Foundations of an Autonomic Manager for Maintaining Quality of Service in Enterprise Data Warehouses

Allan O. Omondi*, Ismail L. Ateya**
Faculty of Information Technology
Strathmore University
Nairobi, Kenya
allan[at]odhiambo.me.ke* iateya[at]strathmore.edu**

Gregory N. Wanyembi
School of Information Sciences & Knowledge Management
University of Kabianga
Kericho, Kenya
gwanyembi[at]gmail.com

***Abstract*— Data stored in an Enterprise Data Warehouse (EDW) is an essential asset to enterprises. Through efficient access to data (where efficiency is quantitatively measured in terms of speed), SMEs can enhance their growth, productivity, and global competitiveness. This can in turn lead to a positive impact on a country's Gross Domestic Product. The purpose of this paper is to present the building blocks required to maximize the speed of data access from EDWs in a self-adaptive manner. Reinforcement Learning (RL) in a fully observable, stochastic environment is proposed. The subsequent solution to a Markov Decision Process is highlighted as the core part of the RL.**

Keywords— ***Autonomic computing; enterprise data warehouse; Markov Decision Process; Markov Reward Process; Reinforcement learning***

I. INTRODUCTION

As computing systems get more optimized, the complexity involved in managing them increases rapidly and this can result in a barrier to further growth. Autonomic computing enables such systems to adapt to unpredictable changes while hiding intrinsic complexity. A study by [1] indicated that today's high availability requirements put greater demands on computing systems to be self-adaptive in order to maintain a desirable Quality of Service (QoS) in the presence of system faults, variable environmental conditions, and dynamic user expectations. The study further noted that even though system administrators are better at understanding the overall problem context than computers, they are prone to long reaction times, fatigue, errors, and varying and potentially inconsistent expertise.

One of the key visions of the Government of Kenya is to transition the country into a knowledge economy by the year 2030. This means an economy in which growth is dependent on the quantity, quality, and accessibility of information available to be used for innovation rather than dependency on traditional means of production such as land. The Kenyan national Information Communications and Technology (ICT) masterplan acknowledges that enhancing the growth, productivity, and global competitiveness of Small and Medium Enterprises (SMEs) has the potential of increasing the Gross Domestic Product (GDP) of a country [4]. Access to data by SMEs is a crucial catalyst in creating a knowledge economy.

The few personnel that SMEs can afford usually play multiple roles in the enterprise. This leads to a proclivity to rely on system administrators to perform all Information Technology (IT) duties in the enterprise. The over-reliance on system administrators results in a lower QoS as they strive to cope with the demand for their expertise. One of the areas in IT that is directly affected by the low QoS from the system administrators is administration of the Enterprise Data Warehouse (EDW). The effect of this poor administration is experienced through low data access speeds that could have otherwise been avoided through proper system administrative tasks.

A. Our Contribution

Our main contribution is a mathematical model of an EDW that represents the decisions that the EDW can make autonomously at runtime in order to maintain a desirable, pre-defined QoS. The aim of making this contribution is to develop the foundation that can be used to build a self-healing and self-adaptive system that is not fully dependent on the intervention of system administrators.

Exception handling code embedded within a system can be used to maintain the desirable QoS. This would work by coding the system to throw exceptions if the QoS falls below a certain threshold and then handling the thrown exceptions using exception handlers. However, it is important to note that the occurrence of runtime phenomena is stochastic in nature and asynchronous with respect to the flow of the application logic. A study by [2] indicated that for this reason, it is preferable to gather the complex adaptation logic into a component separated from the application logic. Another approach in contrast to embedded exception-handling code and championed by the IBM autonomic computing team, is to

implement the autonomic computing and self-healing system distinct from and external to the managed system. This is as modelled by the IBM Monitor-Analyse-Plan-Execute (MAPE) loop [2].

Our contribution combines the best features of the IBM MAPE loop with the best features of the DMAIC data-driven strategy defined in [3] to derive the paradigm depicted in Fig. 1.

