Scientific Research Publishing

Journal of Software Engineering and Applications, 2017, 10, 273-287
http://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

CACS: Cloud Environment Autonomic Computing System

Rizik M. H. Al-Sayyed[1], Hussam N. Fakhouri[2], Sharefa F. Murad[3], Sandi N. Fakhouri[4]

[1]King Abdullah II School for Information Technology, Department of Business Information Technology, The University of Jordan, Amman, Jordan
[2]King Abdullah II School for Information Technology, Department of Computer Science, The University of Jordan, Amman, Jordan
[3]Faculty of Computer Science, Middle East University, Amman, Jordan
[4]Income and Sales Tax Department, Amman, Jordan
Email: r.alsayyed@ju.edu.jo, h.fakhouri@ju.edu.jo, smurad@meu.edu.jo

How to cite this paper: Al-Sayyed, R.M.H., Fakhouri, H.N., Murad, S.F. and Fakhouri, S.N. (2017) CACS: Cloud Environment Autonomic Computing System. *Journal of Software Engineering and Applications*, **10**, 273-287.
https://doi.org/10.4236/jsea.2017.103016

Received: October 17, 2016
Accepted: March 20, 2017
Published: March 23, 2017

Abstract

This work proposes the adoption of Autonomic Computing System (ACS) in Cloud environment. ACS was first introduced by IBM to create systems capable of managing automatic self-configuration, self-healing, self-optimization and self-protection. These systems detect errors that cause failure, and then recover and reconfigure itself. The concept is wildly adapted by many software applications that have many restoring and recovery functionality such as operating systems (e.g. Windows Server 2012). This paper proposes a cloud ACS (CACS) for cloud computing environment that monitors, diagnoses, checks and heals cloud applications automatically and immediately with almost unnoticeable recovery time. In order to evaluate CACS, an application has been developed and applied for real time cloud applications. The results of different experiments scenarios demonstrate the ability of adopting the proposed system to heal well cloud applications. CACS is also compared with Windows Server 2012 operating system in terms of healing ability, speed, cost, methodology and other informative information. CACS showed domination in almost all of these properties.

Keywords

Cloud Applications, Self-Healing, Auto-Restoring, Auto-Backup

1. Introduction

The wide and fast spread of Internet motivated large number of companies to adopt the cloud solution and offer their services and business online, *i.e.* through the World Wide Web (WWW). This wide spread requires more research to be

developed to handle this quota of cloud applications in order to manage and self-heal themselves. ACS was first proposed by IBM in 2007 [1]. ACS includes the ability of the application to recover itself by detecting any fault or unexpected authorized and unauthorized changes on applications' files. Cloud ACS (CACS) applications require a 24/7 auto-monitoring of the applications as well as a fast recovery mechanism that keeps the online functionality and service of applications offered to customers available all the time. The importance of developing fast CACS is motivated by the effect that may occur to business applications and the need to avoid any intermission of these running applications even for a short period of time. For example, an online business such as a bank may lose customers' trust and lose financially if any customers-related application stopped functioning for just few hours. Many factors may affect online applications and cause them to stop as mentioned by Qin *et al.* in [2]. These factors may be either internal or external. Viruses and worms for example are considered internal factors that may affect the server that hosts the applications; however, we will not consider internal factors in this paper. On the other hand, the external factors include many attackers that attack the application and change the content of cloud files for different reasons, including the use of different methods such as xss, sql injection [3].

In cloud environment, the application on hosting cloud server could face many problems including the deletion, replacement or modification of a component. The risk of having one of these three problems is very high. For instance, when an attacker replaces an application component by another one that functions in the same way as the original one but has minor changes, it could (as an example) allow the attacker to steal credit card information which will cause a serious problem for both customers and owners. Most online cloud applications owners do not perform tests to check if the component has been changed or not; this is due to the complex architecture of this kind of applications and the lack of knowledge at the owner level. This paper proposes a solution to such problem and many others by applying CACS that has the functionality of self-healing, self-monitoring, self-diagnosis and self-recovering to keep the cloud application in good health!

Software systems have many anomalous conditions that appear among the components of software systems. To handle such situations the software architecture for this purpose has been splitted into two layers: functional layer and healing layer. This type of software systems provides software with many capabilities [4].

Some authors introduce architecture for hybrid software models which combine "endogenous" and "exogenous" approaches [5] where the architecture of the multi-agent system flexibly allows agents to adapt the changes in behaviour of the context providing a cooperative adaptation in the system.

This research, mainly focuses on techniques for "self-healing" cloud applications from functional failures by automatically detecting failures, diagnosing faults, and healing these applications to behave and run as supposed to before

the failure happens.

To evaluate CACS mechanism, we perform a black box testing on the tested software considering the whole cloud application files as one component. The main goal is to ensure that this component runs well and has not been accidentally or intentionally changed when compared with the original file. CACS ensures that all software components remain the same, without any modifications or changes by any other external authorized and unauthorized effect and to ensure that these components have not been omitted or deleted from the server and that the application's directory does not contain any injected or added files.

Our research suggests the existence of a system that analyzes the content of the released application components, a mechanism for monitoring the application, a mechanism for diagnosing and detecting of failure and a healing mechanism that brings back the software to its healthy status.

The major contribution of this paper is to define an automatic mechanism for cloud applications fault recovery despite the cause of the fault. In summary the research defines a mechanism for an external self-healing software application that monitors, diagnoses and detects a failure automatically and efficiently. The development and implementation of CACS considered a framework that managed cloud application files regardless of their programming language. We also provide an experimental result that demonstrates the efficiency of the proposed system.

The reset of this paper is organized as follows: Section two presents related work on ACS, and the mechanisms that are used for recovering different application; Section three presents a full description of the proposed system; Section four demonstrates the evaluation experiments and discusses their results; and finally Section five provides the conclusion and future work.

