A Fuzzy Scheduling Mechanism for a Self-Adaptive Web Services Architecture

Anderson Francisco Talon[1,2] and Edmundo Roberto Mauro Madeira[1]
[1]*Institute of Computing, University of Campinas (UNICAMP), Av. Albert Einstein 1251, Campinas, SP, Brazil*
[2]*Information Systems, Faculty FGP (FGP), Rua Prof. Massud José Nachef 2855, Pederneiras, SP, Brazil*

Keywords: Fuzzy Monitoring, Self-Adaptive Monitoring, pro-Active Monitoring, Web-Service Monitoring, e-Contract Violation.

Abstract: The rise of web services have become increasingly more visible. Monitoring these services ensures Quality of Service and it is the basis for verifying and potentially predicting e-contract violations. This paper proposes a fuzzy scheduling mechanism that attempts to predict a possible e-contract violation based on historical data of the provider's services. Consequently, there is a self-configuration on the architecture that changes service priority, making the provider processes the high priority services before low priority services. This prediction can also helps the self-optimization of the architecture. A decrease of e-contract violations can be observed. Though it is not always possible to predict a failure, the architecture is capable of self-healing by using recovery actions. Comparing the fuzzy scheduling with others known in the literature, an improvement of 31.52% in the e-contracts accomplishment is observed, and a decrease of 35.59% in average response time was achieved. Furthermore, by using the fuzzy scheduling, the overload of the provider was better balanced, varying at most 8.43%, while the variation in other scheduling mechanisms reached 41.15%. The results show that the fuzzy scheduling mechanism is promising.

1 INTRODUCTION

Service-Oriented Computing (SOC) can help the integration of heterogeneous platforms and the construction of complex applications by combining simple services. However, these integrations and compositions can create some functional problems, such as: (i) services can change in the provider; (ii) services can stop working in the provider; and/or, (iii) services can run completely different than expected, due to a functional programming error.

Furthermore, maintaining non-functional properties, such as response time, availability, reliability, and security can become a difficult problem to solve.

Because of those problems, it is essential to monitor the web service compositions. For the consumers, it is important to know if the provider is respecting the established electronic contract (e-contract). For the providers, it is important to know if they are satisfying functional and non-functional features as required by their consumers.

The main contribution of this paper is to propose a fuzzy scheduling mechanism to predict if the provider may cause an e-contract violation. Based on this early prediction, some actions can be taken, such as: (i) the consumer can select another provider which would be able to attend its needs; (ii) the provider can increase its processing capability to be able to accomplish all e-contracts; and/or, (iii) both parts can renegotiate the e-contract changing QoS (Quality of Service) values.

This research is based on a previous architecture (Fantinato *et al.*, 2010). The researchers presented a monitor that examines service executions to verify if QoS levels are satisfied. The main difference between our proposed architecture is the monitor, which predicts e-contract violations before they actually happen. Three modules were added to the architecture: *analyzer*, *optimizer*, and *recovery*.

The *analyzer* module uses a fuzzy system to predict e-contracts violations. All analyses to predict e-contract violation are made in parallel with the service execution. The *optimizer* module changes the service priority based on the analysis results. A priority queue is adapted according to the analysis results. If an e-contract violation happened, the

Talon, A. and Madeira, E.
A Fuzzy Scheduling Mechanism for a Self-Adaptive Web Services Architecture.
DOI: 10.5220/0006321705290536
In *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017) - Volume 1*, pages 529-536
ISBN: 978-989-758-247-9

recovery module will try to fix it. Only the *analyzer* module uses an artificial intelligence technique.

The proposed mechanism improves e-contract accomplishment. A fuzzy system was used to change the service priority, processing higher-level services first. With this priority queue, the service average response time was decreased and the service availability was increased.

When analysing the results, an improvement can be observed in the system's performance. Comparing the fuzzy scheduling mechanism with the queue, random, shorter duedate, shorter response time, and shorter processing time scheduling mechanisms, there was an increase in the e-contract accomplishment and a decrease in the average response time.

2 FUNDAMENTALS

The use of web service technology can decrease the implementation time of new services, because of the reusability and integration of the system in distinct platforms.

A contract is an agreement between two or more parties to establish mutual relationships in business or legal obligations. The e-contracts are used to describe agreements between organizations of electronic business on the internet. They may include QoS attributes agreed between the parties involved.

