International Innovative Research Journal of Engineering and Technology
Vol. 7, Issue. 4, June 2022, pp. 10~18
ISSN: 2456-1983, DOI: 10.32595/iirjet.org/v7i4.2022.157

Challenging the Robustness of Self-Managing Computing Systems for QoS Controller Design

Sangwoo Jeon[1], Sakthivel Velusamy[2]
[1]Department of Computer Science and Engineering, Konkuk University, Seoul, South Korea
[2]School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, 600127, Tamilnadu, India

Article Info

Article history:

Received Mar 15, 2022
Revised May 20, 2022
Accepted Jun 11, 2022

Keywords:

Robustness
self-managing computer systems
QoS controller design
computing installations

ABSTRACT

A widespread interest in lowering the requirement for human involvement by enabling systems to operate autonomously has been sparked by the high expense of running massive computing installations. The extent to which control theory can offer an analytical and architectural basis for developing self-managing systems is examined in the present research. For the QoS management of such systems, these methods often employ fixed, adaptive, or single model based control techniques. However, it is challenging to create a single model or controller that would provide the required QoS performance over all of these systems' operational zones due to the variable system dynamics and unforeseen environmental changes. The goal of a novel approach known as self-managing computer systems is to incorporate into the systems the means by which they can automatically modify configuration parameters to ensure that the system's Quality of Service requirements are continuously satisfied. In this research, we assess the resilience of such approaches under high variable workloads in terms of request service times and inter-arrival times. This research also makes a contribution by evaluating how workload forecasting techniques are used in QoS controller design.

Corresponding Author:

Sangwoo Jeon,
Department of Computer Science and Engineering,
Konkuk University, Seoul, South Korea.
Email: jswp5580@konkuk.ac.kr

1. INTRODUCTION

The complexity of computer systems is rising. Large and heterogeneous numbers of hardware and software components, multi-layered architecture in system design, and workloads that are unpredictable—particularly in Web-based systems—are some of the factors that contribute to complexity in systems. These explanations explain why performance management of complex systems is costly and challenging for humans to do. A novel strategy known as self-managing computer systems involves incorporating the necessary mechanisms into the systems to self-adjust configuration parameters in order to guarantee that the system's Quality of Service (QoS) needs are continuously satisfied [1]. The papers presented at a recent workshop demonstrate the growing interest in self-managing systems. In this investigation, we offer a method to build controllers that run frequently (e.g., every few minutes) to find the optimal configuration for the system given its workload. This is achieved by combining combinatorial search techniques with analytical performance models. We initially use a simulated multithreaded server to demonstrate and motivate the concepts. Then, we present experimental findings from an actual Web server running a workload produced by the Scalable URL Reference Generator, utilising the methods presented here.

Owing to the high cost of ownership of computer systems, the sector has launched several initiatives to lighten the workload for management and operations. Microsoft's Dynamic Systems Initiative, HP's Adaptive Infrastructure, and IBM's Autonomic Computing are a few examples. All of these efforts aim to lower operating costs through more automation; as operator error has been found to be a primary cause of system failures [2], the ideal scenario would be for systems to be self-managing without the need for human interaction. Although the idea of automated operations has been around for 20 years as a means of responding to shifting demands, malfunctions, and—more recently—attacks, the automation's use is still

somewhat restricted. We think that this is partly because there is a lack of basic knowledge about how automated actions impact system behaviour, particularly system stability. Control theory is used in several fields of engineering, including mechanical, electrical, and aeronautical engineering, to build feedback systems.

