Spectrum-Based Fault Localization on a Collaboration Graph of a System-of-Systems

Yong-Jun Shin, Sangwon Hyun, Young-Min Baek, and Doo-Hwan Bae
School of Computing
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon, South Korea
{yjshin, swhyun, ymbaek, bae}@se.kaist.ac.kr

***Abstract*—A System-of-Systems (SoS) consists of independent and autonomous constituent systems (CSs) which collaborate to achieve an SoS goal. For SoS engineers, it is important to verify the results of the collaboration for an SoS goal. Statistical verification can be used to verify a large and complex SoS and to provide quantitative verification results. However, even when a failure of an SoS goal or a violation of a verification property is detected, it often requires a huge cost to find faults of an SoS because of the size of the SoS and the lack of information about independent CSs. In this paper, we propose a fault localization technique for an SoS to reduce the debugging cost by prioritizing suspicious entities of an SoS. This technique requires only an abstract model of the collaboration, named as a collaboration graph, which includes the presence or absence of CSs and their interactions. It extends a spectrum-based fault localization (SBFL) technique to utilize quantitative results of the statistical verification of an SoS. In the evaluation, we show that it is feasible to apply SBFL to SoS fault localization, and our approach is expected to effectively reduce the debugging space of an SoS.**

***Index Terms*—System-of-Systems, collaboration graph, fault localization, spectrum-based fault localization, debugging**

I. Introduction

A System-of-Systems (SoS) is a large-scale complex system which consists of independent and autonomous constituent systems (CSs) contributing to achieve an SoS-level common goal(s) [1]. The SoS goal is one that is hard to achieve with a monolithic system, thus it is achieved by collaborations and cooperations of the CSs as they interact with each other utilizing SoS resources [2]. To guarantee the goal achievement of an SoS, CSs in an SoS should effectively provide their own capabilities in the form of collaborations. In addition, interactions between CSs should enable an effective collaboration within an SoS.

For SoS engineers, it is important to verify whether an SoS goal can be successfully fulfilled by the collaboration. For the SoS verification, statistical verification is one of the well known techniques that provides quantitative verification results [3], [4]. However, even if the SoS model has demonstrably failed to fully achieve goals or satisfy properties, identifying and localizing faults that induce the failure of an SoS are additional concerns. Because of the size and complexity of an SoS, the cost of finding the faults of an SoS is too high. Furthermore, SoS engineers may not have sufficient information to inspect independent CSs. Therefore, there is a need to reduce the cost of finding faults of an SoS for efficient debugging.

The cost can be cut down by localizing areas or entities that induce failures or by providing search priorities. Studies on fault localization have tried to provide systematic methods to find the location of bugs in a system. The studies have focused on localizing bugs in the source code or components of the system. However, the granularity of localization is not suitable for SoS debugging, and those techniques do not take into account the characteristics of an SoS that the independent and autonomous CSs collaboratively achieve SoS goals. Therefore, a localization technique that considers the unique characteristics of an SoS is necessary. In this paper, we propose a spectrum-based fault localization technique to prioritize debugging of suspicious CSs or interactions that induce an SoS failure. The main goal of this study is to propose a fault localization technique specifically applicable to an SoS, and the major features of our technique can be summarized as follows:

- It applies a spectrum-based fault localization (SBFL) technique to the collaboration graph of an SoS, which is an abstract model of CS collaboration and interactions. This technique aims to localize CSs and interactions that induce a failure of an SoS.
- It considers the lack of information available to SoS engineers about independent and autonomous CSs. By representing an SoS as a collaboration graph, it can be used even when the only information available is the presence or absence of entities participating in the collaboration.
- It utilizes the quantitative results of statistical verification of an SoS. Accumulated statistical verification results of diverse collaborations of an SoS can be used for fault localization.

The remainder of this paper is organized as follows: Sections 2 and 3 introduce the related works and background of this work. Section 4 introduces our fault localization technique on the collaboration graph of an SoS. Sections 5 and 6 show the experiment and evaluation to validate our localization technique. Section 7 introduces the discussion points of this study. Section 8 concludes this study by noting its contributions and pointing towards future work.

