A Variated Monte Carlo Tree Search Algorithm for Automatic Performance Tuning to Achieve Load Scalability in InnoDB Storage Engines

Allan Odhiambo Omondi[1], Ismail Ateya Lukandu[1], Gregory Wabuke Wanyembi[2]
[1]Faculty of Information Technology, Strathmore University, Nairobi, Kenya
[2]School of Computing and Informatics, Mount Kenya University, Thika, Kenya

Abstract**—***Variable environmental conditions and runtime phenomena require system developers of complex business information systems to expose a set of configuration parameters to system administrators. This allows the system administrators to adjust the configuration parameters in response to the changes or in anticipation of future problems. However, these adjustments are prone to error and lack of standards due to varying levels of expertise and unpredictable short-term future states of a business information system. The purpose of this study is therefore to investigate on how to design an algorithm that proactively reconfigures bottleneck parameters without over-relying on an accurate model of an unpredictable stochastic environment. The measure of effectiveness of the reconfiguration is based on the transaction throughput and response-time latency as the server strives towards achieving load scalability. An experiment-based research design was used in conjunction with hypothesis testing to analyze the experiment results. Over 95% of the experiment results provided evidence that the use of the distributed database architecture with the designed algorithm running on each node, resulted in faster transaction throughput and slower response-time latency. The study builds on automatic performance tuning, reinforcement learning, and simulation-based inferential statistics literature by combining the three fields. A Monte Carlo Tree Search with variated selection, expansion, and simulation stages was used as the core of the designed algorithm.*

Keywords**—***Automatic Performance Tuning, Monte Carlo Tree Search, Reinforcement Learning, Distributed Databases.*

I. INTRODUCTION

Scalability in computer-based business information systems is heavily reliant on manual tuning interventions from experienced system administrators. This results in substandard load scalability due to factors such as fatigue, long reaction times, and inconsistent expertise amongst system administrators [1] [2] [3]. To overcome these challenges, this study investigates on how to design an algorithm that proactively reconfigures bottleneck parameters without over-relying on an accurate model of an unpredictable stochastic environment. This is done in order to achieve self-optimization required for load scalability in storage servers.

Recent trends in database theory have led to a proliferation of studies that are inclined towards the support of analytical operations on big data. Central to the entire discipline is the concept of performance tuning in order to maintain an acceptable performance level as the volume of data to be processed scales up. Contrary to some opinions, achieving scalability in database servers through performance tuning is not trivial due to the complexity caused by numerous configuration knobs [4]. Variable environmental conditions and runtime phenomena further add to the complexity.

Previous attempts to automate the performance tuning process have applied linear optimization techniques based on greedy algorithms. For example, constraint programming, mixed integer programming, and local search, all of which are discussed further in [5]. The main shortcoming of these approaches is their unsuccessful attempt to model the variable environmental conditions and runtime phenomena as a linear process.

This study submits the following composite hypothesis in order to perform a relevant investigation:

Null Hypothesis (H_0): Distributed database systems that apply the designed autonomic latency-aware algorithm on average have the same transaction throughput and response-time latency.

Alternative Hypothesis (H_1): Distributed database systems that apply the designed autonomic latency-aware algorithm on average have a faster transaction throughput.

Alternative Hypothesis (H_2): Distributed database systems that apply the designed autonomic latency-aware algorithm on average have a slower response-time latency.

$$H_0: \mu = \mu_{H_0}$$
$$H_1: \mu > \mu_{H_0}$$
$$H_2: \mu < \mu_{H_0}$$

Moreover, the study seeks to answer the following research questions:

i. *How can a latency-aware algorithm be designed to proactively reconfigure bottleneck parameters without over-relying on an accurate model of an unpredictable stochastic environment?*
ii. *How can experiments be performed to test the designed algorithm using an emulation of business-oriented, ad-hoc queries on a test bed?*

Section II. of this article provides a Literature Review that focuses on the theoretical concepts of scalability, profile-guided configuration optimization, and latency. Section III. then provides a detailed explanation of the applied methodology as it answers the second research question. This is divided into 4 sub-sections: the philosophical assumptions and experiment procedure, the experiment test-bed, experiment test-data and the data analysis methods. Sections IV. and V. document the results as well as a discussion of what the results mean. Lastly, Section VI. concludes the article with recommendations for IT practitioners.

II. Profile-Guided Configuration Optimization

There is a direct correlation between a system's scalability level and a business' growth capacity. Systems that support business should be able to handle an increased workload without impacting on their responsiveness to execute any action within a predetermined time interval. As stated in [6], the inability to cope with growth can lead to loss of business. The most critical component of a business information system to consider when implementing scalability is its foundation. The foundation is where it can store and manipulate data prior to applying algorithms to process it into information or even further into knowledge. The foundation is the database which is the prime focus of this study. [4] showed that scalability in database systems can in turn be measured based on transaction throughput (measured in transactions per second) response-time latency (measured in microseconds), and also the degree of resilience to faults.

An increase in the number of client requests to process compute-intensive workloads has a significant effect on a system's performance. This is because it can cause a section of the system to reach a limit in its ability to respond to additional client requests. Despite this, the nature of the workload is not constant. It is therefore necessary to reactively or proactively adjust the system in order to adapt to different workloads. Through these continuous adaptations, the performance of a system can be optimized to maintain it at an acceptable performance level regardless of the nature of the workload.

The following iterative steps can be followed as a general guideline when optimizing the performance of a system:

Step	Description
I	Define the desired performance level. This can be determined by an internal or external Service Level Agreement (SLA). In the case of database systems, this can be quantitatively defined by the transaction throughput (Transactions per Second [TPS]), the average response-time (microseconds), and the workload-related error rate.
II	Measure the performance of the system to define its profile before making any changes to its configurations.
	If the current profile is below the desired performance level, proceed to **Step III.**
	Otherwise, proceed to **Step VI.**
III	Analyse the profile to identify the critical parts of the system, which if improved, will have the most significant impact on the system's performance
IV	Improve the critical parts of the system identified in **Step III**
V	Measure the performance of the system after the reconfigurations have been implemented
VI	If the current profile exceeds the desired performance level, control the current configurations to ensure there are no deviations.
	Otherwise, if the current profile is worse than the previous performance level, then return the system to its pervious state and proceed to **Step I.**

The improvements in Step IV can involve any of the following types of modifications in the case of database systems:
(i) Code optimization
(ii) Caching strategy
(iii) Load balancing
(iv) Distributed computing
(v) Configuration optimization

Step II involves the act of profiling. Profiling in this study refers to a dynamic system analysis technique that instruments a program in order to collect data that measures different metrics of a system. These work metrics include, but are not limited to, the amount of work the system is doing per unit of time (transaction throughput), the time required to complete a unit of work (response-time latency), and the rate of errors per unit of time or per unit of work. In addition to these, resource metrics and event metrics are collected and used to provide further insight.