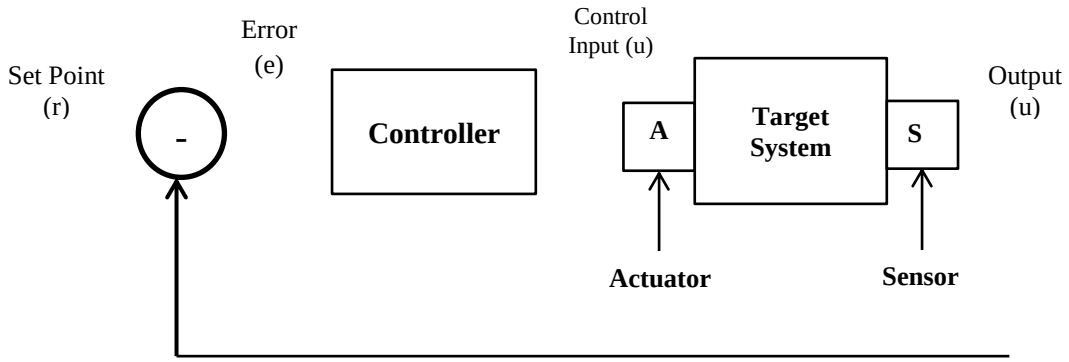

Figure 1. A Control System's Block Diagram

A feedback control system's block diagram is displayed in Figure 1 [3]. The target system is the one that the controller is in control of. A set of performance indicators for features of interest (such response time), also known as outputs, are provided by the target software system. While actuators allow the control input—such as resource allocation—to be changed to alter the system's behaviour, sensors track the target system's outputs. The control system's decision-making component is the controller. Because there is no formal or systematic design process, it is becoming more and more difficult to develop fixed and ad-hoc threshold-based policies that depend on peak demands. Traditional approaches now in use, such manual tuning, have proven to be expensive and error-prone. Implementing strategies that overcome these constraints by giving software systems the ability to make decisions at runtime and manage themselves with little to no human input is crucial.

Following these objectives, the rest of the paper is organised, with Sections 2, 3, and 4 addressing each in turn. Our findings are detailed in Section 5.

2. RELATED WORKS

An application must dynamically detect and react, swiftly and accurately, to changes in the Grid DCE as well as in its own state and requirements in order to address these features and achieve required QoS limits [4]. This usually entails the dynamic adaptation of a number of functional and performance-related elements to the needs of the running application and environment. Maintaining defined QoS limits using current methodologies based on ad hoc human tuning and heuristics is not only time-consuming and prone to error, but also impractical as systems and applications grow in size and complexity. In order to handle increasing complexity, systems and apps must eventually become primarily autonomous, or able to manage themselves with high-level advice from specialists.

The development of self-managing systems in response to various issues arising from highly decentralised and heterogeneous architectures has been facilitated by autonomous computing (AC) [5]. The basis of autonomic computing is biology. In actuality, the body's autonomic nerve system works nonstop to preserve physiological equilibrium. Comparatively speaking, AC is necessary for information technology (IT) to dynamically respond to changes in the environment. Each domain of machine-to-machine communication may exhibit different autonomic behaviour expressions, which frequently leads to a radically different autonomic manager and yet functions building.

Operating conditions in mobile computer environments are very different from those in wired computing environments [6]. Applications specifically need to be able to withstand the extremely dynamic channel conditions that develop when users move around the space. Furthermore, the display qualities, CPU speed, memory capacity, and battery life of the computer devices utilised by various end users may differ. Real-time applications like video conferencing are particularly susceptible to these changes because of their synchronous and interactive character.

System convergence is seen even in the face of modelling errors, innate system nonlinearities, and changes in system parameters over time because of the robustness of the feedback mechanism, which also produces self-correcting, self-stabilizing behaviour in system performance. Still, it's fundamentally a reactionary strategy [7]. In order for the feedback loop to function, measured deviations from the target performance must be addressed by taking corrective action in an effort to bring the deviation down to zero. Sadly, a feedback controller that monitors the current delay won't know about the approaching overload until it really happens. Given that the server response time is a moving average that fluctuates slowly, this response delay is very significant.

Protocol layering has long been the foundation for communication network engineering. This entails creating discrete network functionality (such media access, routing, and flow management) and assembling the entire system via constrained interfaces between the layers carrying out these particular functions [8]. In reality, the layers are dispersed systems with cooperating units spread throughout the network; they are organised in a vertical hierarchy. Every layer uses the services offered by the levels beneath it, and in return, the layers above it can use the services that each layer offers. Only procedure calls and responses are allowed to occur during inter-layer communication, which only occurs between adjacent layers.

Within an online control framework, we examine the design of computer systems that optimise themselves [9]. Control theory offers a methodical approach to resource management in a broad context. It is possible to derive the control actions necessary to maintain a specific quality of service (QoS) by optimising a given cost function if the computer system is adequately modelled and the operating environment is accurately assessed. Additionally, it offers tried-and-true mathematical methods for analysing the accuracy and performance of systems, and it has lately been successfully used to solve issues like task scheduling and QoS adaption in web servers.

