International Journal of Research
Available at http://internationaljournalofresearch.org/

p-ISSN: 2348-6848
e-ISSN: 2348-795X

Volume 01 Issue 08
September 2015

# Security based Optimizing Design to Migrate Cloud

Pokuru Guruteja ; Jayendra Kumar

(M.Tech) ASSISTANT PROFESSOR CVSR ENGINEERING COLLEGE

**ABSTRACT :**
*The on-demand use, high scalability, and low maintenance cost nature of cloud computing have attracted more and more enterprises to migrate their legacy applications to the cloud environment. Although the cloud platform itself promises high reliability, ensuring high quality of service is still one of the major concerns, since the enterprise applications are usually complicated and consist of a large number of distributed components. Thus, improving the reliability of an application during cloud migration is a challenging and critical research problem. A reliability-based optimization framework, named RO Cloud, to improve the application reliability by fault tolerance. RO Cloud includes two ranking algorithms. The first algorithm ranks components for the applications that all their components will be migrated to the cloud. The second algorithm ranks components for hybrid applications that only part of their components are migrated to the cloud. Both algorithms employ the application structure information as well as the historical reliability information for component ranking. Based on the ranking result, optimal fault-tolerant strategy will be selected automatically for the most significant components with respect to their predefined constraints. The experimental results show that by refactoring a small number of error-prone components and tolerating faults of the most significant components, the reliability of the application can be greatly improved.*

Index Terms—Cloud migration, component ranking, fault tolerance, software reliability

**INTRODUCTION :** Cloud computing enables convenient, on-demand network access to a shared pool of configurable computing resources. In the cloud computing environment, the computing resources (e.g., networks, servers, storage, etc.) can be provisioned to users on-demand, like the electricity grid. Start-up companies can deploy their newly developed Internet services to the cloud without the concern of upfront capital or operator expense. However, cloud computing is not only for start-ups, its cost effective, high scalability and high reliability features also attracted enterprises to migrate their legacy applications to the cloud. Before the migration, enterprises usually have the concern to keep or improve the application reliability in the cloud environment. Thus, reliability based optimization when migrating legacy applications to the cloud environment is becoming an urgently required research problem.

In traditional software reliability engineering, there are four major approaches to improve system reliability: fault prevention, fault removal, fault tolerance, and fault forecasting. When turning to the cloud environment, since the applications deployed in the cloud are usually complicated and consist of a large number of components, only employing fault prevention techniques and fault removal techniques are not sufficient. Another approach for building reliable systems is software fault tolerance, which is to employ functionally equivalent components to tolerate faults.

Software fault tolerance approach takes advantage of the redundant resources in the cloud environment, and makes the system more robust by masking faults instead of removing them. Although the cloud platform is flexible and can provide resources on-demand, there is still a charge for using the cloud components (e.g., the virtual machines of Amazon Elastic Compute Cloud or Simple Storage Service). At the same time, legacy applications usually involve a large number of components, so it will be expensive to provide redundancies for each component. To reduce the cost so as to assure highly reliability in a limited budget during the migration of legacy applications to cloud, an efficient reliability-based optimization framework is needed.

In ROCloud, each component is considered as independent and the fault-tolerant strategy selection is carried out on component basis. In the future, we will study the fault tolerance of interrelated components. In addition, ROCloud uses the ratios of component failure to application failure to measure the failure impact of components.

## Comparison Criteria

We compare the approaches with respect to the following eight criteria:
1.Modernization Strategy. The strategy of the proposed approach: one of replacement, redevelopment, wrapping and migration.
2.Legacy System Type. The kind of system to which the technique applies.
3.Degree of Complexity. Time/cost complexity of the method (or NA, if not reported).
Analysis Depth.
4.The strategy used to analyze the legacy system to understand its concepts and locate the important functions to be exposed as part of SOA architecture.
5.The analysis could be shallow or deep depending on the strategy used. Minimal dependency on the existing legacy system components in achieving SOA architecture can provide more flexibility.
6.Process Adaptability. How well the process adapts to the legacy system to minimize the extent of the required modifications.
7.Tool Support. To what degree is the process automated, and if a tool is proposed or implemented.
8.Degree of Coverage. Does the proposed approach present a complete strategy for moving to SOA, or only a specific part of the modernization.
Validation Maturity. Has the proposed approach been applied and validated.