The study applied a statistical and instrumented profiling method which involves modifying a program to enable it to profile itself periodically. This was made possible through the use of the "sys" database that is in turn based on the "performance_schema" database. [7], [8], and [9] showed that statistical profilers are less intrusive to the target system because they rely on periodic sampling. Such a method can therefore be setup to periodically profile a system that is in production.

A self-tuning system must take latency into consideration. Latency in this case refers to the time it takes for an adaptation (a change in system configurations) to become effective in a system. It is possible for the effects of responsive adjustment of configuration parameters to be appreciated after the conditions that warranted the adaptation in the first place are no longer present. Step IV of the process should therefore be designed to be cognizant of this fact.

III. Methodology

A. *Philosophical Assumptions and Experiment Procedure*

The philosophical assumptions made in this study predicate all the choices made concerning the research methodology. The research questions do not seek to understand the dynamic and subjective reality of social actors (the system administrators) in order to make sense of their motives and actions. For this reason, the study applied ontological materialism, which is objective in nature. This objectivity matches with positivism as the epistemological approach because positivism emphasizes on the use of observations in order to justify claims. Given a cross-sectional time horizon, deductive reasoning moves from existing theoretical knowledge, to formation of a testable proposition (a hypothesis), to acceptance/rejection of the proposition by confronting it with factual data. This leads to positivism that uses a mono-method, quantitative choice that can be applied in the form of an experimental research design.

A systematic and scientific approach is preferred in order to design a latency-aware algorithm that automatically optimizes scalability in storage servers. This can be made possible through the use of experiments as outlined in the following steps.

Step	Description
I	Define a realistic and reliable model of the objective function that forms an adequate (not perfect) image of reality. This objective function should consider all the features that have the highest correlation with the system's scalability.
II	Design an algorithm that effectively achieves the pre-defined objective
III	Theoretically analyze the asymptotic behavior of the designed algorithm. This is done in order to measure the level of algorithm correctness, time complexity and space complexity.
IV	Complement the theoretical analysis by conducting controlled experiments in the form of empirical algorithmics. This is done using treatments that manipulate the algorithm and measurements that identify the effect of the manipulation on the scalability of the storage server. Each of the experiments in Step IV is repeated 27 times in order to increase the statistical accuracy of the results and subsequently to guarantee the reliability of the study. The number 27, which is less than 30, is chosen in order to use a T-score as a test statistic during data analysis. This is based on the fact that we do not know the population's standard deviation. The repeating of experiments also contributes towards establishing a temporal precedence which in turn guarantees the internal validity of the research.
Apply the decision rule that will determine whether to reject or fail to reject the null hypothesis. Go back to **Step II** if there is no reason to reject the null hypothesis. Otherwise, proceed to **Step V** if the null hypothesis is rejected in favor of an alternative hypothesis.	
V	Assemble the best-performing algorithm variations into an algorithm library
VI	Implement a real application by importing (making use of) the algorithm library

B. *Experiment Test-Data*

Benchmarking tools reduce the diversity of operations by providing a standard, accurate and repeatable server-side workload that is representative of a real-world production system. The mimicking of a real-world environment in the form of a production system contributes towards guaranteeing the population validity of the study. The degree of population validity in turn contributes towards the generalizability of the results of the study to a population and subsequently increases the study's external validity. Client-side workloads, such as general user-interface functions, are excluded from most benchmarking tools. The benchmarking tool applied in this study works by first generating and loading large volumes of data into the database system. It then generates and submits transactions and their inputs (that is, the server-side workload) to the database system. Lastly, it measures the rate of completed transactions being returned. The Transaction Processing Performance Council's (TPC's) TPC Benchmark is one such example that can be customized depending on the test being conducted. The American National Standards Institute (ANSI) Structured Query Language (SQL) Standard Scalable and Portable (AS^{3}AP) Benchmark is also another example of a benchmarking tool for database systems. This study applied a customized version of the AS^{3}AP benchmark.

The justification for applying it is its combination of OLTP and OLAP workloads in a single experiment. This is unlike TPC-E and TPC-H which separate OLTP and OLAP workloads respectively. This separation is not always ideal given the presence of business applications that are a hybrid of OLTP and OLAP. The study used the AS^{3}AP Benchmark by submitting "bulk append" and "information retrieval" queries to represent OLTP and OLAP workloads respectively. This was done using 20 concurrent virtual users each added at 2 second intervals. 1/3 of the total running time of each experiment was used for pre-sampling to provide the database system adequate time to reach a steady state before the required statistics are collected. The remaining 2/3 of the total running time was then used to conduct the actual sampling (collection of the required statistics). Backup sets of tables were created at the initial stage of the experiment and used to refresh the data to its original state upon completion of each experiment. The refreshing of data was necessary as a scientific control to prevent situations where the results of an experiment are influenced by the result of the previous experiment. The use of a scientific control subsequently increases the reliability of the study.

A. *Experiment Test-Bed*

The experiment test-bed was made up of a distributed database with Maria DB 10.2.14 installed as the Distributed Database Management System (DDBMS). There were 3 nodes in the cluster, each configured as a master with no slaves and there was synchronous replication between all the 3 nodes. The synchronous replication was made possible through the use of the Write-Set REPlication (WSREP) Application Programming Interface (API). The WSREP API implements an eager replication whereby nodes in the cluster synchronize their states (database content) with all other nodes by updating the replicas through a single transaction. A load balancer based on a least connections balancing solution was also configured. The least connections balancing solution worked by forwarding connections to the server with the least number of connections. The distributed system was based on a shared-nothing architecture such that each of the 3 nodes had their own CPU and storage as Virtual Machines (VMs). All the 3 nodes plus the load balancer were running a 64-bit Ubuntu Server 16.04 LTS as the Operating System. Fig. *1* shows the architecture of the test-bed.

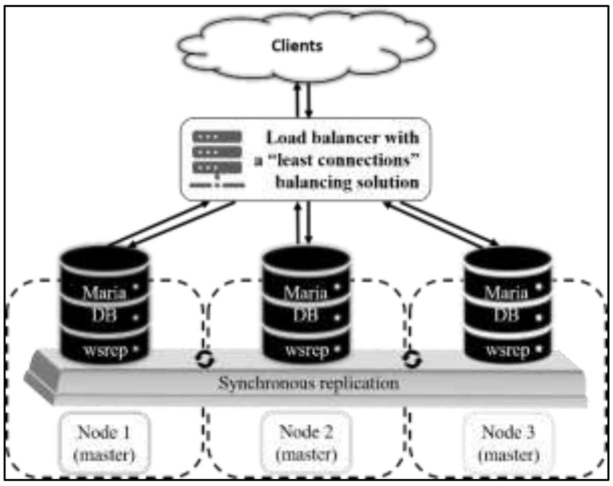

Fig. 1. Architecture of the test-bed.

