**International Journal of Pure and Applied Mathematics**
**Volume 120 No. 5 2018, 939-953**
ISSN: 1314-3395 (on-line version)
**url:** http://www.acadpubl.eu/hub/
Special Issue

http://www.acadpubl.eu/hub/

# Enhanced Self Adaptive Dynamic Software Architecture with Heuristic Approach

Mr. Sridhar Gummalla, Dr. G. Venkateswara Rao , Dr. G. V. Swamy

[1]Computer Science & Engineering, Shadan College of engineering & technology, India.

[2]Gitam Institute Of Technology, Gitam University, India.

[3]Gitam Institute Of Technology, Gitam University, (India.)

E-mail:sridhargummalla1975@gmail.com[1], drgv_swamy@yahoo.co.in[2], vrgurrala@yahoo.com[3]

Abstract

The Enhanced Self-Adaptive Dynamic Software Architecture (ESA-DSA) proposed by us in our prior work is enhanced in this paper with heuristic based approach that makes use of both historical data and QoS needs in order to have knowledge based self-adaptation. An algorithm by name Heuristic Self-Adaptation (HAS) is proposed and implemented. AWS cloud is used to have experiments on real time target system that serve millions of users with Service Level Agreements (SLAs). Influenced by Rainbow framework, the enhanced self-adaptive framework performs QoS and knowledge based approach to save heuristics that help in making well informed decisions from time to time. We built a prototype application to demonstrate proof of the concept. The experimental results revealed that the ESADSA is effective in distributed computing

Index Terms – Software architecture, Dynamic Software Architecture, self-adaptation

## 1. Introduction

Software systems tend to change from time to time. Based on the specified objectives, changed requirements and criteria for change and the necessity of self-adaptation software engineers need to work towards fulfilling the system right from the beginning. However systems are generally built in the real world without considering self-adaptation. In other words, software systems that have been built without self-adaptation throw challenges to software engineers as

they need to make existing software to be self-adaptive. When existing software needs to be made self-adaptive, it is essential for the engineers to have a framework that decouples the functionality of self-adaptation from the target system. Literature found in [1]-[18] showed contributions of researchers on self-adaptation.
In our previous work, we proposed Enhanced Self-Adaptive Dynamic Software Architecture (ESADSA) and evaluated with a case study target system deployed in AWS cloud. This paper covers details of further enhancement of ESADSA with an algorithm known as Heuristic Self-Adaptation (HSA) which makes use of the mined knowledge from time to time and coordinates with the two algorithms to have further optimization of the proposed self-adaptation architecture. It makes use of heuristics gained from the adaptation events and helps improve the accuracy of decision making in the presence of different workloads. It presents experimental results with a real time application which is distributed in nature and deployed in AWS cloud. The framework makes the application self-adaptive with loosely coupled communication between the application and the proposed architecture. The remainder of the paper is structured as follows. Section 2 provides details of decision heuristic. Section 3 presents the proposed self-adaptive dynamic software architecture. Section 4 presents heuristic self-adaptation algorithm. Section 5 presents experimental results while section 6 concludes the paper and provides directions for future work.

## 2. Decision Heuristics

Heuristic is one of the methods used to make decision. More complex optimization models can be used to achieve heuristic. Many studies found that there are approaches related to heuristics to make judgments. Probabilities of events can be estimated with heuristics. In the same fashion, it is important to know that heuristics are very simpler than optimizing methods that are analogous to heuristic. They can lead to optimal judgments. Heuristics also have negative side when there is bias in the heuristic model and cause issues when decisions are made based on that. Some heuristics focus on representativeness. It is to related events with each other. Then the resemblance is used to make judgments. This kind of heuristic leads to judgments that are correct. There is an issue with it when resemblance leads to error. Therefore, it is not correct to make judgments on the basis of resemblance only. Cause and effect is another kind of heuristic makes decisions based whether an event caused by certain cause. It also may result an error when incorrect cause is chosen by mistake.
Random process when used for heuristic, it may not result in randomness always a flip of the coin may result same twice or thrice. Therefore representative heuristics may lead to correctness often but there is possibility of error. There is availability heuristic that is used to

build knowledge related availability of certain objects. There are many factors related to availability of objects. When availability heuristic is used, it may often lead to correct conclusions. At the same time, it can create errors too. Both representativeness and availability heuristics can be combined to have conjunctive fallacy. It can occur due to either availability or representativeness. It is understood that the representativeness and availability heuristics may result in wrong decisions. Towards this end vividness criterion is the reason for misuse of the two heuristics. Anchoring heuristic is yet another kind of heuristic where people tend to adjust their estimates. Estimates start with initial value and then subjected to adjustments based on the conditions surrounding.

