International Journal of Research Studies in Science, Engineering and Technology
Volume 5, Issue 9, 2018, PP 7-18
ISSN : 2349-476X

Scalability and Nonlinear Performance Tuning in Storage Servers

Allan Odhiambo Omondi[1], Ismail Ateya Lukandu[1], Gregory Wabuke Wanyembi[2]

[1]Faculty of Information Technology, Strathmore University, Nairobi, Kenya
[2]Department of Information Technology, Mount Kenya University, Thika, Kenya

***Corresponding Author:** Allan Odhiambo Omondi, Strathmore University, Kenya. Email:aomondi [at] strathmore.edu*

ABSTRACT

The ability to scale in order to handle an increased amount of work is critical to both a business and its Information Technology. However, there are diminishing returns as the number of scalability enhancements are increased. This article submits the fact that there is a need to consider performance tuning as opposed to only horizontal or vertical scalability. Optimization techniques based on a greedy algorithm are reviewed. For example, Constraint Programming, Mixed Integer Programming, and Local Search. The article further indicates that the complexities of performance tuning do not favor modeling it as a linear process. It proposes the use of evidence-based, probabilistic reasoning for system administrators to identify which parameters need to be tuned and how to tune them. An influence diagram is used as an example to show the complexities involved when tuning one parameter has a high probability of affecting many other parameters in the system.

Keywords: *Maria DB, Influence Diagrams, Probabilistic Reasoning, Decision-Making, Optimization, Greedy algorithms, Automation, Distributed databases*

INTRODUCTION

Automation forms an important foundation to make progress in the evolution of human beings. It saves on human resource as it reduces labor requirements per unit of output produced. This saved human resource can then be applied in other fields towards further advancement. With this in mind, there has been a continuous demand for systems that can automatically manage themselves given high-level objectives from system administrators. This has led to the advent of autonomic computing (Kephart & Chess, 2003).Autonomic computing systems in this case are systems that are expected to automatically adapt by reacting to variable environmental conditions and runtime phenomena. This can be achieved through adjusting their configuration parameters as an adaptive response in order to discover the most optimum configuration given the characteristics of the current workload. This study therefore seeks to answer the following two research questions:

- *What are the merits and demerits of optimization techniques that can be used to achieve load scalability in servers?*
- *What are the parameters that need to be considered when performing configuration optimization in storage servers?*

Section 1 of this article provides a Literature Review that focuses on the theoretical concepts of scalability and performance tuning. Section 2 then provides a description of the methodology that was applied to obtain the results. It does this by providing a description of the high-level architecture of the Maria DB Galera synchronous, multi-master distributed database that was used.

The methodology Section concludes with a description of how a workload generator was used to create training and test data. Section 3 provides the results as well as a discussion of what the results mean.

The study submits a probabilistic reasoning approach as opposed to the existing linear optimization techniques. An influence diagram is then used to model a stochastic, nonlinear environment of the key configurations that should be considered when conducting performance tuning of storage servers. Section 4 concludes the article with a discussion of the merits and demerits of automation.

LITERATURE REVIEW

Scalability

Scalability can be defined as the capability of a system to be enhanced to handle an increased amount of work (Shahapure & Jayarekha, 2014). Author (2015) adds on to say that this is done without impacting on its responsiveness to execute any action within a given time interval.

The increased amount of work in the context of a business can in turn be caused by many factors including a peak season of sales or venturing into a new market that increases the number of clients to serve. It can also be based on an expansion of the business by adding an administrative unit that is either in the same premise or in a geographically separate premise. Enhancements of the system are usually made by the system administrators and depend on the system. For example, how a database server is enhanced is different from how an application server or a web server is enhanced. These enhancements are geared towards increasing the performance of a system which in turn enables it to handle an increased amount of work. The performance is in turn based on maximizing the throughput and minimizing the response time. It is also based on maximizing the resilience to faults which can be achieved by not overloading the system. Load balancing, especially in the context of a distributed system, plays a critical role in this case (Anjum & Patil, 2017).

There are two common techniques of enhancing a system to accommodate a growing amount of work: horizontal scaling and vertical scaling. Horizontal scaling is the most common scaling technique and is based on adding or removing nodes from a system. As noted in a study on cloud ecosystems by Anjum and Patil (2017), the nodes can be in the form of Virtual Machines (VMs) as opposed to physical machines. In the case of physical machines, system administrators can configure hundreds of physical machines into a cluster to obtain an aggregate computing power. This aggregate computing power often exceeds that of computers based on a single, traditional processor. Horizontal scaling is however reliant on a high-performance network such as Gigabit Ethernet, Infinity Band, and Myrinet, amongst others. Vertical scaling on the other hand, maintains the number of nodes at a constant level and instead adds or removes the amount of resources that are allocated to each node. Fig shows the difference between the two types of scaling.

