The Journal of Systems and Software 161 (2020) 110490

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Automatic retrieval and analysis of high availability scenarios from system execution traces: A case study on hot standby router protocol

Maged Sheghdara, Jameleddine Hassine*

Department of Information and Computer Science, KFUPM, Dhahran, KSA, Saudi Arabia

ARTICLE INFO

Article history:
Received 22 September 2019
Revised 2 December 2019
Accepted 3 December 2019
Available online 4 December 2019

Keywords:
High availability
Non-functional
Dynamic analysis
Trace segmentation
Error detection and diagnosis
Hot standby router protocol

ABSTRACT

High availability (HA) is becoming an increasingly important requirement in a growing number of domains. It is even mandatory for critical systems, such as networking and communications, that cannot afford downtime. Such systems often monitor the state of crucial services and produce huge amounts of execution trace data, where functional and non-functional log entries are intertwined; hence they are hard to dissociate and analyze. Dynamic analysis aims at capturing and analyzing run-time behavior of a system based on its execution traces. In this paper, we apply dynamic analysis to retrieve and analyze HA scenarios from system execution traces. Our proposed approach aims to help analysts understand and report on how a highly available system detects and recovers from failures. As a proof of concept, we have selected the Hot Standby Router Protocol (HSRP) in order to demonstrate the applicability of our approach. We have evaluated empirically the effectiveness of our technique using four real-world case studies of IP networks running HSRP. Results have shown that high availability scenarios were successfully retrieved and analyzed. Moreover, results have shown that our prototype tool *HAAnalyzer* was able to effectively unveil high availability behavioral and temporal errors, that were seeded in the execution traces.

1. Introduction

A large part of software maintenance is comprehension (Cornelissen et al., 2009a). In order to acquire an adequate level of understanding of a software system, engineers have to spend tremendous amount of time to study different software artifacts, e.g., design document, source code, system test plan, etc. (Corbi, 1989). However, in reality, most of actual systems have outdated and poor documentation, if it exists at all. Dynamic analysis is one common approach used to recover and comprehend system functionalities, using run-time data (Cornelissen et al., 2009a). Typically, a system source code is instrumented in a way that when run, it generates a *trace*, also called a *log* (in this paper, we use the terms *log* and *trace* interchangeably), that contains the sequence of events/actions that a system generates/executes during a particular scenario run. Such traces map high-level descriptions of software behavior (emanating from Software Requirements Specifications (SRSs) and design documents) to low-level implementation, i.e., source code. The resulting execution traces are then used to analyze the system behavior. It is worth noting that the availability of source code is not a mandatory condition to apply dynamic analysis.

Dynamic analysis, however, has its limitations as well. The most important one is the inherit complexity of typical execution traces (Hamou-Lhadj and Lethbridge, 2005). Indeed, the size of an execution trace may range from several thousands to several millions of lines (known as the problem of size explosion (Zaidman, 2006)) and it contains a large amount of noise, making analysis overly complicated. To address this problem, researchers proposed many trace compression, simplification, and abstraction techniques (Reiss, 2005; Watanabe et al., 2008; Reiss, 2006; Koskimies and Mossenbock, 1996; Walker et al., 1998).

Dependability is the property that enables users to rely on a system. It is defined as "that property of a computer system such that reliance can justifiably be placed on the service it delivers" (Laprie, 1992). As well as being a critical property for a wide variety of applications, e.g., telecommunications, banking, etc., dependability is also regarded as being multi-dimensional since it encloses many attributes such as availability and reliability. Avizienis et al. (2004) defined availability as being "the readiness for a correct service". More practically, availability denotes the measure of how often or how long a service or a system component is available for use. Jalote (1994) considered system reliability augmented with fault recovery as the basis for building

* Corresponding author.
E-mail addresses: g201304030@kfupm.edu.sa (M. Sheghdara), jhassine@kfupm.edu.sa (J. Hassine).

https://doi.org/10.1016/j.jss.2019.110490
0164-1212/

system availability. Furthermore, failure recovery may be achieved through component redundancy, repair, or fault masking. A key difference between reliability and availability have been explained by Gokhale et al. (2005) as follows: "reliability refers to failure-free operation during an entire interval, while availability refers to failure-free operation at a given instant of time". Different notions of availability have been discussed in the literature (Resnick, 1996), including basic availability, continuous availability and high availability (HA). High availability is defined as the capacity of a system to satisfy its requirements despite failures (Resnick, 1996) and is quantified by having the services available 99.999% (a.k.a five nines) of the time (Avizienis et al., 2004). Systems implementing high availability capabilities (through features, called *HA features*) are expected to remain operational at all times (even in presence of failures) with no perceived loss of service.

Although, system availability is often considered crucial in the determination of success or failure of critical systems, researchers mainly focused on applying dynamic analysis to capture and understand functional properties of systems and less research addressed system availability (Hassine and Hamou-Lhadj, 2014; Hassine et al., 2018). The major motivation of this research is to apply dynamic analysis to retrieve and analyze high availability scenarios from system execution traces. More specifically, our aim is to help system analysts understand high availability mechanisms, given a set of run-time system execution traces.

Because of the particularity of high availability scenarios, applying dynamic analysis to extract and analyze them is confronted by the following important challenges:

- *Combining and analyzing multiple execution traces*: A key method of providing high availability is through redundant hardware and/or software components, avoiding the single point-of-failure problem. If the *active* (also called *primary*) component is suddenly unavailable (as a result of a failure or a planned maintenance task), the *standby* (also called *secondary*) is promoted to become the new active. Hence, in order to comprehend a high availability scenario, we have to identify all involved physical components, e.g., active and standby, and consider all their execution traces in the analysis.
- *Dissociating high availability from functional log entries*: Achieving high availability requires the implementation of monitoring mechanisms such as heart-beating, state synchronization, etc., to maintain service availability. These high availability mechanisms, implemented by HA features, generate HA log entries that are usually intertwined with functional logs. Hence, there is a need for techniques capable of dissociating them.
- *Segmentation of high availability scenarios into execution phases*: The analysis of execution traces produced by many high availability features, such as, Cisco Hot Standby Router Protocol (HSRP) (Cisco Systems, 2006), Cisco Adaptive Security Appliance (ASA) firewall active/standby redundancy (Deal, 2009), Cisco Route Switch Processor (RSP) stateful switchover (Cisco Systems, 2016), and Automatic Protection Switching (APS) (Maier et al., 2002), revealed that a typical high availability scenario is composed of three main execution phases: (1) failure occurrence, (2) failure detection by the HA feature, and (3) failure recovery by the HA feature. As a consequence, splitting up a trace into multiple execution phases and identifying how the resulting segments are correlated, would enhance the comprehension of HA mechanisms and hence would facilitate their analysis.
- *Diagnosis of high availability scenarios*: An HA execution scenario may exhibit a wrong behavior (e.g., no failure recovery) or a temporary service interruption (e.g., due to a slow recovery from a failure). Such anomalies may be the result of flaws in the HA feature implementation or a wrong user configuration. Hence, it is essential to help analysts detect and analyze such anomalies.

In this paper, we try to address these challenges. We make the following contributions:

1. We provide an automated approach to retrieve high availability scenarios from system traces implementing HA features, allowing analysts to comprehend how failures are detected and handled by a given HA mechanism.
2. We offer a fine-grained rule-based trace segmentation and correlation technique. The resulting segments, called also *execution phases*, as well as the computed correlations are visualized using a multi-color tabular format. Furthermore, our technique allows for the detection and diagnosis of anomalies embedded in execution traces produced by the execution of HA scenarios, such as, absence of recovery after failure detection (i.e., behavioral issue) or slow failure recovery (i.e., temporal issue).
3. As a proof of concept, we have applied our approach to retrieve and analyze HA scenarios extracted from system traces implementing the Cisco Hot Standby Router Protocol (HSRP) (Cisco Systems, 2006). Moreover, we implemented our approach in a prototype tool, called *HAAnalyzer*, that automates all proposed activities. We validated empirically the effectiveness of our approach and prototype tool using four real-world case studies.

The remainder of this paper is organized as follows. Next section presents our proposed approach for retrieving and analyzing high availability scenarios from system execution traces. Section 3 introduces Cisco HSRP (Cisco Systems, 2006) high availability feature, used as our case study. In Section 4, we apply the proposed approach to Cisco HSRP protocol. Section 5 describes briefly our prototype tool, called *HAAnalyzer*. The empirical evaluation of our approach and tool is presented in Section 6. Section 7 presents a comparison with related work and discusses the benefits and potential threats to validity of our proposed approach. Finally, conclusions and future work are presented in Section 8.

2. High availability scenarios retrieval and analysis approach

In this section, we present our proposed approach to retrieve and analyze high availability scenarios from system execution traces.

Fig. 1 describes the six steps of our approach as an UML activity diagram. The approach consists of six major steps: (1) collect and filter the relevant execution traces, (2) merge the filtered traces and order the aggregated trace, (3) segment the resulting trace into execution segments/phases, (4) correlate the identified execution phases, (5) detect and diagnose both functional and temporal problems, (6) visualize the results.

2.1. Execution traces collection and filtering

Typically, an execution trace is composed of a set of run-time events that are chronologically ordered. An execution trace is usually stored as plain text file, which facilitates their integration with simulation and analysis tools. An execution trace event is typically described using, among others, a timestamp, the process ID generating the event, and a brief description of the event. For example, a typical Cisco IOS log message has the following format (Cisco Systems, 2004):

timestamp: \%FACILITY-severity-MNEMONIC: Message-text

where *timestamp* refers to the time of the occurrence of the event or the reception of a system message, *FACILITY* refers to the software module/protocol/device for which logging is set (e.g.,

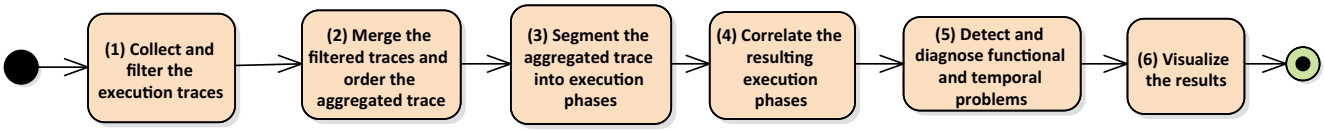


Fig. 1. High availability scenarios retrieval and analysis approach.

PARSER, etc.), *Severity* (a digit between 0 and 7) denotes the severity of the event, i.e., 0 being the most severe (emergency) and 7 being the least severe (debugging), *MNEMONIC* is a string that describes the system message. *Message-text* is a text providing more information about the event being reported.

This definition avoids any confusion between execution and network traces. Indeed, network traces represent data traffic over a wire (collected by hooking a machine to the network or using network packet analyzer software), while execution traces are the result of using logging to record information about a program's internal execution (usually achieved through code instrumentation).

2.1.1. Trace collection

Since achieving high availability may require the presence of separate redundant physical components, traces from all involved components should be considered. Hence, a prior knowledge of the participating systems/components is required.

2.1.2. Trace filtering

The collected execution traces tend to contain a huge amount of data (several thousands to several millions of lines) and a lot of noise, which make them hard to analyze. To address this issue, many trace abstraction and filtering techniques have been proposed. Most of these techniques can be classified into three main categories:

1. *Detection and removal of utility routines*: These simplification techniques aim to detect general purpose routines, that are used across the system (Hamou-Lhadj and Lethbridge, 2006). These techniques generally require careful code instrumentation in order to capture all method calls.
2. *Abstraction (or generalization) techniques*: These techniques are based on the extraction of high-level views from low-level operations (Cornelissen et al., 2008; Pirzadeh et al., 2013; Hamou-lhadj and Lethbridge, 2003; Kuhn and Greevy, 2006).
3. *Pattern detection*: These techniques consist of matching and grouping trace entries based on their similarity (Hamou-Lhadj et al., 2004; Hassine et al., 2018), e.g., keywords, regular expressions, etc.

In our context, we assume that the system source code is not available, so we cannot instrument it. In addition, we assume that the traces collected from systems running HA features are native logs that do not contain any information related to method calls. Hence a detection of utility technique is not applicable in our context. Furthermore, a system that implements an HA feature may also be loaded with dozens of other features, e.g., routing protocols, security and QoS features, etc. Consequently, trying to create an abstract view from a trace produced from such feature-loaded systems may lead to a large set of useless groups (e.g., group for each feature or sub-feature) that are not relevant to system high availability. For this reason, abstraction techniques are not suitable for our context. Moreover, the analysis of systems implementing highly available features, would require a prior knowledge about which HA features are deployed (e.g., HSRP (Cisco Systems, 2006), APS (Maier et al., 2002), etc.), how to recognize a failure (e.g., link down, process crash, etc.), and how to recognize a recovery (e.g., link up, process restarted, etc.). Consequently, a keyword-based filtering technique (belongs to the pattern matching category) is deemed the most appropriate in our context.

