Evaluating the Suitability of SFL for Self-Adaptive Software

Emil Gruettner\textsuperscript{1} and Hans-Gerhard Gross\textsuperscript{2}
\textsuperscript{1}IT-Designers GmbH and \textsuperscript{2}Esslingen University
Esslingen am Neckar, Germany
\textsuperscript{1}emil.gruettner@it-designers.de
\textsuperscript{2}hans-gerhard.gross@hs-esslingen.de

Abstract

Any non-trivial system will inevitably contain residual defects. Self-adaptive software acknowledges this fact by incorporating fault tolerance mechanisms that can react to occurring problems during operation time. Spectrum-based fault localization (SFL) is a lightweight statistics-based automatic diagnosis approach that works by inferring a diagnosis from component activation and pass/fail observation. As lightweight approach, SFL would provide a benefit to self-adaptive software.

In this paper, we assess the performance of SFL in self-adaptive software simulations. Such software is characterized by long operation and occasional failure observation, leading to extremely unbalanced numbers of pass vs. fail observations. Experiments suggest that SFL is able to perform well with many pass observations and a single fail observation, so that it can, indeed, be used for self-adaptive software, and that components that are used often in combination are more difficult to diagnose by SFL.

1 Introduction

Any non-trivial system will inevitably contain residual defects. Self-adaptive software acknowledges this fact by incorporating fault tolerance mechanisms that can react to occurring problems during operation time. This can be achieved through constantly maintaining internal health information of its constituent parts, and isolating the root cause of a failure observation, in case the system health decreases. Once a potential error state is isolated, and the faulty component(s) identified, the system can unleash its full range of inbuilt self-protection, -adaptation, -reconfiguration, -optimization, and -recovering strategies to resume its normal operation [Gonzalez-Sanchez et al., 2011]. The activation of the system is expressed in terms of a binary activity matrix representing for each component whether it has been involved in a particular system transaction. The pass/fail information is expressed in terms of a binary output vector, and produced by an orcale. A diagnosis is determined by calculating the similarity between each component’s activation vector and the output vector. A component whose activity vector (determined through the various system transactions) is more similar to the output vector (produced by the oracle) is more likely faulty than other components, and is, therefore, ranked higher to be the prime suspect. This is similar to what a human diagnostician would do, i.e. rule out those parts of the system that cannot be used to explain a particular error observation, because they have not taken part in producing the erroneous outcome.

SFL has been applied successfully in highly dynamic service-based systems [Chen et al., 2012], and also been tried in self-adaptive systems [Gonzalez-Sanchez et al., 2010]. In particular, this last class of systems would benefit from a lightweight diagnosis implementation such as SFL, due to its ultra-low computational overhead. However, the characteristics of self-adaptive systems are still presenting considerable challenges for SFL.

Even though we can expect residual defects in any deployed system, their activation over time will be likely rare. Even if only a single failure is observed after a long operation time, ideally, the fault-tolerance mechanism should be able to diagnose it and take immediate measures. A related problem comes from the long operation times in the systems under consideration. Activity matrices need to be reset from time to time in order to "forget" older potentially irrelevant observations.

Our main interest as preliminary work is to assess the performance of SFL under such difficult circumstances, and from our considerations, we can derive the following concrete research questions to be addressed in this paper.

**RQ1:** How many pass versus fail observations are required in order to calculate an acceptable diagnosis? We look at the specific case of one single fail observation and a large number of pass observations. This single fail observation is the erroneous state after a long operation time on which the self-healing strategies of the system must react.

**RQ2:** How many overall observations should be considered in the diagnosis when a failure occurs. How can this be determined (dynamically) for a given system? This is
RQ3: To which extent does the topology of the system, the organization of the components and their interactions, affect these two other considerations? We realize that the formation of main execution paths through a system may have an effect on diagnosability.

The remainder of this paper is structured as follows. Section 2 briefly introduces SFL and its application in self-adaptive software, and provides the scope of the work. Sect. 3 introduces an SFL simulator and details the simulation setup used for our experiments. Sect. 4 presents the simulation results. Sect. 5 discusses the experimental results and the limitations of the approach. Related work is presented in Sect. 6. Finally, Sect. 7 summarises and concludes the paper and provides a future work outlook.