The test-bed aimed to model a real-world environment whereby the normal architecture is that of a distributed database for the sake of High Availability/Disaster Recovery (HA/DR) features. This was done in order guarantee the ecological validity of the research. Ecological validity subsequently contributes towards generalizability of the results of the study to a population as part of external validity.

B. Data Analysis Methods

The study was willing to take a maximum risk of 5% for rejecting the null hypothesis when it is true (Type I error). The value of 5% was arrived at with the aim of striking an adequate balance between Type I and Type II errors, both of which are negative. Consequently, a p-value of 0.05 or less was used to measure how often an outcome happens over repeated execution of experiments.

As indicated in the experiment procedure outlined in Section A, the decision rule determines whether to reject or fail to reject the null hypothesis. The decision rule applied in the study states that the null hypothesis should be rejected if at least 95% of all the experiments executed for a specific treatment or variation of the algorithm result in a faster transaction throughput and query response time.

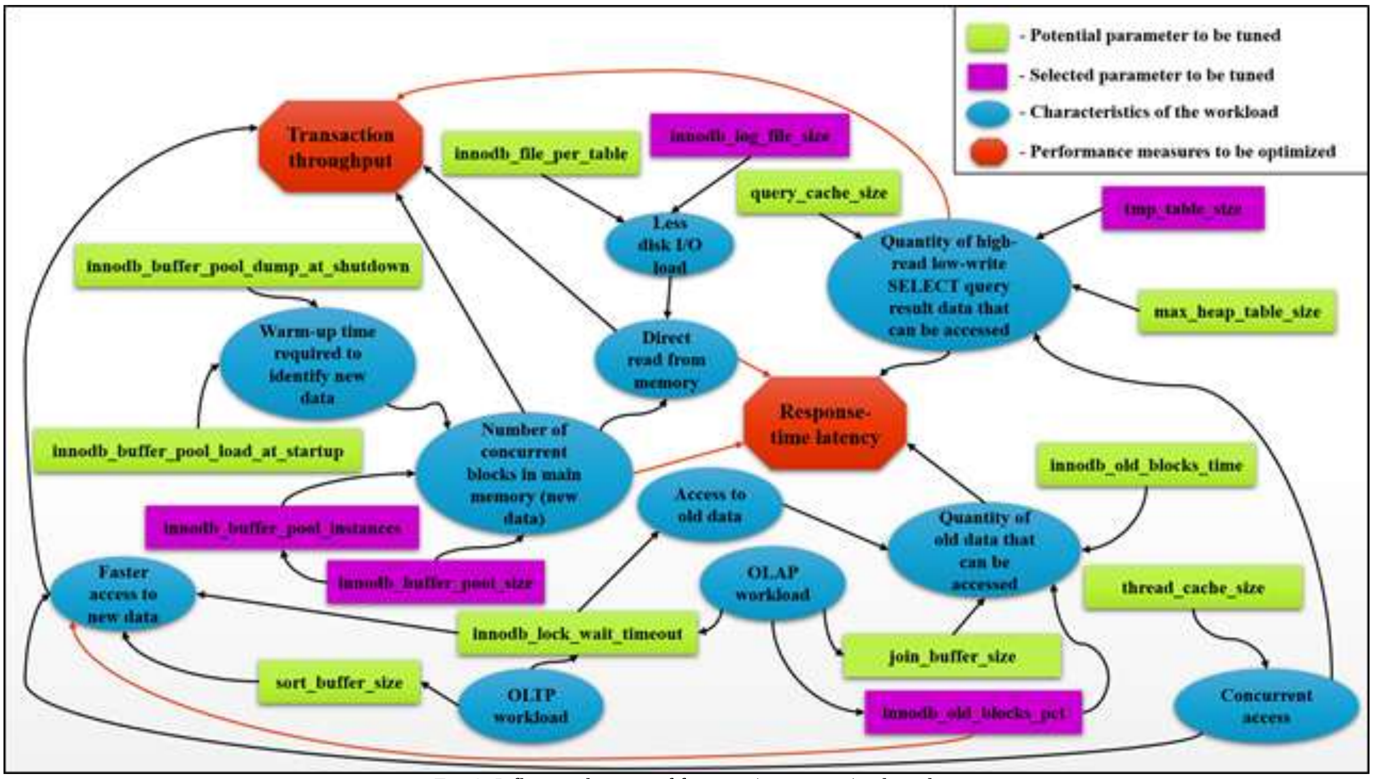
- Potential parameter to be tuned
- Selected parameter to be tuned
- Characteristics of the workload
- Performance measures to be optimized
Transaction throughput
innodb_file_per_table
innodb_log_file_size
query_cache_size
tmp_table_size
Less disk I/O load
Quantity of high-read low-write SELECT query result data that can be accessed
max_heap_table_size
innodb_buffer_pool_dump_at_shutdown
Warm-up time required to identify new data
Direct read from memory
Response-time latency
innodb_buffer_pool_load_at_startup
Number of concurrent blocks in main memory (new data)
innodb_old_blocks_time
innodb_buffer_pool_instances
Access to old data
Quantity of old data that can be accessed
innodb_buffer_pool_size
Faster access to new data
OLAP workload
thread_cache_size
innodb_lock_wait_timeout
join_buffer_size
sort_buffer_size
OLTP workload
innodb_old_blocks_pct
Concurrent access

Fig. 2. Influence diagram of features (parameters) selected.

A one-tailed test (right-tail for testing the 1st alternative hypothesis and left-tail for testing the 2nd alternative hypothesis) involving a T-score was used to measure the level of difference between the results and what was expected. The T-score supports the transformation of an individual score into a standardized form for easier comparison. The greater the difference from the expected T-score, the more evidence there is that the results of an experiment are significantly different from the average expected results. Given that the null hypothesis represents the expected results then the null hypothesis cannot be true when the actual results are different from the expected results. The decision rule can therefore be extended to state:

Reject H_0 if $\text{T score}_{\text{calculated}} > \text{T score}_{\text{tabular}}$ in the case of the 1st alternative hypothesis (H_1) and

Reject H_0 if $\text{T score}_{\text{calculated}} < \text{T score}_{\text{tabular}}$ in the case of the 2nd alternative hypothesis (H_2).

IV. Results

A. Feature Selection

A typical database system contains hundreds of features (parameters) that can be used in optimizing the transaction throughput and response-time latency [4]. However, some of these features are either redundant or irrelevant and can therefore be discarded without negatively impacting the algorithm. The main advantage of discarding irrelevant or redundant features through feature selection is that it enhances generalization of the algorithm by reducing overfitting.

