Modern Applied Science; Vol. 12, No. 10; 2018
ISSN 1913-1844 E-ISSN 1913-1852
Published by Canadian Center of Science and Education

Automatic Recovery of Database Structure (ARDS)

Esra F. Alzaghoul[1], Hussam N. Fakhouri[2] & Fawaz A. Alzaghoul[3]

[1] King Abdullah II School of Information Technology, The University of Jordan, Amman, Jordan

Correspondence: Hussam N. Fakhouri, Abdullah II School of Information Technology, The University of Jordan, Amman, Jordan. E-mail: h.fakhouri@ju.edu.jo

Received: February 3, 2018 Accepted: August 14, 2018 Online Published: September 26, 2018

doi:10.5539/mas.v12n10p71 URL: https://doi.org/10.5539/mas.v12n10p71

Abstract

One of the evolutions of information technology which is a very fascinating feature is the application of the auto recovery. This feature enables an external system to automatically diagnose other systems, detects the error that causes the failure, then recovers and reconfigures the system. The concept of software and web auto recovery is widely used in much software such as windows operating system which restores and recovers tools. Since the aim is to fast recover the application and keep it running and available as optimal as possible then it will be suitable to apply this capability to the database applications to fast recover from any unexpected change that may happen. This paper proposes an auto-recovery system that monitors, diagnoses, checks and heals database applications automatically and immediately with unnoticeable recovery time. The aim is to recover and to redo the changes that happened to the database by internal unauthorized user or external intrusion. To test the practical applicability of the proposed methodology, an application has been developed to demonstrate the methodology and apply it for real time database applications. The results of experiments performed on different scenarios demonstrated the ability of the proposed framework to recover database applications.

Keywords: Database Applications, auto Recovery Applications, Self-Healing, Auto Restore, and Auto Backup

1. Introduction

Database and Software development engage a lot of researchers for the development of a complete system which is typically secure, significantly used, reliable, configured, most effective, self controlled, facilely updatable, healed and distinguished. The word self-healing is deeply connected the field of autonomic computing. Autonomic computing refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users.

Started by IBM in 2001, this initiative ultimately aims to develop computer systems capable of self-management, to overcome the rapidly growing complexity of computing systems management, and to reduce the barrier that complexity poses to further growth (Horn, 2001). The system makes decisions on its own, using high-level policies; it will constantly check and optimize its status and automatically adapt itself to changing conditions. An autonomic computing framework is composed of Autonomic Components (AC) interacting with each other. An AC can be modeled in terms of two main control loops (local and global) with sensors (for self-monitoring), effectors (for self-adjustment), knowledge and planner/adapter for exploiting policies based on self and environment awareness. This architecture is sometimes referred to as Monitor-Analyze-Plan-Execute (MAPE).

1.1 Characteristics of Autonomic Systems

Self-configuration: Automatic configuration of components; Self-healing: Automatic discovery, and correction of faults (Montani, and Cosimo, 2008).

Self-optimization: Automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements.

Self-protection: Proactive identification and protection from arbitrary attacks.

Self-regulation: A system that operates to maintain some parameter, e.g., Quality of Service (QoS), within a reset range without external control.

Self-learning: Systems use machine learning techniques such as unsupervised learning which does not require external control.

Self-awareness (also called Self-inspection and Self-decision): System must know itself. It must know the extent of its own resources and the resources it links to. A system must be aware of its internal components and external links in order to control and manage them.

Self-organization: System structure driven by physics-type models without explicit pressure or involvement from outside the system.

Self-creation (also called Self-assembly, Self-replication): System driven by ecological and social type models without explicit pressure or involvement from outside the system. A system's members are self-motivated and self-driven, generating complexity and order in a creative response to a continuously changing strategic demand.

Self-management (also called self-governance): A system that manages itself without external intervention. What is being managed can vary dependent on the system and application.

Self-description (also called self-explanation or Self-representation): A system explains itself. It is capable of being understood (by humans) without further explanation.IBM has set forth eight conditions that define an autonomic system (Horn, 2001).The system must know itself in terms of what resources it has access to, what its capabilities and limitations are and how and why it is connected to other systems. Be able to automatically configure and reconfigure itself depending on the changing computing environment. Be able to optimize its performance to ensure the most efficient computing process. Be able to work around encountered problems by either repairing itself or routing functions away from the trouble. Detect, identify and protect it against various types of attacks to maintain overall system security and integrity. The system must be able to adapt to its environment as it changes, interacting with neighboring systems and establishing communication protocols. To rely on open standards and cannot exist in a proprietary environment and anticipate the demand on its resources while keeping transparent to users.Even though the purpose and thus the behavior of autonomic systems vary from system to system, every autonomic system should be able to exhibit a minimum set of properties to achieve its purpose:

Automatic: This essentially means being able to self-control its internal functions and operations. As such, an autonomic system must be self-contained and able to start-up and operate without any manual intervention or external help. Again, the knowledge required to bootstrap the system (Know-how) must be inherent to the system (Tianfield, &Unland, 2004).Adaptive: An autonomic system must be able to change its operation (i.e., its configuration, state and functions). This will allow the system to cope with temporal and spatial changes in its operational context either long term (environment customization/optimization) or short term (exceptional conditions such as malicious attacks, faults, etc.) (Hudaib, and Fakhouri, 2016).

Aware: An autonomic system must be able to monitor (sense) its operational context as well as its internal state in order to be able to assess if its current operation serves its purpose. Awareness will control adaptation of its operational behavior in response to context or state changes. The fast and recent web spread motivated large numbers of companies to offer their services and business through the World Wide Web (WWW) and to use the database 24/7 hours and seven days in the week, the dataset should be accessible all the time. This wide spread of applications require more research to develop database applications that has the ability to be auto recovered. Auto recovery includes the ability of the application to be recovered detecting any fault or unexpected unauthorized change that happened to the web application files. Auto recovery of database applications require a 24/7 auto monitoring of the applications and a fast mechanism of recovery that can keep the online functionality and service offering to the customer available all the time. Many factors may affect adatabase application and cause it to stop. These factors may be either internal or external. The internal factor includes viruses, worms that may affect the server that hosts the database application or a used to the system who may alter the tables of the database in an unauthorized way to get illegal access to the database application. The external factors include attackers that attack the application and change the content or the structure of the database table or data for different reasons including the use of different methods such as XSS,and SQL injection (Hudaib, and Fakhouri, 2016).