II. Related Work

Fault localization technique is one that identify locations of faults in a software, and several techniques have been proposed [5]–[7]. They localize faults or root causes of the software failure by using static code analysis or testing execution information. Some studies also have presented methods to localize faults on a concurrent program that has non-deterministic behavior [6], [7]. However, the localization unit of those studies is a statement or a predicate of the source code. Since an SoS is not a monolithic system but a large-scale system with many constituent systems combined, it is difficult to directly apply these methods to SoS engineering.

Several techniques have localized faults of a large and complex system with multiple components, such as distributed network systems [8]–[11]. Some studies represent a target system as a graph or a set of component systems [8], [9], [11], and they utilize execution information from the components or the structure of the system to localize faults. However, since an SoS is not a simple set of components but a group of CSs that dynamically collaborate, a fault localization technique considering the characteristics of SoS collaboration is needed.

Some other techniques manipulated the component systems of the distributed system or extracted information from them. Qi *et al.* utilized a neural network to analyze the execution information of the component system [10]. Pham *et al.* proposed a fault injection method to determine the root causes of faults in the distributed system [8]. Rish *et al.* presented an active probing algorithm that maximizes information gain and minimizes the size of the probe set adaptively to identify performance-degrading components [11]. However, because the CSs that make up an SoS are developed and operated independently and they often collaborate for the SoS goal in a bottom-up manner, SoS engineers find it difficult to rely on manipulation of or information from CSs.

Previous techniques have attempted to minimize the fault localization cost of large-scale systems. However, a technique for SoS debugging, considering the collaborative goal achievement by independent and autonomous CSs and the limited information SoS engineers can glean from CSs, is needed. In this paper, we propose a fault localization approach for SoS.

III. Background

Spectrum-based fault localization (SBFL) is one of the fault localization techniques that localize suspicious entities using information of program spectra. A program spectrum is an executed trace of a program with specific granularity, such as statements, predicates, or methods [5]. A key idea of the SBFL is that the more times an entity is executed in failed cases, the more suspicious the entity is. The input of the SBFL technique is a set of pairs of program spectra and the testing result of the spectra. Recent studies of SBFL use both failed and successful test cases [5]. By using these inputs, the SBFL is able to calculate a suspiciousness value for each entity of a program, such as a statement of source code.

In this paper, we extend Tarantula [12], which is one of the widely used SBFL techniques, for fault localization on the collaboration of an SoS. Because of the size of an SoS, it is hard to apply typical SBFL on the SoS using statement or predicates of source code as a localization entity. However, in terms of the collaboration of the SoS, an entity participating in a collaboration can show a spectrum of collaboration and can be regarded as a localization entity. A set of entities in an SoS collaboration as a spectrum and their verification result can be used as an input of the SBFL technique. In this paper, we propose an extension of the SBFL technique specifically for SoS, regarding CSs and their interactions participating in an SoS collaboration as localization entities.

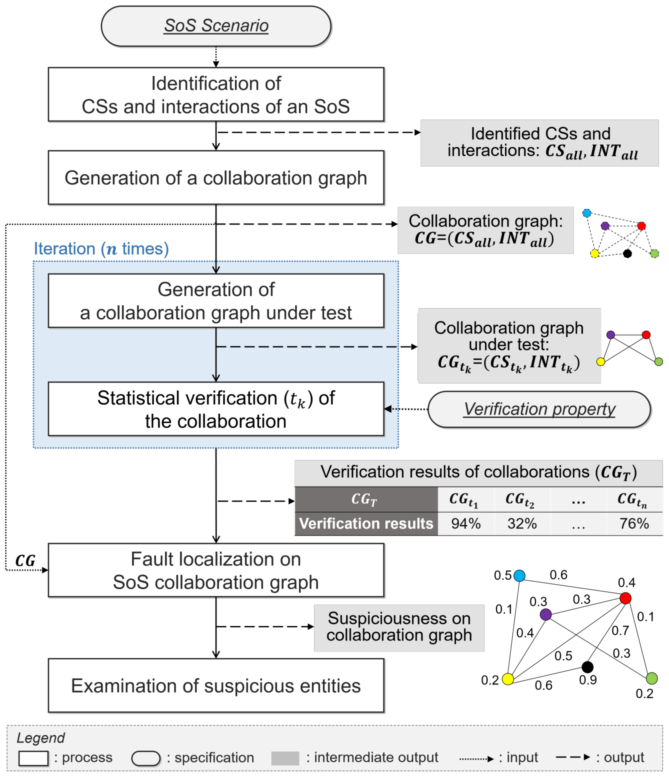
SoS Scenario
Identification of CSs and interactions of an SoS
Identified CSs and interactions: CS_{all}, INT_{all}
Generation of a collaboration graph
Collaboration graph: $CG=(CS_{all}, INT_{all})$
Iteration (n times)
Generation of a collaboration graph under test
Collaboration graph under test: $CG_{t_k}=(CS_{t_k}, INT_{t_k})$
Statistical verification (t_k) of the collaboration
Verification property
Verification results of collaborations (CG_T)
CG_T
CG_{t_1}
CG_{t_2}
…
CG_{t_n}
Verification results
94%
32%
…
76%
CG
Fault localization on SoS collaboration graph
Suspiciousness on collaboration graph
0.5
0.6
0.4
0.1
0.3
0.3
0.1
0.4
0.7
0.5
0.3
0.9
0.2
0.6
0.2
Examination of suspicious entities
Legend
: process
: specification
: intermediate output
: input
: output