2. Related Work

A framework for runtime monitoring and recovery of cloud service conversations is proposed by Simmonds *et al.* [6]. An improved development environment of self-healing software to show the capabilities of self-managing was proposed by Park *et al.* [7]. A hybrid approach for self-management that combines an endogenous self-adaptation approach with an exogenous self-healing approach was proposed by Weyns *et al.* [8]. A framework that helps in monitoring and testing scalability of cloud applications on the cloud was proposed by Vasar *et al.* [9]. A novel and simple approach for securing access to sensitive content on the cloud and web environment where the content of sensitive data is protection of the content is proposed by Zohrevandi *et al.* [10].

Athanasopoulos *et al.* [11] proposed an approach for mining service abstractions, based on an agglomerative clustering algorithm. Their experimental findings suggest that the approach is promising and can serve as a basis for future research. Newsome *et al.*, [12] a system for software with self-diagnose and self-monitor, the main concern was to make the system recover and heal from different attacks and vulnerabilities, he wanted to make a better root with self-

healing property. They classified the process into two layers: healing layer and functional layer, by the inspiration of the human biology characteristics and methods to make benefit of the normal biological process (*i.e.* wound healing) [13]. An overview of the existing approaches that focus on the self-healing branch of the research was proposed by Harald *et al.* [14]. The importance of communication in self-healing and how its failure affect the recovery and healing of the software, was illustrated by using the architectural model of Dabrowski *et al.* [15]. 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 was proposed by Elkorobarrutia *et al.* [16]. Self-recovery is very essential in the world of continuous attack. Its importance came into existence due to the good efforts it offers as an automatic method; without human interaction; to recover information and data and offer a security level for interrupted services; this model was proposed by Anand *et al.* [17] to detect and recover failure.

3. Proposed System for Cloud Autonomic Computing System (CACS)

CACS consists of automatic exterior healing system that monitors cloud files and manage to maintain it unchanged at 24/7 working rate. The proposed system apply black box testing concept to verify the stability of the cloud applications files. Hence there is no need to examine neither the internal code nor the flow of its internal functions; rather than that CACS conceders the clouds application files as one component and test its characteristics such as the existence of the component, file size, hash key, creator and its correct location path. The system monitors, diagnoses and recovers the cloud application files immediately at the time of the external or internal effect that could cause any unexpected change. In order to achieve that the proposed system were designed with three main phases and have a life cycle run to guarantee the full time running of the cloud files.

Figure 1 demonstrates the three main phases of CACS: phase 1: pre-healing, phase 2: healing process and phase 3: post healing. The following subsections describes in details their main structure and functionality.

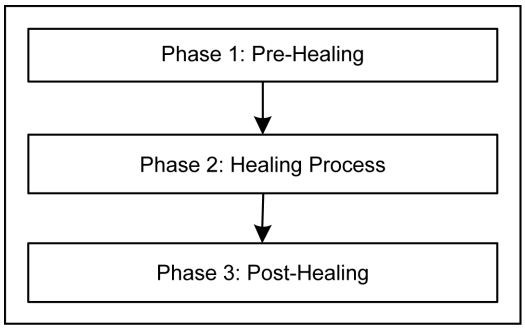


Figure 1. CACS main phases.

3.1. Phase 1: CACS Pre-Healing

Pre-healing phase is the initial phase that prepares the system for the healing phase. Starting by initializing the system and goes throw building the CACS database. After that the cloud application files (***cloudsite***) were identified and backup copies were created. The Pre-Healing phase also consists of running the implemented system settings for the first time to determine and select the specific folder for the cloud application to monitor and sets the initial parameters needed.

This phase also analyse the ***cloudsite*** files by gathering information such as: the file size, date of creation, manufacturers of the file and the hash key. The output of the released application is the input of the self-healing phase. As shown in Figure 2, the output component is mainly the cloud applications files and other files such as the assemblies of the ***bin*** folder.

Moreover, this phase includes building a database that stores all the information's about the cloud application that results from the analysis step, containing the major and necessary information for the diagnosis process. The aim of using a database is to keep a fast and organized method for diagnosing and referencing cloud application components for any time in order to access a review or make diagnosis. The phase also comprises creating a copy of original components of the ***cloudsite*** to be reused later in the healing phase. This copy will be compressed and stored in a separated directory specified by the CACS system and not on the published cloud directory. The CACS system is independent programming language; this enhance the system with the capability of analysing any type of cloud application files in any programming language such as PHP, ASP, HTML, etc. Figure 2 illustrates a flowchart for CACS steps in Pre-Healing Phase.

3.2. Phase 2: CACS Self-Healing

This phase consists of four basic processes; it starts by monitoring and ends by fixing process. Figure 3 illustrates the four main processes for CACS Self-Healing.

The first step in phase 2 is **monitoring**. In which the system observe the ***cloudsite*** component's (files) for 24/7, This includes tracking all the cloud application components as well as the ***cloudsite*** folder for any changes detected, including the deletion, replacing, modification and addition of any new component to the cloud directory folder.

The second step in phase 2 is **comparing**. This step conduct a deep comparison between the monitored components and the status record stored in the database by the analysis phase. As mentioned previously, the database contains full details of all components of the cloud application that are required for the diagnosis. The result provided by this step will be the input of the diagnosis step.

The third step in phase 2 is **diagnosis**. This step a decision will be made whether to make an action or not, by mean if the system needs to be healed or not (*i.e.* it is in good health). In this step a solution to the system will be required and a suggestion if the system is infected or becomes in faulty state or in a good health.

