**FACTOR TOLERANCE BASED APPROACH TO IMPROVE RELIABLE CLOUD MIGRATION BY USING ROCLOUD ALGORITHM**

S Poonkuzhali and S Pramila

Address for Correspondence
Professor¹, Department of Information Technology, Rajalakshmi Engineering College, Thandalam, Chennai, India.
Research Scholar², Bharath University & Professor, Dept. of Information Technology, Rajalakshmi Institute of Technology, Chennai, India

**ABSTRACT**
Cloud computing is a mainstream aspect of Information Technology. It provides skyscraping agility, scalability, elasticity and low computational cost. These characteristic of cloud has made the enterprises to migrate their legacy applications to cloud environment. Although it has several advantages, achieving high reliability during migration is becoming extremely difficult. Thus in order to migrate the applications to cloud environment, reliability is considered as the imperative requirement. To address this issue, a reliability based framework is designed. It implements the concept of ROCloud approach for migrating their entire components and sometimes partial components to cloud. It identifies the application’s critical component by using the Invocation and Coupling Information. Based upon these factors, significant value of the component is calculated by using Page Rank algorithm. The fault tolerance capacity of each of these significant component is evaluated by using various strategies such as Recovery Block, NVP, and Parallel. Through the process of refactoring a small number of error-prone components, reliability of the application is improved.

**KEYWORDS:** Cloud computing migration, cloud, Reliability.

1. **INTRODUCTION**
Cloud computing technology is an approach for delivering service to the customers. It provides on-demand service and hence called as “Pay as per use” model. The characteristics of cloud includes self-healing, multi tenancy, SLA driven, linearly scalable, virtualized and flexible. This has made the customers to think in every possible way. It enables the service based on the service level agreements for the automatic reconfiguration of the resources. The new survey “The path to value in the cloud” is found that the leading functional areas of cloud adoptions are engineering and development, management and operation. Through a detailed assessment by involving detailed review of current applications, analyzing the application landscape across multiple dimensions, evaluating cloud readiness of each application in scope, a much calibrated approach can be set out for entrepreneurship to adapt to cloud model. Operationally unstable or frequently altered application do better when kept on-premise. Strategies for improving the cloud reliability are integrating power management software, automated replication software and triggered live migration. Consumers and companies are increasingly recognizing the value in the various aspects of cloud computing. As the industry matures, it has better addressed concerns over security, stability and pricing. This will lead to greater adoption throughout the industry. Computing is quickly shifting to the cloud in every way imaginable. It will be a long transition, as cloud computing can only be truly ubiquitous once connectivity is available everywhere. These features of the cloud have attracted the enterprises to migrate their legacy applications to the cloud. But migration should not degrade the performance of the application. Thus it becomes necessary to improve the reliability of the applications involved in the migration process. Improving the reliability can be achieved through the identification of significant components that are to be migrated to cloud environment.

Yuan-Shun Dai et al described the Cloud reliability analysis and modeling as not easy tasks because of the complexity and large scale of the system. The fault-tolerant matrix operation was published in 1984 by Huang and Abraham, in which an Algorithm-Based Fault Tolerance (ABFT) method was proposed. Mei et al. (2008) investigated the problems of dynamic computing service registration, large data storage and access, adaptability and quality discovery in cloud computing. A survey of trust and privacy in cloud computing was presented by Cachin et al. (2009). The survey focused on data storage in cloud grids and how to provide confidentiality, integrity, availability and service guarantee. Chakrabarti et al. discussed security issues in grid computing. Since clouds may be formed by multiple grids from different entities, it then becomes critical to address the security issues such as confidentiality, integrity and service availability. The privacy-friendly client cloud computing was also presented. Wang et al. (2006) proposed to use erasure correction coding in file preparation towards cloud computing storage. Cloud computing involves running applications which is available on different virtual servers which are distributed geographically. These virtual servers are made in such a way that different service level agreements and reliability issues are met. There exists multiple instances of the virtual server. Each application available in the cloud is replicated. It access different parts of the hardware infrastructure. In case of failure of one component, it is directly replaced by another replica. Thus it protects the information available and provides a rapid recovery from the failure. The services offered can be categorized into three main groups. They are Software as a service, Platform as a service and Infrastructure as a service. Software as a Service is the technology available to provide applications to the customers on their demand. It has a single instance of service on the cloud and provides service to multiple end users. The customer need not invest any upfront like servers; license at the same time the provider’s side cost is lowered. The best examples to this kind of service are provided by Microsoft, Zoho, Google, Salesforce etc. The second type of service in cloud computing is
Platform as a Service. In this type the development environment or the layer of software is encapsulated and offered as a service. The higher level of layer can be successfully built in the form of the lower level layer. The PaaS providers offer a predefined combination of OS and application servers includes Linux, PHP, MySQL and apache. The best example of this service is the Google’s App Engine. The customer has the freedom to build his own applications, which run on the provider’s infrastructure. Infrastructure as a Service provides the basic storage and computing capabilities as standardized services over the network. Servers, storage systems, networking equipments, data Centre space etc. are pooled and made available to handle workloads. The customer would typically deploy his own software on the infrastructure. Some common examples are Amazon, 3 Tera, GoGrid, etc. The applications can be deployed in three ways. They are public, private and Hybrid cloud. The cloud needed for deployment can be chosen by the enterprise and the cloud integrators advice has to be taken for an effective integration of the cloud in the real time environment.

