use. So, there will be a transition from the basic analytics in traditional business intelligence systems to more advanced analytics.

In light of the reasons given above, coupled with the fact that many organizations use SQL/ RDBMS to store, manage and process data for their information needs and/or for decision making, there is a need for us to be able to exploit advanced statistical techniques and methods, which abounds in advanced analytical/statistical tools like R, while not loosing the benefits of robust data management and processing capabilities offered by RDBMSs. According to [42], Hadoop does not seem to be a viable option since advanced analytics are not absolutely parallel. Therefore, Hadoop will suffer significant performance problems, hence, not a rational consideration. Nevertheless, the domain of advanced analytics can be seen as still in its formative years, and the competition for systems that will eventually rule is open. Various major DBMSs are therefore being positioned in response to this and one of such move is the provision and the use of

extended RDBMS features that support R. It is therefore, important to study the current phase of development of each of these systems and to evaluate the implementation approaches as well as functionalities, performance and completeness the systems are offering, hence the choice of the topic for study.

1.3 Thesis Statement (Hypothesis) Declaration

In general, we expect integration of advanced analytical tools (R) with DBMSs for the purpose of performing complex analytics to provide benefits including the following: Provision of in-database R analytic capability will enhance analytic performance, eliminate latencies required for data movement (lag to extract data and deliver it to analytical platform) or eliminates data movement and duplication, preserves data or information security and reduce latency time from raw data to new information, exploit parallelism of database systems, expand analytical capabilities of database systems, and promise reduced costs and risks. Based on these supposed benefits, we arrived at the following hypothesis statement which establishes the premise of this research work:

Coupling advanced analytical tool (R) with database systems leads to better and enhanced analytic performance than stand-alone advanced analytical tool (R)

The experiments of the study therefore attempt to verify (validate or refute) this assertion. The study also reasoned why and to what extent the hypothesis is right or wrong. In doing so, the DBMSs were selected such that the architecture of R integration implementations are different and as much as possible representative of the various architectural implementations options discussed in section 2.2 of chapter 2.

This thesis report presents the findings and analyses of mainly analytical performance among database coupled implementations of R compared to stand-alone implementation of R. In specific terms, this work presents the current state of development (level of maturity) of integration of R and DBMSs that were studied. The report compared and contrasted some integration implementations (identifying strengths and weaknesses), available functionalities (whether all base R features are available in the coupled or integrated systems implementations or not) and analytic performance results among the DBMSs as well as with a baseline of stand-alone R. The recorded analytic performances, in line with the hypothesis or otherwise, from the empirical work (experimental or benchmark tests results) were discussed and somehow explained with reference to the architectural implementation of R integration by the various DBMSs that were studied.

1.4 Research Questions

In an effort to gain insight into the appraisal of coupling databases and advanced analytical tool (R), the study examined the following research questions:

(i) What is the current level of development (*completeness*) of integration of R with DMBS?

(ii) How is the *performance* of coupling databases with advanced analytical tool (R) compared to stand-alone analytical tool (R)?

(iii) How is the *scalability* of coupling databases with advanced analytical tool (R) compared to stand-alone analytical tool (R)?

(iv) What are the inherent implications of architectures of R integration that impact performance?

(v) Are there any lessons to be learnt on the way forward?

The terms, *completeness*, *performance* and *scalability*, as used in the context of this study have the following meaning: *Completeness* refers to the level of development of R integration with DBMS. That is whether the functionalities available in stand-alone R are also available in the in the R and DBMS integrated systems. *Performance* as used refers to the time taken by the systems to perform analytic tasks. In other words, how fast the systems execute the analytic routines. The shorter the time taken, the higher we rate the performance of the system. *Scalability* as used in our context basically means how scalable the execution of analytic routines by the systems are when the amount of data involved increase. The smaller the increase in the time taken by a system to run the analytic tasks when data grows, the more scalable we say the system is.

1.5 Scope of the study

This study focused on understanding the performance, scalability and completeness of some selected DBMSs with regard to R integration implementation. This choice of scope is in line with our motivation for the study as explained in section 1.2 of this chapter. The empirical work (experiment) covers evaluation of matrix operations which form the core of advanced analytic routines and functions. It also covers assessment of program execution control flows which of course characterize automated or computer ran routines. The study does not cover benchmarking of *intra-command parallelism* of matrix operations because it is a whole subject matter on its

own and cannot be adequately dealt with within the available time for the study. Intra-command parallelism as used here refers to breaking down a command into smaller parts and executing the sub-parts in parallel or concurrently. This means that not only are jobs/tasks distributed across processors but parts (sub-parts) of a job/task meant for an operator/command are ran in parallel within the operator/command. In addition, only one set of computing configuration setup (RAM, Speed, Core, OS, HDD, etc) was used for the experiment.

The benchmark tests carried out in this study were focused on coupling of R with only relational DBMSs and did not cover non-relational DBMSs or NoSQL databases or data stores (such as graph database-NeoDB; or document stores- MongoDB). This is mainly due to time and resource constraints. The limited list of selected DBMSs used in the benchmark test include Oracle, PostgreSQL, DB2 and SQL Server. R integration with other database systems such as SAP HANA, Sybase RAP, and Cloudera Impala were only reviewed in the theoretical work.

The study focused on directly coupling R at only the data layer of analytical stack and did not cover coupling R at the other levels/layers (analytic layer and presentation layer) in the analytic stack shown in figure 2.1 on page 13 in section 2.2 of chapter 2. This is to allow us have enough time to carry out the relevant experiments required to verify the expected benefits (mentioned in section 1.3) of coupling R with DBMSs (data layer). It is also to ensure that our empirical work is in alignment with the motivation and the need for the study so that we do not unjustifiably invest time and effort in issues which are not of major concern.

Apart from accessing database objects (tables, views, etc) from within R environment or directly accessing R functions inside databases, it must be noted that R can be indirectly coupled with databases by first embedding the R routines as procedures inside applications written in high level languages such as C#, C, C++, Java or COBOL. These external procedures can then be called from the DBMSs to carry out the required analytics. Thus data layer integration of R via the application layer. We did not cover this possibility in much detail since our focus is on integrating R directly at the data layer.

Besides, due to inability to get allocation of the computing resources from the Hasso Plattner Institute (HPI) Future SOC Lab as planned in the initial proposal, some DBMSs such as SAP HANA which requires significant computing resources and configurations, and Sybase RAP for which there is no free version were left out of the list of candidate DMBSs tested. In addition, Cloudera Impala which uses Hadoop data sets (data stored in HDFS or HBase) and recommended physical memory (impalad daemon) of 128 GB or higher for a node has also been left out owing to no immediate availability of the required resources for the study.

1.6 Contributions of the study

Most advanced analytics are founded on matrix calculations and are expressed as collections of linear algebra operations. But performing substantial floating point operations in SQL will possibly lead to awfully slow performance. So relational simulations of linear algebra operations to carry out advanced analytics will often result in abysmal I/O and CPU performance. Also, such simulations are hard to fathom and realize, and makes code maintenance an expensive venture. Furthermore, SQL simulations are knotty for linear algebra operations that involves iterations (loops or convergences).

In light of issues mentioned above, the study has contributed by scientifically proving, based on the empirical results from the experiments, that better performance is achievable when databases are coupled with advanced analytical tools such as R in carrying out complex analytics. It also contributed that such an approach is the way to go for complex analytics that involves significant amount of data. The study further pointed out that the R integration implementations in which more of the advanced analytic functions have equivalent native SQL counterparts that are executed in-database produces the best performance results. Closely linked to this point is a caveat that the database resident data involved in the analytic process must be efficiently retrieved and passed to the analytic functions, lest there will be worsen performance.

These contributions will be of immense use to statisticians, data analysts, researchers and practising professionals in data management and analytics fields, and developers of database management systems in the sense that it will provide them with relevant information as to how the studied DBMSs compare to each other in respect of integration with R (base R).

Chapter 2

TECHNOLOGIES REVIEW

In this chapter, we take a look at R and also examine the different DBMS architectures with regard to R integration and implementation possibilities. There is also a discussion on the level of development of R integration with respect to the various DBMSs examined.

2.1 Advanced Analytical Tool (R)

R is an open source programming language and environment for statistical computing (or analysis) and graphics (or visualization). It was originally developed by Ross Ihaka and Robert Gentleman at the Statistics Department of the University of Auckland, New Zealand in 1993 in attempt to meet the need for a language that would help them teach introductory statistics to their students [27]. It is similar to the S language which was originally developed at Bell Labs. R is now one of the trendy platforms for data analysis and visualization. It is a free, open-source software with versions available for different operating systems including Windows, Mac OS X, and Linux. It has a large and active worldwide support and research community. There are other well-known statistical and graphing packages such as Excel, SAS, SPSS, Stata, Minitab, etc but R has an edge over them in the sense that it has so many compelling features and benefits which make it mostly the preferred one over the others. In examining the strengths and weaknesses of R, we look at R from the two perspectives; R as a language and R as an environment.

2.1.1 R as a Language (Implementation)

R is a language in the sense that it provides vocabulary (keywords) and a set of grammatical rules (syntax) for instructing computers to perform specific tasks. It therefore enables programming (creation of routines that control actions) of computers and/or expressing of algorithms especially statistical/analytical algorithms. As a language, the benefits and compelling features of R according to [28], include the following:

(i) It is comprehensive and offers almost any type of data analytic techniques.

(ii) It is readily extensible and provides a tool to rapidly program new statistical methods.

(iii) It contains advanced statistical routines not yet available in other packages and new methods are made readily available for download periodically.

(iv) It provides a variety of graphical interfaces for use without need to learn a new language.

(v) It is cross-platform and runs on a wide array of operating systems (Windows, UNIX, Linux, Macintosh) with different hardware specifications (both 32 and 64 bit processors).

2.1.2 R as an Environment (Workspace)

R is also a programming environment because it provides interactive workspace and a collection of facilities (processes, tools and physical settings) for developing, testing and debugging programs in general and statistical/analytical algorithms in particular. Thus, R provides a set of procedures and tools to develop source code of statistical/analytical algorithms. As an environment, the benefits and compelling features of R according to [28], include the following:

(i) It provides an excellent platform for interactive data analysis and exploration, and it has contemporary graphics capabilities for complex data visualisations.

(ii) It provides means of rapidly programming new statistical methods, and importing and exporting of data from/to a wide variety of sources.

Despite the benefits of R as a language, it (base R) is not without some weaknesses. Some of the weaknesses of the base R include:

(i) It analyzes data in memory and so the amount of data that can be analyzed at a time is constrained by the memory capacity of the machine that it runs on. There are however a number of useful packages such as *ff*, *bigmemory*, *filehash*, *ncdf*, etc. which facilitate storing data outside of R's main memory (RAM) and hence overcome this problem.

(ii) Memory management by base R is poor because R can rapidly consume all available memory. This can become an impediment when performing operations on larger datasets. This problem can however be minimized by using 64-bit operating systems to access more memory than memory space accessible by 32-bit operating systems. Also, there are a number of packages useful such as *ff*, *bigmemory*, *filehash*, *ncdf*, etc which facilitate storing data outside of R's main memory (RAM) [28].

(iii) Naturally, base R is single-threaded and does not allow multi-core parallel processing. This becomes a stumbling block for faster performance of operations where parallelism could speed up execution and delivery of results. However, a number of parallelism packages have be developed to overcome this problem. Also, one can run multiple R sessions or make multiple system (*Rscript*) calls to achieve parallel execution of task with R.

Largely, the major weaknesses according to [45], have been addressed in enhanced versions of R, some of which have been commercialized by companies such as Revolution Analytics [1]. For instance, some modifications have been made to the core R engine and advancement of core R algorithms so that they can run in parallel and thus enable exploitation of multiple processors for speed. Also, they are distributed and run out of memory, subverting the limitations imposed by memory size, and can be spread across a big cluster or cloud. A further step has also been taken to develop a server version of R, which has the capacity to expose analytics functionality as a web service API. This makes it possible to integrate R with data layer for advanced statistical processing based on uploaded routines and analyses.

Since the design of R was largely influenced by two languages namely S, from Bell Labs, which was created in the 1970s and Scheme by MIT [9], R adopted S syntax while using semantics of Scheme. Ostensibly, R looks more like S. Despite the superficial similarity between R and S, there exist two fundamental differences between S and R. These key differences lie in scoping and memory management, a consequence from R's Scheme heritage [8]. The distinctions are précised in [21] as follows: The memory management technique employed in R allocates a fixed amount of memory at start-up and manages it with an on-the-fly garbage collector. This means that there is very little heap growth and as a result there are fewer paging concerns than are seen in S. In the case of scoping, variables within functions in S are either local or global while functions in R have access to the variables which were in effect before the definition of the function; a lexical scoping notion of Scheme. Thus in S, global variables get manipulated by functions while in R, it is the variables which are in existence at the time of the function definition. The implication is that *free variables* (variables that are neither function's formal parameters nor function's local variables) within functions in S are determined by a set of global variables while same in R

functions are determined by the environment in which the function is created.

R can be used as an interactive environment, or embedded scripts and models in packages which are integrated with other software modules. It can be used to analyze data from scores of varied data sources including external files or databases [27]. The use of R is rapidly gaining grounds among the analyst community due to its numerous and increasing set of functions which are made available as packages in the main repository called CRAN (The Comprehensive R Archive Network) [51]. R is principally different from other statistical solutions such as SPSS, SAS and the likes in the sense that R is a statistical programming language while the others are statistical command languages. This distinction is key and gives R superiority over the others when integrating with RDBMS. With the help of *RODBC* and *RJDBC* R-side drivers, R can read SQL databases. Also, there are R-specific database drivers such as *RMySql*, *RPostgreSQL*, *RSQLite*, and *ROracle* which allow more or less direct read from and/or write to a database from R under the R-specific protocol (*DBI*) [27]. There is however no R-specific driver (*RSQLServer*) for SQL Server.

2.2 Different DBMS Architectures of R Integration

Integration with R can be provided at different layers within the analytic stack. These layers (shown in the figure 2.1 on page 13), according to [39] and [12] include the following:

(i) Data Layer (e.g. Oracle R Enterprise, Sybase RAP, SAP HANA, IBM Netezza).

(ii) Analytics Layer (e.g. SAS, IBM SPSS, RStudio, Matlab, Zementis).

(iii) Presentation Layer (e.g. Tableau, Jaspersoft BI Software, TIBCO Spotfire's BI Dashboard).

Nonetheless, as defined by the scope of this study in section 1.5 of chapter 1, we focused on the integration at the data layer, hence a presentation of a brief architectural review of R integration with some well-known DBMSs. When we take integration of R at the data layer, there are alternative approaches to implement it. These alternative means of integrating R at the data layer are *outside-in*, *inside-out* and *embedded* [3][10]. In *outside-in* arrangement, R is connected to the database using Java Database Connectivity or Open Database Connectivity (JDBC or ODBC) and R retrieves (pulls) the data to be analyzed from the database. In *inside-out* arrangement, data is transferred (pushed) to R from within the database and the aggregated and/or analyzed results sets are sent back from R to the database. The *embedded* arrangement is where the R environment (part of R environment) and/or the R code execution is made an integral part of the core DBMS.

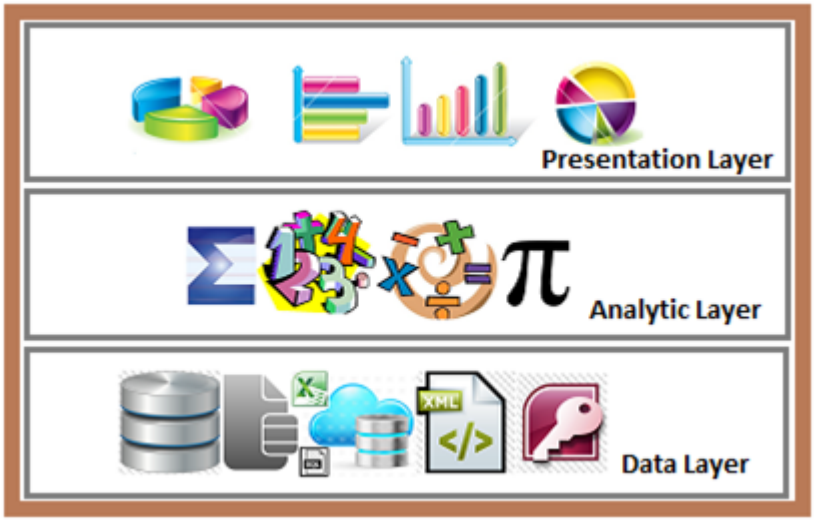

Figure 2.1: Broad classification of different layers within BI analytic stacks

The following are architectural overview of R integration with some DBMSs for analytics:

2.2.1 ORACLE R Enterprise

Oracle R Enterprise (ORE) is a component of Oracle Advanced Analytics (OAA), an option of Oracle Database Enterprise Edition, enables execution of R scripts within SQL queries. There is a set of packages built on top of the R engine to allow R computations to be executed in-database. Queries to the database can thus include a call to an R script registered in the database R script repository. Oracle has mixed approach or allows for the three approaches of integrating R with databases (refer to table 2.3 on page 31). The in-database R script execution is made possible by the following components in the Oracle R Enterprise:

(i) *Oracle R Enterprise (ORE) Transparency Layer*- This component houses set of R packages with functions to connect to the database and use R functionality in-database. It avails the database objects (tables, views, etc) to the R environment as if they were native R objects. The ORE transparency layer converts R commands and scripts into SQL equivalents and exploit the scalability of the database as a compute engine.

(ii) *Oracle In-Database Statistics Engine*- This component has a collection of statistical procedures and functions that match statistical libraries frequently used. It extends the databases

statistical functions library and advanced analytical computations by providing support for the entire R language and the statistical functions available in the *base R* and selected R packages. R commands and scripts are translated and executed seamlessly.

(iii) *Embedded R Execution*- Oracle R Enterprise packages on the server support the execution of R commands within SQL queries and PL/SQL statements. Embedded R is executed in spawned R engines that can run in parallel. Embedding R enables execution of R algorithms on very large data stores and scheduling embedded R for lights-out processing.

R closures (functions) can be executed using an R or SQL API while exploiting data parallelism, and producing complex visualisations and results. R functions that do not map to any native functions are handled by *extproc* remote procedure calls to multiple R engines running on multiple database servers or nodes. Generally, Oracle database employs the external procedure agent, *extproc*, to support external procedures. Oracle R Enterprise exploit *extproc* to support in-database execution of R. When the external procedure is invoked, the Oracle database starts an *extproc* agent and the established connection is used to pass instructions to the agent for executing the procedure. The agent loads required DLL or shared library, runs the external procedure, and passes back results returned by the external procedure. The interfaces, known as *Embedded-Layer RQ Functions*, pass streams of data to multiple instances of R for parallel processing in either row by row, group of rows, or table of rows fashion.

The three layers can be distinguished in the following ways: While the transparency layer intercepts all R commands/scripts and converts them into SQL counterparts for database native execution, the ORE statistics engine acts as a database library, housing both the native database statistical functions and other functions found in base R and selected R packages. The Embedded R Engine however enables data parallel execution, efficient data transfer between the database and the R engine and support leveraging of computing resources such as memory, processors, etc. Figure 2.2 on page 15 shows architecture of Oracle R Enterprise with the three components.

ORE provides two interfaces for the embedded R execution: an interface for R and another interface for SQL.

(i) *The R Interface for Embedded R Execution:* The R interface facilitates interactive execution of R scripts at the database server from the client R engine. This interface offers a number of functions for the embedded R execution. The functions include *ore.doEval*, *ore.tableApply*, *ore.rowApply*, *ore.groupApply* and *ore.indexApply*. The execution of R code with in-database R engine in the R interface is planned by the *ore.doEval* and the results returned to the desktop for further analytic operations. The R interface returns the

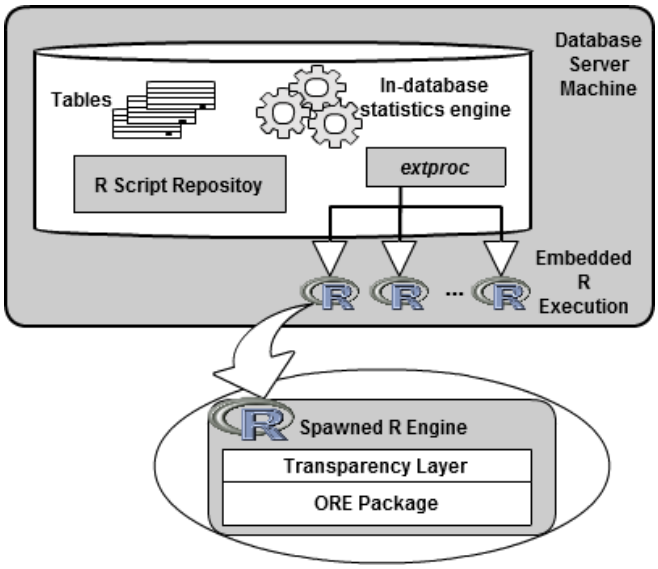

Figure 2.2: Architecture of Oracle R Enterprise. Adapted from [26]

results to the client as R objects that can also be passed as arguments to R functions. The need to dispatch data from the database to R is eliminated. Rather, the R function is posted to the database where the functions are processed in parallel and the results are only retrieved into R's memory if required for further operations or analysis in R.

(ii) *The SQL Interface for Embedded R Execution:* The SQL interface enables interactive execution from any SQL interface such as SQL Developer or SQL Plus. It also makes it possible for R scripts to be incorporated in production database-based systems. To facilitate executing R scripts in the SQL interface, ORE offers a number of SQL embedded R executions functions. These functions include *rqEval*, *rqTableEval*, *rqRowEval* and *rqGroupEval*. The SQL interface allows results to be stored directly in the database. Before invoking the R scripts from SQL, the R script has to be first created in the database's R script repository. Once the script is created, it can be invoked via SQL.

Table 2.1 on page 16 presents the embedded R functions offered by the two interfaces provided by ORE. These functions are well explained in *Oracle R Enterprise User's Guide* [34].

R Interface Function	SQL Interface Function	Purpose
ore.doEval	**rqEval**	Invoke stand-alone R scripts
ore.tableApply	**rqTableEval**	Invoke R script with full table input
ore.rowApply	**rqRowEval**	Invoke R script one row at a time, or multiple rows in ”chunks”
ore.groupApply	**rqGroupEval**	Invoke R script on data indexed by grouping column
ore.indexApply	**N/A**	Invoke R script N times
ore.scriptCreate	–	Create an R script in the database
ore.scriptDrop	–	Drop an R script in the database

Table 2.1: Embedded R functions of ORE interfaces. Adapted from [30]

The following code snippets give the syntax and illustrates simple use of the functions, starting with the R interface functions;

Listing 2.1: Function ore.doEval()

```
#SYNTAX:
ore.doEval(FUN[, ..., FUN.VALUE = NULL, FUN.NAME = NULL])

#EXAMPLE:
myIFunctionExx <- ore.doEval(
  function() {
    myFunction <- t(c(1:5))
    myFunction
     }, ore.connect = TRUE);
```

This invokes stand-alone R script (function) named `myFunction` in the R script repository of the database without input data.

Listing 2.2: Function ore.tableApply()

```
#SYNTAX:
ore.tableApply(X, FUN[, ..., FUN.VALUE = NULL,
    FUN.NAME = NULL, parallel = FALSE])

#EXAMPLE:
myIFunctionExx <- ore.tableApply(
  dbrestbl,
  function(dat) {
  m <- mean(dat);   s <- sd(dat);
  res <- c(mean = m, stddev = s);
  res});
```

This invokes R script with an entire table named `dbrestbl` provided as input all at once to the function.

Listing 2.3: Function ore.rowApply()

```
#SYNTAX:
ore.rowApply(X, FUN[, ..., FUN.VALUE = NULL,
    FUN.NAME = NULL, rows = 1, parallel = FALSE])

#EXAMPLE:
myIFunctionExx <- ore.tableApply(
  dbrestbl,
  function(dat) {
  m <- mean(dat);   s <- sd(dat);
  res <- c(mean = m, stddev = s);
  res},
  rows=25);
```

This activates multiple parallel invocation of R script with a chunk size of 25 rows of a table named `dbrestbl` provided as input at a time to the running functions.

Listing 2.4: Function ore.groupApply()

```
#SYNTAX:
ore.groupApply(X, INDEX, FUN[, ..., FUN.VALUE = NULL,
    FUN.NAME = NULL, parallel = FALSE])
#EXAMPLE:
myIFunctionExx <- ore.groupApply(dbrestbl,
                INDEX = dbrestbl$age_grp,
                parallel = TRUE,
                function(dat) {
                library(ORE)
                ore.connect("rquser", "orcl", "localhost", "rquser")
    c(mean=mean(dat), stddev=sd(dat));
                })
```

In this case, the data table named `dbrestbl` is partitioned by the `age_grp` variable, the mean and the standard deviation are computed, and then combined as results for output.

Listing 2.5: Function ore.indexApply()

```
#SYNTAX:
ore.indexApply(times, FUN[, ..., FUN.VALUE = NULL,
```

```
    FUN.NAME = NULL, parallel = FALSE])
#EXAMPLE:
myIFunctionExx <- ore.indexApply(5,function (x, scale = 50) x / scale)
```

This example causes the function to be ran five (5) times, with each run of the function invoked in separate engine at the database server.