The service monitoring has the task of following process/service execution and taking actions when certain requirements of QoS are not being met (Papazoglou *et al.*, 2008).

Autonomic Computing is an approach to design self-managing computing systems that have a minimal human interference. The self-managing system can be divided into four properties: self-configuration, self-optimization, self-healing, and self-protection. Self-Adaptive Systems have contained some elements of these properties for some time. The terms autonomic computing, self-managing systems, and self-adaptive systems can be used synonymously (Huebscher and McCann, 2008).

A fuzzy system has mechanisms based on fuzzy sets and/or fuzzy logic for the treatment of imprecision. This imprecision can be expressed by variables whose values are represented by fuzzy sets. These codification of these variables allows the generalization of information associated with imprecision (Yager and Filev, 1994).

3 PROPOSED ARCHITECTURE AND MECHANISM

The proposed architecture is based on the work of (Fantinato *et al.*, 2010) for business process execution. A business process is a composition of web services. In their work, they presented a monitor that follows up services execution to verify if the QoS levels are met. The main difference from the original architecture and the one presented in this paper is the addition of the analyzer, optimizer, and recovery modules, developed for monitoring purposes, which have the ability to predict e-contract violations before they happen.

The proposed architecture and fuzzy scheduling mechanism were presented in (Talon and Madeira, 2015b). In the current paper, comparisons between the fuzzy scheduling mechanism and other traditional scheduling mechanisms are shown. The other scheduling mechanisms are: (i) **Queue**: first incoming request will be the first to be answered (FIFO – first in, first out); (ii) **Random**: a random number is assigned to each request and the provider answers the requests according to the order; (iii) **Shorter Duedate**: the request that has the closest time to a violation will be answered first by the provider; (iv) **Shorter Response Time**: the provider answers the services with the shortest response time first; and, (v) **Shorter Processing Time**: the provider answers the services with the shortest processing time first.

The architecture is composed of four entities (*provider*, *consumer*, *monitor*, and *negotiator*) and involves five phases for business process execution (**negotiation**, **monitoring**, **optimization**, **recovery**, and **renegotiation**). Each entity is composed of repository(ies) and module(s). The *Service-Oriented Computing* (*SOC*) is responsible for the communication between services that are stored on a *Service Repository* (*SR*).

The *negotiator* entity is responsible for the phases of (**re)negotiation**. It is composed of a *negotiator* module and an *e-Contract Repository* (*ECR*). The **negotiation** phase is responsible for the creation of the e-contract and its QoS parameters. The **renegotiation** phase is responsible for the modification of the e-contract and its QoS parameters. Both phases, **(re)negotiation**, will not be treated in this paper. For the purpose of this paper, it is assumed that e-contracts exist and all entities have them. The e-contracts are stored on the *ECR*.

The proposed architecture with its entities and modules can be observed in Figure 1.

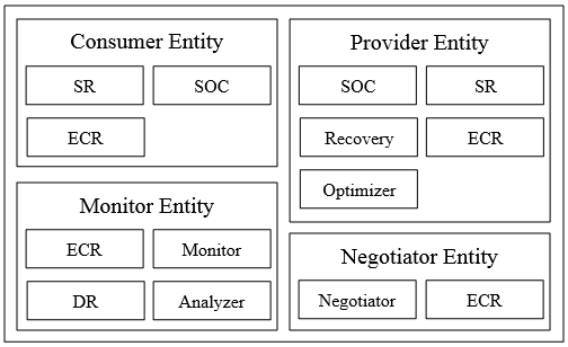

Figure 1: Proposed Architecture.

The *monitor* in the proposed architecture can be classified as soft-intrusive and asynchronous. It is soft-intrusive because a little change in *consumer* and *provider* are necessary. Both of them need to add a new message to the *monitor* into their code. It is asynchronous because all analyses to predict e-contract violation are made in parallel with the service execution. The architecture monitors two features: response time and availability. In this paper, the monitor is considered light-weight (where the monitor handles only one web service feature), but the architecture also supports heavy-weight monitoring (where only one monitor handles different features of the web service). The two monitors work together, where the first one controls the response time and the second one the availability, both for the same e-contract QoS.

The *monitor* module intercepts all messages between the *provider* and the *consumer*. These messages, and their response time and process time, are stored into the *Data Repository* (*DR*).

The *analyzer* module uses the data from the *DR* to estimate the possibility of an e-contract violation, which is determined by a fuzzy system. The fuzzy system is used because it is a technique that treats imprecision, and in this case, it is not possible to guarantee when there will be an e-contract violation.

