Context-Sensitive Data Security for Business Applications' Performance Optimization

Arjun K Sirohi
Oracle USA Inc,
Bellevue, WA, USA
Arjun.Sirohi@oracle.com

***Abstract*— the importance of data security in enterprise business applications has risen sharply in recent years, especially due to the many new regulations and the rise in cloud-based environments. Most solutions use Role Based Access Control (RBAC) which secures the data using data security predicates (DSP) in the form of Structured Query Language (SQL) WHERE clauses. The current methods cause a huge drag on the SQLs' performance. In this paper, I focus on the effects of RBAC on the performance of SQL queries. I propose a solution to achieve performance goals in RBAC data security by ensuring that the DSPs are made context-sensitive, keeping in mind the context in which the end user is navigating the application. By ensuring context-sensitive data security, the complexity of the SQL queries is reduced and the underlying RDBMS can create simpler, better optimized execution plans. Several experiments using existing applications show very good performance improvements.**

Keywords— Data Security, RBAC, Performance Improvement, Context-Sensitive Data Security.

I. INTRODUCTION

Information security and specifically data access security has been a subject of much research for past many years. Controlling who has access to what data in an enterprise application system is a key concern while implementing security. One of the standard concepts to control data access is the Role Based Access Control (RBAC) which has become the de facto industry standard. In 2004, The American National Standards Institute, International Committee for Information Technology Standards (ANSI/INCITS) adopted Sandhu, Ferraiolo, Kuhn RBAC proposal as an industry consensus standard [1, 2]. Wikipedia describes role-based access control (RBAC) [3] as an approach to restricting system access to authorized users. This is adopted and implemented by many manufacturers of widely used enterprise applications. While there has been much research on RBAC, its implementation and security benefits, there has not been much research on the impacts of such a data security model on the performance and scalability of enterprise business applications. Most commercial enterprise business applications use a relational database like Oracle at the back end and all processing of data is managed through SQL queries. At the lowest grain, the performance of such SQL queries often dictates the performance and scalability of the application, among other factors. While there are many different application layers involved in data security, in this paper I examine the existing RBAC approach of applying data security predicates based on enterprise applications' job roles, duty roles and data roles and their impacts on the performance of SQL queries in the underlying relational database. While RBAC-based data security may not have huge performance impacts when users have only one defined role, performance degradations start to arise when users have multiple roles in an enterprise and they have access to data via many different data security predicates' sub-queries. For example, a sales manager using a Customer Relationship Management application such as Lead Management or Opportunity Management could have multiple roles and implicit roles through group memberships and hierarchy. Based on these roles, the sales manager's access to data would be controlled by a union of all the data security predicates, which may sometimes overlap. [4] The performance problems are further exacerbated by the fact that modern enterprise business applications have a very complex data model with the data typically stored in many different database tables. Accessing such data even without data security predicates often results in complex SQL queries whose performance is a challenge by itself. Such performance problems affect dynamically generated SQLs, which is the new norm. Appending of RBAC-based data security predicates to such complex SQL queries often results in poor performance in the database. The use of the term "context-aware" for computer applications is not very new and was described by Schilit et al. in 1994 [8]. The term has also been applied to business process management [6]. While context can have many different facets and categorizations, Kaltz et al. [7] identified user and role to be one of such categorizations. Existing work done by researchers in applying context to RBAC in computer applications has been mostly on the aspects of implementation. For example, Kulkarni and Tripathi have discussed the issues related to the implementation of RBAC for pervasive computer applications [9] by utilizing context information. In the realm of enterprise business applications, the user, the user's role(s) in the enterprise and the task that the user aims to perform at a given point in time are the key aspects of RBAC context. The impacts of such RBAC policies and systems on the performance and scalability of enterprise business applications has not been the focus of much research. In this paper, I focus on the effects of RBAC on the performance of SQL queries in an enterprise business application and propose the use of context in conjunction with RBAC to optimize the performance and scalability of such

applications. In this paper, I use the term "context-sensitive" instead of "context-aware" to describe such aspects of an enterprise business application user's context that have a direct bearing on the performance of SQLs and thereby on applications that use RBAC. A context-sensitive RBAC architecture is not only context-aware but it goes one step further to make use of such context awareness to take certain actions that would significantly improve the performance of the SQL queries and as a result improve the performance and scalability of the application.