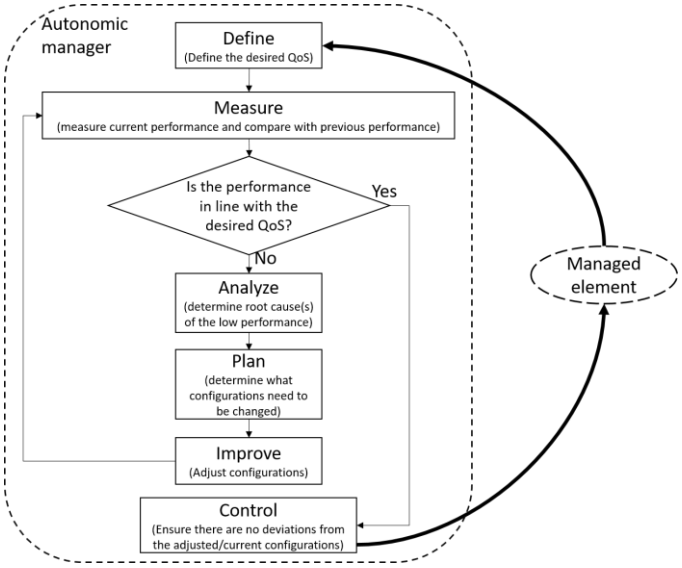

Fig. 1: Adapted combination of the DMAIC data-driven strategy and the IBM MAPE loop

The remainder of the paper is organized as follows. Section II presents our approach. This approach is further divided into a number of sub-sections. The first sub-section is presents a derivation of the optimization model. This is followed by a connection to Reinforcement Learning (RL) as a way to solve the optimization problem. The last two sub-sections further describe the Markov Reward Process and Markov Decision Process used by the RL. Section III concludes the paper.

II. Our Approach

A. The Optimization Model

Let $I = \{A_1, A_2, \ldots, A_n\}$ be the set of all possible attributes in a relation consisting of n attributes. Since not all of the attributes are optimal in terms of their ability to promote faster access to data, then some can be considered in a partition and others can be left out. A partition can then be defined as, a subset from all possible attributes $P_k \in I$ such that $P_k = \{A_{k1}, A_{k2}, \ldots, A_{kn}\}$, whereby each attribute in the subset P can be referenced by a query during execution.

The query under discussion can in turn be grouped together with other queries such that they collectively form a workload of m queries; $W = \{Q_1, Q_2, \ldots, Q_m\}$. In this case, each query Q_q, has a different execution cost. This execution cost is directly dependent on whether the query references an attribute that is in the partition defined by P_k. The dependency is such that if a number of attributes are commonly referenced by workloads, then groping them together in one partition on the storage medium increases the speed of data access.

Given that attributes can be grouped together into a partition P_k, we can have k sets of P each with unique combinations of attributes as members of the set. These k partitions can be grouped into a configuration C expressed as $C = \{P_1, P_2, \ldots, P_k\}$. C thus becomes the set of all possible partitions.

The objective is therefore to find that one partition that will maximize the speed of data access from the EDW. This is defined as the most optimum partition. We adopt the quantitative definition of an optimum partition as one which accrues the most benefit. Benefit can in turn be defined quantitatively as the difference in speed between executing a query without using the chosen partition and executing a query using the chosen partition. Mathematically represented as shown in (1).

$$b_{qk} = cost(q, \emptyset) - cost(q, P_k) \quad (1)$$

Where;

$cost(q, \emptyset)$ = the cost of running query q without using any partition

$cost(q, P_k)$ = the cost of running query q while using the chosen partition

Even though one of the main advantages of partitioning is that it can improve the performance of queries, it can also have certain disadvantages. The disadvantage in this case is based on a negative benefit accrued from the need for the partitions to be maintained and the extra storage space required. Fortunately, the main queries executed on an EDW involve selection of data. This is as opposed to insertion and updating of data which is common in databases that support heavily used Online Transaction Processing (OLTP) systems. Therefore, as [5] argue, the number of changes in an EDW are not as many as the changes in a database that supports OLTP systems. As a result, the updates to the partitions are not as frequent. They also argue that the cost of storage has been falling rapidly and is now more affordable.

We however argue that the impact of using a partition, P_k, cannot be fully realized unless the cost (negative benefit) of having that partition is also considered. This implies that the actual cost should be the benefit of using the chosen partition minus the cost of maintaining the chosen partition. This can be represented as shown in (2).

$$\sum_{q=1}^{m}\sum_{k=1}^{p} b_{qk} - \sum_{k=1}^{p} f_k \quad (2)$$

This represents the benefit of using partition k to support the execution of query q. The same query is run using

different partitions, that is partition $k+1$, then $k+2$, then $k+3$, all the way until the last partition, which is partition p.