The *optimizer* module uses information regarding the analysis in order to change the service priority. The main idea about the priority queue is: if a service is more probable to violate the e-contract, it should have a higher priority over other services. The service should run before by the *provider*.

The *analyzer* and *optimizer* modules have the task of trying to accomplish e-contracts. However, a contract break can still happen. Faced a contract break, the *recovery* module tries to fix it by increasing the service priority. The priorities of the *recovery* module are greater than the priorities of the *optimizer* module. In other words, if there was a contract break, this service would run before others.

3.1 Priority Queue

The priority queue was set with ten priority levels. Services can receive a value ranging between 0 and 9. The provider should run services with higher priority before services with lower priority.

The priority levels are divided into four groups. The first group represents no possibility of violation. The second one represents the possibility of violation, where these levels are determined by a fuzzy system. The architecture uses 5 fuzzy levels because of the combination between the linguistic terms of the fuzzy sets. The third group corresponds to an e-contract violation, according to the response time. Lastly, the fourth group represents an e-contract violation, according to the availability. The availability feature has higher priority than the response time feature because it will not respond at the agreed time if the service is unavailable.

More details are given as it follows:

- *Level 0 / Group 1*: All services start at this level. Their execution will not result in contract violation.
- *Level 1, 2, 3, 4, and 5 / Group 2*: Execution of services at this level may result in contract violation. The analyzer module determines the possibility of violation from the historical data and sends this information to the optimizer module. Level 1 represents very-low possibility of violation; Level 2, low possibility; Level 3, medium possibility; Level 4, high possibility; and, Level 5, very-high possibility of violation.
- *Level 6 / Group 3*: Execution of services at this level violated the required response time on the consumer side only. In this case, the provider processes the request in time, but the response does not arrive at the consumer in time. The monitor detects it and informs the recovery module.
- *Level 7 / Group 3*: Execution of services at this level violated the required response time on the provider side (and on the consumer side as well). In this case, the provider is not processing the request in time. The monitor detects it and informs the recovery module.
- *Level 8 / Group 4*: Execution of services at this level represents a small availability violation, which means violations up to a set acceptable rate of violation. For example, if the availability is set to 95%, the rate of violation is 5%. Therefore, a small availability violation is greater than or equal to 90%, and smaller than 95%.

- *Level 9 / Group 4*: Execution of services at this level represents a big availability violation, which means violations are greater than an acceptable rate of violation. For example, if the availability is set to 95%, the rate of violation is 5%. Therefore, a big availability violation is smaller than 90%.

3.2 Fuzzy System Definitions

The architecture and the fuzzy scheduling mechanism use a fuzzy system to determine the priority of each service. By using two monitors (one controlling the response time and other the availability), the possibility of e-contract violation could differ between themselves. This means that the service can have different priority levels. To fix the conflict, the optimizer module uses the higher priority level as the service priority level.

Four linguistic variables are used to generate the system's fuzzy rules: *inclination*, *maximum*, *order*, and *minimum*. The description of the variables is as follows. *Inclination* variable: if the delay in responding to a service is increasing, the service should start earlier. *Maximum* variable: if the time to respond to a service is close to the maximum value of the e-contract response time, the service should start earlier. *Order* variable: a service with a faster processing time should run before services with a slower processing time. *Minimum* variable: if a service is close to the minimum e-contract availability value, this service should start earlier.

The *inclination* variable is determined by the inclination of a straight line. This line is calculated by an interpolation of historical data. On a growing line, the inclination of first-degree equations is between 0 (zero) and 90 (ninety) degrees. The linguistic terms and the fuzzy sets are represented in Figure 2.

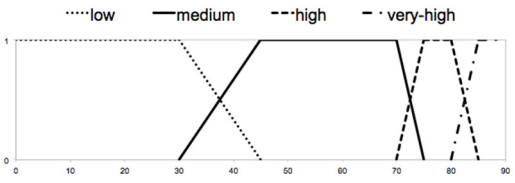

Figure 2: *Inclination* Variable.

The relation between the current response times and the maximum response time set on the e-contract determines the *maximum* variable. The current response times are determined by historical data. The *maximum* variable quantifies how much time is required for the current response times to reach the maximum response time. This variable has a domain between 0 (zero) and 1 (one). The linguistic terms and the fuzzy sets are represented in Figure 3.

