HIGH RELIABILITY APPROACHES IN CLOUD APPLICATIONS FOR BUSINESS – RELIABILITY AS A SERVICE (RAAS) MODEL

Orges Çiço[1], *Zamir Dika*[2], *Betim Çiço*[3]

[1] Faculty of Engineering, Canadian Institute of Technology, Tirana
[2] Faculty of Contemporary Sciences and Technologies, SEE University, Tetovo
[3] Department of Computer Engineering, EPOKA University, Tirana
e-mails: [1] orges.cico@cit.edu.al, [2] z.dika@seeu.edu.mk, [3] bcico@epoka.edu.al
[1, 3] Albania, [2] Macedonia

Abstract: We introduce in this paper the concept of a new model Reliability as a Service (RaaS) on cloud software systems. The concept is supported by common fault tolerant approaches which should be adopted as part of the cloud software architecture in order to obtain a reliability close to the five 9s (99,999%) as stated by Software Availability Forum (SAF). Since there is no one particular need for every software system architecture, the RaaS could constitute the building blocks for highly available cloud systems, by adopting and recommending possible optimal techniques throughout a well established framework. The article backs up most of the statements on case study that proves the efficiency of the fault tolerant techniques in improving overall system reliability.

Keywords: Cloud Software System, Reliability as a Service, Software Availability Forum

1. INTRODUCTION

Cloud based system will shape the future of computing in the next generations to come. The novel has started during the past decade due to the advantages cloud systems offer related to cheap setup time cost, high scalability, flexible resource exploitation on demand. Nowadays, main initial concerns in cloud systems have been related to security issues. However, recently the need for high availability has become a hot topic which obviously has direct cost impact. Cloud providers face several challenges regarding availability of their system in fulfilling the Service Level Agreement (SLA). Developers could contribute while exploiting cloud Software as a Service (SaaS), Platform as a Service (PaaS) or Infrastructure as a Service (IaaS). Thus, moving business applications, but not only, to the cloud will soon face the need for a framework and approaches to be adopted in order to ensure the development of qualitative and highly reliable software.

Many research works have provided a good evaluation of the different Cloud infrastructure and their reliability [1,3, 9]. However, almost never there have been standardized fault tolerant techniques or evaluation of the current ones on real case study. In this paper work, it is presented the exploitation of the design, data and temporal diversity on a Google Cloud Applications. The evaluation has been based on the well-known Software Reliability Engineering (SRE) process widely discussed in the past 40 years [10]. Usually distributed applications represent a great challenge to build a correct and complete operational profile, but in our case study we have put our focus mainly on the most critical operations and have applied the previously mentioned techniques only for the most critical software components.

Windfarmdesigns[1] has been developed in Django framework. The software includes a unique algorithm performing the optimization of the possible wind farm layouts [13]. The algorithm implementation has been done from separate development teams in two different programming languages Python and MATLAB based on N-Version Programming Design Diversity Technique [3]. The algorithms are run in parallel on the same Google Compute Engine (GCE) instance. The distributed architecture of the software system is a good example for proving the effectiveness of temporal diversity techniques to improve the availability/reliability of the overall system, at virtual machine instances creation in the cloud.

The second applied technique is related to temporal and data diversity based on Retry and Recovery control Blocks (RcB, RtB) when requesting cloud instance creation.

Both techniques implemented and following the methodology proposed from the SRE process mentioned earlier have provided proof of their effectiveness from the data obtained during the experiment.

Section 2 provides a quick overview on how cloud services work, while sections 3, 4 and 5 provide that actual state of the art, methodology and results obtained from the case study and the testing experiments. Conclusions and possible future work have been proposed at the end of the paper.

2. BACKGROUND

Most cloud systems are based on the following service layers of abstraction:

1. SaaS (Software as a Service)
2. PasS (Platform as a Service)
3. IaaS (Infrastructure as a Service)
4. MBaaS (Mobile Backend as a Service)

All the three layers are stacked up providing the backbone for most cloud architectures today. Clients usually exploit the services at different levels of abstraction based on their immediate necessities.

[1] Windfarmdesigns refers to the project found at http://www.windfarmdesigns.com

The Fig.1 describes a possible representation of the stacked up services and their interdependency with cloud users. The figure tries to present an encapsulated representation so that it can be obvious for the end user that services starting from core hardware augment themselves by reusing the other underlying services. This representation helps understanding the concept described in the paper where adding other service layers could involve some or all of the existing ones.

