Context-Aware Framework for Performance Tuning via Multi-action Evaluation

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 IEEE 39th Annual Computer Software and Applications Conference (COMPSAC), vol.3, pp.318-323, 1-5 July 2015. ISBN 9781467365635. It is available online at:

http://dx.doi.org/10.1109/COMPSAC.2015.156

Context-aware Framework for Performance Tuning via Multi-action Evaluation

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*—Context-aware systems perform adaptive changes in several ways. One way is for the system developers to encompass all possible context changes in a context-aware application and embed them into the system. However, this may not suit situations where the system encounters unknown contexts. In such cases, system inferences and adaptive learning are used whereby the system executes one action and evaluates the outcome to self-adapts/self-learns based on that. Unfortunately, this iterative approach is time-consuming if high number of actions needs to be evaluated. By contrast, our framework for context-aware systems finds the best action for unknown context through concurrent multi-action evaluation and self-adaptation which reduces significantly the evolution time in comparison to the iterative approach. In our implementation we show how the context-aware multi-action system can be used for a context-aware evaluation for database performance tuning.**

Keywords-context-aware systems; self-adaptation; multi-action evaluation; database performance tuning

I. INTRODUCTION

A context-aware system reacts to changes in the perceived environment so that computing output is best suited to the current context. There are many definitions of the term context – by location [1], by location combined with behavior [2] or by encompassing multitude of factors such as the definition given by Dey [3]: "Context is 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". This definition allows context-aware system developers to decide what constitutes a context in their application.

A context-aware application does context inference on the basis of the so-called 5W1H (Where, When, What, Who, Why, How) factors [4]. Expanding on this, context-aware applications look at the who's, where's, when's and what's (that is, what the user is doing) entities and use this information to determine why the situation is occurring [5]. However, it is not actually the application that determines why a situation is occurring, but the designer of the application. This dependency on the application designer to capture the context changes is likely to introduce inaccurate contexts and inflexible context definitions [6]. Moreover, the context inference may fail if the system encounters a context which the designer did not foresee.

Self-learning and self-adapting methods can be employed to overcome the aforementioned limitations. These methods use an iterative approach to finding the best possible action when the system encounters unknown context. However, when there are large numbers of actions to evaluate, the time to find the best action takes much longer, which delays significantly the system reaction to the context change.

There are similarities between the reaction of a context-aware system to a context change and the approach of a database administrator (DBA) to a database performance issue and the steps taken to resolve it. Foremost, a DBA has to determine if the performance has regressed (high CPU, high response times) which is similar to sensing a context change in context-aware applications. If the DBA is aware of the reason for the performance issue this would be a known context. However, many factors could affect the database performance [7] and the DBA may not know the root cause outright which can be considered as unknown context.

In such cases the DBA creates a replica of the production database, runs a similar workload to recreate the problem behavior, then hypothesize the root cause and a set of potential solutions to be evaluated one by one [8, 9]. This approach is similar to a context-aware system carrying out each action (potential solutions in DBA's case) iteratively. Once a fix is found, the DBA will know how to detect and fix similar situations in the future in the same way a context-aware system self-adapts and evolves to recognize more and more context changes.

This paper introduces a concurrent multi-action evaluation framework which identifies unknown context and evolves itself to recognize more context changes over time. Our framework implementation showcases how the abstract context-aware concepts could be used for experiment-based database performance tuning and confirms the much higher efficiency of the concurrent evaluation approach.

The rest of this paper is organized as follows. Section II reviews related work on context and self-adapting context-aware models and experiment-based database performance tuning models. Section III provides a description of the proposed framework and formal modelling of it for the database performance tuning case. Section IV describes the implementation and experiment setup. Section V presents experimental results of the evaluation. Finally, the paper concludes with Section VI which summarizes the findings from the evaluation and outlines directions for future work.