In this paper, we discuss our methodology for designing self-configurable and self-managing computer systems utilising analytic performance models. This strategy is demonstrated in a number of our publications. Here, we offer a comprehensive structure, outline the difficulties, and provide a summary of the outcome. The second section covers our overall strategy for managing computer systems [10]. Protocols and procedures are required for effective QoS objective negotiation, oversight, and enforcement. These systems are heterogeneous; hence platform-neutral language must be used to define QoS objectives and contracts.

3. METHODS AND MATERIALS

3.1 Design and control system

There are typically two primary processes in the design of a control system. First, it is necessary to build a formal link between the control input and the output. This link is known as the behavioural model of the system in control theory. System identification (SID) is a technique that measures input and output data in order to build the model. The second phase, which comprises controller design, simulation, analysis, and testing, then makes use of the system paradigm. In addition to meeting operational objectives, the feedback controller must respond to unforeseen disruptions and unmodeled system dynamics. Furthermore, there exist recognized formal approaches and methodologies for the design, development, and analysis of the control system's operational specifications, such as stability, settling time, and exceeding.

Different control systems, including fixed gain, adaptive, and reconfigurable controllers, can be developed using the main approaches mentioned above. The design, benefits, and drawbacks of various control strategies are covered in the sections that follow, along with examples of their use in QoS management from the literature.

3.1.1 Control with fixed gain

A fixed-gain control scheme's topology resembles that of Figure 1.1 [11]. As previously indicated, SID experiments are carried out offline in order to construct the system model. Usually, the system's behavioural model is described by autoregressive exogenous input (ARX) models. The ARX model has the following standard form:

$$x(l) = \sum_{j=2}^{m} b_j \, x(l-j) + \sum_{j=2}^{n} a_i \, v(l-e-i) \qquad (1)$$

Where l is the current sample instance, $x(l)$ are the model's order, b_j, a_i are its parameters, e is the delay—the amount of time it takes to notice a change in the input in the output.

The input and output data obtained from SID experiments are used to determine the order, parameters, and latency of the system model. The controller is developed with the goal of minimizing the difference between the target setting point and the actual output signal using the model that was created during the model identification step. For the purpose of designing a controller that meets the control objectives, a model that is accurate enough is needed. Even with model uncertainties and imperfections, well-designed feedback control architecture can successfully manage performance. For this reason, the various

versions of the Proportional Integral Derivative (PID) controller are frequently employed because of its ease of use, resilience to modelling mistakes, and capacity to reject disturbances. Gains, which are tuning parameters in these controllers [12], can be changed to get the required performance requirements.

3.1.2 Adaptive management

With adaptive control, some of the drawbacks of fixed gain controllers are circumvented by dynamically predicting model parameters and modifying the controller's gains to meet high-level design goals. Changes in the system model as a result of different situations are recorded and incorporated into the controller design online. The multi-model behaviour of the software systems is thus captured by adaptive control. Adaptive control, however, operates under the fundamental premise that the model parameters either stay constant or change gradually over time.

On the other hand, performance degradation leading to high transient reactions and temporal instabilities is widely documented in the literature. Software systems are susceptible to rapidly changing circumstances, such as unexpected spikes in demand [13], "Slashdot" effects, component failures, and the garbage collection process. Other restrictions on adaptive control include the computing expense resulting from online design and estimate.

As it takes time to develop the estimations, the start-up performance could not be adequate. Additionally, in order for these methods to estimate the model quickly and accurately, the input signal must contain a wide enough frequency range to excite the system (a condition known as persistently stimulating).

3.1.3 Reorganising the control system

Through online adjustments to the controller parameters, the adaptive control method offers greater flexibility than the non-adaptive scheme. Still, the loop's filters and other component arrangements, as well as the controller algorithm, remain constant across time. As operational conditions and disturbances change, various control algorithms or loop arrangements might offer more control. The fundamental concept of the reconfiguring control scheme is to adjust the control loop structure, models, and algorithms to deal with the system's shifting working areas and external circumstances. Furthermore, it is possible to circumvent certain prerequisites for adaptive control (such a continuously stimulating signal and gradually altering conditions) by merging several fixed gain controllers and choosing an appropriate one at runtime.

Moreover, it is possible to mix fixed and adaptive control systems to enhance performance in rapidly changing circumstances. As a result, this method helps to choose suitable control loop configurations at runtime and capture the multi-model aspect of the software system. However, because of the design and assessment of the various models and controllers, there are compromises between the system's runtime overhead and design complexity.