## 3. Proposed Self-Adaptive Dynamic Architecture

In case of complex software systems, it is inevitable to have a framework that contains explicit model for self-adaptation. It should have targeting mechanisms that need to take care of runtime changes needed. When a problem arises, it should trigger adaptation mechanism with a model that can make decision on the course of action based on the problem arose. Once it is initiated, the mechanisms need to propagate changes in the target system. External systems that control a target system have many benefits over internal control systems. The rationale behind this is that the external system can achieve separation of concerns. In other words, it can achieve self-adaptation by having loosely coupled interaction with the target system. This can help even to convert legacy systems to have self-adaptive capabilities. Self-adaptive systems are therefore important to have external mechanisms with strategy selection, problem detection, model management and forcing adaptations. These modules are to be reused across the target systems.

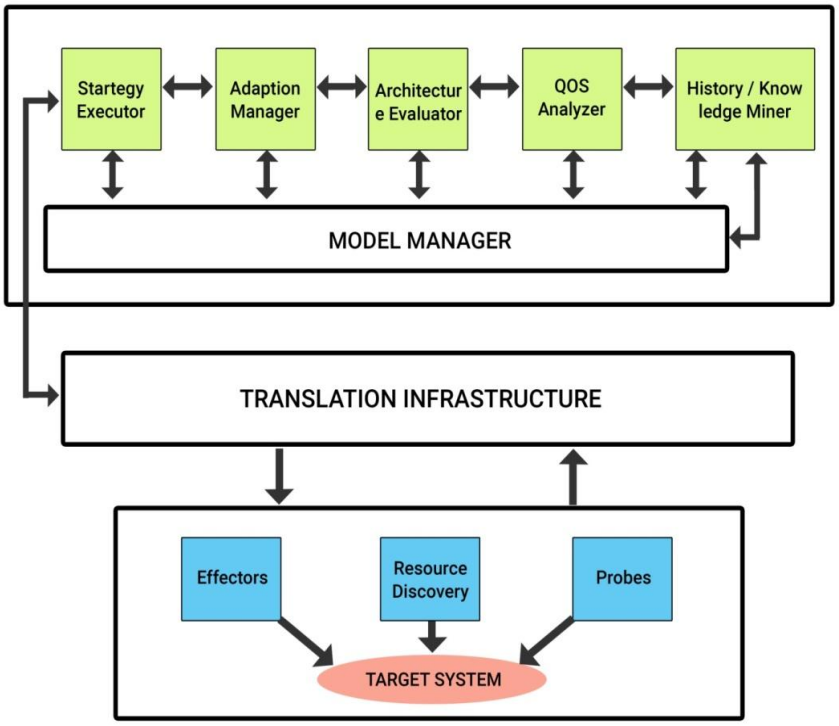


Figure 1: Proposed Self-Adaptive software architecture

The proposed self-adaptation architecture is as shown in Figure 1. It contains the system layer and architectural layer. In the architectural layer two important modules are used to extend the functionality of original Rainbow framework.

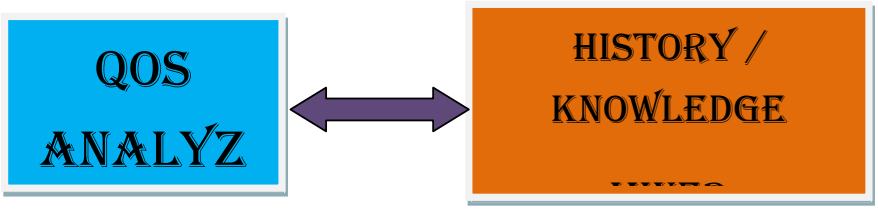


Figure 2: QoS analyzer and knowledge miner modules

Here QoS analyzer and knowledge miner modules are subjected to changes using heuristic approach. Before providing details of the heuristic approach used in this chapter, here are some important heuristic approaches for decision making. QoS depends on the need of an application. When there are applications that need self-adaptation with required QoS, it is essential to understand the

parameters. Quality parameter such as latency is used to analyze QoS and also mine history of the previous incidents and corresponding adaptations. In the case of distributed applications that run in cloud, there are possible Service Level Agreements (SLAs) between service providers and users. Latency is one of the SLAs that are considered for the experiments. Latency is the time taken to retrieve required data from server. The latency time when considered, it is important for the target system to have analysis and make necessary steps to have self adaptation.