After running the database application on the hosting server many problems rise including illegal access to the database, creating illegal username or passwords to an authorized people, adding new tables or modifying a table by adding new columns that serve an attacker to the database without noticing by the administrator of the system. The risk of having one of these factors is very high. For example, adding unauthorized user to the application would allow the attacker for example to steal critical information such as credit card information which will cause a major problem. Most of web application owner do not perform tests to check if the table structure or any change had been modified to the database table or the users who access the database have been changed

In this research, we mainly focus on techniques for database applications auto recovery from different types of illegal or unauthorized modifications and changes by automatically detecting failures, diagnose the problem, and

heal the database application to behave and run as supposed to be before the problem happens. The proposed approach insures that the database table structure and relations and the legal accessed users are the same without any modifications or changes by any other external unauthorized effect and no columns has been added, omitted or deleted from the server and that the database application does not contain any injected or added users to access it illegally. Our research suggest the existence of an approach that analyze the content of the database structure and relations, a mechanism for monitoring the database application, a mechanism for diagnosing and detecting of any illegal change or modification and a healing mechanism that bring back the database to its original status.

The major research contribution is defining a mechanism for database application auto-recovery despite the cause either caused by internal illegal user or an intrusion. The mechanism is automatic and relay on the diagnosis of the database application structure. In summary this research defines a mechanism for healing of database applications by external self healing approach and recover from failure. Design an approach that can monitor, diagnose and detect failure automatically, efficiently. Develop an implementation of the framework for any database and provide an experimental result that demonstrates the efficacy of the approach. This paper suggests a solution for the above mentioned problems by proposing an approach that is able to auto recover database system by monitoring, detecting and healing the structure of tables and relations in the database application.

The reset of this paper is organized as follows: section two present the related work, while section three presents a full description of the proposed auto-recovery of database structure (ARDS) framework; section four contains the experimental results and evaluation; and finally section five presents the conclusion.

2. Related Work

Many researchers worked to develop self-healing systems (Sidiroglouet al., 2009 Khan, 2011 , Frei, &Serugendo, 2015, Hudaib, and Fakhouri, 2016, Dashofy, et al., 2002, Philip Koopman, 2003, George et al., 2002, Dabrowski et al., 2002, Keromytis "2009). George et al., 2002, Presented a cell-based programming model inspired from biology and speculate on biologically inspired strategies for producing robust, scalable and self-healing software systems. Goutam K. S., 2007, described various issues on designing a self-healing software application system that relies on the on-the-fly error detection and repair of web application or service agent code and data, illustrated critical points of the emergent research topic of Self–Healing Software System. Dashofy, et al., 2002, presented tools and methods that implemented infrastructure elements in the context of an overall architecture-based vision for building self-healing systems, and concluded with a gap analysis of current infrastructure vs. the overall vision, and plans for fulfilling that vision. Philip Koopman, 2003, proposed a taxonomy for describing the problem space for self-healing systems including fault models, system responses, system completeness, and design context.

Krueger et al. (2010) introduced a methodology for a self-healing web application and presented the prototype TokDoc andevaluated its detection performance and runtime using real HTTP. Simmonds et al. (2010) described a framework for runtime monitoring and recovery of web service conversations. Park et al. (2009) improved development environments of self-healing software to facilitate self-managing capabilities such as self-healing and proposed the use of deployment diagram for self-healing framework to determine problems arising in external environments. Lemoset.al, (2013) presented a hybrid approach for self management that combines an endogenous self-adaptation approach with an exogenous self-healing approach. Vasar et al. (2012) presented a framework that helps in monitoring and testing scalability of web applications on the cloud. Zohrevandi et al. (2013) proposed a novel and simple approach for securing access to sensitive content on the web concerned with the protection of the content associated with sensitive websites. Tran et al. (2014) presented a new technique, Nyx, to make web applications more energy for OLED based mobile devices. The basic idea of Nyx is to replace the large, light colored background areas of web applications with dark colors (preferably, black) to reduce the energy consumed by OLED screens. Brumley et al. (2007) presented a self-healing architecture for software systems where programs self-monitor and detect exploits, self-diagnose the root cause of the vulnerability, self-harden against future attacks, and self-recover from attacks. Fayyadet.al, (2014)presented a self-healing architecture for software system based on one of the biological processes that have the ability to heal by itself- the wound-healing process, which is divided into two layers, functional layer and healing layer. Sidiroglou et a, (2002) provided a critique of the current state of the art and offers the position that self healing as a concept should be relegated to the status of autonomic computing: a goal worth aiming for as a way to push the boundary of the possible rather than an achievable end in itself. Harald et.al focused on the self healing branch of the research and gives an overview of the current existing approaches. Dabrowski et al. (2002) used architectural models to characterize how architecture, topology, consistency-maintenance mechanism, and failure-recovery strategy each contribute to self-healing during communication failure. Elkorobarrutia et al. (2008) described an approach of inserting a self-healing mechanism in components that are specified according to a state chart and whose implementations also offer the possibility to act on them in terms of state; i.e. forcing the component to some state and rolling back one transition. Anand et

al. (2010) proposed self-healing systems that prove increasingly important in countering system software based attacks, which recover and secure to the data from interrupted services. Self-healing systems offer an active form of decision support, without human intervention that can detect the fault and recover from the fault. Angelos (2007), Attempted to characterize self-healing software systems by surveying some of the existing work in the field. focusing on systems that effect structural changes to the software under protection, as opposed to block level system reconfiguration. Locastoet. al (2007), introduced and explored the concept of Application Communities: collections of large numbers of independent instances of the same application, and demonstrated the feasibility of the scheme using Selective Transactional emulation (STEM) as both the monitoring and remediation mechanism for low-level software faults, and provided some preliminary experimental results using the Apache web server as the protected application.