Furthermore, it could be necessary to have previous knowledge about the system and the surrounding environment in order to develop appropriate reconfiguration methods. Another problem that can arise during control reconfiguration is buzzing. When a system alternates between controllers or loop configurations repeatedly without offering the desired control, it is referred to as chattering. Significant drops in performance could result from this.

3.2 Controller Method

More specifically, the controller is predicated on the idea that a computer system is improved by a Quality of Service (QoS) controller that i) keeps track of system performance, ii) keeps track of how the system's various resources are being used, and iii) periodically runs a controller algorithm known as controller intervals (CI) to find the optimal configuration for the system (refer to Figure 2) [14]. The controller algorithm generates reconfiguration directives as a result, telling the system to adjust its configuration.

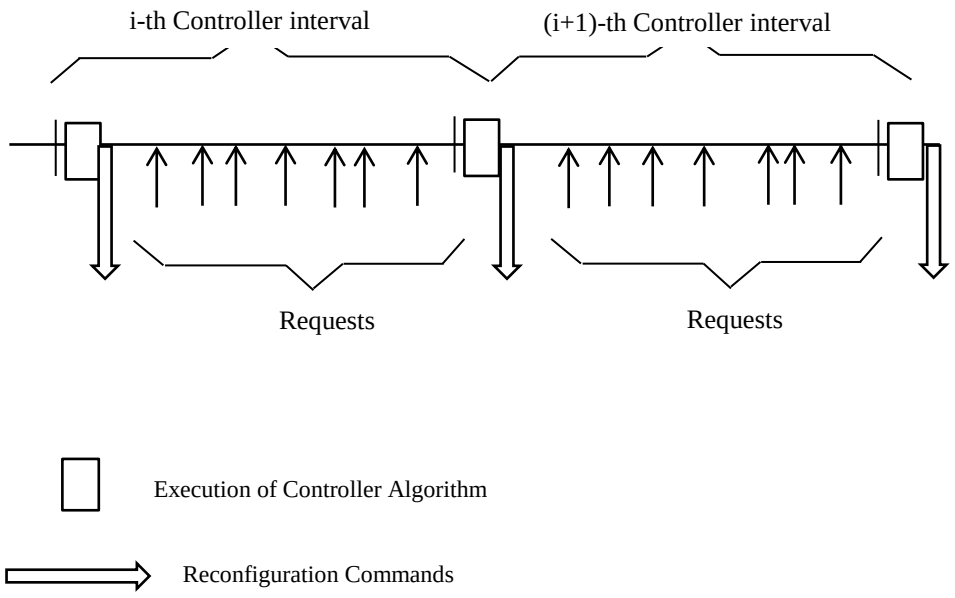

Figure 2. Intervals for the controllers

The most useful illustration of the QoS controller's architecture is provided by Figure 3. The four primary parts of the QoS controller are the Performance Model Solver (4), Workload Analyser (3), QoS Controller Algorithm (5), and Service Demand Computation (2). In order to calculate the throughput, the Service Demand Computation (2) component gathers utilisation statistics (1) on all system resources (such as CPU and discs) in addition to the number of completed requests (7). The ratio of resource utilisation to system throughput can be used to calculate the service demand of a request, or the overall average service time of a request at a resource. The Queuing Network (QN) model of the computer system, which is solved by the Performance Model Solver component, uses the service demands (8) that this component computes as input parameters.

The Workload Analyser (3) component examines the stream of incoming requests (6), computes average arrival rate and other workload intensity statistics, and forecasts the workload intensity for the upcoming controller interval using statistical methods. The workload intensity values (9) that this component computes, whether present or predicted are also used as input parameters for the Queuing Network model that the Performance Model Solver component (4) solves. The QoS Controller Algorithm sends requests (10) to this component to solve the QN model associated with a particular system configuration.