A wrapper technique known as stepwise regression was used to select a subset of 5 most relevant features to be used by the algorithm. The criterion used to decide which feature was retained in the model upon each feed-forward iteration of the stepwise regression was that the feature's β-coefficient should have the highest, absolute t score. Stepwise regression was used in conjunction with underlying knowledge from domain experts obtained through reviewed literature. The 5 features (parameters) selected are depicted in the influence diagram in Fig. 2. Appendix A further provides raw data of the stepwise regression runs.

The 3 most relevant features for low-read, high-write OLTP workloads were:

i. **innodb_old_blocks_pct** (x_3): It specifies the maximum percentage of the buffer pool dedicated to storing "old" blocks of data and indexes that were frequently used in the past. The value can range between 5% and 95% of the entire buffer pool. A smaller percentage value enables faster eviction of less frequently used old blocks of data and indexes from the buffer pool. This eviction in turn creates room for more frequently used "new" blocks of data to be stored in the buffer pool. It subsequently reduces the bottleneck caused by disk IO and has a direct impact on transaction throughput for OLTP workloads.

ii. **tmp_table_size** (x_12): It specifies the size allocated for temporary tables. Temporary tables are used when processing complex queries that involve joins and sorting. The size allocated to tmp_table_size must be the same as the size allocated to **max_heap_table_size**. The results showed that this parameter has a direct impact on transaction throughput for OLTP workloads. The results of the stepwise regression were contrary to the expectation that the parameter has a direct impact on response time for OLAP workloads.

iii. **innodb_buffer_pool_instances** (x_2): It divides the buffer pool into equal-sized instances each of which manages its own data. It subsequently reduces concurrency problems caused by a shared buffer pool. According to the study's stepwise regression results, it has a direct impact on the transaction throughput of high-write, concurrent OLTP workloads.

Whereas, the 2 most relevant features for high-read, low-write OLAP workloads were:

i. **innodb_buffer_pool_size** (x_1): It specifies the amount of RAM that should be set aside to store frequently used blocks of data and indexes. The results showed that it has a direct impact on the response time of high-read OLAP workloads.

ii. **innodb_log_file_size** (x_8): It reduces the disk I/O because of less flushing of checkpoint activity. It also increases the speed of database writes and the durability of transactions. According to the stepwise regression results, it directly affects the response-time of OLAP workloads.

The study categorized the possible values of the parameters into groups, referred to as tactics. These were: low, medium, and high. Using tactics as a collection of parameter values is a more reasonable approach because it is common for parameters to have different units of measurement. For example, innodb_old_blocks_pct uses percentage units while tmp_table_size uses main memory in Megabytes as units.

Key Parameters	**Range of Values**		
	Low	**Medium**	**High**
innodb_old_blocks_pct (x_3)	95%	50%	5%
tmp_table_size (x_12) = max_heap_table_size (x_13)	64M	128M	192M
innodb_buffer_pool_instances (x_2)	64 instances	33 instances	2 instances
innodb_buffer_pool_size (x_1)	1504M	1981M	2458M
innodb_log_file_size (x_8)	376M	496M	615M

B. *Algorithm Design*

The study quantitatively defined the current state of the database by measuring the transaction throughput (measured in Transactions Per Second [TPS]) and the response-time latency (measured in microseconds). Given the current state, it is necessary to determine the action to perform that will lead to a subsequent state that has the highest possible value. This is true even though the state with the highest possible value is not the immediate next state. The challenge is that the correct action is not known, given that there are variable environmental conditions and runtime phenomena. The state that is a result of an action therefore varies depending on the environmental conditions and runtime phenomena which are stochastic in nature.

Monte Carlo simulations enable us to estimate the value of subsequent states through numerous random simulations. This enables us to identify the action that, if performed, will lead to the state with the highest possible value. Past research has demonstrated that combining the generality of random simulations and the precision of a tree search in the form of a Monte Carlo Tree Search, has a significantly positive effect on the effectiveness of an autonomous agent [10]. The following Sections describe the stages of the Monte Carlo Tree Search with a variation at the selection stage. The term "node" in the following Sections is used to refer to the state of the database system.

1. *Variated Selection Stage*

The selection stage selects a node according to a predefined utility function. The selection continues through the search tree until it encounters a node that has not been fully expanded to allow its children to be explored. However, the selection stage faces the challenge of focusing too much on the most promising nodes while neglecting nodes whose immediate reward is inadequate but may turn out to lead to a superior reward in the long-run.

[10] shows that it is possible to variate the selection stage by combining Upper Confidence Bound applied to Trees (UCT) and lean Last Good Reply with Forgetting (lean-LGRF). UCT makes use of confidence intervals to balance between exploitation of known good nodes and exploration of nodes that are not currently the best but may lead to good nodes along the tree. Therefore, no potential node is starved of selections, and at the same time, favorable nodes are selected more often than their counterparts. This is essentially a form of balancing between order (exploitation) and chaos (exploration). Chaos is necessary to provide an adequate amount of challenge required for growth. The UCT formula is as shown in (1):

$$\underset{i \in childrenOfTheRootNode}{\operatorname{argmax}} \left(\frac{\gamma(n_i)}{x(n_i)} + \sqrt{c \times \frac{\log(t)}{x(n_i)}} \right) \qquad (1)$$

Where:
$\boldsymbol{\gamma(n_i)}$ is the total discounted value of selecting node i. It is discounted because we cannot get to a terminal state that determines a final win or loss. If i is an OLTP workload, the value is based on transaction throughput and if it is an OLAP workload, the value is based on response-time latency.
$\boldsymbol{x(n_i)}$ stands for the total number of simulations that have occurred for the node i. $\frac{\gamma(n_i)}{x(n_i)}$ is therefore the value of the node and forms the exploitation parameter of the UCB formula.
$\boldsymbol{c}$ stands for the exploration parameter. An increase in the value of $\boldsymbol{c}$ results in more exploration. The study used a value of $\boldsymbol{c = 5,000}$ as the baseline
$\boldsymbol{t}$ stands for the total number of simulations that have occurred for the parent of node i

Lean-LGRF on the other hand works by randomly simulating scenarios and keeping track of the reactions that resulted in a high reward at the end of the simulation. [11] [12] provide deeper insight of LGRF. This can be expressed as shown in Fig. *3*.

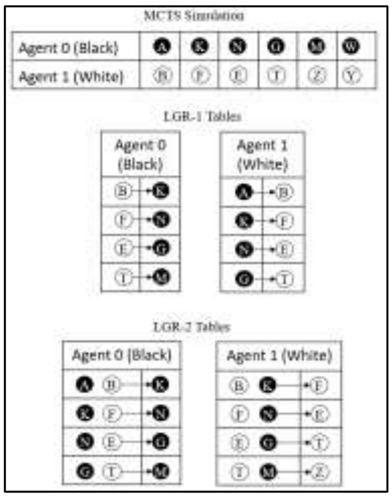

Fig. 3. Illustration of Last Good Reply with Forgetting (LGRF).

