AN ADAPTIVE SERVICE REPLICATION FRAMEWORK FOR MANAGING DIFFERENT RESPONSIVENESS LEVELS

M. F. Mohamed, H. F. Elyamany and H. Nassar

Department of Computer Sciences, Faculty of Computers & Informatics, Suez Canal University, Ismailia, Egypt.

marwa_fikry@ci.suez.edu.eg - hany_elyamany@ci.suez.edu.eg and nassar@ci.suez.edu.eg

Abstract: Service replication becomes the area of interest for many providers as it can be utilized to improve both performance and availability of their own services. With the increasing number of consumers, the providers are obligated to maintain the service responsiveness (i.e. Performance and availability) with respect to the signed Service Level Agreement (SLA) between the provider and consumers. In this paper, our previous adaptive replication framework is modified to provide different responsiveness levels to available consumers. The suggested framework is calculating a specific load threshold (i.e. The time needed to initiate the adaptive service replication process) according to the response time registered within the SLA contract through applying the linear regression model. Specifically, when the current load of the running service gets over the calculated threshold, the adaptive replication process will be initiated automatically to preserve the responsiveness level that the consumer demands. The paper basically describes two different scenarios to show how the suggested framework operates for individual consumers and organizations. Finally, the experimental results demonstrate that the proposed framework prohibits the violation of response time parameter in the SLA contract.

Keywords: Load balancing, Replication, Responsiveness, SLA and Web services

1. Introduction

Many organizations often aim to apply and perform online IT solutions to improve their business. With the rapid growth of the global economy and the need to reform big consortiums to provide flexible, well-established and low-rate services for a huge diverse number of clients, the organizations always seek to evolve their IT solutions to being self-adaptive disciplines. These disciplines basically embrace a set of particular software for continuous checking the status of the deployed services and providing the demanded business as expected with respect to the different requirements of the client or consumer side [1].

Nowadays, most of the organizations usually sign an electronic agreement, often called a Service Level Agreement (SLA), with the consumers in order to determine and manage the rights and obligations for each side (e.g. Availability, performance and cost). Basically, SLA is a legal contract established between the service provider and consumers in which some specific Quality of Services (QoS) parameters including performance and availability are defined, managed and monitored [2]. The providers are always being demanded to introduce various SLA templates with different QoS values in order to receive a broad-spectrum of consumers. Responsiveness is a specific scale for measuring the satisfaction degree of consumers when accessing the provided services in terms of evaluating the
predefined QoS parameters inside the signed SLA. Specifically, responsiveness is the responsible factor for determining the success of a service-oriented environment through calculating a unique value, probably the response time, equivalent to how far the consumers are fulfilled with the published services [3-4].

Service-Oriented Architecture (SOA) is an architectural style to decompose the software into a set of reuse, scalable, interoperable and business-encapsulated components; typically called web services [5]. A Web service (WS) or shortly service becomes the main element of any online and integrated business. It is usually identified by a unique URI, uses XML for defining and describing a public interface and can be easily bound with other software systems via exchanging XML based messages, named SOAP, conveyed with Internet protocols [6]. Services are sometimes experienced by heavy incoming requests that possibly let it falls in availability and performance challenges and in turn making the consumers awkward with accessing these services that might probably lead to the business loss. Moreover, the providers should monitor their own services in order to maintain the signed SLA and discard any violation events that might be occurred at the services or servers side. Furthermore, the providers should preserve their ranked responsiveness as expected to keep their business reputation within an acceptable level in order to attract many more customers. One of the popular solutions to overcome the previous challenges is to increase the number of the provided services that encapsulate the same business with offering different QoS packages through utilizing the replication technology.

