The Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23)

Distributed Spectrum-Based Fault Localization

Avraham Natan, Roni Stern, Meir Kalech

Ben-Gurion University of the Negev, Israel
avinat123@gmail.com, roni.stern@gmail.com, kalech@bgu.ac.il

Abstract

Spectrum-Based Fault Localization (SFL) is a popular approach for diagnosing faulty systems. SFL algorithms are inherently centralized, where observations are collected and analyzed by a single diagnoser. Applying SFL to diagnose distributed systems is challenging, especially when communication is costly and there are privacy concerns. We propose two SFL-based algorithms that are designed for distributed systems: one for diagnosing a single faulty component and one for diagnosing multiple faults. We analyze these algorithms theoretically and empirically. Our analysis shows that the distributed SFL algorithms we developed output identical diagnoses to centralized SFL while preserving privacy.

1 Introduction

Spectrum-Based Fault Localization (SFL) is a popular approach for diagnosing faulty systems. Primarily used in software diagnosis (Abreu, Zoeteweij, and Van Gemund 2009), SFL aims to identify faults in programs. To achieve this, SFL models the program as a set of components, and runs tests, each of which executes some components. This activity is recorded into activity matrix (spectrum), and the tests outcomes are recorded in an error vector. Some SFL algorithms diagnose single faults, while others diagnose multiple faults.

SFL algorithms are centralized. However, different systems are inherently distributed such as software systems, multi-agent systems, communication networks etc. Applying centralized SFL techniques to diagnose such distributed systems is challenging, especially when communication is costly and the privacy should be considered: First, information on activity of a component may be hard to get or restricted. Second, privacy restrictions or communication load might challenge the performance of a centralized algorithm.

The contribution of this paper is: (1) We define the problem of Distributed SFL (DSFL). (2) We propose two SFL-based diagnosis algorithms for distributed systems: one for single faults (DSFLA-SINGLE) and one for multiple faults (DSFLA-MULTI). (3) We provide empirical and theoretical analysis of soundness, completeness, privacy, communication load and runtime. We show that the distributed algorithms output identical diagnoses to the centralized while preserving privacy. We also evaluate three variations of DSFLA-MULTI, and show that with a small reduction in diagnosis quality, DSFLA-MULTI performs better in terms of privacy, communication load and runtime.

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

2 Background and Related Work

Our work builds on prior work on SFL. Thus, we provide here a brief background on SFL. For a more comprehensive background see Abreu, Zoeteweij, and Van Gemund 2009.

Definition 1 (SFL Problem)**.** *An SFL problem is defined by a tuple $\langle C, R, S, E\rangle$ where C is a set of components, each of which may be faulty; R is a set of runs executed on the system; S is a binary $|R| \times |C|$ matrix where $S_{i,j} = 1$ denotes that component c_j participated in run r_i; and E is a vector of length $|R|$ where $E_i = 1$ denotes that r_i failed, and $E_i = 0$ otherwise. An SFL problem arises when $\exists i : E_i = 1$.*

The matrix S is called *spectrum* and the vector E is called *error vector*. Table 1a shows the spectrum and error vector for a system with 3 components (c_1, c_2, c_3) that is executed 4 (r_1, r_2, r_3, r_4) times. Consider the second row, for example, components c_2 and c_3 were involved, and the run failed.

A solution to an SFL problem is a set of *diagnoses* and a *ranking function* to rank them. A diagnosis is a set of components that *explain* all the failed runs. Diagnosis Δ explains a failed run if at least one of the components in Δ participated in the failed run according to the spectrum S. Formally:

Definition 2 (SFL Diagnosis)**.** *A set of components $\Delta \subseteq \{1, .., |C|\}$ is diagnosis for an SFL problem $\langle C, R, S, E\rangle$ if $\forall i : E_i = 1 \rightarrow \exists j : j \in \Delta \wedge S_{i,j} = 1$.*

Some SFL algorithms diagnose single faults, while others diagnose multiple faults.

Single fault SFL: In this case, a component j is a diagnosis if it participated in at least one failed run. Diagnosis ranking uses **Similarly Coefficients** (Hofer et al. 2015), which are evaluated using four **Similarity Counters** $n_{pq}(j)$, $p, q \in \{0, 1\}$ defined as: $\forall c_j, n_{pq}(j) = |\{i | S_{ij} = p \wedge E_i = q\}|$.

Table 1b shows these counters with respect to Table 1a. For example, $n_{11}(2)=2$ since the number of runs where $S_{i,2}=1 \wedge E_i=1$ is 2. Using **Ochiai** (Abreu, Zoeteweij, and Van Gemund 2007) similarity coefficient with those values yields the probabilities: $P(c_1)=0.67$, $P(c_2)=0.82$, $P(c_3) = 0.41$. The diagnosis set ranked by their probabilities in that case is: $D = \{\langle\{c_2\}, 0.82\rangle, \langle\{c_1\}, 0.67\rangle, \langle\{c_3\}, 0.41\rangle\}$.

	c_1	c_2	c_3	E
r_1	1	1	0	1
r_2	0	1	1	1
r_3	1	0	0	1
r_4	1	0	1	0

(a)

	c_1	c_2	c_3
$n_{11}(j)$	2	2	1
$n_{10}(j)$	1	0	1
$n_{01}(j)$	1	1	2
$n_{00}(j)$	0	1	0

(b)

Table 1: (a) Spectrum and Error Vector for system of 3 components that is run 4 times, and (b) their similarity counters.