The QoS analyzer presented in the previous chapter provides information related to QoS analyzer module. In the same fashion, there is another module described in the previous chapter that is responsible for the analysis of history of adaptations that have been made earlier for the purpose of understanding knowledge and make well informed decisions. Knowledge mining is thus plays an important role in self adaptation. It considers different parameters and the latency of the previous adaptations in different contexts. Apart from this, in this chapter, heuristic approach is used to have incremental knowledge and use it for well informed decision making. Before presenting the proposed algorithm, theoretical concept of decision heuristics is described in the ensuing section.

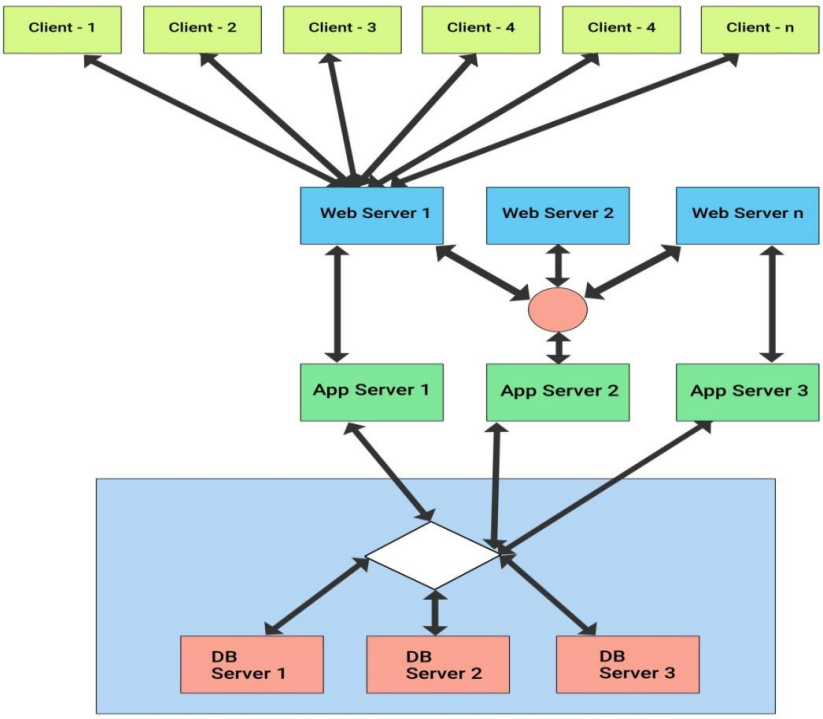


Figure 3: Shows target system architecture (deployed in AWS cloud)

As shown in Figure 3, it is evident that the target system is a distributed C/S application that is deployed in AWS cloud. The system supports millions of users across the globe. However, the services rendered to users may change based on service level agreements (SLAs). The agreements between users and service providers may dynamically change from time to time. This is the reason why this target system needs self-adaptation.