Listing 2.6: Function ore.scriptCreate()

```
#SYNTAX:
ore.scriptCreate(name, FUN)

#EXAMPLE:
ore.scriptCreate("myNumberSquaresAndCubes",
function () {
  numbers <- 1:25;
  squares <-(numbers^2);  cubes <-(numbers ^3);
    data.frame(Number=numbers, Square=squares, Cube=cubes);
})
```

This creates an R script named `myNumberSquaresAndCubes` in the database which can then be used by name in other embedded R script functions.

Listing 2.7: Function ore.scriptDrop()

```
#SYNTAX:
ore.scriptCreate(name, FUN)

#EXAMPLE:
ore.scriptDrop("myNumberSquaresAndCubes")
```

This drops an R script named `myNumberSquaresAndCubes` from the database repository.

The SQL functions provided for in-database R script execution (lights-out processing) may activate multiple in-database R engines based on parallelism configurations of the database. The syntax of each of these functions with simple examples are given as follows:

Listing 2.8: Function rqEval()

```
--SYNTAX:
rqEval(
     cursor(select * from table-2),
     'select <column list> from table-3 t',
```

```
      <R closure name of registered-R-code>
      )

--EXAMPLE:
begin
  sys.rqScriptCreate('dbregfxn',
'function() {

nbrs <- 1:25
dfnbrs <- data.frame(Numbers = nbrs,
        IsEven=abs(nbrs%%2)==0,
        Squares = nbrs^2,
        Cubes = nbrs^3)
dfnbrs

}');
end;

select *
  from table(rqEval(
        NULL,
       'select 1 Numbers, 1 dfnbrs from dual',
       'dbregfxn'));

begin
  sys.rqScriptDrop('dbregfxn');
end;
```

Listing 2.9: Function rqTableEval()

```
--SYNTAX:
rqTableEval(
     cursor(select * from table-1),
     cursor(select * from table-2),
     'select <column list> from table-3 t',
     <R closure name of registered-R-code>
     )

--EXAMPLE:
begin
sys.rqScriptCreate('dbregfxn',
'function(x, param) {
        dat <- data.frame(x, stringsAsFactors=F)
        cbind(dat, ROWPROD = apply(dat,1, prod))
}');
end;

select * from table(rqTableEval(
   cursor(select * from dbrestbl),
```

```
   NULL,
   'select t.*, 1 rowprod from dbrestbl t',
   'dbregfxn' ));

begin
sys.rqScriptDrop('dbregfxn');
end;
```

The example uses all rows from the table `dbrestbl` as input to the R function that takes no additional parameters. It output result comprising the entire input data in addition to the computed `ROWPROD` of values.

Listing 2.10: Function rqRowEval()

```
--SYNTAX:
rqRowEval(
     cursor(select * from table-1),
     cursor(select * from table-2),
     'select <column list> from table-3 t',
     <from table-1 or num_rows>,
     <R closure name of registered-R-code>
     )

--EXAMPLE:
begin
sys.rqScriptCreate('dbregfxn',
'function(x, param) {
        dat <- data.frame(x, stringsAsFactors=F)
        cbind(dat, ROWPROD = apply(dat,1,prod))
}');
end;

select * from table(rqRowEval(
   cursor(select * from dbrestbl),
   NULL,
   'select t.*, 1 rowprod from dbrestbl t',
    1,
   'dbregfxn' ));

begin
sys.rqScriptDrop('dbregfxn');
end;
```

The example illustrates passing one row at a time from the table `dbrestbl` to the R function with no parameters required by the function. The function output is the input table `dbrestbl` with additional column `ROWPROD` which is computed as products of the columns of the row (`ROWPROD`).

Listing 2.11: Function rqGroupEval()

```
--SYNTAX:
rq*Eval(
     cursor(select * from table-1),
     cursor(select * from table-2),
     'select <column list> from table-3 t',
     <grouping col-name from table-1>,
     <R closure name of registered-R-code>
     )

--EXAMPLE:
CREATE PACKAGE numbersPkg AS
  TYPE cur IS REF CURSOR RETURN dbrestbl%ROWTYPE;
END numbersPkg;

CREATE FUNCTION numbersGroupEval(
  inp_cur  numbersPkg.cur,
  par_cur  SYS_REFCURSOR,
  out_qry  VARCHAR2,
  grp_col  VARCHAR2,
  exp_txt  CLOB)
RETURN SYS.AnyDataSet
PIPELINED PARALLEL_ENABLE (PARTITION inp_cur BY HASH (age_grp))
CLUSTER inp_cur BY (age_grp)
USING rqGroupEvalImpl;
```

The example defines `numbersGroupEval`, a private version of `rqGroupEval()`, since there is no `rqGroupEval()` function (it is a virtual function). The data cursor uses all data from `dbrestbl`, and one single grouping column, `age_grp`, is defined for use for the data cursor.

2.2.2 SAP HANA

The execution of R codes by integrating with SAP HANA is achieved by means of two of the alternative approaches of the data layer integration. These approaches of coupling R with SAP HANA are outside-in and inside-out (refer to table 2.3 on page 31). The architectural arrangement of these two approaches are shown in the figures 2.3 and 2.4 on page 22.

In the outside-in arrangement, HANA retains its usual traditional database role and connection from R is established using JDBC/ODBC. In another way, SAP RHANA package can be employed to transfer huge amounts of columnar datasets. In the case of inside-out setup, huge amount of data is transferred to R from within HANA. The aggregated and/or analyzed results sets are sent back from R to HANA.

We take a delve into the inside-out setup. Figure 2.5 on page 23 gives more details and

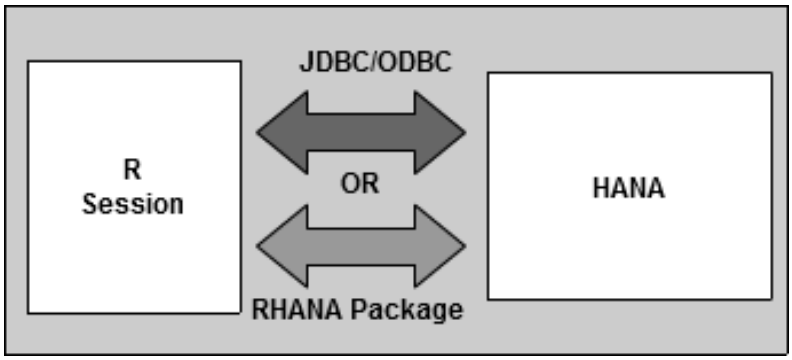

Figure 2.3: R and HANA Integration- Outside-in. Adapted from [4]

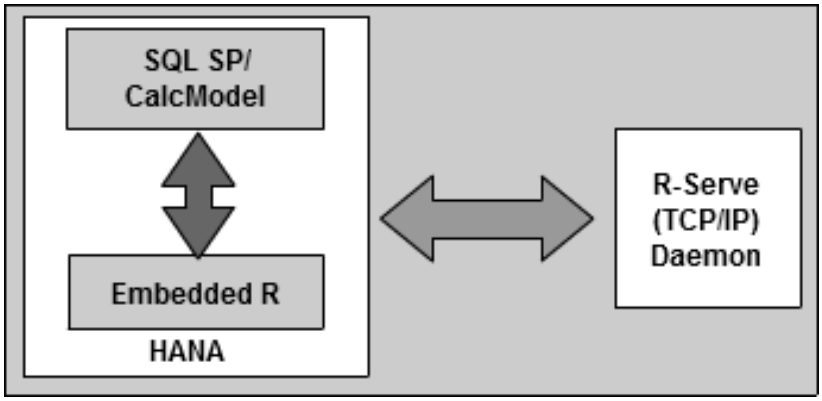

Figure 2.4: R and HANA Integration- Inside-out. Adapted from [4]

specifics of inside-out integration for execution of embedded R codes in HANA stored procedure. The database uses external R environment to execute R codes which are embedded in SAP HANA SQL code in the form of a *RLANG* procedure. The calculation engine of the SAP HANA database is extended to support data flow graphs (calculation models) describing logical database execution plans. The data flow graphs have nodes which can be native (resident) database operations or custom operations, one of which is the R operator. As usual of the operators of the calculation model, the R operator consumes input objects and returns a result table. However, unlike native database operations, custom operators (and for that matter R operator) are not restricted to a static implementation. Their implementation are independently amendable for each node. For an R operator, this is realized by the R function code, which is passed as a string argument to the operator.

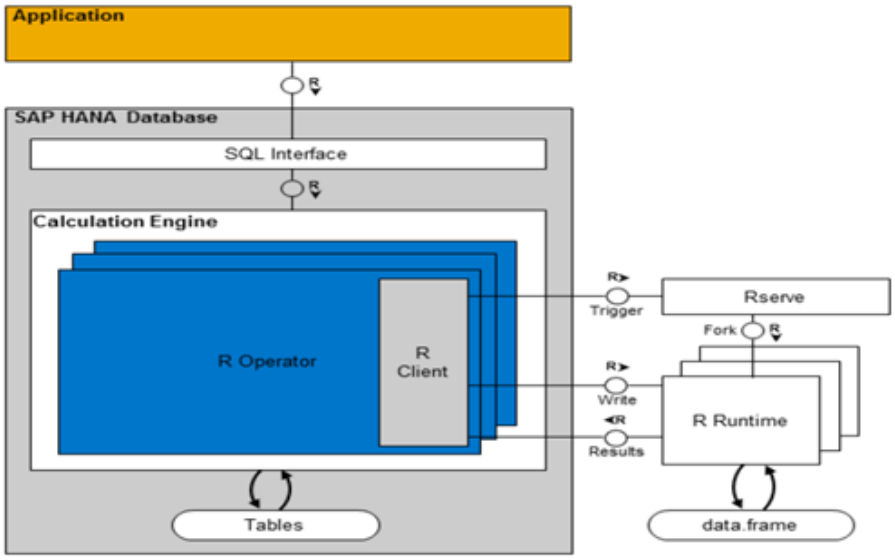

Figure 2.5: Integrated Architecture of SAP HANA. Reprinted from [38]

The arrival of the calculation model plan execution at the R-Operator causes the calculation R-Client of the engine to send a request via *Rserve* mechanism to create a dedicated R process on the R host. The Rserve is a TCP/IP server which enables the use of R capabilities from various languages devoid of necessity to initialize R or link against R library [48]. Subsequently, the R-Client transfers the R function code and its input tables to this R process, and triggers R execution. On completion of execution of the function by the R process, the result which is in the form of vector-oriented R data frame is returned to the calculation engine for conversion to appropriate form.

One key benefit of this architectural arrangement where the overall control flow is positioned in-database is that the database system can activate multiple R processes to run in parallel devoid of any concern to parallelize execution within a single R process because database execution plans are naturally parallel.

2.2.3 PostgreSQL (Postgres)

There are basically three ways to couple R and Postgres, namely via RODBC package, RPostgres interface or PL/R which supports limited types of functionality and only some versions of R. Thus, Postgres permits alternative approaches to R integration with the data layer (refer to

table 2.3 on page 31).

Postgres uses its extended RDBMS feature known as PL/R to exploit user-defined functions inside it (as a data manager) to call R to perform the matrix operations required for complex analytics. Postgres support for external function languages for which wrappers exist as well as the languages the base engine supports eases how R is run in Postgres. The PL/R extension enables running of R inside Postgres database in order to carryout advanced analytics in a simple, and controlled manner. The PL/R provides a procedural language interface to R from Postgres. The procedural languages enable us to script functions that are called within database queries. PL/R allows writing of Postgres functions and aggregate functions in the R statistical computing language. The PL/R loads an R interpreter into the Postgres backend process to facilitate in-database running of R code.

PL/R offers almost all the possibilities of writing functions in the R language and there exists commands for accessing the database through the Postgres Server Programming Interface (SPI) and to raise messages using *elog()*. Access to the database backend internals is not possible except by gaining operating system level access under the permissions of the Postgres user id, as with a C function. Due to this security implication, it is not safe to allow unprivileged database users to use this language. It is better installed as an un-trusted procedural language so that only database privileged users can create functions with it. Also, diligence is required when writing PL/R functions so as to ensure that it is not exploited for unintended purposes. Besides, there is an implementation restriction such that PL/R procedures cannot be used to create input/output functions for new data types.

The RODBC and RProgreSQL provide the traditional means (outside-in approach) of coupling Postgres with R. The RODBC package enables ODBC database connectivity of R with Postgres database while RPostgreSQL package provides Database Interface (DBI) compliant driver for R to access Postgres database.

2.2.4 IBM DB2

In order to be able to execute R scripts in DB2 database, the extended database feature of DB2 can be exploited. This allows embedded SQL user-defined functions and routines to be written in other languages such as C, C++, COBOL, etc to perform analytic operations within any SQL statement that returns a single scalar value, row or a table. This means that apart from the traditional means (outside-in approach) of integrating R and DB2 using RODBC/RJDBC, R script execution can indirectly be embedded in DB2 by first embedding the R rountines inside external stored procedures or functions written in some high level languages such as C, C++, or

COBOL. The external stored procedures are compiled, linked, and cataloged into a load library which is made accessible to a stored procedure workload manager (WLM) address space [13].

The consequence of the use of WLM by external stored procedure is that of slower performance compared to native stored procedure. This is because native stored procedures do not run in WLM address space but only engage the WLM during debugging. All the codes of native stored procedures run under users' task with users' DB2 thread merely switching to SQL procedures package upon stored procedure call. This eliminates queuing and delays as well as decrease dispatch overhead [13], thus making native stored procedures generally faster than external stored procedures. Nonetheless, when it comes to external stored procedures, keeping a procedure in the WLM address space eliminates the need of loading the procedure from the load library, hence improving overall execution performance of the external stored procedure than when not resident in the WLM. The bind of the Database Request Module, DBRM (SQL statements), is performed and a `CREATE PROCEDURE` statement is ran to define the procedure to DB2 [13]. The architectural arrangement is shown in the figure 2.6.

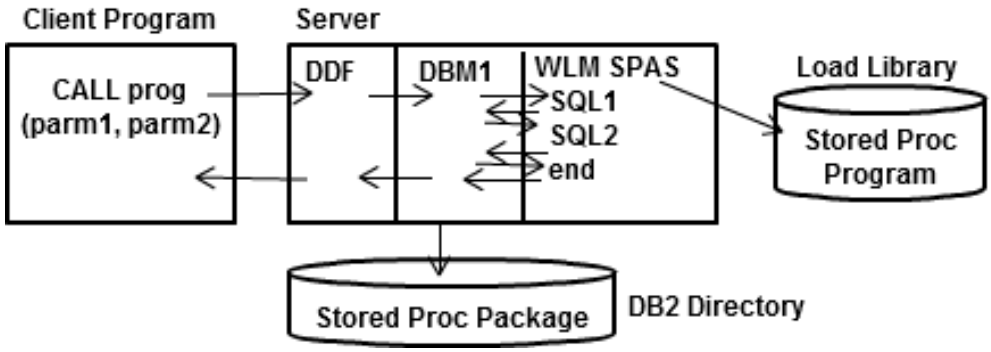

Figure 2.6: DB2 External Stored Procedure Architecture. Adapted from [11]

When the external procedures are called from SQL scripts in DB2 to carry out the required analytics, the DB2 database services address space (DBM1) searches for the procedure in the `SYSIBM.SYSROUTINES` catalog (a table that contains a row for every routine). It then obtains available Task Control Block (TCB) for use by the stored procedure, and the stored procedure address space is instructed to execute the stored procedure. Procedures which are not resident in the WLM address space are loaded from the load library. For a stored procedure to execute, control is passed to the procedure and on encountering embedded SQL statement in the procedure, control is passed back to the DBM1 address space for execution and the result returned to the stored procedure address space. Upon eventual completion of the stored procedure execution,

control is returned to the DBM1 address space and the final results are returned.

The architecture of R integration with DB2, to a large extent, is similar to that of SAP HANA. However, DB2 architecture slightly differ from the described architecture of SAP HANA in the sense that the execution of R script is initiated from within HANA and the data and code shipped to the external R engine while in the DB2 architecture explained, the execution of the R scripts starts from the R external engine and the required data is pulled from DB2 database into the external R environment for analysis. Nonetheless, the indirect embedding of R execution in DB2 via an application layer (embedded SQL programs in C and C++) is not our focus. We therefore, did not cover this technique in much details as our focus is on integrating R directly with the data layer.

Figure 2.7 shows the traditional means of integrating R and DB2 using RODBC/RJDBC. The packages (RJDBC and RODBC) are founded on Database Interface (DBI) conventions for

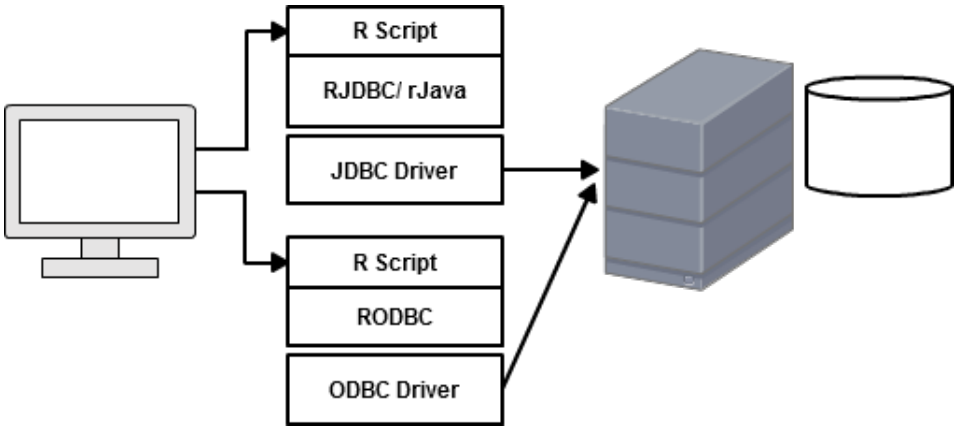

Figure 2.7: R connectivity options for DB2. Adapted from [27]

R . The DBI package hold virtual classes which are implemented by underlying database driver. RJDBC and RODBC respectively use JDBC and ODBC compliant database drivers to facilitate exchange of data between the DB2 database and the R engine.

2.2.5 MS SQL Server

R integration with SQL Server DBMS is not well developed compared to that of the other DBMSs such as Oracle, Postgres, Sybase RAP, etc. The clear option available for working with R and SQL Server is the traditional technique (outside-in). With this technique, R simply integrates with SQL Server using RODBC package and ODBC connection. The RODBC package imple-

ments ODBC database connectivity and enables commencement of database interactions from R to the SQL Server database. There is however no clearly defined means of directly calling R from SQL Server SQL queries. The RODBC makes available two groups of functions; the internal *odbc commands* which implement low-level access to C-level ODBC functions with similar names and the *sql functions* which operate at a higher level to read, save, copy and manipulate data between data frames and SQL tables.

The architectural arrangement of the traditional integration approach of using R together with SQL Server is the same as the generic ODBC connection architecture as shown in figure 2.8. The

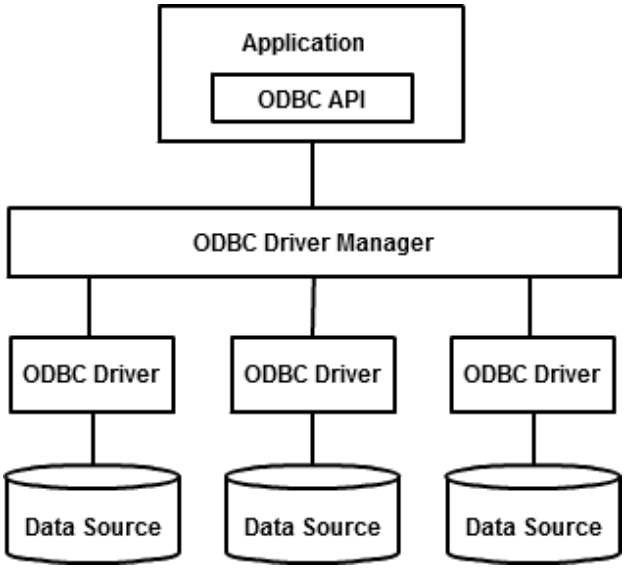

Figure 2.8: ODBC Architecture. Adapted from [47]

ODBC architecture has four (4) components, which as per the understanding from [47], will work for the SQL Server database and the R program/engine as follows:

(i) The *Application Programming Interface* invokes ODBC functions to connect to the SQL Server database, send and receive data, and disconnect.

(ii) The *Driver Manager* provides information to the R engine, dynamically loads required drivers on demand and provides argument and state transition checking.

(iii) The *Driver* processes ODBC function calls and manages all exchanges between the R engine and the SQL Server database. It also translates standard SQL syntax into native SQL of the SQL Server DBMS if required.

(iv) The *Data Source* comprises the data and the SQL Server database engine.

The RODBC package provides access to the database (SQL Server database) through the ODBC interface. The primary functions are given in table 2.2.

Function	**Description**
odbcConnect(dsn, uid="", pwd="")	Open a connection to an ODBC database
sqlFetch(channel, sqtable)	Read a table from an ODBC database into a data frame
sqlQuery(channel, query)	Submit a query to an ODBC database and return the results
sqlSave(channel,mydf,tablename= sqtable, append = FALSE)	Write or update (append=True) a data frame to a table in the ODBC database
sqlDrop(channel, sqtable)	Remove a table from the ODBC database
close(channel)	Close the connection

Table 2.2: Functions provided by RODBC. Adapted from [28]

We must however note that, just as in the case of DB2, R can be indirectly ran in SQL Server database (indirect embedding) by exploiting CLR or the external procedure feature of writing procedure in other languages (such as C, C++ or C#) other than SQL. For instance, by using common language runtime (CLR), we can define a stored procedure, with embedded R routines, as a static method of a class in a C# program and then compile this into an assembly. Then by enabling CLR integration feature of the SQL Server, we can register (using `CREATE ASSEMBLY` statement) this assembly created from the C# program (which has the embedded R routines) inside the database, and then create (using `CREATE PROCEDURE` statement) database stored procedures or functions that reference the registered assembly. Thus, the database stored procedures or functions that references the assembly eventually get indirect access to the R routines embedded in the assembly and so the routines can be called from within the database.

The issue of limited development of the integration of SQL Server with other advanced analytical tools (e.g. R) could to some extent be attributed to the fact that a lot of work has be put into SQL Server to have its own in-database data mining or analytics facility. SQL Server now has some highly advanced analytics and data mining features which simplify building of even sophisticated analytics and data mining solutions. The analytics feature is integrated into the steps of the data life cycle for discovering insights buried in data. Also, the analytic functionality, to some extent, can be extended and enhanced to create intelligent solutions. But compared

to R, there is almost limitless analytics possibilities in terms of functionalities, flexibility, extensibility, adaptability and community support (packages) obtainable in R. So, directly integrating SQL Server with R will provide a more powerful in-database SQL Server Analytics solution, combining the analytic prowess, customizability, extensibility and huge community support of R with the user-friendliness and robust data management capability of SQL Server DBMS.

2.2.6 SYBASE RAP

Sybase RAP is the Sybase database edition for the financial market. It is an integrated real-time analytics platform with capacity to capture, store, and analyze market-related and trade-related data for both real-time and historical analyses [35]. It provides integration with R thereby allowing for faster analytic algorithm development and extensive back-testing on historical data.

For embedded approach of R integration with database, Sybase RAP integrated analytics platform for R has two (2) key components namely *RAPCache* (an in-memory cache) and *RAPStore* (a historical database). The RAPStore is a column-oriented SQL database, which stores massive amounts of historical data in compressed form for high performance analytics and querying by several concurrent users. The RAPStore invokes R programs using SQL code, extending the RAPStore analytics capability and making R functions available as SQL functions in-database. The RAPCache is an in-memory SQL programming language database with query and transactional capabilities offered via a standard SQL interface and provides immediate access to new data on arrival.

Functions that run on R server are accessed via the RAPStore User-Defined Functions (UDF) infrastructure and invoked as SQL functions within RAP. The UDFs can be written in C or C++ to be invoked via standard SQL language. The UDFs in turn invoke R programs. Thus, functions in R can be invoked using SQL in the RAPStore. Figure 2.9 shows invocation of R function using

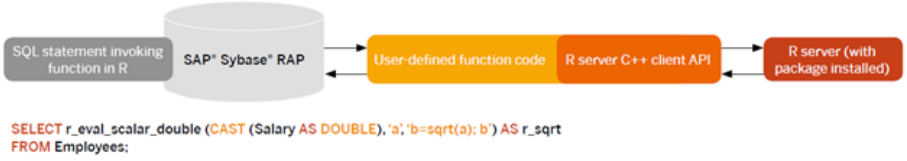

Figure 2.9: Invoking R Functions Using RAPStore UDF Features. Reprinted from [35]

RAPStore UDF feature. In a traditional way (outside-in approach to data layer integration), data from the historical store can be accessed from R using the RJDBC package, which allows JDBC access to the RAPStore.

2.2.7 CLOUDERA IMPALA

Cloudera Impala is an open source MPP (Massively Parallel Processing) query engine [14]. It runs natively on Apache Hadoop and supports low-latency, interactive queries on Hadoop datasets stored in HDFS (Hadoop Distributed File System) or HBase [46]. Cloudera Impala uses Hadoop as a storage engine but distant itself from MapReduce algorithms, and rather employ distributed queries, a concept inherited from MPP databases. This takes away the need to move (migrate datasets into specialized analytic systems) or transform (change data to proprietary formats) data and so processing of hefty datasets (using MapReduce) and running of interactive queries can take place on the same system using the same data and meta-data in carrying out analysis. R integrates with Impala by means of outside-in approach to data layer integration. This is done using generic Impala ODBC or JDBC driver in a similar fashion as discussed for ODBC and SQL Server in sub-section 2.2.5 on page 26. Integrating R with Impala facilitates fast, interactive querying on top of Hadoop datasets and data can be further processed or visualized within R. Figure 2.10 shows a diagram of how JDBC/ODBC can connect R with Cloudera Impala.