2 Background and Approach

2.1 Spectrum-based Fault Localization

SFL calculates a diagnosis from observations about component involvement in system transactions and pass/fail outcome of a transaction [Gonzalez-Sanchez et al., 2011]. Component involvement is expressed in terms of block-hit spectra, representing which component was activated in a transaction [Reps et al., 1997; Zoeteweij et al., 2007]. Execution of system transactions adds spectra to a so-called activity matrix, and a binary verdict (pass=0, fail=1) to the output vector. The diagnosis is calculated by applying a similarity coefficient (SC) to each component’s activation vector and the output vector. It denotes the likelihood of a component to be the faulty one, and determines the position of the component in the diagnosis ranking. Table 1 illustrates this process with a simple system comprised of 10 components, with C3 being the faulty one. The activity matrix denotes which components have been activated per transaction \(t_1 - t_6\), and the output vector indicates a pass(=0) or a fail(=1). A transaction corresponds to a single test execution of the system. The similarity (\(SC_{t6}\)) between each component activation vector and the output vector determines the ranking of a component in the diagnosis. In this case, component C3 is correctly ranked as the prime suspect, indicated through the 100% similarity between C3’s activation vector, and the output vector. Any binary similarity coefficient may be used in order to produce a diagnosis, however, Ochiai (\(SC_{t6}\)) was found to work best in experiments [Abreu et al., 2006].

<table>
<thead>
<tr>
<th>Cmp</th>
<th>Character counter</th>
<th>(t_1)</th>
<th>(t_2)</th>
<th>(t_3)</th>
<th>(t_4)</th>
<th>(t_5)</th>
<th>(t_6)</th>
<th>SC_{t6}</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>def count(string)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.82</td>
</tr>
<tr>
<td>C1</td>
<td>string.each_char</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.82</td>
</tr>
<tr>
<td>C2</td>
<td>if c==[A-ZaZ]/</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.89</td>
</tr>
<tr>
<td>C3</td>
<td>let += 2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.00</td>
</tr>
<tr>
<td>C4</td>
<td>elseif c==[0-9]j/</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.89</td>
</tr>
<tr>
<td>C5</td>
<td>let += 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.89</td>
</tr>
<tr>
<td>C6</td>
<td>dig += 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.71</td>
</tr>
<tr>
<td>C7</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.71</td>
</tr>
<tr>
<td>C8</td>
<td>elseif not c==[A-ZaZ]-[0-9]j/</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.58</td>
</tr>
<tr>
<td>C9</td>
<td>other = 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.58</td>
</tr>
<tr>
<td>C10</td>
<td>return let, dig, other</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.82</td>
</tr>
</tbody>
</table>

Output vector (verdicts) 1 1 1 1 0 0

2.2 SFL for Self-Adaptive Software

Applying SFL in a self-adaptive software context brings up a number of issues [Gonzalez-Sanchez et al., 2010; Piel et al., 2012]. First, the identification of a (complete) transaction is not obvious, since input stimuli are continuously arriving and the system responds accordingly, changing its internal state and producing output. For SFL to work, it requires a clear separation of system transactions, i.e. separation of the spectra. This can be addressed through adequate service execution platforms that handle unique transaction IDs, as demonstrated in [Chen et al., 2012].

Second, the size of the activity matrix must be confined. When applied offline, SFL is based on a number of test transactions with a limited number of observations, leading to a limited number of spectra in the activity matrix. However, when applied online, in a constantly operating system, activity matrices are likely to grow almost indefinitely in length. Eventually, older observations need to be dropped, in order to calculate the similarity coefficients for recent spectra. Older observations from a distant past may have much weaker relevance for a present diagnosis than more recent observations.

Third, problems are aggravated, if we assume that diagnosis must commence already after a single failure. In SFL, this leads to activity matrices, that, although confined to a limited number of past observations, will comprise a large number of successful system transaction, and small number of failed system transactions. In the extreme case, we will end up with one single failed transaction, i.e. the last transaction that initiates self-healing activities. Because SFL is based on statistics, our hypothesis is that this may present a challenge for the calculation of diagnoses.

2.3 Scope of the Work

In order to address our research questions, we perform a number of different system simulations with various system topologies. That way, we can gather a statistically significant amount of data on the performance of SFL under various conditions.

Following the self-adaptive software requirement, we assume a long running system with a single fault, i.e. in each experiment only one single component is set to be faulty. We vary the health of components to achieve different lengths of observation histories in order to analyze the performance of SFL and address our first research question. That way, we can assess whether SFL will still come up with useful diagnoses, even when the number of pass vs. fail observations is extremely unbalanced.

