SUMMER INTERNSHIP REPORT

Udit Yadav
Enrollment No- 13UCS073

July 5, 2016

CONTENTS

Part I

ACKNOWLEDGEMENTS

ACKNOWLEDGEMENTS

The internship opportunity I had with Amdocs was a great chance for learning and professional development . Hence, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of this wonderful organisation . I am also grateful for having a chance to meet so many wonderful people and professionals who led me though this internship period.

Bearing in mind previous, I am using this opportunity to express my deepest gratitude and special thanks to our project guide Mr. Bhanu Patial [Software Development Specialist, DLTE, Amdocs] who in spite of being extraordinarily busy with his duties, took time out to hear, guide and keep me on the correct path and allowing me to carry out the project at their esteemed organization and extending during the training.

It is my radiant sentiment to place on record my best regards, deepest sense of gratitude to my mentor Mr. Mohit Anand [Software Engineering Line Manager, DLTE, Amdocs] and Mr. Sameer Gupta [Software Development Manager, DLTE, Amdocs] for their careful and precious guidance which were extremely valuable for my study both theoretically and practically.

I perceive as this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired career objectives. Hope to continue cooperation with all of you in the future. Sincerely,

Udit Yadav
Place:Amdocs, Pune
Date:

Part II

EXECUTIVE SUMMARY

2

EXECUTIVE SUMMARY

This project involved designing a performance enhancer tool specific for the applications working with ORACLE. In this age of technology, saving time and money are the important factors which decide whether a particular product can deliver or not . So, it's equally important to devise such auxiliary tools which save our resources. Hence, this software product was engineered to further support the existing system in a relatively ingenious way instead of relying on the third party vendors for getting the similar level of performance at an unreasonable cost.

Initially, individual modules of the product were designed separately. The whole team brainstormed and integrated ideas for the components. Sketches were created and discussions were held regarding the proposed function of the components and the overall product. During this stage we came across a myriad number of options which were studied extensively. Pros and cons of every possible solutions were addressed and the best combination of modules were finally decided.

Following development of these designs, some modules were developed and tested both individually and in combination with other components. The performance depending on the range of inputs were studied and few drawbacks during the development stage addressed properly. After testing of prototypes, the device was modified to improve efficiency and overall functionality. Some components were removed and replaced by more efficient processes. In order to meet the time criterion and efficiency, additional components were added during early testing of the prototype.

In construction of the final product, some changes were made to improve stability, reliability and effectiveness. Supportive modules were added to the final product to provide a flexibility and stability for the integration with the other main applications.

Finally the tests conducted on the final product yielded consistent results which was quite closer to the expected values.

Hence a short description of the developed product is as follows:" ***xSQLCache is a Pro*C performance improvement product to improve system performance by caching query result sets within client process. Its seamless integration design allows it to be directly added to build process. "***

Part III

INTERNSHIP OVERVIEW

3

INTERNSHIP EXPERIENCE

Usually each day of these months of internship in itself proved to be a great learning experience. So summing these innumerable experiences in just a few sentences is quite tedious. However to be truthful these kind of professional experience would open one's eyes and would definitely bust the common myths about the professional world.

This internship was a golden chance for me before I get into the real professional world. It helped me immensely in acquiring essential professional skills. It was a chance to develop the communication and interpersonal skills, to build knowledge base about a specific field, and to practice a higher level of responsibility and confidence.

Apart from overall individual growth I got the opportunity to be a part of research and development team of the organisation which taught me that research is not always about publishing long papers. The more you implement ideas in real world instead of on paper, the more closer you get towards the success - this is one of the most important lifetime experience I got. Also being in this team I got an exposure to a whole different world of software development in C which many people would runaway on hearing. But because of this experience now I love C as the language of preference where performance always matters.

