Improve Quality of Systems via Dynamic Cost-Aware Selection of Web-Services

Ali Jahani
Department of Electrical and Computer Engineering
University of Tehran
Tehran, Iran
0009-0000-2756-7136

Fattaneh Taghiyareh
Department of Electrical and Computer Engineering
University of Tehran
Tehran, Iran
0000-0003-2039-8100

***Abstract*—Due to the dynamic performance of web services, a pivotal step in ensuring the quality of our system is dynamic service selection. The study aims to reach a more precise objective function for the problem and improve it by considering the cost of service selection. The method we propose takes into account the expenses associated with service selection by executing the service selection process to determine the optimal services solely in instances where there is a notable change in service metrics. To gather the data required for evaluating this work, we used a virtual server provided by Hetzner Online GmBH and collected information about round-time trip by using the global ping network. By implementing this method, we lower the number of service selections, which might slightly decrease the quality of service. We believe our method could lead to a higher overall score in objective function.**

***Keywords*—Web service, Service Selection, Quality-of-Service (QoS), Time-aware, Cost-aware**

I. Introduction

Recently, a large number of web services on the internet are available and many of these web services have similar functionality, which makes selecting a set of web services from a multitude of available options and a challenge [1]. To address this problem, an automated process called service selection is required. The fluctuating performance of web services has led to the introduction of dynamic web service selection [15].

Dynamic web service selection has potential applications in suggesting services for real-time mobile applications, interactive online games [2] or in recommending a new web service composition by predicting its future QoS [3].

Nevertheless, in some scenarios, transitioning between services or developing a new execution plan may incur significant expenses. We dubbed the set of costs in the process of web service selection the selection cost. An example of an expensive web service selection is in streaming media, where the cost of switching web services involves reconnecting users from one server to another, resulting in interruptions that negatively impact the user experience [10]. Therefore, it seems reasonable to find the optimal balance between migrating to better-performing services and the associated costs of such transitions.

For that reason, we introduced a web service selection method for detecting changes in the environment that considers the cost of service selection itself. This study showed that this method can improve the overall cost of our objective by lowering the number of service selections and slightly decreasing the quality of the selected services and in some cases improving all the criteria in our problem.

The rest of this paper is outlined as follows: In Section 2, we introduce two scenarios where switching between web services incurs notable costs. Section 3 introduces a general form for cost-aware dynamic web service selection. Section 4 is the description of an algorithm for finding change points, which is a more concrete implementation of pseudocode in its previous section. Section 5 describes experiments for evaluating the algorithm mentioned in its previous section. In section 6, we describe the results of the experiment. Section 7 concludes the paper.

II. Cost of Service Selection

In this section, we discuss the possible scenarios in which selecting a new service could result in notable costs.

Ultra-low latency streaming: In some live-streaming applications, such as interactive multimedia streaming and video conferencing, sub-second latency is required and is referred to as ultra-low latency streaming [5, 6]. In some cases, like casual gaming, a 100-millisecond change in delay can change the experience of users from acceptable to unacceptable for the majority of people [5]. In the case of web conferencing, a higher mouth-to-ear delay is correlated to the probability of speech overlapping, therefore lowering the percentage of speech time that is considered useful [8]. Consequently, maintaining low latency is essential for ensuring a satisfactory user experience.

To lower the requirements for the user upload bandwidth, we might want to use a Selective Forwarding Unit (SFU) service. Users can send the media stream to this server, and SFU will forward the stream to relevant participants in the conference [9]. This architecture is used in many web conferencing server implementations, such as Jitsi [10] and Odoo Discuss [11]. In this case, as the response time of an SFU service changes, we can switch to the better performing server in the case of finding one. However, this change causes an interruption in the stream, which will affect the quality of experience for the users [7]. In this scenario, service switching is the cost.