II. RELATED WORK

A. Context-aware application models

The simplest model is the single context – single action model. This type of models is commonly used for smart

environments. These models consist of sensory data acquisition, context inference/management and action. Due to the close association this model has with physical hardware each context has one and only one precise action. He et al [10] provide an example of such a smart plant-watering context-aware system. This type of model depends on application developers to capture all possible context changes as such could suffer from inaccurate and inflexible context definitions. A customizable context models is presented in [11] that enable developers to customize the context model to recognize more context changes while [12] make use of a central repository of context knowledge that is periodically updated. But the drawback of having to depend on the system developers is still there. To overcome these limitations self-adapting and self-learning context-aware models are used. In [13] a formal method for incremental context awareness is proposed based on breadth-monotonic model (recognize more situations) and depth-monotonic model (overcome uncertainty). A self-adapting context with the use of context edges (a context edge is the border between two contexts) and context spaces is proposed on [6]. Other self-adapting techniques used by context-aware system include using case base reasoning to address domain specific problems and incomplete data sets [14] and trying to address the lack of domain knowledge through self-adapting whereas [4] proposes a model where both ontological and Bayesian network probabilistic reasoning are used for context reasoning and the context is modelled using ontology. Similarly, the approach described in [15] uses fuzzy sets to allow imperfection in context that is being sensed. IBM's MAPE-K (Monitor, Analyze, Plan, Execute, and Knowledge) loop reference model [16] allows autonomic adaptation of managed elements. The MAPE-K loop also suffers the same limitations as the existing self-adapting context-aware models described earlier primarily due to the fact that MAPE-K uses event-condition-action (ECA) rules for self-adaptation [17]. It is up to the system developers to formulate the ECA rules and to capture all possible ECA combinations. Adaptation will fail if the system encounters an unforeseen event as such MAPE-K loops become infeasible for unknown context handling.

In our proposed framework we eliminate the limitations of these models by employing an experiment-based adaptation mechanism coupled with a knowledge base which is incrementally populated as the system recognizes more and more contexts.

B. *Experiment-based database performance tuning*

Experiment-based database performance tuning models are looked at in the context of our implementation of the proposed framework. As there are magnitudes of factors to consider it is impossible for a DBA to predict the cause for a performance changes without actually trying a real workload on the system [18]. Therefore, the experiment-based performance tuning models give a DBA an accurate understanding on the returns of his tuning effort as opposed to predictive models. In [19] a database's workload and parameter-related prediction model based on neural networks is presented. As it requires training of a neural network beforehand, it lacks the ability to dynamically adapt to change in workload. An experiment-driven query optimization method is described in [8]. At the core of this is a formulation of multiple-query plans based on cardinality sets and then the queries are executed to find better plans. A rapid experiment-defining framework and a high-level language that enable the use of this framework is proposed in [20] which is specifically geared toward production database performance tuning. Compared to query tuning database configuration changes related performance tuning could give high benefits with minimum effort. But modern databases have hundreds if not thousands of parameters and which one to select and what value to set require careful testing. Both [9, 21] present database configuration related experiment-driven performance tuning methods. The response surface mapping technique has been used in both cases to leverage performance tuning based on large number of configuration parameters where [9] uses experimenting away from the production database while [21] uses a cycle stealing algorithm to conduct the experiments on the production database.

Fully automated experiment-driven self-tuning methods are presented in [7, 22]. These systems gather workload statistics overtime and compare current load pattern against historically patterns. If performance seems to have regressed in comparison to historical performance characteristics then a predefined set of actions are implemented to bring the performance to the previous level.

III. PROPOSED FRAMEWORK

The proposed framework consists of three systems, namely the context system, the inference system and the action system. Fig. 1 shows a high-level diagram of the proposed framework and system components. The context system describes context space and the context acquisition mechanism. Following the definition in [3] the database system (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.

