Performance Tuning of Database Systems Using a Context-aware Approach

Asanga Nimalasena
Vladimir Getov

Faculty of Science and Technology, University of Westminster

This is a copy of the author's accepted version of a paper subsequently published in the proceedings of the 9th International Conference on Computer Engineering & Systems (ICCES), pp.98-103, 22-23 Dec. 2014. ISBN 9781479965939. It is available online at:

http://dx.doi.org/10.1109/ICCES.2014.7030936

Performance Tuning of Database Systems Using a Context-aware Approach

Asanga Nimalasena and Vladimir Getov
Faculty of Science and Technology
University of Westminster
London, United Kingdom
Asanga.Nimalasena@my.westminster.ac.uk, V.S.Getov@westminster.ac.uk

***Abstract*—Database system performance problems have a cascading effect into all aspects of an enterprise application. Database vendors and application developers provide guidelines, best practices and even initial database settings for good performance. But database performance tuning is not a one-off task. Database administrators have to keep a constant eye on the database performance as the tuning work carried out earlier could be invalidated due to multitude of reasons. Before engaging in a performance tuning endeavor, a database administrator must prioritize which tuning tasks to carry out first. This prioritization is done based on which tuning action would yield highest performance benefit. However, this prediction may not always be accurate. Experiment-based performance tuning methodologies have been introduced as an alternative to prediction-based performance tuning approaches. Experimenting on a representative system similar to the production one allows a database administrator to accurately gauge the performance gain for a particular tuning task. In this paper we propose a novel approach to experiment-based performance tuning with the use of a context-aware application model. Using a proof-of-concept implementation we show how it could be used to automate the detection of performance changes, experiment creation and evaluate the performance tuning outcomes for mixed workload types through database configuration parameter changes.**

Keywords—performance tuning; context-aware models; database systems

I. INTRODUCTION

Database systems (DBS) are at the heart of any enterprise application. Performance problems in the DBS cascade into all aspects of an enterprise application. Database vendors and application vendors provide guidelines, best practices and even initial configuration settings for the DBS to start off with good performance [1]. But it is the responsibility of the database administrator (DBA) to fine tune the DBS to perform optimally which the general guidelines or initial settings may not do. A properly tuned DBS is more likely to enable a DBA to achieve the service level agreements (SLA) than a non-tuned DBS. However database performance tuning is not a one-off task. DBA have to keep a constant eye on the DBS performance as multitude of reasons would cause the tuning work carried out earlier to be invalidated. Change in hardware or faulty hardware (i.e. failed memory banks reducing system available memory, network bandwidth reduction, failed I/O controllers reducing I/O bandwidth) related issues could easily be detected with modern monitoring systems and replacing the component with an exact copy will rectify the issue. But there are other factors that pose much more subtle performance problems. This type of changes include database upgrades (i.e. change in optimizer behaviour), change in configuration parameters (i.e. depreciated configuration settings or change in a default value set at installation), change in workloads and business requirements (i.e. a DBS system used for online transaction processing (OLTP) now being used for transaction processing and decision support system (DSS)), stale statistics on data or a combination of these factors. As a result the database query performance could regress, remain unaffected or improve [2]. The latter two cases would not be of any concern for a DBA but any performance regression must be rectified. Prior to starting performance tuning activities the DBA prioritizes which performance tuning tasks to carry out first. This prioritization is done based on a prediction DBA makes as to which tuning action would yield the highest performance benefit with the lowest cost. The cost could be in the form of monetary loses due to regressed performance as well as time limit imposed by SLA to resolve a performance issue. The DBA uses her knowledge of the DBS, past experiences of handling similar situation and even intuition to make this prioritization. However, this prediction may not always be accurate thus resulting in elongated performance resolution time.

As there are magnitudes of factors to consider it is impossible for a DBA to predict the performance changes without actually trying a workload on the system [3]. Experiment-based performance tuning methodologies have been introduced as an alternative to prediction-based performance tuning methods. In such cases the DBA would create a replica of the production database, run a similar workload to recreate the problem behaviour and then hypothesizes the root cause and set of potential solutions and evaluate them one by one with different sets of experiments [4, 5]. Experimenting on a representative system or on the production system itself allows a database administrator to accurately gauge the performance gain for a particular tuning task before applying it to production DBS.