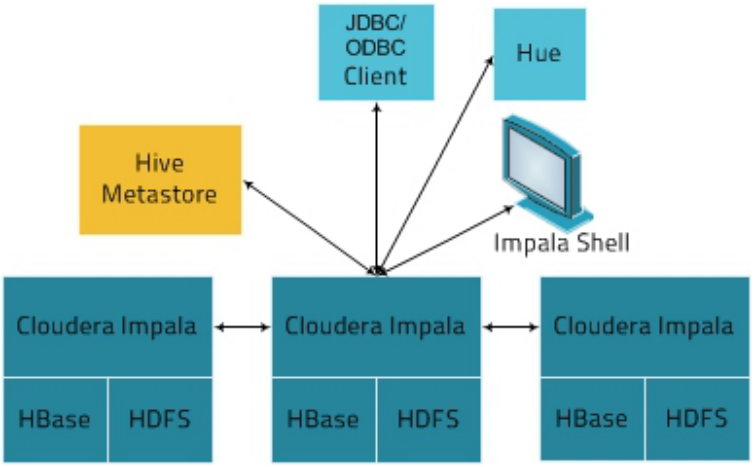

Figure 2.10: Cloudera Impala ODBC drivers. Reprinted from [46]

2.3 Choice of DBMS+R for Empirical Work

Having reviewed the developments on the implementation of R integration with the DBMSs, table 2.3 on page 31 summarizes the possible ways of coupling R with the reviewed DBMSs.

DBMS	Embedded	Outside-In/Inside-Out
Oracle	YES: ORE Server	YES: ROracle,JDBC
PostgreSQL	YES: PLR	YES: RPostgres,RODBC
Sybase RAP	YES: RAP Store- UDF(C,C++)	YES: RJDBC
SQL Server	NO: But (CLR, Ext Proc)	YES: RODBC
DB2	NO: But (CLR, Ext Proc)	YES: RJDBC,RODBC
Cloudera Impala	NO	YES: ODBC,JDBC
SAP HANA	NO	YES: RODBC,RJDBC,RHANA

Table 2.3: Summary of coupling R with some DBMS

The choice of the DBMSs for the empirical work was made by considering the following points:

(i) Differing architecture of R integration implementation

(ii) Representativeness of well-known DBMSs/vendors

(iii) Level of development of R integration

(iv) Availability and accessibility of DBMSs for testing

(v) Active on-going R integration development work or consideration efforts/suggestions

(vi) More generic or non-proprietary nature of the DBMSs

(vii) Fair and objective performance evaluation with respect to other DBMSs

In consideration of these factors, we selected four (4) DBMSs for our empirical work (analytic performance benchmarks). The selected DBMSs are *Oracle*, *PostgreSQL*, *DB2* and *SQL Server*.

2.4 Information (Data) Security

According to [49] and [40], the fundamental goals of security are the attainment of *confidentiality*, *integrity* and *availability* (CIA) of information resources. This CIA principle is generally used as benchmark for appraisal of information systems' security. We explain the three (3) essential goals with reference to [43] as follows: *confidentiality* is the prevention of unauthorized disclosure of information, either in storage, processing or transit; *integrity* is the assurance of accuracy of information such that there is no unauthorized altering of data while in storage, processing or transit; and *availability* is ensuring that information is ready for authorized use when needed.

Applying the CIA principle, we note that there are data/information security consequences when using stand-alone analytical tools in performing analytics. This is because the data involved have to be first retrieved from the DBMS, prepared into the required format and then stored in a file (flat file) before the analysis. The means of creating, storing and exchanging the data files in readiness for analysis by stand-alone analytical tools often leave sensitive data vulnerable. Additionally, data stored in flat files are generally less secure (accidental deletion, virus attack, etc.) than their DBMS counterparts.

We have identified the following issues as additional security challenges that come with using stand-alone advanced analytical tools in performing analytics on data kept in files instead of directly coupling these analytical tools with databases to perform analyses.

(i) The files in which the data is kept are usually not part of managed database solution where some minimum level of built-in security (authentication and authorization) is guaranteed. Therefore, there is higher risk for breach of confidentiality of the data.

(ii) Most often, in cases where users attempt their own encryption of data by with even third-party tool-kits, problems often arise and immediate availability of the data for analysis is jeopardized. Also, the likelihood of infringement on the integrity of the data is increased.

(iii) Also, many a time, there are no security policies (either system enforced or human enforced) regarding handling (for e.g. how data is stored or transferred) of data retrieved from DBMS into these files for analysis by stand-alone tools. So, the CIA goals are readily overlooked.

(iv) Moreover, data kept in flat files are just secure as the host operating system, the file permissions or the file/system protecting programs unlike DBMSs which have added advantages of granular access controls, inherent auditing, encryption and event notification.

(v) Finally, we note that there is scanty information on best practices for securing flat data files and avoiding or mitigating the risks of data kept in these files.

These issues often lead to data/information security beaches and so coupling databases with advanced analytical tools is a better secured approach to analytics than using stand-alone analytical tools with flat data files.

Chapter 3

METHODOLOGY

This chapter presents the methodological and scientific approach that was employed in the study. Specifically, it clarifies the design and planning of the practical steps of conceptualisation, implementation and evaluation. It covers the research approach, data collection and analysis techniques used, and how the results were interpreted. The chapter ends with a discussion on reliability and validity techniques employed in the study.

3.1 Research Approach

According to [16], there are three (3) key elements that go into a research approach. These elements, known as *knowledge claims*, *strategies of inquiry*, and *methods*, combine or contribute to determine whether a research approach is mostly *qualitative*, *quantitative* or *mixed*. [16] explains that: stating knowledge claims means starting a research with certain assumptions about what will be learnt and how these will be learnt during the study; the strategies of inquiry refers to the decisions that guides direction for procedures employed in the research design; and the methods refers to the specific means or techniques of collecting and analysing data.

In our study, the choice of *postpositivist* (empirical observation and measurement, and hypothesis verification) knowledge claims, experimental design strategy of inquiry, and methods of measuring/rating/analyses of performance results put our research approach as mainly *quantitative*. [16] defines a *quantitative research approach* as an approach in which the researcher basically resort to postpositivist claims of knowledge development such as test of theories and use of measurement and observation, employs strategies of inquiry like experiments, and collects statistical data with specific or preset tools.

Our choice of *quantitative research approach* in the study was due to the specificity of the goals of the study; the need to collect numeric performance data (quantitative data) from experiments and carry out various kinds of numeric-based analyses. This study started with tools and technologies review by examining relevant materials, especially recently cited materials including technical manuals, articles, white papers, implementation reviews, books, websites, etc from the internet and the library.

3.2 Data Collection and Sample Selection

This was basically done by the collection of *primary data* from the experiments (benchmark performance tests) that have been carried out. Data was collected from the repeated executions of the experiments. Since the study adapted and used the R benchmark test scripts, the data was collected from the execution of tests for Matrix Calculation, Matrix Functions and Program Control. All the resulting performance data from the tests in exception of outliers were sampled or collected for use.

We used input data in the form of two-dimensional data array which is typical of real world data usually analyzed. This data used for the experiment was generated in R such that the data set consists of floating-point numbers of 1,000 observations (columns) by 16,000 variables (rows). We employed a popular *stochastic process* (a representation of evolution of a random variable or system over time) known as *Brownian Motion* [6][5] and used Donsker's theorem to generate the experimental input data (hypothetical stock array) so that its visual representation is similar to stock option prices. The simplicity and clarity of this approach made us opted for it to generate the hypothetical stock distribution option data for the experiment, barring the fact that Brownian Motion is no longer considered as the real distribution of stock options.

Nonetheless, the principle used is very clear and simply explained by [22] as follows;
if Y_i is a sequence of k variables normally distributed as elements or terms;
and X_i a series given as

$$X_i = (Y_1 + \ldots + Y_i) \cdot (\sqrt{\frac{i}{k}})$$

or

$$X_i = \sum_{n=1}^{i} (Y_n) \cdot (\sqrt{\frac{i}{k}}) \tag{3.1}$$

then the series X_i can stand-in for the Brownian Motion.
The code snippets in listing 3.1 on page 35 illustrates how the data generation was realized in R

with the technique.

Listing 3.1: Generation of experimental input data

```
nobs = 1000;  #One thousand observations (e.g. days)
nvar = 16000; #Ten thousand variables (e.g. stocks)

z = rep(0, nobs*nvar);
x <- matrix(z, nrow = nvar, ncol= nobs, byrow=TRUE);

for (j in 1:nvar) {
  set.seed(j);
  y = rnorm(nobs);
  x[j,] = y;

  for (i in 1: nobs){
    x[j, i] = round(  1/sqrt(nobs)*(sum(y[1:i])*sqrt(i)),  2);
  }
}
write.csv(x, file = "E:/stockDnldDir/stockHis3.csv");
```

Thus, from matrix (X) point of view, we had 16,000 rows by 1,000 columns for our data array. The structure of this data array is illustrated by table 3.1. Each of the cell contains floating-point

$matrix X$	obs_1	obs_2	...	obs_{1000}
var_1				
var_2				
var_3				
...				
var_{16000}				

Table 3.1: Data array (matrix or data frame) of generated data set

number for a specific observation (obs_i) of a particular variable (var_j). Therefore, we have a data array made up of a total of 16 million cells ($16,000 \times 1,000$). The same data array was used throughout the analytic performance tests, with each script running the benchmark seven (7) times and the mean values were taken and used for the analyses after eliminating the minimum and maximum run-time values as reported in Appendix B. For the scalability assessment, the tests were also ran on the first one (1) million cells as well as the first four (4) million cells for standalone R and Oracle.

We note that one challenge encountered has to do with limited number of columns (fields) allowed in tables in some of the DBMSs. For instance, the `CRETE TABLE` statement failed in SQL Server when the number of columns exceeded the maximum allowable number of 1024

columns. In the case of Oracle, a maximum allowable number of 1000 columns in a table holds. So, in order to have same *.csv* file data in all the DBMSs used in the test, we decided to use tables (data frame) with 1000 columns which we were able to create in all the DBMSs without problem.

3.3 Data Analysis

The data collected for the various performance test categories were analyzed category-wise. These categories include Matrix Calculation, Matrix Functions and Program Control. This facilitated objective analysis of the performance of the systems that were studied under the various performance categorisations. Various numeric-based (such as min-max normalization/scaling) and graphically-based (like bar/line charting) analytical processes were employed in the analysis of the empirical data to quantify and visualize the performance differences of the systems tested.

3.4 Results Interpretation

The results from the analyses of the empirical data were interpreted with reference to the architectural implementation of R integration with the DMBSs that were studied, and the categories of the tests conducted. The interpretation covers the strengths and weaknesses of the various implementations with regard to respective specific categories of analytic performance and overall analytic performance. Moreover, the interpretation was done in the context of the hypothesis, précising the extent to which it holds, or otherwise. The interpretation concluded by pointing out the strong points or features of R integration implementation, of the best performed database systems, which support optimal or better analytical performance.

3.5 Data Verification and Reliability

Data was collected from several repetitions of each experiment so as to counteract the effect of skewed one-time run or execution. The mean values of the data from the various repetitions were used in the analyses. This helped to ensure reliability. Also, an automated verification (using scripts) of the data were carried out to ensure that outliers (very extreme values) in the result were identified and eliminated before proceeding with analyses. Of course, we only expected outliers in the results to come from a peremptory interruption of the system, causing some variability in the recorded measurements. There is no guarantee of absolute avoidance of this problem

because such is the design of modern systems and a practical work-around is to remove the results of runs that suffer such problem. This was achieved with the aid of an R package named *mvoutlier package* (multivariate outlier detection based on robust methods) [19]. The following code snippet illustrates;

Listing 3.2: Outlier Detection

```
install.packages("mvoutlier");  #install mvoutlier package
library(mvoutlier);             #load the mvoutlier package
data(bmresults);                #load recorded experimental results
uni.plot(bmresults);            #uni.plot(bmresults, symb=TRUE);
```

The other potential issues that could undermine the results during the measurements were adequately catered for by the additional control measures that were instituted in the experimental design and execution as detailed in chapter 4 (section 4.5) of this report.

Chapter 4

EXPERIMENTAL DESIGN

In this chapter titled experimental design, we explain the various benchmark tests and the core and fundamentals of the codes or scripts used to implement these tests. We also describe the empirical settings of the study and how the controlled experiment was setup to guarantee that accurate and required data was collected from the experimental runs for analyses and interpretation to provide answers to our research questions of interest.

4.1 Research Design

The study adopted and adapted a known (community-developed) R benchmark test scripts (*R Benchmark 2.5*) [24], as well as some Revolution Analytics benchmarks [2] to compare performance of integrating R with databases such as Oracle, PostgreSQL, DB2 and SQL Server, as well as with stand-alone R. The benchmark tests covered *Matrix Calculation*, *Matrix Functions* and *Program Control* categories.

The empirical work involved three main levels: *setup*, *execution* and *analysis*. In the setup level, the design and adaptation of the benchmark of the tests were defined. This included declaring the datasets and data types and sizes, test tasks, performance measures and test re-run strategies and number of re-runs. Each script runs the benchmark seven (7) times and the average values were taken and used for the analyses after the elimination of minimum and maximum run-time values as reported in Appendix B. At the execution level the defined setups were executed (ran) and performance data collected. Finally, the performance data collected was analyzed using exploratory and inferential methods in the analysis level.

4.2 The Benchmarks Tests: The different analytic performance tests with R

The benchmark tests employed in this study are adaptations of the generally accepted benchmark tests which are founded on computations which usually form the core part of analytic solutions of many real-world problems. These computations often take greater fraction of compute time and so our capability to speed up these fundamental computations implies faster results for quicker decision making.

In the creation of our tests to measure the analytic performance of databases coupled with R, we adapted *R Benchmark 2.5* community-developed test scripts [24] to give us assessment of general performance of R coupled with databases. The test scripts are grouped into three (3) categories of benchmarks namely *Matrix Calculation* (five tests), *Matrix Functions* (eight tests) and *Program Control* (five tests), according to the characteristics they evaluate. In addition to the individual units of tests in the three test groupings, we also included in the second test group some other additional adopted/adapted R benchmark tests (*Revolution RevoR Enterprise Benchmark*) [2] developed by Revolution Analytics [1] in order to cover as much core of analytics computations as possible for the benchmarking. These tests included *Singular Value Decomposition*, *Principal Components Analysis* and *Linear Discriminant Analysis*. The tests simulate common real-world computations employed in advanced analytics. The following sections give brief descriptions of the tests included in the three benchmark categories which were eventually used for our study.

4.2.1 Matrix Calculation

This test involves performing *Transposition (t)*, *Exponentiation (^)*, *Sorting (sort)*, *Matrix Multiply* or *Cross-Product (crossprod)*, and *Equation Solving (solve)* of our data in a data structure of two-dimensional array (*matrix* or *data.frame*). The following code snippet illustrates the use of commands/functions for the tasks.

Listing 4.1: Matrix Calculations

```
#(i) Transposition:
  y <- t(x);

#(ii) Exponentiation:
  z <- x^y;

#(iii) Sorting:
```

```
  y <- sort(x, method="quick");

#(iv) Matrix Multiply/Cross-Product:
  y <- crossprod(x);  #Or:  y <- t(x) %*% x;

#(v) Equation Solving:
  z <- solve(crossprod(x), crossprod(x,y));
```

The full code used for the implementation of the test to time execution of these tasks are listed in Appendix A.

The Matrix Calculation benchmark group of tests contributes to assess the capacity to carry out some common matrix computations. These computations include matrix creation, transposition and deformation which contribute to evaluate the ability to create and manipulate matrices. The creation of normally distributed random matrix and taking the power of its elements helps to evaluate speed of random matrix processing, element by element. The sorting of random values benchmark contributes by way of testing the speed of sorting operation while matrix cross-product contributes to evaluate matrix multiplication operations. Lastly in this group of benchmark is linear regression over matrix which helps to test speed of evaluation of linear models.

The operations in this benchmark group can be adversely impacted by database integrated with R implementation architectures that entail or rely on transfer of data and R functions from the database to an external (out-of-database) R engine or environment in order to run the R commands for the task or operations such as in the case of SQL Server, DB2, SAP HANA and in some way Oracle (in cases where R commands do not have matching native functions in the in-database statistics engine). The reason for the impact of such architectures on these operations is that the transfer of data and/or input R commands to the external R environment and the return of the results to the database will impose some time lag costs and so such implementations will suffer performance degradation. As such we did not expect as much performance from SQL Server, DB2, and SAP HANA compared to Postgres and Oracle for this benchmark group of tests.

4.2.2 Matrix Functions

The adapted tests from the Matrix Functions benchmark group of the *R Benchmark 2.5* include performing of *Fast Fourier Transform (fft)*, *Cholesky Decomposition (chol)*, *Eigenvalues(eigen)* and *Determinant (det)*. As noted early on, we also extended this benchmark category by including some tests which were adapted from *Revolution RevoR Enterprise Benchmark*. These additional tests include *Singular Value Decomposition (svd)*, *Principal Components Analysis*

(prcomp/princomp) and *Linear Discriminant Analysis (lda)*. The following code snippet shows the commands/functions employed for these tasks.

Listing 4.2: Matrix Function

```
#(i) Fast Fourier Transform:
  y <- fft(x)

#(ii) Eigen Values:
  y <- eigen(x, symmetric=FALSE, only.values=TRUE)$Value

#(iii) Determinant:
  y <- det(x)

#(iv) Cholesky Decomposition/ Factorization:
  y <- chol(x)

#(v) Inverse:
  y <- solve(x) # y <- qr.solve(x) ***A Bit Faster

#(vi) Singular Value Decomposition:
  y <- svd (x,nu=0,nv=0)

#(vii) Principal Components Analysis (PCA):
  y <- prcomp(x)  #P <- princomp(x, cor=TRUE)

#(viii) Linear Discriminant Analysis (LDA):
  y <- lda(x$v1~ x$v2+x$v3, data=x);
```

The complete codes used to realize the tests to time execution of all these tasks are in Appendix A. A quick general check online ([50]) provides clear and simple ideas and insights in to these matrix operations which we explain accordingly as follows:

Cholesky Factorisation of matrix is used to find answers to linear systems of equations with a symmetric positive definite coefficient matrix, to compute correlated sets of pseudo-random numbers, etc. Given a symmetric positive definite matrix M, the Cholesky decomposition is an upper triangular matrix U with strictly positive diagonal entries such that $M = U \wedge (T)U$. Cholesky Factorisation is available in R as a function named *chol()*.

Singular Value Decomposition (SVD) is a matrix factorisation scheme that employs techniques of linear algebra to do decomposition. SVD algorithm is used in data mining to gain better insights into huge datasets by showing simplified and essential dimensions of an otherwise massive dataset. It helps in reducing huge datasets by removing redundant data (attributes) that are linearly dependent on others from standpoint of linear algebra and can thus help to minimise the number of attributes used in data mining. Given an $i \times j$ matrix of real or complex numbers, M, the SVD is factorisation of M into $M = UAVT$ where U is a matrix whose dimensions are

$i \times i$; V is another matrix whose dimensions are $j \times j$; A is a matrix whose dimensions are $i \times j$ (same dimensions as M). Also, $UTU = I_i$ and $VTV = I_j$ where I_i and I_j are identity matrices with sizes of i and j respectively. SVD is implemented in R as an in-built function named *svd()*. The function basically takes R's native matrix as input and output R's data frame composed of U, A and V.

Principal Component Analysis (PCA) is a useful statistical technique which has common application in finding patterns in data of high dimension. The data is expressed in such a way as to highlight similarities and differences therein. This makes PCA a powerful technique for analysing high dimensional data because once patterns in the data are established, the data can be compressed by reducing the number of dimensions, without any significant loss of information. According to [17] and [52], PCA has some associations with k-means clustering that facilitate clustering of data of high dimension by reducing the data to possible representative minimal set of only indispensable or essential dimensions. PCA is accessible in R as a function called *prcomp()* or *princomp()*.

Linear Discriminant Analysis (LDA) is a classification technique employed in advanced analytics to find linear combination of features which characterises or separates two or more classes of objects or events. The result is used to reduce dimension prior to classification or used as linear classifier. LDA has a close relationship with variance analysis and regression analysis, which as well attempt to express one dependent variable as a linear combination of other features or measurements [20][31]. LDA is available for use in R as a function named *lda()*.

The Matrix Functions benchmark group of tests contributes to the evaluation of execution pace of some pre-programmed matrix functions. The Fast Fourier Transform benchmark evaluates the speed of performing Fast Fourier Transform which is largely employed in processing signal information. The benchmark which employs Eigen value computation assists to assess execution of multivariate analyses while the one that involves determinant computation evaluates performance of functions in matrix calculation packages. Cholesky decomposition also contributes to check performance of pre-programmed functions while the Inverse operation benchmark throws in evaluation of computationally intensive functions in Analytics.

The Matrix Functions group of tests can be negatively impacted by database coupled with R implementation architectures which employ separate workload manager address space and load library for R functions as in the case of DB2 where sometimes when the procedures which are not resident in the Workload Manager address space are first loaded from the load library resulting in some overheads and tardiness. As such, we expected better performance of the other DBMSs' execution of R routines compared to that of DB2 for this benchmark category of tests.

4.2.3 Program Control (Programmation)

This benchmark evaluates performance of both implicit and explicit control flow or control structures (recursion, iteration, decision) by using vector calculations (*Fibonacci numbers*), matrix calculation (*Hilbert matrix*), recursive tasks with *Grand common divisors of pairs*, loops performance evaluation with *Toeplitz's matrix* and collective tasks using *Escoufier's method* on matrix. The following code snippet, taken from the *R Benchmark 2.5* [24], shows the use of commands/ functions and control structures employed for these tasks.

Listing 4.3: Program Control

```
#(i) Fibonacci numbers calculation:
y <- (phi^x - (-phi)^(-x))/sqrt(5);

#(ii) Creation of Hilbert matrix:
y <- rep(1:x, x);
dim(y) <- c(x, x);
y <- 1 / (t(y) + 0:(x-1));

#(iii) Grand common divisors of pairs:
z <- gcd2(x, y);   #gcd2 is a recursive function

#(iv) Creation of Toeplitz matrix:
#Intentional timing loop timing (faster alternative exist)
    for (i in 1:500) {
      for (j in 1:500) {
        x[j,i] <- abs(i - j) + 1
      }
    }

 #(v) Escoufier's method on matrix:
 # Calculation of Escoufier's equivalent vectors
    p <- ncol(x)
    vt <- 1:p
    vr <- NULL
    RV <- 1:p
    vrt <- NULL
    for (j in 1:p) {
      Rvmax <- 0
      for (k in 1:(p-j+1)) {
        x2 <- cbind(x, x[,vr], x[,vt[k]])
        R <- cor(x2)
        Ryy <- R[1:p, 1:p]
        Rxx <- R[(p+1):(p+j), (p+1):(p+j)]
        Rxy <- R[(p+1):(p+j), 1:p]
        Ryx <- t(Rxy)
        rvt <- Trace(Ryx %*% Rxy) /
                sqrt(Trace(Ryy %*% Ryy) *
                Trace(Rxx %*% Rxx))
```

```
            if (rvt > Rvmax) {
              Rvmax <- rvt
              vrt <- vt[k]
            }
          }
          vr[j] <- vrt
          RV[j] <- Rvmax
          vt <- vt[vt!=vr[j]]
        }
```

The entire code used for the tests to time the execution of these tasks is listed in Appendix A.

Program Control or Programmation benchmark group of tests contributes by way of helping to assess the speed and efficiency of executing control flows, such as looping, iteration, decision or recursions, when performing analytic tasks or running analytics scripts (as against Matrix Functions which evaluates pre-programmed or customized functions). The Fibonacci numbers and Hilbert's matrix benchmark tests enable assessment of speed of evaluating intrinsic control flows of vector and matrix calculations in R scripts. The Grand common divisors of pairs benchmark contributes to check performance of recursive functions execution while the benchmark which draws on Toeplitz's matrix assesses the speed of loops execution. Finally, the benchmark which employs Escoufier's method contributes to assess performance on collective or grouping of these programming characteristics in one run.

On the issue of performance expectations for this benchmark group with respect to databases' R implementations, the architectures which have execution control stationed in-database but with actual R command execution handled as calls to external R engine will suffer performance problems. In other words, the architectures that employ repeated or recurrent calls (transfer of execution control) between an external (out-of-database) R engine or environment and the database in order to run R commands will underperform. This is because the program control (especially recursions and iterations) test will impose time and stack space costs as a result of managing stack and also the comparative tardiness/delay of the R command execution calls. The consequence of the to-and-fro actions between handling of the tasks execution (by the R engine or environment) and the overall execution control (by the database) will likely amount to overheads in the exchanges and the back and forth communication or synchronisation. As such we expected performance of SQL Server, DB2, SAP HANA and Oracle (to some degree) to be affected. Ordinarily, we did not expect SAP HANA and Oracle to perform exceedingly well, compared to the other DBMSs or stand-alone R, in this group of tests because their capacity to activate multiple R processes to run in parallel cannot be effectively exploited in this case. But the in-database execution of R, especially in the case of Oracle, leaves the performance expectations irresolute.