Once all partitions in the configuration have been applied in a particular query, q, then the next query ($query\ q+1$) in the workload is selected and the cycle of applying the same various partitions on the new query repeats itself. Query $q+2$ goes through the same, then query $q+3$, then $q+4$, all the way until the last query in the workload, which is query m. Rebuilding the partition in the event of additional data being added to the EDW and new workloads being used, in itself constitutes a query. This query is query q`. The cost f_k is associated with each partition P_k. This is such that k can be 1 (the first partition in the configuration), which is followed by partition $k+1$, then $k+2$, then $k+3$, all the way until the last partition, which is partition p.

However, there can arise a scenario whereby a partition does not need to be rebuilt. This will imply that there is no associated cost for that particular partition P_k. At the same time, there can arise a scenario whereby query $Q_q$does not apply partition P_k. There is therefore no notable benefit in such a case. This can thus be modelled as the optimization model shown in (3).

$$\max\left(\sum_{q=1}^{m}\sum_{k=1}^{p} b_{qk}\,.x_{qk} - \sum_{k=1}^{p} f_k\,.y_k\right) \tag{3}$$

Such that:

$$x_{qk} = \begin{cases} 1, \text{ query } q \text{ uses partition } k \\ 0, \text{ otherwise} \end{cases}$$

$$y_k = \begin{cases} 1, \text{ partition } k \text{ needs to be rebuilt} \\ 0, \text{ otherwise} \end{cases}$$

B. Autonomic Computing through Reinforcement Learning

Through the use of control theory, a controller, *C*, is used to control a system, *P*, in such a way that its actual output, *y(t)*, follows a desired control signal in the form of a reference, *r(t)*. The controller can then be programmed to obtain the error signal, *e*, defined by the difference between the reference and the actual output; *r(t)-y(t)*. In order to tend towards obtaining the reference, the error signal, *e*, is translated into feedback by the controller in the form of input, *u*. It is this error signal that enables the controller to know whether it is on the right track or if it is off target. If it is off target, then it can use *u* to configure the system, *P*, accordingly so as to get back on track. Fig. 2 portrays this graphically.

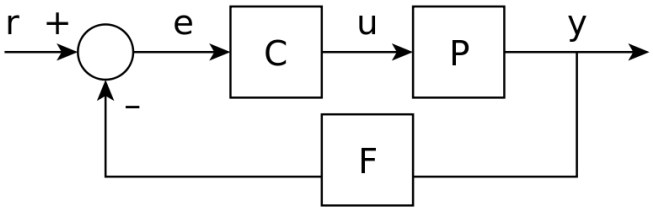

Fig. 2: Single-input-single-output (SISO) control system

Applying the same concept in the context of this research results in a block diagram as shown in Fig. 3. The following section details how we apply reinforcement learning to model the control theory.

Reinforcement Learning sits in the middle of the intersection between many fields of science as the study of the most optimal way to make the best decisions. These fields include machine learning in computer science, operations research in mathematics, optimal control in engineering, bounded rationality in economics, classical and operant conditioning in psychology, and reward system in neuroscience.

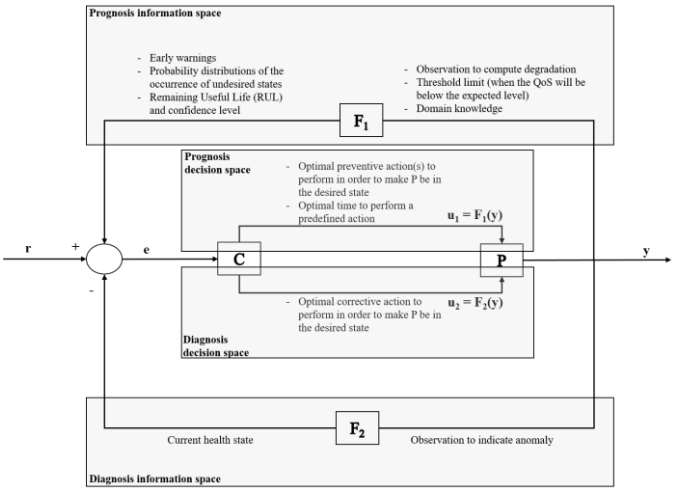

Fig. 3: Proactive decision making framework for maintenance (Adapted from [6], p. 1241)

This paper focuses on the computer science field whereby RL, supervised learning, and unsupervised learning form the three paradigms of machine learning.