SFL for multiple faults: Barinel (Abreu, Zoeteweij, and Van Gemund 2009) is an SFL-based algorithm that addresses this case by combining SFL with model-based diagnosis (MBD). Barinel defines a set of components that participated in a failed run as a *a conflict*. Then it outputs diagnoses by computing a Minimal Hitting Set (MHS) of the set of conflicts. This approach is known in MBD as the *conflict-directed* approach (De Kleer and Williams 1987; Williams and Ragno 2007; De Kleer 2011).

Diagnosis ranking uses a Bayesian approach, by computing $P(\Delta|E)=\frac{P(E|\Delta)\cdot P(\Delta)}{P(E)}$, where $P(\Delta)$ is the prior of Δ and $P(E)$ is a normalization factor. To compute the **Likelihood Function** $P(E|\Delta)$, h_j is defined as the likelihood of faulty component to behave normally, formally:

Definition 3 (Likelihood Function)**.** *Given a diagnosis Δ, spectrum S and error vector E, the **Likelihood Function** L is defined as: $L = P(E|\Delta) = \prod_{E_i \in E} L_i$. With each term L_i defined for a row in the spectrum as the probability of all involved components to behave normally:*

$$L_i = P(E_i|\Delta) = \begin{cases} \prod_{j \in \Delta \wedge S_{ij}=1} h_j & \text{if } E_i = 0 \\ 1 - \prod_{j \in \Delta \wedge S_{ij}=1} h_j & \text{if } E_i = 1 \end{cases} \quad (1)$$

Since h_j is not known, a maximization algorithm such as gradient descent is applied to maximize L.

Example 1. *In Table 1a the conflicts for runs r_1, r_2, r_3 are $\{c_1, c_2\}, \{c_2, c_3\}, \{c_1\}$ respectively. The minimal hitting sets $\Delta_1 = \{c_1, c_2\}$ and $\Delta_2 = \{c_1, c_3\}$ are the diagnoses. As an example, for $\Delta_2 = \{1, 3\}$ the terms corresponding to the rows are:*

$$L_1 = P(E_1|\Delta_2) = (1 - h_1)$$
$$L_2 = P(E_2|\Delta_2) = (1 - h_3)$$
$$L_3 = P(E_3|\Delta_2) = (1 - h_1)$$
$$L_4 = P(E_4|\Delta_2) = (h_1 \cdot h_3)$$

And their product is:

$$L = (1 - h_1) \cdot (1 - h_3) \cdot (1 - h_1) \cdot (h_1 \cdot h_3) \quad (2)$$

Maximizing L with gradient descent yields $L = 0.161$.

2.1 Related Work

SFL has been studied in recent years. Different works improve and demonstrate its usefulness. One work addresses scalability challenges that rise due to the computational overhead when computing the spectrum (Perez, Abreu, and Riboira 2014). They propose dynamic code coverage (DCC) which dynamically increases the granularity of the components. Results show the usefulness of this approach.

A different improvement to SFL incorporates software fault prediction model (Elmishali, Stern, and Kalech 2016) to improve diagnosis ranking. Results show significant improve in diagnosis accuracy and troubleshooting efficiency.

Another work (Elmishali, Stern, and Kalech 2018) shows how SFL can be used in a novel Learn, Diagnose and Plan (LDP) paradigm as the diagnosis part. Results show how this improves SFL when it is part of LDP.

Later study shows SFL usage in diagnosing system exploits (Elmishali, Stern, and Kalech 2020). The components are blocks of code and a trace of tests using them is recorded. They use fuzz testing tool and under-tracing technique to enhance this method, and improve diagnosis accuracy. The method's validity is shown on different software projects.

Another work demonstrates SFL for Multi-Agent Systems (Passos, Abreu, and Rossetti 2015). The authors extend SFL to address agent specific features such as agent autonomy. They show prominent results of roughly 96.96% diagnosis accuracy.

Some work addresses distributed diagnosis with different approaches such as inter-level communication (Pérez-Zuñiga et al. 2018), fuzzy fault isolation (Syfert, Bartyś, and Kościelny 2018) and structural analysis (Perez-Zuniga et al. 2022). Distributed approach to SFL was not proposed previously.

3 Methodology

In this section we address the problem of Distributed Spectrum-Based Fault Localization (DSFL).

3.1 Problem Definition

The presented approaches rely on a central solver to generate and rank the diagnoses. Applying them on distributed systems raises challenges such as single point of failure, communication load, and privacy. To address the latter two, we assume no central solver. Instead, every component has vision of the execution runs in which it participates, denoted as **Local Spectrum** and **Local Error Vector**. Formally:

Definition 4 (Local Spectrum and Local Error Vector). *Given spectrum S, Error Vector E and component c_j, the Local Spectrum S^j of c_j is: $S^j = \{S_{i,*} | S_{i,j} = 1\}$, and the Local Error Vector E^j of c_j is: $E^j = \{E_i | S_{i,j} = 1\}$.*

	c_1	c_2	c_3	E
r_1	1	1	0	1
r_2	-	-	-	-
r_3	1	0	0	1
r_4	1	0	1	0

(a)

	c_1	c_2	c_3	E
r_1	1	1	0	1
r_2	0	1	1	1
r_3	-	-	-	-
r_4	-	-	-	-

(b)

	c_1	c_2	c_3	E
r_1	-	-	-	-
r_2	0	1	1	1
r_3	-	-	-	-
r_4	1	0	1	0

(c)

Table 2: Local spectra S^1, S^2, S^3 and error vectors E^1, E^2, E^3 corresponding to components c_1, c_2, c_3.