A keyword-based trace filtering technique may have as input:

- A system native trace.
- Lower and upper timestamp boundaries (optional).
- A set of keywords specifying (i) the name of the targeted HA feature, e.g, HSRP, (ii) keywords characterizing failure situations, e.g., link DOWN, and (iii) keywords denoting recovery situations, e.g., link UP.
- An optional timestamp boundary.

A generic filtering algorithm consists of at least two main steps:

1. **Step 1**: Exclude entries outside the user defined timestamp boundaries.
2. **Step 2**: Parse the entire trace and remove all entries that do not contain the user specified keywords.

However, depending of the log complexity and its level of granularity, further refinements of the algorithm may be necessary. In Section 4.1, we present an HSRP (Cisco Systems, 2006) specific 3-step trace filtering algorithm (Algorithm 1).

2.2. Execution traces merging and ordering

The output of the previous step is a set of filtered text files, each file corresponds to the trace of one single system/component. In this step, execution traces are merged and sorted according to their timestamps. However, it is worth noting that in case of an HA scenario execution involving heterogeneous systems, e.g., systems from different vendors, all timestamps should be normalized to a common timestamp format. In addition, it is essential to not loose the source of each log entry. In Section 2.2, we have used a table to store logs and we have added an extra field to store the router name from which the trace is collected.

2.3. Trace segmentation

Trace segmentation refers to the process of identifying subsequences (or clusters) of events/actions within a trace. The rationale behind trace segmentation is that every cluster of low-level events is supposed to represent the execution of a higher-level activity. In addition, retrieved clusters should be categorized and characterized. Many segmentation techniques have been proposed in the literature (Günther et al., 2010; Asadi et al., 2010; Aguilar et al., 2015; Gonzalez et al., 2009). Günther et al. (2010) proposed a global trace segmentation approach, where closely related events are grouped into *event classes* and represented as an hierarchy of event classes, i.e., types of events. Asadi et al. Asadi et al. (2010) introduced a heuristic-based technique which uses a genetic algorithm to segment traces. Aguilar et al. (2015) proposed an on-line approach that uses event flow graphs to segment a given execution trace. Gonzalez et al. (2009) proposed a density-based clustering approach. Their approach (Gonzalez et al., 2009) applies density-based algorithms, such as DBSCAN, to group the points that have big density regardless of the model or data structure.

Prior to designing our segmentation technique, we have investigated the structure of traces produced by different HA features, such as Cisco Hot Standby Router Protocol (HSRP) (Cisco Sys-

Algorithm 1: Trace Filtering Algorithm.

```
Procedure Name: FilterTrace
Input  : TraceFile.txt, String keywords, DateTime
         startTimeBoundary, DateTime
         endTimeBoundary
Output: FilteredTrace.txt
      ▷ We define array (TempArr) to insert the filterd
entries in itstring [ ][ ] TempArr = new string [ ];
string[ ][ ] Trace = Read(TraceFile.txt);
int N = Trace.Length;
int related =0;
    ▷ Exclude entries which are outside TimeBoundary
for (int i=0 ; i <N ; i++) do
    if (Trace [x][TimeStamp] < startTimeBoundary OR
    Trace [i][TimeStamp] > endTimeBoundary) then
        Delete Trace[i];
    end
end
  ▷ Exclude entries which do not contain any keyword
for  (int i=0 ; i <N ; i++) do
    foreach (keyword key in keywords) do
        if Trace[i][Header].Contain(key) then
            TempArr.Add(Trace[i]);
        end
    end
end
▷ Exclude entries which do not participate in the HSRP
featureN = TempArr.Length;
for  (int i=0 ; i <N ; i++) do
    related = 0;
    if (TempArr[i][Header].Contain("HSRP")) then
        continue;
    else
        for (int j = N; j >= 0 ; j − −) do
            if ((TempArr[i][Router] == TempArr[j][Router])
            AND (TempArr[i][Interface] ==
            TempArr[j][Interface]) AND
            TempArr[j][Header].Contain("HSRP")) then
                related = 1;
                break ;
            else
                continue;
            end
        end
        if (realted == 0) then
            Delete TempArr[i];
        end
    end
end
FilteredTrace.write(TempArr);
```

tems, 2006), Cisco Adaptive Security Appliance (ASA) firewall active/standby redundancy (Deal, 2009), Cisco Route Switch Processor (RSP) stateful switchover (Cisco Systems, 2016), and Automatic Protection Switching (APS) (Maier et al., 2002). The investigation showed that a typical high availability scenario is composed of three main execution phases (see Fig. 2):

1. *Failure occurrence*: A service failure represents an impairment in a system's function. A failure is generally preceded by an *error* representing a deviation from the correct state. The hypothesized cause of an error is called a *fault* (Avizienis et al., 2004). A failure is observed when its presence is indicated by an error message or an *unwanted event* in the execution trace. Examples of failures include: network link failure, process crash, route-processor crash, etc.
2. *HA Feature failure detection*: Highly availability features often implement accurate instance monitoring, e.g., heartbeat mechanisms. A high availability feature should monitor the state of both active (i.e., in service) and standby (i.e., backup) elements. Generally, at least three states can be used to describe the status of monitored elements:
   - *Inactive*: state that corresponds to a failed (not in-service) or an un-monitored element.
   - *Standby*: state that denotes a backup (called also *secondary* or *protect* in the terminology of some HA features) element (healthy element waiting to takeover the active role).
   - *Active*: state that refers to an in-service *active* (called also *primary* and *working* in the terminology of some HA features) element.

   For example, the HSRP (Cisco Systems, 2006) protocol maintains 6 states, ranging from *Initial* (i.e., Inactive) to *Active*, as described in Section 3.
   In what follows, we introduce the concept of HA state *upgrade* and *downgrade*, that will be used to characterize failure detection and recovery. An HA state change can be either a state *upgrade* or a *downgrade*:

   Inactive, **Standby**, **Active**

   $\xrightarrow{\text{Upgrade}}$

   $\xleftarrow{\text{Downgrade}}$

   As a general rule, the detection of a failure of a monitored element is associated with an HA state downgrade. For instance, when an *active* link goes down, its state is downgraded from *Active* to *Inactive*. Similarly, when a *standby* link goes down, its state is downgraded from *Standby* to *Inactive*.
3. *Failure recovery*: Upon detection of a failure, and in presence of a redundant component, an HA feature should trigger a fail-over. A failure recovery is generally associated with an HA feature state upgrade. For example, in case of a component failure, a fail-over would move all active services to the standby component. Hence, the HA state of the backup component is upgraded from *Standby* to *Active*. In case of a failure of an active network link, a fail-over would switch all traffic to the standby link, which becomes in active state. Once the failed link is repaired, its state will be upgraded from *Inactive* to *Standby*.

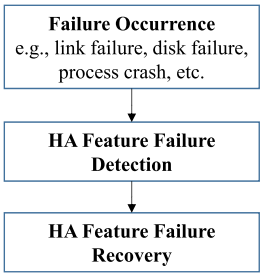


Fig. 2. Main typical HA scenario steps.

In our approach, we segment the filtered and merged trace into segments, called "execution phases". Each execution phase is composed of a contiguous sequence of log entries, since we aim to preserve the order of entries within a trace, to be able to link a failure to its corresponding recovery (see trace correlation

procedure in Section 2.4). Segmentation is similar to clustering but in clustering, it is not required to preserve the order of entries.

Several techniques for searching log cut-points (Sultana et al., 2017) have been proposed, such as, interpolation search, binary search, jump search, and linear search. We suggest the use of a linear search because we have to check every single log entry. Our segmentation procedure relies on the HA feature state change. That is, whenever we encounter an HA feature state downgrade or upgrade, we create a new execution phase. Algorithm 2 describes our HSRP (Cisco Systems, 2006) specific segmentation procedure.

Algorithm 2: Trace Segmentation Algorithm.

```
Procedure Name: SegmentTrace
Input  : FilteredTraces.txt        ▷ Filtered and Merged Traces
Output: Segmented Trace
string [ ][ ] Trace = Read (FilteredTrace);
▷ N refers to phase number
int N=1;
for (int i=0 ; i <Traces.Length ; i++) do
    if (Trace[i][Header].Contains("HSRP-5-STATECHANGE")
    AND (Trace[i][Status
    change].Contains(HSRP-Upgrading) OR Trace [i][Status
    change].Contains(HSRP-Downgrading)) then
        Trace[i][Phase]=++N;                ▷ New phase
    else
        Trace[i][Phase]=N;
    end
end
```

2.4. Execution phases correlation

Execution phases correlation aims at connecting causes to effects. It has been applied as a root cause analysis (RCA) approach to find the reasons of failures in case of a bug or a security breach (Zawawy, 2012). There are several approaches for phase correlation, such as, rule-based, probabilistic, and model-based approaches.

In our context, a probabilistic approach is not adequate since we don't deal with neither variations nor uncertainties that require the use of distributions instead of fixed values. In addition, our proposed approach suggests the use of native logs without the need for building a formal model, hence making a model-based approach not appropriate. Consequently, in this research, we adopt a rule-based correlation approach since it is the most suitable to our context and assumptions.

Indeed, our designed rules intend to correlate a failure occurrence (Step 1 in Fig. 2) to its corresponding recovery (Step 3 in Fig. 2). These rules have typically the following form:

< **If** Condition(HA state, etc.) **then** Correlation (Phase(x), Phase(y)) >.

where *Condition* denotes a first order predicate that includes, among others, checks on HA state upgrade/downgrade, and *Correlation* denotes the existence of a cause-effect relationship between the execution phases that contain log entries *x* and *y*.

Correlations rules should be designed with the intent to correlate: (1) a failure occurrence to its detection by the HA feature, (2) a failure occurrence to its recovery (as a result of a fail-over), and (3) a detection of a failure by the HA feature (HA state downgrade) to its recovery (HA state upgrade). In Section 4.5, we define three correlation rules for HSRP (Cisco Systems, 2006).

2.5. Error detection and diagnosis

The output of the correlation phase would serve as a basis for checking the HA feature behavior and diagnose potential anomalies in the collected traces. Indeed, the correlation phase has three possible outcomes:

1. **Trace exhibits a wrong behavior**: Such functional problems may be due to faulty feature implementation or to a feature interaction problem. For example, the HA feature fails to detect a failure (absence of an HA state downgrade) and does not recover (no fail-over is triggered).
2. **Trace exhibits a temporal problem**: The system recovers from a failure but the recovery takes place outside of the tolerable/acceptable response time, i.e., slow recovery. Temporal issues are identified using "Low" correlation between the phase that contains failure and the phase that contains the recovery.
3. **Trace exhibits a correct behavior**: A correct HA scenario, where the system recovers from a failure within an acceptable period of time. That is, the phase that corresponds to the recovery is highly correlated ("High" correlation) with the phase that contains the failure.

2.6. Results visualization

Several visualization techniques have been proposed to visualize and navigate the content of a trace (Duan and Cleland-Huang, 2006a; Cornelissen et al., 2009c). These techniques have used different notations, e.g., tables (Winkler, 2008), Charts (Maoz et al., 2007; Cornelissen et al., 2009b; 2011; Osmari et al., 2014; Fittkau et al., 2015; Mohror et al., 2010; Maoz and Harel, 2011), matrices (Duan and Cleland-Huang, 2006b), graphs (Osmari et al., 2014; Kugele and Antkowiak, 2016; Aboussoror et al., 2012), scatter plot (Osmari et al., 2014), map with labeled and colored boxes (Marcus et al., 2005), 3D skyline-like view (Antoniol et al., 2005), Use Case Maps extended with metadata availability annotations (Hassine and Hamou-Lhadj, 2014; Hassine and Gherbi, 2012; Hassine, 2011; 2015), and Aspect Oriented Use Case Maps (AoUCM) (Hassine et al., 2013).

In our context, we opted for a tabular representation to visualize trace analysis results. Different colors are used to distinguish different types of events and correlations, see Figs. 5(a) and (b) for examples of HSRP related results. A complete description of the produced tables can be found in Section 5.

3. Cisco Hot Standby Router Protocol (HSRP)

The increased demand for resilient and reliable data networks by Internet Service Providers (ISPs) led network device manufacturers to develop high availability features. One of the goals of high availability features is to minimize downtime and service interruption, through the elimination of single points of failure. Redundancy protocols (Pavlik et al., 2014), e.g., Hot Standby Router Protocol (HSRP), Virtual Router Redundancy Protocol (VRRP), Gateway Load Balancing Protocol (GLBP), etc., come as a solution to ensure high availability and to eliminate single points of failure.