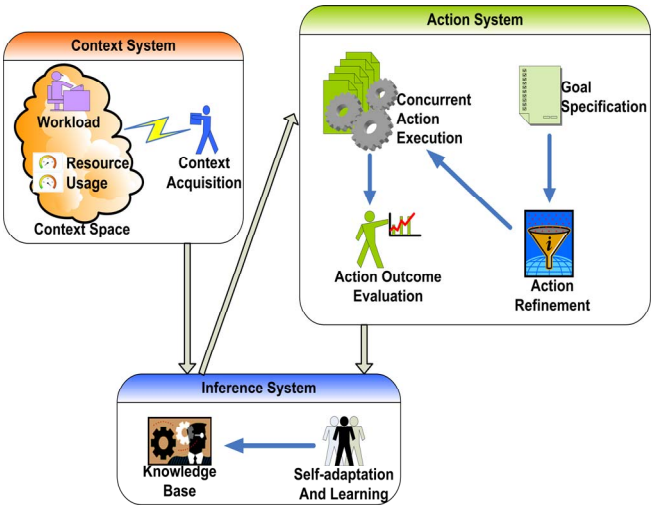

Figure 1. High-level system diagram of proposed framework

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 and any other resource type DBS uses to service the workload. The context acquisition is responsible for sensing and acquiring these context values. Context acquisition is also expected to transform heterogeneous context value types in multiple units into single unit of measurement allowing comparison of contexts. For this implementation two workload types were defined for *W*; *W={OLTP,DSS}* where OLTP is Online transaction processing workload types and DSS is Decision Support System (DSS) workload types. As resource usage *R* the CPU load for each workload type was considered $R=\{CPU_{OLTP}, CPU_{DSS}\}$. It is typical in enterprise scenarios that usage pattern remains constant over time as application uses a similar set of queries [19]. Therefore any change to workload type or resource usage pattern is considered a context change and context inference is carried out.

The inference system consists of a knowledge base and a self-adaption and learning mechanism. 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 self-adaptation and learning mechanism updates the knowledge base with the outcome from the action system.

Knowledge base represents each context in terms of workload type and associated resource usage and was modelled using web ontology language (OWL). The objective was not to create a full ontology but to have an OWL representation of context [23] that would allow leveraging of OWL inherent inference capabilities for context inference. With the use of the OWL only the distinct workload types are needed to be defined, in this case the OLTP and DSS workload types. Other workload types are modeled as complex classes based on the distinct workload types OLTP and DSS. The current context is said to have a mix workload if the resource usage has an OLTP portion as well as a DSS portion. The OWL class specification for Mix workload type is given below.

```
<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 as a union of all three aforementioned workload types. Because of this generic workload type the domain specific workload types (e.g. Order search workload, sales report workload, etc) could be inferred upon and deduced to one of the three aforementioned workload types thus allowing the implementation of the framework to be independent of any domain specific constructs. The OWL specification of this generic workload type is given below.