QoS aware web service composition: In the case of web service composition, it is known that creating a new Quality of Service (QoS) aware execution plan is an NP-hard problem. To address this, genetic algorithms and other meta-heuristic techniques are typically utilized [4], making frequent re-compositions either costly or impractical. In this scenario, evaluation of changes in the QoS to find the better services is costly.

Considering the scenarios mentioned, we developed a dynamic service selection process that considers the cost of service selection itself.

III. Selection cost aware dynamic web service selection

Service selection is the process of finding the best set of discovered services [13], and it can be done for a single task or for composite tasks [12]. In both cases, we must optimize the main objective of the problem at hand by selecting the service or services, which in the streaming scenario is the quality of experience and in the web service composition scenario is the quality of composite service.

To account for the dynamic performance of the web services, we must use dynamic web service selection techniques to select the best performing set of services at run time. However, other dynamic web service selection techniques may run the selection process at every invocation of the service [15]. For this reason, we need to reduce the number of times needed for service selection. To do this, we can only run the service selection process when a change in the environment is detected or when the change in the environment has not changed as much. We can see the general form of this process as a pseudocode in Fig. 1.

```
1:  $significantChange$        ▷ How much change in the environment should be considered significant?
2:  $EnvChanged$   ▷ Function to tell if the environment has changed.
3:  $env \leftarrow Null$   ▷ The environment in which the services were selected in.
4:  $services \leftarrow Null$        ▷ Selected services.
5:  procedure ServiceSelection
6:      if env is null then
7:          $env \leftarrow BootstrapEnv()$
8:          $selectedServices \leftarrow SelectBestServices(env)$
9:      end if
10:     $cenv \leftarrow GetCurrentEnvironment()$
11:     if $(EnvChanged(env, cenv))$ then
12:         if $(Change(env, cenv) > significantChange)$ then
13:             $env \leftarrow cenv$
14:             $services \leftarrow SelectBestServices(env)$
15:         end if
16:     end if
17: end procedure
```

Fig. 1. Selection cost aware dynamic web service selection

IV. Change-point detection of response time in web-services

To test the usefulness of this method, we decided to test the live-streaming scenario mentioned in section 2. To achieve this, we need to collect round-trip time from probes across the internet to act as the SFU services, look for changes in their patterns, and design an algorithm for change point detection of round-trip time.

A. Data collection

The data collection process involved gathering round-trip times (RTTs) from a number of distinct probes across various autonomous systems in 5-minute intervals using the Global Ping Network. The data was sourced from a virtual private server (VPS) situated in Finland, which was provided by Hetzner Online GmbH. The collection was executed in two distinct time frames: from September 26, 2023, to October 16, 2023, using 51 probes, which we will call the first data set from on, and from July 12, 2024, to July 18, 2024, using 129 probes, which will be called the second data set.

B. Patterns in round-trip times

It is shown that routing events can change the round-trip time in network paths [16]. So by finding these change points in the data, we can discover changes in the round-trip time.

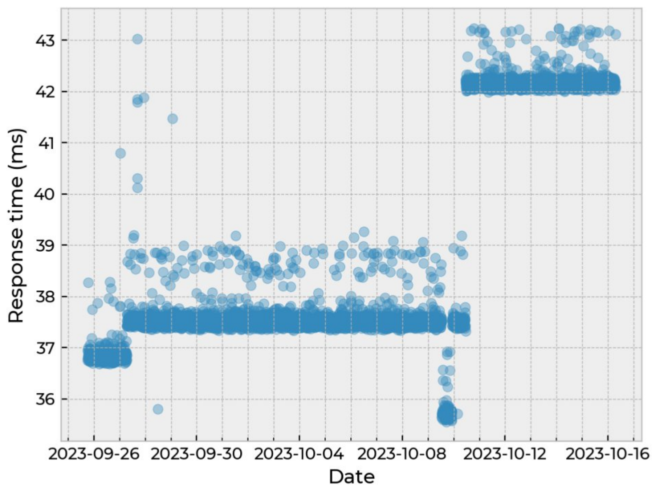

Fig. 2. Response time change in probe number 81 from first data set.