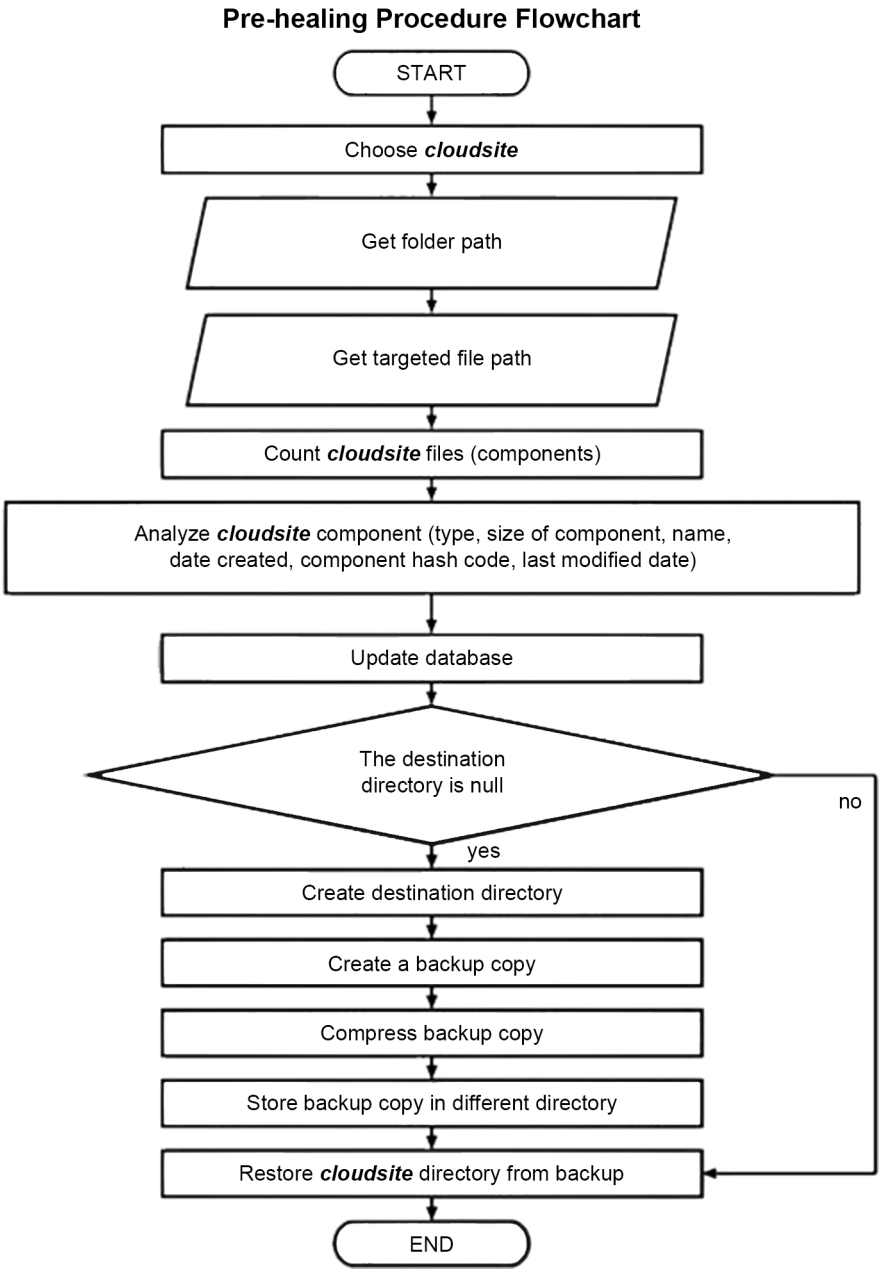


Figure 2. CACS pre-healing procedure flowchart.

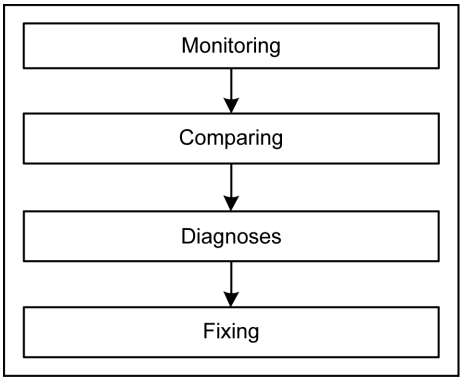


Figure 3. CACS self-healing processes.

The fourth step in phase 2 is **fixing**. It is the process of restoring the original component of the system and replacing or compensating the affected component in order to maintain the system in a good health. In this step the solution to the problem suggested by the diagnosis step is applied; when a fault is detected, the latest saved copy of the application monitored will be restored to the ***cloudsite*** directory. The restoring is triggered by detecting a change. This process will take only few seconds before the application can restart online and became available again for the users. The changes along with the healing event will be stored in the database for further analysis and the process is automated.

To take a deep look at CACS self-healing, we present a full details flowchart for the mechanism in Figure 4.

3.3. Phase 3: CACS Post-Healing

After the healing process ends, the post healing phase starts. Figure 5 shows the three steps involved in this phase: storing changes in the database, storing affected component and analysing reasons, and updating all cloud application components.

