Journal of Information Technology Management
ISSN #1042-1319
A Publication of the Association of Management

DISTRIBUTED RELATIONAL DATABASE PERFORMANCE IN CLOUD COMPUTING: AN INVESTIGATIVE STUDY

ALAN T LITCHFIELD
AUCKLAND UNIVERSITY OF TECHNOLOGY
alan.litchfield@aut.ac.nz

AWADH ALTHWAB
AUSTRALIAN NATIONAL UNIVERSITY
u6443129@anu.edu.au

CHANDAN SHARMA
AUCKLAND UNIVERSITY OF TECHNOLOGY
chandan.sharma@aut.ac.nz

ABSTRACT

This article reports on a study that demonstrates the weak points found in major relational database engines that were set up in a Cloud Computing environment, in which the nodes were geographically distant. The study undertook to establish whether running databases in the Cloud provided operational disadvantages. Findings indicate that performance measures of RDBMS' in a Cloud Computing environment are inconsistent and that a contributing factor to poor performance is the public or shared infrastructure on the Internet. Also that RDBMS' in a Cloud Computing environment become network-bound in addition to being I/O bound. The study concludes that Cloud Computing creates an environment that negatively impacts RDBMS performance in comparison to the n-tier architecture for which common RDBMS' were designed.

Keywords: Cloud Computing, Distributed Relational Databases, Network and I/O Latency, Query Planning

INTRODUCTION

There exists a concern common to organisations that are considering the transition of geographically distributed operational systems to a Cloud Computing (CC) based environment. That following a period of significant hype about the promises that CC would provide an organisation, the reality experienced by those that have made a naive attempt to transition to the Cloud, found the performance of systems lacking. Also that when dealing with large datasets, CC is increasingly seen as a mainstream technology [31]. This creates a significant concern for the enterprise, for example, at as January 2016, survey results indicate that of a sample size of 1060 IT professionals, 77% use private cloud

services, 89% use public cloud services, 71% use hybrid, and perhaps more significantly enterprises are using multiple cloud services to satisfy their requirements [47]. Moreover, this report also states that the main challenge for cloud adoption is not security but a lack of resources and expertise.

This paper presents the results of a study focussed specifically on identifying where significant performance issues in a naively deployed database system occur. The starting point for the study is with the premise that any distributed Relational Database Management System (RDBMS) requires a network and nodes (typically the n-tier architecture), that applications of RDBMS' are normal in an organisational context, and that the performance of RDBMS has been a concern for researchers for many years. The n-tier architecture operates on a client/server model and includes a database system and at least one server application or middleware, through which users gain access to the database system [17, 15]. However, the CC architecture differs from an n-tier architecture such that CC infrastructure includes the public Internet and typically abstracts the physical architecture through virtualisation [23, 26].

In CC, the user may obtain direct access to data or via a Service-Oriented Architecture (SOA), for example through an Application Programming Interface (API) made available by a service provider. In an n-tier architecture, programmatic controls exist within a data centre and amongst its networks and servers as nodes and hooks but, unlike the public network infrastructure, these have greater and controllable bandwidth [3]. This situation, of a shared and limited bandwidth, constrains the performance of applications that run in CC [38]. Thus, for example, query optimisation in distributed RDBMS' in n-tier architecture suffer from performance issues [9, 32, 39, 43]. Therefore, whereas RDBMS' normally operate on n-tier architecture, we investigate RDBMS performance operating on a cloud-based architecture to determine where break points exist. What we have found is that all the systems up for testing failed at least one of the tests but that they failed them in different ways.

The paper is structured as follows; in the next section, related research work is presented, then the method developed for the study is described in brief, the results of experiments are presented and then discussed, and finally, in the conclusion is presented an outline of the findings and indications of further research both planned for and currently underway.

RELATED WORK

CC is abstract and encompasses various technologies and practices. Attempts to define CC typically focus on some feature set, such as classes of technology, protocols, patterns of use, and so on. Both Geelan [18] and Buyya, Yeo, and Venugopal [7] include economies of scale in definitions of CC, for example reduction of the overall cost of cloud infrastructure consumption by service consumers compared with the equivalent resource requirements off-cloud. The authors also focus on Service Level Agreement (SLA) provision to define the level of quality of service expected from either party. Additionally, scalability and the ability to optimise resource use empowers users to have full control over their spending on IT services [46]. These resources are typically consumed on a pay-per-use basis and service providers are responsible for guaranteeing infrastructure to an agreed SLA.

CC has three models of service delivery: public, private, and hybrid clouds [18]. The models differ in how they are managed, so that a Public Cloud (PuC) involves many customers accessing services from different locations and using the same infrastructure (primarily the Internet). The Private Cloud (PrC) is managed either by the organisation itself or it may be outsourced. The implications for an organisation with a PrC are significant, and this is especially important because access to its resources is more limited than a PuC. Facilitating the expansion of the PrC, using the resources of the PuC, creates a Hybrid Cloud (HyC).

An important feature of the Cloud-based environment is a high level of availability for services, data and other IT resources [31]. However, merely moving a company's computing resources to a cloud platform without sufficient feasibility assessment for accessibility and performance may lead to bottlenecks. Technical bottlenecks or network insufficiency can in turn lead to data unavailability especially when data move between cloud nodes over networks with limited bandwidth. Database locking, lack of storage capacity [for example, 44] and cache flushing can also cause bottlenecks in Cloud-based systems.

This study is focussed the effect of PuCs on the performance of relational databases in a Cloud-based environment. To review existing literature, we consider some of the wider issues associated with data base performance. Li, Yang, Kandula, and Zhang [30] conduct a comparison between PuCs and conclude that considerable differences exist between PuC providers and this makes it difficult to choose which provider to use. Further, Iosup, Ostermann, Yigitbasi, Prodan, Fahringer, and Epema [22] suggest that PuCs appear to suit small databases but that they demonstrate deficiencies when heavy workloads (such as scientific analytics) are involved, but Thakar, Szalay, Church, and

Terzis [44] and Hashem, Yaqoob, Anuar, Mokhtar, Gani, and Khan [21] disagree and state that PuCs such as Amazon Elastic Cloud Computing (EC2) and Microsoft SQL Azure are suited to scientific tasks. Gunarathne, Wu, Qiu, and Fox [20] add that PuCs can be used for big data tasks performed on high dimensional data residing on heterogeneous databases, where complex queries consume intensive computational resources. But I/O performance inconsistencies occur on PuCs due to the existence of shared infrastructure (the Internet) and the potential for improperly tuned VMs on hypervisors.

The RDBMS is a mainstay of enterprise information systems management. Since Codd [11] established relational theory, the approach to the structuring of data in relations and defining simple relationships has shown itself to be robust and flexible. In a study considering the uptake of noSQL systems, Mc Kendrick [35] states that 77% of the study's sample consider structured data as central to their daily business activities, and that 92% use RDBMS' compared to 11% that have deployed NOSQL databases. The likelihood of the RDBMS being fully replaced by other types of database is not significant, however performance issues still need to be resolved.

Query optimisation is an area that has been a focus of research, resulting in a significant body of knowledge. Choosing an efficient query plan appears complex because many variables are computed. For instance, the RDBMS has to estimate the number of tuples that a query selects and the number of tuples retrieved by every operation in the execution plan. The RDBMS also needs to estimate the computational resources required for execution so that CPU usage, I/O and memory allocation variables may be estimated. Moreover, the RDBMS may compare plans before it chooses one plan [10]. While query optimisation approaches enable multiple paths to an efficient query execution [12], Shao, Liu, Li, and Liu [42] suggest that issues exist with the performance of existing optimisation methods. The authors present a new optimisation system for MS SQL Server based on a hierarchical queuing network model. With this model, they achieve on average a 16.8% improvement in the performance of SQL Server compared with existing optimisation methods, and increases transaction throughput by 40%.

This study is not directed at the development of a new algorithm or assessing specific optimisations in a distributed environment, query optimisation presents persistent challenges [9]. However, a factor that emerges as important to the study where to enable intermediate operations that may be intended to optimise the local processing of a query and where data are moved between locations, the distributed environment adds significant complexity [8]. This leads to more variables when choosing optimisation plans [39, 27]. This is not a new problem and is not one introduced by CC, for example Liu and Yu [32] claim that in deciding whether or not it is the inefficient implementation or unsuitable execution plans chosen by RDBMS that cause long processing queries, more investigation is required. Their findings suggest that network overhead is observed to be a major influence. More recently, in the CC context, similar methods are employed for large dataset processing while noSQL databases introduce new approaches for query optimisation that appear to improve performance [48, 9].

Traditionally, RDBMS' have been deployed on the client-server model, where database systems may communicate directly with the server and network. Changes to data volume, infrastructure, and platform technologies suggest that RDBMS' appear to cope less well [48]. Conflicting views as to whether RDBMS can still be used in the era of large datasets exist. There has been a suggestion of architectural issues with the relational data model that reduce the effectiveness of RDBMS when processing large datasets in CC [31]. Supporting this view, Durham, Rosen, and Harrison [14] report that the data model can be a significant factor when handling large datasets. That is, by pulling data across the public network, RDBMS do not perform efficiently with big data and that impacts performance. The implication is that joins between distributed tables may be problematic and perhaps should be avoided. An alternative view is that it is not the architecture that is the issue but they way in which it has been deployed in the Cloud.

Relational database performance in CC

We contend that the naive deployment of RDBMS in a Cloud context has introduced additional performance challenges and so we summarise research on query performance. In general, performance is done by benchmarking tools, for example, Transaction Processing Performance Council (TPC) tools that include SPECCpu benchmark, which works to evaluate any given computer system and recommend the best CPU for the workload [16], and TPC-C that evaluates DBMS' that suit Online Transaction Processing (OLTP) applications [28]. The application of TPC-C to measure the performance of databases is extensive but in CC, there is little attention given to domain specific characteristics as they relate to RDBMS performance.

RDBMS performance within the Cloud Distributed Database (CDD) context has not been well described. For instance in their study, Minhas, Yadav, Aboulnaga, and Salem [37] fail to take into account the effect of the Internet and at 2Gb, the data size was small. Their study

concludes that running a database over virtualised environment creates I/O disk overhead but that such an overhead does not have a large impact during the runtime of queries. Thakar, Szalay, Church, and Terzis [44] have similar findings with regard to CC affecting I/O performance. Also, Ivanov, Petrov, and Buchmann [24] conclude that CC is more suitable for read-mostly work and that for other purposes like OLTP applications and data-intensive workload, CC poses challenges. However, Bose, Mishra, Sethuraman, and Taheri [5] find that although RDBMS' perform better off-Cloud, they achieve between 85% and 92% of native performance in CC and conclude that CC is capable of handling database-intensive workloads.