Fig. 1. Overall process of spectrum-based fault localization on a collaboration graph of an SoS

IV. Fault Localization of a System-of-Systems

A. Overall process

We propose a spectrum-based fault localization (SBFL) technique to prioritize suspicious entities on a collaboration of an SoS. Figure 1 shows the overall process of localization for an SoS. From the given SoS scenario specification, SoS engineers identify CSs and interactions that belong to the SoS. An SoS may have multiple CSs that can participate in the collaboration. In addition, two CSs may or may not be able to interact with each other. Identified CSs and interactions from the scenario specification are modeled as a set of nodes and edges of the collaboration graph. A collaboration graph, which is an abstract model of collaborations in an SoS, is generated as a combination of a set of nodes and a set of edges by a definition of a graph. In this paper, although each CS and interaction may have detailed specifications, we abstract an

SoS and focus on defining a model representing a collaboration at the SoS level and applying an SBFL technique.

From the collaboration graph of an SoS, SoS engineers can generate a partial collaboration of the SoS, called a collaboration graph under test. Statistical verification shows the extent to which an SoS goal is satisfied by the collaboration. For statistical verification, an SoS goal is represented as a verification property [3], [13]. The verification result is a quantitative possibility of satisfying the goal by the given collaboration. For SoS verification, we utilize statistical model checking (SMC). With the iterative verification, accumulated pairs of collaboration graphs under test and statistical verification results are inputs of the fault localization technique. At this point, the suspiciousness of each node or edge of an SoS collaboration graph is calculated. The suspiciousness of an SoS entity is a quantitative indicator of how often an SoS goal was not achieved when the entity participated in SoS collaboration. The equation of the suspiciousness of an SoS entity can be extended from various SBFL formulas, and we use the Tarantula formula [12]. The suspiciousness of an entity is a value between 0 and 1, where a larger number means a higher possibility of inducing the failure of an SoS goal. Based on the calculated suspiciousness values, failure-inducing CSs or interactions can be prioritized to be considered first by SoS engineers for SoS failure debugging.

Calculating the suspiciousness of an entity is basically increasing (decreasing) the suspiciousness of an entity in a collaboration graph under test which has low (high) goal achievement, respectively. Details of the input/output and the method of the SBFL technique on an SoS collaboration graph are described in the next sections.

B. Collaboration Graph of a System-of-Systems

An SoS consists of multiple CSs which can interact with each other and share information. The collaboration of CSs allows an SoS to achieve its goals. However, even though CSs belong to an SoS, the CSs are independent and autonomous, so each is limited in the information it can offer to SoS engineers. Therefore, in this paper, we represent the SoS collaboration as a graph-formed model, named as a collaboration graph. The collaboration graph CG is defined as:

$$\begin{aligned} CG =& (CS_{all}, INT_{all}) \\ CS_{all} =& \{v_i | v_i \text{ is an } i\text{th constituent system of an SoS}, \\ & 1 \leq i \leq m, \text{where } m \text{ is the total number of} \\ & \text{constituent systems in an SoS}\} \\ INT_{all} =& \{e_{v_a,v_b} | e_{v_a,v_b} \text{ is an interaction between } v_a \text{ and } v_b, \\ & \text{where } v_a, v_b \in CS_{all}, a \neq b, e_{v_a,v_b} = e_{v_b,v_a}\} \end{aligned} \tag{1}$$