Hot Standby Router Protocol (HSRP) (Cisco Systems, 2006) is the most widely used protocol for default gateway redundancy in enterprise data centers. HSRP was developed by Cisco and is documented in RFC2281 (Li et al., 1998) as "a protocol that provides a mechanism which is designed to support non-disruptive fail-over of IP traffic in certain circumstances" (Li et al., 1998). HSRP provides a mechanism for router/gateway failure detection and recovery. It allows data traffic to flow through the backup router/gateway following a failure of the primary gateway. By sharing a virtual IP address (must belong to the same subnet address in use on the

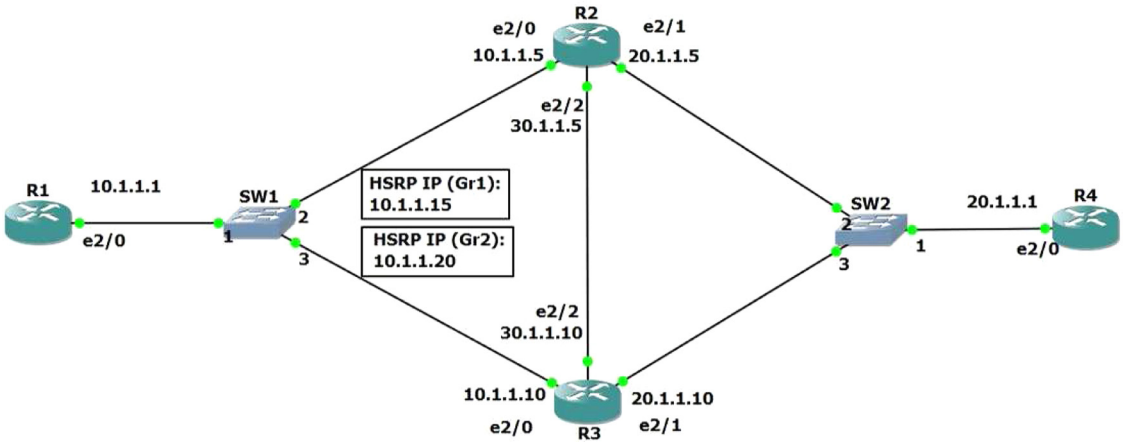


Fig. 3. Testbed topology 1.

LAN) and a MAC (Layer 2) address, two or more routers can act as a single "virtual" router, known as an HSRP group or a standby group. Within an HSRP group, only one single router is responsible for routing and forwarding packets that hosts send to the virtual router IP address. This router is called *active router*. Another router, called *standby router* is elected as a backup and is ready to take the role of the active router in case the latter fails. This role change is called " fail-over". If the standby router fails (in presence of more than 2 routers) or becomes the active router, then another router is elected as the standby router. Furthermore, HSRP has the ability to trigger a fail-over if the HSRP enabled interface on the router goes down. It is worth noting that a network may have multiple HSRP groups, each with a unique IP address. In the rest of the paper, we refer to an interface that belongs to router R, as "R-InterfaceName".

Fig. 3 illustrates a typical HSRP network topology having two HSRP enabled routers R2 (through interface E2/0) and R3 (through interface E2/0). R3-E2/0 represents the active node, while R2-E2/0 represents the standby node.[1] The HSRP group virtual IP is 10.1.1.15 (part of the subnet of R2-E2/0 and R3-E2/0). The traffic that flows from R1 to R4 is sent to the HSRP virtual IP address and HSRP selects the active node to send the traffic through it. If R3-E2/0 fails, router R2 becomes active and R2-E2/0 starts getting all traffic.

The HSRP election of the active and standby routers depends on two main elements, HSRP priority and IP address (Li et al., 1998). The HSRP member with the highest priority is elected as the active router and the second highest priority is elected as the standby router (Li et al., 1998). In case of equal priorities, the router with the highest IP address is elected as the active node. Furthermore, the configuration of *preempt* option (i.e., using "standby preempt" command), enables us to always know which node is active when all nodes are in-service.

HSRP uses two main timers: (1) *Hello interval time*: Interval between successive HSRP hello messages sent between the active and the standby nodes. Its default value is 3 seconds, and (2) *Hold interval time*: Interval between the reception of a hello message and the presumption that the sending router has failed. Its default value is 10 seconds. Another important HSRP feature is *delay timer*, which specifies the HSRP fail-over time. Generally, lower delay timer values helps achieve faster fail-over times. However, in case the failed active node recovers quickly, the *preempt* option would trigger another fail-over, which may take place before the convergence of the routing protocol(s), leading to traffic loss. One recommended practice is to configure the HSRP delay timer to postpone the preempt action until the stabilization of the routing protocol(s).

Each router in the HSRP group participates in the protocol by implementing a simple state machine (Li et al., 1998). These state machine states describe the router's current role and externally visible behavior. A state change is triggered by user configuration or by other HSRP group members state changes (Li et al., 1998). A router may be in one of these HSRP states:

1. Initial: This is the starting state and indicates that HSRP is not running, i.e., the router interface is not configured for HSRP.
2. Learn: The router has not determined the virtual IP address, and has not received *Hello* messages from the active router yet. The router does not participate in the election of active/standby nodes.
3. Listen: The router knows the virtual IP address, but is neither the active router nor the standby router. The router is only listening to *Hello* messages from those routers.
4. Speak: The router knows the virtual IP address, sends periodic Hello messages, and is actively participating in the election of the active and/or standby router.
5. Standby: The router is a candidate to become the next active router in case of failure of the active router. It sends periodic Hello messages.
6. Active: The router is currently forwarding packets that are sent to the group's virtual address. The router sends periodic Hello messages.

It is worth noting that these HSRP states are listed with respect to their level of involvement in HSRP behavior, i.e., Initial state is the lowest level (router is not involved) and Active being the highest level of involvement (router is forwarding the traffic). This ordering represents the basis of our correlation rules, introduced in Section 2 and described in details in Section 4.4.

For a detailed description of HSRP, the reader is referred to RFC 2281 (Li et al., 1998).

4. Applying the proposed approach to Cisco HSRP protocol

As a proof of concept, we apply our proposed approach to Cisco HSRP protocol (Cisco Systems, 2006).

4.1. HSRP Trace collection and filtering

The first step in our approach is to collect execution traces from Cisco routers that are involved in the HSRP setup. Traces can be collected from (1) a router console, (2) Terminal (same as console,

[1] In this paper, we use the term "node" to refer to the router or to the interface

but displays traces through router's VTY line) (3) router buffer (using a fixed size RAM storage), (4) SysLog Server (router sends the trace data to the syslog server), and (5) SNMP traps (router uses SNMP traps to send traces to an external SNMP server) (Cisco Support Community, 2009). In our experiments, we have collected traces using the routers consoles.

An example of a trace entry denoting an HSRP-related event is:

May 15 17:23:04.263: %HSRP-5-STATECHANGE: Ethernet2/0 Grp 1 state Standby -> Active

where the FACILITY is HSRP protocol, 5 is the severity denoting a *notification*, STATECHANGE represents the MNEMONIC, the message text describes the fact that interface Ethernet 2/0 part of HSRP group 1 has changes its state from Standby to Active.

In the rest of the paper, we will refer to "%FACILITY-severity-MNEMONIC" (introduced in Section 2.1) portion of a log entry as *Header*.

4.2. Trace filtering

Since we opted for a keyword-based filtering approach, the chosen keywords should fall within these two categories: (a) failure related keywords, and (b) recovery related keywords. For HSRP, a good candidate set of keywords, to be used as input to our filtering algorithm (also used as user input to *HAAnalyzer*, our HSRP-specific prototype tool), may include (1) the router interface status, e.g., LINK-CHANGED (status of the interface is down), LINK-UPDOWN (status of the interface is up), LINEPROTO-UPDOWN (change in the interface status to UP or DOWN), (2) HSRP interface status, e.g., HSRP-STATECHANGE (the HSRP state of a specific interface has changed), TRACKING-STATE (i.e., when tracking feature is configured, the status of the HSRP tracked interface has changed). In addition to these keywords, the analyst may provide time boundaries (i.e., lower and upper bound) to only retrieve trace entries that fall within a specific time period.

The trace filtering process is described in Algorithm 1 and consists of three steps:

1. **Step 1**: Parse the entire trace and exclude entries outside the user defined time boundaries, i.e., *startTimeBoundary* and *endTimeBoundary*. The *timeStamp* attribute of log entries are compared with the user provided time boundaries.
2. **Step 2**: Remove any entry that does not contain the user specified keywords. The attribute *Header* contains information about the event source, e.g., protocol name, process name, its severity, and its type. During the traversal of the resulting *Trace* array, if the header attribute of an entry contains one of the specified keywords, we add the corresponding entry to the temporary array *TempArr*, otherwise, we delete the entry.
3. **Step 3**: In the final step, we discard any entry that is not related to the HSRP involved groups. To do so, we traverse the *TempArr* from the beginning and we check whether every interface participates in HSRP (by checking the attribute *Header*). An interface that is related to HSRP is deleted from *TempArr*. Finally, the content of *TempArr* is written to *FilteredTrace.txt* file.

4.3. Trace merging

The output of the previous step is a set of filtered trace files, where each file corresponds to one router. In order to merge these traces (without loosing information about the trace source) and sort the resulting trace, we have created a table that contains the following fields:

1. *Router name*: refers to the name of the router.
2. *TimeStamp*: denotes the log entry timestamp.
3. *Head*: refers to "%FACILITY-severity-MNEMONIC" portion of the log entry.
4. *Interface*: denotes the interface name.
5. *Group*: refers to the HSRP group number, if any.
6. *Status change*: denotes a change in interface state (i.e., from UP to DOWN and vice versa), or a change in the HSRP state, (e.g., *Standby* to *Active*).

The table is populated using all entries of all filtered traces. The log entry *Message-text* is used to populate the interface name, group, and status change table fields. Finally, all entries are sorted according to their timestamps.

4.4. Execution trace segmentation

Before describing our HSRP specific segmentation procedure, we introduce the concept of *HSRP Upgrade* and *HSRP Downgrade*. As described in Section 3, HSRP protocol state machine has six states, ordered according to the high availability status of the interface. We assume that we have a state upgrade or downgrade, when we move from one state to another, as follows:

HSRP States:

Initial, Learn, Listen, Speak, Standby, Active

$\xrightarrow{\text{Upgrade}}$

$\xleftarrow{\text{Downgrade}}$

We formalize the concept of HSRP state *Upgrade* and *Downgrade* as follows:

Definition 1 (HSRP State Upgrade and Downgrade)**.** Let HSRPStates = [Initial, Learn, Listen, Speak, Standby, Active] be an array, composed of the six HSRP states.

We define the type of an HSRP state change as a function:

TypeHsrpSC: HSRPStates $\times$ HSRPStates $\rightarrow$ {"Upgrade", "Downgrade"}.

Given two HSRP states s1 and s2, TypeHsrpSC (s1,s2) is defined as:

$$\begin{cases} Upgrade, & \text{if } index(s1) < index(s2) \\ Downgrade, & \text{if } index(s1) > index(s2) \end{cases} \quad (1)$$

Algorithm 2 describes our segmentation procedure. It is based on the type of HSRP state change, i.e., whenever we encounter an HSRP state downgrade or upgrade, we create a new execution phase. A link failure is supposed to cause an HSRP state downgrade, while a recovery (part of a fail-over) is supposed to result in an HSRP state upgrade. The resulting phases are numbered sequentially.

Fig. 4 shows the result of the segmentation process when applied to traces collected from routers R2 and R3 (part of the network topology illustrated in Fig. 3). The filtered and merged traces are segmented into eight execution phases. For example, phase 2 starts with an HSRP upgrade of interface E2/0 in R2 (from *Standby* to *Active*), followed by phase 3 that describes another HSRP upgrade of interface E2/0 in R3 (from *Speak* to *Standby*).

4.5. Execution phase correlation

Before presenting our HSRP specific correlation rules, we define the following access functions, used to retrieve various log fields:

Definition 2 (Access functions)**.** Let E be the set of merged and segmented log entries, *interfaces* be the set of enabled interfaces, *phases* be the set of phase numbers.