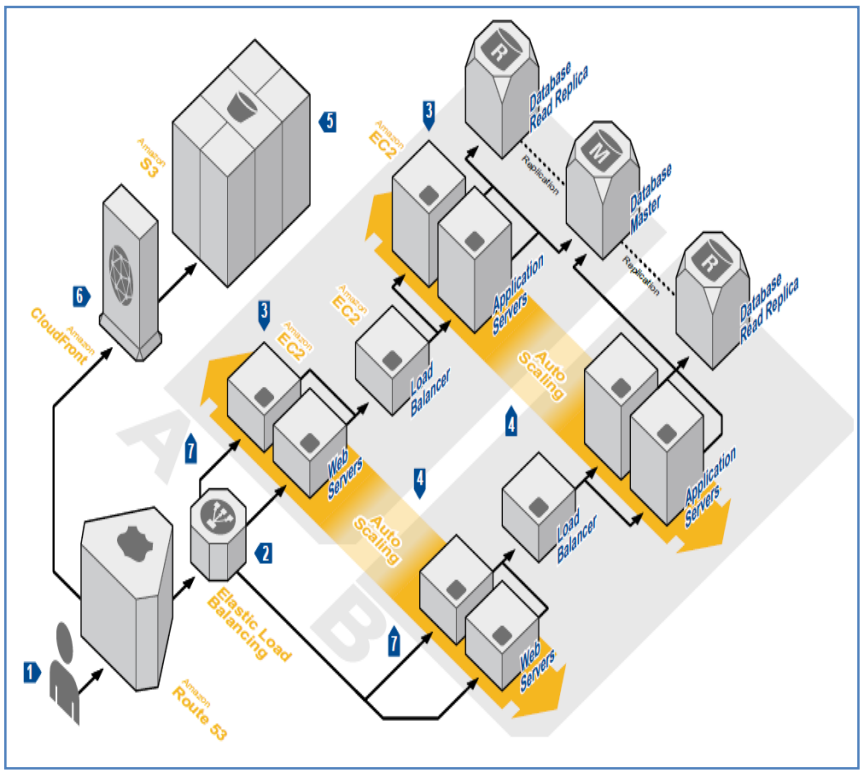


Figure 4: AWS reference architecture with web servers, application servers and database servers

As shown in Figure 4, the target system deployed is running in the environment provided in AWS. It makes use of databases, application servers and web servers. However, it needs run time self-adaptation to be achieved to work in tune with the requests and SLAs. Heuristic adaptation is the knowledge based adaptation used to improve the qualitative approach used by ESADSA.

## 4. Heuristic Self-Adaptation

A heuristic self-adaptation algorithm is proposed to have decision heuristics for self-adaptation. This algorithm works based the three parameters provided to it. The first parameter is to QoS needs of the application or target system while rendering its services to its users.

The second parameter is the present state and the third parameter is the goal state that can meet the given QoS as per SLA related to latency. Therefore the first parameter is latency which is to be achieved.

Algorithm: Heuristic Self Adaptation

Inputs: latency lat, Present state P, goal state G

Output: self-adaptation to achieve latency

```
1.  Initialize resources vector R
2.  Initialize resource used vector RU
3.  Initialize estimated latency eLat
4.  While not G = P
5.    R=findReourcesAvailable()
6.    RU=findResourceUsed()
7.  eLat=estimateLatency(R, P)
8.    If eLat<=lat then
9.      Do nothing
10.     Save P
11.   Else
12. Self-Adaptation(P, R, RU)
13.     Save P
14.   End If
15. End While
```

Algorithm1: Heuristic Self Adaptation

As shown in Algorithm 1, it is evident that the algorithm takes the three aforementioned parameters and continuously monitors the present state of the target system and use heuristic to achieve the goal state that can help in achieving SLA.

```
Algorithm: Self-Adaptation
Inputs : P, R, RU
Output: Self Adaptation Decision
    1.  If SVM classifier is already built then
    2.      Update the model
    3.  Else
    4.      Load training set
    5.      Build SVM classifier with training set
    6.      Obtain class label
    7.      If class label = 1 then
    8.          If R-RU supports then
    9.              Invoke one connected server group
    10.         End if
    11.     Else label = 2 then
    12.         Invoke two connected server groups
    13.     End if
    14. End If
```

Algorithm 2: Self Adaptation

As presented in algorithm 2, the SVM (Support Vector Machines) is one of the popular data mining algorithms that is used to have heuristic to gain business intelligence to make well informed decisions. The SVM classifier is built with previous history of states, parameters and the decisions made. The SVM classifier generates a model that is considered as required heuristic and that gets updated from time to time. Thus it can provide required business intelligence for self adaptation. Since SLAs provides guarantee of services to cloud users, it is important for the target application to self-adapt. The target application is described in Chapter 4. It is a distributed application deployed in AWS cloud. It has millions of users. When concurrent users need the latency as per the SLA, the system needs to adapt to the situation and make well informed decisions related to resource availability and resource allocation to ensure that the customers are given service with the committed latency.

# 5. Experimental Results

This section provides experimental results with respect to self-adaptation of target system with respect to the latency, depends on the service level agreements.