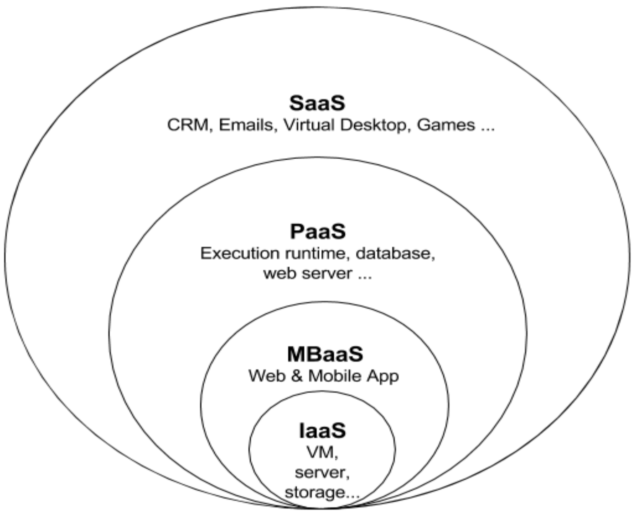
SaaS
CRM, Emails, Virtual Desktop, Games ...
PaaS
Execution runtime, database, web server ...
MBaaS
Web & Mobile App
IaaS
VM, server, storage...

Fig.1. Elliptic view of the cloud services & their interdependency with cloud users

Infrastructure as a Service (IaaS) offers what is commonly required as a basic functionality from the cloud system relates to its existing infrastructure in terms of hardware resources such as computation resources, data storing/backup, load balancing, scalability, security etc.

Platform as a Service (PaaS) offers the appropriate environment for developing and deploying applications. This are typically identified as services related to web servers and their execution environment for different programming and scripting languages, operating systems, databases etc.

Software as a Service (SaaS). Software distribution and their runtime environment is nowadays relying on distributed service providers, commonly known as cloud providers. In this case the provider is fulfilling all the management of the previously mentioned services IaaS and PaaS and the end user is only exploiting the different software functionalities, running on the cloud.

Mobile "backend" as a Service (MBaaS) cloud endpoints such as cloud storage, computing services, information/data retrieval etc. are accessed through well-defined application programming interfaces (APIs). This approach is fully integrated with services related to social media sharing and authentication; push up notifications etc. commonly exploited from mobile applications.

3. STATE OF THE ART OF ADOPTED FAULT TOLERANT AND HIGH RELIABILITY TECHNIQUES IN CLOUD SERVICES

Based on literature review the High Reliability (HR) concerns have been classified as follows based on a simplified view of Service Availability Forum (SAF):

1.Virtualization and Middleware approaches. The main types of failures addressed in the proposed framework are at different levels such as: a) Application, b) Virtual Machine and c) Host.

2. Layer based approach. Three different layers: a) Underlying Technologies - Addressing HR at Virtual Machine Level; b) Services - Providing fault tolerant mechanisms as services configurable from the cloud providers; c) Middlewares - Handle mainly service operation, configuration and decision making according to failure scenario.

3.1. Common Fault Tolerant techniques adopted at service level

3.1.1. Redundancy

The redundancy service can offer different levels of availability depending on the redundancy model, the redundancy strategy, and the redundancy scope.

The redundancy model, Fig. 2 refers to the many different ways HA systems can combine active and standby replicas of hosted applications.

There exist four models [7]: 2N, N+M, Nway and Nway active.

The 2N ensures one standby replica for each active application.

Moreover the scope of redundancy implementation could be at any layer such as: Application, Virtual and Physical Machine Layer.

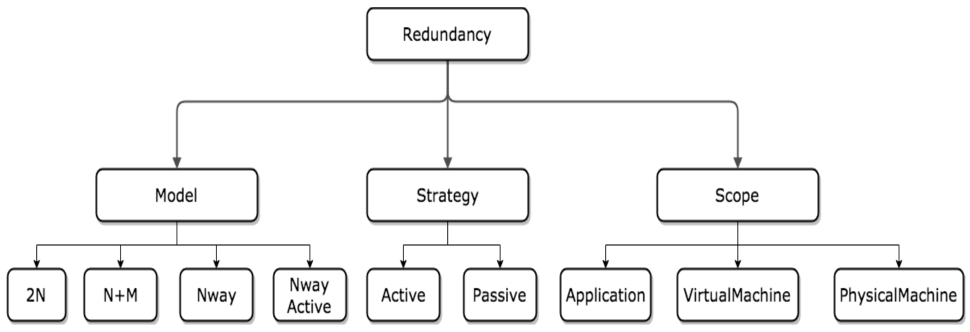

Fig. 2. The redundancy model

3.1.2 Data replication

Data replication, Fig. 3 is used to maintain state consistency between replicas. The main problem associated with this service is the question of how to govern the trade-off between consistency and resource usage [11]. In Cloud, the replication may be achieved either by copying the state of a system (checkpoint) or by replaying input to all replicas (lock-step based) [12].

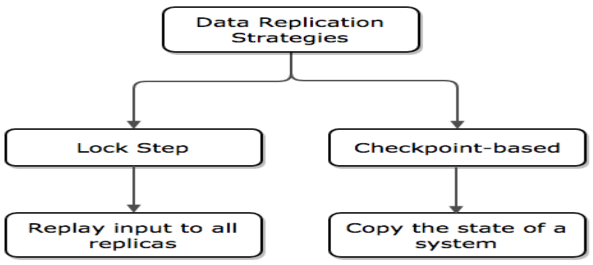
Data Replication Strategies
Lock Step
Checkpoint-based
Replay input to all replicas
Copy the state of a system

Fig. 3. Data replication classification

3.1.3. Monitoring

Monitoring is a crucial service in an HA Cloud. Through this service, applications' health is continuously observed to support other services. The primary goal of this service is to detect when a replica is down, but robust implementations can also follow the health indicators of an application (CPU and memory utilization, disk, and network I/O, time to respond requests) which will help to detect when a replica is malfunctioning [9]. It can also be done at virtual and physical machine level.