Tables 2a, 2b, 2c show the local spectra and error vectors of components c_1, c_2, c_3. Solving the SFL problem using one spectrum may output wrong diagnoses. To that end we define the **Distributed SFL (DSFL) problem**:

Definition 5 (DSFL problem). *A* DSFL *problem is defined as* $\langle C, R, \{S^j\}_{j=1}^{|C|}, \{E^j\}_{j=1}^{|C|} \rangle$*, where* C *is a set of components,* R *is a set of runs, and* S^j *and* E^j *are the local spectrum and error vector of component* c_j*. A* DSFL *problem arises when* $\exists j, i : E_i^j = 1$.

A solution to DSFL is a diagnosis for the joint SFL problem defined by the union of the local spectras and local error vectors. A naive solution collects the information from the components to a single solver. Such approach may have high communication costs, and neglects privacy. We discuss the topic of privacy loss in distributed SFL in Section 4.

In the following sections we describe distributed versions of the SFL algorithms that address privacy for single fault and for multiple faults. They share information in order to reach similar output to the centralized algorithms, while minimizing the private information revealed.

Algorithm 1: DSFLA-SINGLE

Input: c_j - the current component
Result: P - probability of diagnosis $\Delta = \{c_j\}$

1 $n_{11}(j),\ n_{10}(j),\ n_{01}(j),\ n_{00}(j) \leftarrow 0, 0, 0, 0$
2 **for** $i \in S^j$ **do**
3 $\quad q \leftarrow E_i^j, n_{1q}(j) \leftarrow n_{1q}(j) + 1$
4 **for** $j' \in [1, ..., |C|]\ s.t.\ j' \neq j$ **do**
5 $\quad nn_{01},\ nn_{00} \leftarrow request\_missing(c_j, c_{j'})$
6 $\quad n_{01}(j), n_{00}(j) \leftarrow n_{01}(j) + nn_{01}, n_{00}(j) + nn_{00}$
7 $P \leftarrow similarity(n_{11}(j),\ n_{10}(j),\ n_{01}(j),\ n_{00}(j))$
8 **return** *P*

3.2 Distributed SFL for Single Fault Problems

Here we present our distributed algorithm for diagnosing single faults (DSFLA-SINGLE). Since the algorithm handles single faults, each component is a potential diagnosis and we only need to present the ranking. Each component c_j should calculate its similarity counters $n_{11}(j),\ n_{10}(j),\ n_{01}(j),\ n_{00}(j)$ and use them with a similarity coefficient. The challenge is that a component knows only the outcomes of runs it participates in, as shown in Table 2, making it able to calculate only $n_{11}(j),\ n_{10}(j)$. To address this challenge, each component c_j requests from other components information about $n_{01}(j)$ and $n_{00}(j)$ as indicated by their spectrum. This information is summed up by c_j to the true values of $n_{01}(j)$ and $n_{00}(j)$.

Algorithm 1 presents the process of ranking component c_j. The component initializes $n_{pq}(j)$ (line 1), and calculates $n_{1q}(j)$ using the rows in its local spectrum (Lines 2-3). Then, the component requests information from other components (Lines 4-6), by calling the *request_missing* procedure (The details of this procedure are listed in Algorithm 2). Note that these request messages are sent to other components by order of their appearance in the spectrum. To allow this, we assume the components to have a predefined order. This allows a component to avoid sending data it knows was already sent by previous components. Finally, a similarity coefficient is used to calculate the rank. Algorithm 2 describes the *request_missing* method. Component $c_{j'}$ provides the data requested by c_j. For every row in its local spectrum, $c_{j'}$ checks if c_j has already received the information about this row from a previous component, or c_j has this row in its local spectrum (Line 3). If this is not the case, $c_{j'}$ updates either nn_{01} or nn_{00} according to the error vector (Line 4). The reason for component $c_{j'}$ handling only rows for which it is the first component that observes the row, is to avoid duplicate reports by different components on the same rows.

Algorithm 2: request_missing

Input: c_j - the component requesting the data
Input: $c_{j'}$ - the component returning the data
Result: $nn_{01},\ nn_{00}$ - counter data

1 $nn_{01},\ nn_{00} \leftarrow 0, 0$
2 **for** $i \in S^{j'}$ **do**
3 $\quad$ **if** $(\nexists j'' < j'\ s.t.\ S_{i,j''}^{j'} = 1) \bigwedge S_{i,j}^{j'} = 0$ **then**
4 $\quad\quad q \leftarrow E_i^{j'}, nn_{0q}(j) \leftarrow nn_{0q}(j) + 1$
5 **return** $nn_{01},\ nn_{00}$

Example 2. *We demonstrate how component* c_2 *calculates its similarity counters using its local spectrum presented in Table 2b. After executing lines 2-3 of Algorithm 1,* c_2 *has* $n_{11}(2)$=2 *and* $n_{10}(2)$=0*. Next, it requests information from* c_1 *and* c_3 *in that order.* c_1 *considers runs* r_1*,* r_3*,* r_4 *of* S^1 *(Table 2a). Since* $S_{1,2}^1$=1*, run* r_1 *is skipped, since it means that* c_2 *has the same run in* S^2*. Runs* r_3*,* r_4 *show that* c_2 *does not have them, so* c_1 *uses them to calculate* nn_{01}=1*,* nn_{00}=1*. The same request is sent to* c_3*.* c_3 *considers its local spectra and finds out that for each of its runs, either* c_2 *has them, or* c_1 *has already addressed them, and returns nothing. At the end of the process,* c_2 *has the same counter values as shown in Table 1b.* c_2 *uses them to calculate the likelihood of it being faulty by similarity coefficient algorithm.*

3.3 Distributed SFL for Multiple Faults Problems

Here we present our algorithm for diagnosing a distributed system with multiple faults (DSFLA-MULTI). It first generates diagnoses and then ranks them.