Everyday I learnt new ways of implementing ideas. Also keeping these implementations easy and simple is what my mentor would always say because this way not only you can understand but also you can get the things done in more smarter way. Working in the back end of many processes is always frustrating - this is what I used to think earlier. But here I got a whole new outlook about this. I finally realized back end is where you can actually know how simultaneously uncountable number of processes occur and tweaking these processes to improve the whole system can be fun and at the same time they can prove to be a great learning experience.

Part IV

PROJECT OVERVIEW

4

INTRODUCTION

4.1 PROJECT VISION

Development of a software tool called ***xSQLCache*** for the improvement in the performance of what is arguably the most common type of SQL accesses: **compiled SELECT** statements. The improvement is achieved transparently, without the need to modify source programs. The performance improvements include ***saving***:

- CPU processing time for the process that issues SELECT statements.
- Overhead for interfacing with the ORACLE database server.
- Telecommunications overhead in case of remote databases.
- CPU and I/O for the ORACLE database server, reducing its workload and thus improving the overall system-wide database performance.

The SQL statements are coded in C programs.

- For a C program: the program is processed by the xSQLCache preprocessor for C and then by the standard Pro*C ORACLE precompiler.
- Control blocks and their settings as generated by the ORACLE precompiler and by the xSQLCache preprocessor are not modified.
- Enough storage is defined and available for keeping active data in.

5

ASSUMPTIONS AND CONSTRAINTS

5.1 ASSUMPTIONS

The following assumptions need to be true when this tool is integrated in the application :

- The tables in databases are **update-insensitive**.
- The retrievals are **repetitive SELECTs** issued from the same process.
- There are no run-time errors or warning conditions other than no data found.
- Enough storage is defined and available for keeping active data in.

5.2 CONSTRAINTS

In general, using xSQLCache for tables/accesses not within its scope may result in some slight degradation in performance, negligible in most cases. However, the use of xSQLCache must be strictly avoided in the following cases:

- Programs that SELECT from update-sensitive tables (when the most up-to-date values must be obtained from tables that are or may be updated concurrently to retrievals from them). Note that as the tables to be handled by xSQLCache must be explicitly defined by the administrator or the programmer, this is not really a problem.
- Programs that switch between ORACLE users (i.e., CONNECT as one ORACLE user and then CONNECT to another). Note that this is not very common.

6

PROJECT ORGANISATION STRUCTURE

6.1 RESOURCE ASSESSMENT

Preliminary analysis of the requirements were done on the basis of which corresponding plan for the development stages were devised.

6.2 DESIGN

Following the resource assessment, various primitive designs of the product were discussed intensively. After the analysis of pros and cons of the designs finalization of the design was done.

6.3 ROLES AND RESPONSIBILITIES

After various designs were discussed, roles were assigned to each member of the developer team.

6.4 DEVELOPMENT

All the members in their respective environment did their respective task. Here the developers had the complete freedom to try their own ideas without adversely affecting the rest of the development team.

6.5 INTEGRATION

In a common environment all the developers commit their work where all the modules of the entire project were combined and tested.

6.6 TESTING AND IMPLEMENTATION

After the integration stage , the testing of the product was done for various test cases. Accordingly, revisions were done in the corresponding modules which required attention.

Part V

PROJECT DEVELOPMENT STAGES

7

STAGE: RESOURCE ANALYSIS(PREREQUISITE KNOWLEDGE)

7.1 BRIEF INTRODUCTION TO SQL

7.1.1 *What is SQL*

SQL (pronounced SEQUEL) is the programming language that defines and manipulates the database. SQL databases are relational databases, which means that data is stored in a set of simple relations.

7.1.2 *What SQL can do ?*

SQL can :

- execute queries against a database
- retrieve data from a database
- insert records in a database
- update records in a database
- delete records from a database
- create new databases
- create new tables in a database
- create stored procedures in a database
- create views in a database
- set permissions on tables, procedures, and views

7.1.3 *Ways to use SQL*

In the original version of SQL, users typed commands into a file or directly at the terminal and received responses immediately. People still sometimes use it this way for creating tables and for debugging, but for the vast majority of applications, SQL commands come from

inside programs, and the results are returned to those programs. The SQL standard defines a module language to embed SQL in a variety of programming languages.