Table 4.1 shows the benchmark categories and summaries of characteristics assessed by the various tests employed for each category.

Benchmark Category	**Benchmark Function**	**Characteristic Assessed**
Matrix Calculations: Common Matrix Computations	Transposition	Flipping (transpose) computations
	Exponentiation	Exponential computations
	Sorting	Sorting of vector or factor
	Cross-Product	Cross-product computations
	Equation Solving	Solving linear system of equations
Matrix Functions: Computationally-intensive/ Pre-programmed/ Package-functions	Fast Fourier Transform	Fourier Analysis/Transform
	Eigen Values	Multivariate analysis
	Determinant	Matrix calculation with pre-programmed function
	Cholesky Decomposition/ Factorisation	Pre-programmed function
	Inverse	Computationally intensive function
	Singular Value Decomposition	Matrix factorization/ decomposition (rotation & scaling)
	Principal Components Analysis	Finding pattern/data reduction/ orthogonal transformation
	Linear Discriminant Analysis	Dimensionality reduction/ linear classification
Program Controls: Custom Function (Recursions, Iterations, Decisions)	Fibonacci numbers calculation	Inherent control flow of vector calculation
	Creation of Hilbert matrix	Matrix calculation in scripts
	Grand Common Divisors of Pairs	Recursive computations
	Creation of Toeplitz Matrix	Loops execution
	Escoufier's Method on Matrix	Collective/grouped programming characteristics

Table 4.1: Summary of characteristics assessed by the benchmarks

4.3 Computing Resource Configurations

Table 4.2 details the configuration of the computing resources employed to carry out the benchmark tests.

Resource	Configuration
Processor Brand	Intel (model F1X66EA#ABU)
Processor Type	Intel Core i7
Processor Speed	2.4 GHz
Processor Count	4
RAM Size	8 GB
Computer Memory Type	DDR3 SDRAM
Hard Drive Size	1 TB
Operating System	Windows 8 (64-bit)
Brand	Hewlett Packard
Virtual Memory	8192MB/8GB

Table 4.2: Details of computing configurations

4.4 Performance Results Measurement

Basically, there are two (2) R functions for timing codes and these are `proc.time()` and `system.time()`.

The `proc.time()` function measures the amount of time (in seconds) that a currently running R process has already taken in executing. In order to use this function, one applies the notion of a stop-watch by first taking note of the start time, running all the codes being timed, and then recording the end time and deducting the start time from the end time. The result of the difference between the start and end times gives the time taken to execute the codes. The following code snippet illustrates a simple use of `proc.time()` to measure the time R takes to execute some codes:

Listing 4.4: Timing with proc.time()

```
x <- rnorm(500000)
y <- rep(NA, 500000)

#Start the watch!
```

```
ptm <- proc.time()
y <- x * 5

#Stop the watch!
proc.time() - ptm

user    system    elapsed
0.00    0.02    0.02
```

The results are presented as three time values under the headings *user*, *system*, and *elapsed*. The user and system times are respectively measures of total user and system CPU times of the running R process and any child processes on which it waited. The elapsed time is the actual elapsed time since the process was started. It is the difference in times since the start of the timing clock. This is equal to the sum of user and system times if the chunk of code was run altogether. The values for user and system times are defined by the operating system (OS) of the system on which the code was run. According to [37], *User time is the CPU time charged for the execution of user instructions of the calling process whiles System time is the CPU time charged for execution by the system on behalf of the calling process.*

The `system.time()` function takes as argument R expressions and an optional logical value (with default value of `TRUE`) which indicates whether garbage collection should be performed just before the timing or not. The `system.time()` function returns the times an expression (which is passed as argument) takes to execute. In order to use `system.time()` to time codes, one may have to write function wrappers around the routines or methods being timed and pass the function as an argument to the `system.time()`. As stated early on, `system.time()` merely calls `proc.time()`, evaluates the expression being timed, then calls `proc.time()` again and then returns the difference between the results of the two `proc.time()` calls. The following snippet of codes demonstrates the use of `system.time()` to time codes.

Listing 4.5: Timing with system.time()

```
x <- rnorm(500000)
mymult <- function(x){
  return(x*5)
}

system.time(y <- mymult(x))

user    system    elapsed
0.02    0.00    0.01
```

The results of `system.time()` has the same meaning as discussed for `proc.time()`.

But, as can been seen from the code listing, the `system.time()` superficially differ from `proc.time()` in that there is no need to explicitly record the start and end times, and then find the difference between these times to get the time taken to execute codes. All these are automatically handled by the function unlike the case of `proc.time()` where one have to explicitly take care of these. The preference of use of either one of the two functions over the other basically depends on the nature of the code being timed. In our evaluation (analytic performance benchmark tests), we employed `system.time()` throughout the evaluation because it allowed us to put collections of code lines written for the various benchmarks into blocks and just wrapped it around the code blocks to access the performance metrics. It also made writing, debugging and enhancement updates of our codes clearer and simpler as we have very clear segregation of our objective; focus on writing the codes for the analytic tasks first and thereafter the timing or evaluation of the codes for the tasks.

4.5 Experimental Controls

Since the validity of the experiment is directly affected by its construction and execution, we attached extreme importance to the design and execution of the experiment using a number of control measures. These measures are explained as follows:

Regarding the use of `system.time()` to measure the performance of the various operations or routines of the benchmark scripts, the timing evaluation of the same expression can vary significantly depending on whether the evaluation activates a garbage collection or not. In order to surmount this problem we made sure that our test environment was in a clean state before timing the test codes. This was done by using a function called `gc()`, garbage collection. We were also aware of the fact that the default for `system.time()` function is to call garbage collection prior to evaluation of an expression, but since we had already explicitly called `gc()`, we did not state it again in the call to `system.time()` for timing our codes. For example;

Listing 4.6: Calling garbage collection function, gc()

```
invisible(gc());
exec_time <- system.time({
a <- t(b + c);
})[3]
```

This forces garbage collection (`gc()`) to be carried out just before the timing evaluation of the expression.

We avoided experimental bias, the favouring of certain outcomes over others, and made our

inferential task easier by limiting the number of factors at play, using exactly the same computing resource configuration for the various DBMSs tested. In more concrete terms, we maintained the same amount of computing resources and environment in carrying out the tests on all the DBMSs. Thus we controlled configuration variables such as RAM, Speed, OS, HDD, etc. This provided levelled grounds to compare and contrast results from the tests.

Also, to avoid effect of one-off extreme results on the overall outcome of the study, several timing runs were performed for each type of experiment on the DBMSs and the mean values used in the analyses. This is because systems can be peremptorily interrupted and so it is important to carry out many runs to check for variability and nullify differences to improves overall measurements. This helped to ensure reliability and objectivity as any effect of slight fluctuation in results from one run to another is nullified by the average results.

Lastly, we employed deletion of temporary objects and objects that were no longer needed during each re-run. This was achieved by calling *rm("obj_1", "obj_2")* to remove specific objects (obj_1, obj_2) from memory or *rm(list=ls())* which removes all objects from memory, effectively providing a clean state for the runs. Also, *daemons* running on the experimental system were stopped to ensure that they do not unjustifiably compete for and share in the resources that were available to the experimental setup/environment.

The use of these fundamental control mechanisms in our experimental design and execution effectively ensured elimination and mitigation of potential disparities that may have arisen in the deployment and running of the benchmark and removed both expected or unexpected problems.

4.6 Setting up and Configurations of Test Environment

The following are some details about the setup and configuration of integrating R with the four (4) DBMS evaluated in the analytic performance benchmark testing.

4.6.1 R + Oracle Experimental Setup

With reference to the guides *Oracle Database Installation Guide* [32], *Oracle R Enterprise Installation and Administration Guide* [33] and *Oracle R Enterprise User's Guide* [34], the following tasks were performed in order to get Oracle R Enterprise (ORE) server up and running on the experimental machine:

(i) Oracle Database 12c (64-bit) was installed.

(ii) Open Source R 2.13.2 was then installed.

(iii) Then Oracle R Enterprise Server 1.1 was installed.

(iv) Thereafter, Oracle R Enterprise Client Supporting Packages
`(ROracle_1.1-9, DBI, png_0.1-4)` were installed.
The packages are installed in `$ORACLE_HOME/R/library`.

Configuring extproc for embedded R execution:
The default configuration of `extproc`, for use by Oracle R Enterprise for embedded R execution, was maintained and so no changes were made to either `listener.ora` or `tnsnames.ora`. The environment variables in table 4.3 were set.

Variable	Value
R_HOME	`C:\Program Files\R\R-2.13.2`
PATH	`%ORACLE_HOME%\bin`
ORACLE_SID	`orcl`
ORACLE_HOME	`E:\app\Orauser\product\12.1.0\dbhome_2`
R_LIBS_USER	`E:\app\Orauser\product\12.1.0\dbhome_2\R\library`

Table 4.3: Environment variable settings for ORE

Note:
`%ORACLE_HOME% = E:\app\Orauser\product\12.1.0\dbhome_2`
`%R_HOME% = C:\Program Files\R\R-2.13.2`

After the installation, the following post-installation steps were carried out:

Creation of ORE user:
A user, `RQUSER`, for Oracle R Enterprise was created by running the script `(demo_user.bat)` that came with the Oracle R Enterprise server.

Listing 4.7: Creating ORE User

```
> demo_user.bat
```

Granting of privileges to the ORE user:
Some essential privileges (`CREATE TABLE`, `CREATE PROCEDURE`, `CREATE VIEW`, `CREATE MINING MODEL`) were granted to the user (`RQUSER`).

Listing 4.8: Granting privileges to the ORE user

```
SQL> GRANT CREATE TABLE TO RQUSER;
SQL> GRANT CREATE PROCEDURE TO RQUSER;
SQL> GRANT CREATE VIEW TO RQUSER;
SQL> GRANT CREATE MINING MODEL TO RQUSER;
```

Granting of RQADMIN Role to ORE user:
The `RQADMIN`, a role created by the Oracle R Enterprise server installation was granted to the `RQUSER`. This role is required by a user in order to be able to execute embedded R.

Listing 4.9: Granting RQADMIN role right

```
SQL> GRANT RQADMIN TO RQUSER;
```

4.6.2 R + Posgres Experimental Setup

We used PL/R which is the driver that implant the R interpreter in Postgres to facilitate in-database R execution. With the help of the *PL/R Users Guide* [15], we carried out the following setup tasks on the experimental machine:

(i) Installation of PostgreSQL 9.3.2 Database (64-bit).

(ii) Installation of Open Source R 2.13.2.

The environment variables in table 4.4 were set.

Variable	**Value**
R_HOME	`C:\Program Files\R\R-2.13.2`
PATH	`C:\Program Files\R\R-2.13.2\bin`

Table 4.4: Environment variable settings for Postgres PL/R

Afterwards, the PostgreSQL Server service was restarted on the experimental system in order for the changes to take effect and then the following post-installation procedures were carried out:

Language Installation into Database:

Listing 4.10: PL/RLanguage Installation

```
--Language Installation (manually) into Database
CREATE FUNCTION plr_call_handler()
RETURNS LANGUAGE_HANDLER
AS '$libdir/plr' LANGUAGE C;

CREATE LANGUAGE plr HANDLER plr_call_handler;
```

Loading PL/R functionality into the database:
In order to enable the use of PL/R in PostgreSQL database, we created language and support functions in the PostgreSQL database and then loaded the help functions into the database. The PL/R was enabled in our database by running the following command on the database.

Listing 4.11: Enabling PL/R functionality in database

```
CREATE EXTENSION plr;
```

Testing creation and calling of PL/R functions:
The following helper functions were created and tested.

Listing 4.12: Testing PL/R functionality

```
--Empty body function: function name same as R function
CREATE OR REPLACE FUNCTION paste(text, text)
RETURNS text[] AS '' LANGUAGE 'plr';
SELECT paste('Hello','World');

--Non-empty body function: function name not same as R function
CREATE OR REPLACE FUNCTION mufxn()
RETURNS float[] AS
'
n<-c(1:10);
m<-mean(n);
s<-sd(n);
res<-c(mean=m, stddev=s);
res<-as.matrix(res);
res;
'
LANGUAGE 'plr';
SELECT mufxn();
```

4.6.3 R + DB2 Experimental Setup

The following tasks were carried out on the experimental machine in setting up DB2 to work with R:

(i) Installation of IBM DB2 Express-C 10.5 (64-bit)

(ii) Installation of Java (64-bit)

(iii) Installation of Open Source R 2.13.2. [RGUI environment (64-bit client)]

(iv) Installation of RJDBC package from RGUI

Listing 4.13: Installing RJDBC Package

```
> install.packages("RJDBC");
```

After the installation, testing of RJDBC loading and connecting to the database to retrieve data was performed as follows:
R was started and the package was loaded:

Listing 4.14: Loading RJDBC Package

```
> library(RJDBC);
```

Then the DB2 JDBC driver was loaded:

Listing 4.15: Loading DB2 JDBC Driver

```
  > jcc = JDBC("com.ibm.db2.jcc.DB2Driver",
"C:/Program Files/IBM/SQLLIB/java/db2jcc4.jar");
```

A database connection was then established:

Listing 4.16: Loading RJDBC Package

```
> channel = dbConnect(jcc,
"jdbc:db2://localhost:50000/SAMPLE",
user="sedem", password="********");
```

A test query was run on a sample data (selecting all records from the `stockHist` table) in the DB2 database and the results put into a data frame and then output to the console:

Listing 4.17: Test query on data

```
> rst = dbSendQuery(channel, "SELECT * FROM stockHist");
> dfr = fetch(rst, -1);
> dfr;
```

4.6.4 R + SQL Server Experimental Setup

The following tasks were carried out on the experimental machine in setting up SQL Server to work with R:

(i) Installation of SQL Server 2012 (64-bit).

(ii) Installation of Open Source R 2.13.2. [RGUI environment (64-bit client)]

(iii) Installation of RODBC package from RGUI

Listing 4.18: Installing RODBC Package

```
> install.packages("RODBC")
```

After deciding on the 64-bit client, we configured an ODBC DSN, `rConnectSqlServer`, to connect to the database using *SQL Native Client 11.0* driver. Then testing of RODBC loading and connecting to the database to retrieve data was performed as follows:

Listing 4.19: Loading RJDBC Package

```
> library(RODBC)
> channel <-odbcConnect("rConnectSqlServer ", uid="sa", pwd="**********")
> dataArray <- sqlQuery(channel, "SELECT * FROM stockHist")
> head(dataArray)
> close(channel)
>
>
> library(RODBC)
> channel<-odbcConnect("rConnectSqlServer")
> dataArray<-sqlFetch(channel,'stockHist')
> head(dataArray)
> odbcClose(channel)
```

SQL Server can also run R via Common Language Runtime (CLR) Integration. To achieve this, the following tasks were carried out to enable use of SQL scripts that indirectly run R with the initiation from the database: We created the usual R script files required to be ran. Then we wrote a high level language (C# CLR) utility program that calls R (by making use of *R.NET library* [36]) and pass R commands to it for execution (it used `source()` command and passed the full path of the R script file). The program was then built/compiled to get the corresponding dynamic link library (.dll file). Afterwards, the CLR integration feature of the SQL Server was enabled. Again, in the SQL Server database, an assembly was created from the DLL obtained. Finally stored procedures were created based on the assembly and SQL queries that make use of the stored procedures were written and ran with the name of the R script files as input parameters. Likewise, these same tasks are applicable to DB2 in case R needs to be run via Common Language Runtime (CLR) Integration. But in the case of DB2, the CLR assembly is kept outside the database away from the stored procedure definitions located in-database.

The approach of passing R script file as a parameter in the SQL queries has a number of advantages which include the following:

(i) One can run or modify the R script file without having to recompile the C# utility program for a new a DLL, which will require update of the assembly and related stored procedures, as long as the name remains same

(ii) There is no need to struggle with the laborious and error-prone task of stringing all the R commands in the C# utility program that creates the needed DLL.

(iii) The process that runs the code (R script) of the stored procedure runs in the SQL Server process space.

(iv) It allows for portability of R codes as we do not need to modify existing R script files in order to run them from database.

Chapter 5

EMPIRICAL FINDINGS

In this chapter, the primary results from the experiments are presented. Other (secondary) results and how they were computed from the primary results are also presented in this chapter while the next chapter analyzes the evidence thereof either in agreement or otherwise with the thesis statement declaration.

The table 5.1 on page 58 provides records of the average execution times for the various benchmark categories employed in the study as discussed in section 4.2 (sub-sections 4.2.1, 4.2.2 and 4.2.3) on pages 40 through 44. However, the details of the execution times for each of the individual benchmark tests in the three categories by the various DBMSs are captured in Appendix B (B.1, B.2 and B.3). In table 5.1 and other tables of results, the *MCxx* are the Matrix Calculations benchmark results, while the *MFxx* are the Matrix Functions benchmark results and the *PCxx* are the Programmation or Program Control benchmark results. In specific terms, the following are the representations of the tests; *MC01*: Transposition; *MC02*: Exponentiation; *MC03*: Sorting; *MC04*: Matrix Multiplication/ Cross-Product; *MC05*: Equation Solving; *MF01*: Fast Fourier Transform; *MF02*: Eigen Values; *MF03*: Determinant; *MF04*: Cholesky Decomposition/ Factorization; *MF05*: Inverse; *MF06*: Singular Value Decomposition; *MF07*: Principal Components Analysis; *MF08*: Linear Discriminant Analysis; *PC01*: Fibonacci numbers calculation; *PC02*: Creation of Hilbert matrix; *PC03*: Grand common divisors of pairs; *PC04*: Creation of Toeplitz's matrix; *PC05*: Escoufier's method on matrix.

In order to obtain good overall reflection of the results, we employed a simple application of statistical concept of *min-max normalisation*. We normalized the results from the various benchmarks before aggregating them. This was done using a simple but robust approach of scaling the values by linear transformation of the form $f(x) = ax + b$, thereby putting the different

Benchmark	Stand-alone R	PostgreSQL	Oracle	DB2	SQL Server
MC01	9.65	18.62	0.44	3.86	3.71
MC02	14.86	23.95	5.88	9.01	8.90
MC03	10.17	19.40	1.26	4.40	4.25
MC04	16.54	25.82	7.71	10.55	10.68
MC05	57.79	77.22	40.97	48.69	46.15
MF01	12.45	22.70	4.15	8.09	6.71
MF02	100.98	110.94	100.49	95.27	94.91
MF03	19.21	28.51	10.34	13.4	13.46
MF04	48.50	57.97	40.13	42.79	42.53
MF05	58.79	68.68	51.73	53.54	53.16
MF06	67.7	76.90	59.45	61.59	61.65
MF07	215.18	226.36	212.74	213.41	209.26
MF08	47.61	57.75	39.46	42.50	43.06
PC01	2.71	2.78	2.77	2.70	2.77
PC02	0.30	0.40	0.31	0.28	0.29
PC03	0.63	0.60	0.43	0.65	0.66
PC04	0.51	0.50	0.51	0.53	0.51
PC05	0.38	0.38	0.36	0.38	0.38

Table 5.1: Average runtime results (time in seconds)

ranges for the various benchmarks into one scale and giving a notionally common scale. This made adding up of the various results meaningful since the effects of certain gross or absolute influences are removed. To put the transformation, $f(x) = ax + b$, in concrete terms, we used

$$X_{norm} = N_{min} + \frac{(X_{dat} - D_{min})}{(D_{max} - D_{min})} \times (N_{max} - N_{min}); \tag{5.1}$$

where:
X_{dat} is the value being normalized;
D_{min} is the lower bound of result range;
D_{max} is the upper bound of result range;
N_{min} is the lower bound of normalized range (pegged at 1);
N_{max} is the upper bound of normalized range (pegged at 10);
X_{norm} is the result of the value normalized (normalized value).

For example, to normalize SQL Server's average run-time value for the test *MC01*, we com-

puted it using the following values as input:
$X_{dat} = 3.71$;
$N_{min} = 1$;
$N_{max} = 10$;
$D_{min} = 0.44$;
$D_{max} = 18.62$;
$X_{norm} =?$
Substituting these values into our value normalisation equation 5.1 gives us

$$X_{norm} = 1 + \frac{(3.71 - 0.44)}{(18.62 - 0.44)} \times (10 - 1)$$

$$X_{norm} = 2.61883 \approx 2.62$$

which is given as the normalized value of the *MC01* benchmark result for SQL Server in table 5.2. This normalized values table (table 5.2) gives the complete results of normalizing the average run-times obtained (refer to table 5.1 on page 58).

Benchmark	Dmax	Dmin	Nmax	Nmin	Stand Alone R	Postgre SQL	Oracle	DB2	SQL Server
MC01	18.62	0.44	10.00	1.00	5.56	10.00	1.00	2.69	2.62
MC02	23.95	5.88	10.00	1.00	5.47	10.00	1.00	2.56	2.50
MC03	19.40	1.26	10.00	1.00	5.42	10.00	1.00	2.56	2.48
MC04	25.82	7.71	10.00	1.00	5.39	10.00	1.00	2.41	2.48
MC05	77.22	40.97	10.00	1.00	5.18	10.00	1.00	2.92	2.29
MF01	22.70	4.15	10.00	1.00	5.03	10.00	1.00	2.91	2.24
MF02	110.94	94.91	10.00	1.00	4.41	10.00	4.13	1.20	1.00
MF03	28.51	10.34	10.00	1.00	5.39	10.00	1.00	2.52	2.55
MF04	57.97	40.13	10.00	1.00	5.22	10.00	1.00	2.34	2.21
MF05	68.68	51.73	10.00	1.00	4.75	10.00	1.00	1.96	1.76
MF06	76.90	59.45	10.00	1.00	5.26	10.00	1.00	2.10	2.13
MF07	226.36	209.26	10.00	1.00	4.12	10.00	2.83	3.18	1.00
MF08	57.75	39.46	10.00	1.00	5.01	10.00	1.00	2.50	2.77
PC01	2.78	2.70	10.00	1.00	2.12	10.00	8.88	1.00	8.88
PC02	0.40	0.28	10.00	1.00	2.50	10.00	3.25	1.00	1.75
PC03	0.66	0.43	10.00	1.00	8.83	7.65	1.00	9.61	10.00
PC04	0.53	0.50	10.00	1.00	4.00	1.00	4.00	10.00	4.00
PC05	0.38	0.36	10.00	1.00	10.00	10.00	1.00	10.00	10.00

Table 5.2: Normalized-average runtime results (normalized to range [1,10])

Table 5.3 gives the aggregated results of the average run-time benchmark results after the normalization. Thus, for each of the three benchmark category, we sum up the normalized values of the results of the individual tests for respective systems (stand-alone R and R integrated with DBMSs).

Benchmark	Stand-Alone R	PostgreSQL	Oracle	DB2	SQL Server
MC	27.02	50.00	5.00	13.14	12.37
MF	39.18	80.00	12.96	18.72	15.66
PC	27.45	38.65	18.13	31.61	34.63
OVERALL	***93.65***	***168.65***	***36.09***	***63.46***	***62.66***

Table 5.3: Overall normalized average run-time results

The results are also presented in charts as shown in figures 5.1, 5.2, 5.3 and 5.4 on pages 60 through 63 for easy and clear observation of the performance differences among the various systems assessed for the benchmark categories.

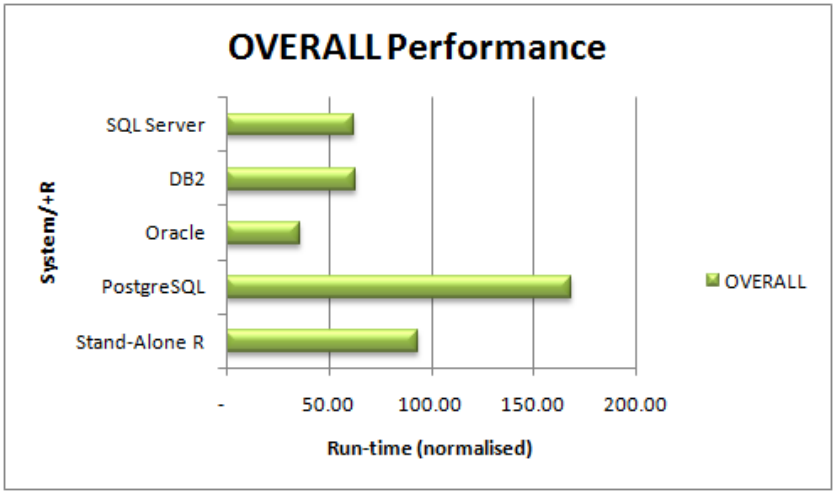

Figure 5.1: Chart of overall benchmark results (average run-times)

The chart in figure 5.1 shows the average overall analytical performance of the various systems tested. Clearly, Oracle's performance lead all the others with a particularly impeccable performance compared to PostgreSQL and stand-alone R. The overall performance of PostgreSQL was the worst which was quite unexpected. The performance of SQL Server and DB2 come next after Oracle's, with SQL Server having a slight edge over DB2. The overall performance of

stand-alone R trailed the overall performances of the DBMSs in exception of PostgreSQL which is quite startling. This development is discussed further in chapter 6 to identify the possible causes of this result.