```
<owl:Class rdf:ID="Workload">
    <rdfs:subClassOf>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Mix"/>
          <owl:Class rdf:about="#DSS"/>
          <owl:Class rdf:about="#OLTP"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:subClassOf>
    <rdfs:subClassOf
  </owl:Class>
```

In the proposed self-adapting model the knowledge base starts off with few known contexts. This initial knowledge could be derived from DBA's knowledge of the system, past experience and is not expected to be extensive.

The action system is responsible for concurrent action execution 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. To achieve this action system uses goal specification and action refinement. The goal specification defines the extremities of the variable parameter used in the actions. In the implementation the actions consisted of executing a representative workload using a different configuration value for each experiment. Representative workload is created by capturing the top resource consuming queries for each workload type. The goal specification in this case would be the minimum and maximum values to be used in the experiments but not necessarily the maximum and minimum values parameter is configurable for the DBS. These extremities are denoted as G_{lo} and G_{hi} and are considered elements of the configuration parameter space.

The action refinement limits what action qualifies to be in the action space. Without the limiting effects of the action refinement context-aware system would have to experiment on every value between G_{lo} and G_{hi} which would be a resource and time intensive endeavor. The action limiting process starts by identifying the context that is closest to the unknown context. 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. The configuration parameter setting of this known context is used to derive the initial action. This is denoted as A_k and defined as a function of the configuration parameter $configuration_k$ of the closest known action.

To derive other actions in the action space three integer parameters are introduced. They are the lower bound expansion range denoted by *p* which specifies 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 denoted by Δ. Having defined these three parameters the total number of actions (or experiments) which need to be executed could be defined as a union of three action sets.

$$
\begin{aligned}
\textit{Action space} = \{\ & A_k\,(\textit{configuration}_k) \ \cup \\
& A_p\,(\textit{configuration}_p) \ \cup \\
& A_q\,(\textit{configuration}_q) \\
& \mid\ p = \{1 \,..\, n\},\ n > 0,\ q = \{1 \,..\, m\},\ m > 0, \\
& \quad \textit{configuration}_k - p\Delta \geq G_{lo,}, \\
& \quad \textit{configuration}_k + q\Delta \leq G_{hi}, \\
& \quad \Delta > 0 \\
& \}
\end{aligned}
$$

The concurrent experimentation takes place in a private workbench which ensures that configuration changes in each action do not affect the current state of the DBS. The final phase of the action system is the outcome evaluation. The evaluation criteria for choosing the action that results in the highest benefit depends on the domain the context-aware system is implemented in. For the database performances tuning the actions are evaluated based on a minimizing function. That is the best action is the one with the configuration parameter that resulted in minimum resource usage such as the CPU usage for the execution of queries [18]. Thus the best configuration parameter for the unknown context could be formally defined as

$$
\begin{aligned}
& \textit{configuration}_{best} = \{ \\
& \quad \forall \textit{configuration}_i \in \{\textit{action space configurations}\} \\
& \quad \exists A_i\,(\textit{configuration}_i)\text{: minimum (Resource Usage } (A_i)) \\
& \}
\end{aligned}
$$

Once the best setting for the configuration parameter is known for the current context it could be used to update the knowledge base so the system recognizes this context in the future (learning and adaptation) and at the same time DBS configuration setting is changed to optimize the workload execution.

IV. IMPLEMENTATION