## Replacement of Legacy Systems

Although replacement is not one of the strategies advocated by the surveyedpapers, it may make sense to retire the application and replace it with an off-the-shelf package or a complete rewrite of the legacy system from scratch. Twopossible reasons are if the business rules in the application are well understood inthe organization, and the legacy system involves obsolete or difficult to maintaintechnologies

## Wrapping Strategies

Wrapping provides a new SOA interface (e.g. WSDL) to a legacy component, making it easily accessible by other software components. It is a black-box modernization technique, since it concentrates on the interface of the legacy system,hiding the complexity of its internals.

## Redevelopment Strategies

we use the term redevelopment to refer to reengineering approaches. Reengineering is the analysis and adjustment of an application in order to represent it in a new form. Reengineering can include activities such as reverseengineering, restructuring, redesigning, and re-implementing software. The following approaches use reverse engineering and reengineering to add new SOAfunctionality to existing legacy systems.
There are three main issues in service-oriented reengineering: service identification, service packaging, and service deployment. Identification of servicesfrom a legacy system is not an easy task. Software reengineering can play animportant role in migration to the service-oriented environment. It is especiallyapplicable to legacy systems with the following characteristics:

1. The legacy system needs to be migrated to a distributed environment andcan be wrapped and exposed as a Web Service.
2. The legacy system has embedded reusable and reliable functionality withvaluable business logic.
3. Some of the components in the legacy system are more maintainable thanthe whole legacy system.
4. The embedded functionality is useful to be exposed as independent services.
5. Target components need to run on different platforms or vendor products.

## N–Version Programming

## Recovery Block Technique

This technique was evolved as a result of first long term systematic investigation of multiversion technique initiated by Brian Ran dell in early 1970s . In this technique, alternate software versions are organized in a manner similar to the dynamic redundancy (standby) technique in hardware. It’s objective is to perform runtime Software Fault Tolerance detection by an acceptance test performed on the results delivered by the first version. Recovery is considered complete when acceptance test is passed. Checkpoint memory is needed to recover the state after a version fails, to provide a valid starting operational point for the next version (Fig 1).

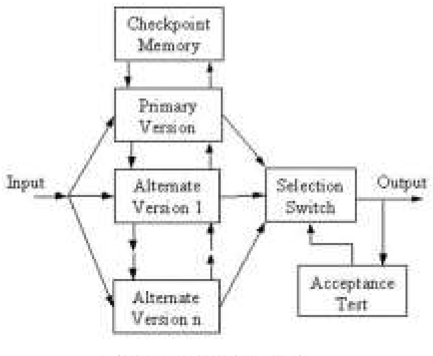


Figure 1. Recovery Block model

## N-version programming

The NVP investigation project was started by A.Avizienis in 1975. N-fold computation is carried out by using N independently designed software modules or “versions” and their results are sent to a

decision algorithm that determines a single decision result.
The fundamental difference between the RB and NVP approaches is the decision algorithm. In the RB approach, an acceptance test that is specific to the application program being implemented must be provided. In the NVP approach, a decision algorithm that delivers an agreement/disagreement decision is implemented. The N-Version programming isdefined as the independent generation of N>=2 software modules, called "versions", from the same initial requirements . "Independent generation" refer to the programming effort by individual or groups that do not interact with each other with respect to programming process. As the goal of NVP is to minimize the probability of similar errors at decision points, different algorithms, programming languages, environments and tools are used wherever possible.

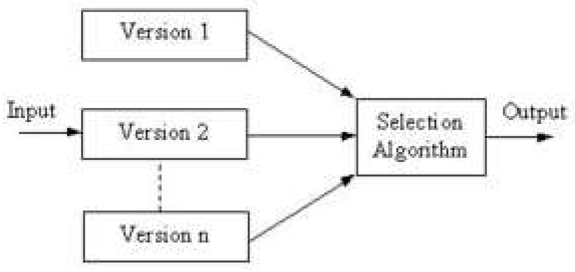
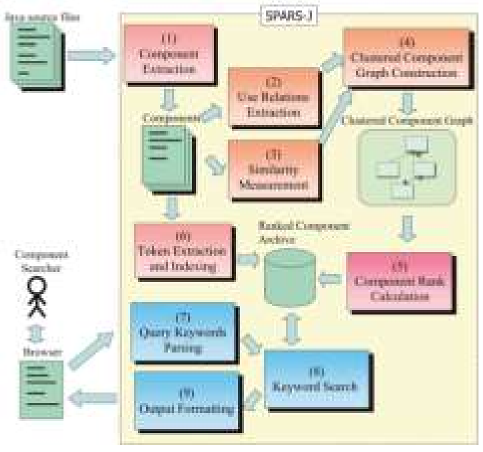


Figure 2. N-Version programming model

In NVP, since all the versions are built to satisfy the same requirements, it requires considerable development effort. But the complexity is not greater than inherent complexity of building a single version. Comparison of outputs and declaration of single result is carried out by output selection algorithm or voting algorithm (Figure 2). The output selection algorithms should be capable of detecting erroneous version outputs and prevent the propagation of bad values to main output. The output selection algorithm should be developed considering the application attributes like safety and reliability.