CS_{all} refers to a set of all CSs belonging to an SoS, and is abstracted as nodes of the collaboration graph. INT_{all} refers to a set of all interactions that may exist between two arbitrary CSs, and is abstracted as edges of the collaboration graph. In this paper, an interaction is regarded as a bi-directional edge. SoS engineers can identify CSs and interactions and graphically model the collaboration of the SoS.

	Collaboration graph under test (CG_{t_k})							
Collaboration graph of SoS (CG)	1	2	3	4	5	6	*Sus.	Rank
Node (CS_{all}) {								
CS1	●		●		●	●	0.48	6
CS2	●	●		●			0.4	7
CS3 // fault			●	●		●	0.73	2
CS4	●	●		●	●	●	0.48	5
}								
Edge (INT_{all}) {								
CS1-CS2	●						0.1	9
CS1-CS3			●				0.6	4
CS2-CS3				●			0.7	3
CS2-CS4	●	●					0.25	8
CS3-CS4 // fault				●		●	0.8	1
}								
Goal achievement probability	0.9	0.6	0.4	0.3	0.7	0.1		

*Suspiciousness

Fig. 2. Example of SBFL on a collaboration graph of an SoS

An SoS can dynamically create a collaboration using resources, including CSs and their interactions, to maximize SoS goal achievement. SoS engineers can select some CSs and interactions and verify their collaboration. The collaboration can also be represented as a graph and we call it a collaboration graph under test. The *k*th collaboration graph under test CG_{t_k} is defined as:

$$\begin{aligned} CG_{t_k} &= (CS_{t_k}, INT_{t_k}) \\ CS_{t_k} \subseteq CS_{all}&, INT_{t_k} \subseteq INT_{all} \end{aligned} \tag{2}$$

CG_{t_k} is a subgraph of the CG of an SoS because all possible CSs and interactions of an SoS are represented in the collaboration graph, and some of them are selected for verification of potential collaboration.

In order to achieve the SoS goal, the collaboration of various CS and interaction combinations can be considered, and each of them can have different goal achievement. The goal achievement of each collaboration is obtained through statistical verification [3]. We assume that SoS engineers have a number of collaboration graphs under test (CG_{t_k}s) and their verification results. The set of pairs of a CG_{t_k} and verification result is the input of our SBFL technique for an SoS. The suspiciousness value calculation is covered in the next section.

C. Suspiciousness calculation on collaboration graph

Given a set of pairs of an SoS collaboration graph under test (CG_{t_k}) and a verification result, the diverse potential collaborations of an SoS and their expected goal achievement can be shown. This information is used to localize CSs and interactions that induce SoS failure, in the same way that a test suite is used to find a software bug. Figure 2 shows an example of our technique. In the example, the collaboration graph (CG) of an SoS has four CSs and five interactions that can participate in a collaboration, and the graph is represented by sets of nodes and edges. There are six CG_{t_k}s, and dots show participation of an entity in a collaboration. Every CG_{t_k} has a goal achievement probability that was statistically verified. In the example, *CS3* and the interaction between *CS3* and *CS4* have faults which affect SoS goal achievement. Collaborations including the faulty CS or interaction have relatively lower goal achievement probabilities than the others. Based on the diverse combinations of SoS entities and goal achievement results, the suspiciousness values are calculated and the faulty CS and interaction get higher suspiciousness values than the other entities.