3. Proposed Framework: Auto-Recovery of Database Structure (ARDS)

To explain the proposed approach we start by explaining the purpose of ARDS development, then present ARDS main components, and show the auto recovery mechanism of ARDS. ADRS main purpose to solve the leak that may happen if the internal user is the attacker and he modified the database either a columns or table without noticing of the administrator of the database and this may cause leak of information that can be accessed illegally and unauthorized. ARDS came to solve this problem and the leak in the security system that other software systems don't solve for example the virus and worm attack can be solved by installing antivirus, the external attack can be solved by installing a firewall but the internal user who can access the database legally can't be discovered by their two software because the user in this case can access the database legally from a username and password that he use but the change that he may do to the database is illegal as shown in figure 1.

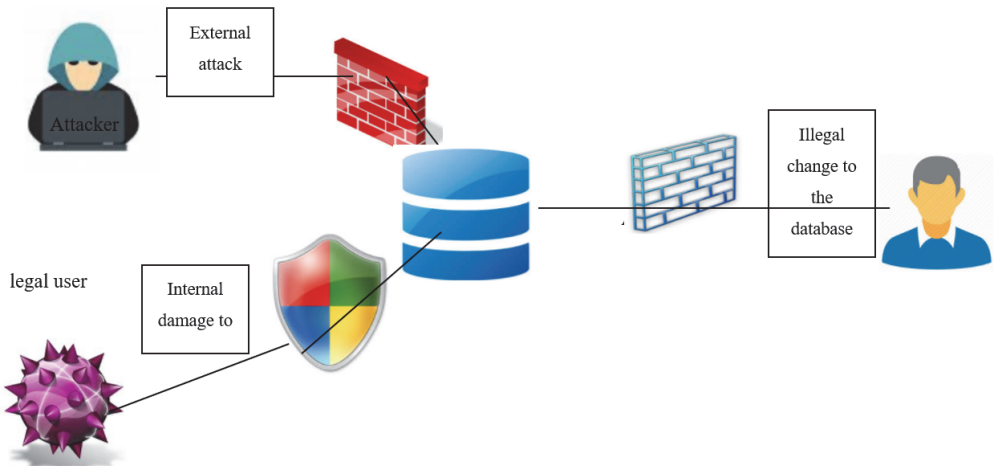

Figure 1. Auto-recovery of database Structure (ARDS)

ARDS is developed for online web database and after publishing the database on the hosted server, its model suggests the integration, developing and building an auto recovery software (ARDS) for the targeted hosted database. ARDS is able to read the database structure and use it as an input for the requirement analysis later to be used for the auto recovery of the database. Analyzing the database structure and relations provide a wide strong knowledge base for the auto recovery of the system that leads it through the diagnosis and contain all the required information about. Developing ARDS aims to perform a healing process for the targeted database. Building ARDS mainly rely on the nature and structure of the published database tables and relations in its final structure. ARDS framework has two phases pre-healing and healing phase (auto recovery) and each phase consist of many steps.

3.1 The Pre-Healing Phase

In this phase, the proposed Framework will know the structure of the database and its data types as explained in figure 2. Furthermore, analysis and planning in the software building phase guide the process phase through correct diagnosis and recovery. The proposed framework proposes developing a controller software of the database that will be distributed upon request by the user in many cases such as database unexpected changes, a suspected behavior, and change in the performance or the excepted results of the database. It will have the capability to diagnose, and heal the database.

The pre-healing procedure for proposed framework ;

1. If there is no information about the database in ARDS or If information is not up-to-date
2. Read Database
3. Count tables number
4. Get tables names
5. Count column number for each table
6. Get column names for each table
7. Get columns data type
8. Store information in ARDS database
9. Store backup copy of the database
10. Update database store website information from step 3
11. Create a backup copy and store it compressed in different folder
12. End if

Figure 2. The pre-healing procedure for proposed framework

The ARDS main components are shown in Figure 3 which illustrates that the ARDS knowledge will be based on the design and implementation processes.

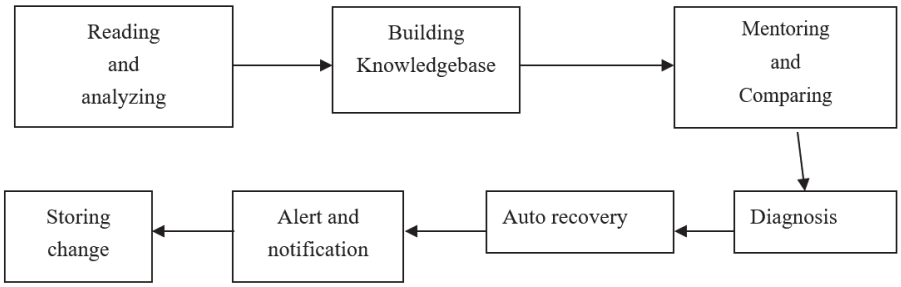

Figure 3. The ARDS main components

A full detail for the description of the main components of each phase and the procedure in each phase is illustrated in sections 3.2 to 3.3.

3.2 The Healing Phase

running the proposed framework: when the user detects a change in the software as mentioned in previously he run ARDS on the same system that face the problem, ARDS has a build in database and knowledge base that has information about the software, its structure and component

Read and analyze: The development process of ARDS includes the following: -Building database: This step is done by storing all of the information about the original database tables structure, relations and user (i.e. name of tables, types, name of columns, data types, date created) in the ARDS database records that will be used in the diagnosis of the software

- Monitoring: which keeps a 24/7 monitoring status for any change that may happen to the hosted database? It keeps track of all the tables, columns type and number for any change that may cause any changes that will be detected and monitored such as deletion of table or column, adding a column and modification of the columns or attribute type.
- Comparing: As soon as the proposed framework runs it will start the checking process, gather information about the installed software and compare it with the knowledge base that it has. This process provides the diagnosis process with the results of comparing the monitored component with record stored on the original database in ARDS. The database contains a full detail of all components of the web application that are required for the diagnosis.
- Diagnosis: decide wither the system need to be healed or it is in good health. In this step a solution to the system will be required and suggested if the system is infected or at defect state.
- Healing: it is the process of reusing the original database backup copy of the system and replacing or

compensating the affected component in order for the system to be in good health. In this step the solution to the problem is applied; when a fault is detected during the diagnosis phase, the latest application saved copy will be restored to the web application directory. The restoration is triggered by detecting a change. This process is in the order of seconds. The changes along with the healing event will be stored in the database for further analysis and the process is automated.