Similar to a control system, RL uses feedback to define how well an agent is performing towards achieving its goal of maximizing rewards as it traverses through a process in time. Fig. 1 can therefore be modified as shown in Fig. 4 thus resulting in a continuous loop that strives for constant improvement. We propose that any autonomous, self-healing system can be built by implementing a continuous loop in this manner.

C. The Markov Reward Process

Given that an observation, O_t, can be made from the EDW at time *t*, an action performed based on the observation made, and a reward received based on the action performed, then we can have a history, H_t, such that $H_t = O_1, A_1, R_1, \ldots, O_t, A_t, R_t$.

A dynamical system has a direct relationship between the amount of computation performed and the quality of the output given. This is such that the more computations that are performed, the fewer the number of resources available to compute and produce the output at the required pace. It would therefore be contradictory to have an autonomic manager that seeks to increase the speed of data access but is in itself computationally demanding to the extent that it leads to a reduction in the speed of data access.

One of the ways a non-compute intensive autonomic manager can be realized is by not storing and processing the history since time *t=1*. A summary of the history can be obtained in the form $S_t = f(H_t)$ such that S_t is the state at time *t*. This implies that all previous states can be discarded and only the representation of the current state considered when the agent is deciding what action to perform next. We can therefore deduce that a Markov state defines the future as independent of the past given the present: $H_{1:t} \rightarrow S_t \rightarrow H_{t+1:\infty}$

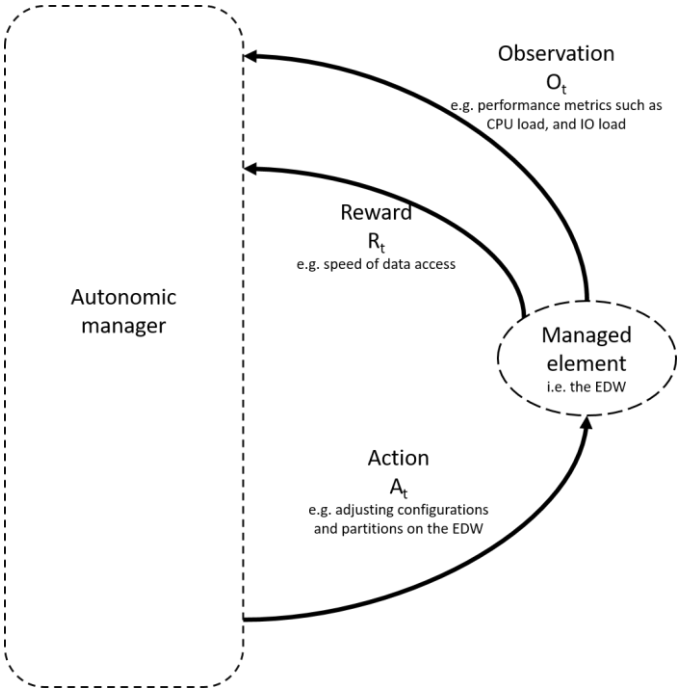

Fig. 4: RL concepts applied to the autonomic manager

As the autonomic manager performs actions to traverse through each state, it receives the reward defined by (3) without applying the maximization. An important function that it should be able to perform is to look forward into the future in order to determine the expected reward if it performs a certain group of actions. This can be modelled as a value function in the form:

$$\begin{aligned} v(s) &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s] \end{aligned} \quad (4)$$

This implies that the value of a state, *s*, is the immediate reward that is received from being in that state (R_{t+1}) plus the value of all other states in future ($v(S_{t+1})$). γ is considered as the discount factor in order to ensure that the reward at time *t* is much higher than the reward at time *t+1*, thus giving a higher priority to immediate rewards than to future rewards. One reason for giving less priority to future rewards is because there is uncertainty in the future. It also makes it mathematically valid by avoiding a summation to infinity.