7.2 EMBEDDED SQL

7.2.1 *What is "Embedded" SQL*

Embedded SQL refers to the use of standard SQL statements embedded within a procedural programming language.

7.2.2 *SQL vs Embedded SQL*

Embedded SQL is a collection of these statements: All SQL commands, such as SELECT and INSERT, available with SQL with interactive tools ,also Dynamic SQL execution commands, such as PREPARE and OPEN, which integrate the standard SQL statements with a procedural programming language. Embedded SQL also includes extensions to some standard SQL statements. Embedded SQL is supported by the Oracle precompilers. The Oracle precompilers interpret embedded SQL statements and translate them into statements that can be understood by procedural language compilers.

Each of these Oracle precompilers translates embedded SQL programs into a different procedural language: Pro*C/C++ precompiler and Pro*COBOL precompiler.

7.2.3 *Advantages of Embedded SQL*

- In embedded SQL, host language takes care of the variables and other input and output functions whereas SQL is used to handle the database. Thus, tasks are carried out effectively.
- High overhead tasks like parsing and optimizing are done in the development cycle which increases efficiency of CPU resources.
- By using DBRM(Database Request Module), portability can be achieved. In one system precompiling can be done and it can be be moved to the other system for further processing.
- The programmer will not know the private complex DBMS routines. He just has to create the embedded SQL source program code.

7.3 UNDERSTANDING PRO*C AND ITS FEATURES

7.3.1 *Introduction*

Embedded SQL is a method of combining the computing power of a high-level language like C/C++ and the database manipulation capabilities of SQL. It allows you to execute any SQL statement from an application program. Oracle's embedded SQL environment is called Pro*C.

A Pro*C program is compiled in two steps. First, the Pro*C precompiler recognizes the SQL statements embedded in the program, and replaces them with appropriate calls to the functions in the SQL runtime library. The output is pure C/C++ code with all the pure C/C++ portions intact. Then, a regular C/C++ compiler is used to compile the code and produces the executable.

7.3.2 *Pro*C Syntax SQL*

All SQL statements need to start with EXEC SQL and end with a semicolon ";". You can place the SQL statements anywhere within a C/C++ block, with the restriction that the declarative statements do not come after the executable statements.

7.4 OVERVIEW OF ORACLE DATABASE

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multi user environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

7.5 BASICS OF CACHE MECHANISM

In this project we implemented some basic hashing techniques for cache structure.

7.5.1 *Hash-Table*

To store the data in associative manner data structures such as **hash tables** are mostly used. In hash table, data is stored in such a way that each data value has its corresponding unique index value which

is generated using hashing function. With prior knowledge of the index of the desired data, access of the data becomes very fast.

Since, it becomes a data structure in which insertion and search operations are very fast irrespective of size of data Hash Table can be used as a storage medium.

7.5.2 *Hash Function*

The hash function is a mapping from the input space to the integer space that defines the indices of the hash-table. In other words, the hash function provides a way for assigning numbers to the input data such that the data can then be stored at the hash-table index corresponding to the assigned number.

7.6 SQL QUERY ANALYSER

The SQL query analyser is a tool which helps in debugging the user given statements and also for extracting useful information. Here, the statement is broken down into its component parts determining what type of a statement it is whether it is a DML(Data Manipulation Language) or a DDL(Data Definition Language) or a select query. The checks that are next done on it are the syntax and the semantics checks. The syntax check validates the statement based on the SQL grammar. The semantics checks make sure whether the statement is a valid one in the light of the schema objects that we have i.e. do those tables, views etc. exist in the schema or not and whether we have the access to those objects and are proper privileges in place for us to execute that statement, whether there are ambiguities in the statement.