Having seen the overall performance of the various systems, we delve a little deeper to examine the various benchmark categories that constitute the overall result. We therefore, present the respective charts of the three benchmark categories as follows:

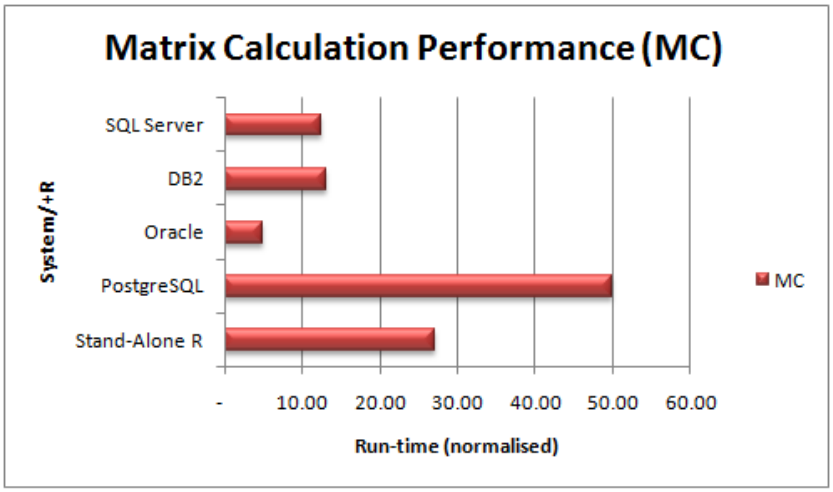

Figure 5.2: Chart of Matrix Calculation (MC) benchmark results (average run-times)

As reflected by the results in table 5.1 on page 58, it is clear that in the first benchmark category, Matrix Calculations (MC), Oracle performed very well leading in all the five tests. This is followed by SQL Server and DB2 which have about the same performance results, but with SQL Server having a slight edge over DB2. It can be seen that the performance of PostgreSQL trailed in all the five tests in this category. Stand-alone R performed poorly compared to the others but better than PostgreSQL. More discussions on these results are made in section 6.1 of chapter 6. It is interesting to also note that the performance of Oracle compared to stand-alone R ranges from about *1.4* to *21.9* times faster in the various tests and from *1.8* to *42.3* compared to PostgreSQL. Figure 5.2 shows the overall performance ranking in this category with Oracle outpacing the rest while PostgreSQL trailed all the others.

In the second benchmark category, Matrix Function (MF), Oracle maintained performance lead in most of the tests (six out of the eight tests- MF01, MF03, MF04, MF05, MF06 and MF08) but with a snag performance in two of the tests (MF02 and MF07) which saw SQL Server leading in those two. Further discussions on these results are advanced in section 6.2

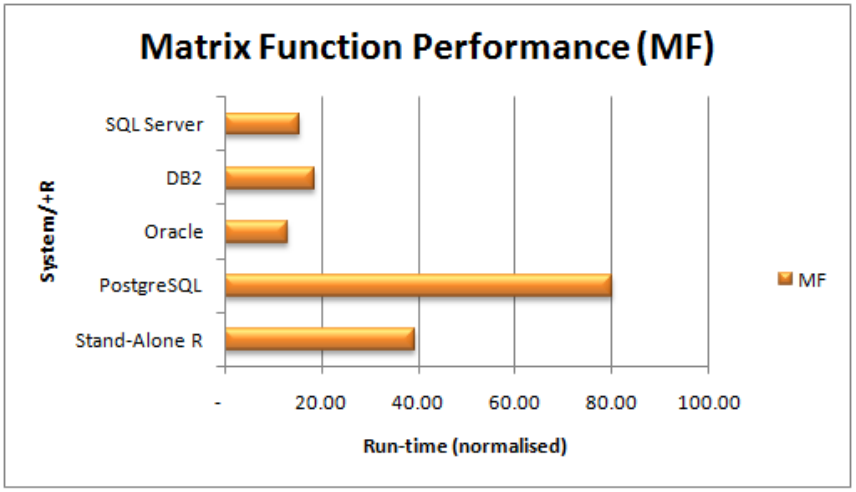

Figure 5.3: Chart of Matrix Function (MF) benchmark results (average run-times)

of chapter 6. It is remarkable to observe that the performance of Oracle in two of the tests (MF01 and MF03) is approximately two and three times faster than stand-alone R while same compared to PostgreSQL is about three and five times faster. Again, SQL Server and DB2 have comparable performances except in those couple of tests that SQL Server led the rest. The general performance positioning in this benchmark category is shown in the chart in figure 5.3 which shows Oracle lead but with close contention from SQL Server.

In the last category of benchmark, Program Control (PC), Oracle lead in the overall test. This is followed by stand-alone R, outperforming DB2, SQL Server and PostgreSQL. Once more, SQL Server and DB2 have about the same performance. One significant observation in this benchmark category is that there seems not to be very clear-cut performance differences by the various systems. There are mixed performances from the various systems and PostgreSQL also lead in one of the tests (PC04). This test, PC04, is a looping program control test using Toeplitz matrix and does not involve passing of database resident data for analytics which would have otherwise caused hitch in PostgreSQL's performance. Moreover, there is no need for parallelism when looping and so PostreSQL performed well. We discuss these results in section 6.3 of chapter 6. Figure 5.4 on page 63 gives a graphical view, in a bar chart, of the overall performance results of this benchmark category.

In testing analytics execution scalability, where we ran the benchmarks for different amounts of analyzed datasets (1 million cells, 4 million cells and 16 million cells), we observed that increases in execution time for routines with increased data are smaller for Oracle+R compared

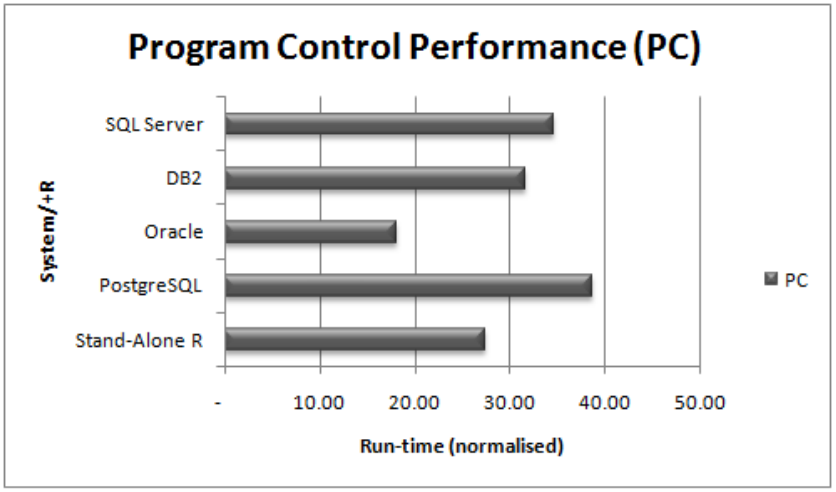

Figure 5.4: Chart of Program Control (PC) benchmark results (average run-times)

to stand-alone R. The results are found in Appendix B.7 on page 120. This means that R coupled with DBMSs perform better than stand-alone R when it comes to analytics execution scalability. It is also important to note that stand-alone R simply does not work for very huge that datasets.

Regarding the results of the experiments, there is one observation worth mentioning. We realized that the results obtained with *average run-times* is about the same as that obtained when *minimum run-times* were used. In other words, the performance patterns recorded remain exactly the same while there are no significant variations in the actual recorded values. This indicates the robustness of the experimental design and the running of the tests in the controlled environment because reliability of the figures of the results from the experimental runs is high.

Chapter 6

DISCUSSIONS ON THE FINDINGS

In this chapter, discussions on the empirical findings are presented and the implications emanating from the findings are also put forward. The discussions involve analyses of the empirical results, as presented in the previous chapter (chapter 5), to unearth the extent of agreement or otherwise with the thesis statement declaration (hypothesis). We proceed to examine and discuss the findings from the benchmark categories as follows:

6.1 Matrix Calculation (MC) Results

The overarching impressive performance of Oracle is understandable as Oracle first finds matching/equivalent functions to use in-database. For instance, the transparency framework and statistics engine support several common operators and functions including *matrix multiplication*, *cross-product*, *solve* and *exponentiation*. Since this benchmark category basically involves common functions, it is obvious that the native equivalent functions are implemented in the Oracle's in-database statistic engine, hence the tremendous performance. Even in cases where there are not matching native functions, Oracle has the capacity to spawn multiple R engine sessions for parallel execution of tasks [7]. The performance of SQL Server and DB2 are as expected and the slight lead by SQL Server can be explained to have been caused by the advantage of storing the CLR assemblies with the embedded R routines in-database as opposed to a totally out-of-database R routines execution (or storing of CLR assemblies with embedded R routines in file

system) in the case of DB2.

The disagreeable surprise performance came from PostgreSQL which saw a deteriorated performance compared to even stand-alone R. This result triggered further investigations to find out exactly what accounted for the worsen performance. Some database configuration parameters (especially, the *shared_buffer*) tweaking (changing default value of 128MB through the values 256MB, 512MB and 1024MB) were undertaken which resulted in insignificant performance improvement. This was not enough to even be on a par with the performance of stand-alone R. The *shared_buffers* configuration parameter sets the amount of memory the database server uses for shared memory buffers. It determines how much memory is dedicated to PostgreSQL to use for caching data.

Our investigation revealed that when a task does not require significant amount of data to be passed to the R functions, such as in the case of PC benchmark category, PostgreSQL performed competitively well, almost as Oracle. Further experiment to confirm or refute this finding was carried out. This additional experiment involves timing only the retrieval of database resident data as matrix (*codes of the experiment listed in Appendix A.6*) and the result, as presented in the table 6.1, corroborate the finding of first investigation.

Retrieving DB Data	**Run-1**	**Run-2**	**Run-3**	**Run-4**	**Run-5**	**Run-6**	**Run-7**	**Total**	**Average**
Oracle	0.15	0.14	0.14	0.14	0.14	0.15	0.14	**1.00**	**0.14**
PostgreSQL	20.12	19.05	19.12	19.12	19.03	19.11	19.12	**134.67**	**19.24**

Table 6.1: Timing of retrieving database resident data (time in sec)

Thus, it takes the database integrated R far less time to retrieve the data in Oracle than in PostgreSQL. This therefore implies that for optimum performance, the manner and technique used to access data when databases are coupled with advanced analytical tools is so crucial that if not handled well, there could be a weaken performance.

For the purpose of benefit of hindsight, it is worth stating that one of our challenge was the initial thought that the observed depreciated performance of PostgreSQL was probably because of the obnoxiously wide (1000 columns) nature of the table hosting the input data to be passed to the analytic functions. This is because while such wide tables may be pinned in *RAM Cache* for faster data retrieval by some database systems, others may not, making them slow on data access. *RAM Cache* (also known as *Cache Store*, *Memory Cache* or *L2 Cache*) is basically a high-speed memory or storage mechanism made of static RAM (SRAM) and offered in CPUs in order to speed up access to data and instructions stored in RAM.

However, additional investigation revealed that the time taken to directly fetch rows from

the wide table (`SELECT * FROM stockHist`) in PostgreSQL using *psql* is more than double (2.66 times) the time taken to do same in Oracle using *sqlplus*. This implies that the poor performance of PostgreSQL-coupled-R during the analytics using the database resident data is not exclusively the consequence of the PostgreSQL-coupled-R's implementation but also the database itself (data retrieval). This implication was further corroborated as we also found out that keeping less number of columns (10) but more rows (essentially the same number of cells) produces similar snag performance, dispelling the initial idea that the problem was being caused by the wide nature of the table.

6.2 Matrix Function (MF) Results

The difference in performance margins between Oracle on one hand and SQL Server and DB2 on the other hand is not as big as in the case of matrix calculations benchmark category of tests. This could be attributed to the fact that the tests here involve invocation of package functions or pre-programmed and custom functions which are not likely to have matching/equivalent native functions in Oracle. Thus, Oracle, SQL and DB2 will be using similar techniques of executing the functions in out-of-database R engine but with Oracle having an advantage of its *rqTableEval* command, used in the Oracle benchmark. This is because the *rqTableEval* command enables invocation of R script/function with entire table provided as input all at once, and also ensures efficient data transfer between the database and the spawned R engines [34]. Also, these performance improvement gaps compared to stand-alone R are a bit less. This is due to overheads generally associated with loading appropriate analytic function libraries before function execution and/or use of out-of-database R environment. So, part of the total performance gains obtained by the DBMSs compared to R are eroded, hence the less performance gaps. Yet, the poorer performance of PostgreSQL can be attributed to this overhead of out-of-database execution of custom-made R tasks by the DBMSs, coupled with the PostgreSQL's revealed issue (through the experiment of timing just the retrieval of database resident data as matrix listed in Appendix A.6) of inefficient retrieval and passing of data, as previously discussed during the case of matrix calculation results in the latter part of section 6.1.

6.3 Program Control (PC) Results

The program control category of benchmark does not involve any significant amount of data and so it was not unexpected when we had stand-alone R performed better or equally well. The implication of this observation is that for less data intensive analytics, stand-alone R still remains

a viable and competitive option, barring other concerns such as data security. Furthermore, this benchmark category somewhat confirms our previous analysis that the unexpected poor performance from PostgreSQL was partly due to the inefficient retrieval and passing of data to the R analytic functions. This is because PostgreSQL also performed competitively well in this benchmark category as compared to its performance in the other two benchmark categories which involve analytic operations on data stored in tables within the database.

Oracle's unmatched performance in the other two benchmark groups is of course due to the architectural arrangement which makes use of in-database statistics engine and multiple spawning of R sessions. But in this benchmark (PC) category, the tests were mainly about program control (loops and recursions, no significant data involved) for which there is no actual need for in-database analytical engine and parallel execution. Therefore PostgreSQL's R implementation and even stand-alone R, both of which do not have such performance enhancing Oracle's R implementation features, were able to perform equally well because those features could not be exploited in this group of tests.

In order to confirm that the performances by the various systems in this benchmark category are not significantly different, a *paired t-test* was performed using the two extreme result sets (Postgres' and Oracle's) for the benchmark category. The mean performance difference $(M = 0.06, SD = 0.074, N = 5)$ was not significantly greater than zero, $t(4) = 1.69$, two-tail $p = 0.166$, providing evidence that there is no notable or considerable difference in the performances of the two DBMSs. A 95% C.I. about mean performance difference is $(-0.036, 0.148)$. Hence, given the little variability in the performance results, the performance difference could have been just as low as an average of 0.036 seconds to a high of 0.148 seconds (see the 95% confidence interval) which are not substantially apart. So, in essence, the different architectural arrangements of R implementation by the database systems virtually have no impact on performance as far as *programmation* (program control) of analytical processing is concerned.

6.4 Overall Results

On the whole, Oracle's performance lead all the others. This is then followed by performance from SQL Server, DB2, Stand-alone R and PostgreSQL in the respective orders listed. Owing to the closeness of the performance results of DB2 and SQL Server, we again conducted a *paired t-test* to establish if there is significant difference in their overall performances. The mean performance difference $(M = 0.48, SD = 1.143, N = 18)$ was not significantly greater than zero, $t(17) = 1.77$, two-tail $p = 0.094$, providing indication that there is no significant difference in the performance of the two DBMSs. The disappointing performance from PostgreSQL was

mainly due to data accessibility issues. This means that the techniques for making data available to analytic functions/routines play critical role as far as in-database analytics performance is concerned.

Database resident data retrieval/transfer techniques can make or break performance and so special effort must be directed towards best possible methods. For instance, retrieving and passing of entire table of data at one go as in the case of Oracle produces better results. Ideally, providing support for data-parallel processing (i.e. multiple partitioning of data so that the subparts are separately processed in parallel) will produce better results in data-intensive analytic tasks where parallelism may be exploited. Oracle does have this feature known by name as the functions *rqTableApply()*, *rqGroupApply()* and *rqRowApply()* as introduced in sub-section 2.2.1 under section 2.2 of chapter 2. Implementation of similar features by other DBMSs for efficient retrieval and transfer of data to analytic functions/routines will boost performance.

6.5 Implications of the findings to the research questions

Having examined and discussed the experimental results, recommending coupling databases and advanced analytical tools cannot be overemphasised. Such an arrangement would help to achieve better analytical performance, data security, system reliability and enhanced system management in general. But an important point of note which the empirical findings revealed is that retrieval and passing of data to analytical routines is very crucial and must be given proper attention if considerable performance gains are to be achieved.

The study also established a relation between the architectural arrangement of database systems integrated with advanced analytical tools (R), and the various analytic performance gains achievable. Oracle integration with R serves as a good example. It revealed that the architectural arrangement that produces the best performance results is a design that has native database equivalent of analytic functions executed within database. This arrangement only calls external full-blown advanced analytic engine into action when no native function match of the analytic function or command is found during execution. This implies that the more native function equivalent or match of analytic functions there are in-database, the greater the performance improvements or gains.

Returning to our research questions, we provide the following answers as implied from the study:

(i) *What is the current level of development (completeness) of integration of R with DMBS?*
From our review of advanced analytical tools (R) and the various DBMS we note that the

level of development (completeness) of integration of R with DMBS is growing. Most capabilities of stand-alone R (base R) are obtainable from the DBMSs which make provision for integrating and executing R. There is however room for improvement and this is evidenced in the in-database analytics strategies being pursued by the various well-known DBMSs vendors such as Teradata, SAP, EMC (Greenplum), Oracle and IBM by opening up ways of database systems integration with advanced analytical tools such as R and SPSS.

(ii) *How is the performance of coupling databases with advanced analytical tool (R) compared to stand-alone analytical tool (R)?* Coupling databases with advanced analytical tools produces better performance compared to stand-alone R provided the data being analyzed is efficiently retrieved for analytic functions/routines as pointed out by our findings. Additionally, efficiently passing data is equally important as seen in the case of Oracle's *rqTableEval* which passes entire table as input at one go, improving performance. Absence of these will rather lead to a worsen performance as seen in the case of PostgreSQL. However, for less data-intensive analytic tasks, the performance of stand-alone analytical tools remain equally competitive.

(iii) *How is the scalability of coupling databases with advanced analytical tool (R) compared to stand-alone analytical tool (R)?* The scalability of coupled database and advanced analytical tool (R) is a better compared to stand-alone analytical tool (R) [*refer to scalability tests result table and charts in Appendix B.7 on page 120*]. It is important to mention that for very large datasets, stand-alone R simply does not work and that is where the scalability advantage of the coupled system (R+DBMS) becomes hugely manifested. This is because DBMSs by design are scalable with respect to changing conditions such as growth in data volumes, increased workloads and users, and evolving business requirements. For coupled systems that are handled within this same scalable infrastructure, the analytics capacity likewise scale. There is also additional benefit of reduced data transfer and centralized deployment.

(iv) *What are the inherent implications of architectures of R integration that impact performance?* The innate parallelism nature of databases inure to the benefit of running R analytics in database resulting in higher performance. Likewise, the scalability of database systems facilitates and enhances support for growth in data volumes, user numbers and workloads when integrated with R for analytics. Thus, analytics implemented with databases coupled with R are highly scalable. However, DBMS architectures extended with R analytic functionalities, in the form of user-defined functions, are limited by sanctioned extensions of the vendor. For universal, robust analytics with database systems, a full framework

capable of executing advanced or statistical third-party code is required. R integration architecture with an in-database analytic engine produces higher performance. In case of light-weight (not having all functionalities) in-database analytic engines, there is a need to call external full-fledged analytic engine into action when the computations cannot be ran in database. So, database resident data would be transferred to R engine located outside the database and this causes reduction in the margin of performance gains attainable. However, when the architecture allows for activation of multiple instances of the external analytic engine, an equally good performance results are obtainable. To recap, the location of the analytic engine with respect to the database, the existence or development of native SQL equivalent of analytic functions, the extent of exploitation of database parallelism and the scalability of databases all in one way or the other impact performance of databases coupled with advanced analytical tools.

(v) *Are there any lessons to be learnt on the way forward?* The key lesson to be learnt which came about as a result of unexpected snag performance from PostgreSQL DBMS is that the technique employed for retrieving and passing data to analytic functions makes huge impact on the analytic performance regardless of how fast the substantive analytic task is performed. Therefore, caution must be exercised, taking into consideration this factor when evaluating potential solutions for adoption. More so, the concept of coupling databases and advanced analytical tools applies best to analytics which involves considerable amount of data. This is evidenced in the results of our first two benchmark categories (MCs and MFs) which involve significant datasets and produces higher performance results for databases coupled with R. Besides, we think that coupling of databases and advanced analytical tool will very well apply to real-time and recurrent analytics as well as analytics which is cross-functional. This is because of its (coupled systems) preferred centralized deployment, and the elimination of the time lag or delay associated with preparing and moving data to stand-alone analytical or statistical tools.

Chapter 7

CONCLUSIONS AND FUTURE STUDIES

In this chapter, the conclusions of our study are presented and suggestions for future studies are advanced.

7.1 Conclusions

Even though our work focused on relational DBMS and R, it is obvious that our findings are to a large extent potentially applicable to other types of DBMS and/or advanced analytical tools or statistical packages. From the outcome of the study the following conclusions have been reached:

(i) We cannot overemphasis the several benefits that if one requires analysis of data, it must be done from within the database. Also, there is more to data (e.g. linear algebra concepts required for advanced data analytics) than just relational model. Embedding statistics/advanced analytics into relational databases can be likened to a two-edged sword in that if done well the results are tremendously good, else very bad.

(ii) Statistical algorithms are inclined to linear algebra, rather than relational algebra which forms the foundation for SQL and RDBMS. It is therefore important not to throw caution to the wind when coupling relational databases and advanced analytical tools in order to realize synergy of the two. Also, a point of note is that research in the field of statistics continues to find novel algorithms and implementations while research into relational database

is almost grinding to a halt. Therefore, merging the two worlds can be very beneficial especially to the relational database world.

(iii) Looking into the future, in-database analytics is going to be a critical requirement because analytics will be integrated with many applications to deliver insight in one way or the other. In other words, the need for separate analytics application will be eliminated or substituted for applications with embedded advanced analytics functionalities. For example, there would be advanced analytical/statistical algorithms offered as a service where complex optimized and up-to-date statistical routines will be provided and consumed on-demand basis.

(iv) Of course, one challenge will be tackling the integration with applications written in numerous different programming languages. Service Oriented Architecture/SOBI will become even more useful in this situation. Also, APIs will provide answers to the difficulties of integration by facilitating provision of analytics for consumption. This will basically entail provision of a programmable interface that defines analytical function/procedure independent of any programming language. Our implementation of R integration with SQL Server database by exploiting SQL Server CLR feature to deploy an assembly with embedded R routines in-database is one such approach. Such integrations using APIs should not negatively impact performance because the APIs will just provide the analytic algorithm that will be sent to the database (function shipping) for the analytic processing. Thus, a case of sending algorithm to data (function-shipping) instead of data to algorithm (data-shipping) and so we expect the impact on performance to be positive.

(v) The challenge of having to implement from scratch each new advanced/statistical technique in the DBMSs, which inevitably leads to a sophisticated and protracted development process is eliminated by coupling databases with advanced analytical tools (R). This brings about several benefits particularly performance upgrades. It is in line with this that there is intense pursuit among DBMSs vendors to offer even more sophisticated analytics inside database.

(vi) The manner in which data is effectively and efficiently pulled or pushed to analytic functions or routines is very critical and hugely impact overall in-database analytics performance. A case of note is PostgreSQL where there seems not to be a very efficient way of doing it, leading to huge cost and hence poor in-database analytics performance in contradiction to expectations.

(vii) Finally, in recommending coupling databases and advanced analytical tools such as R, we note that the architecture of choice must help to facilitate efficient retrieval and passing of data from the database objects (tables, procedures, etc) to the analytic functions, lessen or eliminate data movement, reduce run-time overheads, maintain data security, and also reduce development overheads for introducing new and maintaining existing advanced analytics code in the DBMSs.

7.2 Future Studies

This study has brought up some matters that are worth further investigation. Notable among the subject matters resulting from this study which require further examination in order to advance the understanding of the performance of coupling advanced analytical tools (R) with databases include the following topics;

(i) Benchmarking of in-memory databases such SAP HANA, column-oriented databases like Vertica or databases systems which require substantial main memory (RAM) such as Cloudera Impala. Also, benchmarking performances of document-oriented storage database systems such as MongoDB and Cassandra is important.

(ii) Comparison of the order of change in performance (improvement or deterioration) by integrating R with relational databases and NoSQL databases.

(iii) The maximum extent of improvement that parallelism and scalability can bring about in the coupled systems (R+DBMSs).

(iv) Running the same benchmarks on different operating systems, and also with varying amount or size of datasets to measure performance differences brought about by these factors.

(v) Effective and efficient retrieval and passing of database resident data to analytic/statistical functions for optimum overall analytics performance. A case of PostgreSQL's abysmal performance could be taken on as a starting point.

(vi) Benchmarking on datasets with varied attribute properties and comparing the performance differences or similarities with respect to the different attributes' characteristics of the datasets.