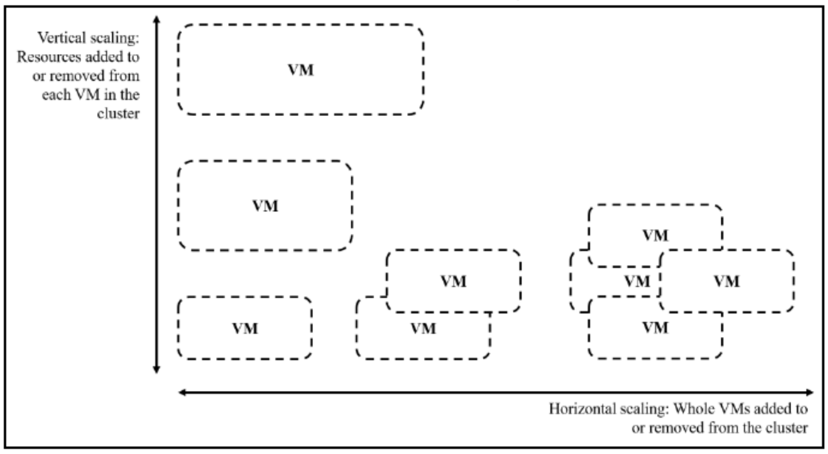
Vertical scaling: Resources added to or removed from each VM in the cluster
VM
VM
VM
VM
VM
VM
VM
VM
VM
Horizontal scaling: Whole VMs added to or removed from the cluster

Figure1. *Horizontal scaling versus vertical scaling*

Vertical and horizontal scalability can in turn be grouped into different perspectives depending on what is being addressed. The first perspective, administrative scalability, addresses the capability of a system to maintain an acceptable level of throughput, response time, and resilience as the business expands across numerous departments. If the expansion is across multiple geographical regions, for example in the case of new market entry, then it is addressed by the perspective of geographic scalability. The third perspective, functional scalability, addresses the ability to enhance a system by adding new functional requirements at minimal effort. This article focuses on the fourth perspective, which is load scalability. Load scalability addresses the ability of a system to expand or contract its resource pool to accommodate a changing load. The resource pool can be in the form of memory in a shared memory architecture, or in the form of storage in a shared storage architecture. Scalability is a significant issue not only in technology, but also in business. The number of clients demanding for services has the potential of increasing in cases where clients consider the services offered by a business as valuable. Information Technology applied in Computer-Based Information Systems acts as a critical enabler to assist a business to cope with the growing demand for services. An increase in the number of clients is good for business growth and stability. Therefore, businesses in various sectors of an economy desire the ability to handle a growing number of clients; that is, the ability to scale. Inability to cope with growth can lead to loss of business (Rao, Murthy, & Devi, 2018). The enabling infrastructure of a business, usually in the form of cloud computing, is expected to also scale to support the business. It is therefore important for both technology and businesses to be scalable.

Scalability is directly proportional to the amount of energy consumed. This is especially true for vertical scaling which results in high server density in data centers. The increase in server density has also been championed by manufacturers who convince their clients to utilize every available space in their data centers given the Total Cost of Ownership (TCO) required to maintain a data center. The TCO in this case is distributed across three subsystems: Information Technology (IT), power, and cooling (Gomes, et al., 2017). The increase in server density with the aim of scalability has led to the energy consumption in data centers to be quadrupled over the last decade. In fact, a study by Afonso and Moreira (2017) showed that a data center with 2,000 m^2 and an energy consumption density of 1,000 Wm^{-2} has a peak cooling subsystem consumption that is equivalent to a 20,000 m^2 office building. And this was the case without considering the IT and power subsystems. Data centers consume 3% of global electricity production and account for 200 million metric tons of CO_2 (Rallo, 2014). Given that energy efficiency is a priority for competitiveness, there is a need for businesses to achieve scalability in an energy efficient manner.