8

STAGE: MODULAR DEVELOPMENT

8.1 SQL STATEMENT ANALYSER

This module analyses the input statement and accordingly useful information are extracted. Type of statement, number of variables, nature of those variables and their location, etc. can be inferred from the use of this module. In the later stage this module will again be modified on the basis of the preliminary test results.

8.2 ENVIRONMENT VARIABLES

Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer. They can be said in some sense to create the operating environment in which a process runs. For example, an environment variable with a standard name can store the location that a particular computer system uses to store temporary files this may vary from one computer system to another. A process which invokes the environment variable by (standard) name can be sure that it is storing temporary information in a directory that exists and is expected to have sufficient space.

8.3 CONTROL DEFINITIONS FILES

8.3.1 *Overview*

xSQLCache_Configuration_handler is a control definition file comprising of various control records to be handled by xSQLCache. Each of the record consist of a keyword, an equal sign(=)and a value for the keyword terminated with a new line. It contains values of various parameters related to xSQLCache. When some program is executed it is consulted to see what parameters are in effect. The parameters present are performance statistics, debug message level, enable/disable cache system, table name or set of table name.

8.3.2 *Keywords*

The keywords are :

- PER_STAT
  - This section describes the use of 'PER_STAT' keyword in xSQL_Configuration_handler File. It set a value that generates the performance statistics of the xSQLCache. It is used to effectively diagnose performance problems. The value to be set should be either 'Y'(stands for Yes) or 'N'(stands for No). 'Y' signifies that it is enabled and 'N' signifies it is disabled. Any other value not starting with 'Y' will disable performance statistics.
  - PER_STAT should be written in uppercase, it is case-sensitive. The default value set by it is 'N'.
  - The general synopsis for 'PER_STAT' is PER_STAT=Y/N. Here is an example of doing this: PER_STAT=Y.
- MSG_LEVEL
  - This section describes the use of 'MSG_LEVEL' keyword in xSQL_Configuration_handler File. It sets a value that represents a debug information/message for printing data in logs/Standard output.The value to be set should be one of the following:
    - 'A' (stands for all).

      Signifies that it will display both exception and informational messages.
    - 'E' (stands for exception).

      Signifies that it will display exception messages.
    - 'I' (stands for informational).

      Signifies that it will display informational messages.
    - 'W' (stands for warning).

      Signifies that that it will display some warning messages.
    - 'F' (stands for fatal).

      Signifies that that it will display some fatal error messages.
    - 'D' (stands for debug).

      Signifies that that it will display some debug information messages.
  - MSG_LEVEL should be written in uppercase, it is case-sensitive. No default value is set by it, keep it empty if no value is to be set.

  - The general synopsis for 'MSG_LEVEL' is MSG_LEVEL=A/E/I/W/F/D

    Here is an example of doing this: MSG_LEVEL=A .

- CACHE_ENABLE
  - This section describes the use of 'CACHE_ENABLE' keyword in xSQL_Configuration_handler File. It sets a value that states whether the cache should be enabled or disabled. The value to be set should be 'Y'(stands for yes) or 'N'(stands for no).'Y' signifies that cache should be enabled and 'N' signifies that it is disabled. Any other value not starting with 'Y' will disable cache.
  - CACHE_ENABLE should be written in uppercase, it is case-sensitive. The default value set by it is 'N'.
  - The general synopsis for 'CACHE_ENABLE' is CACHE_ENABLE=Y/N.

    Here is an example of doing this: CACHE_ENABLE=Y.

- MAX_MEM
  - This section describes the use of 'MAX_MEM' keyword in xSQL_Configuration_handler File. It sets a value that specifies the maximum amount of memory(in Kilobytes) that xSQLCache can use. Low value degrades performance and when specified value exceeds requirements extra storage is not used.
  - "NOTE: MAX_MEM to be followed immediately by an =(equal) sign else the parser will generate syntax error".
  - MAX_MEM should be written in uppercase, it is case-sensitive.
  - The general synopsis for 'MAX_MEM' is MAX_MEM=size.