Private clouds are built exclusively for the sole enterprise or an organization. The main reason for opting private cloud is security. It offers a greater control and addresses many issues which is not handled in public cloud. There are two major variations available in the private cloud. They are on-premise cloud and externally hosted private cloud. In theon-premise private cloud the internal clouds are hosted within one’s own data center. This model provides a more standardized process and protection. The limitations of this approach includes size and scalability. This is best suited for applications which require complete control and configurability of the infrastructure and security. Externally hosted Private Cloud: This type of private cloud is hosted externally with a cloud provider, where the provider facilitates an exclusive cloud environment with full guarantee of privacy. This is best suited for enterprises which don’t prefer a public cloud due to sharing of physical resources. Hybrid Clouds combine both public and private cloud models. With a Hybrid Cloud, service providers can utilize third party Cloud Providers in a full or partial manner. It has more flexibility of computing. The Hybrid cloud environment is capable of providing on-demand, externally provisioned scale.

Reliability is probably the most important of the characteristics inherent in the concept of software Quality. It has become the major concern as more users come to depend on the services offered by cloud. Reliability becomes increasingly important, especially for long-running or mission critical applications. It is intimately connected with defects, and as Jones points out, defects represent the largest cost element in programming. Software reliability concerns itself with how well the software functions to meet the requirements of the customer. Reliability is a customer oriented measure rather than a developer oriented. It relates to the operation rather than a design of the program. It is dynamic than static. It is more important for examining the significance of trends, for setting objectives and for predicting when those objectives will be met. When an enterprise needs its Legacy applications to migrate to the cloud environment, it requires a deep understanding about the application to be migrated to cloud.

2. RELATED WORK

Software reliability engineering is centered around a very important software attribute Reliability. Software Reliability defined in the Handbook of software Reliability Engineering (1996) as the failure free software operation for a specified period of time in a specified environment. It is one of the attributes of the software quality. Software reliability is a key part in software quality. The study of software reliability can be categorized into three parts: modeling, measurement and improvement. Jiantao Pan (1999) stated Software reliability modeling has matured to the point that meaningful results can be obtained by applying suitable models to the problem. There are many models exists, but no single model can capture the necessary amount of the software characteristics. Assumptions and abstractions must be made to simplify the problem. There is no single model that is universal to all the situations. Software reliability measurement is naive. Development process, faults and failures found in all factors related to software reliability. The Software reliability improvement is hard. The difficulty of the problem stems from insufficient understanding of software reliability and in general, the characteristics of software. Stieber.H.A (1997) proposed an approach which allows the detection of unreliable software components and the comparison of reliability of different software versions-even if testing is done in a classical manner. Logarithmic Poisson Execution Time Model (LPETM) given by Sathya Prasad et al (2012) is a software reliability model which predicts the expected failures and related reliability quantities. Irene Eusgeld et al suggested the models for software reliability assessment; the approaches presented includes black-box reliability models, reliability models that are based on other software metrics and white-box reliability models that build on knowledge about the internal structure of the system. AasiaQuyorum et al (2010) proposed an approach for Improving Software Reliability by using Software Engineering. A sensitivity analysis based reliability prediction is developed by Swapna et al. (2002). It describes about the performance and reliability predictions to the changes in parameters of the individual modules. Upasna Jaglan (2012) has discussed about the history of software reliability engineering, the current trends and existing problems, and specific difficulties. These are the various topics under which software reliability characteristics are defined.