Diagnosis Generation

The components generate diagnoses in two stages. First, each component calculates *local diagnoses* by applying a Minimal Hitting Set (MHS) algorithm (De Kleer 2011; Rodler 2022) using its local spectra. This is done in parallel by all components. Then, c_1 sends the set of local diagnoses it computed to c_2. Subsequently, c_2 refines its set of local diagnoses based on the diagnoses it received from c_1. This continues sequentially, with the last component having a complete set of diagnoses. This set is identical to the one generated by the centralized version of the algorithm.

Algorithm 3 lists the pseudo-code for the algorithm described above from the perspective of component c_j. First,

c_j computes the local diagnoses set *LD* by executing MHS algorithm on its local spectrum (Lines 1-2). Then, it receives a set of previously calculated diagnoses *PD* from c_{j-1} (line 3). Note that if j=1 then *PD*=$\emptyset$. Next, c_j adds *PD* and *LD* as elements of a set and computes its hitting set to get a combined global diagnosis set *GD* (Line 4). *GD* is then refined by removal of duplicate diagnoses and super-sets (Line 5). The generated diagnoses are sent to the next component, or output in case of the last component (Lines 6-8).

Before continuing with the running example, we demonstrate how lines 4-5 in Algorithm 3 work. Line 4 applies MHS on a two-element set, in which the elements are sets *LD* and *PD* to receive a set of sets of sets, and line 5 refines the resulting set by unifying the elements of the sets of sets, and later filtering out subsets. We demonstrate this with a short example. Suppose for example, that $LD = \{\{2\},\{1,3\}\}$ and $PD = \{\{1\}\}$. $MHS(\{LD, PD\})$ returns the set: $GD = \{\{\{2\},\{1\}\},\{\{1,3\},\{1\}\}\}$. After unifying and duplicate removing we receive: $GD = \{\{1,2\},\{1,3\}\}$.

Algorithm 3: Diagnose

Input: c_j - the current component
Result: *GD* - a set of global diagnoses
1 *LC* $\leftarrow local\_conflicts(S^j)$
2 *LD* $\leftarrow$ *MHS*(*LC*)
3 *PD* $\leftarrow$ recieve_diagnoses(c_{j-1})
4 *GD* $\leftarrow$ *MHS*($\{$*PD*, *LD*$\}$)
5 *GD* $\leftarrow refine$(*GD*)
6 **if** $j = |C|$ **then**
7 **return** *GD*
8 $diagnose(c_{j+1}$, *GD*)

	LD	PD	GD
1	$\{\{1\}\}$	$\emptyset$	$\{\{1\}\}$
2	$\{\{2\},\{1,3\}\}$	$\{\{1\}\}$	$\{\{1,2\},\{1,3\}\}$
3	$\{\{2\},\{3\}\}$	$\{\{1,2\},\{1,3\}\}$	$\{\{1,2\},\{1,3\}\}$

Table 3: Example of one diagnosis generation process.

Example 3. *Table 3 demonstrates the iterative process for generating diagnoses described in Algorithm 3. Each of the three rows is executed by the component with the corresponding number, shown in column 1. Column 2 shows the local diagnoses (LD) and is executed in parallel. Columns 3-4 present the sequential process of receiving previous diagnoses (PD) and refining them (GD) at each component. The example is run on 3 components, each of which has one of the spectra shown in Table 2. First, considering its local spectrum, each component calculates its local diagnoses* LD_j *(column 2). Then* c_1 *starts with a previous diagnosis set* $PD_1 = \emptyset$*. This leads to* $GD_1 = LD_1$*. Next,* c_1 *sends* GD_1 *to* c_2 *as* PD_2*.* c_2 *then executes MHS on the set* $\{PD_2, LD_2\}$ *to get* $GD_2 = \{\{1,2\},\{1,3\}\}$*, and sends it to* c_3 *as* PD_3*.* c_3 *then executes MHS on the set* $\{PD_3, LD_3\}$ *to get* $GD_3 = \{\{1,2\},\{1,3\}\}$*. Here the diagnosis process stops, the diagnoses are* $\Delta_1 = \{c_1, c_2\}$ *and* $\Delta_2 = \{c_1, c_3\}$*.*

Ranking Diagnoses

Ranking is also challenging since the components do not possess the entire likelihood function L (Def. 3), so none of them can maximize it by itself. To address this, we define the **Local Likelihood Function** for component c_j:

Definition 6 (Local Likelihood Function). *Given a diagnosis* Δ*, component* c_j*, local spectrum* S^j *and local error vector* E^j*, the* ***Local Likelihood Function*** L^j *is:* $L^j = P(E^j|\Delta) = \prod_{E_i^j \in E^j} L_i^j$*. With* L_i^j *defined similarly as* L_i *in Def. 3, but with relation to* S^j *and* E^j*.*

Consider, for instance, $\Delta_2 = \{c_1, c_3\}$ (Ex.3), and the spectra and error vectors in Table 2. The local likelihood functions of c_1, c_2, c_3, are presented in Table 4 column 3. This raises a challenge in performing **gradient descent** as done in Barinel, since for each component c_j, some terms are missing ($L^j_{missing}$). As a result, a component can not maximize L by itself. On the other hand, reconstructing L^j to a complete L by exchanging the missing terms will allow components to reconstruct the complete spectrum, and reveal private information, since there is a bijective relation between L_i and row r_i. To that end we propose a distributed version of gradient descent, where the components share two values: (1) the value of L, and (2) the values of $h_j \in H$. This ensures that the complete function L will not be known to the components. It is worth noting that L, L^j and $L^j_{missing}$ are functions. Throughout the demonstration of our algorithms, we denote the numerical values of these functions as l, l^j and $l^j_{missing}$, respectively.