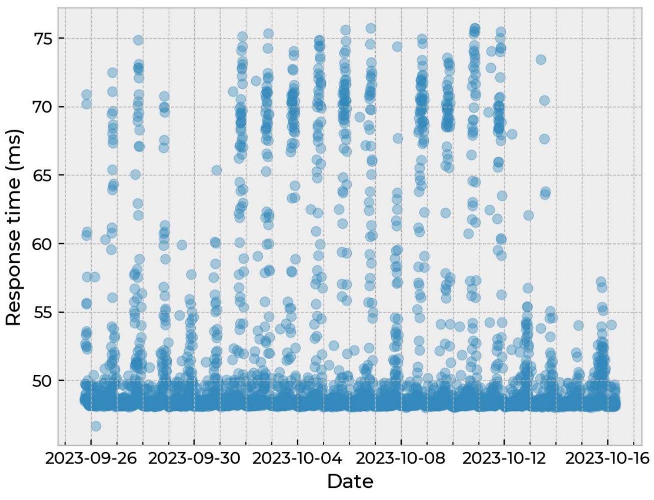

Fig. 3. Response time change in probe number 3 from first data set.

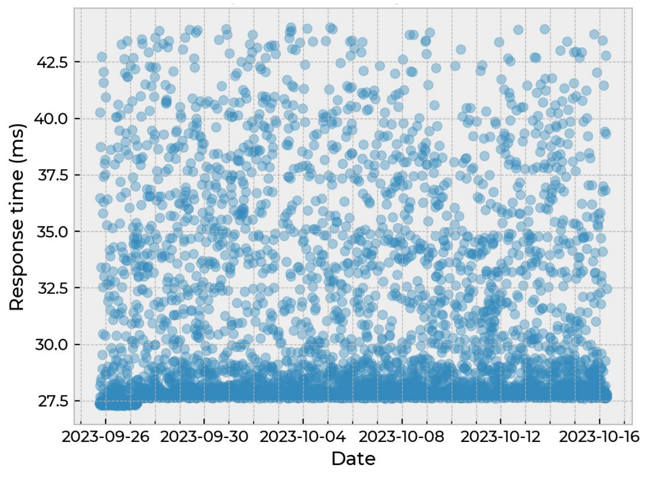

Fig. 4. Response time change in probe number 63 from first data set.

By looking at Figs. 2, 3 and 4, we can list these general observations.

- Response time distributions seems to have changed abruptly at certain change points.
- Daily seasonal patterns may exist.
- The distribution can be multimodal.
- The distribution might be positively skewed.
- Some changes in the distribution can be insignificant, like the 0.5 milliseconds change that can be seen in Fig. 2.

C. Change-point Detection

The preceding section has demonstrated that the distribution of the change point is generally stationary, and sometimes there might be sudden changes due to routing events, except in a few instances where there may be a gradual shift in mean in certain parts of the time series (three out of 51 probes from the first data set were observed to exhibit this particular pattern). It was shown that we cannot make any assumptions about the shape of the distribution, therefore we must use a non-parametric test.