There are similarities between how a context-aware system reacts to a context change and how a DBA employs an experiment-based performance tuning model. A context is defined by Dey [6] as “any information that can be used to

characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". Self-adapting context systems have three main aspects, context sensing, actions or actuators and self-learning/self-adaptation based on action outcomes. These aspects could be transposed to experiment-based DBS performance tuning as follows. DBA has to monitor the DBS to determine if the performance has regressed (i.e. context sensing). Performance regression is detected by way of increase query execution time, high user response time, change in the CPU pattern, memory usage patterns, etc. (i.e. context inference). DBA would carry out a tuning action depending on the symptoms. If the DBA is aware of the reason for the current performance change this would be a known context, as there are many factors that could affect the database performance, DBA may not always know the root cause outright (i.e. unknown context). In an unknown context DBA would engage in the multiple tuning experiments (i.e. actions in the context-aware model), evaluate the outcome of them all and then would decide the most beneficial tuning activity to apply on the production DBS. Once a fix is found the DBA would be knowledgeable to detect similar situations in the future and fix them quickly, the same way context-aware system uses self-adaptation and evolves to recognize more and more contexts.

In this paper we propose a novel approach to experiment-based performance tuning with the use of a context-aware application model. Using our proof-of-concept implementation we showcase how a context-aware system could be used to automate the detection of performance changes, to create appropriate experiments and then to evaluate the performance tuning outcomes for mixed workload types through database configuration parameter changes.

The rest of this paper is organized as follows. Section II reviews related work on experiment-based database performance tuning models. Section III gives a description of our novel approach while Section IV presents the formal modelling of it used for the proof-of-concept development. Section V describes the implementation of the proof-of-concept system. Section VI presents experimental results of the evaluation. Finally, the paper concludes with Section VII giving a summary of findings from the evaluation and directions for future work.

II. RELATED WORK

Experiment-based database performance tuning models are actively developed and researched into by both the academic initiatives and by database vendors. Among the related work two types of experiment-based database performance models could be identified. The first model type is where experimentation takes place offline or offloaded from the production system while the second model runs experimentation on the production system itself in real-time. During a performance crunch configuration parameter tuning could give the quickest fix with minimum effort. After this initial breather DBA could further investigate the root cause of the performance regress and provide a more substantiated solution. But modern databases have hundreds of parameters and deciding which parameter to tune and what value to set for it require careful testing. Wrong parameter or parameter value could worsen an already bad situation. Database vendor literature provides insight into the parameters a DBA is most likely to use for tuning the workload [7, 8, 9]. This vendor literature could be considered as guidelines but may need to be tested and validated for each individual application.

A framework proposed in [10] provides a DBS independent experiment base approach to database tuning via configuration parameter settings. With the use of adaptive sampling it identifies and brings high impact and high performance configuration parameters into the experiment workbench. Using a cycle-stealing paradigm it executes experiments directly on the production database for accurate gauging of benefits to configuration parameters changes. Another configuration parameter related framework is presented on [4] where starting off with a small number of experiments the experiment base is expanded based on estimated benefits from each experiments. A vendor specific experiment base performance tuning tool is described in [3]. An SQL Performance Analyzer (SPA) was introduced by Oracle in the 11g database version and allows the DBA to measure performance changes or impact of configuration parameter changes, database version upgrades, updating statistics, and creation of database objects such as indexes and materialize views. SPA could be used on the production database or a SQL set from production could be transferred to a test system to be used with SPA. However SPA does not provide any way to automate the testing and requires the DBA to execute each experiment. A more robust workload capture and replay system is presented in [11, 12]. DBA could capture production workload with minimum overhead and use the captured workload to conduct experiments on a test system (i.e. offline) to the same level of concurrency as the production system. Database replay works for proactive performance tuning but is not sufficient for reactive performance tuning as workload capture has to happen before the problem period.

An offline experiment-based performance model is presented in [13] which uses neural network to train to recognize the workload patterns on a test system before letting it predict performance on the production database. As it requires training of neural network beforehand dynamic adaptation to performance workloads that were not in the training set could be unpredictable. As neural network internal weight adjustments are opaque to a DBA, validation of the prediction would require a DBA to carry out further tests. If tuning cannot be achieved through configuration parameter changes then next option is for a DBA to tune the relevant queries to get better performance. To successfully tune queries a DBA has to be aware of the characteristics of the data being queried, statistics and the business requirements. In this regard an experiment-based SQL tuning framework is presented in [5]. Using cardinality sets the framework quickly develops new plans that are of the same neighbourhood but with better execution plans. The approach used by Oracle [2, 14] is based on an initial SQL plan which is considered the baseline, while other plans are generated and compared against this one. If better plans are found then they are added to the baseline after verification allowing query performance to always improve