Given a diagnosis to be ranked, each component c_j initializes the group H=$\{h_j$=$1/2\}_{j=1}^{|C|}$, and its local likelihood function L^j. Next a sequential process starts with c_1 and ends with c_M during which the value l is computed, with each component c_j updating it in turn. Next, $c_{|C|}$ broadcasts the final value l to the other components, and then a parallel process occurs, where each component calculates its partial gradient ∇_j and updates h_j accordingly. The updated value of h_j is broadcast to all components, to ensure the values in H are the same for all components.

Algorithm 4 details the ranking algorithm from the perspective of component c_j. c_j receives as input a global diagnosis Δ. First c_j initiates the array of health values $H = \{h_j = 1/2\}_{j=1}^{|C|}$, its local likelihood function L^j and a threshold for the process termination ϵ (Line 1). At the beginning, component c_1 sets the initial values of l and l_{prev} (Lines 2-3). Then, a gradient descent loop follows (Lines 4-16), which halts when l has converged (Line 4). In each iteration, c_j waits to receive the updated value l from the previous component (Line 7). c_j then extends it by multiplying it with all the terms L^j_k of its local estimation function L^j that were not observed by previous components and then evaluates the resulting function (Lines 8-9). Then c_j sends l to c_{j+1}, and waits to receive the final l from $c_{|C|}$ (Line 13). In case that $j = |C|$, c_j broadcasts the updated l to all the components (Lines 10-11). This concludes the sequential stage of calculating the value l. c_j uses l to calculate its own h_j (Lines 14-15). It does so by dividing l by l^j to obtain the

j	H	$\mathbf{L^j}$	$\mathbf{l^j}$	$\mathbf{derivative_j}$	l	$\mathbf{l^3}$	$\mathbf{l^j_{missing}}$	$\mathbf{gradient_j}$	next H
1	1/2, 1/2, 1/2	$(1-h_1)\cdot(1-h_1)\cdot(h_1\cdot h_3)$	1/16	-1/8	1/16	1/32	1/2	-1/16	7/16, 1/2, 1/2
2	1/2, 1/2, 1/2	$(1-h_1)\cdot(1-h_3)$	1/4	0	1/32	1/32	1/8	0	1/2, 1/2, 1/2
3	1/2, 1/2, 1/2	$(1-h_3)\cdot(h_1\cdot h_3)$	1/8	0	1/32	1/32	1/4	0	1/2, 1/2, 1/2

Table 4: Example showing a single iteration of Algorithm 4 for ranking the diagnosis $\Delta_2 = \{c_1, c_3\}$. Each row shows different values that each component possesses. In the beginning the components have columns 2-5. Column 6 is calculated sequentially, and once it is done, each component uses column 6 to compute the values in columns 6-10 in parallel.

Algorithm 4: Rank

Input: c_j - the current component
Input: Δ - a global diagnosis
Result: p - the probability of Δ

1 $H,\ L^j \leftarrow init(\Delta, |C|), \epsilon \leftarrow 0.005$
2 **if** $j = 1$ **then**
3 $\quad l \leftarrow 0, l_{prev} \leftarrow -1$
4 **while** $|l - l_{prev}| > \epsilon$ **do**
5 $\quad l_{prev} \leftarrow l$
6 $\quad$ **if** $j \neq 1$ **then**
7 $\quad\quad l \leftarrow receive\_prev(c_{j-1})$
8 $\quad L^j_{extended} \leftarrow l \cdot \prod_{k:\nexists j' < j \ s.t.\ S^j_{k,j'}=1} L^j_k$
9 $\quad l \leftarrow L^j_{extended}(H)$
10 $\quad$ **if** $j = |C|$ **then**
11 $\quad\quad broadcast(l)$
12 $\quad$ **else**
13 $\quad\quad send(c_{j+1}, l), l \leftarrow receive\_final(c_{|C|})$
14 $\quad l^j_{missing} \leftarrow \frac{l}{l^j}$
15 $\quad \nabla_j \leftarrow l^j_{missing} \cdot \frac{\partial h_j}{\partial L^j}(H)$
16 $\quad h_j \leftarrow h_j + \nabla_j, broadcast(h_j)$
17 **return** p

value $l^j_{missing}$ (Line 14). Then c_j calculates the gradient by multiplying $l^j_{missing}$ with the result of the partial derivative of L^j with respect to h_j (Line 15). As a final step of this iteration, c_j updates h_j and broadcasts it.

Example 4. *We now demonstrate the process of ranking the diagnoses by showing one iteration of ranking $\Delta_2 = \{c_1, c_3\}$. Table 4 shows the values held by each component during the iteration. The process of ranking works alternatively in parallel and in sequence, with the results in columns 2-5 being calculated in parallel, of column 6 in sequence, and of columns 7-10 in parallel again. Throughout this example, when we refer to a column or a row, unless specifically said otherwise, we refer to this table. The algorithm starts simultaneously, where every component initializes $H = \{h_1, h_2, h_3\} = \{1/2, 1/2, 1/2\}$ (column 2), and ϵ is set to 0.005. The local likelihood functions L^j are as shown by column 3, and their values l^j are shown in column 4, with the values of their partial derivative functions with respect to h_j shown in column 5. c_1 initializes l and l_{prev} to 0 and -1. Next, the gradient descent loop is executed (column 6 top to bottom). c_1, as the first component, extends the function L^1 vacuously to get $L^1_{extended} = L^1$. Then, after plugging in H, it gets $l = 1/16$ (Column 6 row 1). l is sent to the next component, c_2, and c_1 starts waiting for the final value of l. c_2, receives $l = 1/16$, and generates $L^2_{extended} = 1/16 \cdot L^2_2 = 1/16 \cdot (1 - h_3)$. (As can be seen in Table 2b, term L^2_2 is the only term that constitutes L^2 for which the condition in Line 7 of the algorithm holds). Next c_2 plugs in H to get $p = 1/16 \cdot (1 - 1/2) = 1/32$ (column 6 row 2). This value is then sent to c_3. c_3 does not contribute any part of L^3, since looking at S^3_2 and S^3_4 shows that L^3_3 and L^3_4 were already evaluated by c_1 and c_2. So c_3, as the last component, broadcasts $l = 1/32$ (column 7). Following is the evaluation of the gradient steps which are done in parallel, and their values are shown in columns 8-10. Every component c_j first saves l as l_{prev} for the sake of l convergence, and then calculates $l^j_{missing}$ (column 8). Next, these values are multiplied by the values of the the partial derivatives of L^j with respect to h_j (column 5), yielding the gradient values of $\nabla = \{-1/16, 0, 0\}$ (column 9). Those values are used to update H (column 10).*