For the study, to assess RDBMS performance, a number of approaches are considered. From these, we have applied the following. In a performance evaluation of PuC with a range of benchmarking tools, Iosup, et al [22] focus on CPU time, I/O, and memory hierarchy. Jackson, Ramakrishnan, Muriki, Canon, Cholia, Shalf, and Wright [25] address network latency in high performance computing applications on PuC. Kohler and Specht [29] investigate RDBMS that are partitioned over a cloud network and uses two performance measures; runtime and the number of tuples returned in a query. Thakar et al [44] investigate large database query runtime performance in analytics. Lloyd, Pallickara, David, Lyon, Arabi, and Rojas [33] use a statistical model to examine CPU time and I/O operations in IaaS for multi-tier applications. Baccelli and Coffman [2] also use runtime with the addition of throughput to analyse the performance of a distributed database, looking at the effect of interrupted services from an update operation that had a pre-emptive privilege over read operations. To avoid two-phase commit, Anderson, Breitbart, Korth, and Wool [1], among other measures, use transaction throughput to measure the effect of serialisation and transaction atomicity. To examine the implementation of different strategies for distributed lock management, Born [4] employ transaction response time and transaction throughput. To study the performance of timestamp ordering concurrency control, Bouras and Spirakis [6] utilise wait time. And, to study different replication models, Gray, Helland, O'Neil, and Shasha [19] utilise wait time.

EXPERIMENTAL METHOD

To develop the experimental method, the following measures from literature are used:

Duration: The time taken by a query to complete and is calculated as the sum of processing time and communication cost. As an indicator, this indicates if an experiment presents issues and as a general rule, the shorter the duration, the fewer the issues. However, a longer duration indicates that further investigation is required [2, 4, 29].

CPU Time: Indicates CPU consumption for the duration of query execution and reported by the RDBMS. It plays a central role once data arrive from disk and is used to select join operators in a query [33, 1, 22].

Disk Operation: The number of physical reads and writes that occur during a query. Since a relational database is I/O bound, this measure provides observation data [1].

Average I/O Latency: The average time an I/O operation takes to finish. It reflects the effect that disk operation has on performance [33, 1, 45, 22].

Logical Read: Represents the number of reads that take place in memory. This measure may be partly associated with CPU time so that when there is high CPU consumption, it is accompanied by a large number of logical reads, although that is not always the case. Note, this measure is used when experiments reveal special cases [1].

Network Traffic: The amount of data that travels a network for each query [4, 25].

Wait Events: The events that systems wait for operations to finish. For example, when the system waits for a cloud network to complete an I/O operation [4, 6, 19, 1, 45].

To study how much the Internet infrastructure negatively impacts RDBMS performance, the investigation employs three database systems, SQL Server 2012, Oracle 11g and MySQL 6.5. However, the manner in which MySQL determines query plans makes it impossible to run the study's queries without significantly more computational resources for it alone. To provide distributed queries, a federated table has been determined as most suitable, however MySQL documentation states that all the records of remote federated tables are fetched and filtering clauses are applied locally. So for the data set used in this study, at least 18 GB of memory is needed to host Fact_Table (Fig. 1), as well as the time required for the data to traverse the network. In addition, when trying to load the data that had previously split into smaller datasets, MySQL seemed unable to clear its memory with each commit function, leading to server crashes. Since the study uses a standardised VM configuration with 8GB of memory (Table 1), MySQL is ruled out.

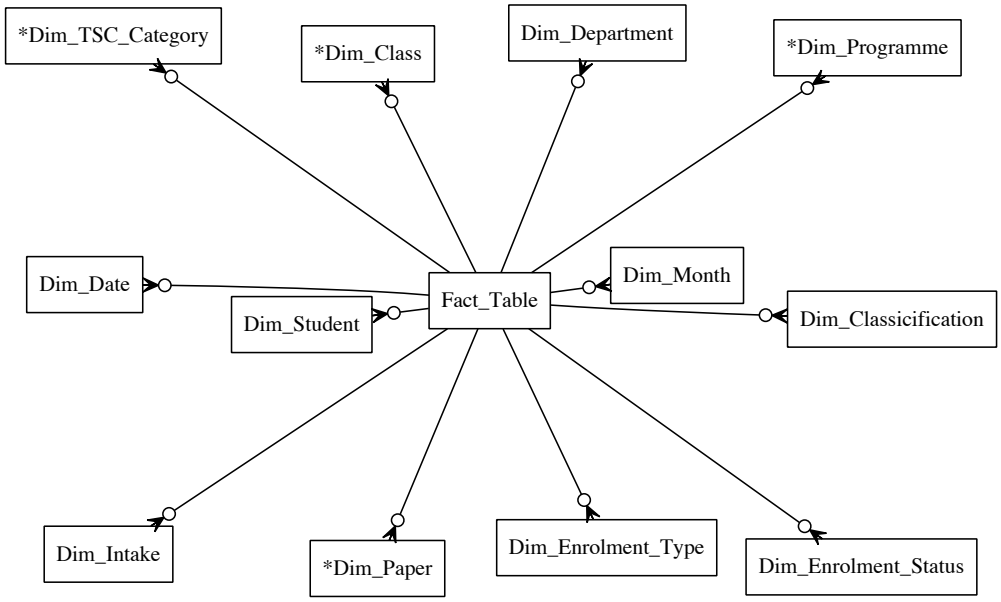

Figure 1: Star Schema

Table 1: Research environment configurations

Server Location	**Virtualization**	**VM configuration**
Amsterdam (Local)	Xenon, Quad Core x 2.13Ghz	Microsoft Windows 7 x64 Operating System, 4 CPU cores, 8 GB RAM, 200 GB of disk space
Auckland (Remote)	Xenon, Quad Core x 2.26Ghz	Microsoft Windows 7 x64 Operating System, 4 CPU cores, 8 GB RAM, 200 GB of disk space

Consequently two database systems are used, Microsoft SQL Server 2012 and Oracle 11g. The CDD includes a distributed database running in VMs on a PuC, via the Internet (Table 1). To create a real time distributed database, two systems are set up in geographically distant locations; Amsterdam (Netherlands), the "local instance" and Auckland (New Zealand), the "remote instance". Each database system and geographic location has two VMs. To obtain comparable results, all four VMs have near identical configurations for memory, CPU (there is slight difference in CPU speed) and operating system. The VMs in Amsterdam contain Fact_Table and since experiments are run from Amsterdam. The other two VMs in Auckland contain the remaining dimensions tables.

In the experiments, the study replicates the real world so that the results obtained can be generalised. To that end a dataset from the university's data warehouse is used. The dataset is large at 120Gb of comma separated values (CSV) files (Fact_Table alone was an 80Gb CSV file), relational, and contains anonymised student Equivalent Full Time Student (EFTS) records. The database consists of thirteen tables in a star schema (Fig 1). While smaller tables are imported directly to both databases, SQL insert scripts are used to insert the data from the large tables (Dim_Student and Fact_Table).

Fact_Table contains 400 million tuples, with a raw data size of 80 GB. The data files for the large tables are separated using unix `split` into files that contain some 10000 records each and then each record is concatenated with a record insert script. The data files are then fed into the database systems sequentially. If too many records are inserted at once, the database systems tend to run short of available memory, despite that each insert script is terminated with a semi-colon, `;`, or `commit` command. From

this problem and further limitations discovered while running complex queries, the dataset is reduced to 100 million tuples, a size of 18GB.

In the experiment, nine SQL queries are run (Table 2). Eight queries retrieve data and one (EXP9) is an update query. Five queries implement inner join (EXP2, EXP3, EXP4, EXP6, and EXP7) while the other three queries implement left join (EXP5), full join (EXP8) and right outer join (EXP1). The eight data retrieval queries use at least two tables, and Fact_Table is used by every query for implementing joins with the dimension tables. Thus in every experiment, data traverses the network.

Query Complexity relates to the number of join conditions and the use of clauses, such as sorting or aggregating data, in a query. The complexity level in EXP1, EXP4 and EXP5 are low. In EXP1, to look for tuples that fit the joining column in Fact_Table (two columns read) and the WHERE clause condition, the optimiser will perform at least one table scan. EXP4 focuses on the collection of a large volume of data, but with less complexity in the query whereas EXP5, assuming that the network will affect performance, is expected to take longer because two columns from Fact_Table are involved.

EXP3 and EXP9 are of medium complexity. The aim of EXP3 is to examine a common query that produces an aggregated result. One can expect this query to run faster than other experiments because it should return a small dataset. However, EXP3 involves a considerable degree of complexity with GROUP BY, HAVING and ORDER BY, in addition to the joining of two tables. EXP9 aims to see how, in order to perform an update, the databases will cope when two distributed tables are joined. It is expected that the query will not take long to complete because while the query updates many tuples, two tables are joined with the WHERE condition.

The complexity of EXP2, EXP6, EXP7 and EXP8 are high. In EXP2 there are more WHERE conditions and more tables than any other experiment, therefore EXP2 should take longer to run. While this experiment produces more data from tables that traverse the network, the actual amount of data should not be large because there are two WHERE conditions with the Fact_Table and an AND operator between WHERE conditions. In EXP6, two large tables are joined, an AND operator is used as a filtering condition, and this condition is based on a dimension table. In EXP7, three tables are joined with the OR operator, the Fact_Table between two dimension tables. It is expected that this experiment will take longer than the previous experiments, with five columns and 100 million rows involved.

Details of the experiments are as follows:

EXP1 Implements a right outer join with two tables; Dim_Student and Fact_Table. Two columns from Fact_Table, Student_Demographic_Key and TOTAL_EFTS and one column from Dim_Student, Student_Demographic_Key are used in the query. The query contains only one condition in the WHERE clause.

EXP2 Implements an inner join between four tables; Dim_Enrolment_Type, Dim_Paper, Dim_Date, and Fact_Table. The query uses nine columns from the four tables with five columns from Fact_Table and four columns from dimension tables. The query uses six conditions in the WHERE clause, therefore it is expected to take longer to run.

EXP3 Implements an inner join with two tables; Dim_Paper and Fact_Table. The query uses four columns, two from each table. The query uses GROUP BY, HAVING, ORDER BY, and DISTINCT clauses.

EXP4 Implements an inner join between two tables; Dim_Student and Fact_Table. The query uses two columns from two tables, where one column from Dim_Student and one column from Fact_Table. The expected amount of generated data is high.

EXP5 Implements a left join between two tables; Dim_Enrolment_Type and Fact_Table. The query uses three columns from two tables with two columns from Dim_Enrolment_Type and one column from Fact_Table. The query is likely to take a long time to tun due to the volume of data moving through the network.

EXP6 Implements an inner join between two tables; Dim_Student and Fact_Table. The query uses four columns, with two from each of the two tables. The query has two WHERE clauses and large datasets.

EXP7 Implements an inner join between three tables; Dim_Programme, Fact_Table, and Dim_Intake. The query uses four columns from the dimension tables and one from Fact_Table. The query has joins between three tables and two conditions in the WHERE clause. Fact_Table is used to link the dimension tables and five columns with 100 million records are involved therefore, a large volume of data will need to travel from Auckland to Amsterdam.

EXP8 Implements a full outer join between two tables Dim_Student and Fact_Table. The query uses a column from Dim_Student and one from Fact_Table with a full join and ORDER BY clause.