and never regress beyond the baseline. Oracle has automated this tuning process with the use of automatic tuning optimizer [15] in 11g and with automatic reoptimization techniques in 12c [16]. A rapid experiment-defining framework and a high level language that enable the DBA to define experiments using the framework is proposed in [17]. The framework orchestrates the scheduling, running and tuning of the system in an automated fashion. DBA is free to devise experiments to check impact of configuration changes.

III. EXPERIMENT-BASED PERFORMANCE TUNING USING CONTEXT-AWARE FRAMEWORK

Having considered many different context-aware models [18, 19, 20, 21, 22] context-aware framework in [23] was adopted for the experiment-based database performance tuning for the following reasons. It has three loosely coupled independent systems with clear distinct roles which enables DBS independence for the implementation. Its concurrent action execution model provides a non-linear model for experimentation and finally the goal specification and action refinement reduce the number of elements in the eligible set of experiments. Fig.1 shows the adoption of the context-aware framework in [23] for the experiment-based performance tuning. Following the definition in [6] the DBS is nominated as the entity that is of concern and the user workload *W* and the resource usage *R* is defined as the contexts describing the state of the DBS (i.e. context space). At any given time the workload *W* could consists of zero or more user workload types. The resource usage *R* describes level of consumption of system resources by the DBS at a given point in time. The resource types include CPU, memory and DBS memory components such as buffer pools, I/O utilization, network utilization and etc. The context-aware system senses the context space for any change in context. It is typical in enterprise scenarios that usage pattern remains constant over time as the application uses a similar set of queries [13, 24]. Therefore any change to workload type or resource usage pattern is considered a context change and context inference is carried out.

The context inference is carried out by the inference system which consists of a knowledge base. When a context change is encountered the knowledge base is queried to identify if the new context values are known. If the inference system is unable to find the new context values in the knowledge base then the action system is invoked.

The action system is responsible for concurrent action execution (i.e. concurrent experimentation) and evaluation when an unknown context is encountered. The goals of the action system are to reduce the number of required actions (experiments) and to complete the actions in a single pass (non-linear) as opposed to iterative manner. The action system uses goal specification and experiment refinement to reduce the number of experiments to conduct. The goal specification defines the extremities of the parameters used in the experiments. Using the goal specification the experiment refinement limits what experiment qualifies to be in the action space (set of experiments to evaluate). The experiment limiting process starts by identifying the context that is closest to the unknown context from the knowledge base. The closeness is measured by the difference of the context values. If more than one context is found to be the closest then the priority of each context is considered. Context setting of this known context is used to devise the initial experiment. Other experiments are then derived from this initial experiment. Experiments are then executed concurrently in a private workbench. The private workbench ensures that experiment setting is opaque to the DBS and does not affect its current state. As all actions are executed concurrently the outcome of each action is known at the same time which gives a non-linear experiment time. The final phase of the action system is the experiment outcome evaluation. For the database performance tuning actions are evaluated based on a minimizing function. The best tuning action is the one that resulted in minimum resource usage such as the elapsed time or CPU usage for the execution of queries, number of data blocks accessed or combination of metrics as used by [3]. This outcome is then used to update the knowledge base so it learns of the unknown context and is also used to adapt the DBS to best suit the current context.

IV. PROOF OF CONCEPT MODELING

A proof-of-concept case was devised where the experiment-based performance tuning is carried out by way of database configuration changes. The experiments consist of executing a representative workload using different set of configuration values for each experiment.