By leveraging cloud services, organizations can deploy their software systems over a pool of resources. However, organizations heavily depend on their business critical systems, which has been developed over a long periods. These legacy applications are deployed the on premises. Thus organization is migrating to the cloud to have an efficient and optimized service. WeiweiQiu et al.
(2014) proposed a reliability based design optimization for cloud migration. It includes two ranking algorithms; One for migrating all the applications to cloud and the other is for migrating hybrid application. The most significant component is identified and ranked. Doaa M. Shawky et al. (2013) proposed a cost effective approach for hybrid migration to cloud. In this approach, coupling among different components of the system is measured. Then, a proposed cost measuring function is used to choose the optimal migration scenarios.

Kasto Inoue et al. proposed a component rank and Relative significance rank for the software component search. It analyzes the actual use relations among the components and propagating the significance through the use of relations. It has considered only the source code as the component while leaving other components for ranking.

Pooya Jamshidi et al. conducted a systematic literature review to identify, taxonomically classified and systematically compare the existing research on cloud migration.

Software fault can be classified into two categories. Faults can be classified according to their phase of creation or occurrence, system boundaries (internal or external), domain (hardware or software), phenomenological cause, intent, and persistence. Gray (1989) classifies software faults into Bohrbugs and Heisenbugs. The Software fault tolerance techniques are designed to allow a system to tolerate software faults that remain in the system after its development. Software fault tolerance techniques are employed during the procurement, or development of the software. When a fault occurs, these techniques provide mechanisms to the software system to prevent system failure from occurring. Software fault tolerance techniques provide protection against errors in translating the requirements and algorithms into a programming language, but do not provide explicit protection against errors in specifying the requirements.

Software fault tolerance techniques have been used in the aerospace, nuclear power, healthcare, telecommunications and ground transportation industries, and others. The basic RcB scheme is one of the two original diverse software fault tolerance techniques. It was introduced in 1974 by Horning, et al. with early implementations developed by Randell in 1975 and Hecht in 1981. The RcB is categorized as a dynamic technique. Its selection of a variant result to the output is made during the program execution based on the results of the acceptance test (AT). The hardware fault-tolerant architecture related to the RcB scheme is stand-by sparing or passive dynamic redundancy. RcB uses an AT and backward recovery to accomplish fault tolerance. We know that most program functions can be performed in more than one way by using different algorithms and designs. These differently implemented function variants have varying degrees of efficiency in terms of memory management and utilization, execution time, reliability, and other criteria. RcB incorporates these variants such that the most efficient module is located first in the series, and is termed the primary alternate or primary try block. The less efficient variant(s) are placed serially after the primary try block and are referred to as (secondary) alternates or alternate try blocks. Thus, the resulting rank of the variants reflects the graceful degradation in the performance of the variants. The NVP is one of the original design diverse software fault tolerance techniques. NVP was suggested by Elmdendorf (1972) and developed by Avizienis and Chen in 1977–1978. Compared with ReB, NVP is a static technique. That means a task is executed by several processes or programs and a result is accepted only if it is adjudicated as an acceptable result, usually via a majority vote. The hardware fault tolerance architecture related to the NVP is N-modal. The processes can run concurrently on different computers or sequentially on a single computer. The NVP technique uses a decision mechanism (DM) and forward recovery to accomplish fault tolerance. The technique uses at least two independently designed, functionally equivalent versions (variants) of a program developed from the same specification. The variants are run in parallel and a DM examines the results and selects the “best” result, if one exists.