A full LGRF algorithm contains both LGR-1 and LGR-2 tables. However, the study applied only the LGR-1 table due to its speed, affordable computational cost, and low memory footprint. Hence the concept of lean-LGRF. UCT and lean-LGRF are thus combined as a variation to the selection stage in the designed algorithm as shown in (2).

$$\underset{i \in childrenOfTheExpandedNode}{\operatorname{argmax}} \left(\frac{w_i}{x(n_i)} + lean\_LGRF(n_0, n_i) + \sqrt{c \times \frac{\ln t}{x(n_i)}} \right) \qquad (2)$$

Where:
$\boldsymbol{lean\_LGRF(n_0, n_i)}$ is a function that returns the value of the node that led to the highest reward given that the previous node was $\boldsymbol{n_0}$ and the current node under consideration is $\boldsymbol{n_i}$. This implies that it is twice as valuable to select a node that leads to the state with the highest possible value.

2. *Variated Expansion Stage*

The expansion phase of the MCTS algorithm checks the selected node to determine if it has any unexplored child nodes. If an unexplored child node is found, it is added to the search tree. Previous studies have demonstrated implementation of the expansion stage by adding one node to the search tree at a time, or by adding multiple nodes at the same time as in multi-threaded architectures. This study variates the expansion stage by adding all the possible children of the unexplored child node before proceeding to the simulation stage.

3. *Variated Simulation Stage*

The simulation stage randomly simulates possible scenarios in a replica environment. Unlike the default MCTS algorithm which simulates one child at a time, the study variated the simulation stage by simulating all the children at the simulation stage. This was made possible due to the categorization of parameters into 3 tactics. The random simulations are continuously carried out until a predefined computational budget is reached. The randomness is applied during the simulation process in order to reduce the level of bias that can be introduced by a human-designed algorithm. The results of the simulations are then recorded and used to calculate the value of subsequent nodes along the tree. The study used a total of 1,000,000 simulations for the simulation stage.

4. *Backpropagation Stage*

The backpropagation stage involves returning the discrete reward value of each node (obtained through simulation) back up the search tree (backpropagation). Fig. *4* provides a graphical representation of the MCTS algorithm.

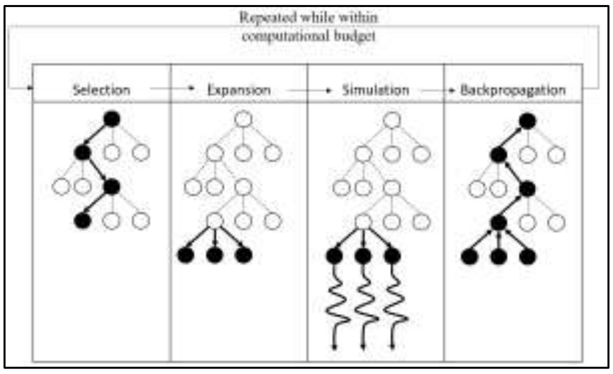

Fig. 4. Representation of the MCTS algorithm.

Algorithm 1 provides an overview of the basic Monte Carlo Tree Search (MCTS) algorithm. Algorithm 2 details the variated selection stage and the default expansion stage. Lastly, Algorithm 3 details the variated simulation and backpropagation stages of the MCTS algorithm.

Algorithm 1: The Basic Monte Carlo Tree Search Algorithm

```
Input  : The state, $s_0$, of the current root node, $n_0$ ($s(n_0)$)
Output: The action, $a$, that leads to the most optimum child
        node that has a desirable state
1 Function MCTS($s_0$)
2     create root node $n_0$ with state $s_0$;
3     while within computational budget do
          /* The SELECTEXPAND function is used to
             select a child node according to a predefined
             utility function: ''UCT lean-LGRF'' in the
             case of this study. This selected child node
             $n_i$ is then added to the search tree in the
             process of expansion. The variated expansion
             stage goes a step further to add all the 3
             children of the child node $n_i$.              */
4         $n_i \leftarrow SELECTEXPAND(s(n_0))$ ;
          /* The SIMULATEBACKPROPAGATE function
             is used to simulate possible scenarios from
             the newly expanded node $n_i$. A reward value
             is calculated by executing each of the
             possible actions available in the newly
             expanded node. This is then stored in $\Delta$ and
             backpropagated up the tree to update the node
             values.                                        */
5         $\Delta \leftarrow SIMULATEBACKPROPAGATE(s(n_i))$ ;
      /* The result of the overall search,
         $a(SELECT(n_0))$, is the action, $a$, that leads to
         the best child of the root node, $n_0$, where the
         exact definition of ''best'' is defined by the
         implementation. In the case of this study,
         ''best'' is the node with the highest discounted
         transaction throughput (for OLTP) or the least
         response-time latency (for OLAP).                  */
6     return $a(SELECT(n_0))$ ;
```

Algorithm 2: Variated Selection and Expansion Stages of the Monte Carlo Tree Search Algorithm

```
Input  : The state, $s_0$, of the current root node, $n_0$ ($s(n_0)$)
Output: The child node $n_i$ that has been selected through
        either exploitation or exploration. In addition to $n_i$, all
        its 3 children are also specified as output.
1 Function SELECTEXPAND($n_0$)
2     while $n_0$ is non-terminal do
3         if $n_0$ has been fully expanded then
4             return $\Delta \leftarrow SELECT(n_0)$ ;
5         else
6             return $EXPAND(n_0)$ ;

7 Function SELECT($n_0$)
      /* Determines the best child, $\Delta$, to select out of
         all the other possible children of node $n_0$.
         This is based on the value of the UCT lean-LGRF
         variation to the selection stage.                  */
8     return $\underset{i \in childrenOfTheRootNode}{\arg\max}\left(\frac{\gamma(n_i)}{x(n_i)} + lean\_LGRF(n_0, n_i) + \sqrt{c \times \frac{\log(t)}{x(n_i)}}\right)$ ;

9 Function EXPAND($n_0$)
10    perform $a \in$ untried actions from $A(s(n_0))$ ;
11    add new child $n_i$ to $n_{i-1}$
        with $a(n_i) \leftarrow a$
        and $s(n_i) \leftarrow f(s(n_{i-1}), a)$ ;
12    add all children of $n_i$ to $n_i$ ;
13    return $n_i$, children of $n_i$
```