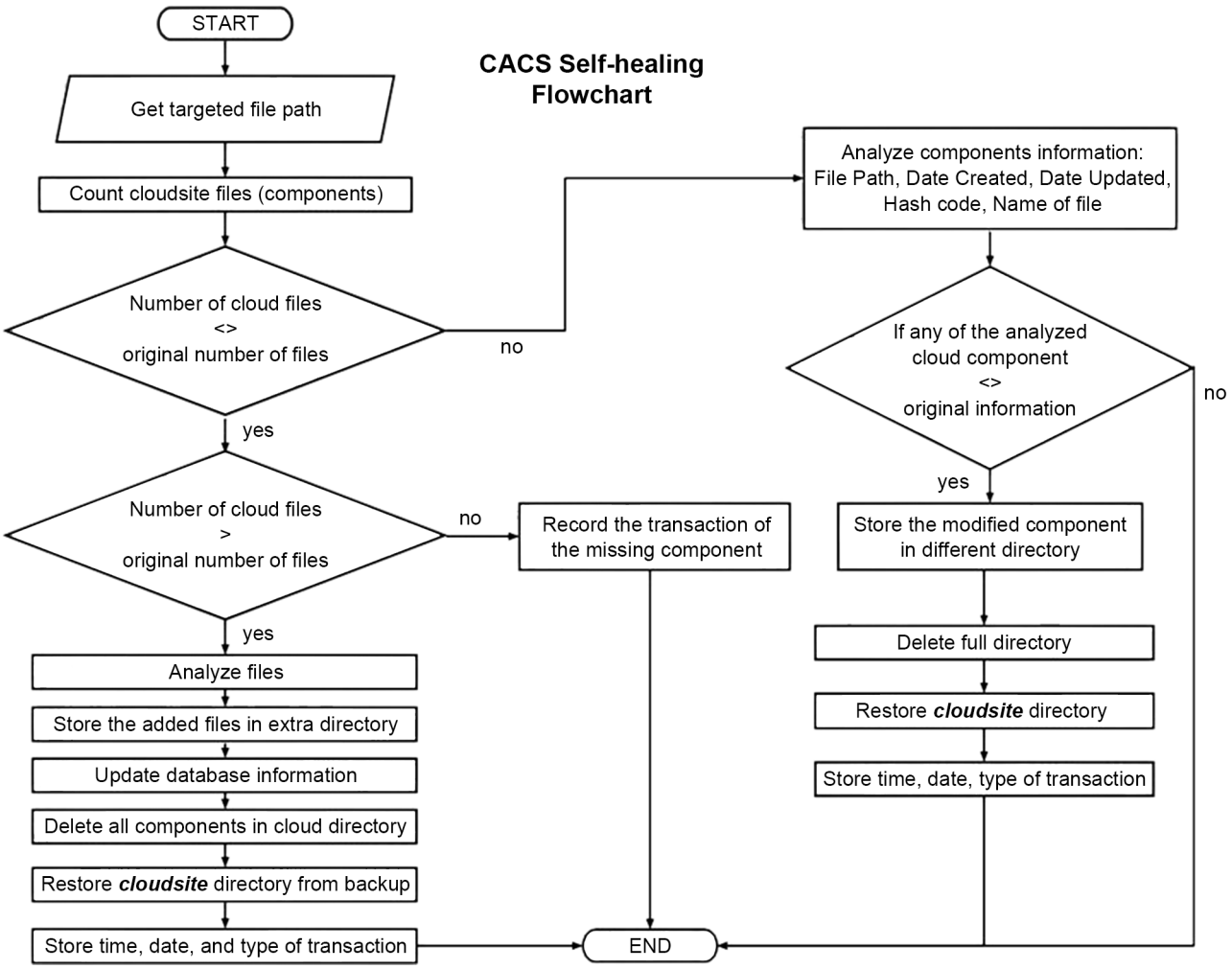


Figure 4. CACS self-healing flowchart.

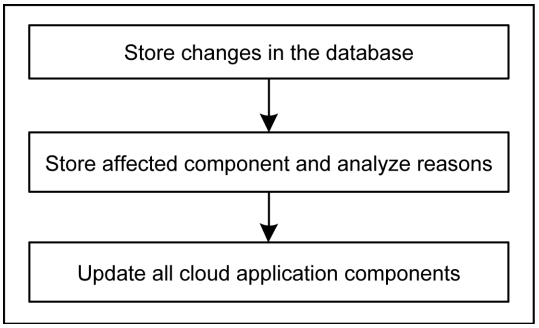


Figure 5. CACS post-healing processes.

The first step in phase 3 is **storing change** in the database. This process records all the information that has been done in the healing process including storing the date and time of healing and the component that has been restored. Storing this information will give the administrator a clear summary about the history of the application after releasing it.

The second step in phase 3 is **storing affected component and analysing reasons**. If the healing process resulted from a change in the component itself either for any of the mentioned reason then keeping this file will give us indicator about the reason that caused the fault and this will help the application developers to avoid such situations and to enhance or develop mechanisms to update the software or the server so that it can face such cases. For example, if the reason for a change was due to an illegal access to the server, then a certain policy could be in effect but if the change was due to a virus then the server should act by clearing the virus itself.

The third step in phase 3 is **updating all cloud application components**. The analysis process is an important step to maintain a future enhanced healthy cloud application because of the previously mentioned reasons and due to the fact that the analysis process results can be used to enhance and update the cloud application itself, and in the case of distributing the application to may servers, the updated component can be distributed to other servers as a precaution to avoid been infected by the same way.

CACS healing involves the following cases:

- Deletion of a component that causes the system to fail to run
- Change of a component by external factor either human or non-human
- Original component replacing
- Addition of external component to the software folder

CACS dynamically modifies the ***cloudsite*** to correct the failure. The changes that have been made will be stored to be analysed in the future by the system administration. However if the same fault is frequently repeated this may indicate the need to analyse the stored information's about the recovery processes that has been made to the affected components.

Analysing the changes along with the results of checking the diagnosis and monitoring will provide a good indicator about the reasons that cause the system failure. It also gives a brief overview about the main causes and their indicators.

By defining the reason the system administrator can find an appropriate solution to handle the problem for good.