Table 1 Latency at different elapsed times

| Latency at different Elapsed Times | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 150 | 300 | 450 | 600 | 750 | 900 | 1050 | 1200 | 1350 | 1500 | 1650 | 1800 |
| 0 | 0.9 | 0 | 0.9 | 3 | 3 | 0 | 0.9 | 0.8 | 0 | 0.9 | 0.9 |
| 0.3 | 0.7 | 0.3 | 0.7 | 3.5 | 2.4 | 0.3 | 0.7 | 0.8 | 0.3 | 0.7 | 0.9 |
| 0.5 | 0.6 | 0.5 | 0.6 | 3.5 | 3 | 0.5 | 0.6 | 0.8 | 0.5 | 0.6 | 0.9 |
| 0.7 | 0.5 | 0.7 | 0.5 | 2.9 | 2.3 | 0.7 | 0.5 | 0.8 | 0.7 | 0.5 | 0.9 |
| 0.8 | 0.9 | 0.8 | 0.9 | 3 | 3 | 0.8 | 0.9 | 0.8 | 0.8 | 0.9 | 0.9 |
| 0.3 | 0.7 | 0.3 | 0.7 | 2 | 2 | 0.3 | 0.7 | 0.8 | 0.3 | 0.7 | 0.9 |
| 0.5 | 0.6 | 0.5 | 0.6 | 3.2 | 3.1 | 0.5 | 0.6 | 0.8 | 0.5 | 0.6 | 0.9 |
| 0.7 | 0.5 | 0.7 | 0.5 | 3 | 3 | 0.7 | 0.5 | 0.8 | 0.7 | 0.5 | 0.8 |

As shown in Table 1, it is understood that different elapsed times such as 150 to 1800 with increment of 150 seconds is used to observe latency dynamics. The results revealed that the proposed heuristic method is able to adapt to runtime situations.

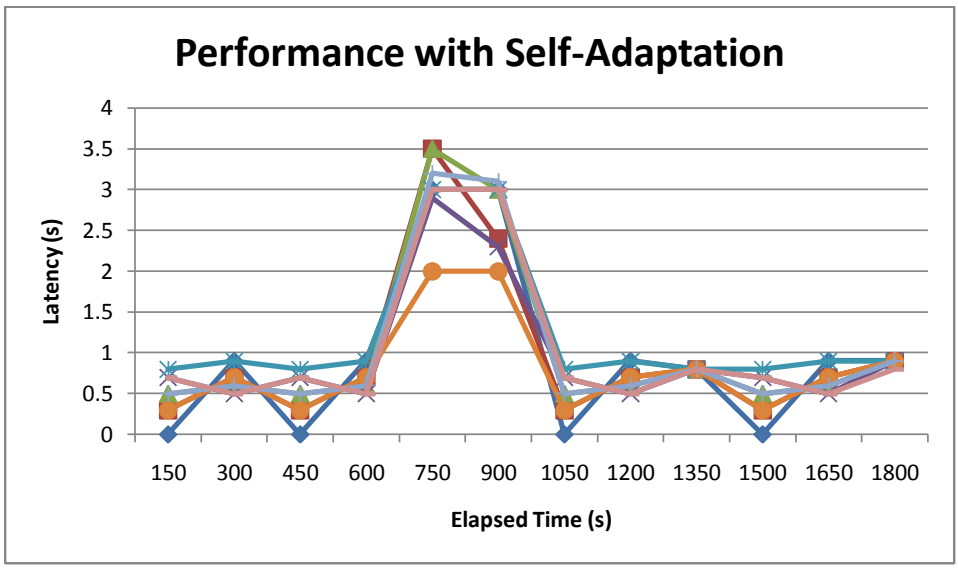


Figure 5: Performance with self-adaptation

As shown in Figure 5, it is understood that different elapsed times such as 150 to 1800 with increment of 150 seconds is used to observe latency dynamics. The results revealed that the proposed heuristic method is able to adapt to runtime situations. The results are with self-adaption. The results without self-adaption are shown in Table 2 and Figure 6 below.

Table 2: Latency with self-adaptation using heuristic approach

| Latency at different elapsed times of self-adaptation with heuristic approach | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 150 | 300 | 450 | 600 | 750 | 900 | 1050 | 1200 | 1350 | 1500 | 1650 | 1800 |
| 0.6 | 0.9 | 0.3 | 0.8 | 0.4 | 0.4 | 0.6 | 0.75 | 0.8 | 0.3 | 0.9 | 0.9 |
| 0.35 | 0.7 | 0.3 | 0.7 | 0.6 | 0.6 | 0.3 | 0.7 | 0.8 | 0.3 | 0.7 | 0.7 |
| 0.5 | 0.6 | 0.5 | 0.6 | 0.3 | 0.3 | 0.5 | 0.6 | 0.8 | 0.5 | 0.6 | 0.8 |
| 0.7 | 0.5 | 0.7 | 0.5 | 0.26 | 0.25 | 0.7 | 0.5 | 0.8 | 0.7 | 0.5 | 0.5 |
| 0.65 | 0.9 | 0.8 | 0.9 | 0.7 | 0.7 | 0.8 | 0.9 | 0.8 | 0.8 | 0.9 | 0.3 |
| 0.3 | 0.7 | 0.3 | 0.7 | 0.4 | 0.4 | 0.3 | 0.7 | 0.8 | 0.3 | 0.7 | 0.9 |
| 0.5 | 0.6 | 0.5 | 0.6 | 0.3 | 0.3 | 0.5 | 0.6 | 0.8 | 0.5 | 0.6 | 0.9 |
| 0.4 | 0.5 | 0.7 | 0.5 | 0.6 | 0.6 | 0.7 | 0.5 | 0.8 | 0.7 | 0.5 | 0.8 |

When compared with self-adaptation details given in the Table 2, the results revealed that there is certain issue with service level agreement.