The suspiciousness of an entity x in a collaboration graph, either a CS (v) or an interaction (e), is defined with the following equation:

$$suspiciousness(x) = \frac{\frac{failedProb(x)}{totalFailedProb}}{\frac{passedProb(x)}{totalPassedProb} + \frac{failedProb(x)}{totalFailedProb}} \tag{3}$$

Equation 3 is extended from the Tarantula technique [12] to utilize statistical verification results. In the equation, there are four parameters. $passedProb(x)$ is the sum of *goal achievement probabilities* of CG_{t_k}s that include the entity x. In the example of Figure 2, $passedProb(x)$ of the edge *CS3-CS4* is $0.4(= 0.3 + 0.1)$. Similarly, $failedProb(x)$ is the sum of goal failure probabilities, which is ($1-$*goal achievement probability*), of CG_{t_k}s that include the entity x. In the example, $failedProb(x)$ of the edge *CS3-CS4* is $1.6(= 0.7 + 0.9)$. $totalPassedProb$ is the sum of goal achievement probabilities of all CG_{t_k}s. Likewise, $totalFailedProb$ is the sum of goal failure probabilities of all CG_{t_k}s. In the example, $totalPassedProb$ and $totalFailedProb$ are $3(= 0.9 + 0.6 + 0.4 + 0.3 + 0.7 + 0.1)$ and $3(= 0.1 + 0.4 + 0.6 + 0.7 + 0.3 + 0.9)$, respectively.

Following Equation 3, suspiciousness of all nodes and edges can be calculated using the verification results. In addition, based on the calculated suspiciousness values, CSs and interactions of an SoS can be prioritized for efficient debugging. An entity of an SoS that has a higher possibility to induce SoS failure may have a higher rank in the debugging order by virtue of its suspiciousness. Therefore, SoS engineers can reduce debugging costs.

V. Experiment

To validate our localization technique, we apply our technique to a specific SoS example, mass casualty incident (MCI) response SoS. A goal of the SoS is rescuing more than 80% of patients within a given time frame. There are two kinds of heterogeneous CSs: rescue robot and patrol drone. A rescue robot randomly searches patients in the area of an incident and rescues a patient relying on its rescue rate. A patrol drone scouts the area of an incident faster than robots, helping robots to find patients faster and relying on the patient recognition rate of its camera. The location information of patients discovered by the drones is transmitted to the robots. The interaction has a message delivery delay as an attribute. The independent robots, drones, and interactions can have different capabilities or attributes, which can affect the patient rescue of the MCI response SoS.

Based on the scenario, all CSs and interactions are modeled as a collaboration graph of the SoS, and their capabilities and attributes were initialized randomly. To validate whether our technique can localize failure-inducing CSs and interactions, we randomly inserted faults into the capabilities of CSs and interactions. In the fault-seeded SoS, the collaboration graph under test is randomly selected and statistically verified. The statistical verification was conducted by an open source tool, SIMVA-SoS (Simulation-based Verification and Analysis for SoS) [13][1]. SIMVA-SoS executes the discrete event simulation and statistical model checking using the SPRT algorithm on the input model. The verification results were used for fault localization, and the localization results were analyzed, whether or not fault-seeded entities were localized. Details of the scenario and fault seeding settings are in Table I.

TABLE I
EXPERIMENT SETUP

Parameter	Value
Simulation time	130 tick
SoS goal (MCI response SoS shall achieve a patient rescue rate of $x\%$ or greater within simulation time.)	0.8
The number of patients	100
Size of the map	20×20 tiles
The total number of rescue robots of SoS	10
The total number of patrol drones of SoS	10
The total number of interactions of SoS	100
The number of faulty rescue robots of SoS in ***scenario 1***	2
Rescue rate of a rescue robot **(fault-seeded - *scenario 1*)**	about 1 **(0.5)**
The number of faulty patrol drones of SoS in ***scenario 2***	2
Patient recognition rate of a patrol drone **(fault-seeded - *scenario 2*)**	about 1 **(0.1)**
The number of faulty interactions of SoS in ***scenario 3***	2
Interaction delay **(fault-seeded - *scenario 3*)**	1 **(50)** tick
The number of rescue robots participating in a partial collaboration graph	3
The number of patrol drones participating in a partial collaboration graph	3
The number of interactions participating in a partial collaboration graph	3
The size of a set of pairs of collaboration graph under test and verification result used for a localization	100