The experimental setup used for our implementation is shown on Fig. 2. It consists of a primary DBS and a standby DBS. The primary DBS is part of the context space from which context values are acquired. The standby DBS is used as the experimentation environment. The context-aware experiment-based performance tuning system was developed as a Java desktop application with a graphical interface. It ran on a desktop computer with 8 GB RAM, 2.8 GHz Intel dual core processor and using a 1.7 JVM. Two Oracle DB connection services were created in the database for each of the workload type *W*, thus enabling the measurement of the resource usage *R* of each workload type. Context sensing was carried out by polling the database every 5 seconds and evaluating the CPU usage of each service assigned to the workload types which only incurred 0.000015% of the total hourly CPU time available on the server. A change in the resource usage pattern is considered a context change if there was a 3% difference in the CPU usage between 3 consecutive samples. When an unknown context is encountered the context-aware application would identify the high resource-consuming queries for each workload type and bring them to the private workbench for experimentation. In this case, the experimentation would be the concurrent execution of queries under various configuration parameter settings.

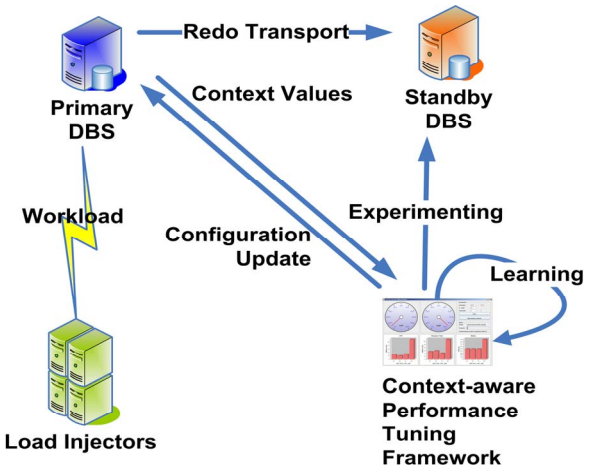

Figure 2. Implementation setup

The chosen DBS was Oracle 12.1 for both primary DBS and standby DBS which ran on a server with 12GB RAM, 2.0 GHz Intel quad core processor server. Workloads were generated against two of Oracle database's sample schemas namely OE (OLTP workload types) and SH (DSS workload types). The data volumes of the sample schemas were increased with the use of Swingbench. Oracle configuration parameter optimizer_index_cost_adj was chosen as the parameter to demonstrate the concrete implementation of the framework. The rule of thumb is to set it to lower values for OLTP which makes query plans bias towards index access and to higher values for DSS in favor of full table scans. DBAs are likely to leave it at default value for fear of setting it to an incorrect value. The rule of thumb may not work for every application and therefore this parameter is an ideal candidate for experiment-based performance tuning. In addition, it gives a direct correlation between the resource usage of each plan and the parameter value being used.

By comparing the resource usage of various queries under different parameter value settings it is possible to identify which parameter is beneficial to be used for the current context (i.e. workload type). The value that could be set for the parameter has a wide range (1-10000) but for an enterprise application the applicable range is much narrower, which is the concept demonstrated by the goal specification. The parameter could be set at session level which is visible only to the candidate session and opaque to other database sessions. As such it provides a private workbench to carry out the concurrent. Although optimizer_index_cost_adj is used here, any configuration parameter could have been used with the context-aware model. The CPU used by each session for experiment (query) under various parameter settings was chosen as the statistics on which the action evaluated is based upon. The CPU time represents the total time consumed by a query on CPU and is not affected by contention issues and is provided by Oracle DB internal statistics gathering process [24].

The knowledge base was modelled using the Java implementation of Protégé OWL API while for inference the Java API for Protégé Pellet Reasoner was used. The knowledge base was initially populated with three context facts – one relating to OLTP only workloads, another to DSS only workloads and one for mix workload type where CPU usage was distributed (90:10) among OLTP and DSS workloads.