Table 2: Summary of Experiments

EXP	TJ	No.T	No.CS	No.CP	QC
1	Right Outer Join	2	3	2	Low
2	Inner Join	4	9	6	High
3	Inner Join	2	4	3	Medium
4	Inner Join	2	2	3	Low
5	Left Join	2	3	3	Low
6	Inner Join	2	4	5	High
7	Inner Join	3	5	6	High
8	Full Outer Join	2	2	4	High
9	Inner Join	2	2	1	Medium

KEY:
TJ = Type of Join
No.T = Number of Tables involved in the query
No.CS = Number of Columns involved in Selection
No.CP = Number of Columns involved in Projection
QC = Query Complexity

`EXP9` Implements an inner join on Fact_Table and Dim_Paper to perform an `UPDATE` query. The records to be updated are filtered with the `WHERE` clause. In practice, this query caused issues with SQL Server and so a variation is required in order for the experiment to complete. The second approach for SQL Server uses the value of Paper_Key from Dim_Paper and sends the value of the retrieved data from Fact_Table to the remote VM, where a separate update procedure performs the update.

Data Collection

Data collection is conducted in two different ways. These are outlined in the following sections.

Data from SQL Server

1. SQL Server profiler is set up on the Amsterdam (local) and Auckland (remote) VMs to capture duration, CPU time, number of logical reads, and number of physical writes. The profiler does not provide the number of physical reads nor does it compute the average I/O latency.

2. Average I/O latencies are calculated.

3. To capture physical reads in both VMs and the execution plan in Auckland, following query is used:

```
SELECT EXECUTION_COUNT, TOTAL_PHYSICAL
   _READS, QP.QUERY_PLAN
FROM SYS.DM_EXEC_QUERY_STATS QS
CROSS APPLY SYS.DM_EXEC_QUERY_PLAN(QS.
   PLAN_HANDLE) AS QP
```

4. Wait events are captured.

5. The execution plan in Amsterdam is obtained using SQL Server's "show execution plan" feature.

6. Network traffic is captured using SQL Server's "show client statistics" feature.

7. SQL Server uses TEMPDB in `EXP8` and `EXP9`. The following steps are taken to capture the number of I/O operations:

   (a) Calculate average I/O latencies during the runtime of the experiments. Including all databases the instance stores, such as TEMPDB and `MASTER`.

   (b) To calculate the number of physical reads, N, we must first determine the number of bytes read in TEMPDB. Equ. 1 is used to convert bytes into kilobytes (KB)

$$x = \frac{y}{1024} \tag{1}$$

   where,

   y = Number of physical reads
   x = Number of bytes read

   The result is divided by 8KB, the default page size in SQL Server.

$$N = \frac{x}{8} \tag{2}$$

Data collection in Oracle

1. The Snapshots section of the Automatic Workload Repository (AWR) feature is used to capture performance statistics. Of the large volume of performance data collected, the following are used:

   Top 5 Timed Foreground Events Provides information about wait classes, such as network and user I/O, and the proportion (%) of each wait from the runtime.

   Foreground Wait Class Used to get the average I/O latency per physical read. If, however, there are physical writes and I/O operations on TEMPDB, then *Foreground Wait Events* is used.

   SQL Statistics Provides runtime and CPU time.

   Segment Statistics Provides the number of logical and physical reads and writes.

2. Oracle's command, `SET AUTOTRACE ON`, is used to get the execution plan from the local instance. The command also provides performance statistics related to the number of I/O operations that happen in TEMPDB files and statistics similar to AWR.

3. The execution plan in Auckland is obtained by querying Oracle's view `V$SQL_PLAN`.

4. Oracle's error log is used to collect data from when an experiment fails.

Data Analysis

Analysis of the collected data is carried out using comparisons and statistical methods. The experiments use more less identical system configurations, making comparisons feasible, and the PuC providers deal with different workloads. Comparisons are carried out as follows:

1. Compare and explain Amsterdam and Auckland execution plans.

2. Compare runtimes between systems.

3. Compare CPU time and explain its relevance to the chosen execution plans.

4. If execution plans create a large number of logical reads, then the values are compared.

5. Number of physical operations are compared and their effects on runtime are quantified by measuring the average I/O latency.

6. Wait events provide information about wait times.

Normal distribution of the data are checked and a skewness test [34] is performed on duration, CPU time, physical reads and network traffic. Table 3 shows the results for the skewness test where the Z values are calculated by the formula:

$$Z = \frac{Sk}{SE_{Sk}} \tag{3}$$

where,

Z is skewness value in standard units.

Sk is the skewness value.

SE_{Sk} is the standard error of the skewness value.

Table 3: Skewness test

Measure	Skewness Z value
Duration	(1.226/.378) = 3.24
CPU time	(2.097/.378) = 5.54
Physical reads	(1.793/.378) = 4.74
Network traffic	(1.882/.564) = 3.33

Cramer and Howitt [13] suggest that in order to statistically analyse a dataset the Z values must be in the range ± 1.96. The standard values in this research for skewness (Table 3) are in the range 3–6. Thus data are positively skewed and, to remove skewness so that pragmatic tests such as regression analysis can be carried out, logarithm of data is applied. Since the data sets resulting from this research are not large, extensive statistical analysis such as factor analysis and full regression testing cannot be applied. Correlation testing is applied if two variables affect each other, such as, to study the effect of network traffic on runtime correlation testing. In cases such as CPU time and run time, correlation testing is not suitable because CPU time is part of the runtime so there is always a causal relationship.

To measure the effect that the Internet has on RDBMS performance, we study the effect of execution plans in a CC environment. Data sets are generated for two different databases (SQL server and Oracle) and hence variables, such as CPU time, are independent. To check if there is any difference between sample means, independent sample testing is used and visualised using a scatter plot. The study uses simple regression testing because large data sets are not generated and the model relies on only one independent variable. Confidence intervals are also used to be sure that the model covers 95% of data.

FINDINGS AND ANALYSIS

The study does not compare the two database systems used in the experiments but investigates relational database performance in CDD. The reason for multiple RDBMS' is to reduce the effect of proprietary differences. Whether one database system performs better than the other is not relevant to the purpose of this research, and both systems differ considerably in the results.

Investigating relational database performance in the cloud involves complicating factors including but not limited to the virtualised environment, the requirements necessary for RDBMS to execute queries, and the round-trip between the nodes across the network. Interactions also occur between nodes on the Internet that are unknown and therefore are uncontrollable.

The following sections provide a brief discussion about the results of `EXP1`–`EXP7` and a detailed discussion of the results of `EXP8` and `EXP9`. The Execution Plans are listed in Appendix .

In `EXP8`, Oracle did not complete the experiment and another approach had to be considered in order to complete the experiment. In `EXP9`, SQL Server was not able to complete the experiment and alternative approach was required. The, results of `EXP8` and `EXP9` are explained by considering the execution plans to investigate how each system approaches the query.

Table 4 lists the measures captured for each of the experiments. The execution plans for each experiment have been mapped and comparisons are made to highlight performance issues. The performance statistics obtained are outlined and compared. Finally, wait events are detailed and compared.

Table 4: Performance Measures

EXP	RT	CT	DO	A I/O L	LR	NT	WE
1	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓		✓	✓
3	✓	✓	✓	✓		✓	✓
4	✓	✓	✓	✓	✓	✓	✓
5	✓	✓	✓	✓		✓	✓
6	✓	✓	✓	✓		✓	✓
7	✓	✓	✓	✓		✓	✓
8(SS)	✓	✓	✓	✓	✓	✓	✓
8(O)	✓	✓	✓	✓		✓	✓
9(SS)			✓	✓		✓	✓
9(O)	✓	✓	✓	✓	✓	✓	

Key:

RT = Runtime
CT = CPU Time
DO = Disk operation
A I/O L = Average I/O latency
LR = Logical Reads
NT = Network Traffic
WE = Wait Event
O = Oracle
SS = SQL Server

Experiment Results

As a total of runtime in `EXP1` (Execution Plan 1), Oracle took more time to complete than SQL Server (Table 5). Of the runtime, a considerable portion of time, for example the cumulative CPU time, is consumed by the VMs in Auckland. Oracle also performed more logical reads than SQL Server. It appears that Oracle's `NESTED LOOPS` performs in a suboptimal manner in this configuration. However, the Auckland instance of SQL Server used the `SORT` operator eliminated a similar situation and led to a gain in performance. Negatively, a large number of disk operations in the Auckland instance of SQL Server may impact performance, where SQL Server needs 10ms per read versus Oracle's 6ms. Compare this with the Amsterdam VM disk latency where the former consumes 12ms per read and latter 10ms per read. These variations may be attributable to the slight difference in server configuration at each location.

Table 5: Comparison between RDBMS for EXP1 to EXP7

EXP	N	S	PM				
			RT(sec)	CT	DO	A I/O L(ms)	LR
1	L	SS	89	0.14	171	12	4590
		O	115	4	890	10	2191245
	R	SS	79	21	2439171	10	2439292
		O	97	18	2019545	6	2019840
2	L	SS	359	5	2145	27	
		O	258	5	1453	11	
	R	SS	348	25	2325816	11	
		O	210	25	2019474	8	
3	L	SS	111	0.156	7523	45	
		O	17420	268	3786	33	
	R	SS	100	106	2325789	10	
		O	10685	28	2019573	14	
4	L	SS	21706	218	44316	56	43845
		O	39319	753	988	14	200200451
	R	SS	21695	282	2523460	15	2474516
		O	22352	150	2019541	15	2024368
5	L	SS	14993	142	3	25	
		O	20268	99	5	23	
	R	SS	14983	209	2474178	18	
		O	13406	52	2014059	15	
6	L	SS	22353	226	44316	54	
		O	37166	210	38287	27	
	R	SS	20208	235	2523554	59	
		O	22019	125	2019573	38	
7	L	SS	34890	392	1246	208	
		O	72535	365	961	13	
	R	SS	34884	443	2622482	14	
		O	38775	50	4629901	16	

Key:

N = Node	S = System
PM = Performance Measures	RT = Runtime
CT = CPU time	DO = Disk operation
A I/O L = Average I/O latency	LR = Logical reads
O = Oracle	SS = SQL Server
L = Local	R = Remote

An examination of the I/O averages and the runtimes for both systems show that as the average I/O latency increases the query takes longer to finish. In `EXP2` (Execution Plan 2), SQL Server needed 359 seconds to run the experiment compared to 259 seconds for Oracle (Table 5). The CDD may add additional complexity to the execution plan selection process where for instance, SQL Server's selection of `MERGE JOIN` forces the Auckland instance to use a `SORT` operator to order the data, which adds performance overhead. Although both databases process the same number of tuples, there is considerable difference in CPU consumption. However, while the systems use different join operators, they consume an identical amount of CPU time, indicating that the time needed to join tables increases as the number of joins and volume of data increases. A relationship exists between disk operations (number of physical reads) and the average I/O latency, such that the average I/O latency in the Auckland VMs Fact_Table is far less than the average in the Amsterdam VMs (Dimension tables).

Disk activity may be instrumental in the poor performance of relational databases in a cloud environment. There is a significant variation in the runtime and CPU time

for both systems in EXP3 (Execution Plan 3). SQL Server takes less than two minutes to finish, whereas Oracle takes 4.8 hours. Oracle consumes more CPU time in the Amsterdam VM than at Auckland. The Amsterdam Oracle VM needs 268 seconds to execute EXP3 but the Auckland Oracle VM needs 28 seconds to perform its part. By contrast, SQL Server takes 106 seconds to process EXP3 in the Auckland VM and only 0.156 seconds of CPU time is spent in the Amsterdam SQL Server instance (Table 5). Further, in both database systems, the CPU time appears to be high which highlights the effect of processing a large dataset in a relational CDD. While EXP2 shows a correlation between the average I/O latency and duration, the results of EXP3 show no correlation. The Amsterdam VMs appear to suffer from high I/O, and that reflects the reality of operating in a PuC environment. In EXP3, SQL Server performed more physical reads in the Auckland VM (taking 45ms) than it did in EXP2 (taking 27ms). A similar pattern is observed in Oracle but leads to a different result where the Auckland VM average I/O latency in EXP3 (taking 14ms) is higher than EXP2 (taking 8ms).