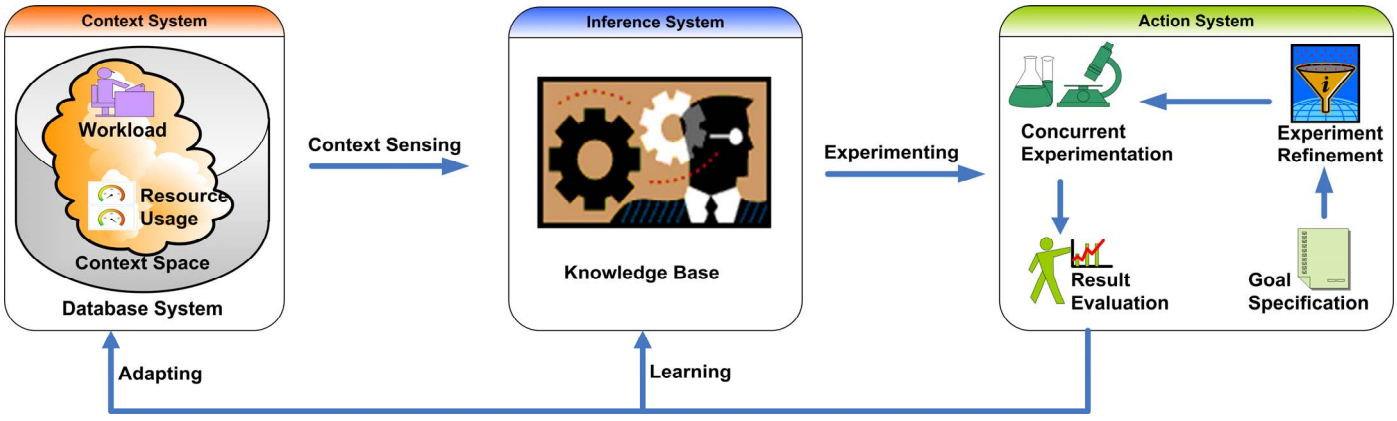

Fig. 1. Context-aware framework for experiment-based database performance tuning

Representative workload is created by capturing the top SQL for each workload type. Formal modelling of the context-aware framework for our proof-of-concept case was as follows. Two workload types were defined for $W=\{OLTP,DSS\}$. Online transaction processing (OLTP) workload types have the characteristics of accessing a small percentage of tuples on few tables for a given query. The Decision Support System (DSS) workload types are ad-hoc or reporting type of queries that access large percentage of tuples in multiple tables. As resource usage context the CPU load for each workload type was considered $R=\{CPU_{OLTP}, CPU_{DSS}\}$.

It's not uncommon for both these workload types to be present in the same DBS [25]. The knowledge base was modelled using web ontology language (OWL). The objective was not to create a full ontology to represent the context space but to have an OWL representation of context [26] that would allow leveraging of OWL features for context inference. With the use of the OWL only the concrete workload types are needed to be defined, in this case the OLTP and DSS workload types. Using the inherent inference capabilities of OWL it is possible to derive other workload types. A mixed workload is specified using OLTP and DSS concrete types as

```
<owl:Class rdf:about="#Mix">
    <owl:disjointWith rdf:resource="#DSS"/>
    <owl:disjointWith rdf:resource="#OLTP"/>
    <rdfs:subClassOf>
      <owl:intersectionOf rdf="Collection">
        <owl:complementOf rdf:resource="#DSS"/>
        <owl:complementOf rdf:resource="#OLTP"/>
       </owl:intersectionOf>
    <rdfs:subClassOf
 </owl:Class>
```

A generic workload type called "Workload" is defined using all three aforementioned OWL workload types (OLTP, DSS, MIX). Because of this generic workload type domain or application specific workload types (e.g. Order search workload, sales report workload and etc) could be inferred upon and deduced to one of the three aforementioned workload types thus making implementation free of any domain specific constructs.

The goal specifications denoted as G_{lo} and G_{hi} are considered elements of the configuration parameter space. The initial experiment devised with the known context configuration is denoted as E_k. With these definitions in place the total number of experiments defined as a function of the configuration parameter is modelled as a union of three sets.

$$\text{Experiment space} = \{ \; E_k(configuration_k) \cup E_p(configuration_p) \cup E_q(configuration_q) \mid p = \{1 .. n\}, n > 0, q = \{1 .. m\}, m > 0, configuration_k - p\Delta \geq G_{lo}, configuration_k + q\Delta \leq G_{hi}, \Delta > 0 \}$$

The lower bound expansion range is denoted by p which specifies the number of actions to define in the direction of G_{lo}. The upper bound expansion range denoted by q specifies the number of actions to define in the direction of G_{hi} and finally the distance between each configuration parameter is denoted by Δ. Values for p, q and Δ are derived from DBA's knowledge and experience. The best configuration parameter for the unknown context is the experiment which resulted in minimum resource usage. For the proof-of-concept case each experiment's CPU usage, DB Time [17] was compared to find the best configuration parameter setting. This could be formally defined as

$$configuration_{best} = \{ \forall configuration_i \in \{experiment\ space\ configurations\} \; \exists E_i(configuration_i): minimum(Resource\ Usage(E_i)) \}$$