Service replication technology can basically enhance services availability through providing multiple replicas of it; for example, if one service fails, other copied services will be possibly initiated in order to respond the consumer requests [7-8]. Moreover, it can reduce the response time through broadcasting incoming requests among the available replicas, particularly during runtime. However, in order to completely guarantee the SLA requirements and prevent any possible violations, replication should be managed and performed automatically with a low human supervision. In other words, replication should be implemented and controlled within an adaptive mode. Specifically, an adaptive replication refers to dynamically transferring services files to other servers when services or servers fail or become overloading [9].

In our previous work [10], we proposed an adaptive replication framework for improving the QoS of services. However, the main disadvantage of this framework is to provide the same responsiveness level for all consumers regardless their diverse QoS requirements. In this paper, the suggested framework in [10] is developed to provide different levels of responsiveness with respect to the defined QoS parameters inside the signed SLA. Particularly, the suggested framework dynamically exploits the incoming requests (i.e. Service load) to automatically provide the number of needed replicas for compromising the response time listed in the SLA contract for each individual consumer or organization. The rest of the paper is organized as follows: Section 2 shows the related work. Section 3 introduces a description of the suggested adaptive replication framework for various responsiveness levels. Section 4 explains the workflow of the proposed framework. Section 5 describes the implementation and performance of the proposed framework. Finally, conclusion and future work are presented in section 6.

2. Related Work

Often, the adaptive replication process is developed through applying two main phases: the first phase relates to an adaptive server selection of available WS hosts as suggested in [3, 12-14]. The second
phase is basically based on automatically transferring a replica to a selected host during runtime as proposed in [9-10, 15].

In the adaptive server selection process, the server or host is selected from a predefined list of available hosts according to a combination of several metrics including current network status, server workload and response time [11]. Specifically, the suggested framework [12] used an adaptive selection of idle servers in order to overcome the main server problems such as failure or overloading. The framework [12] proposed two main components as Longest Delay Service Component Selection (LDCS) for services failure monitoring and Maximum Available Capacity Path (MACP) for investigating the involved servers. When LDCS detects that a particular service is bottlenecked or overloaded, the framework selects another server for hosting a new replica from some specific available servers though utilizing MACP. The suggested framework [13] provided an adaptive re-selection of a higher priority service when the main service or host fails. The authors organized the service priority according to some QoS constraints including the network availability as well as the host machine. The main drawback of this framework [13], it does not consider other crucial metrics within the replica re-selection process such as the service or host load. The adaptive selection or auto-selection process is not necessary to be accomplished by the providers only as in [12-13]; however, it could be established by the consumers as well to choose the best fit service for their requirements [16-17]. Notably, the responsiveness attribute (i.e. Availability and performance) was the most dominant factor in the service or server selection process in all mentioned work [12-13][16-17].

On the other hand, other works have applied the second phase ‘adaptive transferring WS files’ in which the failed or overloaded WSs are automatically replicated to other hosts during runtime such as the presented research in [9-10, 14]. The proposed framework in [9] is aimed to improve the performance of the published Web services. Basically, it studied a dynamic Web service selection and replication processes at runtime. When a service is running on a heavily loaded service host and there are service hosts available having a low workload, the framework decides to generate a new service on low workload hosts using a feature called automatic service replication. However, the framework [9] did not discuss the hosts’ selection strategies. Our proposed framework in [10] has the same target of [9]; it presents an adaptive framework for managing dynamic replication of Web services in a distributed environment including the Service-Oriented Architecture (SOA) environment. The framework aims to improve the web services availability and performance by supporting an automatic replication of the consumed web services according to environment changing conditions that might occur at the service provider side such as failure or increasing load. The difference between [9] and [10], the framework in [10] used the load prediction of the involved candidate servers within the replication process through utilizing a statistical regression technique. The presented work in [15] is similar to our original work in [10]; however, instead of using linear aggression for predicting the load, the authors deployed a generalized time series technique without providing any details or complete discussions about that utilized algorithm and how it has improved the proposed original work in [10].