4. Evaluation

A research method or tool has more chances to be transferred to practitioners if its usefulness is investigated through empirical user studies [18]. In order to evaluate the usefulness of the research presented in this paper an empirical test has been conducted. This section evaluates the ability of CACS to recover from different modifications and unauthorized changes to the files of the cloud applications. In order to do that the following research questions were and then different scenarios have been made to illustrate each case:

- What advantages can we get when using CACS to heal cloud applications that are affected by different performance scenarios?
- What is the time of heal using CACS when compared with other healing approaches?

There are three sources that might affect the ***cloudsite*** components:

1) external non-human factors

- virus
- Worm

2) software

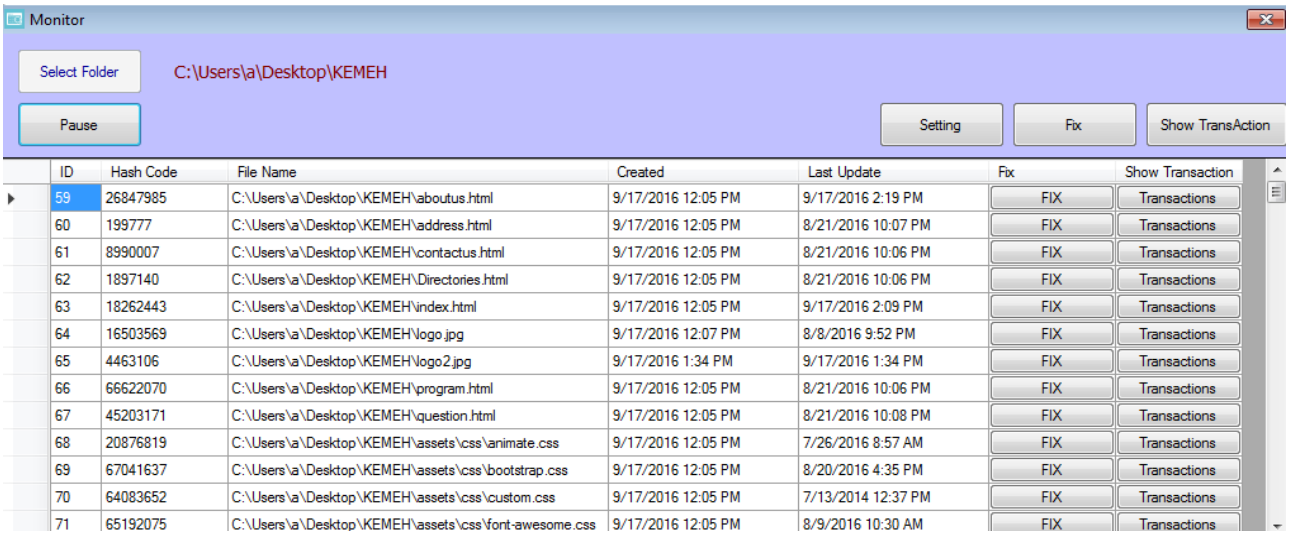
- defect in the components
- conflict with other software
- operating system related

3) external human factors

- attacker
- spy
- fraud

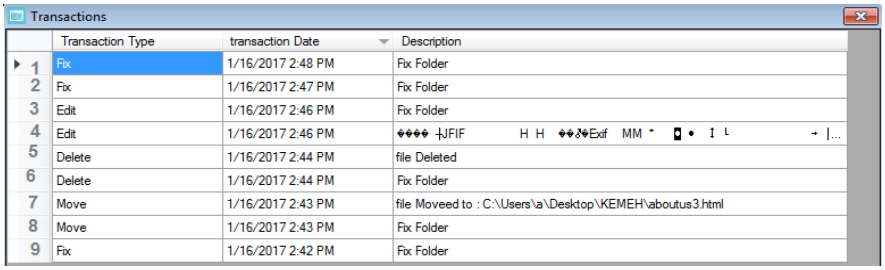
To evaluate the proposed approach we need to evaluate the effectiveness and the ability of the proposed system to recover from any different failure causes. To this reason four experimental scenarios were tested: deletion of a file, moving of a file, replacement of a file and editing a file. We initialized the implemented auto cloud application monitoring system and selected the cloud application directory to be monitored. CACS will analyse the ***cloudsite*** directory and build the database; see Figure 6. The second phase of the experiment will test the effectiveness of the system to heal the deletion case by executing the application and after that a file will be deleted from the cloud directory. To test the effectiveness and performance of the system in detecting the problem of replacing a component, we created a file name with the same name and extension of a specific file on the cloud application directory. For testing the final case of editing a component, we considered manual modification of the component and this is the human modification.

In Figure 7, we illustrated a full description of all the transactions collected. All transactions are sorted in descending order according to date and time. More description of Figure 7 is furnished in the three subsequent sections.