We can test if two different time ranges have different distributions by following these three steps. There is the previous time range set, which are the samples from the previous distribution that services were selected for, and there is the current time range, which is the range of response times between the current time and the *setSize* previous steps. Our first step is to check if these two time ranges follow different distributions by using the Mann-Whitney U test. Then, if their distributions are different and the change in the mean of these two distributions is higher than some threshold, we conclude that a change in the distribution has occurred and the change in the distribution is significant enough. Our final step is to find exactly where change has happened. We can do this by calculating p-values, lowering *setSize* a number of times, and finding the place where it minimizes the p-value. We can see an implementation of this algorithm in Fig. 5.

```
1:  $setSize$   ▷ Size of the set to consider when performing the test.
2:  $detectableChange$   ▷ How much change in the mean of response time should be considered as significant.
3:  $rttArray$   ▷ Array of round-trip times for web service.
4:  $significanceLevel$   ▷ Significance level variable to indicate if the calculated p-value is significant
5:  procedure IsChangePoint($previousSet, currentTime$)
6:      $currentSet \leftarrow rttArray[currentTime - setSize..currentTime]$
7:      $pvalue \leftarrow mannwhitneyu(previousSet, currentSet)$
8:      if ($pvalue < significanceLevel$) then
9:          $preAverage \leftarrow average(previousset)$
10:         $curAverage \leftarrow average(currentset)$
11:         $change \leftarrow abs(preAverage - curAverage)$
12:         if ($change > detectablechange$) then
13:             return $true$
14:         end if
15:     end if
16:     return $false$
17: end procedure
```

Fig. 5. Change-point Detection for web service RTT

Fig. 6 shows the change points detected by this algorithm in one of the probes. For this demonstration, *significanceLevel* was set to be 0.001, *setSize* to 18 and *detectableChange* to 5 milliseconds. Change points that return true in this algorithm are shown by green vertical lines, and change points that return false are indicated by orange vertical lines.

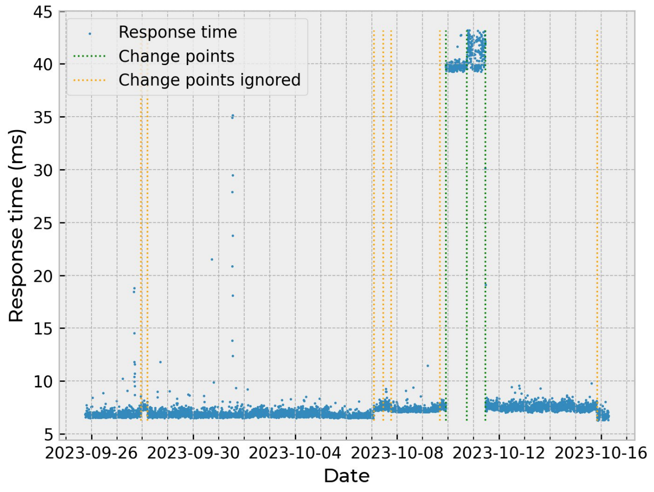

Fig. 6. Change-points in probe number 16 in first data set.

V. Experiments

We have decided to experiment the scenario of multiple SFU services for this problem as described in section 2, in this test we set a maximum acceptable response time constraint and we have to find the server with service which its response time does not exceed the mentioned criteria. First, we pick a number of probes to act as the service providers and check the response time using the RTT time series data in them. Then we have to select a service with minimum average response time. We ran this experiment in two parts. In the first part of the experiment, we check what happens if we pick probes randomly, so we pick 5 random probes from the first data set as the service and run the tests. We decided to run this part of experiment 10 times. In the second part of the experiment, we run the test by picking probes that are physically close to each other. For this part, we chose the probes located in South Jakarta. These probes are both physically close to each other and have unstable RTT. For each part, we test three cases.

The first case of the experiment is selecting a set of web services using the first 12 time steps and keeping the selected services unchanging.

The second case is done by changing the web services when a significant change in the web service is detected using the algorithm shown in Fig. 5. The parameters are set to be as follows, *setSize* is set to be 12, *detectableChange* is changing dependent on the test and *significanceLevel* is 0.001.

The third case is always changing to the best web service by knowing them beforehand to see the best possible performance.

VI. Results

The results section is divided into two parts. In the first part, we access the performance of this algorithm when web services are randomly placed, and in the second part, we see how the algorithm works when services are located close to each other. Each table showing the results has four columns. The first column shows the case of that experiment, as shown in the previous section. The selection column shows the number of times the process of finding the best services has been executed; the switching column shows the number of times services have changed and the unacceptable column is the number of requests with an unacceptable response time. The *m* variable in the first column is the *detectableChange* variable in milliseconds.