Disk operations (Physical reads) creates overhead on runtime and therefore is an important factor. In EXP4 (Execution Plan 4), both systems execute in a nearly identical manner but the runtime measures differ; SQL Server finishes sooner than Oracle, which is 21706 seconds or 6 hours for SQL Server and 39319 seconds or 7.2 hours for Oracle (Table 5). Moreover, Oracle consumes 903 seconds CPU time and SQL Server consumes 500 seconds. Oracle uses the NESTED LOOPS join operator where one row from Dim_Student is selected, and then the operator looks for the matching row among 100 million tuples. Accompanied by a high CPU time, Oracle in Amsterdam produces more logical reads than SQL Server. Further examination of SQL Server CPU time and logical reads suggests that MERGE JOIN is faster than NESTED LOOPS, creating fewer logical reads. SQL Server in Amsterdam's CPU consumption is still relatively high (218 seconds) compared to the Auckland CPU time (282 seconds) where a SORT operator is used. Moreover, when the CPU time of both Auckland VMs are compared, it can be seen that SQL Server consumes more CPU time than Oracle. This is because Oracle does not use the SORT operator.

Local network topology appears to be a factor in performance, for example the Amsterdam VMs suffer higher average I/O latency than the Auckland VMs. Execution Plan 5 shows differences in how EXP5 is executed. For instance, although SQL Server consumes a higher CPU time in Amsterdam, it still takes less time to complete than Oracle. Table 5 illustrates how SQL Server needs 14993 seconds (4 hours and 16 minutes) whereas Oracle takes 20268 seconds (6 hours and 3 minutes). SQL Server's choice of MERGE JOIN requires a SORT operator (compare with EXP4 where this consumes more CPU time than when a HASH JOIN operator is used). In EXP3 and EXP4, although the Auckland instances conduct high I/O traffic, their average I/O latency was not as high as in Amsterdam.

In EXP6, the collected performance data suggests significant variation between systems (Execution Plan 6). SQL Server runs faster than Oracle with more than 4 hours of difference between them. CPU time in Amsterdam indicates that the SQL Server choice of MERGE JOIN consumes more CPU time than Oracle's choice of the HASH MATCH JOIN operator. Moreover, it is clear that the use of the SORT operator leads to a 25 second difference between Auckland VMs. Oracle's consumption of CPU time is 125 seconds, whereas SQL Server consumes 235 seconds of CPU time (Table 5). When joining tables, it matters how many tuples are to be joined and the choice of join operator. In this experiment and compared with EXP5, the SQL Server installation in Amsterdam increases CPU consumption by 6 seconds. This increase appears to be attributable to operations over the cloud network. That is, SQL Server's choice of the MERGE JOIN operator appears to have added processing overhead that has an effect when data needs to be sorted at both ends (a choice that occurs in five experiments) compared with Oracle that uses the MERGE SORT JOIN operator only once, where there is an ORDER BY clause. EXP6 presents the effect of shared computing resources, causing the RDBMS to suffer. In the Amsterdam instance of SQL Server, out of EXP1–EXP5, the highest average I/O read latency was recorded in EXP5 (25ms) but this increased to 54ms per read in EXP6. Additionally, the number of physical reads increased to 44316 reads, compared with 3 reads in EXP5. The duration for EXP6 is longer time than EXP5, and EXP6 displays higher average I/O latency. Overall, SQL Server shows higher average I/O latency than Oracle, but Oracle runs for a longer period of time.

Oracle needed 20 hours to run EXP7 versus 10 hours for SQL Server, even though SQL Server consumed more CPU time than Oracle in both VMs. Further, the CPU time required by SQL Server provides evidence that MERGE JOIN is not an effective join option. By contrast, Oracle employs HASH JOIN twice to join the data but takes 369 seconds of CPU time while SQL Server takes 392 seconds of CPU time (Table 5). This does not mean that the optimiser performs below par but that its choice of MERGE JOIN is less suitable because, on the one hand there are 100 tuples coming from the Auckland instance and on the other hand, this operator needs sorted data in order to function. So, the optimiser uses SORT in the Auckland VM. SORT creates overhead, see the CPU time of the Auckland Oracle VM, which is less than the Auckland SQL Server CPU

time. In addition, the average I/O latencies in EXP7 are the highest where the Amsterdam SQL Server presented 208ms per read. This contributes to a long running query, although 0ms is reported as the average I/O latency per write. The Auckland SQL Server VM experiences less average I/O latency than in EXP6, even though it does more reads, whereas evidence suggests that the Amsterdam SQL Server is adversely affected by the public network. Such variations indicate inconsistencies in RDBMS performance measures, for example in all the experiments so far, Oracle in Amsterdam experienced less average I/O latency than SQL Server, suggesting that variations in performance can occur within the same PuC service provider. Also, the use of the HASH JOIN operator requires less time than the MERGE JOIN operator, suggesting that HASH JOIN is more time efficient than MERGE JOIN.

The next experiment provides more evidence that the Cloud environment is a contributing factor in performance reduction, especially when there is an extensive disk activity for the ORDER BY operator is used. EXP8 joins two large datasets and performs an ORDER BY operation. After 12.32 hours running, Oracle did not complete and the execution plans are lost (Table 6). Note that EXP7 ran for longer without timing out. Additionally, the CPU consumes 410 seconds in Amsterdam and 59 seconds in Auckland (due to the absence of a SORT operator). Also, SQL Server takes the longest time to run and consumes the highest CPU time. Comparing Amsterdam's SQL Server CPU time in EXP7 with EXP8, there is an increase from 392 seconds to 424 seconds even though ORDER BY is performed on disk. Part of the reason may be more logical reads in EXP8 (500000 reads) even though the instances in Auckland performed more logical reads than Amsterdam. Table 6 shows that the Auckland instances consume less CPU time.

Table 6: Comparison between RDBMS for EXP8 and EXP9

EXP	N	S	PM					
			RT(sec)	CT	PR	PW	A I/O L(ms)	LR
8	L	SS	59118	424	54329	111	72	429115
		O	44364	410	0	0	9	0
		OSA	4548	51	901		16	
	R	SS	58015	259	2808983	144	15	2688681
		O	41373	59	4644618	0	40	4049424
		OSA	4103	12	2019546		24	
9	L	SS	86554	166	0	1083	35	831865
		O	1357	0.009	2	0	8	9
		SSSA	500	0.03	2	0	22	
	R	SS	0	882	2325815	360075	8	373065421
		O	1352	239	8582166	2306460	14	11248000
		SSSA	498	27	2325748	920824	33	

Key:

N = Node
S = System
PM = Performance Measures
RT = Runtime
CT = CPU time
PR = Physical Reads
PW = Physical Writes
A I/O L = Average I/O latency
LR = Logical reads
L = Local, Amsterdam
R = Remote, Auckland
O = Oracle
SS = SQL Server
OSA = Oracle Second Approach
SSSA = SQL Server Second Approach

Oracle not finishing EXP8 produces uncertainty and so a different approach is taken. To minimise the effect of the network, the data volume is reduced from 18 GB with 100 million tuples to 10 million. The Oracle Second Approach (OSA) then finishes in 1 hour and 15 minutes (Table 6). Based on this time, the same query with 100 million tuples would have taken 140 hours to run, or more than five days. The execution plan shows that to sort data, there is extensive disk activity which influences runtime. The Auckland instance shows a high average I/O latency per read of 24ms, whereas Amsterdam needs 16ms per I/O read (reading only the Index Table). Each disk write takes an average of 39ms and for a read and takes an average of 9ms to finish.

Table 7: Runtime wait event percentage for SQLServer

EXP	1	2	3	4	5	6	7
EXP1	13.65	3.18	2.44	29.33	8.33	2.58	
EXP2	34.62	1.98		9.09	30.59		7.08
EXP3	9		1.92	11.51	15.35		7.08
EXP4	45.99				65.21		16.24
EXP5	25.39				50.12		12.51
EXP6	43.93				63.32		15.77
EXP7	47.09				65.38		16.29
EXP8	47.56				65.88		32.12
EXP9	49.56						32.12
EXP9 (SSSA)	21.1			25.15	15.12		32.12

Key:

1 = Local OLEDB : Wait which occurs when SQL server calls the SQL server native client OLEDB provider

2 = Local LCK_M_S: Time taken when the local instance is waiting to acquire a shared lock. This lock prevents other transactions from modifying data, but allows multiple concurrent read (SELECT) operations. See https://documentation.red-gate.com/display/SM4/LCK_M_S for detailed information.

3 = Local PAGEIOLATCH_SH: Time taken when the local node waits for buffer to be accessible

4 = Remote PAGEIOLATCH_SH: Time taken when the remote node waits for buffer to be accessible

5 = Remote CXPACKET: Time taken when the remote instance waits for local instance to complete an operation. See http://serverfault.com/questions/290526/high-cxpacket-wait-type-in-sql-server-even-with-maxdop-1

6 = Remote PAGEIOLATCH_EX: Time taken when the remote instance waits for data to be written into the memory from the disk

7 = ASYNC_NETWORK_IO: Network related wait event

SSSA = SQL Server Second Approach

Table 8: Runtime wait event percentage for Oracle

EXP	1	2	3	4	5	6	7	8	9	10
EXP1	20.93		0.11	4.3	29.95			4.8		73.75
EXP2	68.82	20.11			2.81		68.28			24.65
EXP3	91.55	4.65			0.61		98.9	0.06		0.01
EXP4	94.06	3.2			0.53		98.22	0.61		0.48
EXP5	93.55	5.03			0.41	0.08	98.93	0.66		0.04
EXP6	95.71	3.42					99.04			
EXP7	94.55	2.6			0.3		92.92	0.78	0.66	0.09
EXP8	92.94	1.92					93.41	0.5	0.79	0.2
EXP8 (OSA)	89.61	3.42					98.15	0.06		1.29
EXP9		46.23			4.36	4.16		17.02	25.95	1.91

Key:

1 = Local SQL* Net more data from dblink

2 = Local SQL* Net message from dblink

3 = Local SQL* Net more data to client

4 = Local disk file operations I/O

5 = Local db file sequential read

6 = Local db file scattered read

7 = Remote SQL* Net more data to client

8 = Remote db file sequential read

9 = Remote db file scattered read

10 = Remote direct path read

OSA = Oracle Second Approach

Moreover, in an indication that the effect of the public network surpasses the I/O latency effect, the Amsterdam SQL Server instance in EXP8 spends nearly 48% of its time waiting for 3633Mb of data to arrive (Table 7). In the Auckland instance, a long wait period accrues due to the parallelism operation but this seems unavoidable; the parallel manager waits for CXPACKET, while waiting for processors to finish their assigned work. In the meantime in both VM's, Oracle spends more than 90% of time waiting for data to move via the network (Table 8).