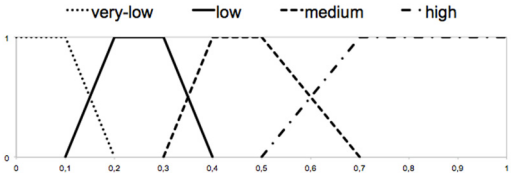

Figure 3: *Maximum* Variable.

The *order* variable is determined by the processing time. If a service has a faster processing time, it should run before services with slower processing time. The processing time is obtained from the historical data. This variable has a domain between 0 (zero) and 1 (one), where zero represents the fastest service processing time and one the slowest. By running the service with the shortest processing time first, the architecture decreases the global average waiting time for all services. The linguistic terms and the fuzzy sets are represented in Figure 4.

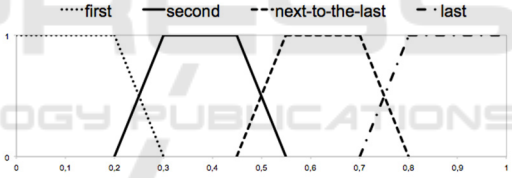

Figure 4: *Order* Variable.

The relation between the current and the minimum availabilities of the e-contract determines the *minimum* variable. The current availability is determined by historical data. The *minimum* variable determines how much the current availability needs to decrease in order to reach the minimum availability. This variable has a domain between 0 (zero) and 1 (one). The linguistic terms and the fuzzy sets are represented in Figure 5.

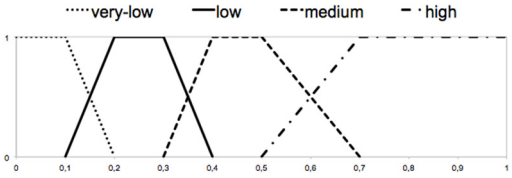

Figure 5: *Minimum* Variable.

The foundation of fuzzy system is composed of twenty-four rules. The rules were created from the combination between linguistic variables and fuzzy sets. The **very-high** and the **very-low** sets have high priority because of the large possibility of violation. The rules were determined empirically. The rules are:

IF *inclination* **IS** very-high **THEN** priority **IS** 5
IF *maximum* **IS** very-low **THEN** priority **IS** 5
IF *minimum* **IS** very-low **THEN** priority **IS** 5
IF *order* **IS** first **AND** *minimum* **IS** low **THEN** priority **IS** 5
IF *order* **IS** first **AND** *minimum* **IS** medium **THEN** priority **IS** 4
IF *order* **IS** first **AND** *minimum* **IS** high **THEN** priority **IS** 3
IF *inclination* **IS** high **AND** *maximum* **IS** low **THEN** priority **IS** 5
IF *inclination* **IS** high **AND** *maximum* **IS** medium **THEN** priority **IS** 4
IF *inclination* **IS** high **AND** *maximum* **IS** high **THEN** priority **IS** 2
IF *order* **IS** second **AND** *minimum* **IS** low **THEN** priority **IS** 4
IF *order* **IS** second **AND** *minimum* **IS** medium **THEN** priority **IS** 3
IF *order* **IS** second **AND** *minimum* **IS** high **THEN** priority **IS** 2
IF *inclination* **IS** medium **AND** *maximum* **IS** low **THEN** priority **IS** 4
IF *inclination* **IS** medium **AND** *maximum* **IS** medium **THEN** priority **IS** 2
IF *inclination* **IS** medium **AND** *maximum* **IS** high **THEN** priority **IS** 1
IF *order* **IS** next-to-the-last **AND** *minimum* **IS** low **THEN** priority **IS** 3
IF *order* **IS** next-to-the-last **AND** *minimum* **IS** medium **THEN** priority **IS** 2
IF *order* **IS** next-to-the-last **AND** *minimum* **IS** high **THEN** priority **IS** 1
IF *inclination* **IS** low **AND** *maximum* **IS** low **THEN** priority **IS** 2
IF *inclination* **IS** low **AND** *maximum* **IS** medium **THEN** priority **IS** 1
IF *inclination* **IS** low **AND** *maximum* **IS** high **THEN** priority **IS** 0
IF *order* **IS** last **AND** *minimum* **IS** low **THEN** priority **IS** 2
IF *order* **IS** last **AND** *minimum* **IS** medium **THEN** priority **IS** 1
IF *order* **IS** last **AND** *minimum* **IS** high **THEN** priority **IS** 0

The inference method determines the priority of each service, with the use of all the rules. Priority is determined by the rule that has the maximum relevance rate.