3. METHODS AND MATERIALS

3.1 Framework for Reliable Migration Cloud Environment

The main idea is to design a framework for a reliable migration of application to cloud environment. The reliability is mainly affected due to the presence of faults in the design. By identifying the failure rate of each component involved in the migration process by using various validation parameters, reliability can be improved. The attributes taken to study about the component are its invocation information, coupling ratio and the failure rate. The component graph is generated based on this information. Significance of each of the component is identified by using page rank algorithm and hence the components are ranked accordingly. The fault tolerance strategy is automatically applied in case of any failure in order to provide an optimal design.

3.1.1 Structured Information Extraction

The structured information includes the application log, design document and legacy code as inputs. The components are identified from the obtained information. The invocation information such as invocation link, invocation frequencies and coupling information can be identified from application trace logs. The identified component’s failure rate is calculated with the help of the input validation, process tail data, data formatting, and Query Service and PDF report service. Each component failure rate is calculated with the help of these attributes.

It has various modules performing its task in a sequential manner. The input given to the system contains the document, legacy code and its application log. The input given to the structured information extraction is analyzed to obtain the component information. The component information extracted may provide the details about the invocation and coupling information. Thus the extraction information on invocation, coupling and failure rate calculation brings out the idea about the components in the application.
3.1.2 Component Graph Generation

The component graph is generated to identify the relationship between the components. It generates the failure rate and impact of each component. When the invocation and coupling is more for a component, it becomes the most important component than other components. The failure of the imperative component will affect the entire components and can cause a major damage to the application. The initial value of each component is taken as 1, as the component will depend on at least one of the other components in the application. The component graph is a directed weighed graph \(G\), which combines together the information of the application structure, invocation and coupling relationship among the components. The weight of each component is calculated by using the formula,

\[
W_{ij} = \frac{q_{ij}}{\sum_{j=1}^{n} q_{ij}}
\]  

(1)

The range of weight varies from \((0 \text{ to } 1)\) \(C_i\) to \(C_j\). This refers to the component ‘j’ is invoked by component ‘i’. When there is no invocation between them, it is assigned as zero. If the weight \(W_{ij}\) is larger, it means the component is more frequently invoked. The total transition probability matrix is

\[
\sum_{j=1}^{n} W_{ij} = 1
\]  

(2)

Adjacency matrix \(AM\) \((n \times n)\) to represent the direct dependencies of component based system. In the matrix, each component is represented by a column and a row. If a component \(c_i\) is dependent on another component \(c_j\), then \(AM[ i; j] = 1\). More formally, the values of all elements in \(AM\) \((n \times n)\) - \(d_{ij}\) \((n \times n)\) are denoted as follows:

\[
d_{ij} = 1 \quad \text{if} \quad (c_i \rightarrow c_j)
\]

\[
d_{ij} = 0 \quad \text{otherwise}
\]  

(3)

If position \(AM [ i; j]\) in the matrix is 1 then there is an edge from \(i\) to \(j\). After the calculation each position denotes that there is a path from a vertex to another vertex.

Let \(CBS\) be consisting of \(n\) components: \(C1; C2; \ldots; Cn\). dependency matrix as follows. Without loss of generality, we only include different types of dynamic dependencies between components:

\[
DM = \begin{bmatrix}
d_{11} & \cdots & d_{1n} \\
\vdots & \ddots & \vdots \\
d_{n1} & \cdots & d_{nn}
\end{bmatrix}
\]  

(4)

The dependency impact in the application can be classified as Data dependency, Control dependency, Interface dependency, Time dependency, Input / Output dependency, context dependency, State dependency, and Cause and effect dependency. In this work, the time and state dependency is considered for failure rate analysis.

Time dependency is analyzed with the deadline as parameter:

\[
T_i < \text{Deadline} \rightarrow \text{Success} 
\]  

(5)

\[
T_i > \text{Deadline} \rightarrow \text{Failure} 
\]  

(6)

State dependency is analyzed with the signed approach. The state dependency signed approach is described by using the following table.

<p>| Table 1. State Table using Signed Approach |
|----|----|----|
| INPUT | PROCESS | OUTPUT |
| +   | +    | +    |</p>
<table>
<thead>
<tr>
<th>+</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
</table>

In Table 1, ‘+’ indicates the successful execution and ‘-’ unsuccessful execution of the component. The failure of any one of the process results in the failure of the component irrespective of the legitimate input. For example in the above state table have three parameters such as input, process and output. The
successful execution of both process results in positive outcome.