4 Theoretical Analysis

4.1 Soundness and Completeness

Here we present theorems of soundness and completeness for DSFLA-MULTI and that DSFLA-SINGLE and DSFLA-MULTI return similar ranking to the centralized versions. For lack of space, we omit the proofs.

Theorem 1. DSFLA-MULTI *is sound and complete.*

Theorem 2. DSFLA-SINGLE *returns the same ranking as centralized coefficient based algorithms.*

Theorem 3. DSFLA-MULTI *returns the same ranking as the centralized Barinel.*

4.2 Privacy

The proposed algorithms address privacy, as it is one of the motivations of our work. We first introduce the aspect of privacy by formally defining **Hidden Information** and **Revealed Hidden Information Fraction**.

Definition 7 (**Hidden Information).** *Given a component c_j with corresponding local spectrum S^j and local error vector E^j, **Hidden Information (HI)** for component c_j is defined as $HI^j = \{S^j_{i,*} |\ r_i \notin S^j\ \wedge \exists S^{j'} :\ r_i \in S^{j'}\}$.*

Simply put, hidden information are the cells of runs that are not in the local spectrum of c_j but are in the local spectrum of at least one other component. We denote by **Revealed Hidden Information (*RHI*^{j})** the amount of HI that is revealed by c_j. We define a metric for the amount of revealed private information as the percent of hidden information that is revealed out of the entire spectrum, formally:

Definition 8 (**Revealed HI Fraction).** *Given a component c_j with corresponding local spectrum S^j and local error vector E^j, and with HI^j, **Revealed HI Fraction (RHIF)** for component c_j is defined as $RHIF^j = \frac{|RHI^j|}{|R|*(|C|+1)}$.*

Simply put, we measure the fraction of cells hidden for c_j that became revealed out of all the spectrum cells.

For example, Table 2b shows S^2 for which there are $|HI^2| = 8$ cells that are hidden for c_2. At the end of the diagnosis, $RHIF^2 = 4/16 = 0.25$. We now review the proposed algorithms with respect to hidden information.

DSFLA-SINGLE:
Consider Algorithm 1. By getting information about $n_{01}(j)$ or $n_{00}(j)$ from $c_{j'}$, c_j can deduce that $c_{j'}$ has a run $r_i \in S^{j'}$ that includes the cells: $S^{j'}_{i,j'} = 1$, $S^{j'}_{i,j} = 0$ and $E^{j'}_i = 0$ (in case of $n_{00}(j)$), or $E^{j'}_i = 1$ (in case of $n_{01}(j)$). This means that for every $nn_{0q} = k$ returned by $c_{j'}$, c_j reveals $3 \cdot k$ cells.

Example 5. *Returning to Example 2, c_2 has $|HI^2| = 8$. The values of the counters $nn_{01} = 1$ and $nn_{00} = 1$, sent by c_1 to c_2, reveal to c_2 that there is one row r_{i_1} in which $S^1_{i_1,1} = 1$, $S^1_{i_1,2} = 0$ and $E^1_{i_1} = 1$, and another row r_{i_2} in which $S^1_{i_2,1} = 1$, $S^1_{i_2,2} = 0$ and $E^1_{i_2} = 0$. Those rows are summed up to 6 cells that are shown in red in Table 5b. Tables 5a,5c show the same for c_1, c_3 with 3 and 6 cells. It follows that $RHIF^1 = 3/16$, $RHIF^2 = 6/16$ and $RHIF^3 = 6/16$.*

	c_1	c_2	c_3	E
r_1	1	1	0	1
r_2	**0**	**1**	-	**1**
r_3	1	0	0	1
r_4	1	0	1	0

(a)

	c_1	c_2	c_3	E
r_1	1	1	0	1
r_2	0	1	1	1
r_3	**1**	**0**	-	**1**
r_4	**1**	**0**	-	**0**

(b)

	c_1	c_2	c_3	E
r_1	**1**	-	**0**	**1**
r_2	0	1	1	1
r_3	**1**	-	**0**	**1**
r_4	1	0	1	0

(c)

Table 5: Information revealed by each component, denoted as the bold values in it's spectrum for DSFLA-SINGLE.

DSFLA-MULTI:
By applying Algorithm 3, a component can reveal some information from the passed diagnoses *PD*. Stern et al. 2012 show that MHS of diagnoses produces a set of minimal conflicts. Armed with this finding, c_j can apply MHS on *PD* received from c_{j-1}, and reveal rows representing the set of minimal conflicts that led to *PD*. This does not mean that c_j can deduce the entire S, since runs that are corresponding to superset of conflicts will not be revealed. Also, successful runs are not conflicts, therefore will not be revealed.