3.1.4. Failure detection

Failure detection is an important service contained in most HR solutions, which aims to identify systems' faults (application, virtual or physical machine level) and provide needed information for services capable of treating problems to maintain service continuity. The authors list some mechanisms used to detect faults like ping, heartbeat and exceptions. From this perspective, failure detection can be classified in two categories according to detection mechanisms: reactive and proactive [10].

The first approach waits for KEEP ALIVE messages, but it identifies a failure after a period of time waiting without any KEEP ALIVE message. The second approach is more robust and is capable of identifying abnormal behaviours in the environment, checking the monitoring service and interpreting collected data to verify whether there are failures or not.

3.1.5. Recovery

Software system and cloud service recovery is part of a broader fault tolerant process containing the steps previously described. The latter usually encompasses error/fault detection, debugging, isolation and recovery.

Different types of recovery have been proposed based on the forward and backward recovery. In cloud services these techniques heavily rely on redundancy [9] at different levels of operation (Virtual or Physical Machine).

Commonly backward recovery involves rolling back the system to a previous state stored as a failure free checkpoint of the system. The most adopted approach are Recover control Blocks (RcB).

Forward recovery tries to continue the system execution by identifying a new state running on a parallel entity. N-version Programming (NVP) is the most commonly adopted approach.

Exploiting the concept of backward and forward recovery where both attempt to return the system to a correct/error-free state the recovery techniques operating onto the cloud have been classified into simple and smart. The simple approach involves rebooting completely the running application on the same/different node (VM) but eventually all data and states information is lost. While the smart recovery exploits fault tolerant approaches adopted mostly by Backward recovery techniques (Monitoring/Checkpointing). They commonly involve creation of parallel backup replicas and checkpointing states so that they can switch their execution upon failure occurrence.

3.2. Other potential Fault Tolerant Techniques adopted at SaaS cloud service levels

Other potential techniques that could be adopted at different service level are the following: SCOP - Self Configuring Optimal Programming, developed by Bondavalli, Di Giandomenico, and Xu [5,6]. Provides with a scheme in optimizing dependability and efficiency, throughout the exploitation of a flexible redundancy architecture, by adjusting at runtime. It exploits redundancy like NVP previously described, growing syndrome space for information gathering and result selection.

Recovery Blocks (RcB) [3] scheme consists of several variants run from an executive and approved from an acceptance test. Sometimes real time implementations add to the scheme a watchdog timer (WDT). From Fig. 4 we can observe that first the RcB tries to ensure the AT the primary alternate through a try block. If the first one fails the others are tried until a successor a final failure is achieved. If a WDT is added then the alternates are tried until the deadline for retrials is not expired.

```
ensure     Acceptance Test
by         Primary Alternate
else by    Alternate 2
else by    Alternate 3
…
else by    Alternate 4
else failure exception
```

Fig. 4. Recovery Block Scheme

Various variants of the technique exist depending on the scenarios:

1. The Distributed Recovery Blocks DRB [2,8] combines distributed and parallel processing and recovery blocks. The technique is applicable at both software and hardware level. The technique exploits a pair of self-checking processing nodes or computing components structured as primary shadow pair, resident on different network nodes. The AT (Acceptance Test) ensures a result confirmation on the distributed nodes following the steps as in Fig. 5.

```
run        RB1 on Node 1 (Initial Primary),
               RB2 on Node 2 (Initial Shadow)
Ensure     AT on Node 1 or Node 2
by         Primary on Node 1 or Alternate on Node 2
else by    Alternate on Node 1 or Primary on Node 2
else failure exception
```

Fig. 5. Distributed RcB architecture

2. Retry Blocks (RtB) [4] is a data diverse technique complementing the previous RcB technique. Similarly it uses AT and Backward Recovery and WDT to ensure that if one algorithm fails the next one could be retried with re-expressed inputs. The schema is expressed in Fig. 6. Similar implementations and concept such as Exponential Backoff [2] are currently used from different cloud providers such as Google etc. However, adopting such approaches at cloud service level and not just end-user level would mitigate a lot of transient cloud issues. Thus cloud systems operating SaaS adopting RtB techniques as proved later in the paper work would ensure increased reliability of their services.

```
ensure          Acceptance Test
by              Primary  Algorithm(Original Input)
else by         Primary  Algorithm(Re-expressed  Input)
else by         Primary Algorithm(Re-expressed Input)
…
…               [Deadline Expires]
else by         Backup Algorithm(Original Input)
else failure exception
```

Fig. 6. RtB scheme

4. RaaS MODEL AND FRAMEWORK PROPOSAL

Most of the previously mentioned techniques have been adopted with an ad-hoc approach to different cloud architectures. However, since many of them address similar issues for different cloud vendors than a framework at cloud service level could be the right approach for future applications that have High Availability/Reliability requirements. Fig. 7 describes a possible cloud service offering Reliability as a Service (RaaS). In the figure can be noticed the two possibilities of applying fault tolerant techniques at different cloud services, as well as providing the end user with application level Framework that would ease the process of integrating high reliability in cloud application development.