Turning to our second research question, we analyze the SFL performance while we limit the observation history with the concept of a so-called sliding window. This technique allows to add more recent system transactions to the activity matrix while older observations can be deleted [Piel et al., 2012].

Finally, to address our third research question, we vary some characteristics of the topologies used in the simulations, and then analyze to which extent the performance of SFL reacts on such changes in topology.

3 Experimental Setup

3.1 SFL Simulator

All experiments were performed with our SFL Simulator [Chen et al., 2013a][1]. It provides functions for setting

up component topologies, executing the topologies thereby gathering coverage information, and calculating diagnoses.

The topology of a system represents the system organization, i.e. the components used in the system and how they are linked to each other. A relevant property of the component links is link probability. In real code, this is implicitly determined through the business logic, but in the simulator it may be described in terms of a likelihood that a component will invoke another subsequently connected component. A link probability of 1.0 means that two components will be always invoked together, leading to tight coupling. Hence, link probability leads to a filtering effect such that some components are invoked more in combination than others, resulting in distinct system execution paths.

In the simulator, each component is defined by its name, health, and failure probability. Health denotes the probability that a component will produce an error when it is activated, i.e. health = 1.0 indicates a completely healthy component. Health < 1.0 denotes an erroneous component, with some probability that it will produce an error when it is invoked. Health = 0.0 denotes a component that will always fail when activated. That way the faulty component in the topology can be determined, plus its fault intermittency. Intermittency represents the likelihood that the faulty component will actually produce an error when it is invoked, i.e. if set to health = 0.5, the faulty component will issue an error in 50% of its invocations.

In contrast, failure probability determines the likelihood of a component to issue a failure immediately when a previous error is propagated to it, or its own fault is activated. Failure probability = 1.0 indicates a faulty component will always produce a failure observation when its fault was activated. Once a fault is activated it is checked in every subsequent component invocation whether or not it leads to a failure observation. If a failure is detected, the execution is stopped. If all component failure probabilities are set to 0.0, the error is detected at the end of an execution.

Based on the topology with components and invocation links, the simulator can be controlled to perform executions. This requires that one or several entry points (components or links) are activated. Every activation of the topology leads to a particular control flow according to the initially defined probabilities, thereby generating component activations to be stored in the activity matrix and pass/fail information to be stored in the output vector. These observations are collected and used to calculate a diagnosis ranking.

### 3.2 Simulation Setup

We performed simulations with various topologies with different number of components. In the interest of space, we only present one topology, \( T1 \), shown in Figure 1. For presentation purposes, we ordered the topology according to each component’s shortest distance from the entry point, which is always \( C0 \). Topology \( T1 \) consists of 13 components, with 27 invocation links between them. All topologies used are adjusted such that tight interaction coupling between components is limited. Tight component coupling was found to inhibit the overall performance of SFL considerably (refer to [Chen et al., 2013b]) and we are not specifically considering this property. However, we keep it in mind and perform some experiments with link probabilities in some topologies set to high values, e.g. 0.6, 0.8 or 1.0.

All simulations performed assume a single-fault setting in the system, i.e. only the health value of one component is set to \( H < 1.0 \) for any simulation run. The health of other components is set to \( H = 1.0 \), i.e. completely healthy. Failure probability is set to a constant value of \( F = 1.0 \), meaning that if a fault is covered, it leads to instant failure observation. Although the impact of low error propagation is not the aim of this paper, we performed simulations with different failure probabilities \( 0.0 < F < 1.0 \) as well. We can state that for our scenario there is no remarkable impact on the performance of SFL.

We acknowledge the fact that assuming only a single fault, and assuming high failure probability is unrealistic for real systems. First, most systems will have more than one fault, however, we consider the special case that a system runs for a long time, then fails, and then is expected to come up with a diagnosis based on this single fail observation and a potentially high number of pass observations. Second, high failure probability is enforced in order to scrutinize only the effect of an unbalanced number of fail vs. pass observations. Even though, these assumptions may be unrealistic, for our initial experiments we would like to confine the number of free variables to a minimum. In future work, we will look at how combinations of these properties affect statistics-based diagnosis.

**Simulation Assessment.** In order to analyze how SFL performs during the experiments, we look at the absolute values of the similarity coefficients \( SC_o \), and determine the success rates of the diagnoses as follows:

- **Distinct positive diagnosis:** the faulty component was determined as the distinct faulty one, i.e. the faulty component is correctly and uniquely diagnosed by the highest \( SC_o \).