Appendix A

Benchmark Codes

A.1 Codes for benchmark on stand-alone R

Listing A.1: Codes for benchmark on stand-alone R (*Adapted from [24][2]*)

```
#==Adapted from R Benchmark 2.5 and RevoR Enterprise Benchmark=====
runs <- 7;         # Number of times the tests are ran
times <- rep(0.00, 7);  # Initialise result vector for the runs

#Setup connecton to data source============================
channel <- "E:/stockDnldDir/stockHist.csv";
#===========================================================


#MC. Matrix Calculation====================================
#===========================================================
remove("a", "b")
#
#MC01. Transposing*****************************************
a <- 0; b <- 0;
for (i in 1:runs) {
  invisible(gc())
  times[i] <- system.time({
  a <- as.matrix(read.csv(channel, header = TRUE, sep = ","));
  b <- t(a);
  dim(b) <- c(2000, 8000);
  a <- t(b);
  })[3]
}
cat(c("\n\n\n","MC01. Transposing- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
```

```
#**********************************************************

remove("a", "b")
#
#MC02. Exponentiation***************************************
a <- 0; b <- 0;
for (i in 1:runs) {
  invisible(gc())
  times[i] <- system.time({
  a <- (abs(as.matrix(read.csv(channel, header = TRUE, sep = ","))))^1000;
  })[3]
}
cat(c("\n\n\n","MC02. Exponentiation- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#**********************************************************

remove("a", "b")
#
#MC03. Sorting**********************************************
a <- 0; b <- 0;
for (i in 1:runs) {
  invisible(gc())
  times[i] <- system.time({
  a <- sort(as.matrix(read.csv(channel, header = TRUE, sep = ",")), method="quick")
  })[3]
}
cat(c("\n\n\n","MC03. Sorting- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#**********************************************************

remove("a", "b")
#
#MC04. Cross-Product****************************************
a <- 0; b <- 0;
for (i in 1:runs) {
  invisible(gc())
  times[i] <- system.time({
  a <- crossprod(as.matrix(read.csv(channel, header = TRUE, sep = ",")))
  })[3]
}
cat(c("\n\n\n","MC04. Cross-Product- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#**********************************************************

remove("a", "b")
#
#MC05. Solving Linear Regression****************************
a <- 0; b <- 0;
for (i in 1:runs) {
  b <- as.double(1:4000)
  invisible(gc())
  times[i] <- system.time({
```

```
  a <- solve(crossprod(array(as.matrix(read.csv(channel, header = TRUE, sep = ",")), dim =
    c(4000, 4000))),
    crossprod(array(as.matrix(read.csv(channel, header = TRUE, sep = ",")), dim = c(4000,
   4000)),b))
  })[3]
}
cat(c("\n\n\n","MC05. Solving Linear Regression- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#==========================================================


#MF. Matrix Functions=====================================
#==========================================================
remove("a", "b")
#
# MF01. Fast Fourier Transform*****************************
a <- 0;
for (i in 1:runs) {
  invisible(gc())
  times[i] <- system.time({
  a <- fft(as.matrix(read.csv(channel, header = TRUE, sep = ",")))
  })[3]
}
cat(c("\n\n\n","MF01. Fast Fourier Transform- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#==========================================================

remove("a")
#
#MF02. EigenValues*****************************************
a <- 0; b <- 0;
for (i in 1:runs) {
  invisible(gc())
  times[i] <- system.time({
  a <- eigen(array(as.matrix(read.csv(channel, header = TRUE, sep = ",")), dim = c(4000, 4
   000)), symmetric=FALSE, only.values=TRUE)$Value
  })[3]
}
cat(c("\n\n\n","MF02. EigenValues- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#==========================================================

remove("a", "b")
#
#MF03. Determinant*****************************************
a <- 0; b <- 0;
for (i in 1:runs) {
  invisible(gc())
  times[i] <- system.time({
  a <- det(array(as.matrix(read.csv(channel, header = TRUE, sep = ",")), dim = c(4000, 400
   0)))
  })[3]
```

```
}
cat(c("\n\n\n","MF03. Determinant- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#**********************************************************

remove("a", "b")
#
#MF04. Cholesky Decomposition*****************************
a <- 0; b <- 0;
for (i in 1:runs) {
  invisible(gc())
  times[i] <- system.time({
  a <- chol(crossprod(array(as.matrix(read.csv(channel, header = TRUE, sep = ",")), dim =
   c(4000, 4000))))
  })[3]
}
cat(c("\n\n\n","MF04. Cholesky Decomposition- Results(sec) of 7 Runs:", "\n"));
print(round(times,2));  #print(times) OR paste(times);
#**********************************************************

remove("a", "b")
#
#MF05. Inverse*****************************************
a <- 0; b <- 0;
for (i in 1:runs) {
  invisible(gc())
  times[i] <- system.time({
  a <- solve(array(as.matrix(read.csv(channel, header = TRUE, sep = ",")), dim = c(4000, 4
   000)))
  })[3]
}
cat(c("\n\n\n","MF05. Inverse- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#**********************************************************

remove("a", "b")
#
#MF06. Single Value Decomposition*************************
a <- 0; b <- 0;
for (i in 1:runs) {
  invisible(gc())
  times[i] <- system.time({
  a <- svd(array(as.matrix(read.csv(channel, header = TRUE, sep = ",")), dim = c(4000, 400
   0)), nu=0, nv=0)
  })[3]
}
cat(c("\n\n\n","MF06. Single Value Decomposition- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#**********************************************************
remove("a", "b")
#
#MF07. Principal Component Analysis***********************
```

```
a <- 0; b <- 0;
for (i in 1:runs) {
  a <- rnorm(2500*2500); dim(a) <- c(2500, 2500)
  invisible(gc())
  times[i] <- system.time({
  a <- prcomp(array(as.matrix(read.csv(channel, header = TRUE, sep = ",")), dim = c(4000,
   4000)))
  })[3]
}
cat(c("\n\n\n","MF07. Principal Component Analysis- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#*********************************************************

remove("a", "b")
#
#MF08. Linear Discriminant Analysis**********************
a <- 0; b <- 0;
for (i in 1:runs) {
  m <- 16000
  n <- 1000
  g <- 5
  k <- round (m/2)

  invisible(gc())
  times[i] <- system.time({
  require ('MASS')
  a <- lda(fac ~.,
    data=data.frame(read.csv(channel, header = TRUE, sep = ","), fac=sample (LETTERS[1:g],
   m,replace=TRUE)),
    prior=rep(1,g)/g,
    subset=sample(1:m, k))
  })[3]
}
cat(c("\n\n\n","MF08. Linear Discriminant Analysis- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#*********************************************************


#PC. Program Control=====================================
#=========================================================
remove("a", "b")
  #
#PC01. Fibonacci Numbers Calculation (Vector Calculation)**
a <- 0; b <- 0; phi <- 1.6180339887498949
for (i in 1:runs) {
  a <- floor(runif(3500000)*1000)
  invisible(gc())
  times[i] <- system.time({
  b <- (phi^a - (-phi)^(-a))/sqrt(5)
  })[3]
}
```

```
cat(c("\n\n\n","Fibonacci Numbers Calculation (Vector Calculation)- Results(sec) of 7 Runs
   :", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#**********************************************************

remove("a", "b", "phi")
  #
#PC02. Hilbert Matrix (Matrix Calculation)*****************
a <- 3000; b <- 0
for (i in 1:runs) {
  invisible(gc())
  times[i] <- system.time({
  b <- rep(1:a, a); dim(b) <- c(a, a);
  b <- 1 / (t(b) + 0:(a-1))
  })[3]
}
cat(c("\n\n\n","PC02. Hilbert Matrix (Matrix Calculation)- Results(sec) of 7 Runs:", "\n")
   );
print(round(times, 2)); #print(times) OR paste(times);
#**********************************************************

remove("a", "b")
#
#PC03. Grand Common Divisors of Pairs**********************
c <- 0
gcd2 <- function(x, y) {if (sum(y > 1.0E-4) == 0) x else {y[y == 0] <- x[y == 0]; Recall(y
   , x %% y)}}
for (i in 1:runs) {
  a <- ceiling(runif(400000)*1000)
  b <- ceiling(runif(400000)*1000)
  invisible(gc())
  times[i] <- system.time({
  c <- gcd2(a, b)
  })[3]
}
cat(c("\n\n\n","PC03. Grand Common Divisors of Pairs- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#**********************************************************

remove("a", "b", "c", "gcd2")
#
#PC04. Toeplitz Matrix (Loops)*****************************
b <- 0
for (i in 1:runs) {
  b <- rep(0, 500*500); dim(b) <- c(500, 500)
  invisible(gc())
  times[i] <- system.time({
  for (j in 1:500) {
    for (k in 1:500) {
    b[k,j] <- abs(j - k) + 1
    }
  }
```

```
  })[3]
}
cat(c("\n\n\n","PC04. Toeplitz Matrix (Loops)- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);
#***********************************************************

remove("b", "j", "k")
#
#PC05. Escoufier's Method*********************************
p <- 0; vt <- 0; vr <- 0; vrt <- 0; rvt <- 0; RV <- 0; j <- 0; k <- 0;
x2 <- 0; R <- 0; Rxx <- 0; Ryy <- 0; Rxy <- 0; Ryx <- 0; Rvmax <- 0
# Calculate the trace of a matrix (sum of its diagonal elements)
Trace <- function(y) {sum(c(y)[1 + 0:(min(dim(y)) - 1) * (dim(y)[1] + 1)], na.rm=FALSE)}
for (i in 1:runs) {
  x <- abs(rnorm(45*45)); dim(x) <- c(45, 45)
  invisible(gc())
  times[i] <- system.time({
  # Calculation of Escoufier's equivalent vectors
  p <- ncol(x)
  vt <- 1:p
  vr <- NULL
  RV <- 1:p
  vrt <- NULL
  for (j in 1:p) {
    Rvmax <- 0
    for (k in 1:(p-j+1)) {
    x2 <- cbind(x, x[,vr], x[,vt[k]])
    R <- cor(x2)
    Ryy <- R[1:p, 1:p]
    Rxx <- R[(p+1):(p+j), (p+1):(p+j)]
    Rxy <- R[(p+1):(p+j), 1:p]
    Ryx <- t(Rxy)
    rvt <- Trace(Ryx %*% Rxy) /
    sqrt(Trace(Ryy %*% Ryy) *
    Trace(Rxx %*% Rxx))
    if (rvt > Rvmax) {
      Rvmax <- rvt
      vrt <- vt[k]
    }
    }
    vr[j] <- vrt
    RV[j] <- Rvmax
    vt <- vt[vt!=vr[j]]
  }
  })[3]

}
cat(c("\n\n\n","PC05. Escoufier's Method- Results(sec) of 7 Runs:", "\n"));
print(round(times, 2)); #print(times) OR paste(times);

remove(list=ls())
cat("\n\n\n\n\n#*****************--- End of Test ---\n\n")**************
```

A.2 Codes for benchmark on Oracle DBMS

Listing A.2: Codes for benchmark on Oracle DBMS (*Adapted from [24][2]*)

```
--==Adapted from R Benchmark 2.5 and RevoR Enterprise Benchmark=====
select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mc01_transposeF'));

select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mc02_exponentiateF'));

select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mc03_sortF'));

select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mc04_crossproductF'));

select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mc05_solveF'));

--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
--=================================================================================

select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mf01_fastfouriertransformF'));

select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mf02_eigenvalueF'));
```

```
select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mf03_determinantF'));

select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mf04_choleskydecompositionF'));

select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mf05_inverseF'));

select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mf06_singlevaluedecompositionF'));

select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mf07_principalcomponentanalysisF'));

select * from table(rqTableEval(
  cursor(select * from STOCKHIST),
  NULL,
  'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
  'mf08_lineardiscriminantanalysisF'));

--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
--==============================================================================

select *
  from table(rqEval(NULL,
    'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
    'pc01_fibonaccinumbersF'));

select *
  from table(rqEval(NULL,
    'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
    'pc02_hilbertmatrixF'));

select *
  from table(rqEval(NULL,
    'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
```

```
    'pc03_grandcommondivisorsF'));

select *
  from table(rqEval(NULL,
    'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
    'pc04_toeplitzmatrixF'));

select *
  from table(rqEval(NULL,
    'select 1 Run1, 2 Run2, 3 Run3, 4 Run4, 5 Run5, 6 Run6, 7 Run7 from dual',
    'pc05_escoufiersmethodF'));
--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
--===============================================================================

begin
sys.rqScriptCreate('mc01_transposeF',
 'function(dat){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MC01. Transposing***********************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- as.matrix(dat);
    b <- t(a);
    dim(b) <- c(2000, 8000);
    a <- t(b);
    })[3]
  }
  #cat(c("\n\n\n","MC01. Transposing- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  #****************************************************************************
  }');
end;

begin
sys.rqScriptCreate('mc02_exponentiateF',
 'function(dat){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MC02. Exponentiation***********************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
```

```
    invisible(gc())
    times[i] <- system.time({
    a <- (abs(as.matrix(dat)))^1000;
    })[3]
  }
  #cat(c("\n\n\n","MC02. Exponentiation- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  #***************************************************************************
  }');
end;

begin
sys.rqScriptCreate('mc03_sortF',
 'function(dat){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MC03. Sorting*****************************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- sort(as.matrix(dat), method="quick")
    })[3]
  }
  #cat(c("\n\n\n","MC03. Sorting- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  #***************************************************************************

  }');
end;

begin
sys.rqScriptCreate('mc04_crossproductF',
 'function(dat){

  runs <- 7;    # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b");
  #
  #MC04. Cross-Product***********************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc());
    times[i] <- system.time({
    a <- crossprod(as.matrix(dat));
  #a <- crossprod(array(as.matrix(dat), dim = c(4000, 4000)),b);
    })[3]
```

```
  }
  #cat(c("\n\n\n","MC04. Cross-Product- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  #********************************************************************************
  }');
end;

begin
sys.rqScriptCreate('mc05_solveF',
 'function(dat){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MC05. Solving Linear Regression************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    b <- as.double(1:4000)
    invisible(gc())
    times[i] <- system.time({
  a <- solve(abs(crossprod(array(as.matrix(dat), dim = c(4000, 4000)))),
    abs(crossprod(array(as.matrix(dat), dim = c(4000, 4000)),b)));
  })[3]
  }
  #cat(c("\n\n\n","MC05. Solving Linear Regression- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  #********************************************************************************
  }');
end;

begin
sys.rqScriptCreate('mf01_fastfouriertransformF',
 'function(dat){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b", "c", "qra")
  #
  # MF01. Fast Fourier Transform***************************************************
  a <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- fft(as.matrix(dat))
    })[3]
  }
  #cat(c("\n\n\n","MF01. Fast Fourier Transform- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  #********************************************************************************
```

```
  }');
end;

begin
sys.rqScriptCreate('mf02_eigenvalueF',
 'function(dat){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MF02. EigenValues**************************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- eigen(array(as.matrix(dat), dim = c(4000, 4000)), symmetric=FALSE, only.values=
   TRUE)$Value
    })[3]
  }
  #cat(c("\n\n\n","MF02. EigenValues- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  #*********************************************************************************

  }');
end;

begin
sys.rqScriptCreate('mf03_determinantF',
 'function(dat){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MF03. Determinant**************************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- det(array(as.matrix(dat), dim = c(4000, 4000)))
    })[3]
  }
  #cat(c("\n\n\n","MF03. Determinant- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  }');
end;

begin
sys.rqScriptCreate('mf04_choleskydecompositionF',
```

```
 'function(dat){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MF04. Cholesky Decomposition*****************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- chol(crossprod(array(as.matrix(dat), dim = c(4000, 4000))))
    })[3]
  }
  #cat(c("\n\n\n","MF04. Cholesky Decomposition- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  }');
end;

begin
sys.rqScriptCreate('mf05_inverseF',
 'function(dat){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MF05. Inverse********************************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- solve(array(as.matrix(dat), dim = c(4000, 4000)))
    })[3]
  }
  #cat(c("\n\n\n","MF05. Inverse- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  #*************************************************************************************
  }');
end;

begin
sys.rqScriptCreate('mf06_singlevaluedecompositionF',
 'function(dat){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b")
  #
```

```
  #MF06. Single Value Decomposition*********************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- svd(array(as.matrix(dat), dim = c(4000, 4000)), nu=0, nv=0)
    })[3]
  }
  #cat(c("\n\n\n","MF06. Single Value Decomposition- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  }');
end;

begin
sys.rqScriptCreate('mf07_principalcomponentanalysisF',
 'function(dat){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MF07. Principal Component Analysis*******************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    a <- rnorm(2500*2500); dim(a) <- c(2500, 2500)
    invisible(gc())
    times[i] <- system.time({
    a <- prcomp(array(as.matrix(dat), dim = c(4000, 4000)))
    })[3]
  }
  #cat(c("\n\n\n","MF07. Principal Component Analysis- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  #*****************************************************************************
  }');
end;

begin
sys.rqScriptCreate('mf08_lineardiscriminantanalysisF',
 'function(dat){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MF08. Linear Discriminant Analysis*******************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    m <- 16000
    n <- 1000
    g <- 5
```

```
    k <- round (m/2)

    invisible(gc())
    times[i] <- system.time({
    require ("MASS")
    a <- lda(fac ~.,
      data=data.frame(dat, fac=sample (LETTERS[1:g],m,replace=TRUE)),
      prior=rep(1,g)/g,
      subset=sample(1:m, k))
    })[3]
  }
  #cat(c("\n\n\n","MF08. Linear Discriminant Analysis- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  }');
end;

begin
sys.rqScriptCreate('pc01_fibonaccinumbersF',
 'function(){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  #remove(list=ls())
  remove("a", "b")
  #
  #PC01. Fibonacci Numbers Calculation (Vector Calculation)************************
  a <- 0; b <- 0; phi <- 1.6180339887498949
  for (i in 1:runs) {
    a <- floor(runif(3500000)*1000)
    invisible(gc())
    times[i] <- system.time({
    b <- (phi^a - (-phi)^(-a))/sqrt(5)
    })[3]
  }
  cat(c("\n\n\n","Fibonacci Numbers Calculation (Vector Calculation)- Results(sec) of 7
   Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  }');
end;

begin
sys.rqScriptCreate('pc02_hilbertmatrixF',
 'function(){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  #remove(list=ls())
  remove("a", "b", "phi")
  #
  #PC02. Hilbert Matrix (Matrix Calculation)************************************
```

```
  a <- 3000; b <- 0
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    b <- rep(1:a, a); dim(b) <- c(a, a);
    b <- 1 / (t(b) + 0:(a-1))
    })[3]
  }
  cat(c("\n\n\n","PC02. Hilbert Matrix (Matrix Calculation)- Results(sec) of 7 Runs:", "\n
   "));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));**********
   ****************************************************
  }');
end;

begin
sys.rqScriptCreate('pc03_grandcommondivisorsF',
 'function(){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  #remove(list=ls())
  remove("a", "b")
  #
  #PC03. Grand Common Divisors of Pairs*****************************************
  c <- 0
  gcd2 <- function(x, y) {if (sum(y > 1.0E-4) == 0) x else {y[y == 0] <- x[y == 0]; Recall
   (y, x %% y)}}
  for (i in 1:runs) {
    a <- ceiling(runif(400000)*1000)
    b <- ceiling(runif(400000)*1000)
    invisible(gc())
    times[i] <- system.time({
    c <- gcd2(a, b)                          # gcd2 is a recursive function
    })[3]
  }
  cat(c("\n\n\n","PC03. Grand Common Divisors of Pairs- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  }');
end;

begin
sys.rqScriptCreate('pc04_toeplitzmatrixF',
 'function(){

  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  #remove(list=ls())
  remove("a", "b", "c", "gcd2")
  #
```

```
  #PC04. Toeplitz Matrix (Loops)*****************************************
  b <- 0
  for (i in 1:runs) {
    b <- rep(0, 500*500); dim(b) <- c(500, 500)
    invisible(gc())
    times[i] <- system.time({
    for (j in 1:500) {
      for (k in 1:500) {
      b[k,j] <- abs(j - k) + 1
      }
    }
    })[3]
  }
  #cat(c("\n\n\n","PC04. Toeplitz Matrix (Loops)- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  }');
end;

begin
sys.rqScriptCreate('pc05_escoufiersmethodF',
 'function(){
  runs <- 7   # Number of times the tests are ran
  times <- rep(0.00, 7);  # Initialize result vector for the runs

  #remove(list=ls())
  remove("b", "j", "k")
  #
  #PC05. Escoufiers Method*****************************************
  p <- 0; vt <- 0; vr <- 0; vrt <- 0; rvt <- 0; RV <- 0; j <- 0; k <- 0;
  x2 <- 0; R <- 0; Rxx <- 0; Ryy <- 0; Rxy <- 0; Ryx <- 0; Rvmax <- 0
  # Calculate the trace of a matrix (sum of its diagonal elements)
  Trace <- function(y) {sum(c(y)[1 + 0:(min(dim(y)) - 1) * (dim(y)[1] + 1)], na.rm=FALSE)}
  for (i in 1:runs) {
    x <- abs(rnorm(45*45)); dim(x) <- c(45, 45)
    invisible(gc())
    times[i] <- system.time({
    # Calculation of Escoufiers equivalent vectors
    p <- ncol(x)
    vt <- 1:p
    vr <- NULL
    RV <- 1:p
    vrt <- NULL
    for (j in 1:p) {
      Rvmax <- 0
      for (k in 1:(p-j+1)) {
      x2 <- cbind(x, x[,vr], x[,vt[k]])
      R <- cor(x2)
      Ryy <- R[1:p, 1:p]
      Rxx <- R[(p+1):(p+j), (p+1):(p+j)]
      Rxy <- R[(p+1):(p+j), 1:p]
      Ryx <- t(Rxy)
      rvt <- Trace(Ryx %*% Rxy) /
```

```
      sqrt(Trace(Ryy %*% Ryy) *
      Trace(Rxx %*% Rxx))
      if (rvt > Rvmax) {
        Rvmax <- rvt
        vrt <- vt[k]
      }
      }
      vr[j] <- vrt
      RV[j] <- Rvmax
      vt <- vt[vt!=vr[j]]
    }
    })[3]

  }
  #cat(c("\n\n\n","PC05. Escoufiers Method- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  }');
end;
```