## Significance of Software Component

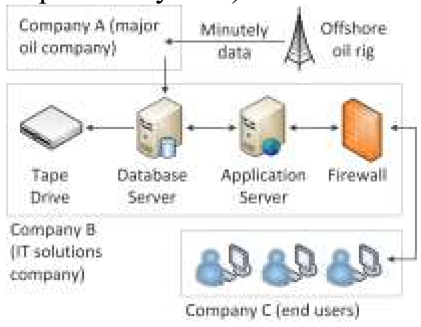
Collections of already developed programs are important resources for efficient development of reliable software systems. In this paper, we propose a novel graph-representation model of a software component library (repository), called component rank model. This is based on analyzing actual usage relations of the components and propagating the significance through the usage relations. Using the component rank model, we have developed a Java class retrieval system named SPARS-J and applied SPARS-J to various collections of Java files. The result shows that SPARS-J gives a higher rank to components that areused more frequently. As a result, software engineers looking for a component have a better chance of finding it quickly. SPARS-J has been used by two companies, and has produced promising results.


Java source files
SPARS-J
(1) Component Extraction
(2) Use Relations Extraction
(3) Similarity Measurement
(4) Clustered Component Graph Construction
Components
Clustered Component Graph
(6) Token Extraction and Indexing
Ranked Component Archive
(5) Component Rank Calculation
Component Searcher
Browser
(7) Query Keywords Parsing
(8) Keyword Search
(9) Output Formatting

## Cloud Migration

### Proposed migration project

The case study organization is a UK based SME thatprovides bespoke IT solutions for the Oil & Gas industry.It comprises of around 30 employees with offices in theUK and the Middle East. It has an organizational structurebased on functional divisions: Administration;Engineering; Support; of which Engineering is the largestdepartment. The migration use-case comprises the feasibility of themigration of one of the organization's primary serviceofferings (a quality monitoring and data acquisitionsystem) to Amazon EC2. The



### A Distributed EvaluationFrame Work

When conducting replication strategies evaluation andelection, there are several challenges to be solved:

• **Evaluation location:**
The service users are usuallyfrom different locations with different network conditions. Therefore, conducting evaluation on the targetWeb services from various locations is necessary.

• **Evaluation accuracy:**
Few service users have goodknowledge on replication strategies, test case generation, test result analysis and so on, making accuratereplication strategies evaluation difficult.

• **Evaluation efficiency:**
It is time-consuming for serviceusers to conduct evaluation themselves. More efficient
approaches are needed.Taking the viewpoint of service users where the remoteWeb service is treated as a black box without any internaldesign or implementation information, The proposed distributedevaluation framework includes a centralized server with anumber of distributed clients. The overall process can beexplained as follows.

1. **Evaluation registration:**
Users submit evaluation requestswith related information, such as the targetWeb service addresses, particular test cases, strategies selectionparameters, and so on, to the server.

2. **Client-side application loading:**
A client-side evaluationapplication is loaded to the user's computer.

3. **Test case generation:**
The *TestCase Generator* in theserver automatically creates test cases based on the interface
of the target Web Services (WSDL files). Twotypes of test cases are created: single test cases forindividual Web service evaluation, and multiple testcases for replication strategy evaluation.

4. **Test coordination:**
Test tasks are scheduled based onthe number of current users and test cases.

5. **Test cases retrieval:**
Distributed client-side applicationsget test cases from the server.