A. Randomly selected web services

For this part of the test, we set the maximum acceptable response time to be 40 milliseconds.

We ran this experiment for ten times and only in three instances this algorithm showed improvement, in all the other cases response time was either always below or above the set threshold, which means in only 30% of experiments it showed a reduction in the number of unacceptable requests. The results of three experiments with improvement is shown in the Tables 1, 2 and 3.

TABLE I. RESULTS IN EXPERIMENT PART 1, SEED 1

Experiment	Selection	Switching	Unacceptable
Case 1	1	0	603
Case 2, m=0	177	5	62
Case 2, m=0.05	77	3	63
Case 2, m=0.2	19	1	90
Case 2, m=0.5	1	1	145
Case 3	-	11	53

TABLE II. RESULTS IN EXPERIMENT PART 1, SEED 4

Experiment	Selection	Switching	Unacceptable
Case 1	1	0	190
Case 2, m=0	238	4	25
Case 2, m=0.5	24	2	11
Case 3	-	16	4

TABLE III. RESULTS IN EXPERIMENT PART 1, SEED 8

Experiment	Selection	Switching	Unacceptable
Case 1	1	0	190
Case 2, m=0	223	4	0
Case 2, m=0.5	20	2	28
Case 3	-	4	0

By looking at the Tables 1, 2, and 3, we can see that, on average, we can reduce 87% of unacceptable results using only the change point algorithm without the *detectableChange,* which by itself on average runs at only 4% percent of time steps. By using the *detectableChange,* of 0.5 milliseoncds we can reduce the 89% of unacceptable results using only 0.008% of time steps, which is an improvement in both the number of selections and reduction of unacceptable requests.

In the case of this specific application that we are testing, the finding the best services is not expensive, however in the case of more computationally expensive selection processes, it could potentially help. In general, switching does not happen as much, and it is unlikely to cause disruption, but on average, by using the half number of switches, we can reduce the 89% of unacceptable results, which shows improvement in both of our criteria.

The results in Table 1 also shows that we can lower the number of unacceptable results by increasing the *detectableChange* to some extend.

B. Physically close web services

This part of the experiment is carried out like the previous part. The only differences are that the maximum acceptable response is now 250 milliseconds and our services are located in South Jakarta. The results of this experiment are shown in Table 4.

Seeing the results, we can see that without using *detectableChange,* we can reach reduce the number of unacceptable requests by 20%, and by using *detectableChange* of one millisecond, we can reduce the number of unacceptable requests by 24%, the number of switches by 44%, and the number of selections by 76%, which is an improvement in all three of our criteria.

TABLE IV. RESULTS IN EXPERIMENT PART 2

Experiment	Selection	Switching	Unacceptable
Case 1	1	0	183
Case 2, m=0	91	18	145
Case 2, m=1	21	10	139
Case 3	-	61	70

Our key takeaway from the results of this experiment is that lowering the sensitivity to changes in the environment in some cases can lower the percentage of unacceptable results. But by lowering the sensitivity even more, the number of unacceptable results could increase again, and by considering the cost of selection and switching costs, we can find an optimal point for this variable. Comparing second and third case also shows us that our results are comparable to the best possible achievable performance in some cases.

VII. CONCLUSION

This paper introduces a new change-point based method for dynamic web service selection that considers both the QoS and the cost of web service selection, as both of them might be affecting the cost of the main problem itself. This method was evaluated in an experiment where the cost of web service selection is important. The work introduced a new method for dynamic web service selection. It also showed that the overall number of service selections, the number of switches between web services, and the number of unacceptable requests can be reduced by introducing a variable to ignore small changes in the environment.

The simulation of the web service, however, did not consider other parameters that affect the QoS, such as reliability and throughput of the web service.

For future works, we should consider throughput fluctuations across time and other important factors in calculating time-varying QoS. We can also look for other information that could help us in finding change points, like listening to messages between BGP routers.