- **Positive diagnosis:** the faulty component was determined as the faulty one with other components ranked the same, i.e. the faulty component is among the highest ranked \( SC_o \), but is not uniquely diagnosed.

- **Negative diagnosis:** the faulty component is not diagnosed as the faulty one, i.e. the faulty component has a lower
to calculate the length of the observation history, and, hence, simulate a sliding window for SFL as introduced and described in [Piel et al., 2011] considers only the spectra of a given history length and discards older ones. We experiment with differently sized sliding windows, i.e. all past observations, and the window sizes set to 50, 20, and 10 past observations.

RQ1: Pass versus Fail Observations. The goal of our simulations is to gather system execution data with activity matrices, error vectors, and verdicts, leading to only one single failed transaction per simulation run, i.e. the execution is stopped after a failure is observed. That way, we can mimic a real system which is running for some time successfully, and then produces a single fail observation. This corresponds to the aim of our original research questions.

In order to create a significant number of experiments with differently sized activity matrices, the health of components $C5 - C12$ in topology $T1$ is altered between $0.0 < H < 1.0$. We do not change the health values of components $C1 - C4$. They are directly connected to the entrance point, $C0$, and act as relay for stimulating the rest of the system.

RQ1 and RQ2: Total Number of Observations. The simulation setup from above is modified to change the length of the observation history, and, hence, simulate a sliding window for confining the activity matrices that are used to calculate the $S_{C_o}$. A sliding window for SFL as introduced and described in [Piel et al., 2011] considers only the spectra of a given history length and discards older ones. We experiment with differently sized sliding windows, i.e. all past observations, and the window sizes set to 50, 20, and 10 past observations.

RQ3: Topology of the System. Using the general component setup shown in Fig. 1, the invocation link probability between the components was additionally altered in steps $L = (0.20, 0.40, 0.60, 0.80, 1.00)$. We set all link probabilities to the same value to eliminate effects of primary invocation paths. These are paths through the system that are likely to be taken much more often, and are often referred to as nominal system execution paths. This property, that most systems exhibit, introduces an interesting limiting factor to statistics-based diagnosis approaches, since it introduces tight component coupling. We will, therefore, consider this as separate research topic in the future.

4 Simulation Results

In the interest of space, we present the simulation results for component $C12$, as the simulation outcomes of components $C5 - C12$ are similar. As already stated, components $C1 - C4$ are not considered in the simulation results.

First, we take a look at the $S_{C_o}$ for different number of pass observations of components $C5 - C12$ (Figure 2) and assess how many pass versus one fail observations are sufficient for SFL to perform a diagnosis. Then, we assess the performance of SFL for different number of pass observations plus changing link probabilities (Figure 3). The effect of a sliding window on the performance of SFL is discussed (Figures 4), and we show how different topologies (adjustments in link probability) affect the diagnoses (Figures 5, 6, 7, and 8).

Since SFL performance is best expressed by distinct positive diagnosis we focus on this property.

RQ1: Pass versus Fail Observations. Figure 2 indicates how the $S_{C_o}$ of distinct positive diagnoses decrease with increasing number of pass observations. We show a topology with loosely coupled components (link probability set to $L = 0.2$). It is important to note that we only consider one fail observation, i.e. the last one that stopped the simulation. The mean $S_{C_o}$ values of components $C5 - C12$, starting with $S_{C_o} = 1.0$ at 1 pass observation decrease asymptotically to zero with a longer observation history. This is to be expected. However, there are two fundamental issues.

- As long as SFL can pinpoint the faulty component correctly long observation history is not a problem. However, eventually, after very long running time, the $S_{C_o}$ will converge to zero and not be useful. At a certain point in time, the observation history must be limited.

- Comparing the topology $T1$ (Fig. 1) with the resulting $S_{C_o}$ hints to a relation. Components $C8, C11, and C12$ are three links away from the entry point, $C5, C6, C7, C9$, and $C10$, are two links away. A filtering effect in execution results in fewer activations of the first group of components, resulting in fewer ‘1’s in the respective activity matrices for these components. This property of systems must be taken into consideration when defining the limitation of observation history.
RQ1 and RQ2: Total Number of Observations. Figure 3 shows the performance of SFL for component C12 set to be faulty (with different link probabilities). With a low number of pass observations (history length), the performance is poor due to low variety in the activity matrices, but increases with more pass observations, i.e. for loosely coupled components (L=0.2; L=0.4). More tightly coupled components cannot be successfully diagnosed, even if the number of pass observations is increased for high link probabilities (L=0.6; L=0.8; L=1.0; compare to [Chen et al., 2013b]).