- Storing changes in database: the unexpected activities and the healing activities will be stored in ARDS database and reported to the administrator. Storing this information will give the administrator a clear summary about the history of the database for future precaution. Keeping this information will give indicator about the reason that caused the fault and the accessed user and this will help us avoid such situations and to enhance or develop mechanisms the server so that it can face such cases. For example if the reason of change was illegal access to the server then certain policy could be taken into consideration or if the change to the component has been done by a virus then the server itself should be treated by clearing the virus.
- Alert and notification: This step is important step so that the administrator immediately knows about the leak that happens to the database and can take fast action to capture the attacker and prevent him from further damage to the database.
- Update: Analyzing the causes of software help to define solutions to avoid such cases to happen and to be update the software released component for further versions. The proposed framework will have the capability to update the component of the software in situations of environmental changes; Since overtime the system original environment (e.g., CPU, operating system, business rules, external product characteristics) for which the software was developed is likely to change.

ARDS consist of automatic exterior healing framework that monitor the database structure and try to keep it in good health 24/7 working time. Our main concern is the finally published and created database on the hosting server. ARDS check the database structure such as the table's structure, tables name, number and data type and relations between tables. The framework will monitor, diagnose, and recover the database immediately at the time of the external or internal effect that could cause the database to change unexpectedly. The proposed system is to guarantee the full time running of the database application without any illegal or unauthorized changes either caused by internal user, intruder or attacker.

The proposed framework exhibit update features results in modification to the software to accommodate changes to its external environment. it can provide additional functionality for the software that includes Enhancement. As software is used, the customer/user will recognize additional functions that will provide benefit. The user can request the enhancement after that it will have a set of knowledge and procedure to apply the enhancement for the database by replacing or adding the required changes. This will be useful when the database is large in size and the enhancement require changing and updating only few tables or procedures in the database; rather than reconstructing the whole database again or redistributing it. The proposed framework can do the enhancement process and then check that the database is in good health according to the rules and knowledgebase stored in ARDS that has all required information about the database that is used for.

Analyzing the reasons determining the solution and distributing it through the proposed framework make it exhibit the Prevention feature of software reengineering process. For example if the reason that cause the database fault is illegal access and deleting of a table or column then analyzing the reasons and discovering the cause will make the software administrator take several procedure to prevent further cause This feature make ARDS more easily corrected, adapted, and enhanced.

ARDS framework provides architectural support external healing for the software in an automated manner and mainly when the attacker to the database is an internal user of the system. The framework corrects the attack by returning the database to its original state.

ARDS framework dynamically modifies the database to correct the failure, the changes that has happen will be stored to be analyzed in the future by the admin and if this error happened often then the recorded information will guide the admin for the reason by analyzing the stored information. If the reason is from the database itself then it will be recorded as new detected defect in the database.

ARDS framework transfer the software from the fail to run or infected and suspected state to original steady state as illustrated in figure 4.

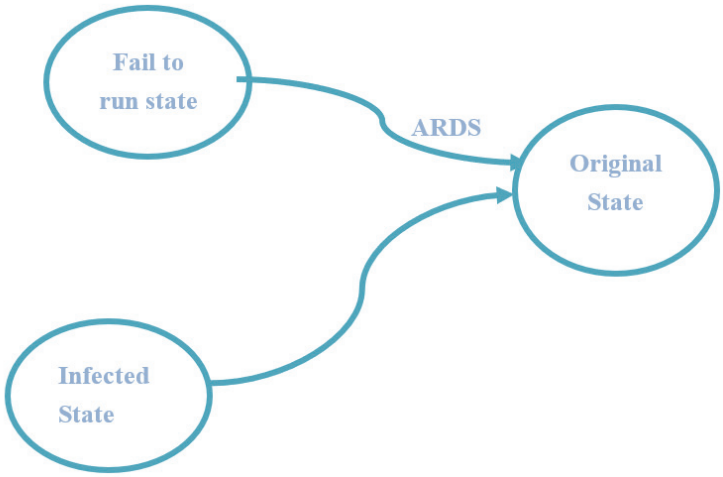

Figure 4. State diagram of software healing using ARDS

3.3 Running Mechanism of ARDS

The running mechanism are shown in figure 5 that explain how ARDS run by the user and how to function to trigger the run of the internal processes.

Prior to the user request and running, ARDS starts to discover candidate failure reasons. After the check and information gathering completes, it starts the comparing phase. It check the application with black box testing technique for the software components that facilitate the detection and reporting of application unexpected changes. In addition, it stores changes of the database so that it can analyzed in the future to avoid such cases to happen.