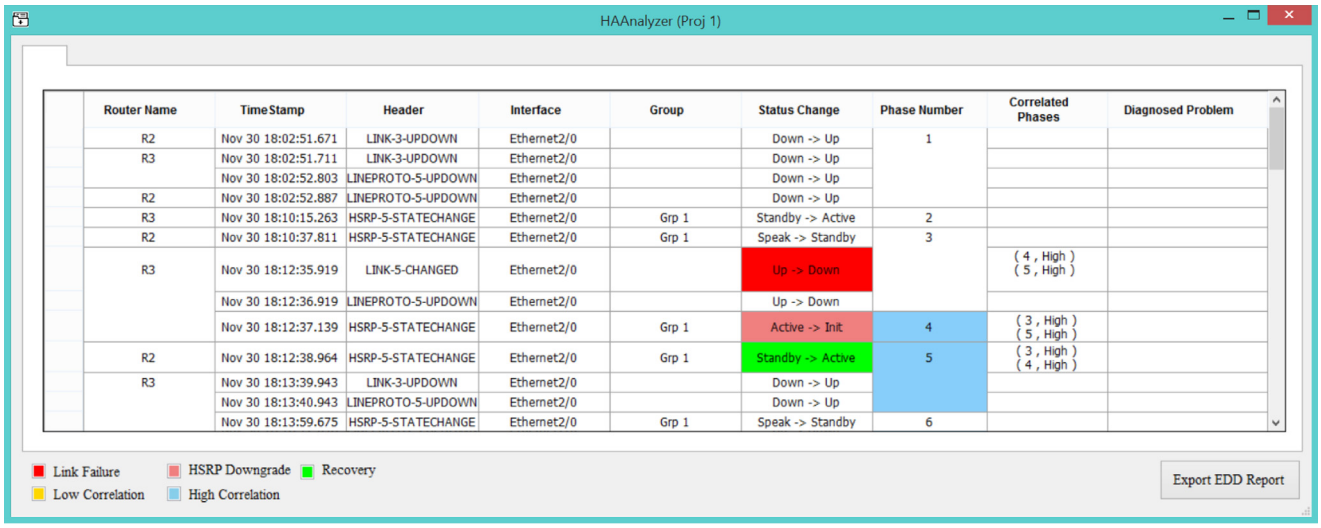
- **Interface**: E $\rightarrow$ interfaces. For a given e $\in$ E, *Interface (e)* returns the enclosed interface.

Router Name	TimeStamp	Header	Interface	Group	Status Change	Phase Number	Correlated Phases
R3	Nov 30 18:02:51.671	LINK-3-UPDOWN	Ethernet2/0		Down -> Up	1	
R2	Nov 30 18:02:51.711	LINK-3-UPDOWN	Ethernet2/0		Down -> Up		
	Nov 30 18:02:52.803	LINEPROTO-5-UPDOWN	Ethernet2/0		Down -> Up		
R3	Nov 30 18:02:52.887	LINEPROTO-5-UPDOWN	Ethernet2/0		Down -> Up		
R2	Nov 30 18:10:15.263	HSRP-5-STATECHANGE	Ethernet2/0	Grp 1	Standby -> Active	2	
R3	Nov 30 18:10:37.811	HSRP-5-STATECHANGE	Ethernet2/0	Grp 1	Speak -> Standby	3	
R2	Nov 30 18:12:35.919	LINK-5-CHANGED	Ethernet2/0		Up -> Down		(4 , High) (5 , High)
	Nov 30 18:12:36.919	LINEPROTO-5-UPDOWN	Ethernet2/0		Up -> Down		
	Nov 30 18:12:37.139	HSRP-5-STATECHANGE	Ethernet2/0	Grp 1	Active -> Init	4	(3 , High) (5 , High)
R3	Nov 30 18:12:38.964	HSRP-5-STATECHANGE	Ethernet2/0	Grp 1	Standby -> Active	5	(3 , High) (4 , High)
R2	Nov 30 18:13:39.943	LINK-3-UPDOWN	Ethernet2/0		Down -> Up		
	Nov 30 18:13:40.943	LINEPROTO-5-UPDOWN	Ethernet2/0		Down -> Up		
	Nov 30 18:13:59.675	HSRP-5-STATECHANGE	Ethernet2/0	Grp 1	Speak -> Standby	6	
	Nov 30 18:16:03.091	LINK-5-CHANGED	Ethernet2/0		Up -> Down		(7 , High)
	Nov 30 18:16:04.091	LINEPROTO-5-UPDOWN	Ethernet2/0		Up -> Down		
	Nov 30 18:16:04.207	HSRP-5-STATECHANGE	Ethernet2/0	Grp 1	Standby -> Init	7	(6 , High)
R3	Nov 30 18:16:11.046	LINK-5-CHANGED	Ethernet2/0		Up -> Down		(8 , High)
	Nov 30 18:16:12.046	LINEPROTO-5-UPDOWN	Ethernet2/0		Up -> Down		
	Nov 30 18:16:12.262	HSRP-5-STATECHANGE	Ethernet2/0	Grp 1	Active -> Init	8	(7 , High)
	Nov 30 19:28:20.367	LINK-3-UPDOWN	Ethernet2/0		Down -> Up		

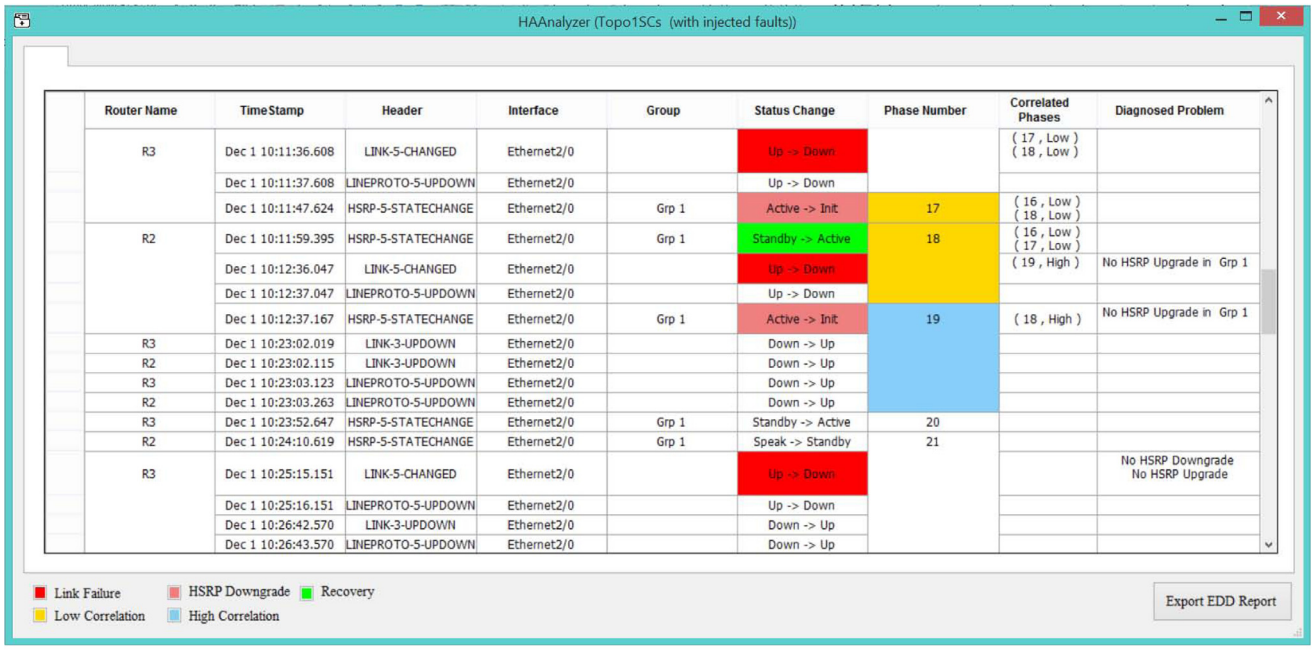
Fig. 4. Segmentation and Correlation of the merged traces of R2 and R3.

Algorithm 3: Trace Correlation Algorithm.

```
Procedure Name: CorrelateTrace
Input  : SegTrace   ▷ Segmented Trace + TemporalDistance
Output: Correlated Trace
string [ ][ ] Traces = Read(SegTrace);
int N = Traces.Length;
▷ Rule1 and 2 implementation
for (x=0 ; x <N ; x++) do
    ▷ Search for link failures
  if (Traces[x][Header].Contain("LINK-DOWN")) then
    string correlatedString =Empty;
    ▷ Search for HSRP downgrade
    for (frwd=x+1; frwd <N; frwd++) do
      if ((Traces[frwd][Router Name] ==
      Traces[x][Router Name]) AND
      (Traces[frwd][Interface] == Traces[x][Interface])
      AND (Traces[frwd][Status Change] ==
      Downgrade)) then
        ▷ Look for recovery
        correlatedString.Append
        (FindRecovery(Traces, frwd, x,
        TemporalDistance));
      end
    end
      ▷ Writing the correlation result
    if  (correlatedString ==Empty) then
      Traces[x][CorrelatedPhases]= "No Correlations" ;
    else
      Traces[x][CorrelatedPhases]= correlatedString;
    end
  end
end
▷ Rule3 implementation
for (x=0 ; x <N ; x++) do
  ▷ Search for HSRP downgrades
  if (Traces[x][Header].Contain("HSRP-5-STATECHANGE")
  AND Traces[x][Status Change] = Downgrade) then
    correlatedString =Empty;
    ▷ Search for HSRP upgrading
    for (frwd=x+1; frwd <N; frwd++) do
      if ((Traces[frwd][Router Name] != Traces[x]
      [Router Name]) AND (Traces[frwd][Group] ==
      Traces[x][Group]) AND (Traces[frwd][Status
      Change] == Upgrade)) then
        if (Traces[frwd][TimeStamp] -
        Traces[x][TimeStamp] <=TemporalDistance)
        then
          correlatedString.Append(Traces[frwd]
          [Phase] + "High");
        else
          correlatedString.Append(Traces[frwd]
          [Phase] + "Low");
        end
      end
    end
    ▷ Writing the result of correlation
    if  (correlatedString ==Empty) then
      Traces[x][CorrelatedPhases]= "No Correlations" ;
    else
      Traces[x][CorrelatedPhases]= correlatedString;
    end
  end
end
```



(a) HAAnalyzser output for a correct trace



(b) HAAnalyzser output for a trace with a seeded error

Fig. 5. HAAnalyzer input/output forms.

- **Phase**: E → phases. For a given e ∈ E, *Phase (e)* returns the phase number of *e*.

We have designed three correlation rules to track the following cause-effect relationships:

1. An interface failure should trigger an HSRP state downgrade of that specific interface.

Correlation Rule1:
if (∃ e1,e2 ∈ E | *e1*[Header]= LINK-5-CHANGED
AND *e1*[Status Change]=Up → Down
AND *e2*[Header] = HSRP-5-STATECHANGE
AND *TypeHsrpSC*(e2[Status Change]) = Downgrade
AND Interface(e1) = Interface (e2))
then
Correlation(Phase(e1), Phase(e2))

2. An interface failure should trigger an HSRP state upgrade of a different interface.

Correlation Rule2:
if (∃ e1,e2 ∈ E | *e1*[Header]= LINK-5-CHANGED
AND *e1*[Status Change]=Up → Down
AND *e2*[Header] = HSRP-5-STATECHANGE
AND *TypeHsrpSC*(e2[Status Change]) = Upgrade
AND Interface(e1) ≠ Interface (e2))
then
Correlation(Phase(e1), Phase(e2))

3. An HSRP state downgrade of an interface should trigger an HSRP state upgrade of another interface. The purpose of this third rule is to catch HSRP fail-overs that are not caused by interface failures, such as, decreasing/increasing the HSRP priority value of an interface.

Correlation Rule3:
if (∃ e1,e2 ∈ E | *e1*[Header]= HSRP-5-STATECHANGE
AND *TypeHsrpSC*(e1[Status Change])= Downgrade
AND *e2*[Header] = HSRP-5-STATECHANGE
AND *TypeHsrpSC*(e2[Status Change]) = Upgrade
AND Interface(e1) ≠ Interface (e2)
AND *e1*[Group] = *e2*[Group])
then
Correlation(Phase(e1), Phase(e2))

Algorithm 3 implements the three rules defined above. We start by looking for the HSRP-enabled failed interfaces (i.e., status change from UP to DOWN). Next, we look for both (1) HSRP downgrade of the same interface and (2) HSRP upgrade of a different interface. Searching for HSRP recovery is modeled as a separate algorithm (described in Algorithm 4). The correlation between an execution phase containing a failure and an execution phase containing its recovery (i.e., HSRP upgrade) is further characterized as "High" or "Low", based on the temporal distance between the two phases (i.e., the difference between the designated timestamps).

Algorithm 4: Find Recovery Algorithm.

```
Procedure Name: FindRecovery
Input  : Traces   ▷ Array + temp + original + Temporal Distance
Output: CorrelatedString
 ▷ Search for HSRP upgrade
for (frwd= temp +1; frwd <N; frwd ++) do
    if ((Traces[frwd][Router Name] != Traces[temp][Router Name]) AND (Traces[frwd][Interface] != Traces[temp][Interface]) AND (Traces[frwd][Group] == Traces[temp][Group]) AND (Traces[frwd][Status Change] == Upgrade)) then
        if (Traces[frwd][TimeStamp] - Traces[original][TimeStamp] <=TemporalDistance) then
            correlatedString.Append(Traces[frwd][Phase] + "High");
        else
            correlatedString.Append(Traces[frwd][Phase] + "Low");
        end
    end
end
return correlatedString;
```

Analysts may provide the temporal distance (in seconds) as input to our *HAAnalyzer* prototype tool. This information may be retrieved from HSRP router configuration, if available. If the recovery took place within the user-defined temporal distance, the correlation is considered as "High", otherwise it is considered as "Low".