The MCI response SoS of our experiment has 10 rescue robots, 10 patrol drones, and 100 interactions between the robots and drones. The CSs and interactions compose an SoS collaboration graph, and collaboration graphs under test, including randomly chosen three each of robots, drones, and interactions, were verified. There are three scenarios for evaluation. In *scenario 1*, there are two faulty rescue robots out of 10, and the rescue rate of a normal robot is about 1 while that of a faulty robot is about 0.5. In *scenario 2*, there are two faulty patrol drones out of 10, and the patient recognition of a normal drone is about 1 while that of a faulty drone is about 0.1. *Scenario 1* and *scenario 2* have artificial faults in CSs, which are nodes of a collaboration graph. *Scenario 3* has faults in interactions between CSs. There are two faulty interactions out of 100, whose interaction delay is 50 tick, but that of a normal interaction is 1 tick. The parameters representing faults of the three scenarios were set so that a fault reduces the SoS goal achievement probability variously from about 10% to 50%. We have repeatedly tested each of these three scenarios 90 times to evaluate the performance and feasibility of our localization technique for an SoS.

[1]SIMVA-SoS: github.com/SESoS/SIMVA-SoS

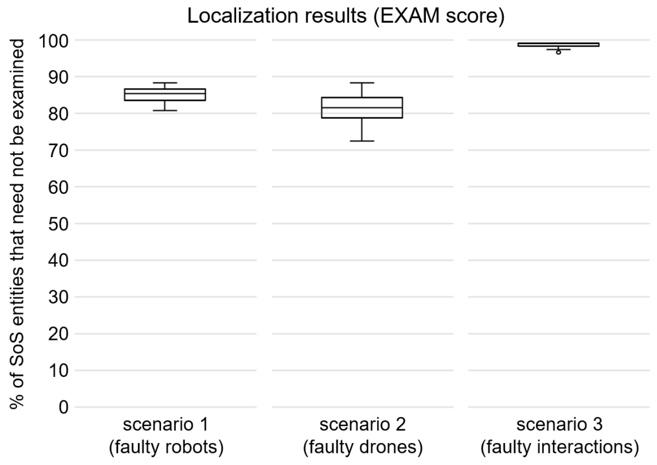

Fig. 3. Reduced costs to find faults of an SoS in experimental scenarios

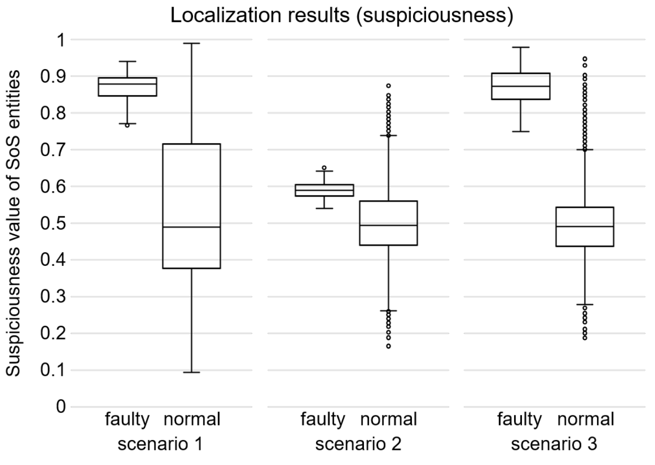

Fig. 4. Suspiciousness values for finding faults of an SoS

VI. Evaluation

To ensure the localization technique for an SoS is actually helpful to SoS engineers seeking to find faults of the SoS, we identified an EXAM score [12] of the localization results for each scenario in Figure 3. The EXAM score is a percentage of SoS entities that engineers do not need to examine in order to find faults in the collaboration graph. The total entities for localization in this experiment are all of CSs and interactions of the SoS collaboration graph, for a total of 120 entities. Figure 3 shows the reduced cost to find faults of the SoS.

In all three scenarios, we can see that on average, about 88% of entities do not need to be examined. In particular for *scenario 3*, where the faults were inserted in interactions, in most cases, all faulty interactions were localized at the top of the suspiciousness list, so about 99% of entities do not need to be examined. *Scenarios 1 and 2*, in which faults were inserted in CSs, scored an average of 85.41% and 81.66%, respectively.