When Oracle crashed the transaction logs of both instances reported the error ORA-12170: TNS: CONNECT TIMEOUT OCCURRED, "The server shut down because connection establishment or communication with a client failed to complete within the allotted time interval. This may be a result of network or system delays; or this may indicate that a malicious client is trying to cause a Denial of Service attack on the server" [40]. An analysis indicates that Oracle attempts to write to a temp file that plays a role in extending the delay, DIRECT WRITE TEMP. This creates a wait of nearly 2% of the runtime. However, reading this data again takes less than 1% of local instance runtime. The I/O read in the remote instance creates a total wait of 1.35% of the runtime, while the remaining 98.15% of its runtime (Table 8) is recorded as a wait for the data to reach the local instance.

SQL Server demonstrates poor performance in EXP9 compared with Oracle. SQL Server in Amsterdam stopped running at 24 hours, therefore there is no execution plan and data are lost. Also, Oracle executed in Auckland instead of Amsterdam but this plan creates fewer complexities than SQL Server's approach. Oracle takes 22.61 minutes to run EXP9 (Table 6) vs SQL Server's 24 hours before the experiment was cancelled. Oracle in Amsterdam consumed only 0.009 seconds of CPU time whereas the Auckland VM shows the highest consumption of CPU time. SQL Server consumes significant CPU time and the reason is evident in the execution plan (Execution Plan 9). SQL Server in Amsterdam reports a large number of logical reads and it consumed 166 seconds of CPU time. While it is normal for a physical write to occur when an update occurs, it is difficult to see why SQL Server has to perform physical writes in Amsterdam. The subquery returns one tuple from Dim_Paper which should not result in a physical write. We think that physical writes are because SQL Server logs updates in the remote instance. Further, Oracle's physical writes in Auckland are a result of the update. Before cancellation, SQL Server performed 630075 physical writes in Auckland. The average I/O latency reflects only an average read latency where for SQL Server, the average write latency in Auckland is 26ms and 21ms for Oracle.

Wait events in Table 7 show that network overhead has a noticeable effect even when the volume transferred is small. This is illustrated in EXP9 where SQL Server shows high waiting periods. For example, SQL Server fetches a tuple from Fact_Table and sends it to Amsterdam to check whether or not the tuple adheres to the conditions in the subquery with a wait time of 32% of the runtime as a consequence. Tuples arrive sequentially at Amsterdam and then are processed, thus it is SQL Server's ANSNC_NETWORK_IO and the network that are implicated in slower processing. Oracle demonstrates a similar pattern where the network wait indicates that Amsterdam waits for acknowledgement from the Auckland VM. Adding together I/O operation wait periods in Auckland shows that the VM spends a large part of the runtime (45%) in I/O operations, thus creating a bottleneck.

While it is clear from EXP9 that performance issues in SQL Server exist, it is difficult to determine where the issues exist. To further investigate SQL Server's approach to update queries, a different approach to EXP9 was undertaken. To ensure that the two tables are indirectly joined, the second approach involved the removal of the subquery and instead matching a selected value from Paper_Key from Dim_Paper and passing that to an update procedure located in the Auckland instance. In SQL Server's second approach, the Auckland instance then pauses for different wait events (Table 7), for example 25% of the runtime is spent on LATCH_EX, which is SQL Server waiting for an exclusive latch. Note that the "wait does not include buffer latches or transaction mark latches" [36]. This indicates that this wait event is not related to I/O operation or data. Further, the Auckland instance pauses for disk-related waits; ASYNC_IO_COMPLETIONT (14.56%) and PAGEIOLATCH_SH (25.15%). ASYNC_IO_COMPLETIONT occurs when a task is waiting for I/Os to finish [36] and PAGEIOLATCH_SH waits for data to be written to memory [36]. Therefore with this approach, EXP9 is I/O bound.

Influence of PuC Network

The aim of the study is to identify where it is that relational database performance in a CC environment is negatively affected. To learn where performance breakpoints may be, two RDBMS have run a series of experiments so that database, server, and network data can be collected. The data analysis indicates where there are weaknesses in performance and from that, the log data are interrogated to find what the RDBMS' did at the time. The analyses demonstrate that relational databases fall victim to two factors: the methods used by RDBMS when executing queries over the Internet and creating network overheads, and the PuC envi-

ronment provides conditions for poor performance.

While the collected data appears at first inconsistent (Table 9), overall they indicate the effect of the PuC. That is, although RDBMS are I/O bound there appears to be a weak relationship between number of physical reads and high average I/O latency in a CC environment where the variable patterns suggest differing levels of WAN overhead. For example, although SQL Server transfers a higher volume of data in `EXP6` and `EXP7`, it finishes faster than Oracle (Table 10), whereas in `EXP2` with a larger dataset, SQL Server takes longer than Oracle, and in `EXP7` SQL Server transfers more data than in `EXP8` but finishes faster. It appears that when more than 22MB is transferred, performance drops (Table 9 and Table 10).

Table 9: Average I/O latency vs Number of physical reads

EXP	S	LPR	LA I/O L(ms)	RPR	RA I/O L(ms)
EXP1	SS	171	12	2439171	10
	O	890	10	2019840	6
EXP2	SS	2145	27	2325816	12
	O	1453	11	2019474	8
EXP3	SS	7523	45	2325789	10
	O	3786	33	2019573	14
EXP4	SS	44316	56	2523460	15
	O	988	14	2019541	15
EXP5	SS	3	25	2474178	18
	O	5	23	2014059	15
EXP6	SS	44316	54	2523554	59
	O	38287	27	2019573	38
EXP7	SS	1246	208	2622482	14
	O	916	13	4629901	16
EXP8	SS	54329	72	2808983	15
	O	N/A	N/A	4644618	40
EXP8 (OSA)	O	991	16	2020096	24
EXP9	SS	N/A	35	2325815	8
	O	2	8	8582166	14
EXP9 (SSSA)	SS	2	22	2325748	33

Key:

S = System
LA I/O L = Local average I/O latency
RA I/O L = Remote average I/O latency
O = Oracle
OSA = Oracle Second Approach
LPR = Local physical read
RPR = Remote physical read
SS = SQL Server
SSSA = SQL Server Second Approach

Table 10: Network traffic vs Runtime

EXP	System	Network Traffic (MB)	Runtime (sec)
EXP1	SS	0.162	89
	O	10	115
EXP2	SS	125	359
	O	21	258
EXP3	SS	0.244	111
	O	1011	17420
EXP4	SS	1242	21706
	O	1584	39319
EXP5	SS	823	14993
	O	1019	20268
EXP6	SS	2864	22353
	O	2572	37166
EXP7	SS	8613	34890
	O	8198	72535
EXP8	SS	3633	59118
	O	N/A	N/A
EXP8 (OSA)	O	187	4548

Key:
SS = SQL Server O = Oracle OSA = Oracle Second Approach

Table 11: Correlation between Duration and Network traffic

		Network traffic Log
Duration	Pearson Correlation	.928**
	Sig.(2-tailed)	.000
	N	16

** = Correlation is significant at the level 0.01 level (2-tailed)

Table 12: Simple regression test

Coefficients [a]

Model		Ustd Coeff.		Std Coeff.	t	Sig.	Conf. Interval	
		B	Std. Error	Beta			LB	UB
1	C	-4.887	1.462		-3.344	0.005	-8.002	-1.753
	NTL	0.692	0.074	0.928	9.325	0.000	0.533	0.851

[a] = Dependent Variable: durationLog

Key:

Ustd Coeff. = Unstandardised Coefficients
Std Coeff. = Standardised Coefficients
Conf. Interval = 95% Confidence Interval for B
LB = Lower Bound
UB = Upper Bound
C = Constant
NTL = networktrafficLog

The study is conducted on a PuC using workloads that would make infrastructure effects obvious. For example, in EXP3, for SQL Server to complete, a wait of 12.43% of runtime for I/O operations is experienced (Table 9). Additionally, in EXP2 and EXP9, Oracle waits of 27.46% and 45% of runtime for I/O reads to finish respectively . In EXP3, Oracle requires the data to be brought over the PuC before it processes the query, even though the count operation could be performed on the table in Auckland. Due to the lack of network capacity, performance issues are created in EXP3 and Oracle takes significantly longer to run than SQL Server (Table 10).

To summarise briefly, both systems wait for the PuC to deliver data but for Oracle, the network wait appears overwhelming. Even when larger datasets are involved, SQL Server queries take less time and this is reflected in wait related events that never go beyond 50% of the runtime, whereas Oracle never drops below 60%. To learn more, correlation and regression tests are carried out. A strong correlation between duration and network traffic is apparent (Table 11). A scatter plot (Figure 2) shows a trend between duration and network traffic which indicates that duration increases as network traffic increases. Moreover, the plot shows the relationship is not directly linear and that there exists a degree of randomness with a cluster at the top of the plot. The cluster indicates a strong relationship between duration and network traffic at higher volumes of data transfer.

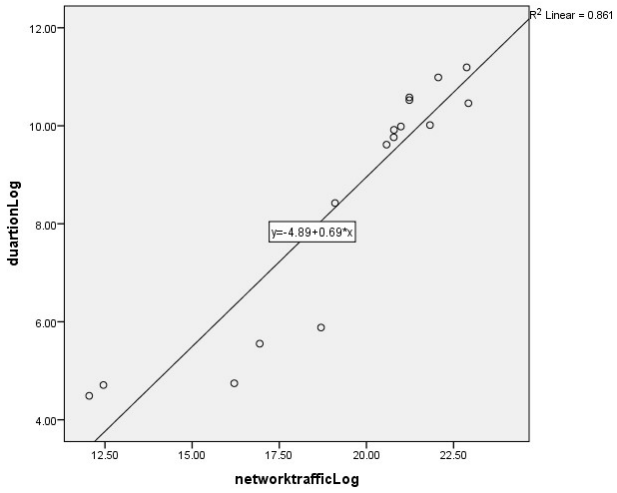

Figure 2: Duration vs Network traffic

To illustrate how an execution plan selection may impact performance, in EXP8 Oracle crashes at the 44364th second from execution start (˜12.32 hours) because, to execute each query, Oracle downloads the required tables and more than 90% of VM time is spent waiting for data to traverse the PuC (Table 8). Additionally, SQL Server's execution of EXP9 causes a long-running query and the highest CPU time of all nine experiments. The steps that SQL Server undertakes, which uses KEYSET CURSOR, suggests that including a sub-query in update statements in SQL Server causes significant performance issue in PuC and when the sub-query is removed in EXP9 SSAA, performance improves.

An initial look at the regression test (Table 12) appears to strengthen the argument for a relationship between network traffic and duration. With 95% confidence, every unit of increase in network traffic increases duration between 0.533 and 0.851. However, note that the curve in Figure 3 shows a more pronounced curvature than is evident in Figure 2. This indicates a degree of non-linearity in the data. The relationship may be explained by the effect of I/O latency in the PuC environment.