	ID	Hash Code	File Name	Created	Last Update	Fix	Show Transaction
▶	59	26847985	C:\Users\a\Desktop\KEMEH\aboutus.html	9/17/2016 12:05 PM	9/17/2016 2:19 PM	FIX	Transactions
	60	199777	C:\Users\a\Desktop\KEMEH\address.html	9/17/2016 12:05 PM	8/21/2016 10:07 PM	FIX	Transactions
	61	8990007	C:\Users\a\Desktop\KEMEH\contactus.html	9/17/2016 12:05 PM	8/21/2016 10:06 PM	FIX	Transactions
	62	1897140	C:\Users\a\Desktop\KEMEH\Directories.html	9/17/2016 12:05 PM	8/21/2016 10:06 PM	FIX	Transactions
	63	18262443	C:\Users\a\Desktop\KEMEH\index.html	9/17/2016 12:05 PM	9/17/2016 2:09 PM	FIX	Transactions
	64	16503569	C:\Users\a\Desktop\KEMEH\logo.jpg	9/17/2016 12:07 PM	8/8/2016 9:52 PM	FIX	Transactions
	65	4463106	C:\Users\a\Desktop\KEMEH\logo2.jpg	9/17/2016 1:34 PM	9/17/2016 1:34 PM	FIX	Transactions
	66	66622070	C:\Users\a\Desktop\KEMEH\program.html	9/17/2016 12:05 PM	8/21/2016 10:06 PM	FIX	Transactions
	67	45203171	C:\Users\a\Desktop\KEMEH\question.html	9/17/2016 12:05 PM	8/21/2016 10:08 PM	FIX	Transactions
	68	20876819	C:\Users\a\Desktop\KEMEH\assets\css\animate.css	9/17/2016 12:05 PM	7/26/2016 8:57 AM	FIX	Transactions
	69	67041637	C:\Users\a\Desktop\KEMEH\assets\css\bootstrap.css	9/17/2016 12:05 PM	8/20/2016 4:35 PM	FIX	Transactions
	70	64083652	C:\Users\a\Desktop\KEMEH\assets\css\custom.css	9/17/2016 12:05 PM	7/13/2014 12:37 PM	FIX	Transactions
	71	65192075	C:\Users\a\Desktop\KEMEH\assets\css\font-awesome.css	9/17/2016 12:05 PM	8/9/2016 10:30 AM	FIX	Transactions

Figure 6. Analysed cloud application directory.



	Transaction Type	transaction Date	Description
▶ 1	Fix	1/16/2017 2:48 PM	Fix Folder
2	Fix	1/16/2017 2:47 PM	Fix Folder
3	Edit	1/16/2017 2:46 PM	Fix Folder
4	Edit	1/16/2017 2:46 PM	♦♦♦♦ JFIF H H ♦♦Exif MM * ♦ • I └ → |...
5	Delete	1/16/2017 2:44 PM	file Deleted
6	Delete	1/16/2017 2:44 PM	Fix Folder
7	Move	1/16/2017 2:43 PM	file Moveed to : C:\Users\a\Desktop\KEMEH\aboutus3.html
8	Move	1/16/2017 2:43 PM	Fix Folder
9	Fix	1/16/2017 2:42 PM	Fix Folder

Figure 7. Transactions description.

4.1. Scenario1: Deletion of a Component

CACS responds to this case by restoring the deleted file from the original copy that has been prepared in the initialization stage. CACS records the problem in the database including the time, date, type of problem and the name of the file that was deleted and replaced see Figure 7, line 5. CACS was very efficient to recover different cloud application component extensions that were tested including PHP, HTML, ASPX, DLL, and CSS. Figure 8 shows the flowchart for the recovery process.

4.2. Scenario 2: Replacement with Similar Component

CACS responds to this case by deleting the full directory of the cloud application and restoring the original copy of the ***cloudsite*** that has been prepared in the initialization stage. CACS records the problem in the database including the time, date, and type of problem and the name of the file that was replaced and recovered; see Figure 7, line 7. Again, CACS was very efficient to recover different cloud application component extensions that were tested including PHP, HTML, ASPX, DLL, and CSS (Figure 9).

4.3. Scenario 3: Modifying a Component

CACS responds to this case by deleting the full directory of the cloud application

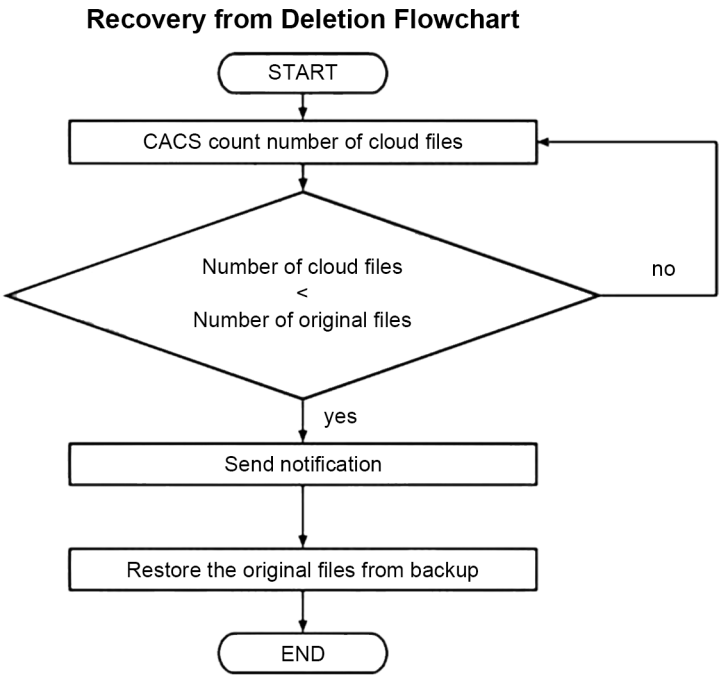


Figure 8. Recovery from Deletion.