II. MOTIVATION AND BACKGROUND WORK

A. *Current RBAC Framework and Seeded Data Security Predicates' Sub-Queries*

In the current implementations in enterprise business application like Fusion Applications from Oracle, the data visibility is controlled through RBAC [10]. Users are granted certain roles and when a user has multiple roles, RBAC ensures that the user's access is the union of all granted roles. The actual implementation of RBAC is achieved through the use of data security predicates' (DSP) sub-queries which are created and stored in a central grants table in the database. The security framework ensures that at run time, the required instance sets are put together based on the grants for a defined job role and duty roles. At run time, when a user logs in to the application and starts to use the business flows which require SQL queries to be sent to the back-end relational database, all of the user's accessible DSP sub-queries are OR-ed together and appended to the main SQL by the framework. As part of business flows, a user typically defines an additional search criteria which gets appended as an IN or EXISTS sub-query to the main SQL. The SQL thus generated at run time takes the following generic form:

SELECT <required columns>

FROM <required tables/views>

WHERE

<Joins>

AND

<Main Query Predicates>

AND

(DSP_Sub-Query-1

OR

DSP_Sub-Query-2

OR

DSP_Sub-Query-3

OR

.....

DSP_Sub-Query-n)

AND

<Business Flow Use Case sub-query>

B. *Impact of RBAC on Performance of SQL Queries*

When analyzing the SQLs for the many use cases, I found that, in a lot of cases, the relational database cost-based optimizer (CBO) was unable to find an optimal execution plan based on the use case scenario being executed. One part of the problem identified was that the DSP sub-queries were written in many different forms, some using EXISTS, some using IN, some using a combination of EXISTS and IN, others using UNION, Nested-Sub-Queries etc. In addition, the problem is compounded by the fact that the DSP sub-queries reference a wide range of different tables, views and Materialized Views (MVs), including hierarchical ones that make these sub-queries extremely complex. Even though CBOs have made excellent advancements in recent years to process SQLs more efficiently, given the SQLs' complexity and size, with many EXISTS and OR'ed DSP sub-queries, CBO often has trouble with handling all the transformations and costing of such SQLs and this has a major impact on their performance. Additionally, the CBO transformations, even when achieved, are very complex and expensive. In existing implementations of RBAC, for a given user, based on his/her job role and duty roles, certain DSP sub-queries are appended to the main SQL at run time. Analysis of a large number SQLs for the use cases brought out another very important finding. It showed that the application and security framework did not make any allowance for the context in which the given user was executing the SQL. For example, when the logged in user switches context in the application to view his/her marketing leads, for example, the SQL generated by the framework at run time appends all the OR'ed data security predicates to the SQL that the user's job/duty role has access to. This is based on the RBAC model that a user's data visibility is controlled by the union of all grants. It did not matter, that the user was looking to find only the data for leads where he/she was the owner of Leads as controlled by owner_id column, and that this column was already the main filtering predicate in the main SQL, making the evaluation of all the DSP sub-queries redundant. As a result, the database CBO needs to evaluate (and most often materialize) all the rows that the user was entitled to see, only to discard them later in the execution plan and keep only the rows that the user wanted to view. This was very evident in the execution plans analyzed. For example, the CBO execution plan statistics may show that the OR'ed DSP sub-queries resulted in being evaluated by the optimizer individually and then combined with a UNION ALL to 1000 rows. Then, subsequent plan operations may show that based on the outer query's context predicate, all but 10 rows out of the 1000 were thrown away. Obviously, this is extremely inefficient and quite a waste of db resources. This lack of context sensitivity not only leads to the complexity of the SQL that adversely affects the optimizer's choice of execution plans, but it also leads to extremely expensive operations and wasted resources at the database in terms of memory, disk input-output (IO) and processor (CPU).The Analysis of the SQLs, their execution plans and results brought out another very interesting finding. I found that for most of the business use cases, the use-case sub-query,