    Here is an example of doing this: MAX_MEM=100.

- TABNAME
  - This section describes the use of 'TABNAME' keyword in xSQL Configuation_handler File. It gives the list of table names to be handled by xSQLCache. It can be a table name or a comma-separated list of table names for a JOIN combination. Table name aliases, separated by spaces from the corresponding table names are allowed.
  - "NOTE: TABNAME to be followed immediately by an =(equal) sign else the parser will generate syntax error".
  - Spaces are allowed before and after commas. TABNAME should be written in uppercase, it is case-sensitive.
  - The general synopsis for 'TABNAME' is TABNAME=tablename,tablename,..etc. There can be zero or more tablename separated by comma.

    Here is an example of doing this: TABNAME=A,B,C .

8.4 CACHE MECHANISM OF XSQLCACHE

8.4.1 *Overview*

xSQLCache caches data retrieved from Oracle in a cache mechanism to achieve performance improvement. When a SELECT statement is issued by application program, xSQLCache first looks for the data in the cache. If the data is already in the cache, xSQLCache retrieves the cached data to calling application. Otherwise, xSQLCache delegates SQL statement request to Oracle and only then caches the data retrieved from Oracle allowing subsequent request for same data to be retrieved cacheable.

8.4.2 *Caching Unit*

The unit of caching includes all values of input host-variables (those appearing in the WHERE clause) and all values of output host-variables (those appearing in the INTO clause) associated with each invocation of a SELECT statement.

8.4.3 *Structure and Content*

The structure of the cache of xSQLCache is based on the Hash (key/data) based mechanism.

1. Entry Key:

   The key is a concatenation of the following fields:

   - Accepted SELECT statement.
   - Input Host Variables All values of host variables appearing in the WHERE clause of the statement including.

2. Entry Data/Value:

   The data is a concatenation of all output host variables (including null indicators, length fields where needed) appearing in the INTO clause.

8.4.4 *Cache Management*

For each accepted statement with no previous errors, Quick-SELECT will attempt to cache the data retrieved from Oracle.

- When storage limitation allows and Hash table is not full, xSQLCache will allocate new key/data buffers and update Hash table with the new cache entry.

- When cache memory is too small to hold even one record, xSQL-Cache will produce an error message and stop handling that SQL statement.

8.5 CONTROLLING RUN-TIME MESSAGES

The minimum severity level of run-time messages defines which messages will be issued. The default is 2, which can be overridden by the corresponding environment variable or MSG_LEVEL the control definitions record. Valid values are :

- 1
  - All exception and informational messages.
- 2
  - Informational messages.
- 3
  - Warning messages.
- 4
  - Fatal error messages.
- 5
  - Debug information messages.

8.6 RUN-TIME STATISTICS

Run-time statistics is an alternative approach to program verification in which individual program traces are checked against a specification. Given a trace of a program execution, we report success if the trace satisfies the program specification, and failure if a fault is detected. Often, however, it is more helpful to watch indicators of an impending failure.

Hence, xSQLCache run-time statistics can be provided for monitoring and evaluating the operation of xSQLCache. Some of the values obtained from the statistics can be used for performance tuning.

9

STAGE: INTEGRATION

In a common environment all the developers commit their work where all the modules of the entire project were combined and tested. Here, we used Git a version control system as the platform for collaboration of all the modules.

Git's design is essential in maintaining a large distributed development project. Some of the special features highlighting Git's implementation choices are as follows:

- Strong support for non-linear development
  - Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history. A core assumption in Git is that a change will be merged more often than it is written, as it is passed around various reviewers. Branches in Git are very lightweight: A branch in Git is only a reference to a single commit. With its parental commits, the full branch structure can be constructed.