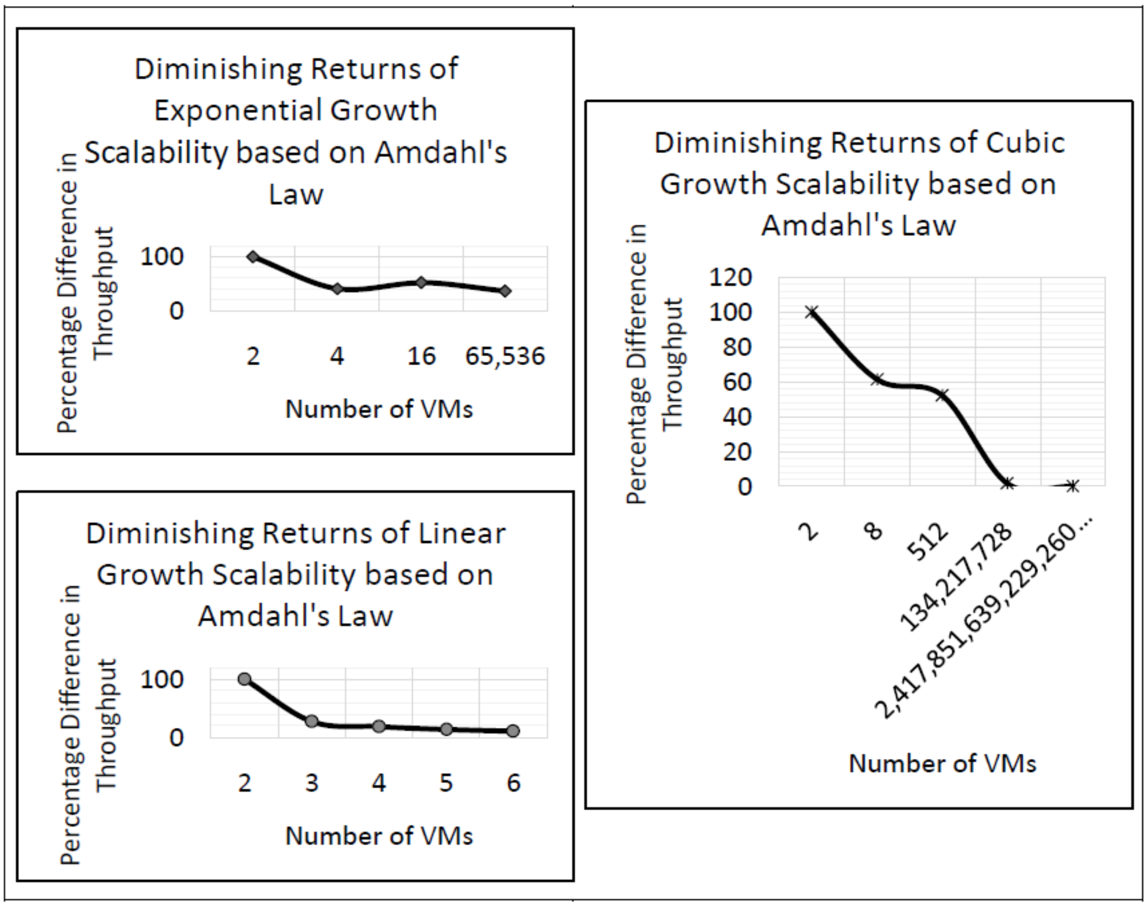

Figure 2. *Diminishing returns based on Amdahl's Law*

The marginal productivity or benefits of a system gradually diminishes as the amount of investment or enhancements are increased. This is referred to as the law of diminishing returns and is well-defined by Amdahl's Law. The addition of VMs in a cluster or the addition of resources assigned to each VM in a cluster tends towards either horizontal scalability or vertical scalability respectively. A system can handle an increase in the amount of work by distributing portions of the work amongst either VMs in the cluster or amongst the added resources in each

VM in the cluster. Suppose that a system can handle an increase in workload by distributing $\frac{7}{10}$ of a program amongst VMs for parallel processing instead of sequential processing in one VM. If α is the fraction of the program that is sequential and $1-\alpha$ is the fraction of the program that can be parallelized, then the maximum throughput that can be achieved by x units of VMs can be defined by Amdahl's Law as shown in (1).

$$\frac{1}{\alpha+\frac{1-\alpha}{x}} \tag{1}$$

A graphical representation of this while holding all other factors constant shows a clear diminishing return as the number of scalability enhancements increases exponentially, in a cubic manner, or in a linear manner as shown in Figure .

Performance Tuning

Performance tuning involves the implementation of activities geared towards improving a system's capability of meeting its non-functional requirements. The non-functional requirements in this case include performance that can be quantitatively measured in form of response time latency and in the form of transaction throughput. Developers of complex systems intentionally expose a set of configuration parameters to system administrators. As explained by Sullivan (2003), this is done because of the realization that a system experiences numerous changes in environmental variables as well as changes in runtime phenomena which includes unexpected input in form of a workload. The study further implied that leaving a system to run without continuously changing its default configurations either reactively or proactively is not good because it prevents it from handling unexpected work loads (Sullivan, 2003). In other words, no single configuration of a system caters for every possible workload. Systems need to be reconfigured or tuned from time to time to enable them to adapt to constantly changing environmental variables and runtime phenomena.

Performance tuning is motivated by a performance problem which is in turn based on identifying a slow or unresponsive system. It is this performance problem that acts as a starting point for the measure-evaluate-improve-learn cycle of quality assurance. It is important to clearly identify the root cause of a problem before attempting to address it (Dostál, 2014). It is this early identification that provides the solver with the required focus and saves on time by ensuring the right thing is done (effectiveness). The root cause of slow or unresponsive systems is overloading which causes some components of the system to reach a limit in their ability to respond to subsequent loads. Other components of the system may remain idle as they wait for the overloaded component to perform its task. This overloaded component, the root cause of a performance problem, is referred to as a bottleneck.

It is important to objectively identify the part of a system that is critical for improving performance. The most critical sections of the system in this case are the bottlenecks, which can be determined by using a profiler. A profiler supports dynamic program analysis which is conducted by instrumenting either the program source code or its binary executable and measuring Key Performance Indicators (KPIs) at runtime.

This is as opposed to qualitatively and subjectively guessing the bottlenecks of a system. This study applied a periodic (non-intrusive) sampling to measure KPIs. Common measurable KPIs include space and time complexity, and frequency and duration of function calls. Zhang, Abbasi, Huck, and Malony (2016), Rodrigues, dos Santos, Guimaraes, Granville, and Tarouco (2014), and Nataraj, Malony, Morris, Arnold, and Miller (2010) showed that statistical profilers are less intrusive to the target program because they rely on periodic sampling.

This allows the target program to run at near full-speed thus supporting the acquisition of a more accurate picture of the target program's execution as well as its bottlenecks.

A performance problem motivates the need to decide which parameters should be reconfigured and how they should be reconfigured. Any problem regarding decision-making involves the task of choosing "the best" amongst alternatives. The measure of goodness can be based on a pre-defined objective such that the decision made is the one that maximizes or minimizes this objective.

For example, if the objective is to enable a server to handle 200,000 queries per second, then the best decision will be the one that maximizes the number of queries per second.

This objective can be generalized to maximizing efficiency and minimizing the error-rate.