4.1. Reliability as a Service (RaaS) Model

The techniques applied might be at different service layers, relying on the approaches mentioned earlier in section 3. Part of the techniques such as Retry Blocks (Data Diversity) can be applied at PaaS layer while others might also be

[2] Exponential Backoff refers to the concept related to error handling requests of end users. Further reference found at https://cloud.google.com/storage/docs/exponential-backoff

applied at different service layers in parallel (e.g. Design diversity and N-Version Programming). Some of the techniques are already available at IaaS layer such as storage redundancy, backup etc. The cloud service provider and the end user should have the freedom of adopting and choosing the service level where to apply the techniques, with minimum effort. Cloud developers should be supported from a framework, which they can exploit to easily build highly reliable cloud applications. Such framework is described in the next section.

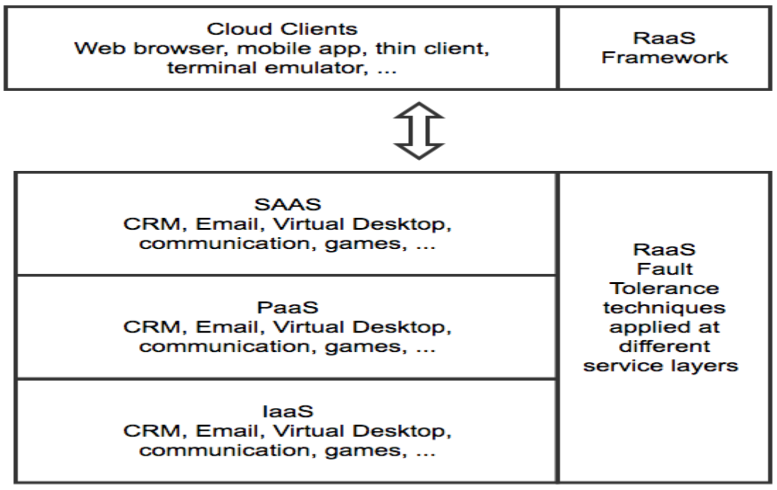
Cloud Clients
Web browser, mobile app, thin client, terminal emulator, ...
RaaS Framework
SAAS
CRM, Email, Virtual Desktop, communication, games, ...
PaaS
CRM, Email, Virtual Desktop, communication, games, ...
IaaS
CRM, Email, Virtual Desktop, communication, games, ...
RaaS Fault Tolerance techniques applied at different service layers

Fig. 7. RaaS Multilevel Service Layer

4.2. Reliability as a Service (RaaS) Framework

In order for the end user (cloud clients) to easily exploit the different fault tolerant techniques a framework solution is proposed especially for PaaS layer, so that coding techniques can be directly applied to the software components implemented from the developer. The latter can manually implement the code or in some cases rely on code auto generation, both part of the framework. It will initially imply development overhead, which will pay off especially in production mode of the cloud software system. The other two service layers might be maintained from the framework especially in terms of manually enabling or not of the different fault tolerant techniques along the PaaS and IaaS layers. In order to automates the process most of the techniques should be fully integrated with proper interfaces (APIs) within the cloud applications. The Fig. 8 provides a possible overview of the framework architecture proposed from us.

From the Fig. 8 we can observe that after first deployment the developer has GUI based option to select from the type of fault tolerant technique to be applied on different service layer for different functionalities translated into unit entities depending on the layer (at RaaS level this can be easily identified as coding unit, the other levels are harder to address and specify a proper unit entity).

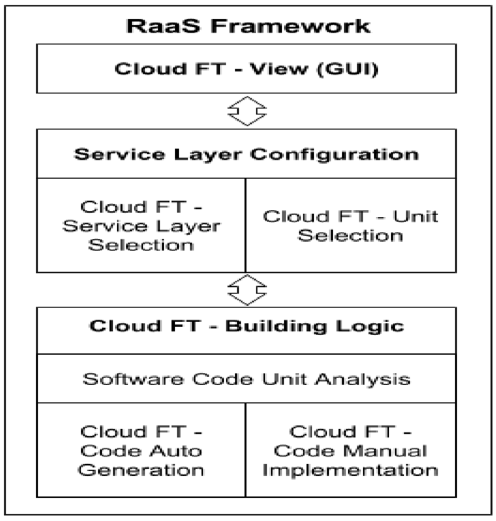
RaaS Framework
Cloud FT - View (GUI)
Service Layer Configuration
Cloud FT - Service Layer Selection
Cloud FT - Unit Selection
Cloud FT - Building Logic
Software Code Unit Analysis
Cloud FT - Code Auto Generation
Cloud FT - Code Manual Implementation

Fig. 8. RaaS Framework Proposal

The first step of the framework is to analyze the unit current state of development, and then suggest for possible manual updates or even auto-generate code when common solutions can be automatically updated based on the previously mentioned techniques.