V. Experiments and 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 5 minute regular intervals in the following sequence. 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_{13}: {$OLTP_1$}. The $OLTP_1$ workload represents the regular online transaction processing workload the DBS is tuned for. The DSS_1 and DSS_2 represent two different decision support queries that are executed infrequently.

Oracle's enterprise manager (OEM) console was used to compare the overall CPU usage for the same workload pattern with and without context-aware experiment-based adaptation. Fig. 3 shows the CPU usage pattern when the DBS configuration was optimized only for OLTP workloads while Fig. 4 represents CPU usage when DBS is able to adapt because of the context-aware system. The OEM plots the CPU usage in terms of active sessions (Y-axis), which is calculated as (active session = CPU used by all non-idle DBS session / wall clock time). There were no other workloads on the test DBS except for the experimental workloads generated and therefore the two graphs on Fig. 3 and Fig. 4 are directly comparable. The X-axis unit is minutes and the time-steps denote workload injection and experiment/adaptation points.

The experiment starts off with the system containing only the $OLTP_1$ workload and the first DSS workload (DSS_1) is injected at T_7. This changes the CPU usage distribution and is detected by the context-aware application as a context change. Inference of this context change reveals that this is an unknown context which triggers concurrent experimentation on the standby DB. During this experimentation period (T_7 – T_8) the CPU usage reaches the same level as in the test without context-aware adaptation (T_1 – T_2). However, once the experiment results have been evaluated and the adaptation has completed, the overall CPU usage is lower for the rest of the time (T_8 – T_9) compared to the period without experiment-based adaptation. The difference in the absolute CPU usage was 150 CPU seconds which means the DBS uses 20% less CPU to execute the same workload after the adaptation. The knowledge base is expanded with the current context space values and the parameter setting suited for it.

The second DSS workload (DSS_2) is injected at T_9 but as the DBS has already been adapted to one DSS workload the increase in the overall CPU usage is lower compared to the same workload mix (T_2 – T_3) without experiment-based adaptation. However, the injection of this second workload results in the detection of unknown context followed by concurrent experimentation in the time period of (T_9 – T_{10}). Evaluation of the experimental results indicated that the already used parameter settings lead to the lowest overall CPU usage for the current context. Thus, experimentation and adaptation do not result in lowering of the CPU usage for this workload period (T_9 – T_{11}) but it is still lower compared to (T_2 – T_3) which had a similar workload mix. The knowledge base is updated with parameter setting so context-aware system is capable of detecting and adapting irrespective of the order of workload occurrence. These observations could be validated by examining the results of the concurrent experiments which is the CPU used per single execution of a query under different parameter settings. These results are used by the action evaluation module to determine the action that maximizes the goal expectations. The use of weighted average ensures the configuration parameter changes do not negatively affect the workload type that is most active at the time of the context change.

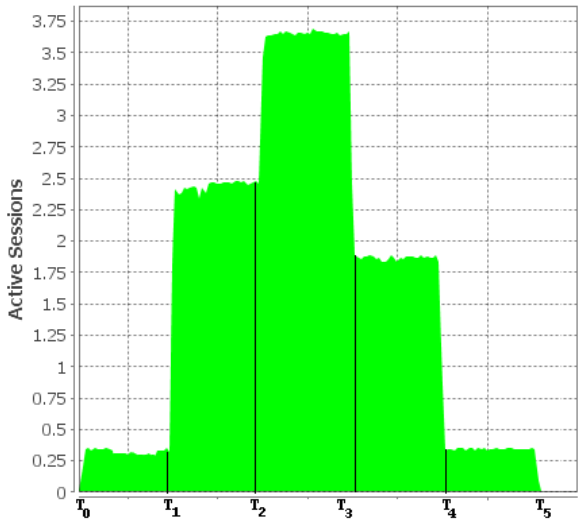

Figure 3. CPU usage without experiment-based adaptation

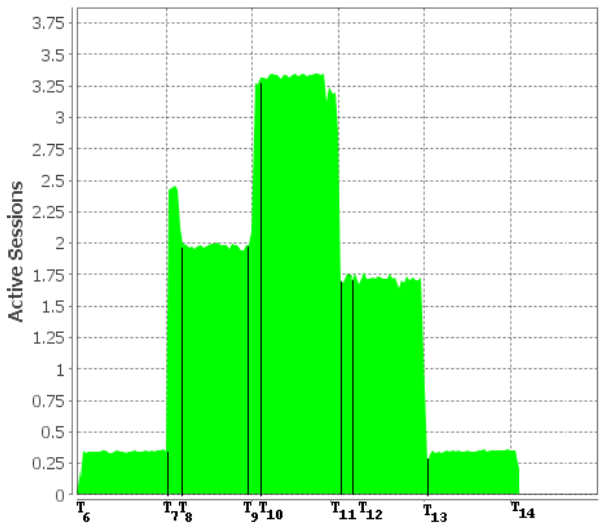

Figure 4. CPU usage with context-aware experiment-based adaptation

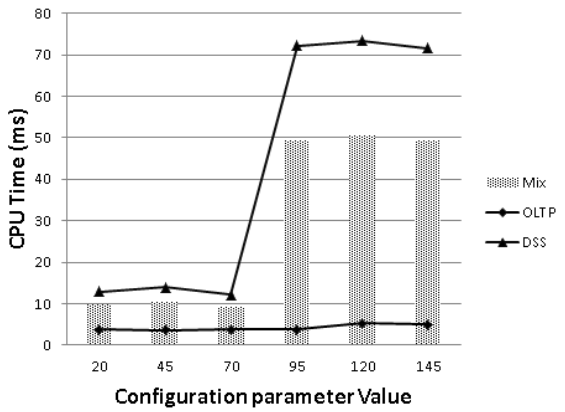

Figure 5. CPU usage for each configuration settings

The system experiments 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. Fig. 5 shows the results from the concurrent execution of experiments under various configuration settings. For example, when setting the configuration parameter to 70 results in the lowest CPU usage to execute the current workloads. However, if only the OLTP workload type is considered then the best configuration parameter setting would be 95. In a mixed workload environment this would lead to regressed performance on the DSS workload queries. Furthermore, these results could be used by the DBA to validate the initial knowledge base facts and modify if these initial facts were not the ideal configuration values.