According to Van Aken, Pavlo, Gordon, and Zhang (2017) the efficiency can be quantitatively measured based on response-time latency and transaction throughput in the case of storage servers. Response-time latency was defined as the speed at which a storage server can respond to a read request (commonly associated with On-Line Analytical Processing [OLAP] workloads). Transaction throughput on the other hand was defined as the speed at which a storage server can collect new data through write requests (commonly associated with On-Line Transaction Processing [OLTP] workloads). Using a performance problem as the foundation, it is possible to model the objective as an optimization problem as shown in (2). The following steps were applied to ensure agreeability of the defined optimization problem:

Table1. *Modelling steps.*

Step	Description
I	Find out what are the decision variables (something that captures the real decisions you are interested in, i.e. what to decide (what you will decide upon))
II	Model the **problem constraints** (tells you what you can do and what you cannot do – essentially defines what can be accepted as a feasible solution, i.e. defines what is a solution)
III	Define the **objective function** (defines what you are trying to maximize or minimize, i.e. defines the quality of your solution

Maximize:	$\sum_{p\in T}\left(\frac{(t_p)y_w + (r_p)z_w}{e_p + a_p}\right)x_p$	(2)
Subject to:	$\sum_{p\in T} V_{pi}x_p \leq \mathrm{K_i}$	
Such that:	$\mathrm{x_p},\ y_w,\ z_w \in \{0,1\},\ (p \in T)$	

This implies that the performance is maximized subject to pre-defined constraints. The pre-defined constraint specifies that if a parameter, p, causes the value,V of a specific hardware i (V_{pi}),to change, then the change should be less than or equal to what the server hardware is capable of handling for the hardware under consideration, K_i.This is on condition that the decision to reconfigure parameter, p, that affects hardware i has been made, that is, on condition that$x_p = 1$. If it has not been chosen, then $x_p = 0$. Performance in such a system can be measured by transaction throughput (t_p) and response time latency (r_p). If the optimization is meant for OLTP workloads, then the value of y_w will be 1 and z_w will be 0. On the other hand, if the optimization is meant for OLAP workloads, then the value of z_w will be 1 and y_w will be 0.

The optimization problem goes a step further to define the negative effect that changing parameter p has on other parameters, that is, e_p. Lastly, a_p represents the adaptation latency of parameter p. The equation implies that to maximize the performance, both e_p and a_p should be kept at minimum levels.

Methodology

Experiment Test Bed

The test-bed was made up of a cluster of nodes that formed a synchronous, multi-master distributed database with high-persistence features based on a shared-nothing architecture. The shared-nothing architecture enabled the system to work with inexpensive hardware that met at least the minimum requirements for a Maria DB Galera Cluster. The shared-nothing architecture also contributed towards the elimination of a single-point-of-failure because each node in the cluster had its own inexpensive combination of CPU, storage, and memory. Fig below depicts the architecture of the test bed.

Training and Testing Data

Spawner workload generator was used to create the dataset. 75% of the dataset was used as training data. The training data was fed into the test bed in order to simulate a real-world scenario. Periodic performance measurements were then conducted in favor of continuous performance measurements which can negatively impact the performance of a system in production. The results of the periodic performance measurements were then used to identify the vital parameters that should be involved in the influence diagram.

The remaining 25% of the dataset was used as test data. The test data was required to confirm intuitive and learned notions of causality. This was accomplished through maximum likelihood estimation by first discretizing continuous variables, and then recording the frequency of occurrence between the values of a node and the combination of the values of its parent(s). The 75:25 training:test data ratio has been used successfully in previous research such as by Feng, et al. (2016) and Shinde and Channe (2018).

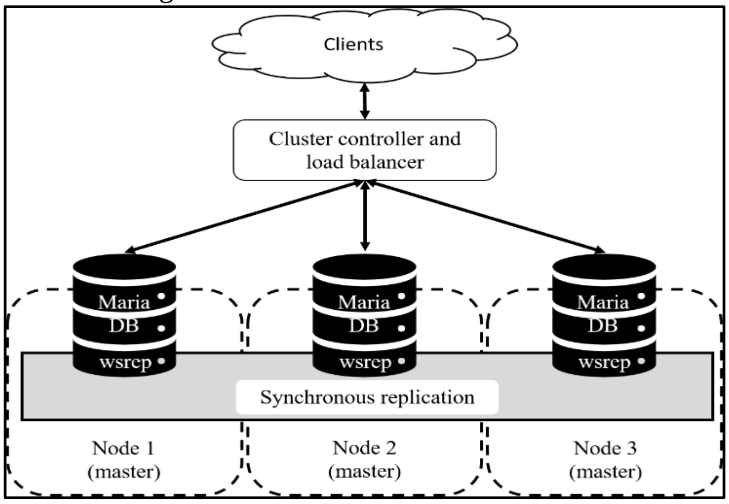

Figure 3. *Architecture of the experiment's test bed*

RESULTS AND DISCUSSION

Part of the following Sections provide a review of literature on the most common optimization techniques. These are based on greedy algorithms and the branch & bound and relaxation concepts. Constraint Programming as well as Mixed Integer Programming are reviewed as techniques that guarantee high quality. Local Search on the other hand is reviewed as a technique that guarantees scalability. The last Section then submits an approach that is based on probabilistic reasoning. The key advantage of its ability to model a stochastic, non-linear environment is highlighted.

Greedy Algorithms

Greedy algorithms make a locally optimal choice with the hope that this choice will lead to a globally optimal solution. They are easy to design (for simple problems) and they can arrive at a locally optimal choice within a short period of time (Qian, Yu, & Tang, 2018). However, greedy algorithms sometimes fail to find the globally optimal solution because they make commitments to certain choices too early which prevents them from finding the best overall solution later.