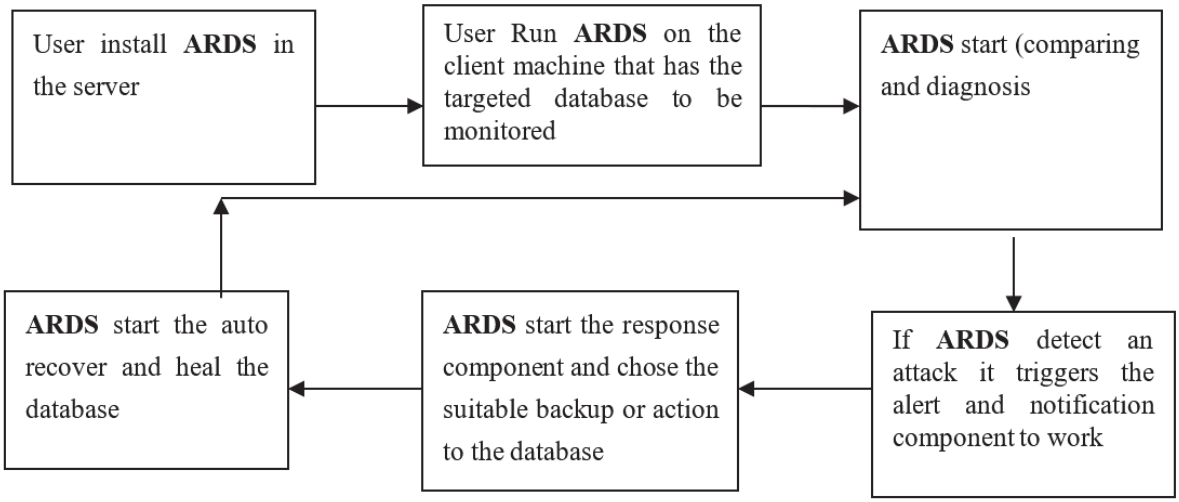

Figure 5. Running mechanism of ARDS

When a failure or fault is detected after in the diagnosis phase, ARDS then initiate a healing mechanism which aim in the final stage to have a healthy software that is fully running and this is done by choosing the suitable backup to replace the infected or missed component. After the diagnosis phase the healing phase of the proposed framework is applied as a very fast repairing mechanism that can be done in unnoticeable time, shows that this is in the order of seconds, and the cost is amortized as it is incurred once per new fault. Once a candidate fault or failure reason is detected and recognized and the backup is selected and imported, ARDS confirms that it is ready for deployment by verifying that it satisfies three criteria: survivability, correctness and suitability. The healing process provides survivability. Deployment of the deleted tables or replacing it provide survivability feature from fault. A Deployment of the backup copy is efficient if the performance implications of the database return to its original status. Correctness is verified if the application runs correctly. As soon as a backup is verified, ARDS

dynamically install it to the software while the application is on the client system. The installation instantiates the component inside the application to correct the application against the particular fault. The modified application will return back its original state before the fault occurs. The report that is sent to the knowledge database is used to produce better component that enhance the application against faults that may occur at a specific program location in the future. The resulting mechanism is input-agnostic, and thus immune to risks related to fault/input polymorphism. The Main processes details are shown in figure 6.

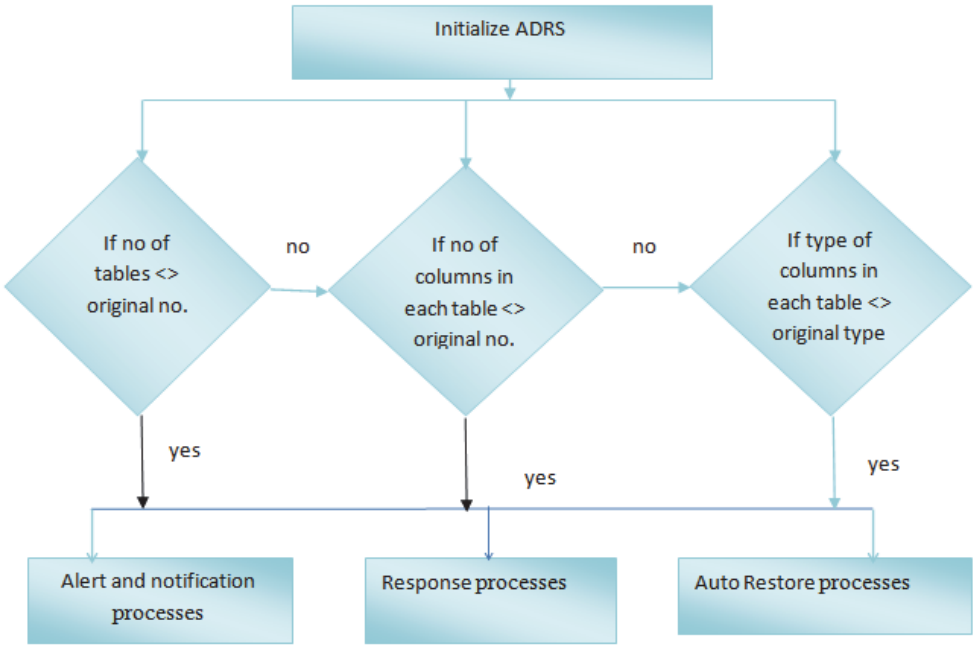

Figure 6. Main processes details

ARDS dynamically modifies the website to correct the failure. The change that has happen will be stored to be analyzed in the future by the admin and if this error happened often then the recorded information will guide the admin for the reason by analyzing the stored information or the affected component.

Analyzing the changes, the result of checking, diagnosis and monitoring give us indicator about the reasons that cause the system to fail. It also gives a brief overview about the main causes and its indicators. By defining the reason the system administrator can find a suitable solution for the reason that caused the problem.

4. Experimental Results and Evaluation

In this section, we evaluate the ability of ARDS to recover from different scenarios that may affect the database and change its normal behavior. More specifically, and through experimental study to evaluate the proposed framework we need to evaluate the effectiveness and the ability of the proposed framework.Wehave rum different scenarios that may face the database and a full description of these scenarios and the procedure that ARDS take to face each problem is described from case 1 to case 7.We also calculated the Recall, precision and Accuracy to measure the performance of the proposed framework.

Case I: Normal: No Change has been done to the database

We initialized the implemented self healing framework and selected the web application directory to be monitored, analyzed the testing website directory and built the database.

ARDS compare the software component with the information that it has on the database. In case of healthy software no action will be taken, the set of procedures performed by the ARDS are shown in figure 7.

1. 1: Decrement the number of tables for the targeted database
2. If (number of the examined tables = number of tables stored in ARDS database)
3. 1: Decrement the number of columns for the targeted database

```
4.  If (number of the examined    columns = number of columns stored in ARDS database)
5.              Loop
6.              Read and analyze columns name and data type
7.                      if (columns name = columns names stored in ARDS
                        database&columns data type = columns data type stored in ARDS
                        database)
8.                      generate healthy software report
9.                      end if
10.             End loop
11. end if
12. end if
```

Figure 7. ARDS Response Algorithm for Normal Case Scenario

Case 2: adding a table to the database: We run ARDS after that we add a table to the tested database ARDS healed the deleted component, a set of procedures performed by the ARDS are shown in figure 8.

```
1.  1: Decrement the number of tables for the experimented database
2.  If (number of the examined  tables > number of tables stored in the ARDS
    database)
3.  Initialize notification component
4.  Send notification to the administrator
5.  Initialize response component
6.  Determine the added table name
7.  Store information about the added table in ARDS
8.  Drop added table
9.  Generate a status report about the action that has been taken by ARDS
10. Send report on to the administrator
11. End if
```

Figure 8. ARDS Response Algorithm for adding a table to the database Case Scenario