The proposed architecture and mechanism are completely flexible. They allow the use of other linguistic terms. The fuzzy sets can be defined by other functions, and their limits can be different. The rule base and the number of rules may be different as well. The fuzzy system definitions presented in this paper, chosen empirically, were used for the test scenario. The initial assessment was necessary to determine if the fuzzy theory was appropriate in solve the problem of improving e-contract accomplishment.

4 RESULTS

Initially, the test scenario is presented. Subsequently, some previous results are showed. Then, the new results of the comparison between the fuzzy scheduling mechanism against others are presented.

4.1 Scenario

To validate the proposed architecture and mechanism, a real scenario was created with two providers (PA, and PB), each one offers with two services (SA, and SB), and eight consumers (CA, CB, CC, CD, CE, CF, CG, and CH). Tests were performed in a Local Area Network (LAN) with average latency of 6ms. The LAN is a real network with real traffic.

To simulate a web service composition, consumers could use one service from the provider, both services from the same provider, or services from different providers. The entire management of the compositions was done by twelve e-contracts (ECA, ECB, ECC, ECD, ECE, ECF, ECG, ECH, ECI, ECJ, ECK, and ECL). The scenario with the services composition can be observed in Figure 6.

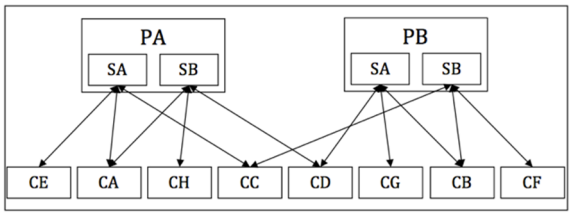

Figure 6: Test Environment.

The consumers, providers, and services involved in the twelve e-contracts used in the test environment can be seen in Table 1. Each e-contract has QoS values for the maximum acceptable value of response time and the minimum acceptable value of availability for each service. Two monitors (MA, and MB) are responsible for the QoS. MA is responsible for the non-functional feature response time and MB is responsible for the non-functional feature availability. For all e-contracts, the maximum agreed response time was 1.25 seconds, and the minimum agreed availability was 95%.

Table 1: e-Contracts.

	CA	CB	CC	CD	CE	CF	CG	CH
PA – SA	ECA		ECE		ECI			
PA – SB	ECB			ECG				ECL
PB – SA		ECC		ECH			ECK	
PB – SB		ECD	ECF			ECJ		

The e-contracts were chosen to simulate requests to one provider only, two services from one provider, and two services from different providers, in order to simulate different situations.

The information stored in the e-contract was: agreed service, the consumer using the service, the provider offering the service, the monitor that monitors the QoS, the non-functional feature of the QoS, and the maximum/minimum value of the non-functional feature monitored.

For the tests, different machines were used. The machines were physically separated and no virtualization was used. Two hosts (Host A, and Host B) were used for the tests. Providers and

Monitors were in Host A and Consumers were in Host B.

The results presented in this paper used Poisson distribution to determine when consumers would make the requests, without external interference. The distribution was parameterized with an average of 7 (seven) requests per minute for all e-contracts, during 30-40 minutes.

4.2 Previous Results

The intelligent architecture/mechanism using fuzzy system aims to prevent a possible e-contract violation by increasing the e-contract accomplishment. This proposal was proved to be very promising in previous works.

When monitoring only the response time, there was an increase of 8.95% in e-contracts accomplishment and a decrease of 31.32% in average response time (Talon *et al.*, 2014). When monitoring only the availability, there was an increase of 18.98% in e-contracts accomplishment. Moreover, in the architecture with two light-weight monitors, there was an increase of 40.41% in e-contracts accomplishment and a decrease of 42.64% in average response time (Talon and Madeira, 2015a). A comparison between light-weight and heavy-weight monitoring was done too. A better performance with the heavy-weight monitoring was observed (Talon and Madeira, 2015b).

4.3 Comparing Scheduling Mechanisms

Various scheduling mechanisms were used to compare the performance of the proposed fuzzy mechanism. The fuzzy scheduling mechanism was compared with queue, random, shorter duedate, shorter response time, and shorter processing time scheduling mechanisms. Details about the mechanisms are described in section 3.