There are numerous improvements to the traditional, pure greedy algorithm. Two such improvements are the addition of the branch & bound concept and the relaxation concept (Ma & Liu, 2016). Decision-making problems involve the task of choosing "the best" amongst alternatives. Consequently, the act of choosing involves the concept of searching through numerous alternatives depending on the problem. These numerous alternatives can be organized in the form of a tree, hence the concept of a "tree search". It is possible (although computationally expensive) to conduct an exhaustive tree search in the process of finding the most optimum choice to make. However, the branch &bound concept improves on this by applying pruning to focus only on the most promising area of the search tree (it reduces the search space). The branching splits the problem into several sub problems while the bounding finds an optimistic estimate of the sequence of choices made.

On the other hand, the concept of relaxation involves making the problem easier to solve. It is through relaxation that a bigger portion of the search tree can be pruned before applying the branch & bound concept. The following three Sections describe further improvements to the traditional, pure greedy algorithm that apply branch & bound as well as relaxation.

Constraint Programming

Constraint Programming (CP) is a paradigm that defines the process of optimizing an objective function with respect to some variables in the presence of constraints. These constraints are in the form of hard limits placed on the value of a variable. For example, (2) limits the possible values of a hardware's configuration by stating that it cannot be above what that hardware can handle. It therefore constrains the possible values that can be assigned during the process of optimization. This can be represented graphically using a search engine and a constraint store as shown in Figure. below. There is continuous interaction between the search engine and the constraint store. The search engine continuously probes the constraint store to check if the value it has found for a variable is within the limits. Given adequate time to continuously probe, CP will find an optimal solution to an optimization problem (or conclude that there is no optimal solution). It is therefore a complete method and not a heuristic.

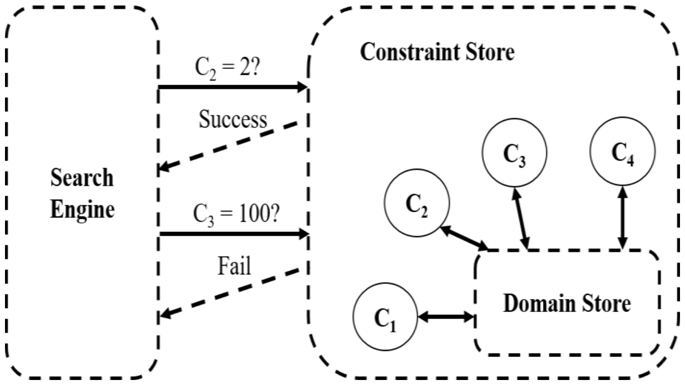

Figure4. *Graphical representation of constraint programming*

The computational paradigm of CP is based on the concept of branching and the concept of pruning. In this case, pruning involves the use of constraints to remove values that cannot belong to any solution.

This is done through the process of feasibility checking and results in the reduction of the search space (Hahn-Goldberg, Beck, Carter, Trudeau, Sousa, & Beattie, 2014).

Unlike branch & bound which focuses on bounding, CP focuses on feasibility checking. This enables its key benefit to be realized, that is, its ability to capture complex, idiosyncratic constraints.

Mixed Integer Programming

Mixed Integer Programming (MIP) borrows several concepts from linear programming. However, unlike linear programming, MIP allows for some of the constraints to be integers. In order to create a MIP model, decision variables, constraints, and an objective function are all required.

Binary values are preferred when assigning values to these variables. Similar to other greedy algorithm improvements, MIP requires good linear relaxation in order to conduct effective pruning. However, a study by Hahn-Goldberg, et al. (2014) on chemotherapy outpatient scheduling provided evidence that showed that CP outperforms MIP.

Local Search

Local Search (LS) works with complete assignments to decision variables and continuously modifies them as it tends towards finding the optimum solution. The optimum solution in this case is defined by a local minima, that is, a position where every neighbor is worse off than the value under consideration. This is unlike CP which works with partial assignments to constraints and continuously checks to see if these assignments can be modified. In order to accomplish this, LS starts with suboptimal (infeasible) solutions and moves towards more optimal (feasible) solutions by performing local moves. A common approach to solving LS optimization problems is based on the max/min conflict concept as described below.

Step	Description
I	Choose the decision variable that appears the most in violations
II	Change the value in order to decrease the number of violations
III	Keep changing until the number of violations is the least (until you reach a local minima)
IV	Use the hypothesis to make predictions (deduction)

Probabilistic Reasoning for Decision-Making

Sullivan (2003) proposed a systematic approach to software tuning that can be applied to an arbitrary software system. This methodology was based on the use of probabilistic and decision-making techniques that have been developed by researchers in Artificial Intelligence (AI), operations research, and other related fields. One of the distinct characteristics of the methodology is the interaction with domain experts during the initial stages to determine how the variables under consideration are inter-dependent or related to their parent and to their ancestors (conditional independencies).The methodology applies the acquisition of knowledge from domain experts as well as from intuitive notions of causality regarding how changing one variable affects other variables in the environment or decision situation. The methodology also applies probabilistic reasoning modeled by influence diagrams and thus outperforms the use of regression models which do not capture elements of the decision maker's objective function (what to maximize or minimize).