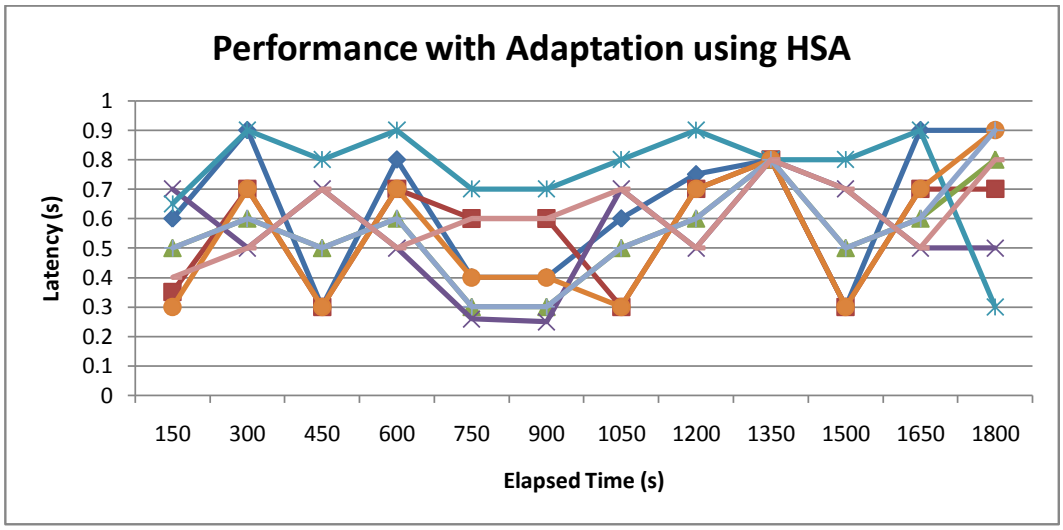


Figure 6: Latency with self-adaptation using heuristic approach

When compared with self-adaptation details given in the Figure 5.2, the results revealed that there is certain issue with service level agreement. Therefore it is understood that the heuristic approach for self-adaptation can provide advantage in achieving desired results without making any manual efforts.

| Latency at different elapsed times between self adaptation and self adaptation with HAS approaches | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 150 E | 150 P | 300 E | 300 P | 450 E | 450 P | 600 E | 600 P | 750 E | 750 P | 900 E | 900 P |
| 0 | 0.6 | 0.9 | 0.9 | 0 | 0.3 | 0.9 | 0.8 | 3 | 0.4 | 3 | 0.4 |
| 0.3 | 0.4 | 0.7 | 0.7 | | 0.3 | 0.7 | 0.7 | 3.5 | 0.6 | 2.4 | 0.6 |
| 0.5 | 0.5 | 0.6 | 0.6 | 0.5 | 0.5 | 0.6 | 0.6 | 3.5 | 0.3 | 3 | 0.3 |
| 0.7 | 0.7 | 0.5 | 0.5 | 0.7 | 0.7 | 0.5 | 0.5 | 2.9 | 0.3 | 2.3 | 0.25 |
| 0.8 | 0.7 | 0.9 | 0.9 | 0.8 | 0.8 | 0.9 | 0.9 | 3 | 0.7 | 3 | 0.7 |
| 0.3 | 0.3 | 0.7 | 0.7 | 0.3 | 0.3 | 0.7 | 0.7 | 2 | 0.4 | 2 | 0.4 |
| 0.5 | 0.5 | 0.6 | 0.6 | 0.5 | 0.5 | 0.6 | 0.6 | 3.2 | 0.3 | 3.1 | 0.3 |
| 0.7 | 0.4 | 0.5 | 0.5 | 0.7 | 0.7 | 0.5 | 0.5 | 3 | 0.6 | 3 | 0.6 |