Given that at each state, the autonomic manager can have multiple options of subsequent states that it can traverse to, then we can assign probabilities to each subsequent state in the form depicted in Fig. 5. We can therefore adjust the autonomic manager's value function to be the immediate value derived from being in a state, say *s*, plus the discounted value of the subsequent state, say *s``*, multiplied by the probability of going to that subsequent state, that is $P_{ss``}$. This gives us the equation shown in (5).

$$v = R + \gamma P v \quad (5)$$

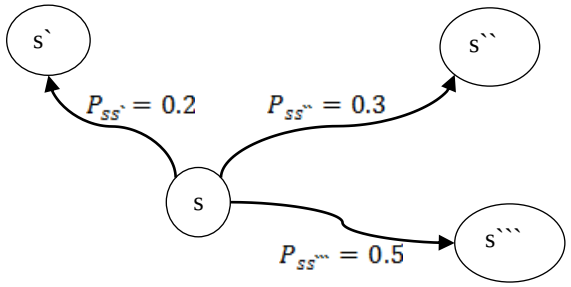

Fig. 5: Probabilities of transition from state, *s*, to subsequent states, *s`*, *s``*, or *s```*

Inductively applying this in a real context can be done through the use of matrices. The real context in this case would involve hundreds of possible states that the EDW can be in and that the autonomic manager can traverse to. This gives us the Markov Reward Process as:

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} R(1) \\ \vdots \\ R(n) \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & & \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} \quad (6)$$

D. Markov Decision Process

The previous cases defined the reward that the autonomic manager derives from being in various states. At this juncture, we assign agency to the autonomic manager. By assigning

agency, the Markov Reward Process becomes a Markov Decision Process. It is this agency that allows the autonomic manager to make decisions on which actions to perform in order to move to a specific state that has an expected value. The possible actions that our autonomic manager can perform revolve around deciding which columns should be in the table's partition. Once it performs this action, it makes an observation of the environment and subsequently receives the reward of performing the specific action. If the reward is positive then it knows that it is on the right track. Note that the reward in this case is as defined in (3) without the maximization. This is translated to mean the speed of data access from the EDW owned by the SME.

The aim is therefore to find the action that will enable the autonomic manager to get the most reward at any given state in the process. It can be modelled as shown in (7).

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma v(S_{t+1}) | S_t = s, A_t = a] \quad (7)$$

Such that $q_\pi(s,a)$ is the action-value function that defines the value that the agent will get if it performs action *a* (defined by a policy π) given that it was in state *s* while performing the action.

The final solution is therefore to find the policy that has a set of actions, which if performed in specific states, will yield the maximum benefit possible. This is the solution to the Markov Decision Process and is subsequently modelled as shown in (8).

$$q_*(s,a) = \max_\pi(q_\pi(s,a)) \quad (8)$$

Where $q_*(s,a)$ is the most optimum action-value pair, that is, the solution to the optimization problem.

III. Conclusion

As opposed to presenting a fully developed autonomous agent, this paper provides the foundations that act as building blocks for one interested in developing an actual autonomous agent. The context is that of an autonomous agent that continuously strives to manage an acceptable QoS for the speed of data access from an EDW.

For future work, we are currently capitalizing on these foundations in order to develop the actual autonomous agent. We do not aim to replace system administrators, but rather compliment them; especially those who are overburdened with multiple duties in Kenyan SMEs.

References

[1] Cheng, S. W., & Garlan, D. (2012). Stitch: A language for architecture-based self-adaptation. *Journal of Systems and Software, 85*(12), 2860-2875.

[2] Su, G., Chen, T., Feng, Y., Rosenblum, D. S., & Thiagarajan, P. S. (2016). An iterative decision-making scheme for markov decision processes and its application to self-adaptive systems. In *International Conference on Fundamental Approaches to Software Engineering* (pp. 269-286). Springer Berlin Heidelberg.

[3] Solanki, M. & Desai, D. (2015). Comparative study of TQM and six sigma. *International Journal of Industrial Engineering & Technology, 5*(4).

[4] Government of Kenya. Ministry of Information Communications and Technology. (2014). *The Kenya national ICT masterplan: Towards a digital Kenya.* Nairobi: Ministry of Information Communications and Technology

[5] Kim, J. W., Cho, S. H., & Kim, I. M. (2016). Workload-Based column partitioning to efficiently process data warehouse query. *International Journal of Applied Engineering Research, 11*(2), 917-921.

[6] Bousdekis, A., Magoutas, B., Apostolou, D., & Mentzas, G. (2015). A proactive decision making framework for condition-based maintenance. *Industrial Management & Data Systems, 115*(7), 1225–1250. doi:10.1108/imds-03-2015-0071