Fig. 4 displays the correlations identified between the different execution phases of the segmented log. For example, there is a high correlation between phases 3, 4, and 5 as follows:

- Between phases 3 and 4, linking the failure (R2-E2/0 went from Up to down) to the HSRP downgrade (from Active to Init state, as a result of applying correlation rule1).
- Between Phases 3 and 5, linking the failure (R2-E2/0 went from Up to down) to the HSRP recovery (HSRP state upgrade from Standby to Active, as a result of applying correlation rule 2).
- Between Phases 4 and 5, linking the HSRP downgrade (from Active to Init) to the HSRP upgrade (from Standby to Active, as a result of applying correlation rule3).

4.6. Error detection and diagnosis

In the HSRP context, the correlation phase has three possible outcomes:

1. **Trace exhibits an HSRP wrong behavior**: The following situations can be diagnosed by our approach and *HAAnalyzer* tool:
   - **HSRP does not detect the failure and does not recover:** An HSRP-enabled interface goes down (i.e., failure) but HSRP does not produce neither an HSRP downgrade of the failed interface nor an HSRP upgrade of another interface within the same group. This is an indication that HSRP feature is not working at all. Given a correct HSRP fail-over trace and in order to mimic such situation, we keep the interface failure and we delete both HSRP downgrade and upgrade. No correlation rules should be triggered following the failure.
   - **HSRP does not detect the failure but a recovery takes place:** An HSRP-enabled interface goes down (i.e., failure) but HSRP fails to detect it, i.e., does not produce an HSRP downgrade of the failed interface. However, an HSRP upgrade takes place. This is an indication that HSRP is not working properly. For example, this may lead to two interfaces being active at the same time. Given a correct HSRP fail-over trace and in order to mimic such situation, we keep the interface failure, delete the HSRP downgrade, and keep the HSRP upgrade. Only Rule 2 should be triggered in this case.
2. **Trace exhibits a temporal problem**: During an HSRP fail-over the time taken by the standby interface to become active is greater than the normal/configured one. Temporal issues are identified as "Low" correlation between a failure and its recovery execution phases.
3. **Trace exhibits a correct behavior**: A correct HSRP scenario, where the system recovers from a failure within an acceptable period of time. That is, the phase that corresponds to the recovery is highly correlated with the phase having the failure.

5. HAAnalyzer: High Availability Analyzer Tool

As a proof of concept, we have built *HAAnalyzer*. The first prototype of our tool is HSRP-specific and aims to retrieve and analyze HSRP high availability scenarios. *HAAnalyzer* is a windows-based application, developed using Microsoft.Net C# language and Microsoft.Net Framework 4.5.

Figs. 5 (a) and (b) display the segmented and correlated trace in a tabular format. Different colors are used to distinguish different types of events and correlations. Link failures are colored in red, HSRP Downgrades are colored in light coral, and recovery-related entries are colored in green. High correlations are colored in light blue, while low correlations are colored in yellow. Fig. 5(a) illustrates the output for a correct trace, while Fig. 5(b) illustrates the output for a trace with seeded errors. The column "Diagnosed Problem" provides the details of a the encountered problem.

Trace entries and their related information are stored using a DataTable object, while the results are displayed using DataGridView control. In addition, the tool allows users to export the error detection and diagnosis (EDD) report to PDF format.

6. Empirical evaluation

The aim of our empirical evaluation is two-fold: (1) validate our approach through the design and execution of HSRP scenarios on

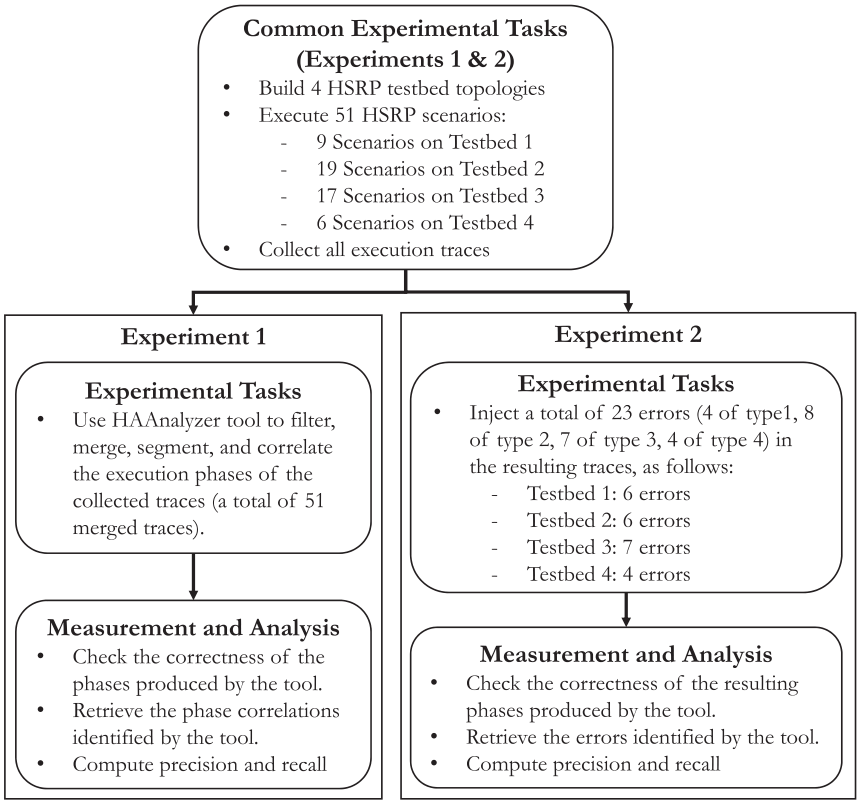


Fig. 6. Experimental design.

four real-world network topologies implementing the HSRP protocol and (2) measure the effectiveness and accuracy of the *HAAnalyzer* error detection and diagnosis capabilities. This goal is achieved by injecting different types of errors in execution traces and checking whether *HAAnalyzer* is able to unveil them.

6.1. Experiment setup

To evaluate our approach, we have used four testbed topologies covering typical HSRP network topologies. The testbeds are built using the Graphical Network Simulator 3 (GNS3) simulation software (GNS3 Technologies Inc., 2018). GNS3 can emulate complex networks since it has the ability to combine actual and virtual devices. In our context, GNS3 supports the Cisco IOS via the use of Dynamips (a Cisco IOS emulator).

Fig. 3 illustrates testbed topology1. The HSRP system consists of two nodes (R2-E2/0 and R3-E2/0) and two HSRP groups Gr1 and Gr2 (their virtual IP addresses are 10.1.1.15 and 10.1.1.20). The Ethernet link between R2-E2/2 and R3-E2/2 is used to synchronize the time between routers R1 and R2 using the Network Time Protocol (NTP). The full description of the four testbed topologies, scenarios, and results, are available online.[2]

6.2. Experiment procedure

Using the 4 testbed topologies, we have designed two experiments: (1) Experiment 1 aims to validate empirically our rule-based trace segmentation and correlation approach using 51 real scenarios (see Section 6.2.1), and (2) Experiment 2 aims to validate the fault detection and diagnosis technique and tool (see Section 6.2.2). Fig. 6 illustrates the main steps of our experimental plan.

[2] http://www.ccse.kfupm.edu.sa/~jhassine/JSS/report.pdf

Table 1
Scenario 2 executed of testbed network topology 1.

Scenario ID:	Topo1-Sc2
Preconditions:	1- Interfaces R2-E2/0 & R3-E2/0 are HSRP-enabled (no priority was specified). 2- R3-E2/0 (Active) & R2-E2/0 (Standby)
Scenario steps:	Shutdown the Active interface

6.2.1. Experiment 1

The first experiment consists of designing and executing a comprehensive set of HSRP related scenarios that mimic realistic network engineers' tasks, e.g., change HSRP configurations, shutdown/unshut interfaces, change IP addresses, etc. A total of 51 scenarios were executed on the four testbeds, as follows, 9 scenarios on testbed 1, 19 scenarios on testbed 2, 17 scenarios on testbed 3, and 6 scenarios on testbed 4. Each scenario is characterized by its ID, preconditions, and the executed steps. Table 1 describes scenario 2, executed on testbed topology 1. Due to the lack of space, we present all scenarios and results corresponding to testbed topology1 in Appendix A. The full description of all scenarios is available online [2].

After collecting all execution traces for the 51 scenarios, we have used *HAAnalyzer* to filter, merge, segment, and correlate the resulting execution phases. The results are described and discussed in Section 6.4.1.

6.2.2. Experiment 2

The second experiment consists of injecting 23 errors of different types (functional and temporal) in our 51 collected traces (from Experiment 1) and check whether our approach and tool can unveil them. Hence a total of 74 traces (51 correct + 23 faulty) were used in the this experiment. As stated in Section 2.5, we

Table 2
Distribution of the injected errors .

Testbed Topology	Type and number of injected errors			
	Type1	Type2	Type3	Type4
Topo1	1	2	2	1
Topo2	1	2	2	1
Topo3	1	3	2	1
Topo4	1	1	1	1
Total	4	8	7	4

Table 3
Result of execution of Scenario 2 on topology 1.

Scenario ID	Observed behavior	Rule	Phase correlations
Topo1-Sc2	R3-E2/0 is in Init state & R2-E2/0 is in Active.	Rule 1 Rule 2 Rule 3	(P3, P4, High) (P3, P5, High) (P4, P5, High)

have designed four types of errors that can be injected in a correct trace:

1. Behavioral errors, where we alter a correct fail-over trace by removing entries relative to HSRP failure detection, i.e., deleting the trace entries relative to HSRP downgrade (we refer to it as **Type1**).
2. Behavioral errors, where we alter a correct fail-over trace by removing both failure detection and recovery, i.e., deleting the trace entries relative to both HSRP downgrade and upgrade (we refer to it as **Type2**).
3. Temporal errors, where we manually change the timestamps to introduce a delay in the HSRP recovery, i.e., HSRP downgrade (we refer to it as **Type3**). All subsequent timestamps in the trace should also be delayed by the same amount to ensure the validity of the trace.
4. Altering a correct trace by duplicating failures and recoveries (we refer to it as **Type4**).

Table 2 describes the distribution of the 23 injected errors across the four testbed topologies.

6.3. Effectiveness measurement

In order to measure the effectiveness of our approach and our *HAAnalyzer* prototype tool, we use computed precision and recall, two metrics widely used in Information Retrieval (IR) (Buckland and Gey, 1994). Precision and recall are calculated based on a confusion matrix (Kohavi and Provost, 1998) showing the predicted (retrieved by *HAAnalyzer*) and the actual (analyzed by hand) classifications. Section 6.4 presents the confusion matrices of our empirical validation.

As defined in Buckland and Gey (1994), precision is the total number of correct predictions over the total number of retrieved items. It is formulated as follows:

$$Precision = \frac{TP\,(TruePositive)}{TP\,(TruePositive) + FP\,(FalsePositive)}$$

Recall is the total number of correct predictions over the total number of relevant items. It is formulated as follows:

$$Recall = \frac{TP\,(TruePositive)}{TP\,(TruePositive) + FN\,(FalseNegative)}$$

For the first experiment and in order to measure the ability of *HAAnalyzer* in finding the correct correlations in a given consolidated scenario trace, we have first analyzed the 51 consolidated scenario traces manually and extracted all existing correlations (i.e., actual correlations), then we have compared them to the ones generated by HAAnalyzer, if any. Each consolidated trace is classified in one of the following categories:

1. **True positives (TP)**: If a trace contains actual correlations between its execution phases, and *HAAnalyzer* reported them accurately (in terms of number and type, e.g., Low, High), then the trace is classified as TP. For example, the consolidated trace that corresponds to scenario Topo1-Sc2 (see Table 3) is a true positive trace since *HAAnalyzer* reported the correct number and type of correlations, i.e., three high correlations.
2. **True negatives (TN)**: If a trace does not contain any actual correlations and *HAAnalyzer* confirmed this fact, i.e., it did not report any, then the trace is classified as TN. For example, the consolidated trace that corresponds to scenario Topo1-Sc1 (see Table 5) is a true negative trace since *HAAnalyzer* did not report any correlations and our manual analysis did not identify any neither.
3. **False positives (FP)**: If a trace does not contain any actual correlations and *HAAnalyzer* showed the existence of correlations, i.e., fake correlations, then the trace is considered as FP.
4. **False negatives (FN)**: If a trace does contain actual correlations and *HAAnalyzer* fails to report them or did not identify them accurately, then the trace is considered as FN.