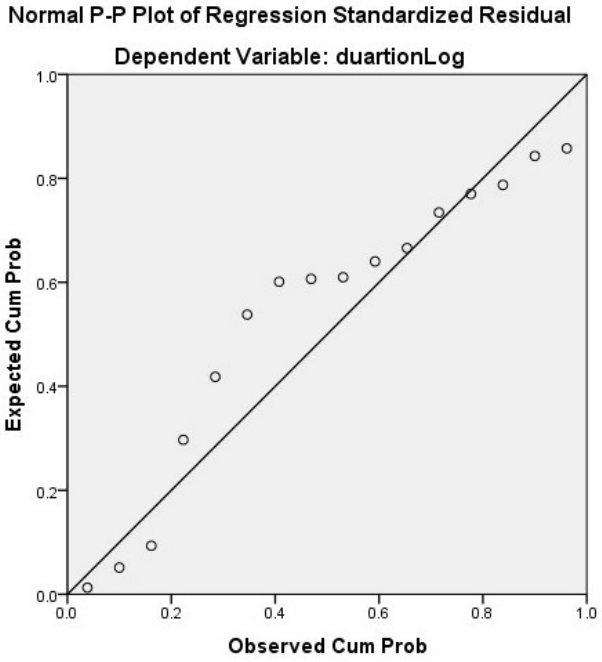

Figure 3: Normality of simple regression test

CONCLUSION

The study presents limiting factors, for example the research originally aimed to use a larger dataset but of the systems that actually completed, most of the experiments took a long time to run and sometimes they did not finish due to technical failures. Consequently, experiments could not be easily compared. Also, due to time constraints there is no control experiment on n-tier architecture. This would have provided a greater degree of surety about the

effect of the PuC. Additionally, the experiments are conducted using a star schema not used in a transaction processing system. While joins are included in the queries, we did not feel this would unduly affect the query results and the significant amount of data provided was preferred. Further, the RDBMS's in the CDD are used in a non-optimised environment compared with Amazon's EC2 and Microsoft's SQL Server on Azure where resources are heavily optimised. This was a deliberate step in order to demonstrate the effect of the Cloud on RDBMS.

Since the study includes no specific performance enhancements, the opportunity exists to redo the experiments both on and off the Cloud. Architectural issues with the relational model on the Cloud may still need to be investigated. To that end, additional experiments may be done on non-relational database systems such as NOSQL or graph database environments. We have investigated opportunities for replicating the experiments in a graph database (Neo4j) as a transaction processing system, where a join in a relational database is somewhat equivalent to an edge traversal between nodes in a graph database. Elements (vertices and edges) maintain direct reference to their adjacent elements, which makes traversing a graph structure within a graph database fast and efficient [41]. The issue at this time is the lack of a consistent or cross application querying language for graph databases. Therefore we are also investigating the development of a structured query language for graphs.

It is usual for RDBMS to be deployed on specifically designed infrastructure with sufficient network bandwidth and consequently, RDBMS generally perform better on n-tier architecture [17, 15, 44, 5, 29, 3]. However, since PuC includes an unknown and heterogeneous mix of infrastructure elements, distributed RDBMS suffer from performance issues and determining specific problem areas is difficult. Thus the study identifies break points when operating an RDBMS on a PuC. To identify specific problem areas, RDBMS query execution plans are investigated, specifically noting the effect between network traffic and query runtime. The study finds that individual product's selection of query execution plan greatly affects performance in different ways. For example, Oracle shows a tendency to transfer large volumes of data before undertaking a query, whereas SQL Server tends to wait for a remote event to complete before continuing a query. Also, sub-queries create performance issues for both of the systems. Further, when no joins are included in the query, then while these issues are not observed, the PuC has a negative impact on performance. This factor was emphasised with the large dataset used in the experiments.

REFERENCES

[1] Anderson, T., Breitbart, Y., Korth, H., and Wool, A. Replication, consistency, and practicality: are these mutually exclusive? *ACM SIGMOD Record*, Volume 27, Number 2, 1998, pp 484–495.

[2] Baccelli, F. and Coffman, E. G. A data base replication analysis using an m/m/m queue with service interruptions. *ACM SIGMETRICS Performance Evaluation Review*, Volume 11, Number 4, 1982, pp 102–107.

[3] Benson, T., Akella, A., and Maltz, D. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, New Delhi, India. ACM. 2010, pp 267–280.

[4] Born, E. Analytical performance modelling of lock management in distributed systems. *Distributed Systems Engineering*, Volume 3, Number 1, 1996, p 68.

[5] Bose, S., Mishra, P., Sethuraman, P., and Taheri, R. *Benchmarking database performance in a virtual environment*, Springer, Berlin Heidelberg, 2009, pp 167–182.

[6] Bouras, C. and Spirakis, P. Performance modeling of distributed timestamp ordering: Perfect and imperfect clocks. *Performance evaluation*, Volume 25, Number 2, 1996, pp 105–130.

[7] Buyya, R., Yeo, C., and Venugopal, S. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *10th IEEE International Conference on High Performance Computing and Communications*, Dalian, China. IEEE., 2008, pp 5–13

[8] Chaudhuri, S. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, Seattle, WA, USA. ACM, 1998, pp 34–43.

[9] Chaudhuri, S. What next?: a half-dozen data management research goals for big data and the cloud. In *Proceedings of the 31st symposium on Principles of Database Systems*, Scottsdale, AZ, USA. ACM, 2012, pp 1–4.

[10] Chaudhuri, S., Dayal, U., and Narasayya, V. An overview of business intelligence technology. *Communications of the ACM*, Volume 54, Number 8, 2011, pp 88–98.

[11] Codd, E. F. A relational model of data for large shared data banks. *Communications of the ACM*, Volume 13, Number 6, 1970, pp 377–387.

[12] Connolly, T. and Begg, C. *Database systems: a*

practical approach to design, implementation, and management. Pearson Education, Harlow, England, 2005.

[13] Cramer, D. and Howitt, D. *The Sage dictionary of statistics: a practical resource for students in the social sciences*. Sage, Thousand Oaks, 2004.

[14] Durham, E., Rosen, A., and Harrison, R. Optimization of relational database usage involving big data a model architecture for big data applications. In *2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, Orlando, FL, USA. IEEE, 2014, pp 454–462

[15] Eriksson, P. A new approach for enterprise application architecture for financial information systems: An investigation of the architectural implications of adopting serialization and RPC frameworks, noSQL/hybrid data stores and heterogeneous computing in financial information systems. Masters thesis, School of Computer Science and Communication, KTH Royal Institute of Technology, 2015.

[16] Feuerlicht, G. and Pokorný, J. *Can Relational DBMS Scale Up to the Cloud?*, Springer, New York, 2013, pp 317–328.

[17] Frerking, G., Blanton, P., Osburn, L., Topham, J., DelRossi, R., and Reisdorph, K. U.S. patent application 10/935,514, 2004.

[18] Geelan, J. Twenty-one experts define cloud computing. *Cloud Computing Journal*, Volume 1, Number 4, 2009, pp 1–5.

[19] Gray, J., Helland, P., O'Neil, P., and Shasha, D. The dangers of replication and a solution. *ACM SIGMOD Record*, Volume 25, Number 2, 1996, pp 173–182.

[20] Gunarathne, T., Wu, T., Qiu, J., and Fox, G. Mapreduce in the clouds for science. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Indianapolis, IN, USA. IEEE, 2010, pp 565–572.

[21] Hashem, I., Yaqoob, I., Anuar, N., Mokhtar, S., Gani, A., and Khan, S. The rise of "big data" on cloud computing: review and open research issues. *Information Systems*, Volume 47, Jan, 2015, pp 98–115.

[22] Iosup, A., Ostermann, S., Yigitbasi, M., Prodan, R., Fahringer, T., and Epema, D. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, Volume 22, Number 6, 2011, pp 931–945.

[23] Ivanov, I. *The Impact of Emerging Computing Models on Organizational Socio technical System*, Springer, Berlin Heidelberg, 2013, pp 3–19.

[24] Ivanov, T., Petrov, I., and Buchmann, A. A survey on database performance in virtualized cloud environments. *International Journal of Data Warehousing and Mining*, Volume 8, Number 3, 2012, pp 1–26.

[25] Jackson, K., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., and Wright, N. Performance analysis of high performance computing applications on the amazon web services cloud. In *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Indianapolis, IN, USA. IEEE, 2010, pp 159–168

[26] Khajeh-Hosseini, A., Greenwood, D., and Sommerville, I. Cloud migration: A case study of migrating an enterprise it system to IaaS. In *IEEE 3rd International Conference on Cloud Computing*, Washington, DC, USA. IEEE, 2010, pp 450–457.

[27] Khan, M. and Khan, M. Exploring query optimization techniques in relational databases. *International Journal of Database Theory and Application*, Volume 6, Number 3, 2013, pp 11–20.

[28] Kiefer, T., Schlegel, B., and Lehner, W. Multe: a multi-tenancy database benchmark framework. In *Technology Conference on Performance Evaluation and Benchmarking*, New Delhi, India. Springer, 2012, pp 92–107

[29] Kohler, J. and Specht, T. Vertical query-join benchmark in a cloud database environment. In *Second World Conference on Complex Systems*, Agadir, Morocco. IEEE, 2014, pp 581–586.

[30] Li, A., Yang, X., Kandula, S., and Zhang, M. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, Melbourne, Australia. ACM, 2010, pp 1–14.

[31] Litchfield, A. and Althouse, J. A systematic review of cloud computing, big data and databases on the cloud. In *Twentieth Americas Conference on Information Systems*, Savannah, USA. AIS, 2014, pp 1–10

[32] Liu, C. and Yu, C. Performance issues in distributed query processing. *IEEE Transactions on Parallel and Distributed Systems*, Volume 4, Number 8, 1993, pp 889–905.

[33] Lloyd, W., Pallickara, S., David, O., Lyon, J., Arabi, M., and Rojas, K. Performance implications of multi-tier application deployments on infrastructure-as-a-service clouds:towards performance modeling. *Future Generation Computer Systems*, Volume 29, Number 5, 2013, pp 1254–1264.

[34] Martin, W. E. and Bridgmon, K. D. *Quantitative and statistical research methods: From hypothesis to results*, John Wiley & Sons, San Francisco, CA, USA, 2012.

[35] McKendrick, J. Big data, big challenges, big op-

portunities: 2012 IOUG big data strategies survey. Technical report, Unisphere Research, Murray Hill, New Providence, NJ, 2012.

[36] Microsoft Corp. *Buffer management*, https://technet.microsoft.com/en-us/library/aa337525%28v=sql.105%29.aspx, July, 2015.

[37] Minhas, U., Yadav, J., Aboulnaga, A., and Salem, K. Database systems on virtual machines: How much do you lose? In *IEEE 24th International Conference on Data Engineering Workshop, 2008*, Cancun, Mexico. IEEE, 2008, pp 35–41.

[38] Moens, H. and De Turck, F. Shared resource network-aware impact determination algorithms for service workflow deployment with partial cloud offloading. *Journal of Network and Computer Applications*, Volume 49, Number 1, 2015, pp 99–111.

[39] Mullins, C. S. Distributed query optimization. *Technical Support*, July, 1996, pp 1–3.

[40] Oracle Corp. *Database error messages*, https://docs.oracle.com/cd/B19306_01/server.102/b14219/net12150.htm, July, 2015.

[41] Rodriguez, M. A. and Neubauer, P. Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, Volume 36, Number 6, 2010, pp 35–41.

