Design Optimized Framework for Migrating a Legacy System to Cloud Environment

Cloud computing enables convenient, on-demand network access to a shared pool of configurable computing resources in the cloud. Although the cloud platform offers promises high reliability, there are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to improve system reliability: fault forecasting, fault removal, fault tolerance, and fault-tolerance engineering. There are four major approaches in traditional software reliability engineering to im
II. OPTIMIZATION FRAMEWORK FOR CLOUD MIGRATION

A. Optimization Challenges in Cloud Migration

First, we use a motivating example to show the challenging problems of this paper. Enterprise A wants to reduce upfront capital investment and system infrastructure maintenance effort. The cloud computing technology satisfies these requirements. Enterprise A decides to migrate its legacy applications to an IaaS cloud, as shown in Fig. 1. The legacy application consists of a number of distributed components. Ensuring reliability of the application is one of the major concerns for making the migration.

To enhance the system reliability, the designer wants to optimize the original design of legacy application by providing fault tolerance mechanisms for its components with replication techniques. When designing fault tolerance mechanisms for the components, the designer needs to consider the following problems:

a) Some components of the legacy application may be implemented by outdated technology and suffer from high failure rates. These components can have great impact on system reliability. Replication techniques are not enough to improve the reliability. For example, providing one replication for a component with failure rate 50 percent can only reduce the failure rate to 25 percent which is still unacceptable. A better approach is refactoring, that is to adopt new technology to rewrite the component and add fault prevention logic (e.g., exception handling), which can dramatically reduce the component’s failure rate. Trade-offs need to be made when considering which components should be re-factored due to cost constraints.

b) The legacy application may consist of a large number of components. It is too expensive to deploy alternative replicas for all the components, since there are costs for using cloud resources (e.g., the virtual machines). To make trade-offs between costs and reliability, the designer chooses to tolerate faults of the most important components, whose failures have great impact on the whole system. However, it is not easy to identify which components have greater impact on system reliability, because:

- The reliability properties of each component may be very different. Some components may already have fault prevention logic (e.g., error checking, exception handling, etc.) and thus are more reliable than others.

- Failures of different components can have different impacts on the system. Components fulfilling critical tasks (e.g., payment) are taken as critical components, while other components accomplishing non-critical tasks (e.g., providing decorative pictures on web pages) are taken as non-critical ones [48]. Failures of critical components have greater impact on the system than failures of non-critical components.
These two characteristics should be considered in combination. A failure-prone non-critical component may have little impact on overall system reliability, while a component for critical task may be carefully designed and already have low enough failure rate. The straightforward approach to only consider components with high failure rates or fulfilling critical tasks as important components may not lead to an optimal solution.

c) Some applications are restricted by enterprise security polices and only part of their components can be migrated to the cloud. For these hybrid applications, the components which are kept in the private data center are potentially important components and they can only use resources in the private data center for fault tolerance.

d) There are a number of fault tolerance strategies. The cloud platform itself may also provide recovery approaches such as virtual machine restart. Different strategies have different overheads and costs. It is a challenging task for the designer to find out the optimal fault tolerance strategies for the significant cloud components.

To address the above problems, we first analyze the legacy application to collect the reliability properties and application structure information. Then, we propose two significant component ranking algorithms in Section 3.2. At last an optimal fault tolerance strategy selection algorithm is presented in Section 3.3.2, which suggests optimal fault tolerance strategies for components with different constraints.

B. Optimization Framework

Fig. 2 shows the overview of our reliability optimization framework (named ROCloud), which includes three phases: 1) legacy application analysis; 2) automated significance ranking; and 3) fault tolerance strategy selection. The processes of each phase are as follows:

a) Both structure and failure information are extracted during the legacy application analysis phase. The structure information extraction consists of two sub-processes: component extraction and invocation extraction. The failure information including failure rate and failure impact are collected from the execution logs and test results of the legacy application. Components with a failure rate higher than the threshold will be re-factored, and their reliability properties will be updated. A component graph is built for the legacy application based on the structure as well as the failure information.

b) In the automated significance ranking phase, two algorithms are proposed for ordinary applications that can be migrated to public cloud and hybrid applications that need to be migrated to hybrid cloud, respectively.

III. APPROACH

Optimization Framework approach has three modules:

a) Legacy application analysis,

b) Automated significance ranking,

c) Fault tolerance strategy selection.

The processes of each phase are as follows:

a) Both structure and failure information is extracted during the legacy application analysis phase. The structure information extraction consists of two sub processes: component extraction and invocation extraction. The failure information including failure rate and failure impact are collected from the execution logs and test results of the legacy application. Components with failure rates higher than the threshold will be re-factored and reliability properties are updated. A component graph is built for the legacy application based on the structure as well as the failure information.

b) In the automated significance ranking phase, two algorithms are proposed for ordinary applications that can be migrated to public cloud and hybrid applications that need to be migrated to hybrid cloud, respectively.

c) The performance, overhead, and cost of various fault tolerance strategy candidates are analyzed and the most suitable fault tolerance strategy is selected for each significant component based on its predefined constraint.

A. Legacy Application Analysis

The structure information includes components and the invocation information. The components are extracted from legacy applications by source code and documentation analysis. The invocation information such as invocation links and invocation frequencies can be identified from application trace logs. Source codes and documentations are useful supplementary materials in addition to trace logs. All the information are represented in a component graph. In this paper, the main optimization goal is reliability, so a more straightforward way is employed to determine which components should be re-implemented: components with failure rates greater than a threshold. The selection of the threshold is dependent on project budget and the target application failure rate.

Fig. 2. Overview of the optimization framework
After refactoring, the component failure rates will be estimated based on test results, and the component reliability property dataset will be updated.

B. Automated Significance Ranking

Based on the component graph, two component ranking algorithms are proposed in this section. The first algorithm ranks components for ordinary applications where all their components can be migrated to the cloud. The second algorithm rank components for hybrid applications which can be partly moved to the cloud. In a distributed application, the failures of the components which are frequently invoked by many other components tend to have greater impact on the system compared with the components which are rarely invoked by others.

Thus these components are considered to be more important from the reliability aspect and should be ranked at the front of component list. Inspired by the Page Rank algorithm, we propose an algorithm to calculate the significance value of each component of the migratory application employing the component invocation relationships and reliability properties.

Based on the component graph and component reliability information, the component ranking algorithm includes the following steps:

1. Initialize by randomly assigning a numerical value between 0 and 1 to each component in the component graph.

2. Compute the significance value for a component $c_i$ by:

$$V(c_i) = \frac{1 - d}{n} \sum_{k \in V(c_i)} V(c_k) w_{ki}$$

With the above approach, the significance values of the components can be calculated by considering the application structure information, the invocation relationships, and the knowledge of component reliability properties in combination. A component with a larger significant value is considered to be more significant. The failures of these significant components will have great impact on other components and thus tend to cause application failures.

C. Fault Tolerance Strategy Selection

Software fault tolerance is widely adopted for critical systems (e.g., airplane flight control systems, nuclear power station management systems, etc.). At the same time, a cloud platform also provides approaches such as virtual machine restart, virtual machine migration, etc. to improve components reliability. By employing these techniques to provide functionally equivalent components, the component failures can be tolerated and thus the overall system reliability can be increased. Three well known software fault tolerance strategies as well as the approaches taking advantage of cloud platform features are introduced in the following with formulas for calculating the failure rate, response time and resource cost.

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 automatically with respect to the time and cost constraints.

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.

More experimental analysis on ROCloud and the impact of in-correct prior knowledge such as invocation frequenties and component failure rates.

REFERENCES


27 www.ijerm.com