Example 6. *Returning to Example 3, we show how c_3 can reveal hidden information using PD_3 sent to it from c_2. Recall that $PD_3 = \{\{1,2\},\{1,3\}\}$, by running MHS on this set, the resulting minimal hitting sets are: $\{\{1\},\{2,3\}\}$. These sets indicate the following rows $r_{i_1} = [1,\ 0,\ 0|\ 1]$ and $r_{i_2} = [0,\ 1,\ 1|\ 1]$. Since $r_{i_2} \in S^3$, it is not hidden information to c_3 ($r_{i_2} \notin HI^3$). However, $r_{i_1} \notin S^3$, therefore $r_{i_2} \in HI^3$. HI^3 is shown in Table 6c, and Tables 6a and 6b show the same for c_1 and c_2. Here, it follows that $RHIF^1 = 0/16$, $RHIF^2 = 4/16$ and $RHIF^3 = 4/16$.*

	c_1	c_2	c_3	E
r_1	1	1	0	1
r_2	-	-	-	-
r_3	1	0	0	1
r_4	1	0	1	0

(a)

	c_1	c_2	c_3	E
r_1	1	1	0	1
r_2	0	1	1	1
r_3	**1**	**0**	**0**	**1**
r_4	-	-	-	-

(b)

	c_1	c_2	c_3	E
r_1	-	-	-	-
r_2	0	1	1	1
r_3	**0**	**0**	**0**	**1**
r_4	1	0	1	0

(c)

Table 6: Information revealed by each component, denoted as the bold values in it's spectrum for DSFLA-MULTI.

5 Evaluation

We experimented on samples inspired by the domain of Internet Delay Diagnosis (Stern and Kalech 2014). In this domain a network of routers forwards packages from sources to destinations. In terms of diagnosis, the routers are components, where a faulty router has a delay in forwarding packages, causing the destination to experience this delay. In terms of SFL, the spectra includes information about the routers each package passed through from the source to the destination, where the columns are the routers and the rows are the traces of the packages. The error vector indicates for each package, whether the destination experienced a delay.

5.1 Experimental Setting

The generation of DSFL problems consists of these steps:
1. Create synthetic networks with x routers (components).
2. Set probability of p to delay a packet for f components.
3. Generate random paths for y packets to traverse the network (system runs). We generate local spectra by these runs.
4. A packet has $(1-p)^f$ chance to reach the destination on time. The local error vectors are filled accordingly.

We generated problems with varying number of components $x \in \{6, ..., 12, 13\}$, faulty components $f \in \{1, 2, 3, 4, 5\}$, fault probability values $p \in \{0.1, 0.2, ..., 0.9\}$ and number of runs $y \in \{10, 20, ..., 50\}$. We conducted 30 examples for each combination. In total we run $54,000$ experiments.

Metrics: We measure the performance of the algorithms with respect to the quality of the diagnosis set they return by computing the average **Wasted Effort**, **Weighted Precision**, and **Weighted Recall** (Elmishali, Stern, and Kalech 2020). To understand wasted effort, assume that we repair the components in the diagnoses in a decreasing order of the diagnoses' score. Wasted effort is the number of healthy components examined until all faulty components are repaired. For weighted precision and recall, note that a diagnosis is an assumption about which components are faulty. Thus, using the knowledge about the ground truth faulty components, we can compute the precision and recall of each single diagnosis. To compute the precision and recall of a set of diagnoses, we compute the weighted averages, where the precision/recall for every diagnosis is weighted by its score as returned by the diagnosis algorithm.

	Single Centr.	Single Baseline	Multi BARINEL	Multi Baseline	Multi MC	Multi SMC
Wasted Effort	2.82	2.82	2.43	2.43	1.48	**1.28**
Weighted Precision	0.36	0.36	0.64	**0.64**	0.62	0.45
Weighted Recall	0.13	0.13	0.50	**0.50**	0.46	0.37
Runtime (seconds)	3.8e-4	1.06e-3	2.57	10.44	1.05	**0.39**
RHIF	-	0.06	-	0.05	3.35e-4	**1.01e-4**
#Messages per agent	-	25.5	-	65.51	13.92	**4.60**

Table 7: Wasted effort percent, weighted precision/recall, runtime, hidden information revealed, and communication load.

Additionally, we measure the average Revealed Hidden Information Fraction **RHIF** and the average **#Messages**. As for #Messages, we define it as the information units passed between the components during the diagnosis process. In that context, we regard a spectrum cell, a component of a diagnosis, a number, and an information request to be with the size of 1 unit. For example, when c_2 passes the diagnoses $\{\{1,2\},\{1,3\}\}$ to c_3, then #Messages=4.

Compared Algorithms: We compare DSFLA-MULTI to two unsound and incomplete variations. Instead of each component sending **ALL** local diagnoses to the next one (Line 8 in Algorithm 3), in (1) **Multi-MC** it sends only **all minimal cardinality** diagnoses, and in (2) **Multi-SMC** it sends only a **single minimal cardinality** diagnosis. From now on, we refer to the baseline variation of DSFLA-MULTI as (3) **Multi-Baseline** and to the DSFLA-SINGLE as (4) **Single-Baseline**. In addition, we compare these algorithms to a centralized version, where all components send their matrices to the diagnoser. We denote these algorithms (5) **Centr. Single** and (6) **Centr. Barinel**.

5.2 Results

Table 7 shows the results. The first three rows present the performance in terms of wasted effort, weighted precision and weighted recall. Here, lower value of wasted effort is better and higher values for weighted precision and recall are better. The last two rows present the results of the average *RHIF* and the average number of messages. Here lower numbers are better. The best values are shown in bold. The first two columns show the results of algorithms dealing with single faults - the centralized and distributed Single. The next 4 columns present the results for the algorithms for multiple faults - the centralized Barinel, Multi-Baseline, Multi-MC and Multi-SMC.