Both previous phases can be used by the service providers to achieve various responsiveness levels. The authors in [3] allowed the service provider to offer different responsiveness levels of WS by using an adaptive replica selection. The re-selection process is based on calculating a set of hybrid metrics including service failure, host failure and dynamic load as well as the customer business-value level. However, the proposed framework in [3] could not determine the initial number of hosts. The proposed framework in [14] is designed for dynamic invocation of geographically replicated web services. The framework used different server selection policies which are incorporated into the client applications in
order to reduce service response time at the client side. However this introduces extra overhead of executing the web service replica selection at the client side.

3. The Adaptive Replication Framework For Various Responsiveness Levels

In our previous work [10], we proposed a framework for managing an adaptive replication of Basic Web services in a Service-Oriented Architecture (SOA) environment. It provides an automatic replication of different services which run independently according to the change that might occur at the provider side. Basically, the suggested framework in [10] issues the necessary decisions when a service fails or being overloaded, and consequently selects the suitable servers in which the new replica will be hosted automatically. However, this framework has fallen short in serving all incoming requests from consumers equally without considering the priority of the consumers and the QoS requirements they are interested with. In this paper, the suggested framework as shown in Figure 1 is developed to provide different levels of responsiveness according to the SLA items signed between the provider and consumers. Particularly, the suggested framework dynamically checks the incoming load and automatically calculates a specific load threshold according to the response time registered within the SLA contract through applying the linear regression model [18].

3.1 Framework Architecture

As depicted in Figure 1, the suggested framework is divided into four layers which are listed as follow:

- **The consumers’ layer** represents the customers who are seeking to access the services.
- **The provider layer** identifies the business owners who are publishing the services and has multiple secondary servers for hosting services and replicas.
- **The SLA layer** manages and maintains the signed contracts between the provider and consumers.
- **The Replication Management Middleware layer** is located between the provider and consumers' layers and manages the intercommunication among them with respect to the listed SLA preferences.

![Diagram](image1.png)

Figure 1: The adaptive replication framework for various responsiveness levels.
The main components of the Replication management middleware are basically divided into three categories: sensors, controllers and actors as demonstrated in Figure 1. Each category is recognized as follows:

- The *Sensors* monitor all possible changes that might be occurred at the secondary servers including failure, load and performance-lack.
- The *controllers* manage and control the Replication management middleware components such as the selector and SLA monitor.
- The *actors* are utilized by the controller to apply modifications within the Secondary servers including the replicator, dispatcher and cleaner.

All previous replication management middleware components are explained in [10], except the SLA monitor. The SLA monitor component is responsible for observing the response time of the provided web services while a particular consumer accesses it. For simplicity, The SLA monitor uses the linear regression model [18] to estimate the corresponding load threshold with respect to the measured response time. Remarkably, the server load is not proportional to the response time; in other words, when the server load increases, the response time often decreases.

Two main steps are applied to evaluate the load threshold of the secondary servers using the linear regression model [18]. In the first step, The SLA monitor detects the response time of all available secondary servers as shown in Figure 2 in which different loads are run on the WS hosts and then the corresponding response times are recorded. The second step is calculating the value of \( y \) “the load threshold” depending on \( x \) “the WS response time” by calculating Equations 1 to 3 [19]. Notably, the estimation of the value \( y \) is necessary to determine whether the replication process is required to be accomplished or not.

\[
\beta_i = \frac{\sum_{i} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i} (x_i - \bar{x})^2} \\
\beta_0 = \bar{y} + \beta_1 \bar{x} \\
y = \beta_0 + \beta_1 x + \varepsilon
\]

(1)  
(2)  
(3)

Where \( \bar{x} \) and \( \bar{y} \) denote average.
3.2 Framework Functionality

Figure 3 represents the main processes occurring between the actors or layers: Consumers and Provider with his/her embedded Secondary servers. Each process is demonstrated as follows:

- **SLA Establishment**: Operating between the service provider and consumers. It's responsible for determining the basic parameters of the SLA (e.g. Response time).