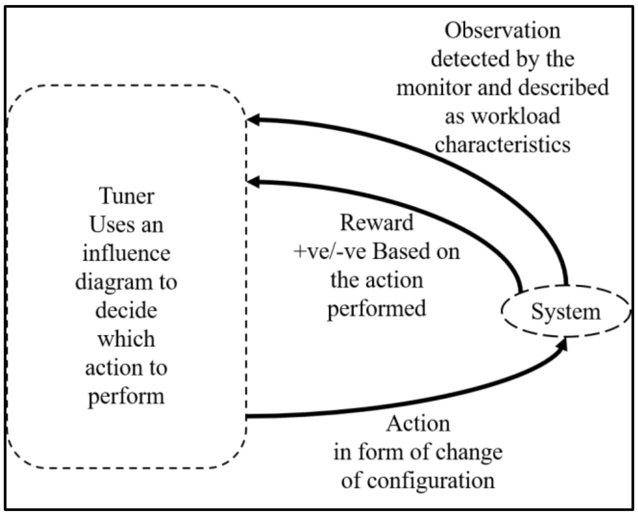
Observation detected by the monitor and described as workload characteristics
Tuner Uses an influence diagram to decide which action to perform
Reward +ve/-ve Based on the action performed
System
Action in form of change of configuration

Figure5. *System-monitor-tuner relationship*

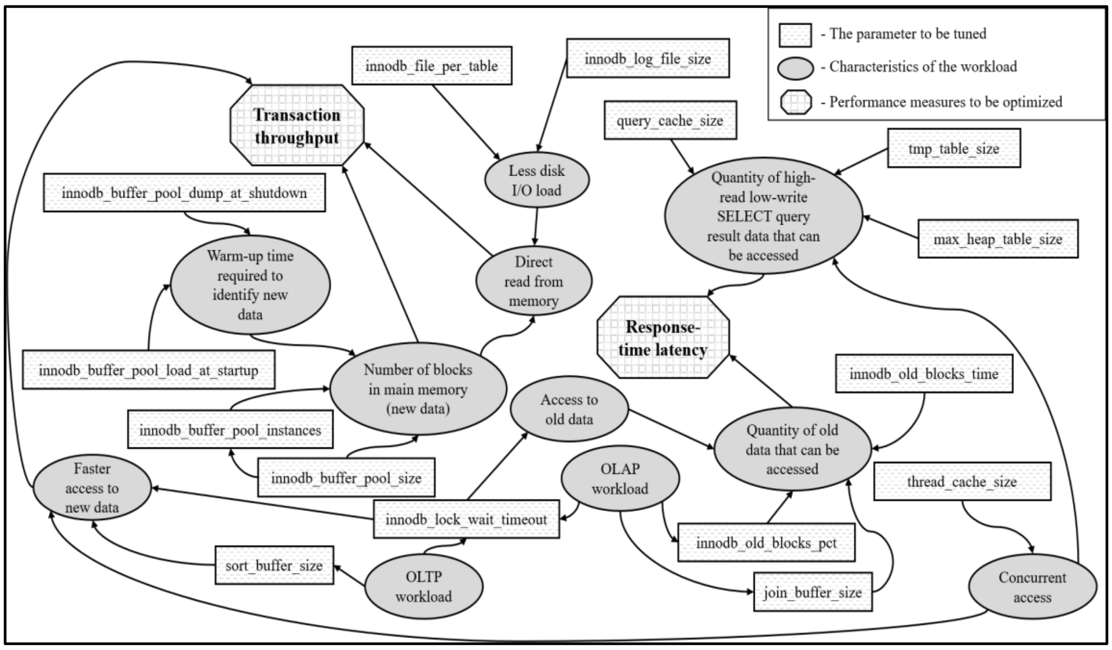
- The parameter to be tuned
- Characteristics of the workload
- Performance measures to be optimized
Transaction throughput
innodb_file_per_table
innodb_log_file_size
query_cache_size
tmp_table_size
max_heap_table_size
Less disk I/O load
Direct read from memory
Quantity of high-read low-write SELECT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

Figure 6. *Influence diagram for a Maria DB synchronous multi-master distributed database*

Influence diagrams can be used as a compact, graphical and mathematical representation of the decision situation. Influence diagrams are becoming a preferred alternative to traditional decision trees (Chajewska, 2000). This is because decision trees suffer from exponential growth in the number of branches with each variable modeled (Hansen, Shi, & Khaled, 2016). In the case of this research, a monitor periodically or in real-time checks the system for any significant changes. If a significant change is detected, the monitor feeds the detected changes to the tuner.

The tuner can then use an influence diagram together with a description of the current state (workload characteristics) to determine the necessary adjustments to each of the configuration settings. Fig. shows the relationship between the system, monitor, and the tuner. Figure . models the complex, nonlinear relationship between the key parameters to be tuned in a storage server.

CONCLUSION

A study by Moreno, Papadopoulos, Angelopoulo, Cámara, & Schmerl (2017) pointed out that reactive autonomic computing systems are appropriate in situations where the time it takes for an adaptation to become effective in the system, that is, the adaptation latency, is low. However, the same is not true for systems which have a high adaptation latency.

It is possible for such systems that have a high adaptation latency to be in a situation whereby the effects of adjusting configuration parameters in response are felt after the conditions that warranted the change in the first place are no longer present.

This article proposes the need to move towards proactive and automatic performance tuning of servers as an opportunity to extend the research further.

Even though one can attempt to argue that automation may lead to the loss of jobs, if used correctly, it saves on the labor requirements per unit of output produced. This saved labor can then be applied in other fields towards further advancement.

REFERENCES

[1] Afonso, C., & Moreira, J. (2017). Data center: Energetic and economic analysis of a more efficient refrigeration system with free cooling and the avoided CO2 emissions. *International Journal of Engineering Research and Application, 7*(11), 1-7.