5. CASE STUDY: SERVICE RELIABILITY COMPARISON FOR CLOUD SYSTEMS ADOPTING THE RaaS FRAMEWORK

The first application relies on two version programming on the IaaS service layer and temporal diversity technique at PaaS layer.

5.1. Windfarmdesigns

A Google cloud based application (Windfarmdesigns) for optimizing wind-farms energy production has been taken into consideration. It provides with the possibility to generate layouts of wind turbine positioning on a wind park, Fig. 9. The user can determine through the interface the input data as well as file on which the optimization will be run and eventually execute the algorithm.

The application is composed from the following three main subsystems:

1. Subsystem 1: Frontend Application Logic (Django Application running on Google App Engine, GAE);
2. Subsystem 2: Backend Application Logic (Django Application running on Google Compute Engine, GCE);
3. Subsystem 3: Engineering Algorithm Implementation (Python/MATLAB Application).

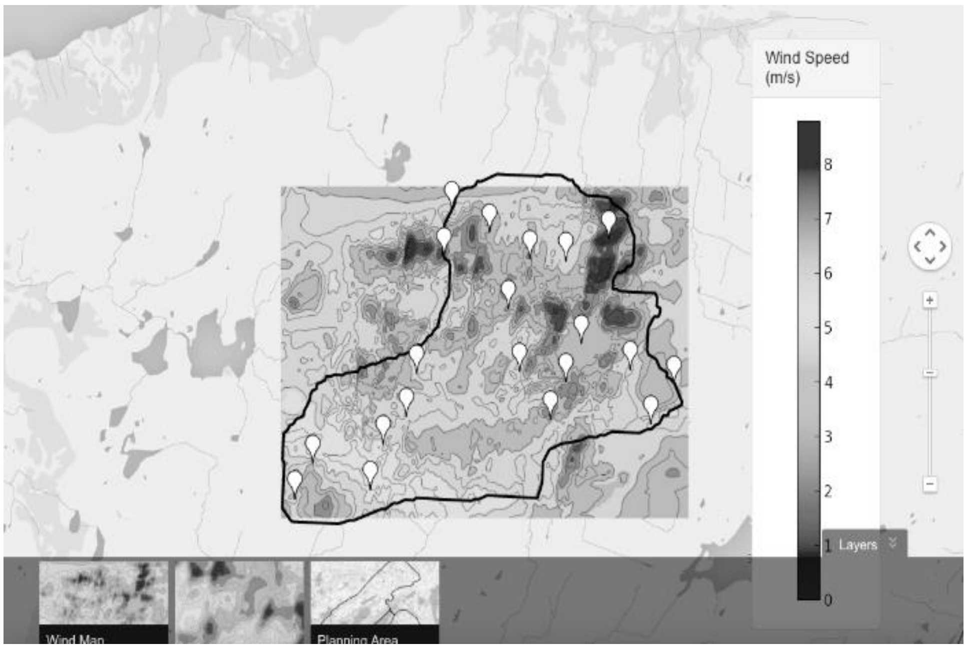

Fig. 9. Computed windfarm with a 20 turbine layout

The frontend application mainly serves the GUI to the end user for him to provide inputs/outputs to the system. Since it is deployed on the cloud infrastructure, in particular GAE, it assures scaling capabilities. The backend application running on GCE serves the following primary roles:

1. Accepts request coming from the frontend this being input parameters and files;
2. Runs the optimization algorithm implemented in Python or MATLAB to generate data;
3. Stores the results and optimization status to the Google Cloud Storage and Google Cloud MySQL respectively.

The primary operations for a registered and logged in user and their probability of occurrence are presented in Table 1.

Table 1.Operational Profile

Test Case Allocation 1000 test/year	Involved Components	Occurrence Probability	Avg.Occurrence Rate/ User/ month	Operations
330	GAE/GCS/GCSQL	33%	100	View Opt. List
170	GAE/GCE	17%	50	Create GCE VM
170	GAE/GCS/GCSQL/ GCE	17%	50	Call Optimization URL
330	GAE/GCS/GCSQL	33%	100	View Opt. Results

Figure 10 represents the Software System Architecture and its primary components through a deployment diagram. The main components where fault tolerance can be applied are on the GAE application, part of the PaaS, especially the instance creation module and the URL calls performed to communicate with the other subsystems. In this case the temporal diversity is a reasonable approach since REST requests are performed through the Google API to initialize other services (in this case GCE instances). However working on a cloud infrastructure not all services during virtual machine creations are immediately available and random issues may occur depending on the infrastructure current state.