Algorithm 3: Simulation and Backpropagation Stages of the Monte Carlo Tree Search Algorithm

```
Input  : The state, $s_i$, of the current node, $n_i$ ($s(n_i)$)
Output: The reward value of node $n_i$
1 Function SIMULATEBACKPROPAGATE($s(n_i)$)
2     while $A(s(n_i))$ is not empty and $x(n_i) < 1{,}000{,}000$ do
3         perform $a \in A(s(n_i))$ at random ;
4         $s(n_i) \leftarrow f(s(n_{i-1}), a)$ ;
          /* $x(n_i)$ is the number of times node $n_i$ has been
             traversed                                      */
5         $x(n_i) \leftarrow x(n_i) + 1$ ;
6         if result is a win then
              /* The number of simulated wins that have
                 occurred when node $n_i$ was traversed; where
                 a ''win'' is defined by the
                 implementation. In the case of this
                 study, ''win'' is the node with the
                 highest discounted transaction throughput
                 (for OLTP) or the least response-time
                 latency (for OLAP).                        */
7             $w_i \leftarrow w_i + 1$ ;
8     SIMULATEBACKPROPAGATE(for all the children of $n_i$) ;
9     BACKPROPAGATE($n_i$) ;

10 Function BACKPROPAGATE($n_i$)
11    while $n_i$ is not null do
12        update $rewardValueOfNode$ $n_i$ ;
          /* $w_i$ = discounted transaction throughput (for
             OLTP) or the least response-time latency (for
             OLAP) for node $n_i$ and all of its children   */
13        update $w_i$ ;
          /* $x_i$ = number of traversals(visits) made on
             node $n_i$ and all of its children             */
14        update $x_i$ ;
15        update $rewardValueOfNode$ $n_i = \frac{\gamma(n_i)}{x(n_i)}$ ;
16        $n_i \leftarrow$ parent of $n_i$ ;
```

A. Algorithm Correctness

For an argument to be considered valid, the conclusion must always coherently follow from the premises. In other words,

$premise \rightarrow conclusion$ (if premise, then conclusion)

The algorithm can similarly be distilled to the form of a conditional in order to prove it.

Theorem: If s_0 is the current root node, then the action a recommended by the variated MCTS algorithm is the action that leads to the most optimum child node that has a desirable state.

$$s_0 \rightarrow a_0$$

Proof: Assume that the most optimum child node that has a desirable state, s_1, is the node that has the maximum value over all other nodes at time $t+1$, that is, $\max_{t+1}(\forall s) = s_1$. This means that the action that led to the root node at time $t+1$, is defined by the action to be taken when the current root node was s_0, at time t_0. That is, $q_*(s_0, a_0)$ where q_* represents the variated MCTS algorithm, will result in s_1. Therefore, $q_*(s_0, a_0) \rightarrow s_1$ and $q_*(s_1, a_1) \rightarrow s_2$. Thus $s_0 \rightarrow a_0$ and $s_1 \rightarrow a_1$ □

The variated MCTS algorithm has a stop property implemented by a computational budget which is based on a pre-defined number of simulations (1,000,000 simulations). This can be observed by the fact that the number of simulations is defined as a constant, finite integer.

The variated MCTS algorithm also has a partial correctness property. This can be demonstrated by showing that "x" is a maximum in Array.

$$\left(\forall_{0\le j<simulations}(x \ge Array[j])\right) \wedge \left(\exists_{0\le j<simulations}(x == Arr[j])\right)$$

Using a loop invariant in conjunction with the stop property:

$$\left(\forall_{0\le j<s}(x \ge Array[j])\right) \wedge (s == simulations) \Longrightarrow \left(\forall_{0\le j<simulations}(x \ge Array[j])\right)$$

Therefore, proving the total correctness property present in the algorithm

B. *Algorithm Time and Space Complexity*

The theoretical analysis of the algorithm shows that the Big-O notation of the algorithm's time complexity is linear, that is, the algorithm's time complexity has an order of n: $O(n)$. On the other hand, space complexity is a measure of the amount of working storage an algorithm requires. The main concern is how space requirements grow in Big-O terms as the size of n (input) grows. The Big-O notation of the algorithm's space complexity is therefore also linear. That is, the algorithm's space complexity has an order of n: $O(n)$.

C. *Empirical Algorithmics*

Empirical algorithmics supports the acquisition of insights into the behavior of a designed algorithm. The experiment procedure outlined in Section III. A was applied and TABLE *1* presents the results.

TABLE 1. Empirical algorithmics results.

	Default	**Low**	**Medium**	**High**	**Adaptive (Automatically switches between high, medium, and low based on the MCTS algorithm) (μ)**
Average Transaction Throughput (Transactions Per Second)	4,958.46 (σ = 350.74)	5,272.92 (σ = 210.94)	5,413.56 (σ = 303.06)	5,784.91 (σ = 435.78)	5,812.75 (σ = 249.11)
Average Response-Time Latency (Microseconds)	3.50 (σ = 0.61)	3.04 (σ = 0.19)	3.02 (σ = 0.14)	3.07 (σ = 0.38)	2.56 (σ = 0.50)
Maximum & Minimum Transaction Throughput (Transactions Per Second)	5,298.36 and 2,935.70	5,648.19 and 4,809.30	5,912.09 and 4,721.93	6,074.99 and 3,665.23	6,454.89 and 5,379.13
Maximum & Minimum Response-Time Latency (Microseconds)	6.00 and 3.00	4.00 and 3.00	4.00 and 3.00	5.00 and 3.00	3.00 and 2.00

V. DISCUSSION OF RESULTS

The results presented in Section IV. E provide evidence that 100% of the experiment results conducted with the algorithm running resulted in a transaction throughput that was higher than the average transaction throughput in an environment running using the default configurations. The results presented above also provide evidence that 100% of the experiment results conducted with the algorithm running resulted in a response-time latency that was lower than the average response-time latency in an environment running using the default configurations. The decision rule can therefore be applied to reject the null hypothesis and state that there is no statistical evidence to reject the 1st and 2nd alternative hypotheses.

The probability of finding that the average transaction throughput for distributed database systems that apply the designed autonomic latency-aware algorithm is greater than the average transaction throughput for distributed database systems that do not apply the designed autonomic latency-aware algorithm given that the null hypothesis is true is less than 0.05.

$$P(\mu > \mu_{H_0} \mid H_0 \text{ is true}) < 0.05$$

This can be standardized using a t distribution such that the area under the t distribution curve that is greater than a t score of 1.705618 given that there are 26 degrees of freedom represents the 5% probability of occurring. Therefore, if a T-score is greater than 1.705618, then we cannot continue to claim that the null hypothesis is true. In such a case, we should reject the null hypothesis in conformance to the decision rule.