In terms of EXAM scores, *scenario 1 and 2*, where there were faults in the nodes of the collaboration graph, showed lower performance than *scenario 3*, where there were faults in the edges. This is because if a node has a fault, the edges connected with that node are highly suspected together. This is a natural result, because the edges belong to the collaboration graph together with the faulty node. However, when a node has a fault, the majority of the edges that are more suspicious than the node are connected to the faulty node, so the SoS engineers can have an intuition to find the real fault. In fact, for *scenarios 1 and 2*, about 60% of the edges that got higher suspiciousness than a faulty CS were connected with the faulty CS. Therefore, the localization results effectively reduced debugging cost.

Figure 4 shows the suspiciousness values for each faulty or normal entity. In all three scenarios, the suspiciousness values of faulty entities are generally higher than those of normal entities, indicating that suspiciousness value identifies a fault. This indicates that the likelihood that entities have a fault can be compared based on their suspiciousness values. Figure 4 also shows the performance difference by scenarios. In *scenario 3*, the suspiciousness values of faulty interactions were higher than that of all normal entities except outliers. On the other hand, *scenarios 1 and 2* had normal entities with higher suspiciousness than faulty CSs. Most of those were interactions connected to faulty CSs as described above.

In Figure 4, the differences between *scenario 1 and 2* are noticeable. The suspiciousness values of faulty CSs in *scenario 2* were lower than those in *scenario 1*. It also appears that *scenario 2* has a lower EXAM score than *scenario 1* in Figure 3. This is because the different faults have different impacts on achieving SoS goals. A suspiciousness is calculated differently depending on a goal achievement probability by statistical verification result. In fact, the fault of a rescue robot in *scenario 1* reduced the goal achievement possibility by about 48%, while the fault of a patrol drone in *scenario 2* reduced the goal achievement by only about 15%. It can be seen that the suspiciousness of an SoS entity is calculated to be higher as the entity further hinders the goal achievement. It confirms that our localization technique distinguishes faulty SoS entities from normal entities and gives a suspiciousness depending on the impact each entity has on the SoS goal.

This experiment is limited to the given scenarios but showed that the SBFL technique can be applied to a collaboration graph of an SoS and effectively reduces the debugging cost. In addition, EXAM score varies depending on whether there is a fault in the node or edge of the collaboration graph, and suspiciousness values are given depending on the impact on goal achievement. Moreover, by demonstrating the feasibility of the localization technique for an SoS, we underscore the need for further research on localization utilizing SoS models and quantitative verification results.

VII. Discussion

SoS Representation. For simulation-based analysis, the accuracy of verification results depends on the precision of the simulation model. From a high-level perspective, an SoS can be defined as a goal-oriented system consisting of heterogeneous and autonomous CSs that each possesses contributory capability. This study models an SoS as a collaboration graph that represents highly abstracted SoS structures and capability-based behaviors. Since an SoS consists of CSs with limited knowledge (i.e., black-box), the abstract representation might be a more realistic approach. Also, the collaboration graphs are shown to be successfully used for fault localization, even though they are quite abstract models. If a simulation model is more elaborately established based on domain knowledge and meta-models [2], the effectiveness of fault localization can be further improved.

Statistical Model Checking. Our approach utilizes simulation-based statistical model checking (SMC) as a quantitative analysis method for fault localization. Among various quantitative verification methods, SMC is one of the well-established verification techniques to verify non-deterministic systems [14]. Since an SoS has a great deal of uncertainty due to its diverse and dynamic nature, SMC can be more proper than other typical verification methods to evaluate and predict the goal achievement quantitatively [3]. For an SoS consisting of black-box CSs, in particular, probabilistic and non-deterministic behaviors should be analyzed in terms of high-level goal achievement. Simulation-based approaches enable engineers to observe emergent behaviors and phenomena caused by collaborations within SoS.

Implementation of Scenario. An SoS is composed of multiple heterogeneous systems from different domains, so it could be valuable if a scenario is implemented based on real-system circumstances. However, the main purpose of this study is to analyze the feasibility of fault localization techniques for SoS engineering. Structure of the scenario used in the experiment is quite simple, but it contains major SoS characteristics in an abstract way. The MCI response SoS consists of autonomous and independent systems that communicate with each other to achieve a higher-level common goal. The diversity of CSs is reflected both in CSs' capability and in CSs' potential for failures. Our simulation model is open-sourced[2], thus not only the collaboration graphs but the component objects can be improved or differently configured.