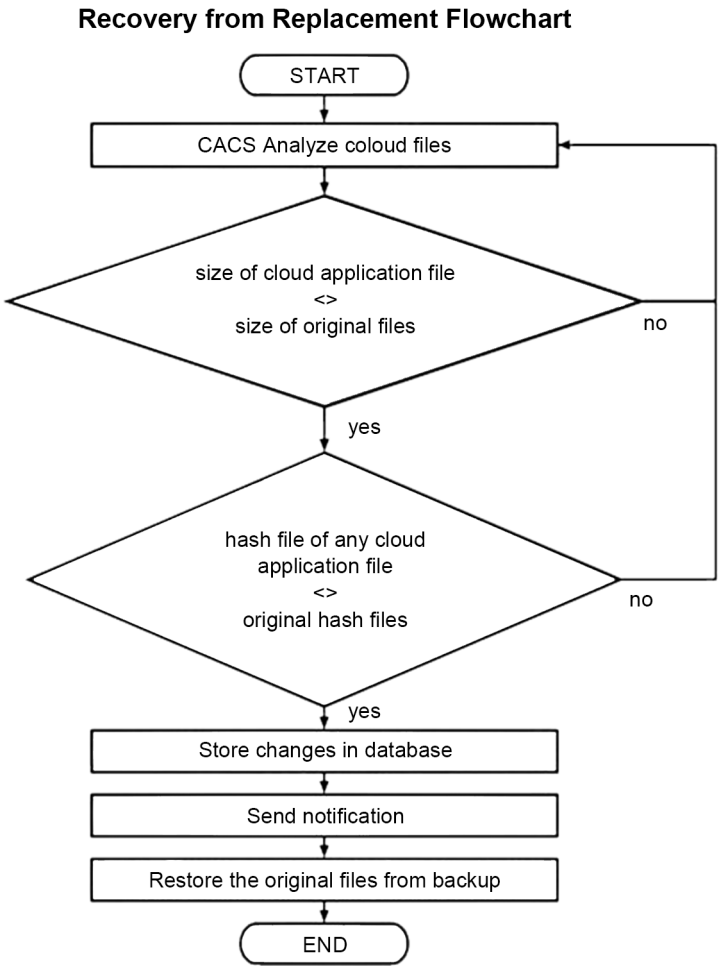


Figure 9. Recovery from replacement.

and restoring the original copy of the ***cloudsite*** that has been prepared in the initialization stage. CACS records the problem in the database including the time, date, and type of problem and the name of the file that was recovered see Figure 7, line 4. Also, CACS was very efficient to recover different cloud application component extensions that were tested including PHP, HTML, ASPX, DLL, and CSS (Figure 10).

4.4. Measuring the Time Required to Heal a Process

In this experiment, we added ***cloudsite*** files of size 10 g and then we deleted 5 m of the file as shown in Table 1. The aim of the experiment is to measure the time needed to heal the system and compare it with windows server system restore.

Since the CACS heals by recovering the ***cloudsite*** files (components) and not the full system restore or recovery, we notice that windows system restoration works by restoring all files in windows server 2012. This took about 2400 second while in CACS took only 5 seconds; this clearly makes the CACS a better choice.

In Table 2, we listed an exhaustive comparison between healing using CACS and Microsoft Windows Server 2012 healing system; this is to give a clear picture about the benefits of employing CACS for complex environments such as cloud.

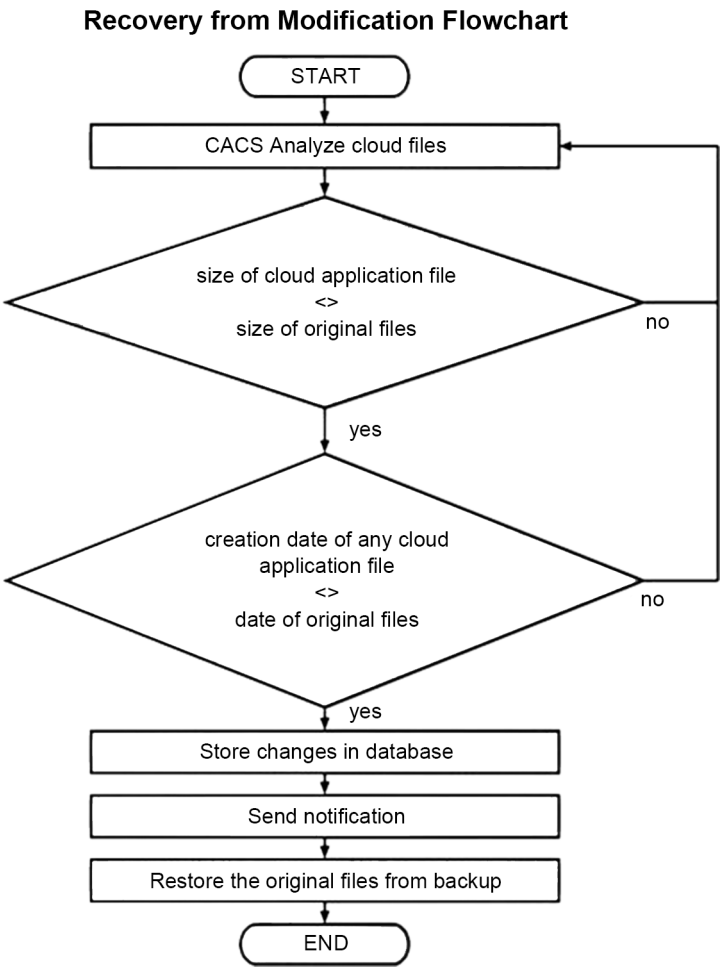


Figure 10. Recovery from modification.

Table 1. Comparison of CACS with windows system restore [windows server 2012].

Method	Size on server	Size of *cloudsite* files	Time for healing
CACS	12 g	10 m	5 seconds
Windows server 2012	12 g	10 m	40 × 60 (2400) seconds

Table 2. Comparison between the proposed system (CACS) and Microsoft Windows Server 2012.