Case 3: deleting a table to the database: We run ARDS after that we Deleted a table to the tested database ARDS healed the deleted component, a set of procedures performed by the ARDS are shown in figure 9.

```
1.  1: Decrement the number of tables for the experimented database
2.  If (number of the examined    tables < number of tables stored in the ARDS database)
3.  Initialize notification component
4.  Send notification to the administrator
5.  Initialize response component
6.  Determine the deleted table name
7.  Store information about the deleted table in ARDS
8.  Create missing table
9.  Import the data of the deleted table from ARDS
10. Generate a status report about the action that has been taken by ARDS
11.   Send report on to the administrator
12. end if
```

Figure 9. ARDS Response Algorithm for adding a table to the database Case Scenario

Case 4: Modifying a table name: To test this case the experimented database has been modified manually. ARDS responded to this case by deleting the modified table and restoring the original table from the database, the set of procedures performed by the ARDS are shown in figure 10.

```
1.  If (number of the examined    tables = number of tables stored in the database of the system)
2.          Loop
3.          Read and analyze tables info (name and date of modification )
```

```
4.                    if (table name <> original table name    )
13.                                 Initialize notification component
14.                                 Send notification to the administrator
5.                    initialize response component
6.                    detect    the modified component
7.                    Store information( new name, date of last modification )    about table in
                      ARDS
8.                    Send report to the administrator
9.                    End if
10.          End loop
11.  end if
```

Figure 10. ARDS Response Algorithm for Modifying a table name

Case 5: adding a Column to the database: We run ARDS after that we add a Column to the tested database ARDS recovered the deleted component effectively, a set of procedures performed by the ARDS are shown in figure 11.

```
1.  Decrement the number of tables for the experimented database
2.  If (number of the examined    tables = number of tables stored in the ARDS database)
3.  loop
4.  If (number of the examined    columns in each table > number of tables columns stored in the
    ARDS database)
5.  Initialize notification component
6.  Send notification to the administrator
7.  Initialize response component
8.  Determine the added columns name
9.  Store information about the added columns in ARDS
10. delete added columns
11. Generate a status report about the action that has been taken by ARDS
12.   Send report on to the administrator
13. End loop
14. end if
```

Figure 11. ARDS Response Algorithm for adding a column to the database Case Scenario

Case 6: deleting a column froma table in the database: We run ARDS after that we deleted a column from a table to the tested database.

ARDS recovered the deleted column, a set of procedures performed by the ARDS are shown in figure 12.

```
1.  Decrement the number of tables for the experimented database
2.  If (number of the examined    tables = number of tables stored in the ARDS database)
3.  loop
4.  If (number of the examined    columns< number of columns stored in the ARDS database)
5.  Initialize notification component
6.  Send notification to the administrator
7.  Initialize response component
8.  Determine the deleted column name
9.  Store information about the deleted table in ARDS
10. dropthe table that has the deleted columns
11. create the dropped table
12. Import the data of the deleted table from ARDS
13. Generate a status report about the action that has been taken by ARDS
14.   Send report on to the administrator
15. End loop
16. end if
```

```
17.  end if
```

Figure 12. ARDS Response Algorithm for deleting a column from a table in the database Case Scenario

4.1.4 Case 7: Modifying a column name or data types: To test this case the experimented database has been modified manually. ARDS responded to this case by deleting the modified a column name effectively by restoring the original table from the database, the set of procedures performed by the ARDS is shown in figure 13.

```
15.  Decrement the number of tables for the experimented database
16.  If (number of the examined tables = number of tables stored in the ARDS database)
12.  If (number of the examined tables columns = number of tables columns stored in ARDS database)
13.        Loop
14.        Read and analyze tables info (columns name and data types)
15.                  if (column name <> original column name or column data type <> original column
                     data type)
15.                              Initialize notification component
16.                              Send notification to the administrator
16.                  initialize response component
17.                  detect the modified component
18.                  Store information (new name, date of last modification) aboutthe tablethat has the
                     modificationin ARDS
19.                  Drop the table
20.                  Create new table
21.                  Import data of the table from ARDS
22.                  Send report to the administrator
23.                  End if
24.         End loop
25.  end if
```

Figure 13. ARDS Response Algorithm for Modifying a column name or data types Case Scenario

We verified that the proposed auto recovery time which showed faster results than the software developed without including the auto recovery and healing process, thus it is more useful in different situations especially in urgent ones or when there is no support team at the site of the client.

The response time including (Contact Time, On-site Response Time (ORT), Fix Time (FXT), and Turnaround Time (TAT). Have been calculated for software without auto recovery properties and the software developed using our proposed framework the results are shown in figure below. The response time for software without auto recovery properties was gathered by contacting and gathering information from 300 software developer. And the response time for the proposed framework was calculated by experiment on different cases illustrated above.

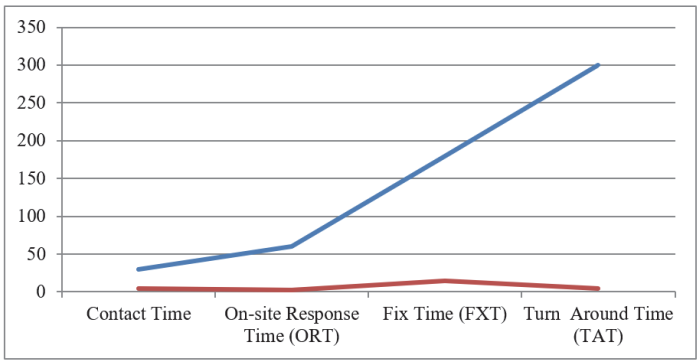

Figure 14. Time comparison between software developed using the proposed auto recovery framework and software developed without auto recovery

Where the blue line represents the average time for software developed without auto recovery, and red color represent software developed using the proposed framework. As we have noticed that the software that don't have auto recovery response time is much higher than the proposed framework.

We calculated the percent delinquent fixes by equ.1 , the proposed framework perform auto recovery and according

to this fact and to the experimental results the percent delinquent fixes was zero for all of the tested cases which is the optimal value to get. Because none of the tested cases showed an exceed in response time.

The percent delinquent fixes = (number of fixes that exceeded the response time criteria by severity level / number of fixes delivered in a specified tim.