V. PROOF OF CONCEPT IMPLEMENTATION

The context-aware system was developed as a java desktop application with a graphical interface. Fig.2 shows the experimental setup. The context-aware application ran on a desktop computer with 8GB RAM, 2.8GHz Intel dual core processor and using a 1.7 java virtual machine (JVM) on Windows 7. The chosen DBS system was Oracle 12.1 which ran on a server with 12GB RAM, 2.0GHz Intel quad core processor and 500GB SAS disks running on RedHat Linux 5.5. Workload was generated using a load injector server which had the specifications of 16GB RAM, 2.4GHz Intel quad core processor, and 500GB SAS disk running RedHat Linux 6.4 All machines were connected via LAN. Workloads were generated against two of Oracle database's sample schemas namely OE (order entry) and SH (sales history) [27]. The query to OE schema was used to simulate OLTP workload (i.e. searching order records) type queries while SH schema was used for simulating DSS workload type. Two services were created in the database for each of the workload types thus enabling to gauge the resource usage of each workload type based on the service metrics. Knowledge base was modelled using java implementation of Protégé OWL API and for inference java API for Protégé Pellet Reasoner was used.

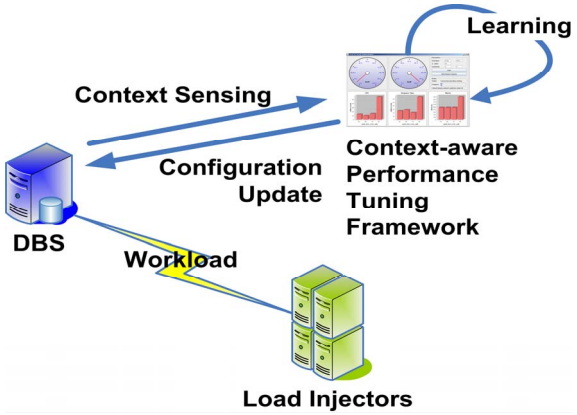

Fig. 2. Proof-of-concept case setup

The knowledge base was initially populated with context facts one relating to OLTP only workloads, another DSS only workloads and one for mix workload type where CPU usage was distributed (90:10) among OLTP and DSS. These values were not validated and they are used to represent DBA setting initial configurations based on experience and intuition.

The configuration parameter optimizer_index_cost_adj [28] was chosen as the parameter to demonstrate the proof of concept. This parameter made a good candidate to demonstrate the proof of concept. It has a wider range (1-10000) but for an enterprise application the applicable range is much narrower which is the concept demonstrated by the goal specification. It is directly applicable to the action refinement model in section IV without requiring any modification. Most importantly the parameter affects the query execution plans thus enabling query performance comparison under each configuration setting. This parameter could be set on two levels, session and system. Session level settings are opaque to other database sessions. As such session level settings provide a private workbench to carry out the concurrent experimentation without affecting other database sessions. There are no clear guidelines to set this parameter in practice. Rule of thumb is to set it to lower values for OLTP workload to make queries bias for index access and higher values for DSS in favour of full table scans. But this rule of thumb may not work for every application. Although optimizer_index_cost_adj is used here any configuration parameter could have been used with the context-aware model.

VI. EXPERIMENTAL RESULTS

Three different workloads were created for the OLTP and DSS workload types ($OLTP_1$, DSS_1, DSS_2) and injected to and removed from the DBS at regular interval. Table I shows the workload mix on the DBS at various times corresponding to Fig. 3 (a) and (b). Oracle's enterprise manager (OEM) was used to monitor the CPU usage (active session = CPU used by all non-idle DBS session / wall clock time) on the DBS during the testing. There were no other workloads on the test DBS except for the experimental workloads generated. The CPU usage pattern is given on Fig.3 where (a) represent the usage pattern when the DBS configuration is optimized only for OLTP while (b) represent when context-aware application is able to experiment and update the configuration parameter to best fit the workload mix. In Fig.3 (b) during the time period of (T7 – T8) there is spike in the CPU usage compared to (a). This spike is due to context-aware system detecting the context change due to injection of DSS_1 workload and carrying out the concurrent experimentation. Once the DBS has adapted comparative CPU usage for the same workload (T8 – T9) is less compared to without adaptation (T2 – T3). Injection of DSS_2 also causes a context change and concurrent experimentation to take place ($T_9 – T_{10}$) but since the DBS is already optimized for a DSS workload the spike in CPU usage is less compared to previous adaptation. But overall CPU usage is less ($T_{10} – T_{11}$) compared to without adaptation ($T_2 – T_3$).