- **SLA Monitoring**: It is an automatic internal operation. It's responsible for adapting the replication middleware to prevent SLA violation and support various levels of responsiveness.

- **WS Replication**: This process is accomplished to maintain the SLA items as expected. For example, it transfers a new replica to selected host to preserve the service performance as stated within the signed SLA.

- **Server Selection**: It is responsible for selecting a new host for holding a new replica. Of course, it considers the secondary server capabilities when establishing the selection decision.

- **WS Deletion**: Unlike the WS Replication process, this operation eliminates the unneeded replicas when the SLA items are achieved ordinarily, without replication.

4. The Workflow of the Adaptive Replication Framework

This section describes two regular scenarios; the first one explains how the adaptive replication framework reacts to individual customers. While, the second scenario shows how the framework responds to the different organizations in which multiple consumers work in it.
4.1. The First Scenario

This scenario assumes that the service provider contracts with many consumers, each one has a unique responsiveness level. So the SLA monitor deals with each individual consumer alone as described in the following steps.

- At the beginning, the SLA is established and signed between the service provider and consumers.
- For each consumer, The SLA monitor estimates the corresponding load threshold according to the response time value registered in the SLA contract through applying the linear regression model [18].
- Every specific period, assumed to be 20 second, the SLA monitor checks the load of running WS and consumed by each individual consumer.
- Two cases may be occurred; at the first one, the current load of the running WS host gets over the load threshold value, then the following steps will be accomplished:
  - The SLA monitor calls the selector, to select the suitable server.
  - Then, it initiates the replicator to send a new copy to the selected server,
  - Meanwhile, it updates the IP of the new replica to new the selected host IP on the Replica_DB.
- In the second case, the load of the running WS hosts doesn't exceed the load threshold, so no actions are taken by the SLA Monitor.

The following algorithm demonstrates the way in which the SLA Monitor works as shown in Figure 4. Also, the sequence diagram in Figure 5 summarizes the main SLA Monitor functionalities.

```java
// Prototype of Get_LT function, given the response time evaluates the load threshold by using linear regression
Float Get_LT (CustomerID, responseTime);

// Prototype of avgLoad given the server IP can get the average load of that server by using load sensor
Float avgLoad (Server_IP);

SLA_Monitor (CustomerID, responseTime, Server_IP)
Begin
  Load_Threshold = Get_LT (CustomerID, responseTime);
  If ( load_Threshold > avgLoad (Server_IP) )
    Begin
      // Select a suitable server where its current load doesn't exceed Load_Threshold
      New_IP = selector (Load_Threshold);
      Replicator (New_IP);  // transfer new replica to New_IP server
      // Update the Serve_IP to New_IP on Replica_DB;
      Serve_IP = New_IP;
    End
End
```

Figure 4: The SLA monitor algorithm for individual consumers.
In this scenario, the two components: the dispatcher and SLA monitor are working together in parallel. The main task of SLA monitor is to determine where the customer requests are executed. But the dispatcher is responsible for forwarding the customer request to a specific replica. When the customer sends a request to WS, the dispatcher uses the customer ID to read the position of requested WS from the Replica DB. Then it forwards the request to the determined WS host ‘secondary server’. After the secondary server processes the customer requests, the dispatcher sends the response back to the consumer.

4.2 The Second Scenario

In this scenario, the service provider groups all consumers which have the same responsiveness level into one unit. For example, when the service provider contracts with an organization which has several employees (i.e. Consumers) work in it; in that case, all employees will have the same responsiveness that the organization is expected to have. The SLA Monitor interacts with all consumers having the same responsiveness as demonstrated in the following steps.