Criteria	Microsoft Windows [System Restore]	Proposed system (CACS)	Antivirus	Firewall	Spyware	Reinstall the cloud server
Recover error resulting from deleting software component	Yes	Yes	No	No	No	Yes
Heal Replaced component that has same functionality	No.	Yes	No	No	No	Yes
Heal at run time	No	Yes	Yes	No	Yes	No
Generate reports of the diagnosis of the problem and the healing process	Yes	Yes	Yes	Yes	Yes	No
Store the affected component for future analysis	No	Yes	Yes	No	Yes	No
Methodology of repairing	Operating system dependent	Automatically Compares, analyses, diagnoses and heals the cloud application files; it returns the file to its original state similar to the manufacturer from the backup files	Only files changed by virus signature or worms	No repairing	Only files changed by virus signature or worms	Install fresh new operating system
State of the healing	To a specified restore point	To the manufacturer state either the original or with updates	No healing	No healing	No healing	No healing
Level of recovery	Full restore	Per file	Per file	No recover y	Per file	Full
Speed of recovery	Relatively slow at least 10 min	Fast less than 1 min (recover only the affected file)	No recovery	No recovery	No recovery	Relatively slow at least 20 min

As can be inferred from Table 2, CACS dominates all the properties; the table is a strong evidence that CACS is better Microsoft Windows system in terms of healing ability, speed, cost, methodology, and other informative information.

5. Conclusions and Future Work

Integrating self-healing approaches into ***cloudsites*** introduces a very efficient improvement for the ***cloudsites*** performance. Many companies tried different methods and approaches that aim at reducing the cost and time needed for the rerun of the ***cloudsites*** after failures and tried to build a software system that has the ability to heal itself. This research presents CACS, an approach for self-healing ***cloudsites***, CACS monitors the software for 24/7 duration and it has the

ability to capture continual information about the specific ***cloudsite*** components that are being monitored. Our experimental results show the efficiency of CACS in detecting failures and errors and efficiency in healing them.

As a future work, we hope that our work may inspire biological software engineering processes aiming to improve the self-learning of the proposed approach and to generalize the concept to self-learning and self-adaptation.

References

[1] Nami, M.R. and Bertels, K. (2007) A Survey of Autonomic Computing Systems. In 3*rd International Conference on Autonomic and Autonomous Systems*, Athens, 19-25 June 2007, 26. https://doi.org/10.1109/conielecomp.2007.48

[2] Qin, F., Tucek, J., Sundaresan, J. and Zhou, Y. (2005) Rx: Treating Bugs as Allergies—A Safe Method to Survive Software Failures. *ACM SIGOPS Operating Systems Review*, **39**, 235-248. https://doi.org/10.1145/1095809.1095833

[3] Tucek, J., Newsome, J., Lu, S., Huang, C., Xanthos, S., Brumley, D. and Song, D. (2007) Sweeper: A Lightweight End-to-End System for Defending against Fast Worms. *ACM SIGOPS Operating Systems Review*, **41**, 115-128. https://doi.org/10.1145/1272998.1273010

[4] Ghosh, D., Sharman, R., Rao, H.R. and Upadhyaya, S. (2007) Self-Healing Systems—Survey and Synthesis. *Decision Support Systems*, **42**, 2164-2185.

[5] Weyns, D. (2010) Capturing Expertise in Multi-Agent System Engineering with Architectural Patterns. In: *Architecture-Based Design of Multi-Agent Systems*, Springer, Berlin Heidelberg, 27-53.

[6] Simmonds, J., Ben-David, S. and Chechik, M. (2010) Guided Recovery for Web Service Applications. *Proceedings of the* 18*th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Santa Fe, 7-11 November 2010, 247-256. https://doi.org/10.1145/1882291.1882328

[7] Park, J., Youn, H. and Lee, E. (2009) An Automatic Code Generation for Self-Healing. *Journal of Information Science and Engineering*, **25**, 1753-1781.

[8] Hudaib, A.A. and Fakhouri, H.N. (2016) An Automated Approach for Software Fault Detection and Recovery. *Communications and Network*, **8**, 158.

[9] Vasar, M., Srirama, S.N. and Dumas, M. (2012) Framework for Monitoring and Testing Web Application Scalability on the Cloud. *Proceedings of the WICSA/ECSA* 2012 *Companion*, Helsinki, 20-24 August 2012, 53-60. https://doi.org/10.1145/2361999.2362008

[10] Zohrevandi, M. and Bazzi, R.A. (2013) Auto-FBI: A User-Friendly Approach for Secure Access to Sensitive Content on the Web. *Proceedings of the* 29*th Annual Computer Security Applications Conference*, New Orleans, 9-13 December 2013, 349-358. https://doi.org/10.1145/2523649.2523683

[11] Athanasopoulos, D., Zarras, A.V., Vassiliadis, P. and Issarny, V. (2011) Mining Service Abstractions (NIER Track). *Proceedings of the* 33*rd International Conference on Software Engineering*, Waikiki, 21-28 May 2011, 944-947.

[12] Newsome, J., Brumley, D., Song, D. and Pariente, M.R. (2005) Sting: An End-to-End Self-Healing System for Defending against Zero-Day Worm Attacks on Commodity Software. CMU-CS-05-191.

[13] Hudaib, A.A., Fakhouri, H.N., Al Adwan, F.E. and Fakhouri, S.N. (2017) A Survey about Self-Healing Systems (Desktop and Web Application).

[14] Psaier, H. and Dustdar, S. (2011) A Survey on Self-Healing Systems: Approaches and Systems. *Computing*, **91**, 43-73. https://doi.org/10.1007/s00607-010-0107-y

[15] Dabrowski, C. and Mills, K. (2002) Understanding Self-Healing in Service-Discovery Systems. *Proceedings of the* 1*st Workshop on Self-healing Systems*, Charleston, 18-19 November 2002, 15-20. https://doi.org/10.1145/582128.582132

[16] Elkorobarrutia, X., Izagirre, A. and Sagardui, G. (2006) A Self-Healing Mechanism for State Machine Based Components.

[17] Padwalkar, A., Patil, S. and Mogre, N. (2015) Designing an Application for Recovery of Data in Cloud Environment: A Problem Definition.

[18] Peeger, S.L. and Menezes, W. (2000) Marketing Technology to Software Practitioners. *IEEE Software*, **17**, 27-33. https://doi.org/10.1109/52.819965.