In general, with longer observation histories, the performance of SFL increases, because the variety in the activity matrices is growing. SFL can indeed diagnose faulty components correctly, even with only one single fail observation and a large number of pass observations. This goes against our initial expectation, when we thought that several fail observations would be necessary to pinpoint the root cause of a defect. For our experiments with the example topology T1 we can determine that at around 100 pass observations vs one fail observation are sufficient in order to come up with good diagnoses, i.e. if the system components are not tightly coupled. In other simulated topologies we determined different pass observations vs one fail observation ratios and therefore, we consider this as separate research topic in the future.

In addition to looking at the total number of pass vs. fail observations, we modified the observation history length with a simulated sliding window. That way, we can assess the impact of such a limiting technique on the performance of SFL. Earlier, we included all observations in the diagnosis and determined the minimal number of past observation points required. Now, we use only the N most recent observations in order to calculate a diagnosis, thereby "forgetting" older observations.

Figure 4 shows the SFL performance for component C12 (low coupling, L = 0.2) with three simulated sliding windows of sizes 50, 20, and 10 past observations. It is interesting to see how the success rate of SFL increases with increasing number of pass observations. At the respective points when older observations are deleted, the success rates stabilize at that point or drop slightly and then move up again. The results shown clearly indicate that short sliding windows are detrimental for SFL, and that longer sliding windows result in better diagnoses. The window size should therefore be chosen to be as long as possible up to the point when the SC, become too small to be discriminable.

We cannot explain this intermediate drop of diagnostic performance at 100 in Fig. 4 that we observed in many experiments with the sliding window technique. This demands further investigation in future work.

RQ3: Topology of the System. Figure 3 already indicates how topology, determined through various invocation link probabilities, can affect the diagnostic performance of SFL. The most successful diagnoses can only be achieved with loosely coupled components, i.e. low link probabilities (L = 0.2; L = 0.4). More (and more concrete) results are presented in Figs 5 and 6, illustrating the diagnostic performance in percent for topologies with L = 0.2 and L = 0.6, respectively. The results in both figures clearly indicate that the link probability directly affects the diagnostic performance of SFL. Higher link probability, representing tighter invocation coupling of components, inhibits diagnosis (Fig 6), whereas lower link probability, representing loose coupling, facilitates diagnosis (Fig. 5).

Higher invocation link probability leads to more components being activated in combination, resulting in less diversity in the respective activity matrices and, hence, in a poorer SFL performance. This can be circumvented, to a certain extent, by increasing the number of observations used for the diagnosis, i.e. both figures show an increase in overall SFL performance for longer observation histories, and it can be explained by an overall increase in diversity of observations in bigger (longer) activity matrices.

However, this is not the case for components that take part in a nominal (main) execution path through the system. A nominal execution path is characterized through the fact that its components are taking part in most system transactions, and are, therefore, more tightly coupled than other components taking part in alternative execution paths. Components that take part in a system’s nominal execution path are, therefore, much more difficult to diagnose by statistics-based techniques, than components taking part in other, alternative execution paths. This is an interesting observation that will be scrutinized in future work.

Another interesting result comes from the fault intermittency of the faulty components. In the simulator, this can be controlled by the health value. Health set to H = 0.0 leads to immediate fault activation once the faulty component is executed. Values 0.0 < H < 1.0 denote the probability...
that the fault will be triggered when the faulty component is activated.

We varied the health values of the faulty components in order to assess the impact on the SFL performance. Figure 7 show the SFL performance plotted against the number of observations used, for low link probability ($L = 0.2$) and different health values (for example $C_{12}$). Figure 8 shows the SFL performance plotted against the number of observations used, for high link probability ($L = 0.6$) and different health values (for example $C_{12}$). It is important to keep in mind that we still consider a large number of pass observations and one single fail observation. SFL is robust w.r.t. high intermittency combined with low component coupling. Figure 7 illustrates that the overall performance of SFL increases with increasing number of observations, regardless of intermittency. On the other hand, SFL reacts sensibly w.r.t. high intermittency combined with high component coupling. Figure 8 illustrates the low performance of SFL in case of high intermittency and high coupling.

5 Discussion

The experiments performed for this work lead us to a number of interesting observations and outcomes that we summarize and discuss in the following paragraphs.