appended dynamically to the SQL at run time, had an equivalent DSP sub-query. This highlights the fact that the rows that qualify using the business flow search sub-query are essentially a sub-set of the rows that qualify from the UNION of all the OR'ed DSP sub-queries of the RBAC framework. There were only a handful of exceptions to this finding where all DSP sub-queries must be evaluated to satisfy the user's query. Taking this a step further, essentially it means that the DSP sub-queries may be logically redundant in many use cases. In enterprise business applications that use RBAC DSP sub-queries to control data visibility, the underlying relational database's CBO faces a dilemma, which is sometimes called the "Tiny-Huge, Huge-Tiny" problem. The dilemma is to decide whether to drive from the main outer query or drive from the RBAC-based DSP sub-queries based on an estimate of which one will be more restrictive in terms of the number of qualifying rows. There are many factors that can influence the CBO's decision and the use of bind variables in dynamically generated SQL queries makes it all the more difficult to make this estimation.

III. Context-Sensitive Data Security for Business Applications' Performance Optimization

Given the many factors affecting performance of SQLs that get appended with multi-role RBAC-based DSP sub-queries, I propose making data security to be context-sensitive at run-time. The proposal achieves this by using a two-layered solution that can significantly improve the performance of such SQLs by using context information to filter the row-set that would result from the union of all the DSP sub-queries. The idea behind this is two-fold – one, simplify the SQL being sent to the database for execution and two, restrict the row-set processing to exactly what the user wishes to retrieve, thereby removing inefficiencies in the execution process. Figure 1 illustrates the concept of context capturing and processing to generate context-sensitive DSP sub-queries.

A. *Context-Capture Module*

In the proposed solution, the "Context-Capture" module resides on the user interface layer of the business application. Its job is to follow the logged-in user's actions and capture the context attributes up to the point where the user action starts the processing of their request to access data from the underlying relational database. These context attributes may be purely physical attributes like geographic location, or may directly correspond to the user's desire to access a specific set of data based on the role and responsibilities being exercised for the particular use case. User-driven context attributes may be filter predicates that map to some table's columns in the underlying database or they could be the user's desire to be acting in a certain job or duty role at that moment. Most modern application development frameworks provide ways of developing such a "Context-Capture" module. For example, Oracle's Application Development Framework (ADF) provides a "Contextual Event Framework" with appropriate listeners that can be used to capture binding attributes, actions and events from the UI [17].

B. *The RBAC "Context-Processor" Module*

This is the most important component in generating context-sensitive data security predicates. It resides in the middle tier of the business application as part of the security framework. The "Context-Processor" takes two inputs. The first input is the context attributes captured from the UI by the "Context-Capture" module. The second input is the complete set of DSP sub-queries from the database that are applicable to the current user which are obtained through the RBAC security framework. For example, take the case where a user has been granted multiple roles that authorize data access from a variety of access paths. The user, while navigating the application, decides that for that particular UI flow, he/she would like to see data returned only for one specific role they are