TABLE I

Time	Workload Mix
T_0, T_6	$OLTP_1$
T_1, T_7	$OLTP_1$, DSS_1
T_2, T_9	$OLTP_1$, DSS_1, DSS_2
T_3, T_{11}	$OLTP_1$, DSS_2
T_4, T_{12}	$OLTP_1$

These observations could be validated by examining the result evaluation of the concurrent experiments. The system started with OLTP only workload type which is a known context based on the initial facts the knowledge base is populated with, where configuration value of 100 was considered the best configuration. When the DSS workload is injected into the DBS the CPU usage pattern gets changed and context inference did not yield a known context. Fig. 4 shows the results from the concurrent execution of experiments under various configurations settings, which shows that setting the configuration parameter to 70 results in lowest CPU usage to execute the current workloads. Fig.5 comparison of DB Time also reaffirms the evaluation. For the mix workload the result evaluation uses a weighted average to calculate the CPU and DB Time usage for each experimental workload.

Fig. 3. (a) CPU usage without experiment-based adaptation. (b) CPU usage with context-aware experiment-based adaptation

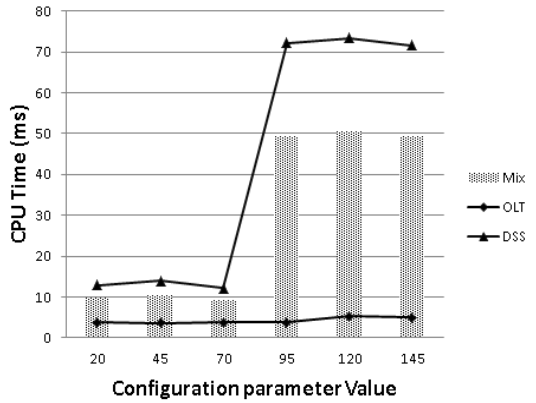

Fig. 4. CPU usage from experiment outcome evaluation

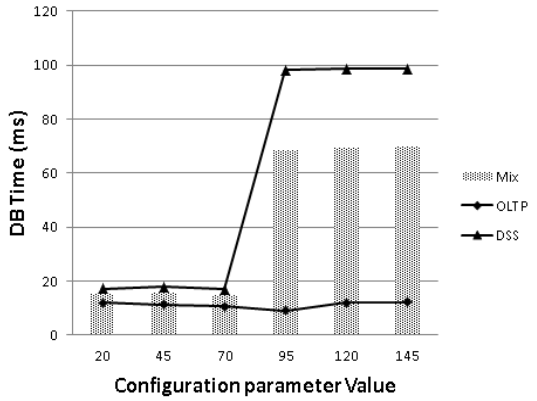

Fig. 5. DB Time comparison from experiment outcome evaluation

VII. CONCLUSION

This paper presented a novel approach to experiment-based database performance tuning using a context-aware approach. A context-aware model was presented and a proof-of-concept implementation was carried out to test the feasibility of the proposed framework. The experimental results have shown that with the use of goal specification and action refinement together with the concurrent action execution and evaluation the implemented system enables DBS to adapt to changing workload types and resource usage patterns. As a result of this work we hope to introduce a new paradigm of context-aware database performance tuning where instead of one setting fits all, the DBS update their settings to best suit the current workloads and assist DBA in performance tuning tasks. As future work we wish to use the context-aware implementation for experiment-based performance tuning of a database application using TCP-C and TPC-H benchmarking workloads.

REFERENCES

[1] P. Belknap, J. Beresniewicz, B. Dageville, K. Dias, U. Shaft, and K. Yagoub, "A decade of Oracle database manageability", IEEE Data Eng. Bull., 2011, pp.20-27.

[2] M. Ziauddin, D. Das, H. Su, Y. Zhu, K. Yagoub, "Optimizer plan change management: improved stability and performance in Oracle 11g", in VLDB '08, 2008.

[3] K. Yagoub, P. Belknap, et al. "Oracle's SQL performance analyzer", IEEE Data Engineering Bulletin, 31(1), 2008.