From our experiments, we assume that SFL is a suitable automated diagnosis technique to be used in the self-adaptive software context. We analysed the performance of SFL w.r.t. increasing number of pass observations and a single fail observation. With a very low number of observations, the SFL performance shown in Fig. 3 is low, i.e. it starts at almost 0% with 1 observation. Even though, the number of pass observations may become large for a constantly running system, SFL performs well as long as the $SC_p$ provide sufficient discriminative power in order to rank the components. Eventually, the size of the observation history must be limited through a sliding window technique. The overall outcome is that SFL performs better with more spectra (observations) due to the introduction of more discriminative information into the activity matrices used for the diagnosis. Our experiments suggest that for the simulated topology $T_1$ used in our work, more than 100 observation points should be used, and can be used successfully, even if only one fail observation is available. In general, a minimum number of observations is important for the performance of SFL. Coming up with a general ratio for any topology demands further investigation in future work.

Since we cannot include an arbitrary number of observations in the diagnosis after a very long running time of a system, past observations must be deleted, eventually. A sliding window represents an adequate and easy to realize technique in order to limit the overall number of observations to be used in a diagnosis when a failure occurs. In fact, limiting the number of past observations is a prerequisite in self-adaptive software. Since the concrete value depends highly on the individual topology of the system, we cannot denote any definitive figure as to how big this window should be. However, from our experiments we can deduce the SFL performance improves with more observations, up to the point when the $SC_o$ approaches zero and proper diagnosis becomes infeasible.

The experiments performed for this work, again, acknowledge our earlier work in which we found that system topology plays a major role in the performance of SFL [Chen et al., 2013b]. Invocation link decoupling between components can be considered a primary facilitator for applying statics-based diagnosis techniques such as SFL in the self-adaptive software context. An interesting issue arises through analyzing primary (nominal) and secondary (exceptional) system execution paths. The components in a nominal system execution path are, by definition, tightly coupled, and hence, activated in combination. This cannot be avoided and it presents a major challenge for automated diagnosis.
Components taking part in secondary system paths are much more straight-forward to diagnose that those on the primary paths. On the other hand, can we assume, that components on the main paths are less error prone, since they have been used and tested more than other components? This is an interesting pointer for future work.

6 Related Work

Fault diagnosis represents an essential function in self-adaptive software. It ensures that only the necessary parts of a system are treated in case of occurring failures. Our earlier work demonstrates how statistics-based diagnosis may be incorporated in self-adaptive systems in general [Piel et al., 2011][Piel et al., 2012]. Other authors report how model-based diagnosis can help pinpoint problematic components [Casanova et al., 2011][Mohamed et al., 2012].

Other related work is aimed at infrastructures for self-monitoring such as [Seltzer and Small, 1997] or [Chen, 2006].

7 Summary, Conclusions and Future Work

Self-adaptive systems can be characterized by long system operation times with rare failure occurrences, leading to potentially huge activity matrices containing mainly pass observations and a single fail observation. The fail observation triggers the self healing activities, one of which is the identification of the failure’s root cause, i.e. the diagnosis. Referring to our original research questions, we looked at:

RQ1: How many pass observations are required in order to calculate an acceptable diagnosis for long running systems? We used a simulator and varied the health of components in order to simulate long running systems. After the first failure occurred we applied SFL and analyzed how SFL performs w.r.t. different lengths of observation history. We found, that SFL is suitable to perform in long running systems with the need for an observation history limitation as well as SFL requires a minimum number of observations for an acceptable performance.

RQ2: How many observations should be considered in the diagnosis when a failure occurs? We implemented a sliding window technique to limit the observation history SFL. This is a necessary requirement for long running systems. We found it imposes no SFL performance penalty, as long as the window is big enough.

RQ3: To which extent does the topology of the system affect these two other considerations? We found, that tight interaction coupling (main invocation paths included) inhibits the performance of SFL, and that lower intermittency may improve the diagnoses, nevertheless. SFL does not perform well where the system invokes similar invocation patterns.

The goal of the work performed for this paper was to get an initial feeling for the problems in self-adaptive software that might inhibit the application of a light-weight, statistics-based, automatic diagnosis technique. We acknowledge the the preliminary nature of the simulations and the results obtained from them. In the future we will repeat these experiments with a real case study system. We will also consider error propagation in our experiments and look at how to deal with components on primary execution paths, which are much more difficult to pinpoint. Furthermore, we will look at the SFL performance where the system invokes similar invocation patterns.

References