| Latency at different elapsed times between self adaptation and self adaptation with HAS approaches | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1050 E | 1050 P | 1200 E | 1200 P | 1350 E | 1350 P | 1500 E | 1500 P | 1650 E | 1650 P | 1800 E | 1800 P |
| 0 | 0.6 | 0.9 | 0.75 | 0.8 | 0.8 | 0 | 0.3 | 0.9 | 0.9 | 0.9 | 0.9 |
| 0.3 | 0.3 | 0.7 | 0.7 | 0.8 | 0.8 | 0.3 | 0.3 | 0.7 | 0.7 | 0.9 | 0.7 |
| 0.5 | 0.5 | 0.6 | 0.6 | 0.8 | 0.8 | 0.5 | 0.5 | 0.6 | 0.6 | 0.9 | 0.8 |
| 0.7 | 0.7 | 0.5 | 0.5 | 0.8 | 0.8 | 0.7 | 0.7 | 0.5 | 0.5 | 0.9 | 0.5 |
| 0.8 | 0.8 | 0.9 | 0.9 | 0.8 | 0.8 | 0.8 | 0.8 | 0.9 | 0.9 | 0.9 | 0.3 |
| 0.3 | 0.3 | 0.7 | 0.7 | 0.8 | 0.8 | 0.3 | 0.3 | 0.7 | 0.7 | 0.9 | 0.9 |
| 0.5 | 0.5 | 0.6 | 0.6 | 0.8 | 0.8 | 0.5 | 0.5 | 0.6 | 0.6 | 0.9 | 0.9 |
| 0.7 | 0.7 | 0.5 | 0.5 | 0.8 | 0.8 | 0.7 | 0.7 | 0.5 | 0.5 | 0.8 | 0.8 |

Table 3: Latency comparison with existing system on self-adaptation

As shown in Table 3, the latency is recorded at different elapsed times for some requests processed by both existing and proposed approaches.

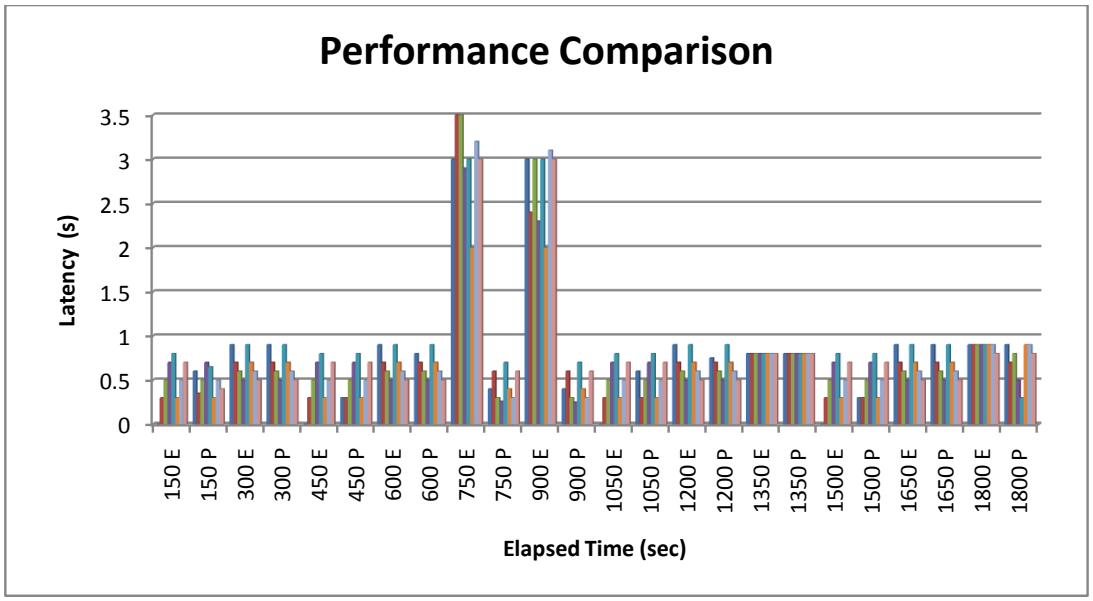


Figure 7: Self-adaptation performance comparison of the proposed with existing system

When compared with self-adaptation details of existing and proposed as given in the Figure 7, the results revealed that there is certain issue with service level agreement that is latency should be less than or 1 second for both approaches. When there is the issue encountered, both performed adaptation. However, the proposed adaptation strategy appeared better than the existing one. The results of existing system are taken from [161]. The existing system does not have heuristic approach but self-adaptation strategy is there. The results show the performance improvement over the existing system.