For the second experiment, we define:

1. **True positives (TP)**: If a trace contains actual errors (i.e., faulty trace) and *HAAnalyzer* diagnosed the injected faults accurately, then the trace is classified as TP.
2. **True negatives (TN)**: If a trace does not contain actual errors (i.e., valid trace) and *HAAnalyzer* did not report any error, then the trace is classified as TN.
3. **False positives (FP)**: If a trace does not contain actual errors (i.e., valid trace) and *HAAnalyzer* identified the existence of errors, then the trace is classified as FP.
4. **False negatives (FN)**: If a trace contains actual errors (i.e., faulty trace) and *HAAnalyzer* failed to identify these errors (i.e., valid trace), then the trace is classified as FN.

Table 4
Number of traces with and without correlations .

Testbed Topology	Traces with correlations	Traces without correlations
Topo1	6	3
Topo2	11	8
Topo3	14	3
Topo4	6	0
Total	37	14

6.4. Experiment results

In this section, we present and discuss the results of both experiments.

6.4.1. Experiment 1 results

Table 3 shows a sample result of the execution of scenario Topo1-Sc2 (described in Table 1) on topology 1. The three rules were triggered leading to three high correlations between phases P3, P4, and P5. Table 7, in Appendix A, shows the results corresponding to testbed 1. As expected, out of 9 resulting traces, 3 did not present any correlations (Topo1-Sc1, Topo1-Sc3, and Topo1-Sc6), while the remaining 6 traces showed the execution of at least Rule 1. Results show that *HAAnalyzer* successfully segmented and correlated all 51 traces. The full description of results is available online [2].

Table 4 provides the number of traces showing correlations vs. traces without correlations, for the four testbeds.

Table 5
Comparison with related work .

Comparison criteria	This paper	Hassine and Hamou-Lhadj (2014)	Hassine et al. (2018)
Targeted HA Features	HSRP (Cisco Systems, 2006)	HSRP (Cisco Systems, 2006)	(1) HSRP (Cisco Systems, 2006) (2) Cisco ASA firewall active/standby redundancy (Deal, 2009) (3) Cisco ASR 9000 RSP stateful-switchover (Cisco Systems, 2016) (4) Cisco IOS-XRv process restartability (Maier et al., 2002)
Log nature (native, pre-processed, user instrumented)	Native	Native	Native
Trace filtering approach	Keyword-based (formally defined)	Keyword-based (informal)	Keyword-based (formally defined)
Trace segmentation and correlation	Rule-based (formally defined)	Rule-based (informal)	Rule-based (informal)
Execution phases	failure occurrence, failure detection, failure recovery	Not specified	User-related, HA-related, Other
Visualization	Tabular format (with colors)	Use Case Maps (UCM) (ITU-T, 2012)	Use Case Maps (UCM) (ITU-T, 2012)
Empirical evaluation	4 HSRP case studies (51 scenarios)	One simple HSRP scenario	4 scenarios (one scenario for each HA feature)
HA Anomaly detection capabilities	Yes	No	No
Identification of failure detection and recovery	Yes	No	No
Tool support	HAAnalyzer	None	Tool integrated with jUCMNav framework (jUCMNav, 2019)
Generalization	No (i.e., HSRP specific)	Not specified	Yes (but requires empirical evidence)

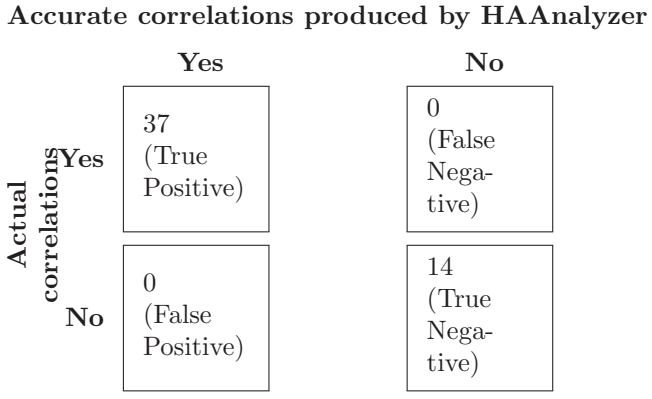


Fig. 7. Experiment 1 confusion matrix .

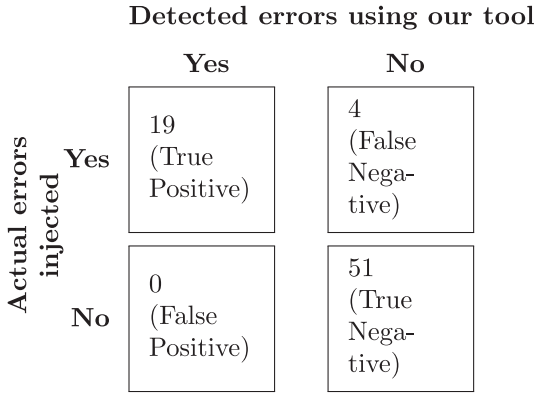


Fig. 8. Experiment 2 confusion matrix.

Based on the confusion matrix in Fig. 7, we have obtained a precision of 100% (i.e., 37 (TP)/(37 (TP) + 0 (FP))) and a recall of 100% (i.e., 37 (TP)/(37 (TP) + 0 (FN))). Results show an excellent effectiveness of *HAAnalyzer* in retrieving and analyzing HSRP scenarios.

6.4.2. Experiment 2 results

The 51 valid scenarios from the first experiment were classified as correct by *HAAnalyzer*; hence classified as true negative cases. As per the 23 injected errors, 19 were caught by *HAAnalyzer*. Based on the confusion matrix in Fig. 8, we have obtained a precision of 100% (i.e., 19 (TP)/(19 (TP)+ 0 (FP))) and a recall of 82.6% (i.e., 19 (TP)/(19 (TP)+ 4 (FN))). The four false negative cases correspond to failures of type 4, considered as correct by our tool. The tool showed high correlations between the injected failure (duplicate from the original one) and the unique recovery. This limitation is due to the fact that the triggering conditions of our three rules were satisfied when processing the newly injected failure (duplicated failure). However, overall, the obtained results were very satisfactory in detecting and diagnosing errors introduced in HSRP scenarios.

7. Discussion

In this section, we will discuss the benefits and potential threats to validity of our approach and prototype tool.

7.1. Comparison with related work

There is a large body of research on applying dynamic analysis (Ball, 1999) to recover high-level views from execution traces (Cornelissen et al., 2009a). Most of the existing approaches focus on the retrieval of system functional aspects and less work was devoted to dependability attributes and more particularly to availability (Hassine and Hamou-Lhadj, 2014; Hassine et al., 2018).

In an early work, Hassine and Hamou-Lhadj (2014) proposed a preliminary approach to recover availability requirements from native execution traces. The resulting availability scenarios were visualized using the ITU-T standard Use Case Maps (UCM) language (ITU-T, 2012) extended with metadata availability annotations (Hassine and Gherbi, 2012; Hassine, 2011). In Hassine and Hamou-Lhadj (2014), native logs are first filtered, then segmented and visualized. However, very little details regarding the filtering,

segmentation, and visualization procedures were provided (no formal description of these steps were presented). To illustrate their proposed approach, the authors (Hassine and Hamou-Lhadj, 2014) used one simple and small HSRP scenario. Hence, their approach cannot be generalized on the basis of a single scenario. Furthermore, no analysis of the produced UCM scenarios was conducted. Moreover, their approach (Hassine and Hamou-Lhadj, 2014) was not automated.

Later, Hassine et al. (2018) proposed a framework that aims to recover high availability scenarios from execution traces. The framework consists of four main steps: log filtering, log merging, log segmentation, and HA scenario visualization. The execution traces are first loaded in a MySQL database then filtered, using a keyword-based approach, to extract only HA feature-related entries. Next, the filtered traces are merged and segmented into three types of execution phases: (1) User (grouping user-related events), (2) HA (grouping HA-related events), and (3) Other (events that are neither classified as User nor as HA). Similar to the approach presented in Hassine and Hamou-Lhadj (2014), the resulting phases are visualized as UCM models extended with metadata availability annotations (Hassine and Gherbi, 2012; Hassine, 2011). The produced UCMs are composed of two UCM path types: (i) normal paths that consist of execution phases representing non HA-related behavior, i.e., phases are classified as User or Other, and (ii) exception paths representing failures and recovery events, i.e., phases classified as HA). Although, the approach is simple (since only two UCM path types were considered, i.e., HA and non-HA paths), it fails to provide any insights about how an HA feature detects and recovers from a failure. Indeed, in case of occurrence of a failure, trace entries related to the HA feature failure detection and recovery are enclosed within the same execution phase. Consequently, they are visualized as part of the same UCM HA exception path. Hence, an analyst will not be able to dissociate the failure from recovery. In our proposed approach, we provide a fine-grained HA analysis framework that distinguishes between failure occurrence, failure detection, and failure recovery, allowing for reasoning about how HSRP reacts to a failure. Furthermore, our framework allows for the detection and diagnosis of errors that may occur in the logs, e.g., due to flawed implementation, inadequate feature configuration, or incomplete logs.

The framework, introduced in Hassine et al. (2018), is supported by a tool incorporated within the jUCMNav framework (jUCMNav, 2019). The applicability of the approach was demonstrated using four scenarios produced from the execution of four high availability features, namely, HSRP (Cisco Systems, 2006), Cisco Adaptive Security Appliance (ASA) firewall active/standby redundancy (Deal, 2009), Cisco Route Switch Processor (RSP) stateful switchover (Cisco Systems, 2016), and Automatic Protection Switching (APS) (Maier et al., 2002). The experimental validation using four different case studies shows that the technique can be generalized to many high availability features. However, only one single scenario of each HA feature is used. Hence, the approach cannot be generalized without conducting an empirical study with a large number of scenarios. In our empirical validation, we have used a total of 51 real scenarios covering 4 typical HSRP network topologies. Furthermore, another drawback of the approach presented in Hassine et al. (2018) is that the analyst must be familiar with the UCM notation in order to understand the produced output. In our technique, we adopt an easy to understand tabular format to visualize the output of the segmentation step, i.e., execution phases. In addition, the produced table is enriched with colors to distinguish log entries denoting failures (e.g., in red) from the ones presenting recoveries (e.g., in green), and to distinguish high from low correlations between execution phases (high correlation is colored in blue while a low correlation is colored in yellow). Therefore, an analyst can easily comprehend various HA scenarios. Furthermore, a brief anomaly diagnosis is also provided in case of unexpected behavior, e.g., no failure recovery. Table 5 summarizes the comparison with the approaches introduced in Hassine and Hamou-Lhadj (2014) and Hassine et al. (2018).

7.2. Benefits of the approach

Our proposed approach has the following benefits:

1. It addresses the problem of recovering high availability scenarios from execution traces. Indeed very little research was devoted to applying dynamic analysis to availability (Hassine and Hamou-Lhadj, 2014; Hassine et al., 2018).
2. It offers a systematic way to filter, segment, and visualize high availability scenarios using an easy to comprehend tabular format, showing the different execution phases along with their correlations. Colors are used to facilitate the identification and categorization of the retrieved correlations.
3. Our proposed correlation rules allow for the detection and diagnosis of both functional and temporal errors in system execution traces.
4. Our approach does not assume the availability of neither the source code of the HA feature under analysis, nor the HA feature configuration. Our technique uses as input native logs without any prior code instrumentation.
5. Our empirical evaluation demonstrated the effectiveness of our approach and prototype tool in retrieving and analyzing HSRP scenarios.
6. Although our *HAAnalyzer* tool is tailored to Cisco HSRP feature, it can be extended with no significant effort to support other Layer 3 (network layer) HA features, such as Virtual Router Redundancy Protocol (VRRP) and Gateway Load Balancing Protocol (GLBP). Furthermore, minimal adaption might be required to support other non-Cisco HA features.

7.3. Threats to validity

Our approach, prototype tool (*HAAnalyzer*), and the empirical evaluation are subject to several limitations and threats to validity, categorized here according to three important types of threats identified by Wright et al. (2010).

In terms of *construct Validity*, one possible risk is the scalability of the proposed approach in handling large execution traces. Indeed, large traces would make execution phases size larger, but does not impact their number, since our segmentation approach uses HSRP upgrade and downgrade as cut-off points. Having large-sized phase would affect the ability of the analyst to analyze the produced correlations. However, this risk can be mitigated by using an effective filtering (i.e., through the selection of a relevant and minimal set of keywords). In addition, the use of colors to distinguish execution phases describing failures (i.e., colored in red) from recovery (i.e., colored in green), and between High and Low correlations, would help minimize this risk. A second possible threat is with respect to our empirical evaluation of the error detection and diagnosis capability. To evaluate our error detection diagnosis capabilities, we have chosen a controlled injection of errors. We believe that having a random error-seeding would create an unrealistic, and may be erroneous log. Using such logs in our analysis would lead to erroneous and fictitious correlations; hence wrong diagnosis.