**Fault-Tolerant Cloud Applications**

**Significant Determination**

The cloud applications are typically large scale and include alot of distributed cloud components. To build a highly reliablecloud applications is a challenging and critical research
problem. To attack this challenge, a component rankingframework, named FTCloud is used for building faulttolerantcloud applications.To reduce the cost so as to develop highly reliable cloudapplications within a limited budget, a small set of criticalcomponents needs to be identified from the cloudapplications. The critical components are identified bydetermining the significant value. Kernel PrincipalComponent Ranking named KPCR Cloud approach isexpected to have better accuracy in selecting the significantvalues for identifying critical components.
1. Ranking
2. Optimal fault-tolerance selection.The procedures of FTCloud are as follows:
1. A component graph is built for the cloud application basedon the component invocation relationships.
2. Significance values of the cloud components are calculatedby employing component ranking algorithms. Based on thesignificance values, the components can be ranked.
3. Most significant components in the cloud application areidentified based on the ranking results.
4. The performance of various fault-tolerance strategycandidates is calculated and the most suitable fault-tolerancestrategy is selected for each significant component.
5. The component ranking results and the selected faulttolerancestrategies for the significant components arereturned to the system designer for building a reliable cloudapplication.

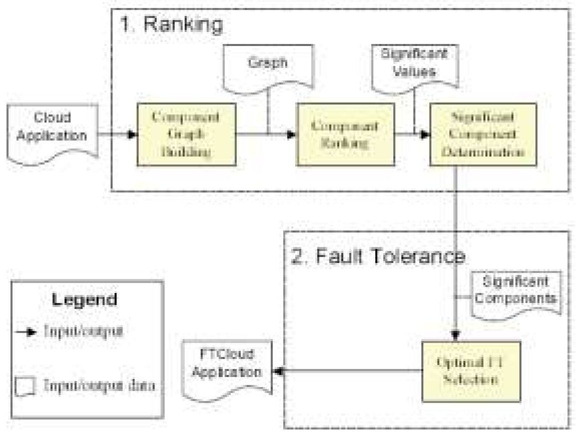


Fig. 2. System Architecture of FTCloud

## CONCLUSION

This paper presents a reliability-based design optimization framework for migrating legacy applications to the cloud environment. The framework consists of three parts: legacy application analysis, significant component ranking and automatic optimal fault-tolerant strategy selection. Two algorithms are proposed in the ranking phase: the first ranks components for the applications where all the components can be migrated to the cloud; the second ranks components for the applications where only part of the components can be migrated to the cloud. In both algorithms, the significance value of each component is calculated based on the application structure, component invocation relationships, component failure rates, and failure impacts. A higher significance value means the component imposes higher impact on the application reliability than others. After finding the most significant components, an optimal fault-tolerant strategy can be selected automaticallywith respect to the time and cost constraints. The experimental results show that ROCloud1 and ROCloud2 outperform other approaches and can greatly improve the application reliability.

In ROCloud, each component is considered as independent and the fault-tolerant strategy selection is carried out on component basis. In the future, we will study the fault tolerance of interrelated components. In addition, ROCloud uses the ratios of component failure to application failure to measure the failure impact of components. While the relationship between component failures and application failures can be complicated, more sophisticated models (e.g., Markov models, fault trees, etc.)will be investigated in the future work.

Our future work also includes:

1. Considering more factors (such as data transfer, invocation latency, etc.) when computing the
weights of invocations links.
2. Taking the constraint factors such as cost into consideration during the ranking phase, and letting the designer know intuitively which components can make the biggest improvement while cost the least.
3. More experimental analysis on the impact of incorrect prior knowledge such as invocation frequencies and component failure rates.

REFERENCES

[1] S. Al-kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, ‘‘VMFlock: Virtual Machine Co-Migration for the Cloud,’’ in Proc. 20th Int. Symp. High Perform. Distrib. Comput., New York,
NY, USA, 2011, pp. 159-170.

[2] A.A. Almonaies, J.R. Cordy, and T.R. Dean, ‘‘Legacy System Evolution Towards Service-Oriented Architecture,’’ in Proc. Int.Workshop SOAME, Madrid, Spain, Mar. 2001, pp. 53-62.

[3] G. Anthes. (). Security in the Cloud. Commun. ACM [Online]. 53(11), pp. 16-18. Available:
http://doi.acm.org/10.1145/1839676.
1839683

[4] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, ‘‘A View of Cloud Computing,’’ Commun. ACM, vol. 53, no. 4, pp. 50-58, 2010.

[5] M.Armbrust,A.Fox,R.Griffith,A.D. Joseph,R.H.Katz,A.Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, ‘‘Above the Clouds: A Berkeley View of Cloud Computing,’’ EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. EECS-2009-28, 2009.

[6] A. Avizienis, ‘‘The Methodology of N-Version Programming,’’ in Software Fault Tolerance, M.R. Lyu, Ed. Chichester, U.K.: Wiley, 1995, pp. 23-46.