**Failure rate analysis**

Theorem: Countable union of countable sets is countable.

As per the above set theorem, the overall failure rate is computed based on the summation of failure rate of individual component. The failure rate \( f(c_i) \) is given by

\[
f(c_i) = \frac{\mu(c_i)}{\sum_{j=1}^{n} \mu_j}
\]

(7)

Where \( \mu(c_i) \) the total times that component \( C_i \) failed and the sum represents the total times that \( c_i \) has been invoked. This helps in deciding the critical and non-critical components. Each component failure rate is compared with the threshold value. If the value is greater than the threshold, it is verified whether it is an important/unimportant component. In case of unimportance, the component is eliminated or refactored.

3.1.3 Identifying the significant component

The component significance is evaluated once its failure rate, invocation and coupling information is identified. The significance of component is based on the fact that, more the invocation and coupling among the components more is its significance. Thus the components structured information provides the significance of the components. Coupling information provides how well each component depends on the other component.

**Algorithm 1: Identification of significant component to migrate**

**Input:** Set of components \( c_i \) where \( 1 \leq i \leq n \); threshold = \( k \)

**Process:**
1. for \( c_i \) generate Invocation Matrix
2. for \( c_i \) generate Dependency Matrix
3. Calculate the Failure Rate \( f(c_i) \) \( \forall c_i \)
   \[ f(c_i) = \frac{\mu(c_i)}{\sum_{j=1}^{n} \mu_j} \]
4. Compare \( f(c_i) \) with \( k \)
5. if \( (f(c_i) > k) \) then Eliminate \( c_i \)
   else Refactor
6. Compute the significance value \( V(c_i) \)
   \[ V(c_i) = \frac{1 - d}{n} * f(c_i) * p(c_i) + d \sum_{k \in N(c_i)} V(c_k) W_{ki} \]
7. Identify the most significant component
8. Apply the fault tolerance technique for the most significant component.

**Output:** Failure rate of \( c_i \); Invocation Frequency of \( c_i \); Coupling Ratio of \( c_i \)

\[ V(c_i) = \frac{1-d}{n} * f(c_i) * p(c_i) + d \sum_{k \in N(c_i)} V(c_k) W_{ki} \]

(8)

\[ p(c_i) = \frac{\mu(a) \alpha(c_i)}{\mu(c_i)} \]

(9)

Where ‘n’ is the number of components in the application, and \( N(c_i) \) is the set of components which invoke component \( C_i \). The damping factor ‘d’ is used to adjust the significant value. If the coupling is more between the components it is considered as a significant component. The dependency is calculated by the parameters like input, process, and output. By using the concept of Domination Factor it is calculated (Input, Parameters, Failure %). If the failure % is more, the domination factor indicates how far the application gets affected.

3.1.4 Fault Tolerance Strategies

3.1.4.1 Recovery Block

The recovery block approach attempts to prevent

residual software faults from impacting on the system environment, and it is aimed at providing fault tolerant functional components which may be nested within a sequential program. The figure explains the operations performed in recovery block strategy. The primary alternate is executed and then the acceptance test is evaluated to provide adjudication on the outcome of this primary alternate. If the acceptance test is passed, then the outcome is regarded as successful and the recovery block can be exited, by discarding the information on the state of the system taken on entry.

**Pseudo code for Recovery Block**

**Input:** Component recognized as failure.

**Method:**

- for each failed component
  - Establish Check point
  - do Acceptance testing of the primary alternatives.
    - If successful then
      - Replace component with primary component.
    - else
      - Repeat from the do process for the next alternative.

**Output:** Fault Tolerant Component.

However if the system test fails or if any error is detected by other means during the execution of the alternate, then an exception is raised and the backward error recovery is invoked. This restores the state of the system to what it was on entry. After such recovery, the next alternate is executed and then the acceptance test is applied again. This sequence continues until either an acceptance test is passed or an alternate has failed the acceptance test. If all the alternates either fail the test or result in an exception, a failure exception will be signal to the environment of the recovery block. Since recovery blocks can be nested, and then the raising of such an exception from an inner recovery block would invoke recovery in the enclosing block.