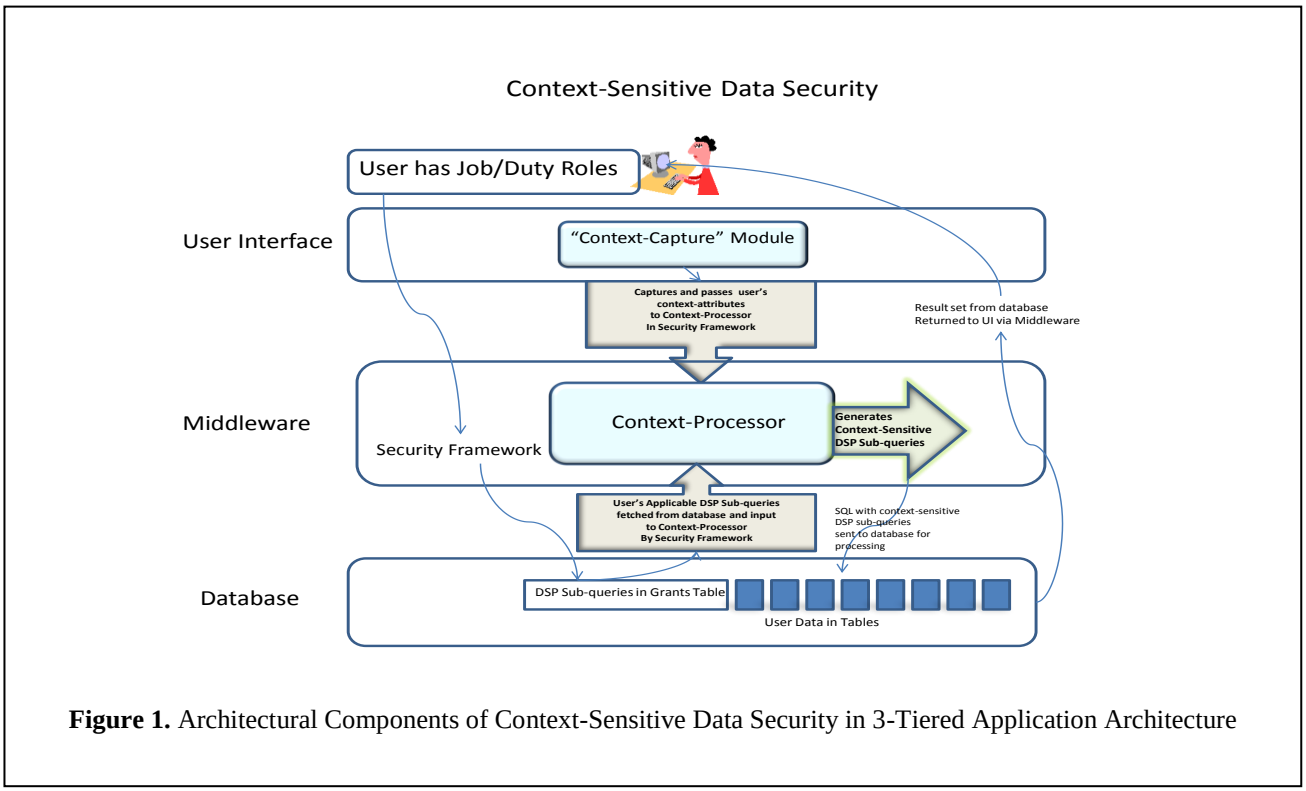

Figure 1. Architectural Components of Context-Sensitive Data Security in 3-Tiered Application Architecture

working on at the moment. In this situation, the "Context-Processor" would use this context attribute to parse the DSP sub-queries to identify which one of these corresponds to the user's desire to be in a particular role for that moment. After identifying the one DSP sub-query, the "Context-Processor" would strip out the remaining DSP sub-queries which are redundant to the user's request. There may be situations where more than one context attribute would need to be processed. In this manner, the "Context-Processor" would process all the captured context attributes to generate a context-sensitive SQL that would then be processed in the normal way. The context-sensitive SQL thus generated would have all the data security redundancies removed and will have additional predicates added to the DSP sub-queries. This action addresses the problem of SQL size and complexity and consequently the CBO's ability to generate optimized and stable execution plans, thereby resulting in better performance of the SQL. In short, the "Context-Processor" in the security framework processes the context attributes and applies these intelligently on the DSP sub-queries to produce context-sensitive SQLs. In the case of enterprise business applications, the context-processor could be implemented by way of writing code using the following generic algorithm:

Start Generate Context-Sensitive DSP Sub-Query

Get captured UI Context Attributes

Retrieve RBAC Data Security Predicates Sub Queries for User

Iterate Until all UI Context Attributes Processed

Iterate Until all DSP Sub-Queries Processed

Validate DSP Sub-Query Applicability

If DSP Sub-Query Applicable to Current Context

Then Keep DSP Sub-Query

Else Remove DSP Sub-Query

End if

Validate Context-Attribute Applicability for Kept DSP Sub-Query

If Attribute Applicable

Apply Context Attribute to DSP Sub-Query

End if

End Iterate

End Iterate

Dispatch Context-Sensitive DSP Sub-Queries for Being Applied to Main SQL

End Generate Context-Sensitive DSP Sub-Query

The final context-sensitive SQL thus generated is simpler, smaller in size and has much better performance as compared to the existing SQL.

C. Example of SQL with Context-Sensitive Data Security

For example, let's say one of the context attributes captured by the "context-capture" process and passed to the "context-processor" was the creation date of an object that relates to a column in a table in the database and the user is interested in a row set that corresponds to a specific value or a range of values for creation date. Also, say the user wishes to see only the records that are directly owned by him rather than all the records to which the user has access to via the union all of RBAC sub-queries. The "Context-Processor" in this case would create context sensitive security by doing two things. First, it will parse all the DSP sub-queries to check where all this creation date is applicable and add it as an additional filter predicate to the affected DSP sub-queries. Second, based on the context, it will strip out the DSP sub-queries that are not required but that are added by the RBAC security framework.

Such a representative use case from Oracle's Fusion CRM Application is shown below. In the example, the desired end result is for a sales manager to see details of Leads where he/she is the owner as reflected in column OWNER_ID in the MKL_LM_LEADS table. The user chooses to limit the search based on the Lead's CREATION_DATE, also from the same MKL_LM_LEADS table. The SQLs before and after context-processing are shown below.

Existing "My Leads" Query:

SELECT

MklLeadEO.LEAD_NAME,

<Column names>

FROM

MKL_LM_LEADS MklLeadEO,

<Other table names>

WHERE

<Joins and predicates>

/ Main Filtering Criteria in Outer SQL that defines My Leads*/*

AND ((((MklLeadEO.OWNER_ID = :BindLoggedInUserResourceId)))

/ the secondary restrictive filter predicate CREATION_DATE */*

AND ((MklLeadEO.CREATION_DATE BETWEEN :BindLowerDateRange AND :BindUpperDateRange))))

AND (:SysEffectiveDateBindVar BETWEEN OrganizationDEO.EFFECTIVE_START_DATE(+) AND OrganizationDEO.EFFECTIVE_END_DATE(+))

AND
(OrganizationDEO.EFFECTIVE_LATEST_CHANGE(+) = 'Y')

/ Union of All OR'ed DSP Sub-Queries Appended by RBAC Security Framework at run time */*

AND

((MklLeadEO.lead_id IN

(SELECT lead_id

FROM MKL_LM_LEADS

WHERE owner_id =

(SELECT HZ_SESSION_UTIL.GET_USER_PARTYID FROM dual)

OR MklLeadEO.lead_id IN DSP-SubQuery2

OR MklLeadEO.lead_id IN DSP-SubQuery3

OR MklLeadEO.lead_id IN DSP-SubQuery4

OR MklLeadEO.lead_id IN DSP-SubQuery5

OR MklLeadEO.lead_id IN DSP-SubQuery6

OR MklLeadEO.lead_id IN DSP-SubQuery7

OR MklLeadEO.lead_id IN DSP-SubQuery8

OR MklLeadEO.lead_id IN DSP-SubQuery9)

Context-Sensitive "My Leads" Query After Context Processing:

SELECT

MklLeadEO.LEAD_NAME,

<column names>

FROM

MKL_LM_LEADS MklLeadEO,

<Other table names>

WHERE

<Joins and predicates>

(:SysEffectiveDateBindVar BETWEEN OrganizationDEO.EFFECTIVE_START_DATE(+) AND OrganizationDEO.EFFECTIVE_END_DATE(+))

AND
(OrganizationDEO.EFFECTIVE_LATEST_CHANGE(+) = 'Y')

/ The context-processor retains only the contextually applicable DSP sub-query, stripping out the remaining ones. In addition, the context predicate CREATION_DATE is added as an additional filter predicate to the retained DSP sub-query */*

AND (MklLeadEO.lead_id IN

(SELECT lead_id

FROM MKL_LM_LEADS

WHERE owner_id =

(SELECT HZ_SESSION_UTIL.GET_USER_PARTYID FROM dual)

AND CREATION_DATE BETWEEN :BindLowerDateRange AND :BindUpperDateRange)

IV. Experimental Results Using Oracle's Fusion Applications

We applied this solution to Oracle's Fusion Applications that are built using the Oracle Application Development Framework (ADF). For the benchmarking experiments to establish the performance gains of context-sensitive data security, we used the Lead Management and Opportunity Management modules of Fusion CRM application [18] against an Oracle 11gR2 database with very promising results. We saw very significant improvements in SQLs' response time (RT), buffer gets, hard parse time and shared memory utilization, the four commonly used parameters for measuring the performance of SQL queries. The simplification of the DSP sub-queries also helped improve the hard parse time of the context-sensitive SQLs. When the context-sensitive DSPs were implemented, we recorded repeatable, significant gains in not only individual SQL performance but also at the database resources level. A sampling of these gains for SQL performance is highlighted below for Lead Management module in Figure 2 for five use cases benchmarked for one user.

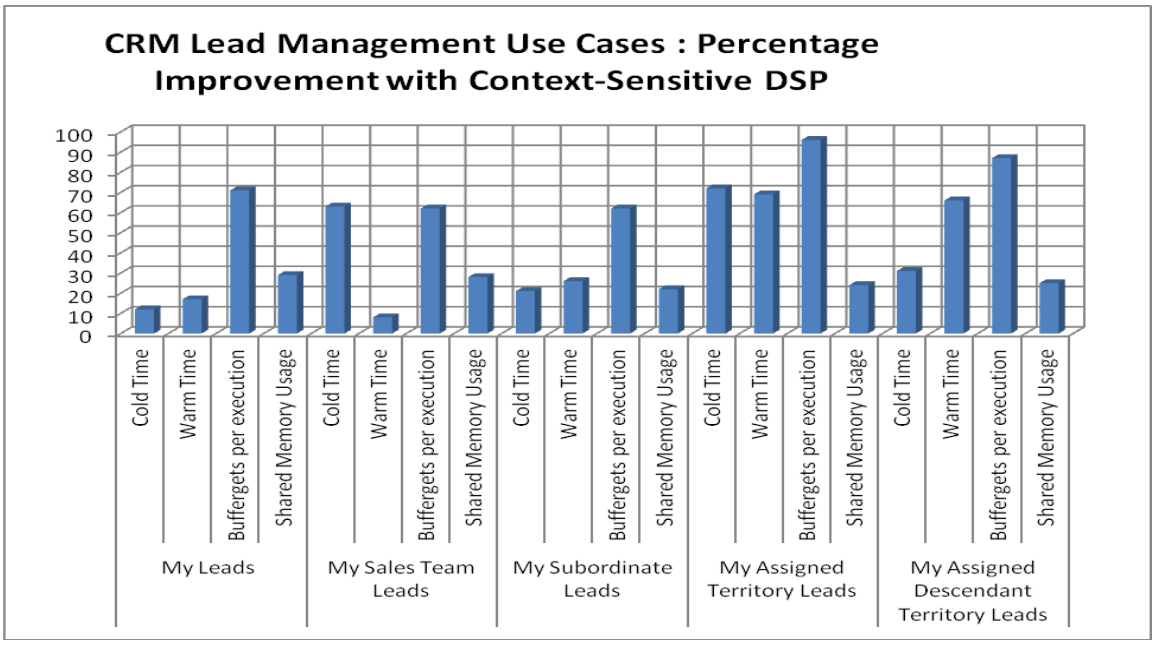

Figure2. Performance Improvements for Lead Management SQLs of Oracle Fusion CRM Application

Similar improvements were recorded in the benchmarking of five use cases for the Opportunity Management module of Oracle Fusion CRM Application as shown in Figure 3.