[2] Anjum, A., & Patil, R. (2017). Load balancing for cloud ecosystem using energy aware application scaling methodologies. *International Research Journal of Engineering and Technology, 4*(5), 479-482.

[3] Autor, D. (2015). Why are there still so many jobs? The history and future of workplace automation. *The Journal of Economic Perspectives, 29*(3), 3-30.

[4] Chajewska, U. K. (2000). Making rational decisions using adaptive utility elicitation. In AAAI/IAAI (pp. 363-369). *American Association for Artificial Intelligence*, (pp. 363-369).

[5] Dostál, J. (2014). Theory of problem solving. *Procedia-Social and Behavioral Sciences, 174*(1), 2798-2805.

[6] Feng, Q. Y., Vasile, R., Segond, M., Gozolchiani, A., Wang, Y., Abel, M., & Dijkstra, H. A. (2016). ClimateLearn: A machine-learning approach for climate prediction using network measures. *Geoscientific Model Development.*

[7] Gomes, D. M., Endo, P. T., Gonçalves, G., Rosendo, D., Santos, G. L., Kelner, J., . . . Mahloo, M. (2017). Evaluating the cooling subsystem availability on a cloud data center. *2017 IEEE Symposium on Computers and Communications (ISCC)* (pp. 736-741). IEEE

[8] Hahn-Goldberg, S., Beck, J. C., Carter, M. W., Trudeau, M., Sousa, P., & Beattie, K. (2014). Solving the chemotherapy outpatient scheduling problem with constraint programming. *Journal of Applied Operational Research, 6*(3), 135-144.

[9] Hansen, E. A., Shi, J., & Khaled, A. (2016). A POMDP Approach to Influence Diagram Evaluation. *International Joint Conference on Artificial Intelligence*, (pp. 3124-3132).

[10] Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer, 36*(1), 41-50.

[11] Ma, X. G., & Liu, X. (2016). A new branch and bound algorithm for integer quadratic programming problems. *Journal of Nonlinear Science and Applications*, 1153-1164.

[12] Moreno, G. A., Papadopoulos, A. V., Angelopoulo, K., Cámara, J., & Schmerl, B. (2017). Comparing Model-Based Predictive Approaches to Self-Adaptation: CobRA and PLA. *12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems.*

[13] Nataraj, A., Malony, A. D., Morris, A., Arnold, D. C., & Miller, B. P. (2010). A framework for scalable, parallel performance monitoring. *Concurrency and Computation: Practice and Experience, 22*(6), 720-735.

[14] Qian, C., Yu, Y., & Tang, K. (2018). Approximation Guarantees of Stochastic Greedy Algorithms for Subset Selection. *Twenty-Seventh International Joint Conference on Artificial Intelligence*, (pp. 1478-1484).

[15] Rallo, A. (2014, March). *Industry Outlook: Data center energy efficiency*. Retrieved September 24, 2018, from Data Center Journal: http://www.datacenterjournal.com/industry-outlook-data-center-energy-efficiency/

[16] Rao, G., Murthy, N. V., & Devi, G. L. (2018). 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, 3*(2), 232-236.

[17] Rodrigues, G. D., dos Santos, G. L., Guimaraes, V. T., Granville, L. Z., & Tarouco, L. M. (2014). An architecture to evaluate scalability, adaptability and accuracy in cloud monitoring systems. *International Conference on Information Networking (ICOIN)* (pp. 46-51). IEEE.

[18] Shahapure, N. H., & Jayarekha, P. (2014). Load balancing in cloud computing: A survey. *International Journal of Advances in Engineering & Technology, 6*(6), 2657-2664.

[19] Shinde, M. P., & Channe, H. (2018). Semi-supervised Learning with Ensemble Method for Online Deceptive Review Detection. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 3*(6), 415-422.

[20] Sullivan, D. G. (2003). *Using probabilistic reasoning to automate software tuning.* Doctoral Dissertation, Harvard University, Computer Science Group, Cambridge, Massachusetts. Retrieved October 01, 2018, from https://www.researchgate.net/profile/David_Sullivan16/publication/221595676_Using_probabilistic_reasoning_to_automate_software_tuning/links/55cb36dc08aeb975674ad1d3.pdf

[21] Van Aken, D., Pavlo, A., Gordon, G. J., & Zhang, B. (2017). Automatic database management system tuning through large-scale machine learning. *ACM International Conference on Management of Data* (pp. 1009-1024). Association for Computing Machinery.

[22] Zhang, X., Abbasi, H., Huck, K., & Malony, A. D. (2016). WOWMON: A machine learning-based profiler for self-adaptive instrumentation of scientific workflows. *Procedia Computer Science, 80*, 1507-1518.

APPENDIX1.PARAMETERS TO BE CONSIDERED

Table1. *List of most critical parameters to be considered during performance tuning*