REFERENCES

[1] S. Li, J. Wen, F. Luo, and G. Ranzi, ‘Time-Aware QoS Prediction for Cloud Service Recommendation Based on Matrix Factorization,’ IEEE Access, vol. 6, pp. 77716–77724, 2018, doi: 10.1109/ACCESS.2018.2883939.

[2] M. Zhang et al., ‘An infrastructure service recommendation system for cloud applications with real-time QoS requirement constraints’, IEEE Syst. J., vol. 11, no. 4, pp. 2960–2970, Dec. 2017.

[3] X. Sun, S. Wang, Y. Xia, and W. Zheng, ‘Predictive-Trend-Aware Composition of Web Services With Time-Varying Quality-of-Service,’ IEEE Access, vol. 8, pp. 1910–1921, 2020, doi: 10.1109/ACCESS.2019.2962703.

[4] A. Strunk, “QoS-Aware Service Composition: A Survey,’ in 2010 Eighth IEEE European Conference on Web Services, Ayia Napa, Cyprus: IEEE, Dec. 2010, pp. 67–74. doi: 10.1109/ECOWS.2010.16.

[5] ‘Low-latency interactive multimedia stream requirements’ Recommendation F.746.1, Telecommunication Standardization Sector of ITU.

[6] A. Bentaleb, M. Lim, M. N. Akcay, A. C. Begen, S. Hammoudi, and R. Zimmermann, Toward One-Second Latency: Evolution of Live Media Streaming. 2023. [Online]. Available: https://arxiv.org/abs/2310.03256

[7] J. Allard, A. Roskuski, and M. Claypool, ‘Measuring and modeling the impact of buffering and interrupts on streaming video quality of experience’, in Proceedings of the 18th International Conference on Advances in Mobile Computing & Multimedia, Chiang Mai Thailand, 2020.

[8] J. He, M. Ammar, E. Zegura, E. Halepovic, and T. K. Ules, ‘QoE metrics for interactivity in video conferencing applications’, in Proceedings of the ACM Multimedia Systems Conference 2024 on ZZZ, Bari Italy, 2024.

[9] A. Eleftheriadis, R. M. Civanlar, and O. Shapiro, ‘Multipoint video-con-ferencing with scalable video coding,’ Journal of Shejiang University SCIENCE A, vol. 7, pp. 696–705, 2006.

[10] jitsi, ‘Jitsi Videobridge’ *GitHub*, Aug. 2024. https://github.com/jitsi/jitsi-videobridge.

[11] Odoo, ‘Odoo SFU’ *GitHub*, Aug. 2024. https://github.com/odoo/sfu.

[12] N. Balaji, G. Sambasivam, S. R. Murugaiyan, S. Basha, T. Vengattaraman, and P. Dhavachelvan, ‘Appraisal and Analysis on Diversified Web Service Selection Techniques based on QoS Factors,’ *Uncst.go.ug*, 2024.

[13] W. Serrai, A. Abdelli, L. Mokdad, and Y. Hammal, ‘An efficient approach for Web service selection’, in 2016 IEEE Symposium on Computers and Communication (ISCC), Messina, Italy, 2016.

[14] G. Kang, J. Liu, M. Tang, and Y. Xu, ‘An effective dynamic web service selection strategy with global optimal QoS based on particle swarm optimization algorithm’, in 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, Shanghai, China, 2012.

[15] S.-Y. Hwang, E.-P. Lim, C.-H. Lee, and C.-H. Chen, ‘Dynamic web service selection for reliable web service composition’, IEEE Trans. Serv. Comput., vol. 1, no. 2, pp. 104–116, Apr. 2008.

[16] H. Pucha, Y. Zhang, Z. M. Mao, and Y. C. Hu, ‘Understanding network delay changes caused by routing events’, *Perform. Eval. Rev.*, vol. 35, no. 1, pp. 73–84, Jun. 2007.