To measure the performance of the proposed framework for all test cases we calculated the Recall, precision and Accuracy:

Recall is the true positivity rate, also referred to as sensitivity. Recall has been calculated as shown in Eq. 1:

$$\frac{True\ Positives}{TruePositives+FalseNegatives} \quad \text{..... Eq. 1}$$

The precision, calculated as the positive prediction value, as shown in Eq.2:

$$\frac{True\ Positives}{TruePositives+FalsePositives} \quad \text{..... Eq.2}$$

Accuracy is the rate with which items have been correctly classified and/or retrieved. Accuracy ahs been calculated as shown in Eq. 3:

$$\frac{True\ Positives+TrueNegatives}{TruePositives+FalseNegatives+TrueNegatives+FalsePositives} \quad \text{..... Eq.3}$$

The precision and recall values has been calculated for the tested cases where the Precision value is 0.884 and the Recall is 0.732 and the Accuracy 0.852.

5. Conclusion

Automated software recovery techniques, approaches and methodologies have improved software performance, maintenance and running time effectively. With the increase in complexity of systems the need for such systems is a must. The automation of software recovery and reducing the human interaction has major improvement of the software development, it plays a key role in reducing the time needed to fix the software issues and reduce the cost for maintenance.

This paper presented automateda framework (ARDS) for illegal or unauthorized modification of the hosted database or any part of its tables or columns. ARDS has the required information about the database that is needed to diagnose and recover the software from failure. Moreover, it is able to use the information to auto recover the database from different faults or attacks that the database may face. A set of cases have been developed to test ARDS. The results of the experiments with different cases illustrated the ability and efficiency of ARDS to diagnose and recover the database even if a single column name or data type has been changed.

References

Alessandra, G., Mauro, P., & Jochen, W. (2009). Achieving Cost-Effective SOFTWARE Reliability through Self-Healing. *Computing and Informatics, 29*, 93-115.

Ammo, K., Christian, G., Konrad, R., & Pavel, L. (2010). TokDoc: A Self-Healing Web Application Firewall. SAC'10, Sierre. https://doi.org/10.1145/1774088.1774480

Angelos, K. (2007). Characterizing Self-Healing Software Systems, Computer Network Security of the Series Communications. *Computer and Information Science, 1*, 22-33.

Boris, K., Ruben, M., Umakishore, R., & Marco, R. (2013). Recovery without Checkpoints in Distributed Event Processing Systems. DEBS'13, Arlington. https://doi.org/10.1145/2488222.2488259

Brumley, D., Newsome, J., & Song, D. (2007). Sting: An End-to-End Self-Healing System for Defending against Internet Worms Book. In Christodorescu, M., Jha, S., Maughn, D., Song, D., & Wang, C. (Eds.), *Malware Detection and Defense, Advances in Information Security* (Vol. 27, pp. 147-170). Springer, United States.

Candea, G., & Fox, A. (2003). Crash-Only Software. In Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS IX), pages 12–20, May 2003.

Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L., & Barham, P. (2005). Vigilante: End-To-End Containment of Internet Worms. In Proceedings of the 20th ACM Symposium on Operating Systems

Principles (SOSP 2005), 133-147, Dec. 2005.

Dabrowski, C., & Mills, K. (2002). Understanding Self-Healing in Service-Discovery Systems. WOSS'02 Proceedings of the First Workshop on Self-Healing Systems, Charleston, 18-19 November 2002, 15-20.

Dashofy, M., André, H., & Richard, T. (2002). Towards Architecture-Based Self-Healing Systems. WOSS'02, Charleston. https://doi.org/10.1145/582128.582133

De Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., ... Weyns, D. (2013). Software engineering for self-adaptive systems: A second research roadmap. In Software Engineering for Self-Adaptive Systems II (pp. 1-32). Springer, Berlin, Heidelberg.

Diaconescu, A. (2003). A Framework for Using Component Redundancy for Self-Adapting and Self-Optimizing Component-Based Enterprise Systems.Proceeding OOPSLA'03 Companion of the 18th annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'03), New York, 2003, 390-391.

Dinkel, M. (2008). A Novel IT-Architecture for Self-Management in Distributed Embedded Systems. PhD Thesis, TU Munich.

Edward, S., Kevin, L., Maxim, S., Chris, R., & Spiros, M. (2010). On the Use of Computational Geometry to Detect Software Faults at Runtime. Proceedings of the 7th International Conference on Autonomic Computing, Washington DC, 7-11 June 2010, 109-118.

Ehlers, J., André, H., Jan, W., & Wilhelm, H. (2011). Self-Adaptive Software System Monitoring for Performance Anomaly Localization. ICAC'11, Karlsruhe.

Elkorobarrutia, X., Muxika, M., Sagardui, G., Barbier, F., & Aretxandieta, X. (2008, March). A framework for statechart based component reconfiguration. In Engineering of Autonomic and Autonomous Systems, 2008.EASE 2008. Fifth IEEE Workshop on (pp. 37-45). IEEE.

Etoh, J. (2010). *GCC Extension for Protecting Ppplications from Stack-smashing Attacks.* Retrieved from http://www.trl.ibm.com/projects/security/ssp/

Fayyad, E. M., Almaadeed, M. A., Jones, A., & Abdullah, A. M. (2014). Evaluation techniques for the corrosion resistance of self-healing coatings.

Frei, R., & Serugendo, G. D. M. (2015). Self-healing software. In The computer after me: awareness and self-awareness in autonomic systems (pp. 71-82).

Fuad, M., Deb, D., & Baek, J. (2011). Self-Healing by Means of Runtime Execution Profiling. Proceedings of 14th International Conference on Computer and Information Technology (ICCIT 2011), Dhaka, 22-24 December 2011, 202-207. https://doi.org/10.1109/iccitechn.2011.6164784

Goutam, S. (2007). Software—Implemented Self-Healing System. CLEI Electronic Journal, 10, 5.

Harald, P., & Schahram, D. (2011). A Survey on Self-Healing Systems: Approaches and Systems. *Computing, 91,* 43-73.

Hervé, C., Leonardo, M., & Mauro, P. (2013). Exception Handlers for Healing Component-Based Systems. *ACM Transactions on Software Engineering and Methodology (TOSEM), 22*, 1-40.