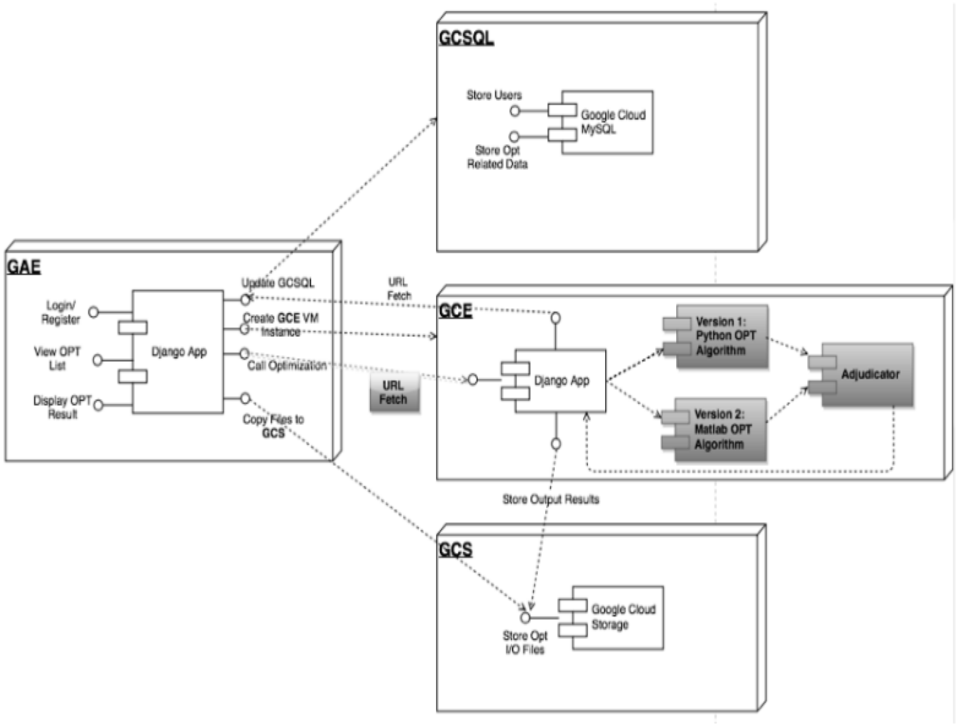

Fig. 10. Software system architecture deployment/component diagram

In our case study we have taken into account the network card initialization for the GCE virtual machine instance. During the application execution a random error occurred due to the fact that the virtual machine instance was not properly responding to a URL fetch request, also presented in Fig. 10.

The cloud infrastructure state was the producer of the transient error occurring in this particular case. The problem was almost totally mitigated with the temporal diversity technique, by repeating the URL request with the same input data several times until a correct result was obtained. The code below shows the implementation in Python of the temporal diversity:

```
i = 0
    while status != "OK" and request: #adjudicator
        fetch_url_request(http://130.211.84.135/wind
        farm/gscon/?elip_param_b=0&elip_param_a=0&ut
        m_zone=32+U+&timestamp=20150604104931…)
     time.sleep(2**i)#retry with backoff philosophy
 i = i + 1
```

while table 2 represents a summary of the test results oriented from operational profile described in Table 1.

Table 2. Test case summary table before and after adopting fault tolerance

Successful tests (Design Diversity)	Successful tests (Temporal Diversity)	Successful tests (No FT applied)	Test Case Allocation 1000 test/ year	Operations
320	320	320	330	View Optimization List
165	165	165	170	Create GCE VM
167	**167**	**92**	170	Call Optimization URL
320	317	**317**	330	View Optimization Results

As noticeable from Table 2 (through the bold values) the newly measurable reliability with the test cases ran has been 0.98 with respect to 0.54 from the previous implementation.

However the reliability of the system is also dependent on other software components, where design diversity technique might be more useful. One of them is the optimization algorithm running on GCE. Although for the particular case study this might not be the most critical operation but in general the GCE is intended to run computationally intensive code thus, high reliability of computations is quite crucial, so that timing overheads are avoided.

From the Software System Architecture Deployment/Component diagram in Fig. 10 its noted the modules for which design diversity is applicable. The algorithm has been developed in MATLAB and Python programming language from different teams which have started from equal requirements. While the running of the computations is started in parallel and the results obtained are evaluated from a voter. The latter checks upon the generated files consistency from both the implementations and provides a result to the end user when at least one of the executions has been performed successfully. From the test cases run it was noted a slight improvement in reliability from 99% to 99.9%.

Such software robustness has been justified by the fact that optimizations may last several days thus an error due to infrastructure or calculations might be quite costly. Thus an improvement even less than 1% in reliability might mean an improvement of several hours computation time.

5.2. Comparing Results

Fig.11 shows some results comparisons obtained while testing the software prove the efficiency of the applied approaches at PaaS level.

More than obviously the results show that adoption of the fault tolerant approaches decreases the lambda failure rate over tests performed approximately 2000 - 8000 hours of operation.