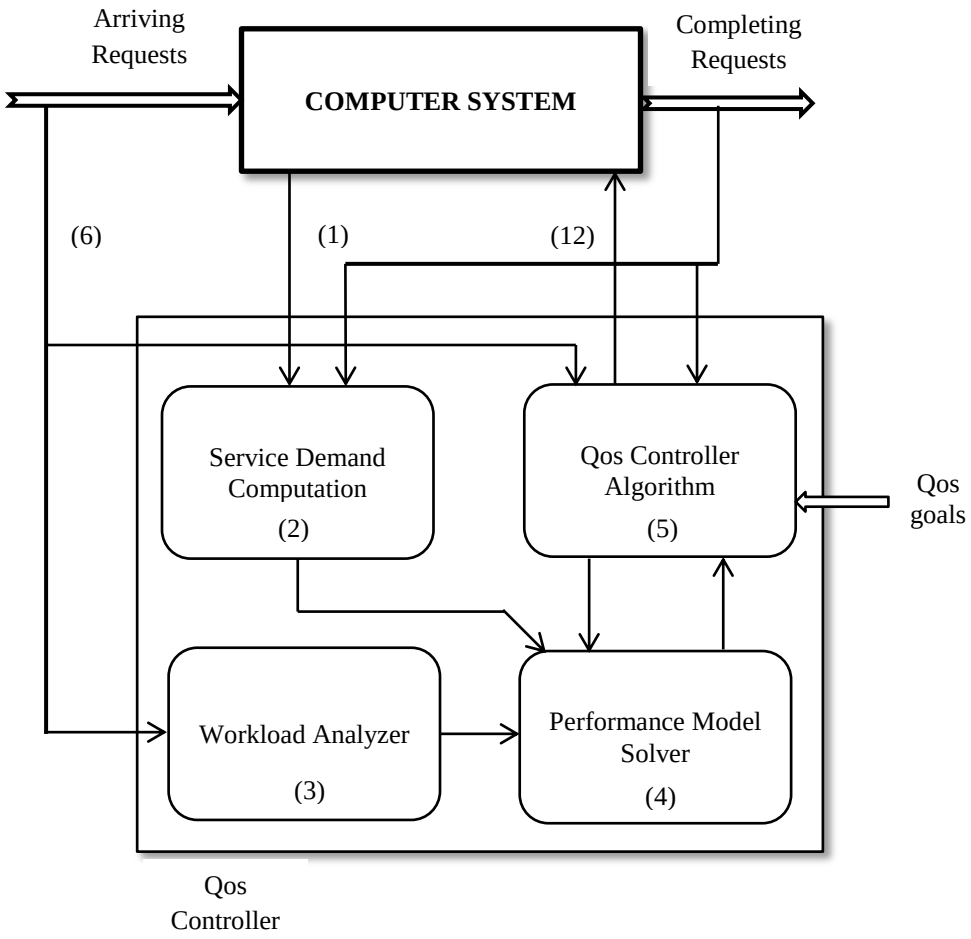

Figure 3. Structure of the QoS Manager

The Performance Model Solver must calculate this QoS value for every point in the configuration point space that the QoS controller algorithm looks at. The QoS controller sends reconfiguration commands (12) to the computer system after deciding on the optimal configuration for the workload intensity levels given by the Workload Analyser.

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

For the experiments pertaining to workload variability, Figure 4 shows the fluctuation of workload intensity, measured in requests/sec, as a function of time, measured in controller interval units. Each experiment lasted 30 CIs, or 60 minutes total because each CI was set for two minutes. The average service requests for the disc and CPU were 0.05 and 0.03 seconds, respectively [15]. Consequently, the system can sustain a maximum theoretical arrival rate of 20 req/sec.