Horn, P. (2001). Autonomic computing: IBM\'s Perspective on the State of Information Technology. https://doi.org/10.1145/2802658.2802660

Hudaib, A., & Fakhouri, H. (2016). An Automated Approach for Software Fault Detection and Recovery. *Communications and Network, 8*, 158-169. https://doi.org/10.4236/cn.2016.83016

Huebscher, C., & McCann, A. (2008). A Survey of Autonomic Computing Degrees, Models, and Applications. ACM Computer Survey, 40, Article No. 7. https://doi.org/10.1145/1380584.1380585

Jiang, M., Zhang, J., Raymer, D., & Strassner, J. (2007). A Modeling Framework for Self-Healing Software Systems. *Lecture Notes in Computer Science, 7003*, 61-68.

Katti, A., Fatta, G., & Naughton, T. (2015). Scalable and Fault Tolerant Failure Detection and Consensus. EuroMPI'15 Proceedings of the 22nd European MPI Users' Group Meeting, Bordeaux, 21-23 September 2015, Article No. 13.

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer, 36*(1), 41-50.

Khan, A. A. (2011). U.S. Patent No. 8,082,471. Washington, DC: U.S. Patent and Trademark Office.

Kramer, J., & J. Magee. Self-managed systems: an architectural challenge. In FOSE ’07: 2007 Future of Software Engineering, pages 259-268, 2007. IEEE Computer Society.

Krueger, T., Gehl, C., Rieck, K., & Laskov, P. (2010, March). TokDoc: A self-healing web application firewall. In Proceedings of the 2010 ACM Symposium on Applied Computing (pp. 1846-1853). ACM.

Li, D., Tran, A. H., & Halfond, W. G. (2014, May). Making web applications more energy efficient for OLED smartphones. In Proceedings of the 36th International Conference on Software Engineering (pp. 527-538). ACM.

Michael, S. (2005). Self-Healing Component in Robust Software Architecture for Concurrent and Distributed Systems. *Science of Computer Programming, 57,* 27-44. https://doi.org/10.1016/j.scico.2004.10.003

Montani, S., & Cosimo, A. (2008). Achieving Self-Healing in Service Delivery Software Systems by Means of Case-Based Reasoning. *Applied Intelligence, 28*, 139-152. https://doi.org/10.1007/s10489-007-0047-1

Naftaly, M. (2003). On Conditions for Self-Healing in Distributed Software Systems. Proceedings of the Autonomic Computing Workshop, 25 June 2003, 86-92. https://doi.org/10.1109/ACW.2003.1210208

Newsome, J., Brumley, D., & Song, D. (2006). Vulnerability–Specific Execution Filtering for Exploit Prevention on Commodity Software. In Proceedings of the 13 Th Annual Symposium on Network and Distributed System Security (NDSS 2006), pages 1-15, Feb. 2006.

Park, J., Youn, H., Lee, J., & Lee, E. (2009). Automatic Generation Techniques of a Resource Monitor Based on Deployment Diagram. ICHIT’09 Proceedings of the 2009 International Conference on Hybrid Information Technology, New York, 189-192.

Qin, F., Tucek, J., Sundaresan, J., & Zhou, Y. (2005). Rx: Treating Bugs As Allergies—A Safe Method To Survive Software Failures. In Proceedings of the 20 th ACM Symposium on Operating Systems Principles (SOSP 2005), 235-248, Oct.

Rinard, M., Cadar, C., Dumitran, D., Roy, D. M., Leu, T., William, J., & Beebee, S. (2004). Enhancing Server Availability and Security through Failure-Oblivious Computing. In Proceedings of the 6thSymposium on Operating Systems Design and Implementation (OSDI 2004), pages 303-316, Dec. 2004.

Selvin, G., David, E., & Lance, D. (2002). A Biologically Inspired Programming Model for Self-Healing Systems. WOSS’02 Proceedings of the First Workshop on Self-Healing Systems, Charleston, 102-104. https://doi.org/10.1145/582128.582149

Sidiroglou, S., Giovanidis, Y., & Keromytis, A. (2005). *A Dynamic Mechanism for Recovery from Buffer Overflow Attacks.* In Proceedings of the 8thInformation Security Conference (ISC2005), 1-15.

Sidiroglou, S., Laadan, O., Perez, C., Viennot, N., Nieh, J., & Keromytis, A. D. (2009). Assure: automatic software self-healing using rescue points. *ACM SIGARCH Computer Architecture News, 37*(1), 37-48.

Sidiroglou, S., Locasto, M. E., Boyd, S. W., & Keromytis, A. D. (2005). Building a Reactive Immune System for Software Services. In Proceedings of the 2005 USENIX Annual Technical Conference (USENIX 2005), pages 149-161, Apr. 2005.

Simmonds, J., Ben-David, S., & Chechik, M. (2010, November). Guided recovery for web service applications. In Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering (pp. 247-256). ACM.

Thorat, P., Raza, S. M., Nguyen, D. T., Hyunseung, G., Choo, I., & Kim, D. S. (2015). Optimized Self-Healing Framework for Software Defined Networks. IMCOM’15 Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication, Article No. 7. https://doi.org/10.1145/2701126.2701235

Tianfield, H., & Unland, R. (2004). Towards autonomic computing systems. *Engineering Applications of Artificial Intelligence, 17*(7), 689-699.

Tom, J., Arjan, R. A., & van Gemund, J. C. (2009). Zoltar: A Spectrum-Based Fault Localization Tool. SINTER’09, Amsterdam.

Tucek, J., Newsome, J., Lu, S., Huang, C., Xanthos, S., Brumley, D., Zhou, Y., & Song, D. (2007). *Sweeper: A Lightweight End-ToEnd System for Defending Against Fast Worms.* In Proceedings of the 2ndEuropean Conference on Computer Systems (EuroSys 2007), 115-128, Mar. 2007.

Vasar, M., Srirama, S. N., & Dumas, M. (2012, August). Framework for monitoring and testing web application

scalability on the cloud. In Proceedings of the WICSA/ECSA 2012 Companion Volume (pp. 53-60). ACM.

Zohrevandi, M., & Bazzi, R. A. (2013, December). Auto-FBI: a user-friendly approach for secure access to sensitive content on the web. In Proceedings of the 29th Annual Computer Security Applications Conference (pp. 349-358). ACM.