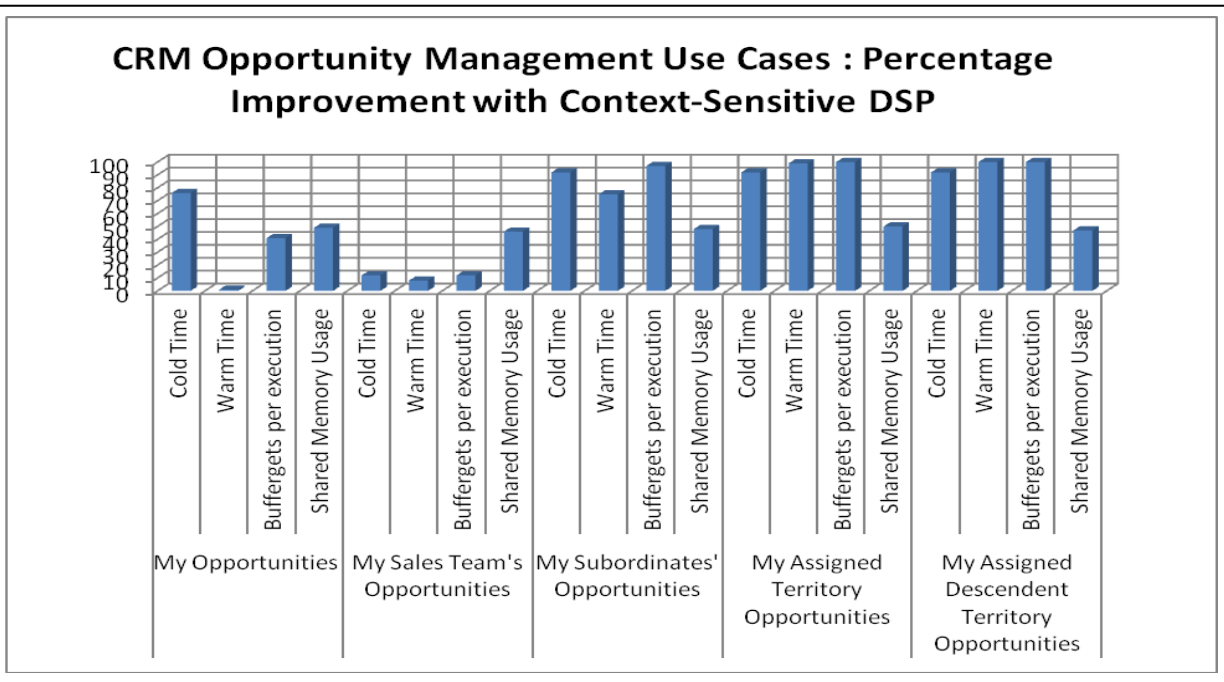

Figure3. Performance Improvements for Opportunity Management SQLs of Oracle Fusion CRM Application

V. CONCLUSION

RBAC based data security has been established as an industry standard practice for many types of applications including enterprise business applications. However, the impacts of

RBAC based data security on performance and scalability has not been the focus of much research. In this paper we have researched, documented and presented the serious performance issues caused by RBAC based DSP sub-queries where a user has been assigned multiple roles and responsibilities. The current methodology of applying all DSP sub-queries bound together with OR conditions without regard to the context in which a user is executing the query poses a serious performance problem and can be a risk to the success of enterprise business applications. we have proposed a solution that takes into account all context attributes to process and produce "context-sensitive" SQL queries that dramatically improve the performance of such SQLs. we have benchmarked many uses cases in Oracle Fusion Applications against an Oracle database. All the use cases showed significant performance gains in terms of Response Time (RT), hard parse time, buffer gets as well as shared memory. Adoption of the proposed solution can thus provide significant performance and scalability improvements in enterprise business applications.