[42] Shao, J., Liu, X., Li, Y., and Liu, J. Database performance optimization for SQL Server based on hierarchical queuing network model. *International Journal of Database Theory and Application*, Volume 8, Number 1, 2015, pp 187–196.

[43] Tewari, P. Query optimization strategies in distributed databases. *International Journal of Advances in Engineering Sciences*, Volume 3, Number 3, 2013, pp 23–29.

[44] Thakar, A., Szalay, A., Church, K., and Terzis, A. Large science databases, are cloud services ready for them? *Scientific Programming*, Volume 19, Number 2—3, 2011, pp 147–159.

[45] Thanos, C., Bertino, E., and Carlesi, C. The effects of two-phase locking on the performance of a distributed database management system. *Performance Evaluation*, Volume 8, Number 2, 1988, pp 129–157.

[46] Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, Volume 39, Number 1, 2008, pp 50–55.

[47] Weins, K. Cloud Computing trends: 2016 state of the cloud survey, http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2016-state-cloud-survey, 2016.

[48] Zhang, Y., Yu, L., Zhang, X., Wang, S., and Li, H. Optimizing queries with expensive video predicates in cloud environment. *Concurrency and Computation: Practice and Experience*, Volume 24, Number 17, 2012, pp 2102–2119.

AUTHOR BIOGRAPHY

Awadh Althwab completed his Master of Computer and Information Sciences with First Class Honours at AUT. For his research thesis, he was also received the Dean's List award. He is presently a research student at the Australian National University (ANU) where he is studying massive graph analytics.

Dr Alan T Litchfield (corresponding author) is Director of the Service and Cloud Computing Research Lab (SCCRL), at the Auckland University of Technology (AUT). He is a partner in a consulting firm that provides specialised services to corporates, government departments and military, is President of the Association for Information Systems (AIS) Special Interest Group on Philosophy in Information Systems, founding Programme Leader for the Master of Service Oriented Computing (MSOC), member of the AUT Programmes and Academic Review Committee (PARC). Dr Litchfield is member of the Institute of IT Professionals (MIITP), Institute of Electrical and Electronics Engineers (IEEE), Association for Computing Machinery (ACM), International Institute for Information Design (IIID), TeX Users Group (TUG), and Association for Information Systems (AIS). His areas of research interest cover service and cloud computing and the philosophy of science and especially in applied areas of information systems research. Most recently, he has been leading research into decentralised and distributed software architectures and applications of blockchains in medical records management, software licensing, utilities management, systems authentication and authorisation, and on the Internet of Things (IoT).

Chandan Sharma is a research student in the School of Engineering, Computer and Mathematical Sciences , Auckland University of Technology. Having completed the Master of Service Oriented Computing with First Class Honours in 2016, he is now a PhD student researching Distributed Computing, Database systems, Formal Methods and Theory of computation. He contributed to this paper in his spare time.

APPENDIX A: EXPERIMENT SQL QUERIES

This appendix provides the SQL queries used in the experiments.

```
EXP1 SELECT D.STUDENT_DEMOGRAPHICS_KEY, F.TOTAL_EFTS
     FROM DIM_STUDENT_DEMOGRAPHICS D
     RIGHT OUTER JOIN MYLINK.LOCAL.DBO.FACT_TABLE F
     ON D.STUDENT_DEMOGRAPHICS_KEY = F.STUDENT_DEMOGRAPHICS_KEY
     WHERE F.TOTAL_EFTS >0;

EXP2 SELECT F.STUDENT_DEMOGRAPHICS_KEY,P.PAPER_KEY, D.CALENDAR_YEAR,
          E.ENROLMENT_TYPE_KEY, E.ENROLMENT_TYPE_GROUP_DESC,
          P.TEACH_DEPT_CODE
     FROM DIM_PAPER P, DIM_DATE D, DIM_ENROLMENT_TYPE E,
          MYLINK.LOCAL.DBO.FACT_TABLE F
     WHERE P.PAPER_KEY = F.PAPER_KEY
     AND E.ENROLMENT_TYPE_KEY = F.ENROLMENT_TYPE_KEY
     AND D.DATE_KEY = F.DATE_KEY
     AND F.PAPER_KEY =13362
     AND F.ENROLMENT_TYPE_KEY = 33
     AND D.CALENDAR_YEAR BETWEEN 2000 AND 2013;

EXP3 SELECT D.PAPER_KEY,PAPER_FULL_DESC, COUNT(DISTINCT
          F.STUDENT_DEMOGRAPHICS_KEY) AS COUNTOFENROLLEDSTUDENTS
     FROM DIM_PAPER D
     INNER JOIN FACT_TABLE F
     ON D.PAPER_KEY = F.PAPER_KEY
     GROUP BY D.PAPER_KEY,PAPER_FULL_DESC
     HAVING COUNT(DISTINCT F.STUDENT_DEMOGRAPHICS_KEY) >=5
     ORDER BY COUNTOFENROLLEDSTUDENTS DESC;

EXP4 SELECT D.STUDENT_DEMOGRAPHICS_KEY, F.PAPER_KEY, F.DATE_KEY
     FROM DIM_STUDENT_DEMOGRAPHICS D
     INNER JOIN FACT_TABLE F
     ON D.STUDENT_DEMOGRAPHICS_KEY = F.STUDENT_DEMOGRAPHICS_KEY;

EXP5 SELECT F.STUDENT_DEMOGRAPHICS_KEY ,D.ENROLMENT_TYPE_KEY,
          D.ENROLMENT_TYPE_GROUP_DESC FROM DIM_ENROLMENT_TYPE D
     LEFT JOIN FACT_TABLE F
     ON D.ENROLMENT_TYPE_KEY = F.ENROLMENT_TYPE_KEY
     WHERE D.ENROLMENT_TYPE_GROUP_DESC = 'INTERNATIONAL STUDENTS';

EXP6 SELECT D.STUDENT_DEMOGRAPHICS_KEY, F.PAPER_KEY, F.DATE_KEY,D.AGE,
          D.LAST_SECONDARY_SCHOOL_COUNTRY FROM DIM_STUDENT_DEMOGRAPHICS D
     INNER JOIN FACT_TABLE F
     ON D.STUDENT_DEMOGRAPHICS_KEY = F.STUDENT_DEMOGRAPHICS_KEY
     WHERE D.AGE > 25 AND F.LAST_SECONDARY_SCHOOL_COUNTRY = 'NEW ZEALAND';

EXP7 SELECT F.STUDENT_DEMOGRAPHICS_KEY, F.PAPER_KEY, F.DATE_KEY,
          P.PROGRAMME_KEY, P.PROGRAMME_FULL_DESC, I.INTAKE_YEAR
     FROM DIM_PROGRAMME P
     INNER JOIN FACT_TABLE F ON P.PROGRAMME_KEY = F.PROGRAMME_KEY
          INNER JOIN DIM_INTAKE I ON F.INTAKE_KEY = I.INTAKE_KEY
     WHERE P.PROGRAMME_FULL_DESC= 'BACHELOR OF ARTS AND
```

```
        BACHELOR OF BUSINESS CONJOINT DEGREES'
    OR I.INTAKE_YEAR>1990;
```

EXP8
```
SELECT D.STUDENT_DEMOGRAPHICS_KEY, F.PAPER_KEY, F.DATE_KEY,
        F.ENROLMENT_STATUS_FIRST_DAY FROM DIM_STUDENT_DEMOGRAPHICS D
    FULL JOIN FACT_TABLE F
    ON D.STUDENT_DEMOGRAPHICS_KEY = F.STUDENT_DEMOGRAPHICS_KEY
    ORDER BY D.STUDENT_DEMOGRAPHICS_KEY;
```

EXP9 Note: different queries due to how the systems do updates.

Oracle query:

```
UPDATE (SELECT F.PAPER_KEY FROM FACT_TABLE@MYLINK F
     WHERE F.PAPER_KEY IN
          (SELECT D.PAPER_KEY FROM DIM_PAPER D WHERE D.PAPER_KEY= '13362'))
     SET PAPER_KEY = '666666');
```

SQL Server query:

```
UPDATE FACT_TABLE
SET PAPER_KEY = '444444'
WHERE PAPER_KEY IN (SELECT D.PAPER_KEY FROM DIM_PAPER D
WHERE D. PAPER_KEY = '13362');
```

APPENDIX B: EXECUTION PLANS

Execution Plan 1: Experiment 1

SQL Server, Amsterdam

Input: Clustered Index Seek on (Dim_student).(PK_Student_Key), Fact_Table
Output: Query Result as Result
1: Run Nested Loops on Dim_student ⋈`Fact_Table`
2: **if** Tuples meets join condition **then**
3: Select Tuples
4: Print Result
5: **else**
6: Discard Tuples
7: **end if**

Oracle, Amsterdam

Input: Index Unique Scan on (Dim_student).(PK_Student_Key), Fact_Table
Output: Query Result as Result
1: Run Nested Loops on Dim_student ⋈ Fact_Table
2: **if** Tuples meets join condition **then**
3: Select Tuples
4: Print Result
5: **else**
6: Discard Tuples
7: **end if**

SQL Server, Auckland

Input: Full Table Scan on Fact_Table, Dim_student
Output: Query Result as Result
1: Run Sort Operator on Fact_Table
2: Run Nested loops on `Fact_Table`⋈ Dim_student
3: Select Tuples
4: Print Result

Oracle, Auckland

Input: Full Table Scan on Fact_Table, Dim_student
Output: Query Result as Result
1: Run Nested loops on `Fact_Table`⋈ Dim_student
2: Select Tuples
3: Print Result

Execution Plan 2: Experiment 2

SQL Server, Amsterdam

Input: Clustered Index Seek on (Dim_Paper).(PK_Dim_Key), Clustered Index Scan on Dim_Date, Clustered Index Seek on Dim_Enrolment_Type, Fact_Table
Output: Query Result as Result

```
1:  Run Nested Loops on Dim_Paper ⋈ Dim_Date
2:  if Tuples meets join condition then
3:      Run Merge Join on Tuples ⋈ Fact_Table
4:      if Tuples meets join condition then
5:          Run Merge Join on Tuples ⋈
            Dim_Enrolemnt_Type
6:          if Tuples meets join condition then
7:              Select Tuples
8:              Print Result
9:          else
10:             Discard Tuples
11:         end if
12:     else
13:         Discard Tuples
14:     end if
15: else
16:     Discard Tuples
17: end if
```

Oracle, Amsterdam

Input: Index Unique Scan on (Dim_Paper).(PK_Dim_Key), Full Table Scan Dim_Date, Table Scan on Dim_Enrolment_Type, Fact_Table
Output: Query Result as Result

```
1:  Run Buffer Sort on Dim_Date
2:  Run Nested Loops on Dim_Enrolment_Type ⋊
    Dim_Paper
3:  if Tuples meets join condition then
4:      Run Merge Join on Tuples × Dim_Date
5:      if Tuples meets join condition then
6:          Run Hash Join on Tuples ⋈ Fact_Table
7:          if Tuples meets hash condition then
8:              Select Tuples
9:              Print Result
10:         else
11:             Discard Tuples
12:         end if
13:     else
14:         Discard Tuples
15:     end if
16: else
17:     Discard Tuples
18: end if
```