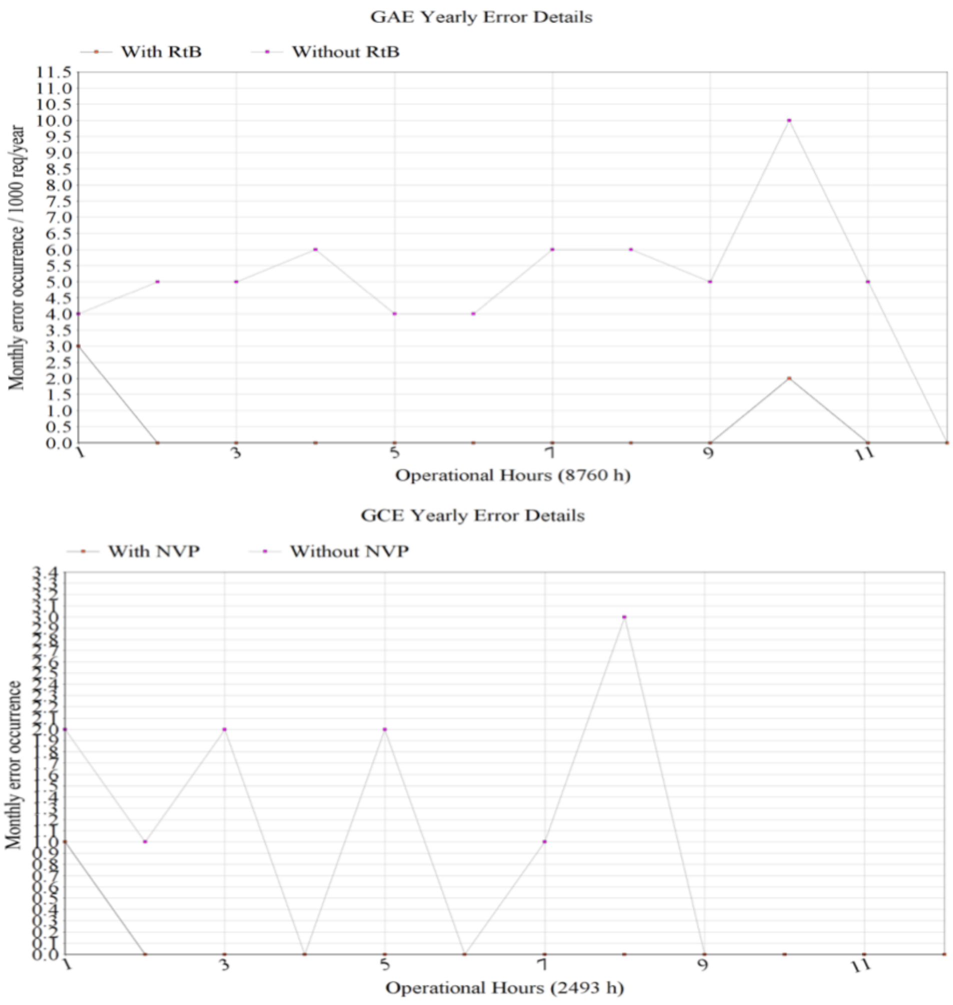

Fig. 11. Comparison of the reliability approaches results on GAE and GCE

5.3. How the techniques can be integrated into the proposed framework across different cloud service

Most cloud services today exploit several programming languages in a distributed collaborative environment of virtual machine servers. Thus this makes easier the process of implementing the different fault tolerant techniques in a unified framework at different service level as proved from the previous case study as shown in Table 3.

Table 3. Fault Tolerant Technique applied at different cloud service level

Suggested Fault Tolerant Technique	SaaS	PaaS	IaaS
Design Diversity (NVP, RcB)			x
Data Diversity (RtB)		x	
Temp. Diversity combined with Design Diversity (RcB)		x	

However, same or more techniques are able to be exploited at different service level. The framework implementation should also rely on common programming languages supported from different service providers at PaaS level. Table 4 shows that most famous providers support almost the same programming languages. Thus a generic framework development could be initiated with the prospective of expanding its programming language support.

Table 4. Programming languages supported by different cloud providers

Cloud Provider	Denomination	PaaS Supported Programming Languages
Google	Google App Engine	Go, PHP, Java, Python, Node, .NET, Ruby
Amazon	AWS Elastic Beanstalk	Java, .NET, PHP, Node.js, Python, Ruby, Go, Dockers
Microsoft Azure	Azure Cloud Services	Java, .NET, PHP, Node.js, Python, Ruby

6. CONCLUSIONS

This work has been based on the adoption of fault tolerant techniques to the Google cloud infrastructure. The case study prove part of the concept related to the establishment of a cloud framework for dependability/reliability purposes. The demand for stable and reliable software with mission critical purposes shall increase as a result of the new cloud exploitation challenges in smart cities and perhaps later on in aeronautics, nuclear, satellite exploring space systems where computational data could take a long time to derive results and it is imperative the reliability of the system to be fulfilled.

Thus in the initial section of the paper we have described the existing situation of the different service layers. Then an overview of the state of the art techniques for high availability/reliability have been described. Moreover the suggestion for other possible fault tolerant techniques have been proposed as well as the chance to integrate them within a single framework that would provide easiness of applicability to most cloud end users and service providers.