These results can be seen in Tables 2, 3, 4, 5, and 6. In all tables, the first line shows the results of the e-contracts accomplishment according to the availability. The second line shows the results of the e-contracts accomplishment according to the consumer's side response time. The third line shows the results of the e-contracts accomplishment according to the provider's side response time. The penultimate line shows the average response time according to consumer's side. The last line shows the average response time according to provider's side.

Table 2 shows the comparison between the fuzzy scheduling and the traditional queue scheduling (first in first out).

Table 2: Fuzzy and Queue comparison.

	Fuzzy	Queue	Increase/Decrease
Availability	90.40% +/- 0.49%	70.65% +/- 1.95%	+27.95%
Consumer's Side Response Time	84.57% +/- 0.57%	64.30% +/- 2.41%	+31.52%
Provider's Side Response Time	86.31% +/- 0.57%	67.44% +/- 1.32%	+27.98%
Consumer's Side Response Time Average	0.843 +/- 0.014	1.279 +/- 0.047	-34.08%
Provider's Side Response Time Average	0.789 +/- 0.014	1.210 +/- 0.027	-34.79%

Table 3 exhibits the comparison between the fuzzy scheduling and a random scheduling. For each request a random number is assigned and the provider responds according to the order of the numbers.

Table 3: Fuzzy and Random comparison.

	Fuzzy	Random	Increase/Decrease
Availability	90.40% +/- 0.49%	73.81% +/- 1.87%	+22.47%
Consumer's Side Response Time	84.57% +/- 0.57%	67.82% +/- 2.84%	+24.69%
Provider's Side Response Time	86.31% +/- 0.57%	70.04% +/- 2.10%	+23.22%
Consumer's Side Response Time Average	0.843 +/- 0.014	1.284 +/- 0.082	-34.34%
Provider's Side Response Time Average	0.789 +/- 0.014	1.225 +/- 0.073	-35.59%

Table 4 shows the comparison between the fuzzy scheduling and the shorter duedate to reach the limit in the e-contract, meaning that the request with the closest time to violation will be responded first by the provider.

Table 4: Fuzzy and Shorter Duedate comparison.

	Fuzzy	Shorter Duedate	Increase/Decrease
Availability	90.40% +/- 0.49%	76.72% +/- 1.31%	+17.83%
Consumer's Side Response Time	84.57% +/- 0.57%	71.25% +/- 1.80%	+18.69%
Provider's Side Response Time	86.31% +/- 0.57%	73.17% +/- 2.19%	+17.95%
Consumer's Side Response Time Average	0.843 +/- 0.014	1.135 +/- 0.052	-25.72%
Provider's Side Response Time Average	0.789 +/- 0.014	1.084 +/- 0.051	-27.21%

Table 5 represents the comparison between the fuzzy scheduling and the shorter response time scheduling.

Table 5: Fuzzy and Shorter Response Time comparison.

	Fuzzy	Shorter Response Time	Increase/Decrease
Availability	90.40% +/- 0.49%	82.51% +/- 0.79%	+9.56%
Consumer's Side Response Time	84.57% +/- 0.57%	78.20% +/- 0.96%	+8.14%
Provider's Side Response Time	86.31% +/- 0.57%	79.64% +/- 1.00%	+8.37%
Consumer's Side Response Time Average	0.843 +/- 0.014	0.980 +/- 0.015	-13.97%
Provider's Side Response Time Average	0.789 +/- 0.014	0.927 +/- 0.017	-14.88%

Table 6 shows the comparison between the fuzzy scheduling and the shorter processing time scheduling.

In all the comparisons, the fuzzy system is better than others scheduling. Though, the results of the

shorter duedate, shorter response time, and shorter process time schedules were close to the results of the fuzzy scheduling. This outcome was expected because the fuzzy system uses the same reasoning as their fuzzy variables.

Table 6: Fuzzy and Shorter Processing Time comparison.