Reject H_0 if $\text{T score}_{\text{calculated}} > \text{T score}_{\text{tabular}}$. Given that $\text{T score}_{\text{calculated}} = 17.81918823$ and $T\ score_{tabular} = 1.705618$, then the null hypothesis should be rejected in favor of the fact that "distributed database systems that apply the designed autonomic latency-aware algorithm on average have a faster transaction throughput".

The probability of finding that the average response-time latency for distributed database systems that apply the designed autonomic latency-aware algorithm is less than the average transaction throughput for distributed database systems that do not apply the designed autonomic latency-aware algorithm given that the null hypothesis is true is also less than 0.05.

$$P(\mu < \mu_{H_0} \mid H_0 \text{ is true}) < 0.05$$

The area under the t distribution curve that is less than a t score of -1.705618 given that there are 26 degrees of freedom represents the 5% probability of occurring. Therefore, if a T-score is less than -1.705618, then we cannot continue to claim that the null hypothesis is true. The H_0 should be rejected if $\text{T score}_{\text{calculated}} < \text{T score}_{\text{tabular}}$. Given that $\text{T score}_{\text{calculated}} = -9.738191311$ and $T\ score_{tabular} = -1.705618$, then the null hypothesis should be rejected in favor of the fact that "distributed database systems that apply the designed autonomic latency-aware algorithm on average have a slower response-time latency."

The results therefore provide no statistical evidence to reject the hypothesis that "distributed database systems that apply the designed autonomic latency-aware algorithm on average have a faster transaction throughput and a slower response-time latency."

VI. Conclusion and Recommendations

A. *Conclusion*

Through deductive reasoning, the study was able to move from existing theoretical knowledge, to formation of testable hypotheses, and lastly to either rejection of the hypotheses as a fallacy or acceptance of the hypotheses as the truth by confronting them with factual data. The confrontation with factual data made possible through the use of observations in the form of experiments. The experiment results presented evidence to accept the following hypothesis as the truth: "distributed database systems that apply the designed autonomic latency-aware algorithm on average have a higher transaction throughput and a lower response-time latency". In addition to this, Section IV. B provided an answer to the first research question whereas Section III. provided an answer to the second research question.

A. *Recommendations for IT Practitioners*

The study recommends that if a business can automate the response to a technical system issue, by all means, it should consider doing so. This is in relation to the high cost of calling system administrators to address the issue during non-working hours. As demonstrated in the study, this can be made possible through the use of Artificially Intelligent (AI), Reinforcement Learning (RL), algorithms that are based on a variated Monte Carlo Tree Search.

It is also important to note that not all technical system issues are emergencies. It is not worth alerting system administrators on technical system issues that are only internally noticeable and that may revert to normal levels without intervention. If not careful, alert fatigue can cause a serious issue to be ignored. However, in the event that it is a genuine emergency and the system administrators need to be informed urgently, it is preferable to inform them on symptoms and not causes of the symptoms. Symptoms in this case are a manifestation of issues that may have a number of different causes. The study recommends that symptoms related to work metrics (high-level observations) should be categorized into either unacceptable levels of throughput, or response-time latency, or workload-related error-rates. The notification of the symptoms must then be followed by a diagnosis to identify the root cause or causes.

The study further recommends that resource metrics and event metrics should be collected periodically to support the reconstruction of detailed snapshots of the system's state. It is these snapshots that can then be used to diagnose the cause of the symptoms identified through work metrics. Resource metrics should be based on the following resources: compute (CPU), primary storage/memory (RAM), secondary storage (HDD/SSD), and network interfaces. The symptoms collected for these resources should subsequently be categorized into either unacceptable levels of resource availability, or resource utilization, or resource saturation, or resource-related error-rate. On the other hand, event metrics automatically capture an action or occurrence that originates asynchronously from the external environment and is recognized by the system. For example, internally generated alerts that communicate the status of critical work such as completion of a task, for example, data backup. The definition of what can be considered as critical work is dependent on the system administrator.

Once the work metrics have been observed, and the resource and event metrics used to gain further insight, then the next stage should be to diagnose the cause of the symptoms observed. This is often the least structured stage and is largely driven by subjective decisions. IT practitioners should also use the observations to make a prognosis of the likely course of the system's state and to be proactive. The study therefore submits the use of self-optimization techniques as a means to standardize the diagnosis and prognosis stages of performance tuning.

The study recommends the channeling of OLAP and OLTP workloads to different servers. This can work best using the architecture described in Section III. B. In such a case, either the load balancer or the actual business information system can be used to channel OLAP workloads to a specific member(s) of the cluster and OLTP workloads to another member(s) of the cluster. The automatic latency-aware algorithm should then run on each member of the cluster and configure the node based on the characteristic workload it serves. This recommendation is made in view of the complexities of majority business information systems that have a hybrid of OLTP and OLAP systems. It will be counterproductive to readjust configurations based on dynamic and unpredictable characteristics of workloads because it is often the case that OLTP-friendly configurations contradict OLAP-friendly configurations and vice versa.

Lastly, given the numerous and diverse configuration values of parameters, the study recommends categorizing parameter values into tactics. Implementing one tactic can therefore involve the implementation of numerous configuration parameters that belong to the tactic. Working with tactics as opposed to individual parameters supports the construction of a reasonable decision tree required by AI-based algorithms that are required to make complex decisions.

References

[1] D. Autor, "Why are there still so many jobs? The history and future of workplace automation," *The Journal of Economic Perspectives*, vol. 29, no. 3, pp. 3-30, 2015.

[2] J. W. Kim, S. H. Cho, and I. M. Kim, "Workload-Based column partitioning to efficiently process data warehouse query.," *International Journal of Applied Engineering Research*, vol. 11, no. 12, pp. 917-921, 2016.

[3] K. Lee, A. C. König, V. Narasayya, B. Ding, S. Chaudhuri, B. Ellwein, A. Eksarevskiy, K. M. J. Wyant, P. Prakash, R. Nehme, J. Li, and J. Naughton, "Operator and query progress estimation in Microsoft SQL Server live query statistics," in *2016 International Conference on Data*, 2016.

[4] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *ACM International Conference on Management of Data*, 2017.

[5] A. O. Omondi, I. A. Lukandu, and G. W. Wanyembi, "Scalability And Nonlinear Performance Tuning In Storage Servers," *International Journal Of Research Studies In Science, Engineering and Technology*, vol. 5, no. 9, pp. 7-18, 2018.

[6] G. Rao, N. V. E. S. Murthy, and G. L. Devi, "A Theoretical Model to Provide Security for Remote Location Aware Cloud Data Centre," *International Journal of Scientific Research in Computer Science,*

Engineering and Information Technology, vol. 3, no. 2, pp. 232-236, 2018.

[7] X. Zhang, H. Abbasi, K. Huck, and A. D. Malony, "WOWMON: A machine learning-based profiler for self-adaptive instrumentation of scientific workflows," *Procedia Computer Science*, vol. 80, pp. 1507-1518, 2016.