- At the beginning, the SLA is confirmed between the service provider and service organization.
- For each organization, The SLA monitor estimates the corresponding load threshold according to the response time value listed in the SLA contract through applying the linear regression model [18].
- Every specific period, assumed to be 20 second, the SLA monitor checks the load of Master replica running which consumed by each organization.
- Two cases may be occurred; at the first one, the current load of the Master replica running gets over the load threshold value, then the following steps will be accomplished:
  - The SLA monitor calls the selector, to select the suitable server.
Then, it initiates the replicator to send a new copy to the selected server, Meanwhile, it increments the number of used replicas by one and updates the secondary replica IP to new the selected host IP on the Replica_DB.

- In the second case, the load of the running WS hosts does not exceed the load threshold, and then the load balancing is not required in this case. Hence, the SLA monitor updates the number of replicas to one when organization requests are already balanced. And also it invokes cleaner to remove the unused replicas.

The following algorithm demonstrates the way in which the SLA Monitor works as shown in Figure 6. Also, the sequence diagram in Figure 7 summarizes the main SLA Monitor functionalities.

```
// Prototype of Get_LT function, given the response time evaluates the load threshold by using linear regression
float Get_LT (Org_ID, responseTime);
// Prototype of avgLoad given the server IP can get the average load of that server by using load sensor
float avgLoad (Server_IP);

SLA_Monitor (Org_ID, responseTime, Server_IP[], Num_Repllicas)
Begin
  Load_Threshold = Get_LT (Org_ID, responseTime);
  If (Load_Threshold > avgLoad (Server_IP[1])
      Begin
        Select the best suitable server
        New_IP = Selector();
        Replicator (New_IP);
        // transfer new replica to New_IP server
        Update the number of replicas by 1 on Replica_DB to balancing between those replicas
        Num_Repllicas ++;
        Update the IP of secondary replica to New_IP on Replica_DB;
        Serve_IP[Num_Repllicas] = New_IP;
      End
    Else
      If (Num_Repllicas > 1)
        Begin
          Call Cleaner to delete unused replicas;
          // Update the number of replica used to one replica on Replica_DB;
          Num_Repllicas = 1;
        End
      End
End
```

Figure 6: The SLA monitor algorithm for organizations.

Also in this scenario, the two components: dispatcher and the SLA monitor are working together in parallel. When the customers “related to an organization” send requests to WS, the dispatcher uses the organization ID to read the number of deployed replicas and WSs IPs from the Replica_DB. If the number of used replicas equals to one, the dispatcher forwards the request to a single secondary server. On the other hand, if the number of used replicas is more than one, the dispatcher balances the requests between these replicas using the round robin algorithm [19]. After the secondary server(s) process the customers requests, the dispatcher sends the response back to the consumers. The following algorithm demonstrates the way in which the dispatcher works as shown in Figure 8. Also, the sequence diagram in Figure 9 summarizes the main dispatcher functionalities.
Figure 7: The sequence diagram of the SLA monitor for organizations

Figure 8: The dispatcher algorithm for organizations

Figure 9: The sequence diagram of the dispatcher scenario.
5. Implementation and Performance

The following experiments have been conducted to explain how the SLA Monitor provides several levels of responsiveness. The deployed testing environment is a VMware [20] simulation consists of 6 servers: 1 MS, 5 SSs and 1 DNS. The capabilities of these servers are established in Table 1.

<table>
<thead>
<tr>
<th>Capabilities</th>
<th>MS</th>
<th>SSs</th>
<th>DNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD</td>
<td>40 GB</td>
<td>40 GB</td>
<td>40 GB</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel® core™ Num of core using by VMware 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>500 MB</td>
<td>384 MB</td>
<td>384 MB</td>
</tr>
<tr>
<td>Language</td>
<td>Microsoft windows server 2003</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following steps explain how exactly the experiment is accomplished:

1. The SLA Monitor simulates different loads on the WS hosts using ApacheBench Version 2.0.40-dev <$Revision: 1.146 $>Apache-2.0 [21] and then it records the corresponding response times, see Table 2.