A.3 Codes for benchmark on Postgres DBMS

Listing A.3: Codes for benchmark on Postgres DBMS (*Adapted from [24][2]*)

```
----Adapted from R Benchmark 2.5 and RevoR Enterprise Benchmark=====
SELECT  mc01_transposeF5('SELECT * FROM stockHist');
SELECT  mc02_exponentiateF5('SELECT * FROM stockHist');
SELECT  mc03_sortF5('SELECT * FROM stockHist');
SELECT  mc04_crossproductF5('SELECT * FROM stockHist');
SELECT  mc05_solveF5('SELECT * FROM stockHist');

SELECT  mf01_fastfouriertransformF5('SELECT * FROM stockHist');
SELECT  mf02_eigenvalueF5('SELECT * FROM stockHist');
SELECT  mf03_determinantF5('SELECT * FROM stockHist');
SELECT  mf04_choleskydecompositionF5('SELECT * FROM stockHist');
SELECT  mf05_inverseF5('SELECT * FROM stockHist');
SELECT  mf06_singlevaluedecompositionF5('SELECT * FROM stockHist');
SELECT  mf07_principalcomponentanalysisF5('SELECT * FROM stockHist');
SELECT  mf08_lineardiscriminantanalysisF5('SELECT * FROM stockHist');

SELECT  pc01_fibonaccinumbersF5();
SELECT  pc02_hilbertmatrixF5();
SELECT  pc03_grandcommondivisorsF5();
SELECT  pc04_toeplitzmatrixF5();
SELECT  pc05_escoufiersmethodF5();

CREATE OR REPLACE FUNCTION mc01_transposeF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MC01. Transposing************************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- as.matrix(pg.spi.exec(arg1));
    b <- t(a);
    dim(b) <- c(2000, 8000);
    a <- t(b);
    })[3]
  }
  cat(c("\n\n\n","MC01. Transposing- Results(sec) of 7 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
  #*******************************************************************************
'
LANGUAGE plr;
```

```
CREATE OR REPLACE FUNCTION mc02_exponentiateF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MC02. Exponentiation***************************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- (abs(as.matrix(pg.spi.exec(arg1))))^1000;
    })[3]
  }
  cat(c("\n\n\n","MC02. Exponentiation- Results(sec) of 7 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
  #************************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION mc03_sortF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MC03. Sorting**********************************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- sort(as.matrix(pg.spi.exec(arg1)), method="quick")
    })[3]
  }
  cat(c("\n\n\n","MC03. Sorting- Results(sec) of 7 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
  #************************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION mc04_crossproductF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
```

```
  #
  #MC04. Cross-Product*************************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- crossprod(as.matrix(pg.spi.exec(arg1)))
    })[3]
  }
  cat(c("\n\n\n","MC04. Cross-Product- Results(sec) of 7 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
  #*********************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION mc05_solveF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MC05. Solving Linear Regression************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    b <- as.double(1:4000)
    invisible(gc())
    times[i] <- system.time({
    a <- solve(crossprod(array(as.matrix(pg.spi.exec(arg1)), dim = c(4000, 4000))),
      crossprod(array(as.matrix(pg.spi.exec(arg1)), dim = c(4000, 4000)),b))
    })[3]
  }
  cat(c("\n\n\n","MC05. Solving Linear Regression- Results(sec) of 7 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
  #*********************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION mf01_fastfouriertransformF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
  #
  # MF01. Fast Fourier Transform****************************************************
  a <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
```

```
    a <- fft(as.matrix(pg.spi.exec(arg1)))
    })[3]
  }
  cat(c("\n\n\n","MF01. Fast Fourier Transform- Results(sec) of 7 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
  #********************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION mf02_eigenvalueF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MF02. EigenValues*************************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- eigen(array(as.matrix(pg.spi.exec(arg1)), dim = c(4000, 4000)), symmetric=FALSE,
   only.values=TRUE)$Value
    })[3]
  }
  cat(c("\n\n\n","MF02. EigenValues- Results(sec) of 7 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
  #********************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION mf03_determinantF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MF03. Determinant*************************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- det(array(as.matrix(pg.spi.exec(arg1)), dim = c(4000, 4000)))
    })[3]
  }
  cat(c("\n\n\n","MF03. Determinant- Results(sec) of 7 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
  #********************************************************************************
'
```

```
LANGUAGE plr;

CREATE OR REPLACE FUNCTION mf04_choleskydecompositionF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MF04. Cholesky Decomposition**************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- chol(crossprod(array(as.matrix(pg.spi.exec(arg1)), dim = c(4000, 4000))))
    })[3]
  }
  cat(c("\n\n\n","MF04. Cholesky Decomposition- Results(sec) of 7 Runs:", "\n"));
  print(round(times,2));  #print(round(times,2)) OR paste(times);
  #*******************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION mf05_inverseF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MF05. Inverse*****************************************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- solve(array(as.matrix(pg.spi.exec(arg1)), dim = c(4000, 4000)))
    })[3]
  }
  cat(c("\n\n\n","MF05. Inverse- Results(sec) of 7 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
  #*******************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION mf06_singlevaluedecompositionF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs
```

```
  remove("a", "b")
  #
  #MF06. Single Value Decomposition*********************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- svd(array(as.matrix(pg.spi.exec(arg1)), dim = c(4000, 4000)), nu=0, nv=0)
    })[3]
  }
  cat(c("\n\n\n","MF06. Single Value Decomposition- Results(sec) of 7 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
  #********************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION mf07_principalcomponentanalysisF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
      #
  #MF07. Principal Component Analysis*******************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    a <- rnorm(2500*2500); dim(a) <- c(2500, 2500)
    invisible(gc())
    times[i] <- system.time({
    a <- prcomp(array(as.matrix(pg.spi.exec(arg1)), dim = c(4000, 4000)))
    })[3]
  }
  cat(c("\n\n\n","MF07. Principal Component Analysis- Results(sec) of 7 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
  #********************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION mf08_lineardiscriminantanalysisF5(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #MF08. Linear Discriminant Analysis*******************************************
  a <- 0; b <- 0;
  for (i in 1:runs) {
    m <- 16000; n <- 1000; g <- 5; k <- round (m/2);
    invisible(gc())
```

```
    times[i] <- system.time({
    require ("MASS");
    a <- lda(fac ~.,
      data=data.frame(pg.spi.exec("--SELECT * FROM stockHist"),
      fac=sample (LETTERS[1:g],m,replace=TRUE)),
      prior=rep(1,g)/g, subset=sample(1:m, k))
    })[3]
  }
  cat(c("\n\n\n","MF08. Linear Discriminant Analysis- Results(sec) of 7 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
  #********************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION pc01_fibonaccinumbersF5()
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #PC01. Fibonacci Numbers Calculation (Vector Calculation)*************************
  a <- 0; b <- 0; phi <- 1.6180339887498949
  for (i in 1:runs) {
    a <- floor(runif(3500000)*1000)
    invisible(gc())
    times[i] <- system.time({
    b <- (phi^a - (-phi)^(-a))/sqrt(5)
    })[3]
  }
  cat(c("\n\n\n","Fibonacci Numbers Calculation (Vector Calculation)- Results(sec) of 7
   Runs:", "\n"));
  print(round(times,2));  #paste(times);
  #********************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION pc02_hilbertmatrixF5()
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b", "phi")
  #
  #PC02. Hilbert Matrix (Matrix Calculation)****************************************
  a <- 3000; b <- 0
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    b <- rep(1:a, a); dim(b) <- c(a, a);
```

```
    b <- 1 / (t(b) + 0:(a-1))
    })[3]
  }
  cat(c("\n\n\n","PC02. Hilbert Matrix (Matrix Calculation)- Results(sec) of 7 Runs:", "\n
  "));
  print(round(times,2));  #paste(times);
  #**************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION pc03_grandcommondivisorsF5()
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b")
  #
  #PC03. Grand Common Divisors of Pairs*****************************************
  c <- 0
  gcd2 <- function(x, y) {if (sum(y > 1.0E-4) == 0) x else {y[y == 0] <- x[y == 0]; Recall
   (y, x %% y)}}
  for (i in 1:runs) {
    a <- ceiling(runif(400000)*1000)
    b <- ceiling(runif(400000)*1000)
    invisible(gc())
    times[i] <- system.time({
    c <- gcd2(a, b)             # gcd2 is a recursive function
    })[3]
  }
  cat(c("\n\n\n","PC03. Grand Common Divisors of Pairs- Results(sec) of 7 Runs:", "\n"));
  print(round(times,2));  #paste(times);
  #**************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION pc04_toeplitzmatrixF5()
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("a", "b", "c", "gcd2")
  #
  #PC04. Toeplitz Matrix (Loops)************************************************
  b <- 0
  for (i in 1:runs) {
    b <- rep(0, 500*500); dim(b) <- c(500, 500)
    invisible(gc())
    times[i] <- system.time({
    for (j in 1:500) {
      for (k in 1:500) {
```

```
      b[k,j] <- abs(j - k) + 1
      }
    }
    })[3]
  }
  cat(c("\n\n\n","PC04. Toeplitz Matrix (Loops)- Results(sec) of 7 Runs:", "\n"));
  print(round(times,2));  #paste(times);
  #******************************************************************************
'
LANGUAGE plr;

CREATE OR REPLACE FUNCTION pc05_escoufiersmethodF5()
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  remove("b", "j", "k")
  #
  #PC05. Escoufiers Method*****************************************
  p <- 0; vt <- 0; vr <- 0; vrt <- 0; rvt <- 0; RV <- 0; j <- 0; k <- 0;
  x2 <- 0; R <- 0; Rxx <- 0; Ryy <- 0; Rxy <- 0; Ryx <- 0; Rvmax <- 0
  # Calculate the trace of a matrix (sum of its diagonal elements)
  Trace <- function(y) {sum(c(y)[1 + 0:(min(dim(y)) - 1) * (dim(y)[1] + 1)], na.rm=FALSE)}
  for (i in 1:runs) {
    x <- abs(rnorm(45*45)); dim(x) <- c(45, 45)
    invisible(gc())
    times[i] <- system.time({
    # Calculation of Escoufier equivalent vectors
    p <- ncol(x)
    vt <- 1:p
    vr <- NULL
    RV <- 1:p
    vrt <- NULL
    for (j in 1:p) {
      Rvmax <- 0
      for (k in 1:(p-j+1)) {
      x2 <- cbind(x, x[,vr], x[,vt[k]])
      R <- cor(x2)
      Ryy <- R[1:p, 1:p]
      Rxx <- R[(p+1):(p+j), (p+1):(p+j)]
      Rxy <- R[(p+1):(p+j), 1:p]
      Ryx <- t(Rxy)
      rvt <- Trace(Ryx %*% Rxy) /
      sqrt(Trace(Ryy %*% Ryy) *
      Trace(Rxx %*% Rxx))
      if (rvt > Rvmax) {
        Rvmax <- rvt
        vrt <- vt[k]
      }
      }
      vr[j] <- vrt
```

```
      RV[j] <- Rvmax
      vt <- vt[vt!=vr[j]]
    }
    })[3]

  }
  cat(c("\n\n\n","PC05. Escoufiers Method- Results(sec) of 7 Runs:", "\n"));
  print(round(times,2));  #paste(times);
'
LANGUAGE plr;
```

A.4 Codes for benchmark on DB2 DBMS

Listing A.4: Codes for benchmark on DB2 DBMS

```
//CLR with C# Part:=====================================
// public class clsbenchmarkdb2
public static void univIndb2Rexe(string name, out string text)
{
  text = name;

  REngine.SetEnvironmentVariables();
  REngine engine = REngine.GetInstance();

  string rscriptfile = "source('E:/RESEARCH/SCRIPTS/sqlserver_benchmark/";
  rscriptfile = rscriptfile + name.Trim() + "');";        //".r')";

  string[] rOutput = engine.Evaluate(rscriptfile).AsCharacter().ToArray();

  text = string.Join("", rOutput);

  engine.Dispose();
}

----Database Part:=======================================
CREATE OR REPLACE PROCEDURE univIndb2Rexe(IN inPAR VARCHAR(60), OUT outPAR VARCHAR(60))
SPECIFIC univIndb2Rexe
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
PROGRAM TYPE SUB
EXTERNAL NAME 'E:\RESEARCH\clrbenchmarkdb2\clrbenchmarkdb2\bin\Debug\clrbenchmarkdb2.dll:
   clrbenchmarkdb2.clsbenchmarkdb2!univIndb2Rexe';

GRANT EXECUTE ON PROCEDURE univIndb2Rexe TO sedem;
GRANT EXECUTE ON SPECIFIC PROCEDURE univIndb2Rexe TO sedem;
GRANT EXECUTE ON PROCEDURE univIndb2Rexe TO PUBLIC;
GRANT EXECUTE ON SPECIFIC PROCEDURE univIndb2Rexe TO PUBLIC;

CALL univIndb2Rexe ('mc01_transpose.r' , ?);
```

A.5 Codes for benchmark on MS SQL Server DBMS

Listing A.5: Codes for benchmark on SQL Server DBMS

```
//CLR with C# Part:====================================

//public class BenchmarkProc
[Microsoft.SqlServer.Server.SqlProcedure]
public static void univIndbRexeA(string name, out string text)
{
  REngine.SetEnvironmentVariables();
  REngine engine = REngine.GetInstance();

  string rscriptfile = "source('E:/RESEARCH/SCRIPTS/sqlserver_benchmark/";
  rscriptfile = rscriptfile + name.Trim() + "');";        //".r')";

  string[] rOutput = engine.Evaluate(rscriptfile).AsCharacter().ToArray();

  SqlContext.Pipe.Send(name.Trim().ToUpper() + "- Results(sec) of 7 Runs:\n");
  text = string.Join("", rOutput);

  engine.Dispose();
}

[Microsoft.SqlServer.Server.SqlProcedure]
public static void univIndbRexeB(string name, out string text)
{
  REngine.SetEnvironmentVariables();
  REngine engine = REngine.GetInstance();

  // Initializes settings:
  engine.Initialize();

  // Returning data via SqlDataReader:
  SqlConnection sqlConn = new SqlConnection("context connection=true");
  SqlCommand sqlCmd = new SqlCommand("SELECT * FROM stockHist", sqlConn);
  sqlConn.Open();

  DataTable sqlDt = new DataTable();
  sqlDt.Load(sqlCmd.ExecuteReader());

  int ncol = sqlDt.Columns.Count;
  int nrow = sqlDt.Rows.Count;

  // .NET Framework array to R vector:
  NumericMatrix datStockHist = engine.CreateNumericMatrix(nrow,ncol);

  for (int row = 0; row < sqlDt.Rows.Count; ++row)
  {
  for (int col = 0; col < sqlDt.Columns.Count; col++)
```

```
  {
    datStockHist[row, col] = (double)sqlDt.Rows[row][col];
  }
  }

  //Pass the data to R environment
  engine.SetSymbol("datStockHist", datStockHist);

  sqlConn.Close();

  //Call source R file script to take over
  string rscriptfile = "source('E:/RESEARCH/SCRIPTS/sqlserver_benchmark/";
  rscriptfile = rscriptfile + name.Trim() + "');";        //".r')";

  string[] rOutput = engine.Evaluate(rscriptfile).AsCharacter().ToArray();

  SqlContext.Pipe.Send(name.Trim().ToUpper() +"- Results(sec) of 7 Runs:\n");
  text = string.Join("", rOutput);

  engine.Dispose();
}

----Database Part====================================
ALTER DATABASE Benchmarkdb SET TRUSTWORTHY ON;
--ALTER DATABASE Benchmarkdb SET TRUSTWORTHY OFF;

USE Benchmarkdb
GO
EXEC sp_changedbowner 'sa'
ALTER DATABASE Benchmarkdb SET TRUSTWORTHY ON

CREATE ASSEMBLY sqlclrBenchmark
FROM 'E:\RESEARCH\clrbenchmark\clrbenchmark\bin\Debug\clrbenchmark.dll'
WITH PERMISSION_SET = UNSAFE;

CREATE PROCEDURE sp_univIndbRexeA
@inPAR nchar(500),
@outPAR nchar(500) OUTPUT AS
EXTERNAL NAME sqlclrBenchmark.[clrbenchmark.BenchmarkProc].univIndbRexeA;

CREATE PROCEDURE sp_univIndbRexeB
@inPAR nchar(500),
@outPAR nchar(500) OUTPUT AS
EXTERNAL NAME sqlclrBenchmark.[clrbenchmark.BenchmarkProc].univIndbRexeB;

DECLARE @outPAR nchar(500)
EXEC sp_univIndbRexeA @inPAR='pc01_fibonaccinumbers.r', @outPAR=@outPAR output
PRINT @outPAR;

DECLARE @outPAR nchar(500)
EXEC sp_univIndbRexeB @inPAR='mc01_transpose.r', @outPAR=@outPAR output
PRINT @outPAR;
```

A.6 Codes for timing data retrieval in Oracle and Postgres

Listing A.6: Timing retrieval of data for analytics with R in Oracle and PostgreSQL

```
--Code for timing retreival of data in Oracle
----------------------------------------------

begin
sys.rqScriptCreate('pass_data',
 'function(dat){

  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  a <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <- (dat);
    #a <- as.matrix(dat);
    })[3]
  }
  #cat(c("\n\n\n","Retrieving Data- Results(sec) of 7 Runs:", "\n"));
  data.frame(Runtimes=t(round(times,2))); #data.frame(Runtimes=round(times,2));
  #*********************************************************************************
  }');
end;

select * from table(rqTableEval(
   cursor(select * from STOCKHIST),
   NULL,
   'select 1 Run1 from dual',
  'pass_data'));



--Code for timing retrieval of data in PostgreSQL
------------------------------------------------

CREATE OR REPLACE FUNCTION pass_data(text)
RETURNS double precision[] AS
'
  runs <- 1   # Number of times the tests are ran
  times <- rep(0.00, 1);  # Initialize result vector for the runs

  a <- 0;
  for (i in 1:runs) {
    invisible(gc())
    times[i] <- system.time({
    a <<- pg.spi.exec(arg1);
```

```
    #a <<- as.matrix(pg.spi.exec(arg1));
    })[3]
  }
  cat(c("\n\n\n","Retrieving Data- Results(sec) of 1 Runs:", "\n"));
  return(round(times, 2));  #print(round(times,2)) OR paste(times);
'
LANGUAGE plr;

SELECT  pass_data('SELECT * FROM stockHist');
```

Appendix B

Benchmark Results

B.1 Matrix Calculation (MC) Benchmark Results

The table below shows the detail results obtained from the run of the benchmarks for Matrix Calculation.

MC01	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run7
Stand-Alone-R	13.71	9.84	8.85	8.81	8.78	8.75	8.78
PostgreSQL	18.62	18.51	18.63	18.63	18.80	18.64	18.53
Oracle	0.42	0.58	0.49	0.41	0.40	0.40	0.41
DB2	6.25	3.85	3.45	3.41	3.33	3.34	3.41
SQL Server	6.75	3.72	3.14	3.09	3.08	3.08	3.08
MC02	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run7**
Stand-Alone R	14.20	14.79	15.04	15.11	15.03	14.92	14.92
PostgreSQL	23.92	23.96	23.92	23.99	23.95	23.94	23.96
Oracle	6.02	6.00	5.80	5.81	5.89	5.81	5.80
DB2	8.92	9.00	9.03	9.04	9.05	9.00	9.03
SQL Server	8.52	9.02	8.98	8.91	8.98	9.03	8.87
MC03	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run7**
Stand-Alone R	9.94	10.07	10.34	10.03	10.35	10.00	10.44
PostgreSQL	19.39	19.42	19.34	19.34	19.46	19.43	19.44
Oracle	1.37	1.35	1.22	1.22	1.21	1.22	1.21
DB2	4.24	4.29	4.45	4.43	4.41	4.46	4.50
SQL Server	4.02	4.25	4.32	4.33	4.30	4.31	4.25
MC04	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run7**
Stand-Alone R	16.27	16.61	16.56	16.59	16.59	16.56	16.57
PostgreSQL	25.82	25.78	25.83	25.79	25.92	25.88	25.74
Oracle	7.81	7.69	7.68	7.69	7.71	7.67	7.69
DB2	10.56	10.48	10.55	10.48	10.83	10.44	10.53
SQL Server	10.39	10.64	10.72	10.72	10.83	10.75	10.74
MC05	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run7**
Stand-Alone R	58.39	57.70	57.67	57.69	57.58	57.74	57.78
PostgreSQL	77.16	77.10	77.17	77.33	77.11	77.32	77.34
Oracle	41.15	41.00	40.92	40.92	40.93	40.87	41.00
DB2	46.45	51.63	49.04	47.52	49.17	47.48	49.57
SQL Server	46.64	46.04	46.21	45.95	46.00	46.16	46.07

Table B.1: Matrix Calculation (MC) benchmark results (time in seconds)

B.2 Matrix Functions (MF) Benchmark Results

The table below shows the detail results obtained from the run of the benchmarks for Matrix Function.

MF01	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run7**
Stand-Alone R	13.12	13.12	12.33	12.28	12.09	12.12	12.07
PostgreSQL	22.78	22.65	22.67	22.62	22.63	22.84	22.68
Oracle	4.27	4.14	4.10	4.17	4.11	4.13	4.15
DB2	7.92	8.69	7.77	10.86	7.58	6.94	6.84
SQL Server	6.66	7.15	6.66	6.62	6.67	6.61	6.58
MF02	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run7**
Stand-Alone R	100.86	100.26	101.15	101.03	101.17	101.15	101.22
PostgreSQL	111.48	110.89	110.78	111.17	110.75	110.78	110.70
Oracle	94.16	94.14	97.55	105.01	105.37	104.81	102.41
DB2	94.72	95.33	95.29	95.53	95.66	95.29	95.09
SQL Server	94.16	94.88	94.97	95.27	94.97	95.23	94.89
MF03	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run7**
Stand-Alone R	19.30	19.20	19.25	19.20	19.21	19.20	19.14
PostgreSQL	28.51	28.61	28.63	28.41	28.42	28.50	28.48
Oracle	10.45	10.31	10.28	10.33	10.30	10.31	10.37
DB2	13.30	13.47	13.51	13.49	13.28	13.27	13.51
SQL Server	13.45	13.43	13.52	13.41	13.54	13.41	13.47
MF04	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run7**
Stand-Alone R	48.66	48.30	48.56	48.41	48.58	48.51	48.48
PostgreSQL	57.96	58.02	58.06	57.91	58.00	57.86	57.95
Oracle	40.31	40.13	40.11	40.11	40.07	40.06	40.12
DB2	42.74	42.72	43.04	42.59	42.74	42.90	42.79
SQL Server	42.75	42.56	42.63	42.37	42.46	42.39	42.56
MF05	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run7**
Stand-Alone R	58.72	59.05	58.59	58.90	58.76	58.78	58.74
PostgreSQL	68.65	68.68	68.59	68.64	68.80	68.89	68.54
Oracle	51.78	51.53	51.42	53.05	51.62	51.40	51.32
DB2	53.42	53.55	53.62	53.59	53.66	53.62	53.33
SQL Server	52.97	53.40	53.14	53.11	53.17	53.16	53.20
MF06	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run7**
Stand-Alone R	67.41	67.61	67.64	67.61	67.75	67.94	67.94
PostgreSQL	76.85	77.00	76.95	76.75	76.86	76.96	76.94
Oracle	59.58	59.29	59.43	59.42	59.51	59.50	59.42
DB2	62.26	61.39	61.59	61.50	61.46	61.50	61.40
SQL Server	60.83	61.95	61.28	62.45	61.81	61.72	61.50
MF07	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run7**
Stand-Alone R	216.48	215.68	214.96	214.95	214.77	214.62	214.78
PostgreSQL	225.39	228.48	226.34	226.50	225.56	226.27	225.95
Oracle	213.78	213.02	212.22	212.68	212.56	212.56	212.33
DB2	210.85	210.06	214.64	210.84	222.93	210.58	213.97
SQL Server	208.35	208.14	207.88	209.94	209.60	211.12	209.80
MF08	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run7**
Stand-Alone R	47.48	47.16	47.78	47.70	47.70	47.66	47.78
PostgreSQL	57.93	57.69	57.56	57.97	57.86	57.72	57.54
Oracle	39.55	39.42	39.26	39.81	39.50	39.33	39.38
DB2	41.78	42.63	42.55	42.50	42.61	42.58	42.87
SQL Server	45.14	42.19	42.25	42.25	42.14	42.89	44.59

Table B.2: Matrix Functions (MF) benchmark results (time in seconds)

B.3 Program Control (PC) Benchmark Results

The table below shows the detail results obtained from the run of the benchmarks for Program Control.

PC01	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run7
Stand-Alone R	2.68	2.70	2.73	2.70	2.70	2.69	2.75
PostgreSQL	2.75	2.74	2.75	2.72	2.78	2.87	2.85
Oracle	2.80	2.77	2.77	2.75	2.75	2.77	2.76
DB2	2.70	2.69	2.72	2.70	2.69	2.71	2.70
SQL Server	2.81	2.84	2.81	2.75	2.73	2.72	2.73

PC02	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run7
Stand-Alone R	0.39	0.33	0.26	0.27	0.28	0.28	0.28
PostgreSQL	0.39	0.40	0.39	0.40	0.41	0.39	0.39
Oracle	0.33	0.32	0.29	0.31	0.32	0.29	0.29
DB2	0.26	0.33	0.28	0.28	0.27	0.26	0.27
SQL Server	0.29	0.33	0.28	0.28	0.28	0.28	0.28

PC03	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run7
Stand-Alone R	0.27	0.47	0.84	0.75	0.58	0.89	0.62
PostgreSQL	0.59	0.64	0.63	0.54	0.61	0.60	0.60
Oracle	0.34	0.44	0.46	0.46	0.40	0.47	0.45
DB2	0.28	0.50	0.84	0.64	0.86	0.76	0.66
SQL Server	0.28	0.53	0.62	0.70	0.84	0.82	0.84

PC04	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run7
Stand-Alone R	0.52	0.52	0.50	0.50	0.51	0.55	0.50
PostgreSQL	0.50	0.50	0.48	0.50	0.50	0.49	0.50
Oracle	0.52	0.51	0.50	0.52	0.51	0.52	0.52
DB2	0.53	0.58	0.53	0.53	0.51	0.52	0.52
SQL Server	0.51	0.52	0.52	0.50	0.51	0.52	0.52

PC05	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run7
Stand-Alone R	0.37	0.37	0.38	0.38	0.38	0.42	0.36
PostgreSQL	0.36	0.39	0.37	0.39	0.37	0.38	0.38
Oracle	0.36	0.36	0.37	0.36	0.36	0.36	0.35
DB2	0.36	0.37	0.37	0.43	0.37	0.37	0.39
SQL Server	0.36	0.43	0.38	0.37	0.37	0.37	0.37

Table B.3: Program Control (PC) benchmark results (time in seconds)

B.4 Summary (averages) of Benchmark Results

The table below shows the summary of results obtained from the run of the benchmarks.

Benchmark	Stand-Alone R	PostgreSQL	Oracle	DB2	SQL Server
MC01	9.65	18.62	0.44	3.86	3.71
MC02	14.86	23.95	5.88	9.01	8.90
MC03	10.17	19.40	1.26	4.40	4.25
MC04	16.54	25.82	7.71	10.55	10.68
MC05	57.79	77.22	40.97	48.69	46.15
MF01	12.45	22.70	4.15	8.09	6.71
MF02	100.98	110.94	100.49	95.27	94.91
MF03	19.21	28.51	10.34	13.40	13.46
MF04	48.50	57.97	40.13	42.79	42.53
MF05	58.79	68.68	51.73	53.54	53.16
MF06	67.70	76.90	59.45	61.59	61.65
MF07	215.18	226.36	212.74	213.41	209.26
MF08	47.61	57.75	39.46	42.50	43.06
PC01	2.71	2.78	2.77	2.70	2.77
PC02	0.30	0.40	0.31	0.28	0.29
PC03	0.63	0.60	0.43	0.65	0.66
PC04	0.51	0.50	0.51	0.53	0.51
PC05	0.38	0.38	0.36	0.38	0.38

Table B.4: Average runtime results (time in seconds)

Benchmark	Dmax	Dmin	Nmax	Nmin	Stand Alone R	Postgre SQL	Oracle	DB2	SQL Server
MC01	18.62	0.44	10.00	1.00	5.56	10.00	1.00	2.69	2.62
MC02	23.95	5.88	10.00	1.00	5.47	10.00	1.00	2.56	2.50
MC03	19.40	1.26	10.00	1.00	5.42	10.00	1.00	2.56	2.48
MC04	25.82	7.71	10.00	1.00	5.39	10.00	1.00	2.41	2.48
MC05	77.22	40.97	10.00	1.00	5.18	10.00	1.00	2.92	2.29
MF01	22.70	4.15	10.00	1.00	5.03	10.00	1.00	2.91	2.24
MF02	110.94	94.91	10.00	1.06	4.44	10.00	4.17	1.26	1.06
MF03	28.51	10.34	10.00	1.00	5.39	10.00	1.00	2.52	2.55
MF04	57.97	40.13	10.00	1.00	5.22	10.00	1.00	2.34	2.21
MF05	68.68	51.73	10.00	1.00	4.75	10.00	1.00	1.96	1.76
MF06	76.90	59.45	10.00	1.00	5.26	10.00	1.00	2.10	2.13
MF07	226.36	209.26	10.00	1.02	4.13	10.00	2.84	3.20	1.02
MF08	57.75	39.46	10.00	1.00	5.01	10.00	1.00	2.50	2.77
PC01	2.78	2.70	10.00	1.03	2.15	10.00	8.88	1.03	8.88
PC02	0.40	0.28	10.00	1.11	2.59	10.00	3.33	1.11	1.85
PC03	0.66	0.43	10.00	1.00	8.83	7.65	1.00	9.61	10.00
PC04	0.53	0.50	10.00	1.02	4.01	1.02	4.01	10.00	4.01
PC05	0.38	0.36	10.00	1.00	10.00	10.00	1.00	10.00	10.00

Table B.5: Normalised-average runtime results (normalised to range [1,10])

Benchmark	Stand-Alone-R	PostgreSQL	Oracle	DB2	SQL Server
MC	27.02	50.00	5.00	13.14	12.37
MF	39.23	80.00	13.02	18.79	15.74
PC	27.58	38.67	18.22	31.74	34.74
OVERALL	**93.82**	**168.67**	**36.24**	**63.67**	**62.85**

Table B.6: Overall normalised average runtime results

B.5 Summary (minimum) of Benchmark Categories

The table below gives the summary of results obtained from the benchmarks categories.