[8] G. D. C. Rodrigues, G. L. dos Santos, V. T. Guimaraes, L. Z. Granville, and L. M. R. Tarouco, "An architecture to evaluate scalability, adaptability and accuracy in cloud monitoring systems," in *International Conference on Information Networking (ICOIN)*, 2014.

[9] Nataraj, A. D. Malony, A. Morris, D. C. Arnold and B. P. Miller, "A framework for scalable, parallel performance monitoring," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 720-735, 2010.

[10] O. L. I. A. &. W. G. W. Omondi, "A selection variation for improved throughput and accuracy of Monte Carlo tree search algorithms," *International Journal of Computer and Information Technology*, vol. 7, no. 6, pp. 286-294, 2018.

[11] J. A. Stankiewicz, M. H. M. Winands and J. W. H. M. Uiterwijk, "Monte-Carlo tree search enhancements for Havannah," in *Advances in Computer Games*, 2011.

[12] H. Baier and P. Drake, "The power of forgetting: improving the last-good-reply policy in Monte Carlo Go," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 303-309, 2010.

APPENDIXES

APPENDIX A: STEPWISE REGRESSION RUNS

The following is a list of the features and their corresponding identification:

x_1: innodb_buffer_pool_size
x_2: innodb_buffer_pool_instances
x_3: innodb_old_blocks_pct
x_4: innodb_old_blocks_time
x_5: innodb_buffer_pool_dump_at_shutdown
x_6: innodb_buffer_pool_load_at_startup
x_7: query_cache_size
x_8: innodb_log_file_size
x_9: innodb_file_per_table
x_10: innodb_lock_wait_timeout
x_11: thread_cache_size
x_12: tmp-table-size
x_13: max-heap-table-size
x_14: sort-buffer-size
x_15: join_buffer_size

OLTP workload submitted concurrently by 20 users

1st Run: y_1 = b_0 + b_1.x_3

y_1 regressed only on:	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_10	x_11	x_12	x_13	x_14	x_15
b_1	3.56E-07	-1.16E+01	3.13E+01	3.87E-02				1.85E-06			1.23E-02	-4.28E-06		-6.99E-05	
t-statistic	8.16E-01	-1.49E+00	2.40E+00	7.40E-02				2.84E-01			3.75E-01	-8.41E-01		-2.06E-01	
ABS(t-statistic)	8.16E-01	1.49E+00	2.40E+00	7.40E-02				2.84E-01			3.75E-01	8.41E-01		2.06E-01	
p-value	4.34E-01	1.66E-01	3.75E-02	9.43E-01				7.82E-01			7.15E-01	4.20E-01		8.41E-01	

2nd Run: y_1 = b_0 + b_1.x_3 + b_2.x_12

y_1 regressed only on:	x_3 + x_1	x_3 + x_2	x_3 + x_3	x_3 + x_4	x_3 + x_5	x_3 + x_6	x_3 + x_7	x_3 + x_8	x_3 + x_9	x_3 + x_10	x_3 + x_11	x_3 + x_12	x_3 + x_13	x_3 + x_14	x_3 + x_15
b_2	1.44E-07	-8.67E+00		-4.31E-01				1.69E-07			-3.24E-04	-5.74E-06		1.80E-05	
t-statistic	3.68E-01	-1.28E+00		-9.39E-01				3.00E-02			-1.10E-02	-1.42E+00		6.20E-02	
ABS(t-statistic)	3.68E-01	1.28E+00		9.39E-01				3.00E-02			1.10E-02	1.42E+00		6.20E-02	
p-value	1.21E-01	6.12E-02		8.52E-02				1.30E-01			1.30E-01	5.20E-02		1.30E-01	

3rd Run: y_1 = b_0 + b_1.x_3 + b_2.x_12 + b_3.x_2

y_1 regressed only on:	x_12 + x_3 + x_1	x_12 + x_3 + x_2	x_12 + x_3 + x_3	x_12 + x_3 + x_4	x_12 + x_3 + x_5	x_12 + x_3 + x_6	x_12 + x_3 + x_7	x_12 + x_3 + x_8	x_12 + x_3 + x_9	x_12 + x_3 + x_10	x_12 + x_3 + x_11	x_12 + x_3 + x_12	x_12 + x_3 + x_13	x_12 + x_3 + x_14	x_12 + x_3 + x_15
b_3	3.50E-08	-7.29E+00		-3.79E-01				-6.32E-07			-6.57E-03			5.98E-05	
t-statistic	9.10E-02	-1.09E+00		-8.56E-01				-1.18E-01			-2.40E-01			2.16E-01	
ABS(t-statistic)	9.10E-02	1.09E+00		8.56E-01				1.18E-01			2.40E-01			2.16E-01	
p-value	1.35E-01	8.14E-02		9.84E-02				1.35E-01			1.32E-01			1.33E-01	

<u>OLAP workload submitted concurrently by 20 users</u>

1st Run: y_2 = b_0 + b_1.x_1

y_2 regressed only on:	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_10	x_11	x_12	x_13	x_14	x_15
b_1	-3.33E-10	-1.64E-03	-3.13E-03	-1.00E-04				8.42E-10			5.72E-06	6.77E-10		-4.37E-08	
t-statistic	-1.94E+00	-4.30E-01	-4.30E-01	-4.30E-01				2.89E-01			2.89E-01	2.89E-01		7.79E-01	
ABS (t-statistic)	1.94E+00	4.30E-01	4.30E-01	4.30E-01				2.89E-01			2.89E-01	2.89E-01		7.79E-01	
p-value	8.16E-02	6.76E-01	6.76E-01	6.76E-01				7.79E-01			7.79E-01	7.79E-01		7.79E-01	

2nd Run: y_1 = b_0 + b_1.x_1 + b_2.x_8

y_2 regressed only on:	x_1 + x_1	x_1 + x_2	x_1 + x_3	x_1 + x_4	x_1 + x_5	x_1 + x_6	x_1 + x_7	x_1 + x_8	x_1 + x_9	x_1 + x_10	x_1 + x_11	x_1 + x_12	x_1 + x_13	x_1 + x_14	x_1 + x_15
b_2		-3.51E-03	3.34E-17	7.12E-19				4.63E-09			-3.71E-20	-4.39E-24		4.53E-22	
t-statistic		-1.04E+00	0.00E+00	0.00E+00				1.73E+00			0.00E+00	0.00E+00		0.00E+00	
ABS (t-statistic)		1.04E+00	0.00E+00	0.00E+00				1.73E+00			0.00E+00	0.00E+00		0.00E+00	
p-value		1.43E-01	2.39E-01	2.39E-01				6.54E-02			2.39E-01	2.39E-01		2.39E-01	