- Distributed development
  - Git gives each developer a local copy of the entire development history, and changes are copied from one such repository to another. These changes are imported as additional development branches, and can be merged in the same way as a locally developed branch.

- Compatibility with existing systems/protocols
  - Repositories can be published via HTTP, FTP, rsync (until Git 2.8.0[30]), or a Git protocol over either a plain socket, or ssh. Git also has a CVS server emulation, which enables the use of existing CVS clients and IDE plugins to access Git repositories. Subversion and svk repositories can be used directly with git-svn.

- Efficient handling of large projects
  - Git has proved itself to be very fast and fetching of history from a local stored repository can be hundred times faster than fetching it from the remote server.

- Cryptographic authentication of history
  - The Git history is stored in such a way that the ID of a particular version (a commit in Git terms) depends upon the complete development history leading up to that commit. Once it is published, it is not possible to change the old versions without it being noticed. The structure is similar to a Merkle tree, but with additional data at the nodes as well as the leaves.
- Pluggable merge strategies
  - Git was designed as a set of programs written in C, and a number of shell scripts that provide wrappers around those programs. Although most of those scripts have since been rewritten in C for speed and portability, the design remains, and it is easy to chain the components together.
- Garbage accumulates unless collected
  - As part of its toolkit design, Git has a well-defined model of an incomplete merge, and it has multiple algorithms for completing it, culminating in telling the user that it is unable to complete the merge automatically and that manual editing is required.
- Periodic explicit object packing
  - Aborting operations or backing out changes will leave useless dangling objects in the database. These are generally a small fraction of the continuously growing history of wanted objects. Git will automatically perform garbage collection when enough loose objects have been created in the repository. Garbage collection can be called explicitly using git gc –prune.

10

STAGE: TESTING AND IMPLEMENTATION OVERVIEW

10.1 INITIALIZATION

- The make/script files for compiling/linking with xSQLCache are prepared.
- The administrator provides a few details to xSQLCache, such as which tables and JOINs to handle, how much storage to use, etc. These definitions can be overridden by a programmer for each process.

10.2 USAGE

- Performance is improved for every SELECT statement within the xSQLCache scope in every program compiled/linked with xSQLCache. No source program needs to be modified for obtaining the performance improvements.
- Any program not compiled/linked with xSQLCache is unaffected by it. In case of separately compiled routines linked together, only those compiled with xSQLCache use it, and the other routines are unaffected.
- If, for whatever reason, xSQLCache has to be disabled, this can be done for a specific program, for a specific process, or globally for all programs.

Part VI

ANALYSIS

11

PERFORMANCE ANALYSIS AND IMPROVEMENTS

xSQLCache provides:

- Performance improvement, including:
  - Saving CPU time for the process that issues SELECT statements, whether the SELECT is successful or results in no data found.
  - Saving system overhead for interfacing with the ORACLE database server.
  - Saving telecommunications overhead for remote databases.
  - Saving CPU and I/O for the ORACLE database server, reducing its workload and thus improving the overall system-wide database performance.
- Transparency – requiring no changes in source programs.
- Ease of installation, operation and disabling.
- Control over the use of resources and participating database tables

With the few exceptions detailed below, the full SELECT syntax is supported, including JOINs for any number of tables, all the built-in functions and any complexity of expressions and subqueries. Host arrays are supported as well. Performance is improved not only for successful SELECTs, but also for SELECTs resulting in no data found.

12

CONCLUSION

xSQLCache is a ProC performance improvement product to improve system performance by caching query result sets within client process. Its seamless integration design allows it to be directly added to build process.

Since it is based on best practices, any process innovation that takes place is added to the reference model of the tool, thereby gradually but continually raising the bar. In addition, now that essential building blocks are in place, more refined processes can be introduced. Now that progress has been made by all teams to fulfill requirements of existence and execution of processes, the next iteration of assessments will be more targeted on effectiveness and efficiency of processes; the key elements of higher levels of maturity.