The results show that the baseline distributed algorithms output the same diagnoses as the centralized algorithms. This is reflected in the identical results between the centralized and the baseline distributed algorithms, both for single faults and multi faults, in terms of wasted effort, weighted precision and weighted recall.

The results also show that the precision and recall decrease for the unsound and incomplete variations of the distributed algorithm (Multi-MC and Multi-SMC). The wasted effort decreases as well although these variations are unsound and incomplete. This can be explained by the fact that the two variations return less diagnoses by definition - which means less healthy components to examine.

In terms of privacy and communication load, the results show that the *RHIF* and the number of messages decrease for the unsound and incomplete variations of the distributed algorithms. This is expected since less sets of diagnoses are sent by the components. This causes the components to discover less conflicts and by that less hidden cells in their respective spectra (https://github.com/avi-natan/DDIFMAS).

6 Discussion

Throughout the paper, we drew some assumptions to simplify the problem.

One such assumption is of perfect communication. Since the components are distributed, our algorithms require the components to be able to communicate information. Here, perfect communication is assumed, i.e., challenges such as disconnection of a component, delays in communication and alternation of communication (accidental or deliberate) were not assumed.

The second assumption is related to the collaboration of the components. We assume a system which is privacy aware yet collaborative. In it, prior agreement related to number of components, component sequencing and spectrum and error vector filling takes place between the components.

Finally, we assume specific vision the components have. For every run they participate in, they also know about every other participating component. This directly influences our definition of the local spectrum. Additionally, we assume that every component that participates in a run, can see its outcome, which means that every component can see the error vector for runs it participates in. Different assumptions about what information is available to each component might influence the methods by which the components share the information they have.

7 Conclusions

In this paper we pointed out the challenges SFL faces when diagnosing distributed systems and formalized the problem of Distributed SFL (DSFL). To solve this problem we presented distributed versions of the mentioned algorithms: DSFLA-SINGLE and DSFLA-MULTI. We evaluated our algorithms theoretically and empirically. Evaluation shows that the algorithms achieve similar output to the centralized algorithms whilst addressing privacy, and that variations of DSFLA-MULTI can preserve more privacy with little cost in diagnosis quality.

Acknowledgments

This research was funded by ISF grant No. 1716/17, by the ministry of science grant No. 3-6078, and (partially) by the The Israeli Smart Transportation Research Center (ISTRC).

References

Abreu, R.; Zoeteweij, P.; and Van Gemund, A. J. 2007. On the accuracy of spectrum-based fault localization. In *Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)*, 89–98. IEEE.

Abreu, R.; Zoeteweij, P.; and Van Gemund, A. J. 2009. Spectrum-based multiple fault localization. In *2009 IEEE/ACM International Conference on Automated Software Engineering*, 88–99. IEEE.

De Kleer, J. 2011. Hitting set algorithms for model-based diagnosis. In *International Workshop on Principles of Diagnosis (DX-11)*.

De Kleer, J.; and Williams, B. C. 1987. Diagnosing multiple faults. *Artificial intelligence*, 32(1): 97–130.

Elmishali, A.; Stern, R.; and Kalech, M. 2016. Data-augmented software diagnosis. In *Twenty-Eighth IAAI Conference*.

Elmishali, A.; Stern, R.; and Kalech, M. 2018. An artificial intelligence paradigm for troubleshooting software bugs. *Engineering Applications of Artificial Intelligence*, 69: 147–156.

Elmishali, A.; Stern, R.; and Kalech, M. 2020. Diagnosing Software System Exploits. *IEEE Intelligent Systems*, 35(6): 7–15.

Hofer, B.; Perez, A.; Abreu, R.; and Wotawa, F. 2015. On the empirical evaluation of similarity coefficients for spreadsheets fault localization. *Automated Software Engineering*, 22(1): 47–74.

Passos, L. S.; Abreu, R.; and Rossetti, R. J. 2015. Spectrum-based fault localisation for multi-agent systems. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Perez, A.; Abreu, R.; and Riboira, A. 2014. A dynamic code coverage approach to maximize fault localization efficiency. *Journal of Systems and Software*, 90: 18–28.

Pérez-Zuñiga, C.; Chanthery, E.; Travé-Massuyès, L.; Sotomayor, J.; and Artigues, C. 2018. Decentralized diagnosis via structural analysis and integer programming. *IFAC-PapersOnLine*, 51(24): 168–175.

Perez-Zuniga, G.; Chanthery, E.; Travé-Massuyès, L.; and Sotomayor, J. 2022. Near-Optimal Decentralized Diagnosis via Structural Analysis. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

Rodler, P. 2022. Memory-limited model-based diagnosis. *Artificial Intelligence*, 305.

Stern, R.; and Kalech, M. 2014. Model-based diagnosis techniques for Internet delay diagnosis with dynamic routing. *Applied intelligence*, 41(1): 167–183.

Stern, R. T.; Kalech, M.; Feldman, A.; and Provan, G. 2012. Exploring the duality in conflict-directed model-based diagnosis. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Syfert, M.; Bartyś, M.; and Kościelny, J. 2018. Refinement of fuzzy diagnosis in decentralized two-level diagnostic structure. *IFAC-PapersOnLine*, 51(24): 160–167.

Williams, B. C.; and Ragno, R. J. 2007. Conflict-directed A* and its role in model-based embedded systems. *Discrete Applied Mathematics*, 155(12): 1562–1595.