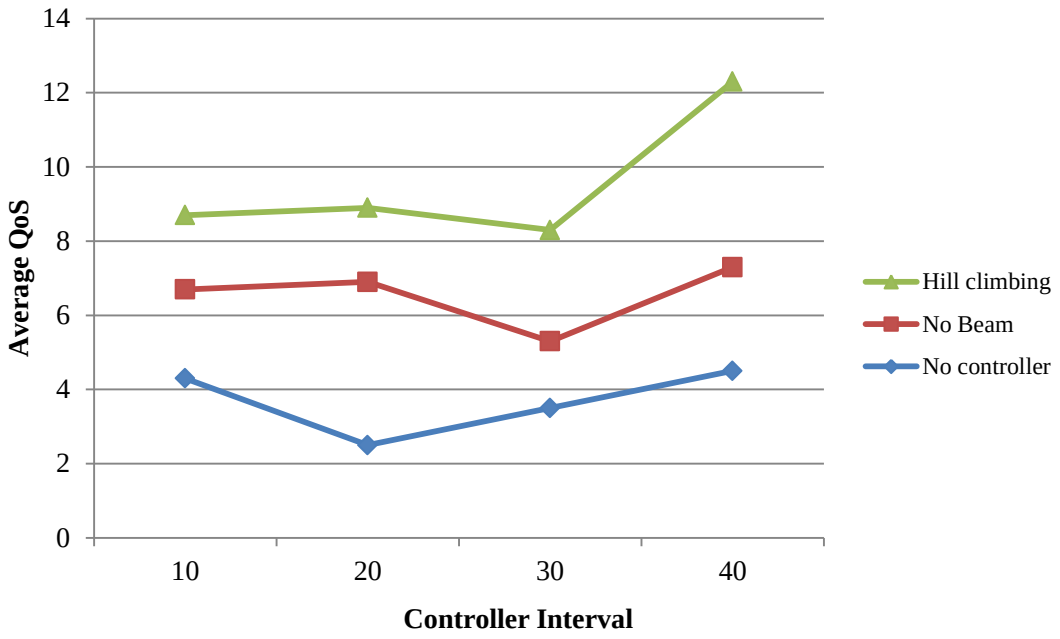

Figure 4. Variation in Workload Intensity for the High Variability Experiments

At CI = 19, the average arrival rate approaches the theoretical limit, peaking at 19 req/sec. It begins at a low value of 5 req /sec. After three CIs, the workload intensity remains at this level before beginning to decline towards 14 requests per second. For every combination of Ca and Cs, ten trials were conducted, and at the conclusion of each CI, 95% confidence intervals for the average QoS value were calculated. Three situations were found to have results for:

Two of them have the QoS controller turned on, while one has the controller turned off. The combinatorial optimisation methods that the controller uses—beam search and hill-climbing—differ between the two outcomes in which the controller is active. The results of our earlier investigation for the situation of exponentially dispersed inter-arrival and service times are displayed in Figure 5. Even at high peak loads, the controlled system maintains substantially higher QoS values than the uncontrolled system.

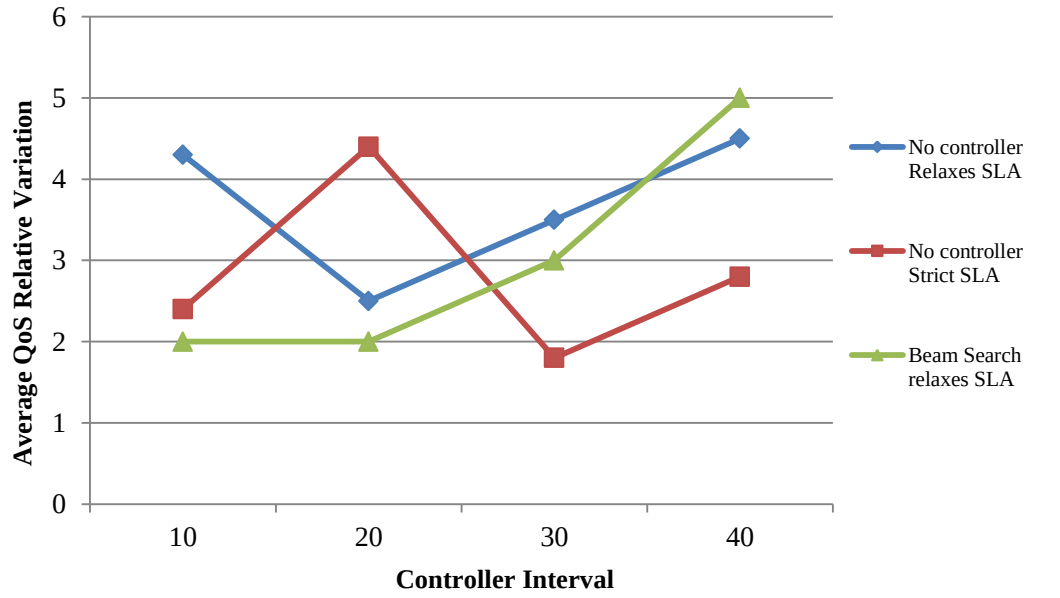

Figure 5. Impact of Tighter and Looser SLAs on Controller Performance

The QoS optimizer module in our proposed self-managing computer systems estimates the expected workload intensity for the upcoming controller interval (CI) based on the average arrival rate of requests from the previous CI. The performance model then uses this value to calculate the QoS worth for a particular

set of setup parameters. The disadvantage of this strategy is that it ignores any patterns of workload growth or decrease over the previous CI. As a consequence, there may be a significant error in estimating the next projected arrival rate and a poor selection of configuration parameters.