Benchmark	Stand-Alone-R	PostgreSQL	Oracle	DB2	SQLServer
MC01	8.75	18.51	0.40	3.33	3.08
MC02	14.20	23.92	5.80	8.92	8.52
MC03	9.94	19.34	1.21	4.24	4.02
MC04	16.27	25.74	7.67	10.44	10.39
MC05	57.58	77.10	40.87	46.45	45.95
MF01	12.07	22.62	4.10	6.84	6.58
MF02	100.26	110.70	94.14	94.72	94.16
MF03	19.14	28.41	10.28	13.27	13.41
MF04	48.30	57.86	40.06	42.59	42.37
MF05	58.59	68.54	51.32	53.33	52.97
MF06	67.41	76.75	59.29	61.39	60.83
MF07	214.62	225.39	212.22	210.06	207.88
MF08	47.16	57.54	39.26	41.78	42.14
PC01	2.68	2.72	2.75	2.69	2.72
PC02	0.26	0.39	0.29	0.26	0.28
PC03	0.27	0.54	0.34	0.28	0.28
PC04	0.50	0.48	0.50	0.51	0.50
PC05	0.36	0.36	0.35	0.36	0.36

Table B.7: Minimum runtime results (time in seconds)

Benchmark	Dmax	Dmin	Nmax	Nmin	Stand-Alone-R	Postgre SQL	Oracle	DB2	SQL Server
MC01	18.51	0.40	10.00	1.00	5.15	10.00	1.00	2.46	2.33
MC02	23.92	5.80	10.00	1.00	5.17	10.00	1.00	2.55	2.35
MC03	19.34	1.21	10.00	1.00	5.33	10.00	1.00	2.50	2.39
MC04	25.74	7.67	10.00	1.00	5.28	10.00	1.00	2.38	2.35
MC05	77.10	40.87	10.00	1.00	5.15	10.00	1.00	2.39	2.26
MF01	22.62	4.10	10.00	1.00	4.87	10.00	1.00	2.33	2.21
MF02	110.70	94.14	10.00	1.00	4.33	10.00	1.00	1.32	1.01
MF03	28.41	10.28	10.00	1.00	5.40	10.00	1.00	2.48	2.55
MF04	57.86	40.06	10.00	1.00	5.17	10.00	1.00	2.28	2.17
MF05	68.54	51.32	10.00	1.00	4.80	10.00	1.00	2.05	1.86
MF06	76.75	59.29	10.00	1.00	5.19	10.00	1.00	2.08	1.79
MF07	225.39	207.88	10.00	1.00	4.46	10.00	3.23	2.12	1.00
MF08	57.54	39.26	10.00	1.00	4.89	10.00	1.00	2.24	2.42
PC01	2.75	2.68	10.00	1.00	1.00	6.14	10.00	2.29	6.14
PC02	0.39	0.26	10.00	1.00	1.00	10.00	3.08	1.00	2.38
PC03	0.54	0.27	10.00	1.00	1.00	10.00	3.33	1.33	1.33
PC04	0.51	0.48	10.00	1.00	7.00	1.00	7.00	10.00	7.00
PC05	0.36	0.35	10.00	1.00	10.00	10.00	1.00	10.00	10.00

Table B.8: Normalised-minimum run-time results (normalised to range [1,10])

Benchmark	Stand-Alone-R	PostgreSQL	Oracle	DB2	SQL Server
MC	26.09	50.00	5.00	12.28	11.69
MF	39.10	80.00	10.23	16.90	15.01
PC	20.00	37.14	24.41	24.62	26.86
OVERALL	***85.19***	***167.14***	***39.64***	***53.80***	***53.57***

Table B.9: Overall normalised minimum runtime results

B.6 Chart of minimum run-times benchmark results

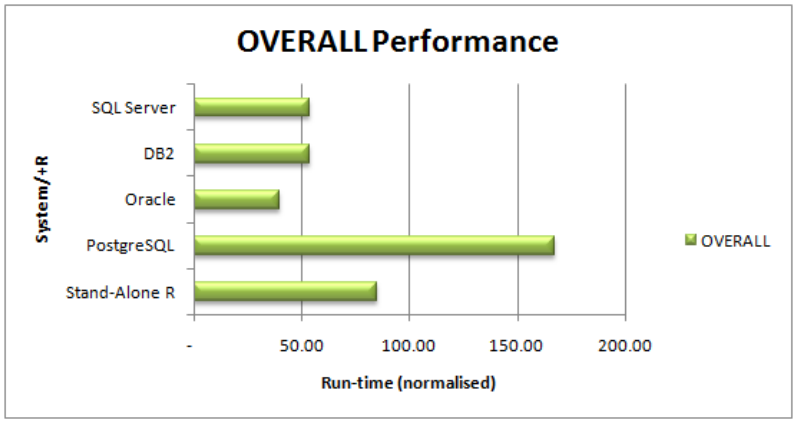

Figure B.1: Chart of overall benchmark results (minimum run-times)

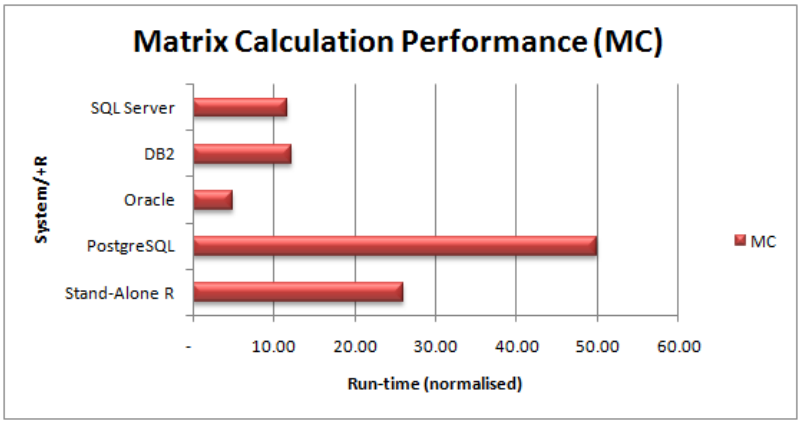

Figure B.2: Chart of Matrix Calculation (MC) benchmark results (minimum run-times)

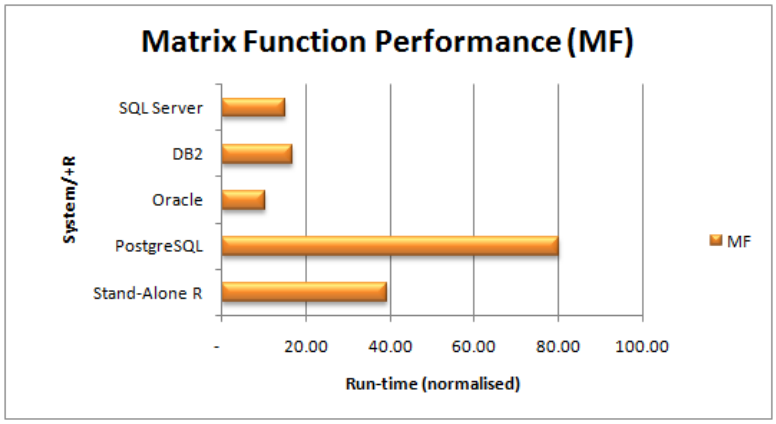

Figure B.3: Chart of Matrix Function (MF) benchmark results (minimum run-times)

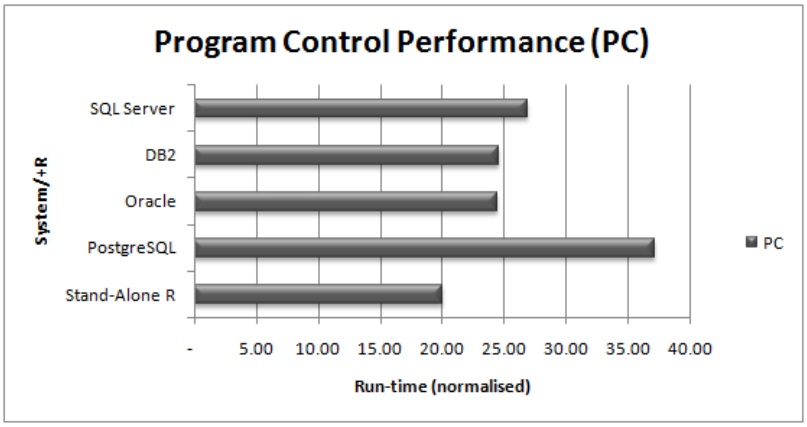

Figure B.4: Chart of Program Control (PC) benchmark results (minimum run-times)

B.7 Scalability Test Results

The table below shows the results obtained from the scalability test runs.

R-Benchmark	***1-million-cells-r***	***4-million-cell-r***	***16-million-cells-r***	***dTimes1-4mc-r***	***dTimes1-16mc-r***	***dTimes4-16mc-r***
MC1	0.72	2.49	9.65	1.77	8.93	7.16
MC2	1.02	3.65	14.86	2.63	13.84	11.21
MC3	0.76	2.58	10.17	1.82	9.41	7.59
MC4	1.15	4.15	16.54	3.00	15.39	12.39
MC5	2.19	9.74	57.79	7.55	55.60	48.05
MF1	0.75	2.87	12.45	2.12	11.70	9.58
MF2	2.61	15.42	100.98	12.81	98.37	85.56
MF3	0.99	3.63	19.21	2.64	18.22	15.58
MF4	1.38	7.31	48.50	5.93	47.12	41.19
MF5	1.66	9.21	58.79	7.55	57.13	49.58
MF6	1.72	9.77	67.70	8.05	65.98	57.93
MF7	3.86	28.00	215.18	24.14	211.32	187.18
MF8	3.63	14.69	47.61	11.06	43.98	32.92

Oracle-Benchmarks	***1-million-cells-ore***	***4-million-cell-ore***	***16-million-cells-ore***	***dTimes1-4mc-ore***	***dTimes1-16mc-ore***	***dTimes4-16mc-ore***
MC1	0.04	0.13	0.44	0.09	0.40	0.31
MC2	0.37	1.48	5.88	1.11	5.51	4.40
MC3	0.10	0.34	1.26	0.24	1.16	0.92
MC4	0.48	1.93	7.71	1.45	7.23	5.78
MC5	0.72	5.26	40.97	4.54	40.25	35.71
MF1	0.15	0.73	4.15	0.58	4.00	3.42
MF2	1.91	13.34	100.49	11.43	98.58	87.15
MF3	0.18	1.30	10.34	1.12	10.16	9.04
MF4	0.66	5.13	40.13	4.47	39.47	35.00
MF5	0.86	6.97	51.73	6.11	50.87	44.76
MF6	0.92	7.60	59.45	6.68	58.53	51.85
MF7	3.20	26.45	212.74	23.25	209.54	186.29
MF8	2.74	12.44	39.46	9.70	36.72	27.02

Table B.10: Results of scalability test runs

The following figures show charts of difference in run-times for different data load scalability tests.

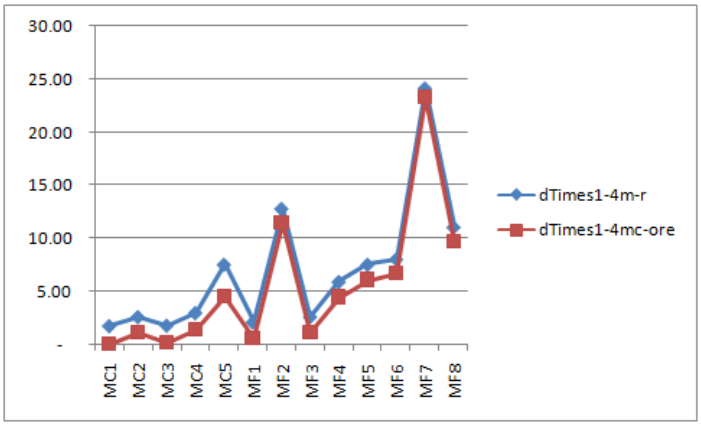

Figure B.5: Scalability result chart- dataset increase from 1 to 4 million cells

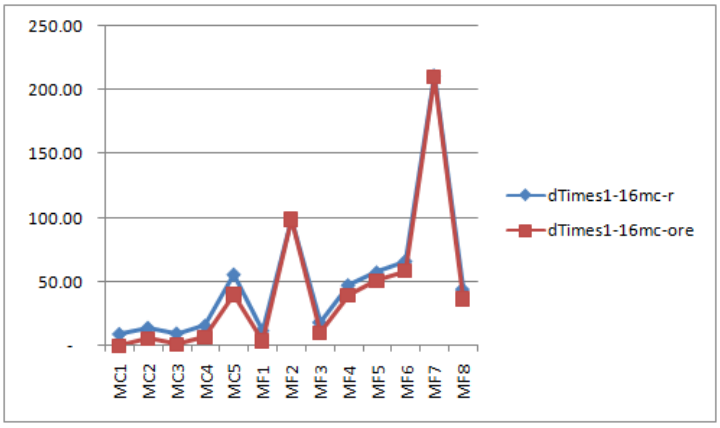

Figure B.6: Scalability result chart- dataset increase from 1 to 16 million cells

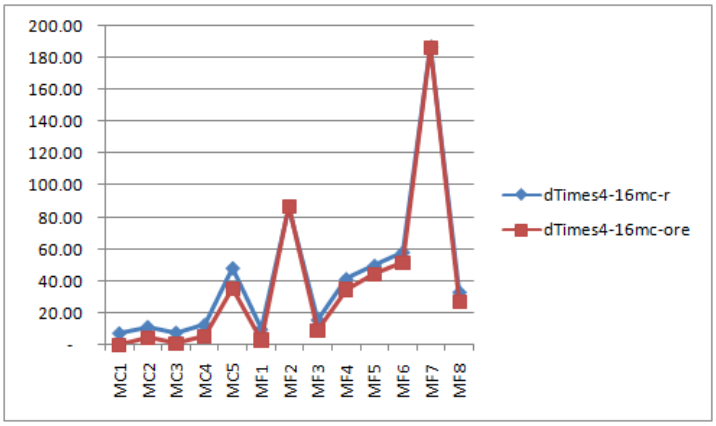

Figure B.7: Scalability result chart- dataset increase from 4 to 16 million cells

Coded Name	Full Meaning
1-million-cells-r	Average run-times for 1 million cells in R (sec)
4-million-cell-r	Average run-times for 4 million cells in R (sec)
16-million-cells-r	Average run-times for 16 million cells in R (sec)
dTimes1-4mc-r	Difference in run-times between 1- and 4- million cells in R (sec)
dTimes1-16mc-r	Difference in run-times between 1- and 16- million cells in R (sec)
dTimes4-16mc-r	Difference in run-times between 4- and 16- million cells in R (sec)
1-million-cells-ore	Average run-times for 1 million cells in Oracle (sec)
4-million-cell-ore	Average run-times for 4 million cells in Oracle (sec)
16-million-cells-ore	Average run-times for 16 million cells in Oracle (sec)
dTimes1-4mc-ore	Difference in run-times between 1- and 4- million cells in Oracle (sec)
dTimes1-16mc-ore	Difference in run-times between 1- and 16- million cells in Oracle (sec)
dTimes4-16mc-ore	Difference in run-times between 4- and 16- million cells in Oracle (sec)

Table B.11: Legend for coded names in scalability test result tables and charts

References

[1] Revolution Analytics. URL: http://www.revolutionanalytics.com/.

[2] Revolution Analytics. *Revolution RevoR Enterprise Benchmark Details: Benchmark Scripts*. URL: http://www.revolutionanalytics.com/revolution-revor-enterprise-benchmark-details.

[3] Jitender Aswani and Jens Doerpmund. *Advanced Analytics with R and SAP HANA*. [Online; accessed 03-March-2014]. 2012. URL: http://www.slideshare.net/JitenderAswani/na-6693-r-and-sap-hana-dkom-jitenderaswanijensdoeprmund.

[4] Jitender Aswani and Jens Doerpmund. *Advanced Analytics with R and SAP HANA: LETS TALK CODE*. [Online; accessed 05-March-2014]. 2011. URL: http://datatable.r-forge.r-project.org/randsaphana-dkom.pdf.

[5] L BACHELIER. "THE THEORY OF SPECULATION". In: ().

[6] Louis Bachelier. *Théorie de la spéculation*. Gauthier-Villars, 1900.

[7] Charlie Berger. *In-Database Analytics: Statistics and Advanced Analytics with R-Oracle R Enterprise*. ORACLE, 2011. URL: http://www.oracle.com/technetwork/database/options/odm/oracle-r-enterprise-oow11-517498.pdf.

[8] Roger Bivand and Albrecht Gebhardt. "Implementing functions for spatial statistical analysis using the language". In: *Journal of Geographical Systems* 2.3 (2000), pp. 307–317.

[9] Roger Bivand and Markus Neteler. "Open source geocomputation: using the R data analysis language integrated with GRASS GIS and PostgreSQL data base systems". In: *Proceedings of the 5th International Conference on GeoComputation*. 2000.

[10] *Bringing R to the Enterprise*. Tech. rep. Also available as http://www.oracle.com/technetwork/database/options/advanced-analytics/r-enterprise/bringing-r-to-the-enterprise-1956618.pdf. 2013.

[11] Robert Catterall. *z/OS Stored Procedures- Trends and Technology*. [Online; accessed 25-March-2014]. IBM, 2011. URL: http://www.bwdb2ug.org/PDF/DB2_for_zOS_Stored_Procedures_Trends_and_Technology.pdf.

[12] Elaine Chen. *Using R and Tableau*. Tech. rep. Also available as http://www.tableausoftware.com/sites/default/files/media/using-r-and-tableau-software_0.pdf. 2013.

[13] Linda Claussen. *DB2 Native SQL Procedures: The Future of Computing?* IBM. 2012. URL: http://ibmdatamag.com/2012/08/db2-native-sql-procedures-the-future-of-computing.

[14] *Cloudera Impala*. Cloudera, Inc. URL: http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html.

[15] Joseph E Conway. *PL/R User's Guide - R Procedural Language*. Available at http://www.joeconway.com/plr/doc/plr-US.pdf. 2003–2009.

[16] John W Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage, 2013.

[17] Chris Ding and Xiaofeng He. "K-means clustering via principal component analysis". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 29.

[18] Xixuan Feng, Arun Kumar, Benjamin Recht, and Christopher Ré. "Towards a unified architecture for in-RDBMS analytics". In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM. 2012, pp. 325–336.

[19] Peter Filzmoser, Moritz Gschwandtner, Maintainer Peter Filzmoser, and TRUE LazyData. "Package 'mvoutlier'". In: (2013).

[20] Ronald A Fisher. "The use of multiple measurements in taxonomic problems". In: *Annals of eugenics* 7.2 (1936), pp. 179–188.

[21] *Genesis*. URL: http://cran.r-project.org/doc/html/interface98-paper/paper_1.html.

[22] Edwin Grappin. *Generate stock option prices - How to simulate a Brownian motion*. http://probaperception.blogspot.com.es/2012/10/generate-stock-option-prices-how-to.html. Blog. 2012.

[23] Philippe Grosjean. *R Benchmark 2.4*. 2007. URL: http://www-cs-faculty.stanford.edu/~uno/abcde.html.

[24] Philippe Grosjean. *R Benchmark 2.5*. 2008. URL: http://r.research.att.com/benchmarks/R-benchmark-25.R.

[25] Joseph M Hellerstein, Christoper Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. "The MADlib analytics library: or MAD skills, the SQL". In: *Proceedings of the VLDB Endowment* 5.12 (2012), pp. 1700–1711.

[26] Mark Hornick and Tim Vlamis. *Oracle R Enterprise Hands-on Lab*. [Online; accessed 12-March-2014]. ORACLE, 2014. URL: http://www.vlamis.com/storage/papers/Hornick%20-%20ORE%20Hands-On%20Lab.pdf.

[27] Grant Hutchison. *Using R with databases*. IBM. 2014. URL: http://www.ibm.com/developerworks/data/library/techarticle/dm-1402db2andr/dm-1402db2andr-pdf.pdf.

[28] Robert Kabacoff. *R in Action: Data analysis and graphics with R*. Manning Publications Co., 2011.

[29] Donald Knuth. *Comparison of mathematical programs for data analysis*. 2008. URL: http://www.scientificweb.com/ncrunch/ncrunch5.pdf.

[30] Sherry LaMonica. *Introduction to ORE Embedded R Script Execution*. Oracle. 2012. URL: https://blogs.oracle.com/R/entry/analyzing_big_data_using_the1.

[31] Geoffrey McLachlan. *Discriminant Analysis and Statistical Pattern Recognition*. Vol. 544. John Wiley & Sons, 2004.

[32] *Oracle Database Installation Guide*. Also available as http://docs.oracle.com/cd/E11882_01/install.112/e47689.pdf. ORACLE. 2013.

[33] *Oracle R Enterprise Installation and Administration Guide*. Also available as http://docs.oracle.com/cd/E11882_01/doc.112/e36763.pdf. ORACLE. 2013.

[34] *Oracle R Enterprise User's Guide*. Also available as http://docs.oracle.com/cd/E11882_01/doc.112/e36761.pdf. ORACLE. 2013.

[35] *Real-time Analytics in the Capital Markets: Sybase RAP Real-time Data Collection and Historical Analysis*. Tech. rep. Also available as http://www.sybase.es/files/White_Papers/SY-RAP-Real-time-Analytics-in-Capital-Markets-WP.pdf. 2012.

[36] *R.NET*. Microsoft. URL: https://rdotnet.codeplex.com/.

[37] *Running Time of R*. URL: http://stat.ethz.ch/R-manual/R-devel/library/base/html/proc.time.html.

[38] *SAP HANA R Integration Guide*. 1st ed. Also available as http://help.sap.com/hana/SAP_HANA_R_Integration_Guide_en.pdf. SAP. 2014.

[39] David Smith. *R integrated throughout the enterprise analytics stack*. http://blog.revolutionanalytics.com/2012/02/r-in-the-enterprise.html/. Blog. 2012.

[40] Michael G Solomon and Mike Chapple. *Information security illuminated*. Jones & Bartlett Learning, 2005.

[41] Stefan Steinhaus. *Comparison of mathematical programs for data analysis*. 1999.

[42] Michael Stonebraker. *What Does 'Big Data' Mean? (Part 2)*. http://cacm.acm.org/blogs/blog-cacm/156102-what-does-big-data-mean-part-2/fulltext. Blog. 2012.

[43] Gary Stoneburner. "SP 800-33. Underlying Technical Models for Information Technology Security". In: (2001).

[44] Bharat Sukhwani, Hong Min, Mathew Thoennes, Parijat Dube, Balakrishna Iyer, Bernard Brezzo, Donna Dillenberger, and Sameh Asaad. "Database analytics acceleration using FPGAs". In: *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM. 2012, pp. 411–420.

[45] Pete Swabey. *Putting the R in Analytics*. 2012. URL: http://www.information-age.com/technology/information-management/2099883/putting-the-r-in-analytics.

[46] István Szegedi. *Integrating R with Cloudera Impala for Real-Time Queries on Hadoop*. 2013. URL: http://bighadoop.wordpress.com/2013/11/25/integrating-r-with-cloudera-impala-for-real-time-queries-on-hadoop/.

[47] *Understanding ODBC Architecture*. Microsoft. URL: http://msdn.microsoft.com/en-us/library/aa266933(v=vs.60).aspx.

[48] Simon Urbanek. "A fast way to provide R functionality to applications". In: *Proceedings of DSC*. Citeseer. 2003, p. 2.

[49] Michael Whitman and Herbert Mattord. *Principles of information security*. Cengage Learning, 2011.

[50] Wikipedia. URL: http://en.wikipedia.org/wiki/Main_Page.

[51] Robert Young. *Going Beyond the Relational Model with Data*. 2011. URL: https://www.simple-talk.com/content/print.aspx?article=1359.

[52] Hongyuan Zha, Xiaofeng He, Chris Ding, Ming Gu, and Horst D Simon. "Spectral relaxation for k-means clustering". In: *NIPS*. Vol. 1. 2001, pp. 1057–1064.