3.1.4.2 NVP N-Version Programming

The major objectives of the NVP process are to improve the independence of version development and to employ design diversity in order to minimize the probability that two or more member versions will produce similar erroneous results that coincide in time for a decision action.

3.1.4.3 Parallel Strategy

The parallel strategy follows the same technique as in the N version programming; but the result is taken from the module whose result is first arrived. It invokes all the n functional equivalent components in parallel and the first returned response will be employed as the final result. A parallel module fails only if all the redundant components fail. The failure probability is calculated by the product of all the n components.

**Pseudo code for NVP**

**Input:** Component recognized as failed.

**Method:**

- Distribute the input to N-versions in order to maximize the isolation and minimize the related Software failure.
  1. Receive response from functionally equivalent versions.
  2. Version with majority voting is replaced for the faulty component.

**Output:** Fault Tolerant Component.

4. RESULTS AND DISCUSSION

The previous section described the various methods and approaches for improving the reliability in the cloud migration process. The component graph represents application’s invocations and coupling
information among its components. The application depends on number of process to execute a particular task. The task may be dependent or independent based on the need of processing. The failure rate is calculated for each component in order to calculate the overall reliability of the application. Table 1 represents the state dependency between the components. It describes that the failure of the earlier process, results in the failure of further processing of the application. Figure 2 depicts the invocation and coupling between the components. The applications are represented in red color circle while the components present in the application are represented in green color. The number of times the components are invoked is represented over the link. For example in the image conversion process, there are input file process, pixel extractor, RGB Converter and output file process.

The component graph illustrates the processing of 12 input files for various performing tasks like word, image and pdf conversions with the availability of three resources. The recovery block strategy is applied in such a way that the failure of service by one resource is replaced or handled by another resource. The invocation information and failure rate are provided over the link in the component graph. For example, the word to pdf conversion has three sub process like input file process, pdf process and output file process. The failure rate of this process is calculated by using eq. (1). Each sub process has been invoked multiple times in order to complete the task. But it may fail in case of unavailability of resources.

Fig. 2. Component Graph

Fig. 3. Failure Rate of the Components in an Application

In the Fig. 3, failure rate of the component is represented. It has five fields namely component name, application name, total count, invocation count and failure rate. The name of component and application is given in first two fields followed by the total number of invocation to a particular component in an application is provided in total count column. Here, the input file submitted to the word to pdf converter is 6 with the failure rate 0 as it has responded for all the six files. But in the later part of processing the failure rate has increased to 1.0 due to the unavailability of service by the resources.

The failure rate calculation is followed by calculating significant value of the components. It is calculated by using eq. (8). The components in the application are ranked based on the values obtained from failure rate, impact rate and significance value of the other components. The significant values arrived in fig. 4 revealed that the image conversion process pixel extractor plays a vital role as it has highest the significant value and ranking. In case of failure of the component pixel extractor, there is high possibility for the application getting failure. It is observed that the components having high significant value has more impact on the application. It would make the failure more frequently. Hence providing a fault tolerance to these components helps in improving the reliability of the application.

Fig. 4. Significant Value of the Components

In the fig. 4, Failure rate of components is depicted in a graphical format. The x- axis represents various components in an application, y- axis represents the invocation count of the components. The bars represents the count and failure rate of the components. The failure rate decides how frequently the components get failed in an application.

Fig. 5. Graphical Representation of the Component’s Failure Rate.
5. CONCLUSION

The proposal is to improve the reliability of the application during migration to the cloud environment. It involves extracting the applications structured information, for identifying the components coupling and invocation frequency. The dependency among the components is more when the number of frequency and the ratio of coupling are high. The component graph generated would help to identify the relationship between the components present in the application. By using the above the collected information the failure rate of each component is calculated. The component with higher failure rate will have greater impact on the application when it degrades or fails. Thus by eliminating or refactoring these bottleneck components, the software reliability of the application can be improved to a greater extent.

REFERENCES
23. S.Pearson, Taking account of privacy when designing cloud computing service, in the proc ICSE workshop software engineering, Challenges in cloud, may 2009.
35. ZaipengXie, survey of software fault tolerance techniques, University of Wisconsin-Madison.