SQL Server, Auckland

Input: Full Table Scan on Fact_Table
Output: Query Result as Result

```
1:  Run Sort Operator on Fact_Table
2:  Run Merge Join onFact_Table ⋈ Tuples
3:  Select Tuples
4:  Print Result
```

Oracle, Auckland

Input: Full Table Scan on Fact_Table
Output: Query Result as Result

```
1:  Run Hash Join onFact_Table ⋈ Tuples
2:  Select Tuples
3:  Print Result
```

Execution Plan 3: Experiment 3

SQL Server, Amsterdam

Input: Clustered Index Scan on (Dim_Paper).(PK_Dim_Paper), Fact_Table
Output: Query Result as Result

1: Run Merge Join Dim_Paper ⋈ Fact_Table
2: **if** Tuples meet join condition **then**
3: Run Sort on Tuples
4: Run Filter on Tuples
5: **if** Tuples meet having condition **then**
6: Select Tuples
7: Print Result
8: **else**
9: Discard Tuples
10: **end if**
11: **else**
12: Discard Tuples
13: **end if**

Oracle, Amsterdam

Input: Full Table Scan on Dim_Paper, Fact_Table
Output: Query Result as Result

1: Run Hash Match on Dim_Paper *hash*`Fact_Table`
2: **if** Tuples meet hash condition **then**
3: Run Hash Group By on Tuples
4: Create View
5: Run Hash Group By on Tuples
6: Filter Tuples
7: **if** Tuples meet having condition **then**
8: Run Sort on Tuples
9: Select Tuples
10: Print Result
11: **else**
12: Discard Tuples
13: **end if**
14: **else**
15: Discard Tuples
16: **end if**

SQL Server, Auckland

Input: Full Table Scan on Fact_Table, Dim_Paper
Output: Query Result as Results

1: Run Merge Join on`Fact_Table` ⋈ Dim_Paper
2: Run Hash Match$_{(\mathfrak{J})}$ on Tuples
3: Run Hash Match$_{(\mathfrak{J})}$ on Tuples
4: Run Sort Operator on Tuples
5: Run Sort Operator on Tuples
6: Run Stream Aggregate $_{(\mathfrak{J})}$ on Tuples
7: Filter Tuples
8: Select Tuples
9: Print Result

Oracle, Auckland

Input: Full Table Scan on Fact_Table, Dim_Paper
Output: Query Result as Results

1: Run Hash Match on`Fact_Table`*hash* Dim_Paper
2: Select Tuples
3: Print Result

Execution Plan 4: Experiment 4

SQL Server, Amsterdam

Input: Clustered Index Scan on (Dim_student).(PK_student_Key), Fact_Table
Output: Query Result as Result
1: Run Merge Join on Dim_Student $\bowtie$ Fact_Table
2: **if** Tuples meet join condition **then**
3: Select Tuples
4: Print Result
5: **else**
6: Discard Tuples
7: **end if**

Oracle, Amsterdam

Input: Index Unique Scan on (Dim_Student).(PK_Student_Key), Fact_Table
Output: Query Result as Result
1: Run Nested Loops on Dim_student $\bowtie$ Fact_Table
2: **if** Tuples meet join condition **then**
3: Select Tuples
4: Print Result
5: **else**
6: Discard Tuples
7: **end if**

SQL Server, Auckland

Input: Full Table Scan on Fact_Table, Dim_student
Output: Query Result as Result
1: Run Sort Operator on Fact_Table
2: Run Merge Join on Tuples $\bowtie$ Dim_student
3: Select Tuples
4: Print Result

Oracle, Auckland

Input: Full Table Scan on Fact_Table, Dim_student
Output: Query Result as Result
1: Run Nested loops on`Fact_Table` $\bowtie$ Dim_student
2: Select Tuples
3: Print Result

Execution Plan 5: Experiment 5

SQL Server, Amsterdam

Input: Clustered Index Scan on (Dim_EnrolmentType).(PK_Dim_Enrolment), Fact_Table
Output: Query Result as Result
1: Run Merge Join on Dim_Enrolment_Type $\rtimes$ Fact_Table
2: **if** Tuples meets join condition **then**
3: Select Tuples
4: Print Result
5: **else**
6: Discard Tuples
7: **end if**

Oracle, Amsterdam

Input: Full Table Scan on Dim_Enrolment_Type, Fact_Table
Output: Query Result as Result
1: Run Hash Match Outer on Dim_Enrolment_Type *hash* Fact_Table
2: **if** hash condition met **then**
3: Select Tuples
4: Print Result
5: **else**
6: Discard Tuples
7: **end if**

SQL Server, Auckland

Input: Full Table Scan on Fact_Table, Dim_Enrolment_Type
Output: Query Result as Result
1: Run Sort Operator on Fact_Table
2: Run Merge Join on`Fact_Table` $\rtimes$ Dim_Enrolment_Type
3: Select Tuples
4: Print Result

Oracle, Auckland

Input: Full Table Scan on Fact_Table, Dim_Enrolment_Type
Output: Query Result as Result
1: Run Hash Match Outer on `Fact_Table` *hash* Dim_Enrolment_Type
2: Select Tuples
3: Print Results

Execution Plan 6: Experiment 6

SQL Server, Amsterdam

Input: Clustered Index Scan on (Dim_Student).(PK_Student_Key)
Output: Query Result as Result
1: Run Merge Join on Dim_Student ⋉ `Fact_Table`
2: **if** Tuples meets join condition **then**
3: Select Tuples
4: Print Result
5: **else**
6: Discard Tuples
7: **end if**

Oracle, Amsterdam

Input: Table Access Full on Dim_Student
Output: Query Result as Result
1: Run Join on Dim_Student ⋉ Fact_Table
2: **if** Tuples meets join condition **then**
3: Select Tuples
4: Print Result
5: **else**
6: Discard Tuples
7: **end if**

SQL Server, Auckland

Input: Full Table Scan on Fact_Table, Dim_Student
Output: Query Result as Result
1: Run Sort Operator on Fact_Table
2: Run Merge Join on `Fact_Table` ⋉ Dim_Student
3: Select Tuples
4: Print Result

Oracle, Auckland

Input: Full Table Scan on Fact_Table,`Fact_Table`as e , Simple Table Scan on Fact_Table, Dim_Student
Output: Query Result as Result
1: Run Join on`Fact_Table` ⋉ e
2: Run Join on Tuples ⋉ Fact_Table
3: Run Join on Tuples ⋉ Dim_Student
4: Run Simple Table Scan on Fact_Table
5: Run Simple Table Scan on e
6: Run Simple Table Scan on Dim_Student
7: Select Tuples
8: Print Result

Execution Plan 7: Experiment 7

SQL Server, Amsterdam

Input: Clustered Index Scan on (Dim_Programme).(PK_Programme_Key), Clustered Index Scan on (Dim_Intake).(PK_Dim_Intake), Fact_Table
Output: Query Result as Result

1: Run Table Spool$_{(LazySpool)}$ on Dim_Intake
2: Run Nested Loops on Dim_Intake $\ltimes$ Dim_Programme
3: **if** Tuples meets join condition **then**
4: Run Merge Join on Tuples $\ltimes$ Fact_Table
5: **if** Tuples meets join condition **then**
6: Select Tuples
7: Print Result
8: **else**
9: Discard Tuples
10: **end if**
11: **else**
12: Discard Tuples
13: **end if**

Oracle, Amsterdam

Input: Table Access Full on Dim_Programme, Table Access Full on Dim_Intake, Fact_Table
Output: Query Result as Result

1: Run Hash Match on Dim_Intake $\ltimes$ Fact_Table
2: **if** Tuples meet hash condition **then**
3: Run Hash match on Tuples $\ltimes$ Dim_Programme
4: **if** Tuples meet hash condition **then**
5: Select Tuples
6: Print Result
7: **else**
8: Discard Tuples
9: **end if**
10: **else**
11: Discard Tuples
12: **end if**

SQL Server, Auckland

Input: Full Table Scan on Fact_Table, Dim_Programme, Dim_Intake
Output: Query Result as Result

1: Run Sort Operator on Fact_Table
2: Run Merge Join on `Fact_Table` $\ltimes$ (Dim_Programme $\ltimes$ Dim_Intake)
3: Select Tuples
4: Print Result

Oracle, Auckland

Input: Full Table Scan on Fact_Table,`Fact_Table`as e, Dim_Intake
Output: Query Result as Result

1: Run Join on `Fact_Table` $\ltimes$ e
2: Run Join on Tuples $\ltimes$ Fact_Table
3: Run Join on Tuples $\ltimes$ Dim_Intake
4: Select Tuples
5: Print Result

Execution Plan 8: Experiment 8

SQL Server, Amsterdam

Input: Clustered Index Scan on (Dim_Student).(PK_Student_Key), Fact_Table
Output: Query Result as Result
1: Run Merge Join on Dim_Student $\bowtie$ Fact_Table
2: **if** Tuples meets join condition **then**
3: Sort Tuples
4: Select Tuples
5: Print Result
6: **else**
7: Discard Tuples
8: **end if**

Oracle Second Approach, local

Input: Index Fast Full Scan on (Dim_Student).(PK_Student_Key), Fact_Table
Output: Query Result as Result
1: Run Join on Dim_Student $\bowtie$ Fact_Table
2: **if** Tuples meets join condition **then**
3: Create View
4: Select Tuples
5: Print Result
6: **else**
7: Discard Tuples
8: **end if**

SQL Server, Auckland

Input: Full Table Scan on Fact_Table
Output: Query Result as Result
1: Run Sort Operator on Fact_Table
2: Run Merge Join on Fact_Table $\bowtie$ Dim_Student
3: Select Tuples
4: Print Result

Oracle, Auckland

Input: Full Table Scan on Fact_Table, Dim_Student
Output: Query Result as Result
1: Run Join on Fact_Table $\bowtie$ Fact_Table
2: Run Join on Tuples $\bowtie$ Dim_Student
3: Simple Table Scan on Fact_Table
4: Simple Table Scan on Dim_Student
5: Select Tuples
6: Print Result

Execution Plan 9: Experiment 9

SQL Server Second Approach, local

Input: Clustered Index Scan on (Dim_Paper).(PK_Paper_Key), Constant Scan on Fact_Table
Output: Query Result as Result
1: Run Nested Loops on `Fact_Table` ⋉ Dim_Paper
2: **if** Tuples meets join condition **then**
3: Update Tuples
4: Print Results
5: **else**
6: Discard Tuples
7: **end if**

Oracle, Amsterdam

Input: Dim_Paper, Table Full Access on Fact_Table
Output: Query Result as Result
1: Run Nested Loops on Dim_Paper ⋉ Fact_Table
2: **if** Tuples meets join condition **then**
3: Update Tuples
4: Print result
5: **else**
6: Discard Tuples
7: **end if**

SQL Server, Auckland

Input: Table Scan on Fact_Table, Dim_Paper
Output: Query Result as Result
1: Run Nested Loops on`Fact_Table` ⋉ Dim_Paper
2: Table Update on Tuples
3: Update Tuples
4: Print Result

Oracle, Auckland

Input: Full Table Access on Fact_Table, Dim_Paper
Output: Query Result as Result
1: Run Nested Loops on `Fact_Table` ⋉ Dim_Paper
2: Update Tuples
3: Print Result