	Fuzzy	Shorter Processing Time	Increase/Decrease
Availability	90.40% +/- 0.49%	86.05% +/- 0.81%	+5.05%
Consumer's Side Response Time	84.57% +/- 0.57%	81.24% +/- 0.98%	+4.09%
Provider's Side Response Time	86.31% +/- 0.57%	83.21% +/- 0.27%	+3.72%
Consumer's Side Response Time Average	0.843 +/- 0.014	0.876 +/- 0.008	-3.76%
Provider's Side Response Time Average	0.789 +/- 0.014	0.827 +/- 0.005	-4.59%

Even though the results are close, especially between the fuzzy scheduling and the shorter processing time scheduling, the proposed approach displayed better load balancing in the providers. When using fuzzy scheduling, all e-contracts were treated similarly, with little variation amongst them. With others scheduling, some e-contracts were accomplished at 100% of the time, while other e-contracts had high rates of violation. Table 7 shows the variation between the highest and lowest values. The columns related to "e-contract accomplishment" represent the number of contracts that were accomplished compared to the total number of executed contracts (accomplished contracts plus violated contracts) and the columns related to "average response time" represent the response time in seconds.

Table 7: Variation – Best and Worst Results.

	e-contract accomplishment			average	
	availability	response time		response time	
		consumer	provider	consumer	provider
fuzzy	5.776%	8.436%	6.996%	0.145	0.155
queue	20.090%	23.023%	21.206%	0.470	0.450
random	18.873%	18.296%	17.183%	0.463	0.480
shorter duedate	14.200%	16.813%	15.600%	0.465	0.455
shorter response time	22.686%	21.966%	22.293%	0.575	0.570
shorter processing time	36.080%	41.156%	41.0265	0.790	0.780

Table 7 shows that the highest variation of e-contract accomplishment using fuzzy scheduling was 8.43%. The scheduling with the lowest variation of e-contract accomplishment was the shorter duedate with 14.2%, which represents an increase of 68.44% in the variation. The highest variation of average response time using fuzzy scheduling was 0.155 seconds. The scheduling with the lowest average variation of response time was the queue with 0.45 seconds, which a consequent increase of 190.32% in the variation.

5 RELATED WORKS

Related works that deal with one or more topics relevant to our research are presented in this section.

With respect to monitoring, it is possible to list a few works. A soft-intrusive monitoring can be found in (Michlmayr *et al.*, 2009), while an asynchronous monitoring can be found in (Wetzstein *et al.*, 2009). The monitor proposed in this paper has three distinct aspects combined: soft-intrusive, asynchronous, and light-weight/heavy-weight monitoring. The current monitor can handle both the light-weight and the heavy-weight monitoring. One difference between the researches with asynchronous monitoring and ours is that others discovered a failure after process execution while ours attempted to predict it during process execution (in parallel).

Some authors use one or more autonomic properties in their work. The self-configuration property can be found in (Mannava and Ramesh, 2012), the self-optimization property can be found in (Gounaris *et al.*, 2008), and the self-healing property can be found in (Angarita *et al.*, 2016). In (Alférez *et al.*, 2014) a self-adaptive system is presented. The architecture proposed in this paper has three autonomic properties combined: self-configuration, self-optimization, and self-healing.

Other authors have used fuzzy theory in web service environments: (Shafiq *et al.*, 2014) and (Chouiref *et al.*, 2016). However, none use fuzzy theory to optimize the e-contracts accomplishments in web services. The vast majority of works are related to the discovery or selection of services.

The work of (Pernici and Siadat, 2011) is similar to ours in terms of environment phases (formation, execution, monitoring, and adaptation) and fuzzy base rules. However, their architecture defines actions to replace or negotiate services and ours defines actions to improve current services.

6 CONCLUSIONS

The main goal of this work was to propose a fuzzy scheduling mechanism to predict possible e-contract violations based on services historical data. With the prediction, this work tries to increase e-contract accomplishment (consequently decrease e-contract violation) using the fuzzy theory.

The proposed architecture can be classified as self-adaptive because it shows three self-* properties: (i) *Self-Configuration Property*: Based on a fuzzy system, the analyzer module changes

services priority. The provider processes services with high e-contract violation possibility first; (ii) *Self-Optimization Property*: The optimizer module uses all analysis made from historical data to take pro-active actions in order to decrease the average response time of services and to increase the average services availability; and, (iii) *Self-Healing Property*: As soon as the monitor detects an e-contract violation, the recovery module is responsible for fixing the violation.

Comparing the fuzzy scheduling mechanism with other scheduling mechanisms, an improvement of 31.52% is observed in the e-contracts accomplishment and a decrease of 35.59% in average response time. Furthermore, using the fuzzy scheduling mechanism, the overload of the provider was better balanced varying at most 8.43%, while for the other scheduling mechanisms the variation reached 41.15%. In all comparisons, when the fuzzy system determines the order of the services, the results are better than other scheduling mechanisms.