[4] S. Babu, N. Borisov, S. Duan, H. Herodotou, and V. Thummala.. "Automated experiment-driven management of (database) systems". In *Proceedings of the 12th conference on Hot topics in operating systems* (HotOS'09), 2009.

[5] H. Herodotou and S. Babu. "Automated SQL tuning through trial and (sometimes) error". In *Proceedings of the Second International Workshop on Testing Database Systems* (DBTest '09), 2009.

[6] A.K. Dey, "Understanding and Using Context", *Personal and Ubiquitous Computing,* vol. 5, no. 1, 2001, pp. 4-7.

[7] Oracle Corp, "Oracle database performance tuning guide", http://bit.ly/1uGeuCw, last accessed 2014/07/11.

[8] Postgresql performance optimization. http://bit.ly/1oU9sy4, last accessed 2014/07/11.

[9] MySQL Reference Manual. Optimization http://bit.ly/1mZOPnB, last accessed 2014/07/11.

[10] S.Duan, V.Thummala and S. Babu, "Tuning database configuration parameters with iTuned", Proceedings of the VLDB Endowment, v.2 n.1, 1246-1257, 2009.

[11] L. Galanis et al., "Oracle database replay", SIGMOD Conference Pg. 1159-1170, 2008.

[12] R. Colle, L. Galanis, S. Buranawatanachoke, S. Papadomanolakis, Y. Wang, "Oracle database replay." Proceedings of the VLDB Endowment 2, 1542–1545, 2009.

[13] A.A. Khattab, A. Algergawy and A. Sarhan, "NNMonitor: performance modeling for database servers", *Computer Engineering & Systems (ICCES), 2013 8th International Conference*, pp.301,306, 26-28, 2013.

[14] Oracle Corp., white paepr "SQL plan management in Oracle Ddatabase 11g", http://bit.ly/1nHPllB, last accessed 2014/07/11.

[15] P.Belknap, B.Dageville, K.Dias,K. Yagoub, "Self-Tuning for SQL performance in Oracle database 11g", Data Engineering, pp.1694,1700, 2009.

[16] Oracle Corp., white paepr , "Optimizer with Oracle databsae 12c", http://bit.ly/1mP0VkV, last accessed 2014/07/14.

[17] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, and G. Wood, "Automatic performance diagnosis and tuning in Oracle.", In Proc. Conf. on Innovative Data Systems Research (CIDR), 2005.

[18] T. Cioara et al., "A self-adapting algorithm for context aware systems", Proc. 9th Roedunet Int. Conference (RoEduNet), pp. 374-379, 2010.

[19] S.W. Loke, "Incremental awareness and compositionality: A design philosophy for context-aware pervasive systems", Pervasive and Mobile Computing. 6 (2), 239-253, 2010.

[20] N. O'Connor, R. Cunningham and V. Cahill, "Self-adapting context definition", Proc. 1st Int. Conf. Self-Adaptive and Self-Organizing Systems (SASO '07), pp. 336-339, 2007.

[21] N. Nwiabu, I. Allison, P. Holt, P. Lowit and B. Oyeneyin, "Situation awareness in context-aware case-based decision support", IEEE 1st Int. Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), pp. 9-16, 2011.

[22] K. Kwang-Eun and S. Kwee-Bo, "Development of context aware system based on Bayesian network driven context reasoning method and ontology context modeling", Proc. Int. Conf. Control, Automation and Systems (ICCAS), pp. 2309-2313, 2008.

[23] A. Nimalasena and V. Getov, "System evolution for unknown context through multi-action evaluation", Computer Software and Applications Conference Workshops, 2013 IEEE 37th Annual, pp.271,276, 2013.

[24] M.Holze. "Self-management concepts for relational database systems", Ph.D Thesis, University of Hamburg, 2012. http://bit.ly/1iWAecS, last accessed 2014/07/14.

[25] Oracle Corp., "In-Memory acceleration for the real-time enterprise", http://bit.ly/1joMmgY, last accessed 2014/07/14.

[26] D. Ejigu, M. Scuturici and L. Brunie, "Semantic approach to context management and reasoning in ubiquitous context-aware systems", in Proceedings of ICDIM, 2007.

[27] Oracle Corp., "Database sample schemas", http://bit.ly/1mg67Mm, last accessed 2014/07/14.

[28] Oracle Corp., "Database reference 12c release 1". http://bit.ly/1wHMJJV, last accessed 2014/07/14.