However, the primary objective has been to understand their usefulness through experimental case study, where different techniques have been applied and comparisons have been done with respect to the system versions not adopting the techniques. However the framework proposal must be applied further by actually developing the service for at least one cloud infrastructure such as Google, Azure, Amazon cloud.

In this paper work only the conceptual architecture of the framework has been proposed sustained from the proof provided by experimental case study and cloud capabilities in terms of infrastructure, programming languages supported, current architectures implemented etc.

REFERENCES

[1] Morris, M. A., *An Approach to the Design of Fault Tolerant Software*, M.Sc. thesis, Cranfield Institute of Technology, 1981.

[2] Hecht, M., and H. Hecht, *Fault Tolerant Software Modules for SIFT*, SoHaR, Inc. Report TR-81-03, April 1981.

[3] Avizienis, A., and J. P. J. Kelly, Fault Tolerance by Design Diversity: Concepts and Experiments, *IEEE Computer*, Vol. 17, No. 8, 1984, pp. 67.

[4] Amman, P. E., and J. C., Knight, Data Diversity: An Approach to Software Fault Tolerance, *IEEE Transactions on Computers*, Vol. 37, No. 4, 1988, pp. 418-425.

[5] Bondavalli, A., F. Di Giandomenico, and J. Xu, *A Cost-Effective and Flexible Scheme for Software Fault Tolerance,* Technical Report No. 372, University of Newcastle upon Tyne, 1992.

[6] Bondavalli, A., F. Di Giandomenico, and J. Xu, Cost Effective and Flexible Scheme for Software Fault Tolerance, *Journal of Computer System Science & Engineering*, Vol. 8, No. 4, 1993, pp. 234-244.

[7] Mossetti, G., C. Poloni, B. Diviacco, Optimization of wind turbine positioning in large Windfarms by means of a generic algorithm, *Journal of Wind Engineering and Industrial Aerodynamics*, Vol. 51, No. 1, January 1994, pp. 105-116.

[8] Kim K. H., The Distributed Recovery Block Scheme, in M. R. Lyu (ed.), *Software Fault Tolerance,* New York: John Wiley & Sons, 1995, pp. 189-209.

[9] Alexandrov T, Dimov. A Software availability in the cloud In: *Proceedings of the 14th International Conference on Computer Systems and Technologies*, 2013, pp.193–200. ACM. http://dl.acm.org/citation.cfm?id=2516814.

[10] Imran A, Ul Gias A, Rahman R, Seal A, Rahman T, Ishraque F, Sakib K. *16th International Conference on Computer and Information Technology (ICCIT)*, 2013, pp. 271–276. IEEE. http://ieeexplore.ieee.org/document/ 6997344/ ?arnumber=6997344.

[11] Chen T, Bahsoon R, Tawil A-RH. Scalable service-oriented replication with flexible consistency guarantee in the cloud. *Inform Sci.*, 264, 2014, pp.349–370.

[12] An K, Shekhar S, Caglar F, Gokhale A, Sastry S (2014) A cloud middleware for assuring performance and high availability of soft real-time applications. *J Syst Arch* 60(9), 2014, pp. 757–769.

[13] *Availability Management Framework*. http://devel.opensaf.org/SAI-AIS-AMF-B.04.01.AL.pdf.

Information about the authors:

Orges Çiço - Holds a "laurea" (a.k.a. MSc) in Computer Engineering from the Polytechnic University of Turin, Italy and is currently a PhD Candidate at Tirana University in the Department of Statistics and Applied Informatics. His PhD field topic is in Cloud Computing Business Solutions and Cloud Software Reliability concerning Fault Tolerant Techniques. Currently he is a Full Time Lecturer at Canadian Institute of Technology (CIT) and the Director of the Metropolitan Incubator Center (MI) (NGO part of the University Metropolitan Tirana). His research interest are related to Cloud Computing Software Systems mainly concerning their reliability and fault tolerance. Faculty of Engineering, Canadian Institute of Technology, Tirana, Albania. Phone: +355 69 6700007, orges.cico@cit.edu.al

Zamir Dika, PhD, is a professor at the South East European University. His research focuses on Electronic Commerce and Information Systems. During his experience as a lecturer in SEEU, he was engaged in courses and research in System Analysis and Design, E-commerce, Information Systems Design, Advanced Topics in Information Systems, etc. Meanwhile he was engaged in many commercial projects related to Information Systems design and development, both for companies as well as Academia. Faculty of Contemporary Sciences and Technologies, SEE University, Tetovo, Macedonia, e-mail: z.dika@seeu.edu.mk

Betim Çiço, PhD, is a professor at the Epoka University, Tirana, Albania. His research focuses on Digital Systems, Computer Architecture, Cloud Computing. During his experience as a lecturer in EPOKA, he was engaged in courses in Digital Systems, Computer Architecture, Advance topics in Software Engineering, Research Methodology. Meanwhile he was engaged in European Projects, FETCH, DAAD project related to Software Engineering and Reverse Engineering. Phone: +355 69 2094229, bcico@epoka.edu.al

Manuscript received on 15 July 2017