	Parameter	Description
1.	innodb_buffer_pool_size	Specifies the amount of main memory that can be used to store frequently used blocks of data and indexes. The larger the value, the more the quantity of data and indexes that can be stored in memory. This subsequently reduces the bottleneck caused by disk IO. An ideal value is 70-80% of the total available memory on a dedicated database server with primarily XtraDB or InnoDB tables. However, if the value of this parameter is too large, then memory swapping can occur which makes the performance of the server even worse. The tradeoff is that the larger the value of this parameter, the longer the server will take to initialize. **Affected resource:** main memory
2.	innodb_buffer_pool_instances	This parameter divides the InnoDB buffer pool into a specific number of instances such that each instance manages its own data structures and takes an equal portion of the total buffer pool size. This helps to reduce contention concurrency. An ideal value is greater than or equal to 1GB for each instance. For example, if the innodb_buffer_pool_size is 8GB, then there will be 8 instances each with a 1GB buffer pool when the innodb_buffer_pool_instances is set as 8. **Affected resource:** main memory
3.	innodb_old_blocks_pct	The InnoDB buffer pool has two sub-lists. One sub-list for recently used information, and another sub-list for older information. By default, 37% of the list is reserved for the old list but this value can be changed by adjusting the value of the innodb_old_blocks_pct parameter. This value can be changed to anything between 5% and 95%. A smaller old sub-list enables faster eviction of less frequently used data from the buffer pool, thus giving room for more frequently used data to be stored in the new sub-list. **Affected resource:** main memory
4.	innodb_old_blocks_time	This parameter specifies the delay in milliseconds before a block can be moved from the old sub-list to the new sub-list in an Inno DB buffer pool. The default value (in Maria DB 5.5) is 0 which implies no delay, but this value can be set to a non-zero value as well. A non-zero delay helps in situations where full table scans are performed in quick succession. For example, when performing logical backups, full table

		scans in quick succession are expected. In such cases, it is better to ensure that data which is accessed only once remains in the old sub-list so that it can be evicted from the buffer pool instead of being moved to the new sub-list. **Affected resource:** main memory
5.	innodb_buffer_pool_dump_at_shutdown	This parameter enables the buffer pool state to be dumped into disk before the server is shutdown. It can be set to either ON or OFF. By default, it is OFF. **Affected resource:** main memory
6.	innodb_buffer_pool_load_at_startup	This parameter works with the previous parameter, i.e. innodb_buffer_pool_dump_at_shutdown to restore the buffer pool to the state it was in before the server was shutdown. It can be set to either ON or OFF and by default it is OFF. Setting innodb_buffer_pool_dump_at_shutdown and innodb_buffer_pool_load_at_startup to both ON eliminates the warmup time required for the buffer pool to identify and store the most frequently accessed data because it can pick up from where it left off before the server was shutdown. **Affected resource:** main memory
7.	query_cache_size	Specifies the size in Bytes that is available forstoring the results of SELECT queries. Storing these results is useful for OLAP workloads that have a high-read and low-write environment. However, the query cache cannot be enabled in MariaDB Galera cluster versions prior to “5.5.40-galera”, “10.0.14-galera”, and “10.1.2”. An ideal value is to set query_cache_size=0 orquery_cache_type=OFFand use other techniques to increase the performance of OLAP workloads, e.g. good indexing, and setting up a load balancer to spread the read load. This is because the query cache is a well-known bottleneck. **Affected resource:** cache memory **Affected workload:** OLAP
8.	innodb_log_file_size	Redo logs are used to make sure database writes are fast anddurable. They are also used during a recovery from a server crash however, larger log files can cause slower recovery in the event of a server crash. In as much as they can make recovery from a server slow, larger log files mean less disk I/O due to less flushing checkpoint activity.The size can be 1MB to 512GB (>= MariaDB 10.0) or 1MB to 4GB (<= MariaDB 5.5) **Affected resource:** storage
9.	innodb_file_per_table	This parameter allows some of the database tables to be kept in separate storage devices. This can greatly improve the I/O load on the storage. Default value is innodb_file_per_table=ON (>=MariaDB 5.5) and innodb_file_per_table=OFF (<=MariaDB 5.3) **Affected resource:** storage
10.	innodb_lock_wait_timeout	This parameter sets the time in seconds that an InnoDB transaction waits for an InnoDB row lock before giving up with a “timeout exceeded” error. When the timeout is exceeded, the statement (not the transaction) is rolled back. OLAP workloads benefit from a high innodb_lock_wait_timeout. OLTP workloads on the other hand benefit from a low innodb_lock_wait_timeout. The default value is innodb_lock_wait_timeout=50 and the range is 0 to 1073741824 (>= MariaDB 10.3) and 1 to 1073741824 (<= MariaDB 10.2) **Affected resource:** CPU **Affected workload:** OLAP and OLTP
11.	thread_cache_size	This parameter sets the number of threads that the server should cache for re-use. Increasing this parameter helps servers with high volumes of connections per second so that most connections can use a cached thread as opposed to a new thread. It can range from 0 to 16384. The default value is thread_cache_size=0 (<=MariaDB 10.1) and thread_cache_size=auto (from MariaDB 10.2.0) **Affected resource:** CPU **Affected workload:** OLTP
12.&13	tmp-table-size and max-	This parameter sets the default size of a temporary table. The

	heap-table-size	temporary tables are used when processing complex queries that involve joins and sorting. This parameter therefore helps to prevent disk writes. It should have the same size as max-heap-table-size. An ideal value is assigning 64MB for every GB of RAM on the server. **Affected resource:** main memory **Affected workload:** OLAP
14.	sort-buffer-size	This parameter specifies the amount of memory in a buffer that is to be allocated to each session performing a sort operation. This value should be minimized for OLTP workloads that are known to have many small sorts. The default value is 2M, but an ideal minimum value is 16K. **Affected resource:** buffer memory **Affected workload:** OLTP
15.	join_buffer_size	This parameter is used to set the size of the buffer used for queries that cannot use indexes and thus perform a full table scan. An ideal value is to minimize it globally and to set a high value for session that require large full joins. **Affected resource:** buffer memory **Affected workload:** OLAP