Another criticism about our empirical evaluation is that we haven't used an industrial case study. Besides the difficulty and hassle of obtaining real traces from an ISP production networks, we believe that our produced traces were very effective in covering a large number of HSRP scenarios. Indeed, we have used 4 different network topologies and executed 51 different scenarios. In

addition, we have tried to load our HSRP participating routers with many features. Furthermore, any additional log entries that are generated from additional features, would be discarded from the trace as a result of the filtering phase. Finally, the conducted empirical evaluation showed that our approach and tool are very effective, in retrieving and analyzing HSRP scenarios, achieving 100% precision and recall for the first experiment, 100% precision and 82.6% recall when it comes to error detection. However, no similar tools are available to compare with.

In terms of *internal validity*, there is a risk of using partial traces, which may impact negatively the accuracy of our computed correlations. Indeed, for example, missing recovery log entries after the occurrence of a failure may lead to the wrong conclusion about the failure handling. To avoid such a risk, a careful check of the completeness of the produced log is necessary. A related possible threat is that our approach requires the availability of traces from all routers participating in the HA scenario. Failing to provide all logs would hinder the obtained results. Therefore, the analyst should know all the interacting components and collect their traces before proceeding with the analysis.

Our approach requires prior domain knowledge in order to be able to choose the appropriate HA feature keywords and to be able to analyze and understand the produced results. However, the use of inaccurate keywords or missing important keywords represents another threat and may lead to wrong results. To mitigate this threat, a set of relevant keywords may be provided to help the analyst.

In terms of *external validity*, merging execution traces from different participating network nodes from different vendors may represent a potential risk. Indeed, each vendor has its own logging format and constraints of device configurations. So many discrepancies may arise; such as, event logger priority (i.e. we can find many devices with different priorities) and time references. These issues may lead to merged trace with incorrect chronological order of entries. Another threat may arise if we try to generalize our approach to cover other Cisco high availability features like VRRP and GLBP (since the log format is the same as HSRP). This risk can be mitigated if we carefully choose the set of keywords (used in the filtering step) for the new feature. In addition, our designed correlation rules would require modifications to accommodate the new features.

8. Conclusion and future work

In this paper, we have proposed a fully automated approach to retrieve and analyze high availability features from system execution traces. The proposed approach helps analysts and maintainers understand how failures are handled by high availability features, prior to a maintenance task. Execution traces are collected from systems running high availability features, then filtered and segmented into execution phases separating failure related events from recovery related events. A rule-based system was designed to identify the correlations between the resulting execution phases. These correlations are visualized using an easy to comprehend tabular format decorated by colors to distinguish failures from recovery and to categorize correlations into “High” and “Low”. Furthermore, our proposed approach allows for the detection and diagnosis of both functional and temporal errors that a trace may contain. As a proof of concept, we have implemented our approach in a prototype tool, called *HAAnalyzer*, that is tailored to Cisco Hot Standby Router Protocol (HSRP) high availability feature. We have evaluated empirically our approach and tool by conducting two experiments.

For future work, we intend to improve the capabilities of *HAAnalyzer* by proposing and maintaining a set of HSRP relevant keywords that analysts can choose from and by offering other forms of visualization. In addition, we plan to extend our *HAAnalyzer* tool by creating profiles for other high availability features, such as VRRP and GLBP.

Authors Contributions

Mr. Maged Sheghdara is an MS Student who completed his thesis under the supervision of Dr. Jameleddine Hassine. Maged Sheghdara developed and implemented the proposed approach. He contributed mainly in the development and testing of the HAAnalyzer tool (Section 5). In addition, he was responsible of the empirical validation part (data collection and analysis). He participated in the write up of all sections of the initial version of the paper. Jameleddine Hassine is the principal investigator. He initiated the research idea and supervised all phases of the work (formal definition of the approach, data interpretation and analysis). He participated in the write up of all sections of the paper. Jameleddine Hassine was responsible for implementing all changes requested by the reviewers for both revisions.

Declaration of Competing Interests

None.

CRediT authorship contribution statement

Maged Sheghdara: Data curation, Formal analysis, Investigation, Software, Validation, Writing - original draft. **Jameleddine Hassine:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Writing - original draft, Writing - review & editing, Supervision, Validation.

Acknowledgment

The authors would like to acknowledge the support provided by the Deanship of Scientific Research at KFUPM for funding this work through project No. IN131031.

Appendix A. Testbed Topology 1 Scenarios and Results

Table 6 describes a total of 9 scenarios executed on testbed topology 1.

Table 7 shows the results of the executed scenarios on testbed topology 1. For each executed scenario, we describe the observed behavior, specify the executed rules and present the produced correlations. Scenarios Topo1-Sc1, Topo2-Sc1, Topo2-Sc8, and Topo2-Sc17 did not trigger any rule; hence did not produce any correlations between execution phases. Indeed, in these scenarios, we start with all HSRP-enabled interfaces in shutdown state. Bringing these interfaces to UP state does neither cause any interface failure nor produce an HSRP downgrade. Therefore, the observed behavior is correct. Scenarios Topo1-Sc3, Topo2-Sc3, and Topo2-Sc5 did not trigger any rule; hence did not produce any correlations between execution phases. Indeed, in all these scenarios the HSRP group consists of only two nodes and the preempt property was not configured. Unshuting the down interface, will move it to Standby state, which does not cause any HSRP downgrade. Therefore, the observed behavior is correct. Scenario Topo1-Sc6 scenario does not produce any HSRP-related events; hence no phase correlations were produced. Indeed, neither an HSRP upgrade nor an HSRP downgrade took place when we change the HSRP IP address. Therefore, the observed behavior is correct.

In Scenarios Topo1-Sc7 and Topo1-Sc8, we have changed the hold time to 25 seconds while the user-defined temporal distance was 10 seconds. Therefore, the HSRP protocol has changed the

Table 6
Testbed topology 1: HSRP scenarios descriptions .

Scenario ID	Preconditions	Scenario steps
Topo1-Sc1	1- Interfaces R2-E2/0 & R3-E2/0 are HSRP-enabled (no priority was specified). 2- Both interfaces R2-E2/0 & R3-E2/0 are in shutdown state.	Unshut both interfaces.
Topo1-Sc2	1- Interfaces R2-E2/0 & R3-E2/0 are HSRP-enabled (no priority was specified). 2- R3-E2/0 (Active) & R2-E2/0 (Standby)	Shutdown the Active interface
Topo1-Sc3	1- Interfaces R2-E2/0 & R3-E2/0 are HSRP-enabled (no priority was specified) 2- R3-E2/0 (Init) & R2-E2/0 (Active)	Unshut R3-E2/0
Topo1-Sc4	1- Interfaces R2-E2/0 & R3-E2/0 are HSRP-enabled. 2- R3-E2/0 (Active) & R2-E2/0 (Standby)	Shutdown the Standby interface
Topo1-Sc5	1- Interfaces R2-E2/0 & R3-E2/0 are HSRP-enabled. 2- R3-E2/0 (Active) & R2-E2/0 (Init)	Shutdown the Active interface
Topo1-Sc6	1- Interfaces R2-E2/0 & R3-E2/0 are HSRP-enabled. 2- HSRP IP is 10.1.1.15/24.	Change the HSRP IP at runtime to 10.1.1.50
Topo1-Sc7	1- Interfaces R2-E2/0 & R3-E2/0 are HSRP-enabled. 2- The hello time is changed to 20 seconds and the hold time to 25 seconds. 3- R2-E2/0 (Standby) & R3-E2/0 (Active)	Shutdown the Active interface
Topo1-Sc8	1- Interfaces R2-E2/0 & R3-E2/0 are HSRP-enabled. 2- The hello time is changed to 3 seconds and the hold time to 25 seconds. 3- R2-E2/0 (Standby) & R3-E2/0 (Active)	Shutdown the Active interface
Topo1-Sc9	1- Two HSRP groups Gr1 & Gr2 sharing the same interfaces R2-E2/0 & R3-E2/0. 2- R3-E2/0 is the Active node of Gr1 and the Standby node of Gr2. 3- R2-E2/0 is the Active node of Gr2 and the Standby node of Gr1.	Shutdown the Active node of Gr1

Table 7
Testbed topology 1: HSRP scenarios results .

Scenario ID	Observed behavior	Rule	Phase correlations
Topo1-Sc1	R3-E2/0 is in Active state & R2-E2/0 is in Standby state.	–	–
Topo1-Sc2	R3-E2/0 is in Init state & R2-E2/0 is in Active.	Rule 1	(P3, P4, High)
		Rule 2	(P3, P5, High)
		Rule 3	(P4, P5, High)
Topo1-Sc3	R3-E2/0 is in Standby state.	–	–
Topo1-Sc4	R2-E2/0 is in Init state	Rule 1	(P4, P5, High)
Topo1-Sc5	R3-E2/0 is in Init state	Rule 1	(P5, P6, High)
Topo1-Sc6	No HSRP-relevant events.	–	–
Topo1-Sc7	R3-E2/0 is in Init state & R2-E2/0 is in Active state after 25 seconds.	Rule 1	(P3, P4, High)
		Rule 2	(P3, P5, Low)
		Rule 3	(P4, P5, Low)
Topo1-Sc8	R3-E2/0 is in Init state & R2-E2/0 is in Active state after 25 seconds.	Rule 1	(P3, P4, High)
		Rule 2	(P3, P5, Low)
		Rule 3	(P4, P5, Low)
Topo1-Sc9	1- R3-E2/0 is in Init state for Gr1 & Gr2.	Rule 1	(P5, P6, High)
	2- R2-E2/0 is in Active state for Gr1.	Rule 1	(P5, P7, High)
		Rule 2	(P5, P8, High)
		Rule 3	(P6, P8, High)

status of the standby interface into active after 25 seconds. Hence, a total of four correlations (two in each scenario) are marked as “Low”. Therefore, the observed behavior is correct.

References

Aboussoror, E.A., Ober, I., Ober, I., 2012. Seeing errors: model driven simulation trace visualization. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (Eds.), Model Driven Engineering Languages and Systems: 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30–October 5, 2012. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 480–496. doi:10.1007/978-3-642-33666-9_31.

Aguilar, X., Fürlinger, K., Laure, E., 2015. Automatic on-line detection of mpi application structure with event flow graphs. In: Träff, J.L., Hunold, S., Versaci, F. (Eds.), Euro-Par 2015: Parallel Processing. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 70–81.

Antoniol, G., Merlo, E., Guéhéneuc, Y.-G., Sahraoui, H., 2005. On feature traceability in object oriented programs. In: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering. ACM, New York, NY, USA, pp. 73–78. doi:10.1145/1107656.1107672.

Asadi, F., Penta, M.D., Antoniol, G., Gueheneuc, Y.G., 2010. A heuristic-based approach to identify concepts in execution traces. In: 2010 14th European Conference on Software Maintenance and Reengineering, pp. 31–40. doi:10.1109/CSMR.2010.17.

Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C., 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secure Comput. 1 (1), 11–33.

Ball, T., 1999. The concept of dynamic analysis. SIGSOFT Softw. Eng. Notes 24 (6), 216–234. doi:10.1145/318774.318944.

Buckland, M., Gey, F., 1994. The relationship between recall and precision. Journal of the American Society for InformationScience 45 (1), 12. Last updated - 2013-02-24

Cisco Systems, 2006. Hot Standby Router Protocol Features and Functionality. https://www.cisco.com/c/en/us/support/docs/ip/hot-standby-router-protocol-hsrp/9234-hsrpguidetoc.pdf. Last accessed, November 2019.

Cisco Support Community, 2009. How to configure logging in Cisco IOS. [Online; accessed 25-October-2018] https://supportforums.cisco.com/t5/network-infrastructure-documents/how-to-configure-logging-in-cisco-ios/ta-p/3132434/.

Cisco Systems, 2004. Internetworking Technologies Handbook. Cisco Press. http://books.google.com.sa/books?id=3Dn9KIIVM_EC

Cisco Systems, 2016. High availability and redundant operation. http://www.cisco.com/c/en/us/td/docs/routers/asr9000/hardware/overview/guide/asr9kOVRGbk/asr9kOVRGHAredundancy.pdf. Last accessed, Sep 2019.

Corbi, T.A., 1989. Program understanding: challenge for the 1990s. IBM Syst. J. 28 (2), 294–306. doi:10.1147/sj.282.0294.