REFERENCES

[1] National Institue of Standards and Technology. Role based access control accessed on the World Wide Web Jan 2013 at http://csrc.nist.gov/groups/SNS/rbac/faq.html

[2] Sandhu, R., Ferraiolo, D. and Kuhn R. 2004. "American National Standard for Information Technology – Role Based Access Control", ANSI INCITS 359-2004, February 3, 2004.

[3] American National Standards Institute Standards for RBAC accessed on World Wide Web at http://www.incits.org/INCITS_Published_Standards.pdf Jun 2013.

[4] D.F. Ferraiolo and D.R. Kuhn (1992) "Role Based Access Control" 15th National Computer Security Conf. Oct 13-16, 1992, pp. 554-563.

[5] D.F. Ferraiolo, D.R. Kuhn, R. Chandramouli, Role Based Access Control (book), Artech House, 2003, 2nd Edition, 2007.

[6] D.R. Kuhn, "Mutual Exclusion of Roles as a Means of Implementing Separation of Duty in Role-Based Access Control Systems" Second ACM Workshop on Role-Based Access Control, 1997.

[7] R. Chandramouli, R. Sandhu, "Role Based Access Control Features in Commercial Database Management Systems", 21st National Information Systems Security Conference, October 6-9, 1998, Crystal City, Virginia.

[8] S. Gavrila, J. Barkley, "Formal Specification for Role Based Access Control User/Role and Role/Role Relationship Management" (1998), Third ACM Workshop on Role-Based Access Control.

[9] W.A. Jansen, "Inheritance Properties of Role Hierarchies," 21st National Information Systems Security Conference, October 6-9, 1998, Crystal City, Virginia.

[10] R. Chandramouli, "Business Process Driven Framework for defining an Access Control Service based on Roles and Rules", 23rd National Information Systems Security Conference, 2000.

[11] RS Sandhu, EJ Coyne, HL Feinstein, CE Youman - Computer, 1996 - ieeexplore.ieee.org accessed on World Wide Web at http://csrc.nist.gov/rbac/sandhu96.pdf

[12] Oracle® Fusion Applications Security Guide 11g Release 1 (11.1.4) accessed on the World Wide Web at http://docs.oracle.com/cd/E28271_01/fusionapps.1111/e16689/F323388AN16D1F.htm

[13] Rosemann, M., & Recker, J. (2006). "Context-aware process design: Exploring the extrinsic drivers for process flexibility". In T. Latour & M. Petit. 18th international conference on advanced information systems engineering. Proceedings of workshops and doctoral consortium. Luxembourg: Namur University Press. pp. 149–158.

[14] Kaltz, J.W., Ziegler, J., Lohmann, S. (2005). "Context-aware Web Engineering: Modeling and Applications" (PDF). Revue d'Intelligence Artificielle 19 (3): 439–458. doi:10.3166/ria.19.439-458

[15] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In IEEE Workshop on Mobile Computing Systems and Applications, pages 85–90, Santa Cruz, CA, US, 1994.

[16] Kulkarni, Devdatta, and Anand Tripathi. "Context-aware role-based access control in pervasive computing systems." Proceedings of the 13th ACM symposium on Access control models and technologies. ACM, 2008.

[17] Oracle Application Development Framework accessed on the World Wide Web at http://www.oracle.com/technetwork/developer-tools/adf/overview/index.html

[18] Oracle Fusion CRM Application accessed on the World Wide Web at http://www.oracle.com/us/products/applications/fusion/customer-relationship-management/index.html.