In further work, experiments will be run with more services in each providers, to test the impact of the fuzzy system. Tests will also be performed to compare the proposed approach with other methods (statistical regression, machine learning, neural networks, etc.). In addition, the use of genetic algorithms to optimize the mechanism will be investigated.

ACKNOWLEDGEMENTS

We would like to thank FAPESP and CNPq for the financial support.

REFERENCES

Alférez, G. H., Pelechano, V., Mazo, R., Salinesi, C., Diaz, D., 2014. Dynamic adaptation of service compositions with variability models. *Journal of Systems and Software*. Volume 91, Pages 24-47, ISSN 0164-1212, May.

Angarita, R., Rukoz, M., Cardinale, Y., 2016. Modeling dynamic recovery strategy for composite web services execution. *World Wide Web* 19, 1 (January 2016), 89-109.

Chouiref, Z., Belkhir, A., Benouaret, K., Hadjali, A., 2016. A fuzzy framework for efficient user-centric Web service selection. *Appl. Soft Comput.* 41, C (April 2016), 51-65.

Fantinato, M., Gimenes, I. M. S., Toledo, M. B. F., 2010. Product Line in the Business Process Management Domain. In: Kyo C. Kang, Vijayan Sugumaran, Sooyong Park. (Org.), *Applied Software Product Line Engineering*, 1st ed. Boca Raton, FL: Auerbach Publications, pp. 497-530.

Gounaris, A., Yfoulis, C., Sakellariou, R., Dikaiakos, M. D., 2008. A control theoretical approach to self-optimizing block transfer in Web service grids. *ACM Trans. Auton. Adapt. Syst.* 3, 2, Article 6 (May 2008), 30 pages.

Huebscher, M. C., McCann, J. A., 2008. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.* 40, 3, Article 7 (August 2008), 28 pages.

Mannava, V., Ramesh, T., 2012. Multimodal pattern-oriented software architecture for self-configuration and self-healing in autonomic computing systems. In *Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology* (CCSEIT '12). ACM, New York, NY, USA, 382-389.

Michlmayr, A., Rosenberg, F., Leitner, P., Dustdar, S., 2009. Comprehensive QoS monitoring of Web services and event-based SLA violation detection. In *Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing* (MWSOC '09). ACM, New York, NY, USA, 1-6.

Papazoglou, M. P., Traverso, P., Dustdar, S., Leymann, F., 2008. Service-Oriented Computing: A Research Roadmap. *International Journal of Cooperative Information Systems*, Vol 17 No. 2, 233-255.

Pernici, B., Siadat, S. H., 2011. Selection of Service Adaptation Strategies Based on Fuzzy Logic. In *Proceedings of the 2011 IEEE World Congress on Services* (SERVICES '11). IEEE Computer Society, Washington, DC, USA, 99-106.

Shafiq, O., Alhajj, R., Rokne, J., 2014. Log based business process engineering using fuzzy web service discovery. *Knowledge-Based Systems*. Volume 60, Pages 1-9, ISSN 0950-7051, April.

Talon, A. F., Madeira, E. R. M., Toledo, M. B. F., 2014. Self-Adaptive Fuzzy Architecture to Predict and Decrease e-Contract Violations. *Intelligent Systems* (BRACIS), 2014 Brazilian Conference on, Sao Paulo, pp. 294-299.

Talon, A. F., Madeira, E. R. M., 2015a. Improvement of E-Contracts Accomplishments by Self-Adaptive Fuzzy Architecture. *Services Computing* (SCC), 2015 IEEE International Conference on, New York, NY, pp. 507-514.

Talon, A. F., Madeira, E. R. M., 2015b. Comparison between Light-Weight and Heavy-Weight Monitoring in a Web Services Fuzzy Architecture. In *Procedia Computer Science*, Vol. 64, pp. 862-869.

Wetzstein, B., Leitner, P., Rosenberg, F., Brandic, I., Dustdar, S., Leymann, F., 2009. Monitoring and Analyzing Influential Factors of Business Process Performance. In *Proceedings of the 2009 IEEE International Enterprise Distributed Object Computing Conference* (edoc 2009) (EDOC '09). IEEE Computer Society, Washington, DC, USA, 141-150, 2009.

Yager, R. R., Filev, D. P., 1994. "Essentials of Fuzzy Modeling and Control". Wiley-Interscience, New York, NY, USA.