VI. Conclusion and Future Work

This paper introduces a novel context-aware application framework which enables self-adaptation through concurrent action execution when a context-aware application encounters an unknown context. Our solution eliminates the limitations of the currently existing approaches by employing an experiment-based adaptation mechanism. A concrete implementation of this framework has been completed showing how it could be used in experiment-based database performance tuning. The experimental results have shown that with the use of the context-aware approach a DBS adapts to changing workload types resulting in lower resource usage.

Our immediate future plans include the introduction of a new paradigm for context-aware database performance tuning where instead of the one-setting-fits-all approach, DBS automatically update and evolve their settings to best suit the current workloads; thus providing substantial assistance for the DBA to efficiently tune the performance.

References

[1] B.N. Schilit and M.M Theimer, Disseminating active map information to mobile hosts, IEEE Network, 8 (5), 22-32, 1994.

[2] P.J. Brown, J.D. Bovey and X. Chen, Context-aware applications: from the laboratory to the marketplace, IEEE Personal Communications, 4 (5), 58-64, 1997.

[3] A.K. Dey, Understanding and using context, Personal and Ubiquitous Computing, vol. 5, no. 1, pp. 4-7, 2001.

[4] 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.

[5] J. Madhusudanan, A. Selvakumar and R. Sudha, Frame work for context aware applications, Computing Communication and Networking Technologies (ICCCNT), pp. 1-4, 2010.

[6] 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.

[7] 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.

[8] H. Herodotou and S. Babu, Automated SQL tuning through trial and (sometimes) error. *Workshop on Testing Database Systems*, 2009.

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

[10] J. He, Y. Zhang, G. Huang and J. Cao, A smart web service based on the context of things. ACM Trans. Internet Technology. 11 (3), 13:1-13:23, 2012.

[11] L.Yu, Z.Wang, Y. Huang and S. Chen, Building Customizable Context-aware Systems, Service Sciences (IJCSS), 2011 International Joint Conference on , vol., no., pp.252,256, 25-27 May 2011.

[12] J. Chang, S. Na and M. Yoon, Intelligent Context-Aware System Architecture in Pervasive Computing Environment, Parallel and Distributed Processing with Applications, 2008. ISPA '08. International Symposium on , vol., no., pp.745,750, 2008.

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

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

[15] C. Anagnostopoulos and S. Hadjiefthymiades, Advanced Inference in Situation-Aware Computing. IEEE Trans. Systems, Man and Cybernetics, Part A: Systems and Humans, 39 (5), 1108-1115, 2009.

[16] IBM, An architectural blueprint for autonomic computing, , http://ibm.co/1IP7TvG, last accessed 2014/07/14.

[17] M.C. Huebscher and J.A. McCann, A survey of autonomic computing—degrees, models, and applications, ACM Computing Surveys (CSUR), v.40 n.3, p.1-28,2008.

[18] K. Yagoub, P. Belknap, B. Dageville, K. Dias, S. Joshi, and H. Yu, Oracle's SQL performance analyzer, IEEE Data Engineering Bulletin, 31(1), 2008.

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

[20] N.Borisov and S.Babu, Rapid experimentation for testing and tuning a production database deployment. 16th International Conference on Extending Database Technology (EDBT '13), 125-136, 2013.

[21] 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.

[22] P.Belknap, B.Dageville, K.Dias, K. Yagoub, Self-Tuning for SQL Performance in Oracle Database 11g, Data Engineering, pp.1694,1700, 2009.

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

[24] 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.