We introduced a module in charge of short-term workload forecasting to address this weakness. The most recent average arrival rates for the last N tiny sub-intervals are stored in this module using a sliding window of N values. These sub-intervals are all Δ seconds long. In order to ensure that N × Δ does not surpass the duration of a controller interval, N and Δ are selected. For short-term forecasting, numerous methods can be applied.

All types of data cannot, however, be accurately forecasted using any one method. Consequently, balanced moving averages, polynomial regression, and exponential smoothing are the three methods used by the forecasting module. Since exponential smoothing is well-known for being effective at producing predictions from time series data that show upward or downward trends, it was incorporated. The following is how exponential smoothing calculates a prediction: This indicates that there will be periods of time where the workload remains relatively steady for a considerable amount of time before shifting dramatically. In these cases, weighted moving averages are a useful strategy.

Since polynomials can approximate any continuous function very well, polynomial regression was selected as the third forecasting technique. The fitting gets better with increasing polynomial degree. However, we chose a reasonably high polynomial degree—six—in order to avoid significantly increasing the controller's overhead during the regression model computation.

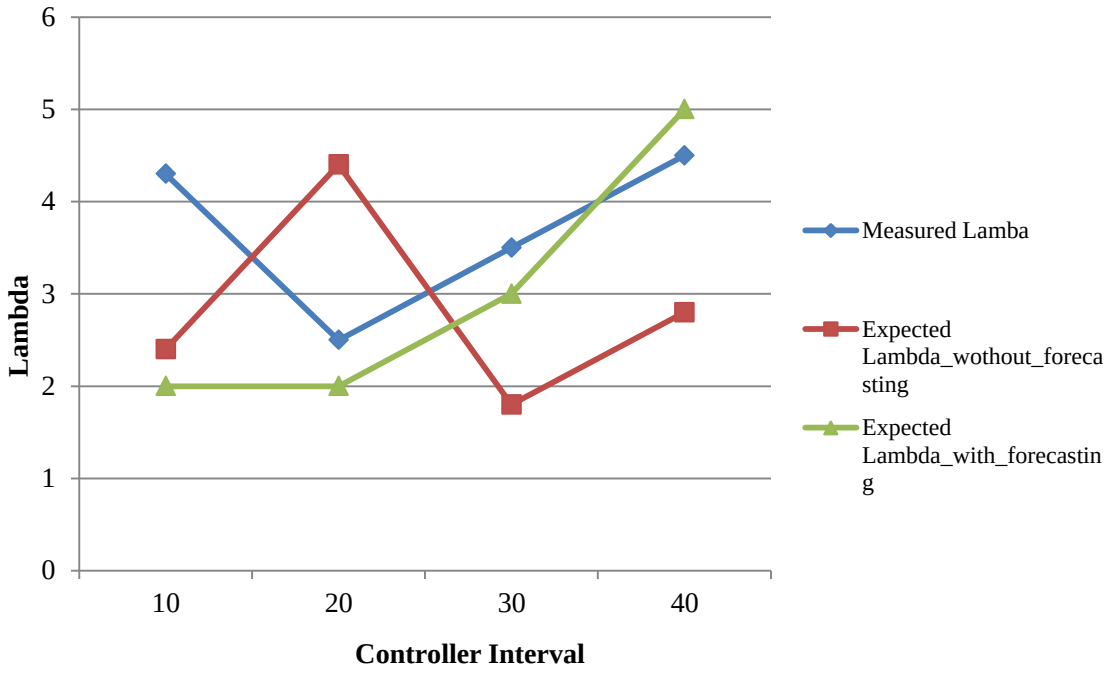

Figure 6. Variation in Workload Intensity for the Workload Forecasting Experiments

Figure 6 shows a comparison between the actual measured arrival rate and the projected arrival rate at each controller interval when the forecasting module was engaged or disabled. Please take note that the data in the sliding window for forecasting is only available at this time, which is why we begin at the second controller interval in this picture. Two peaks at 30 req/sec at CI = 8 and CI = 24 are present in the real workload.

With each new average arrival rate entry placed into the sliding window, all three models are reconstructed. We now calculate what the predicted value would be based on each of the three models. In order to evaluate the quality of the fits, we additionally calculate each model's R2 value using the least squares errors approach. At this point, the model with the highest R2 value forecasted the value that the forecasting module returns.