## 6. Conclusions And Future Work

In this paper we proposed heuristic approach followed with ESADSA architecture. The algorithm is known as Heuristic Self Adaptation (HSA) algorithm. The algorithm takes care of QoS needs and history availability to update a model containing heuristics to leverage decision making process. This algorithm is also tested in the Amazon AWS cloud with the deployed web based application. The results are observed with HSA and without HSA and found significant difference in the optimizations. The application with comprehensive functionality of the ESADSA proved to be more efficient in self-adaptation automatically and dynamically. The experimental results revealed the increased utility of the architecture proposed. In future we intend to make ESADSA to have more sophisticated and comprehensive approaches to various SLAs to be considered for self-adaptation.

# References

[1].Michael M. Gorlick and Rami R. Razouk. Using Weaves for software construction and analysis. In Proc. of the 13th International Conf. of Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, USA,(1991), 23-34

[2].John C. Knight, Dennis Heimbigner, Alexander L. Wolf, Antonio Carzaniga, Jonathan C. Hill, PremkumarDevanbu, and Michael Gertz. The Willow survivability architecture.In Proc. of the 4th Information Survivability Workshop, (2001), 1-14.

[3].PeymanOreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, NenadMedvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptative software.IEEE Intelligent Systems, 14(3), (1999), 54- 62.

[4].Dale E. Seborg, Thomas F. Edgar, and Duncan A. Mellichamp. Process Dynamics and Con-trol. Wiley Series in Chemical Engineering. John Wiley & Sons, New York, (1989), 1-13.

[5].IBM. An architectural blueprint for autonomic computing, (2004), 1-14.

[6].VahePoladian. Tailoring Configuration to User's Tasks under Uncertainty. PhD thesis, Carnegie Mellon University School of Computer Science, 5000 Forbes Avenue, Pittsburgh, PA 15213, (2008), 1-24.

[7].David Garlan, Jeff Kramer, and Alexander Wolf, editors. Proc. of the 1st ACM SIGSOFT Workshop on Self-Healing Systems (WOSS '02), ACM Press, New York, (2002), 1-19.

[8].Mary Shaw and David Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall, (1996), 1-12.

[9].David Garlan, Robert T. Monroe, and David Wile. Acme: Architectural descriptions of component-based systems. In Gary T. Leavens and MuraliSitaraman, editors, Foundations of Component-Based Systems, Cambridge University Press, Cambridge,(2000), 47-68.

[10] Eric M. Dashofy, Andre« van der Hoek, and Richard N. Taylor. A highly-extensible, XML-based architecture description language. In Proceedings of WICSA2, Massachusetts, USA, Kluwer Academic Publishers, New York, (2001), 28-31.

[11].David Garlan, Shang-Wen Cheng, and Bradley Schmerl. Increasing system dependability through architecture-based self-repair. In Rogerio de Lemos, Cristina Gacek, and Alexander Romanovsky, editors, Architecting Dependable Systems, Lecture Notes in Computer Science, Springer-Verlag, Inc. New York, (2003), 61-89.

[12] Gregory D. Abowd, Robert Allen, and David Garlan. Formalizing style to understand descrip-tions of software architecture.ACM Trans. Softw. Eng. Methodol., 4(4), (1995), 319-364.

[13]Peter H. Feiler, Bruce Lewis, and Steve Vestal.Improving predictability in embedded real-time systems. Technical Report CMU/SEI-2000-SR-011, Carnegie Mellon University Software Engineering Institute, Pittsburgh, PA 15213, (2000), 1-16.

[14]Bridget Spitznagel and David Garlan.Architecture-based performance analysis. In Proc. of the 10th International Conf. on Software Engineering and Knowledge Engineering, Knowledge Systems Institute, (1998), 146-151.

[15]Robert Allen, Steve Vestal, Dennis Cornhill, and Bruce Lewis.Using an architecture description language for quantitative analysis of real-time systems.In Proc. of the 3rd International Workshop on Software and Performance, ACM Press, (2002), 203-210.

[16]Jeff Magee and Jeff Kramer.Dynamic structure in software architectures. In SIGSOFT '96: Proc. of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering, ACM, New York, (1996), 3-14.

[17]Robert J. Allen. A Formal Approach to Software Architectures.PhD thesis, Carnegie Mellon University School of Computer Science, (1997), 1-16.

[18]Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. Documenting Software Architecture: Views and Beyond. Pearson Education, Inc, (2003), 1-14.