<table>
<thead>
<tr>
<th>RT Load</th>
<th>0.10477</th>
<th>0.10445</th>
<th>0.09883</th>
<th>0.09469</th>
<th>0.09148</th>
<th>0.09056</th>
<th>0.08744</th>
<th>0.08273</th>
<th>0.0822</th>
<th>0.07872</th>
</tr>
</thead>
<tbody>
<tr>
<td>95</td>
<td>80</td>
<td>75</td>
<td>70</td>
<td>75</td>
<td>65</td>
<td>60</td>
<td>55</td>
<td>50</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Then, it calculates the needed values $\beta_1$, $\beta_0$, and $y$ as explained in section 3.1, see Table 3.

$$
\beta_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} = 1640.95, \quad \beta_0 = \bar{y} + \beta_1 \bar{x} = -77.79
$$

3. In this step, the two scenarios explained in Section 4, behave differently as demonstrated next:

In the individual consumer scenario, for each consumer, the SLA monitor estimates the corresponding load threshold according to the response time values registered in the SLA contract. Table 4 provides examples of the response time values named $x$ and the corresponding load threshold $y$. 

155
Table 4: The response time $x$ and the corresponding load threshold $y$.

<table>
<thead>
<tr>
<th>Response time $x$</th>
<th>Load threshold $y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0718 μs</td>
<td>40 %</td>
</tr>
<tr>
<td>0.084 μs</td>
<td>60 %</td>
</tr>
<tr>
<td>0.0962 μs</td>
<td>80 %</td>
</tr>
<tr>
<td>0.1084 μs</td>
<td>100 %</td>
</tr>
</tbody>
</table>

In the initial state, all consumers’ requests are forwarded to the target WS host, as shown in Figure 10. Notably, each host publishes a specific WS that accessed by different consumers. The SLA monitor is continuously observing the different hosts and calculating the response time for each host and matching it with the registered value inside the SLA contract for each consumer. Basically, if the current load of the running WS host gets over the estimated load threshold value for a consumer, then the SLA monitor initiates the replicator to send a new copy to the best selected server as explained in [10]. In Figure 11, the current load of WS host identified by the IP ‘192.168.2.3’ gets over the load threshold ‘40%’ for a particular consumer. Therefore, the replicator transfers a new copy to the new host identified by the IP ‘192.168.2.6’. Meanwhile, the SLA monitor updates the WS host’s IP to ‘192.168.2.6’ within the Replica_DB.

For the organization scenario, the service provider and service consumer are initially determined the basic parameter of the SLA (i.e. Response time $x = 0.09316$ μs). Then the SLA monitor uses the response time 0.09316 μs to estimate the corresponding load threshold $y = 75\%$. The
following experiment shows that the adaptive replication framework succeeded to get the correct load threshold “75%” in order to compliances with the SLA parameter (i.e. Response time). The experiment is accomplished by passing (1000) requests with a different concurrent number representing the number of employees in an organization (e.g. 6, 9 and 12) and then recording the corresponding response time values. Figure 12 depicts that the average response time resulting from this experiment don’t exceed the response time value registered within the SLA contract.

6. Conclusions

In this paper, an adaptive replication framework for different responsiveness levels is introduced. The framework includes several components for managing the replication process in addition to a particular component named ‘SLA Monitor’ which is observing the response time of the provided web services while a particular consumer accesses a service. Moreover, it applies the linear regression model to calculate the corresponding the load threshold with respect to the response time registered within the SLA contract. Furthermore, two different scenarios are demonstrated as well as the experimental results are provided. The experiment examined the suggested framework within two separate scenarios: consumers and organizations in which the response time is measured. The shown outcomes established that the framework maintains the response time parameter as stated in the SLA contract. For the future work, we plan to apply the adaptive replication framework for different responsiveness levels on the composite services. In such scenario, two different web services are integrated together to implement a particular business process, one of them might be failed or both might be overloaded, therefore, the framework should be adopted to handle such these situations. Finally, we plan to apply the adaptive replication with different responsiveness on other distributed systems such as cloud computing environments.

References