VIII. Conclusion

Finding the faults for debugging failures of an SoS goal is highly time consuming because of the size of SoS and lack of information about independent CSs. In this paper, to reduce the debugging cost of an SoS, we proposed a fault-localization technique that prioritizes suspicious CSs and interactions of an SoS. It extends a spectrum-based fault localization (SBFL) technique for a collaboration graph of an SoS. It represents an SoS as a collaboration graph, and it can be derived from the limited knowledge of SoS engineers about the SoS collaboration. In addition, it utilizes the quantitative results of SoS statistical verification. In the evaluation, we showed that our technique can localize the suspicious CSs and interactions of an SoS. The localization results of the evaluation scenario reduced the debugging cost of the SoS by an average of 88%. Current evaluation was done with artificial faults on abstract scenarios and models, but we will extend our target model to more complex SoS such as platooning systems with various scenarios for localizing emergent failures in future work.

[2]github.com/SESoS/SIMVA-SoS/tree/localization

Acknowledgements

This research was supported by the Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. 2015-0-00250, (SW Star Lab) Software R&D for Model-based Analysis and Verification of Higher-order Large Complex System), and Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT (2017M3C4A7066212).

References

[1] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: basic concepts, model-based techniques, and research directions," *ACM Computing Surveys (CSUR)*, vol. 48, no. 2, p. 18, 2015.

[2] Y.-M. Baek, J. Song, Y.-J. Shin, S. Park, and D.-H. Bae, "A meta-model for representing system-of-systems ontologies," in *2018 IEEE/ACM 6th International Workshop on Software Engineering for Systems-of-Systems (SESoS)*. IEEE, 2018, pp. 1–7.

[3] D. Seo, D. Shin, Y.-M. Baek, J. Song, W. Yun, J. Kim, E. Jee, and D.-H. Bae, "Modeling and verification for different types of system of systems using prism," in *Proceedings of the 4th International Workshop on Software Engineering for Systems-of-Systems*. ACM, 2016, pp. 12–18.

[4] A. Mignogna, L. Mangeruca, B. Boyer, A. Legay, and A. Arnold, "Sos contract verification using statistical model checking," *arXiv preprint arXiv:1311.3632*, 2013.

[5] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[6] F. Koca, H. Sözer, and R. Abreu, "Spectrum-based fault localization for diagnosing concurrency faults," in *Testing Software and Systems*, H. Yenigün, C. Yilmaz, and A. Ulrich, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 239–254.

[7] E. H. d. S. Alves, L. C. Cordeiro, and E. B. de Lima Filho, "Fault localization in multi-threaded c programs using bounded model checking," in *2015 Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE, 2015, pp. 96–101.

[8] C. Pham, L. Wang, B.-C. Tak, S. Baset, C. Tang, Z. T. Kalbarczyk, and R. K. Iyer, "Failure diagnosis for distributed systems using targeted fault injection." *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 503–516, 2017.

[9] A. B. Sharma, H. Chen, M. Ding, K. Yoshihira, and G. Jiang, "Fault detection and localization in distributed systems using invariant relationships," in *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*. IEEE, 2013, pp. 1–8.

[10] G. Qi, L. Yao, and A. V. Uzunov, "Fault detection and localization in distributed systems using recurrent convolutional neural networks," in *International Conference on Advanced Data Mining and Applications*. Springer, 2017, pp. 33–48.

[11] I. Rish, M. Brodie, S. Ma, N. Odintsova, A. Beygelzimer, G. Grabarnik, and K. Hernandez, "Adaptive diagnosis in distributed systems," *IEEE Transactions on neural networks*, vol. 16, no. 5, pp. 1088–1109, 2005.

[12] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM, 2005, pp. 273–282.

[13] M. Jin, D. Shin, and D.-H. Bae, "Abc+: extended action-benefit-cost modeling with knowledge-based decision-making and interaction model for system of systems simulation," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM, 2018, pp. 1698–1701.

[14] K. Sen, M. Viswanathan, and G. Agha, "Statistical model checking of black-box probabilistic systems," in *Computer Aided Verification*, R. Alur and D. A. Peled, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 202–215.