Cornelissen, B., Zaidman, A., van Deursen, A., 2011. A controlled experiment for program comprehension through trace visualization. IEEE Trans. Softw. Eng. 37 (3), 341–355. doi:10.1109/TSE.2010.47 .

Cornelissen, B., Zaidman, A., van Deursen, A., Moonen, L., Koschke, R., 2009. A systematic survey of program comprehension through dynamic analysis. IEEE Trans. Softw. Eng. 35 (5), 684–702. doi:10.1109/TSE.2009.28.

Cornelissen, B., Zaidman, A., van Deursen, A., van Rompaey, B., 2009. Trace visualization for program comprehension: acontrolled experiment. In: 2009 IEEE 17th International Conference on Program Comprehension, pp. 100–109. doi:10.1109/ICPC.2009.5090033.

Cornelissen, B., Zaidman, A., Holten, D., Moonen, L., van Deursen, A., van Wijk, J.J., 2008. Execution trace analysis through massive sequence and circular bundle views. J. Syst. Softw. 81 (12), 2252–2268. doi:10.1016/j.jss.2008.02.068.

Cornelissen, B., Zaidman, A., Van Deursen, A., Van Rompaey, B., 2009. Trace visualization for program comprehension: acontrolled experiment. In: Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on. IEEE, pp. 100–109.

Deal, R., 2009. Cisco ASA Configuration, 1 McGraw-Hill, Inc., New York, NY, USA.

Duan, C., Cleland-Huang, J., 2006. Visualization and analysis in automated trace retrieval. Requirements Engineering Visualization, 2006. REV'06. First International Workshop on. IEEE. 5–5

Duan, C., Cleland-Huang, J., 2006. Visualization and analysis in automated trace retrieval. 2006 First International Workshop on Requirements Engineering Visualization (REV'06 - RE'06 Workshop) doi:10.1109/REV.2006.6. 5–5

Fittkau, F., Finke, S., Hasselbring, W., Waller, J., 2015. Comparing trace visualizations for program comprehension through controlled experiments. In: 23rd IEEE International Conference on Program Comprehension, pp. 266–276. doi:10.1109/ICPC.2015.37.

GNS3 Technologies Inc., 2018. Graphical Network Simulator, GNS3. http://www.gns3.com. Last accessed, July 2018.

Gokhale, S.S., Crigler, J.R., Farr, W.H., Wallace, D.R., 2005. System availability analysis considering hardware/software failure severities. In: 29th Annual IEEE/NASA Software Engineering Workshop, pp. 47–56. doi:10.1109/SEW.2005.43.

Gonzalez, J., Gimenez, J., Labarta, J., 2009. Automatic detection of parallel applications computation phases. In: IEEE International Symposium on Parallel Distributed Processing, pp. 1–11. doi:10.1109/IPDPS.2009.5161027.

Günther, C.W., Rozinat, A., van der Aalst, W.M.P., 2010. Activity mining by global trace segmentation. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (Eds.), Business Process Management Workshops. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 128–139.

Hamou-Lhadj, A., Lethbridge, T., 2006. Summarizing the content of large traces to facilitate the understanding of the behaviour of a software system. In: 14th IEEE International Conference on Program Comprehension (ICPC'06), pp. 181–190. doi:10.1109/ICPC.2006.45.

Hamou-lhadj, A., Lethbridge, T.C., 2003. Techniques for reducing the complexity of object-oriented execution traces. In: Proceedings of the 2nd Annual Designfest on Visualizing Software for Understanding and Analysis. Citeseer, pp. 35–40.

Hamou-Lhadj, A., Lethbridge, T.C., 2005. Measuring various properties of execution traces to help build better trace analysis tools. In: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2005), pp. 559–568. doi:10.1109/ICECCS.2005.57.

Hamou-Lhadj, A., Lethbridge, T.C., Fu, L., 2004. Challenges and requirements for an effective trace exploration tool. In: Proceedings of the 12th IEEE International Workshop on Program Comprehension, pp. 70–78. doi:10.1109/WPC.2004.1311049.

Hassine, J., 2011. Early availability requirements modeling using Use Case Maps. In: Eighth International Conference on Information Technology: New Generations (ITNG 2011), pp. 754–759. doi:10.1109/ITNG.2011.133.

Hassine, J., 2015. Describing and assessing availability requirements in the early stages of system development. Softw. Syst. Model. 14 (4), 1455–1479. doi:10.1007/s10270-013-0382-0.

Hassine, J., Gherbi, A., 2012. Exploring early availability requirements using Use Case Maps. In: Ober, I., Ober, I. (Eds.), SDL 2011: Integrating System and Software Modeling. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 54–68.

Hassine, J., Hamou-Lhadj, A., 2014. Toward a UCM-based approach for recovering system availability requirements from execution traces. In: Amyot, D., Fonseca i Casas, P., Mussbacher, G. (Eds.), System Analysis and Modeling: Models and Reusability: 8th International Conference, SAM 2014, Valencia, Spain, September 29–30, 2014. Proceedings. Springer International Publishing, Cham, pp. 48–63. doi:10.1007/978-3-319-11743-0_4.

Hassine, J., Hamou-Lhadj, A., Alawneh, L., 2018. A framework for the recovery and visualization of system availability scenarios from execution traces. Inf. Softw. Technol. 96, 78–93. doi:10.1016/j.infsof.2017.11.007.

Hassine, J., Mussbacher, G., Braun, E., Alhaj, M., 2013. Modeling early availability requirements using aspect-oriented use case maps. In: SDL 2013: Model-Driven Dependability Engineering - 16th International SDL Forum, Montreal, Canada, June 26–28, 2013. Proceedings, pp. 54–71. doi:10.1007/978-3-642-38911-5_4.

ITU-T, 2012. Recommendation Z.151 (10/12), User Requirements Notation (URN) language definition, Geneva, Switzerland. http://www.itu.int/rec/T-REC-Z.151/en

Jalote, P., 1994. Fault Tolerance in Distributed Systems. Prentice-Hall, Inc..

jUCMNav, v7.0.0, http://softwareengineering.ca/jucmnav. University of Ottawa, Canada, last Accessed Jan 2019.

Kohavi, R., Provost, F., 1998. Confusion matrix. Mach. Learn. 30 (2–3), 271–274 .

Koskimies, K., Mossenbock, H., 1996. Scene: using scenario diagrams and active text for illustrating object-oriented programs. In: Proceedings of IEEE 18th International Conference on Software Engineering, pp. 366–375. doi:10.1109/ICSE.1996.493431.

Kugele, S., Antkowiak, D., 2016. Visualization of trace links and change impact analysis. In: 24th IEEE International Requirements Engineering Conference Workshops (REW), pp. 165–169. doi:10.1109/REW.2016.039.

Kuhn, A., Greevy, O., 2006. Exploiting the analogy between traces and signal processing. In: 22nd IEEE International Conference on Software Maintenance (ICSM 2006), 24–27 September 2006, Philadelphia, Pennsylvania, USA, pp. 320–329. doi:10.1109/ICSM.2006.29.

Laprie, J.-C., 1992. Dependability: basic concepts and terminology. In: Dependability: Basic Concepts and Terminology. Springer, pp. 3–245.

Li, T., Cole, B., Morton, P., Li, D., 1998. Cisco Hot Standby Router Protocol (HSRP). RFC 2281 (Informational). http://www.ietf.org/rfc/rfc2281.txt

Maier, G., Pattavina, A., De Patre, S., Martinelli, M., 2002. Optical network survivability: protection techniques in the wdm layer. Photon. Netw. Commun. 4 (3–4), 251–269.

Maoz, S., Harel, D., 2011. On tracing reactive systems. Softw. Syst. Model. 10 (4), 447–468. doi:10.1007/s10270-010-0151-2.

Maoz, S., Kleinbort, A., Harel, D., 2007. Towards trace visualization and exploration for reactive systems. In: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007), pp. 153–156. doi:10.1109/VLHCC.2007.27.

Marcus, A., Xie, X., Poshyvanyk, D., 2005. When and how to visualize traceability links? In: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering. ACM, New York, NY, USA, pp. 56–61. doi:10.1145/1107656.1107669.

Mohror, K., Karavanic, K.L., Snavely, A., 2010. Scalable event trace visualization. In: Lin, H.-X., Alexander, M., Forsell, M., Knüpfer, A., Prodan, R., Sousa, L., Streit, A. (Eds.), Euro-Par 2009 – Parallel Processing Workshops: HPPC, HeteroPar, PROPER, ROIA, UNICORE, VHPC, Delft, The Netherlands, August 25–28, 2009, Revised Selected Papers. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 228–237. doi:10.1007/978-3-642-14122-5_27.

Osmari, D.K., Vo, H.T., Silva, C.T., Comba, J.L.D., Lins, L., 2014. Visualization and analysis of parallel dataflow execution with smart traces. In: 27th SIBGRAPI Conference on Graphics, Patterns and Images, pp. 165–172. doi:10.1109/SIBGRAPI.2014.2.

Pavlik, J., Komarek, A., Sobeslav, V., Horalek, J., 2014. Gateway redundancy protocols. In: Computational Intelligence and Informatics (CINTI), 2014 IEEE 15th International Symposium on. IEEE, pp. 459–464.

Pirzadeh, H., Shanian, S., Hamou-Lhadj, A., Alawneh, L., Shafiee, A., 2013. Stratified sampling of execution traces: execution phases serving as strata. Sci. Comput. Program. 78 (8), 1099–1118. doi:10.1016/j.scico.2012.11.002.

Reiss, S.P., 2005. Dynamic detection and visualization of software phases. SIGSOFT Softw. Eng. Notes 30 (4), 1–6. doi:10.1145/1082983.1083254.

Reiss, S.P., 2006. Visualizing program execution using user abstractions. In: Proceedings of the 2006 ACM Symposium on Software Visualization. ACM, New York, NY, USA, pp. 125–134. doi:10.1145/1148493.1148512.

Resnick, R. I., 1996. A modern taxonomy of high availability. http://www.generalconcepts.com/resources/reliability/resnick/HA.htm

Sultana, N., Paira, S., Chandra, S., Alam, S.K.S., 2017. A brief study and analysis of different searching algorithms. In: 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT), pp. 1–4. doi:10.1109/ICECCT.2017.8117821.

Walker, R.J., Murphy, G.C., Freeman-Benson, B., Wright, D., Swanson, D., Isaak, J., 1998. Visualizing dynamic software system information through high-level models. SIGPLAN Not. 33 (10), 271–283. doi:10.1145/286942.286966.

Watanabe, Y., Ishio, T., Inoue, K., 2008. Feature-level phase detection for execution trace using object cache. In: Proceedings of the 2008 International Workshop on Dynamic Analysis: Held in Conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008). ACM, New York, NY, USA, pp. 8–14. doi:10.1145/1401827.1401830.

Winkler, S., 2008. On usability in requirements trace visualizations. In: 2008 Requirements Engineering Visualization, pp. 56–60. doi:10.1109/REV.2008.4.

Wright, H.K., Kim, M., Perry, D.E., 2010. Validity concerns in software engineering research. In: Proceedings of the FSE/SDP workshop on Future of software engineering research. ACM, pp. 411–414.

Zaidman, A., 2006. Scalability solutions for program comprehension through dynamic analysis. In: Conference on Software Maintenance and Reengineering (CSMR'06), pp. 4pp.–330. doi:10.1109/CSMR.2006.46.

Zawawy, H., 2012. Requirement-based Root Cause Analysis Using Log Data. University of Waterloo Ph.D. thesis. http://hdl.handle.net/10012/6851

Maged Sheghdara holds an M.Sc. degree in software engineering from King Fahd University of Petroleum and Minerals in 2019, KSA. He has a B.S. degree in computer science from Al-Ahgaff University in Yemen. His research interests include Software Requirements Engineering and dependability analysis.

Jameleddine Hassine is an Associate Professor at the department of Information and Computer Science of King Fahd University of Petroleum and Minerals (KFUPM). Dr. Hassine holds a Ph.D. from the Faculty of Engineering and Computer Science at Concordia University (2008) and a M.Sc. from the School of Electrical Engineering and Computer Science (SEECS) at the University of Ottawa (2001). Prior to this, he earned a Computer Engineering Diploma from the National School of Computer Science (Tunis, Tunisia) (1997). Dr. Hassine has several years of industrial experience within world-wide telecommunication companies: Nortel Networks (2000–2001) and Cisco Systems (2005–2010). Dr. Hassine research interests include software engineering, mining execution traces, high availability modeling, formal verification of distributed systems, and Communication protocols. He is actively involved in several funded research projects and has over 40 publications on various research topics in his field. Dr. Hassine has been on the editorial boards of a number of international journals, and served in the organization of many international conferences.