5. CONCLUSION

The argument made in this work is that self-managing systems can be constructed using control theory as an analytical and architectural foundation.

According to our experimental findings, the suggested method was successful in identifying a sizable number of controller failures that could have a detrimental effect on the self-adaptive system's resilience. Our method hasn't been able to find any catastrophic, restart, or hindering failures in the controller, despite the relevant amount of failures found. While the limited observability of the controller's internal behaviour may be the cause of this, other variables, such the controller's architectural resilience, may also be a likely explanation for the observed outcomes.

In this instance, knowing the precise QoS values of two places in the search space is less significant than accurately comparing them. The study's findings also demonstrate how workload forecasting, when applied to a regulated system, can enhance quality of service (QoS), particularly when the workload intensity is approaching saturation. Additionally, it was demonstrated that compared to the non-controlled system, the controlled system is far less sensitive to the SLA values.

REFERENCES

[1] Menascé, D. A., & Bennani, M. N. (2003, December). On the use of performance models to design self-managing computer systems. In Int. CMG Conference (pp. 1-9).

[2] Bennani, M. N., & Menascé, D. A. (2004, May). Assessing the robustness of self-managing computer systems under highly variable workloads. In International Conference on Autonomic Computing, 2004. Proceedings. (pp. 62-69). IEEE.

[3] Diao, Y., Hellerstein, J. L., Parekh, S., Griffith, R., Kaiser, G. E., & Phung, D. (2005). A control theory foundation for self-managing computing systems. IEEE journal on selected areas in communications, 23(12), 2213-2222.

[4] Bhat, V., Parashar, M., Liu, H., Khandekar, M., Kandasamy, N., & Abdelwahed, S. (2006, June). Enabling self-managing applications using model-based online control strategies. In 2006 IEEE International Conference on Autonomic Computing (pp. 15-24). IEEE.

[5] Alaya, M. B., Matoussi, S., Monteil, T., & Drira, K. (2012, September). Autonomic computing system for self-management of machine-to-machine networks. In Proceedings of the 2012 international workshop on Self-aware internet of things (pp. 25-30).

[6] Khandekar, M. D., Kandasamy, N., Abdelwahed, S., & Sharp, G. C. (2005). An online predictive control framework for designing self-managing computing systems. Multiagent and Grid Systems, 1(2), 63-72.

[7] Bai, J. (2008). A model integrated framework for designing and optimization of self-managing computing systems (Doctoral dissertation).

[8] Boutaba, R., & Xiao, J. (2007). Self-Managing Networks. Cognitive Networks: Towards Self-Aware Networks, 77-95.

[9] Kandasamy, N., Abdelwahed, S., Sharp, G. C., & Hayes, J. P. (2004, June). An online control framework for designing self-optimizing computing systems: Application to power management. In Self-star Workshop (pp. 174-188). Berlin, Heidelberg: Springer Berlin Heidelberg.

[10] Cámara, J., De Lemos, R., Laranjeiro, N., Ventura, R., & Vieira, M. (2014). Testing the robustness of controllers for self-adaptive systems. Journal of the Brazilian Computer Society, 20, 1-14.

[11] Patikirikorala, T., Colman, A., Han, J., & Wang, L. (2011, May). A multi-model framework to implement self-managing control systems for QoS management. In Proceedings of the 6th international symposium on software engineering for adaptive and self-managing systems (pp. 218-227).

[12] Nijim, M., Xie, T., & Qin, X. (2005). Performance analysis of an admission controller for CPU-and I/O-intensive applications in self-managing computer systems. ACM SIGOPS Operating Systems Review, 39(4), 37-45.

[13] Reich, C., Bubendorfer, K., & Buyya, R. (2007). A Scalable Self-Managing Architecture for WSRF Services.

[14] Khandekar, M. D., Kandasamy, N., Abdelwahed, S., & Sharp, G. C. A Control-based Framework for Self-Managing Distributed Computing Systems.

[15] Menascé, D. A. (2020). Self-managed computer systems: Foundations and examples. In Enterprise Information Systems: 21st International Conference, ICEIS 2019, Heraklion, Crete, Greece, May 3–5, 2019, Revised Selected Papers 21 (pp. 17-36). Springer International Publishing.