Scheduling Fault Tolerant Cloud Applications using Component Ranking

Aswathi Vandana P
PG Scholar
Department of CSE (PG)
Sri Ramakrishna Engineering College, Tamilnadu, India

Bhaggiaraj S
Assistant Professor (Sr. Gr.)
Department of IT
Sri Ramakrishna Engineering College Tamilnadu, India

ABSTRACT

Cloud is an emerging technology where the providers provide various services to Information Technology by adopting the concept of service oriented architecture, distributed, autonomic, and utility computing. In the present competitive world, building a highly dependable cloud application and opting for the optimal fault tolerant technique for cloud components has become crucial. In this paper, a component ranking framework is needed for identifying critical components along with the ranking prediction framework for selecting optimal cloud services. Additionally, Kernel Principal Component Ranking approach is proposed to have better accuracy in selecting the significant values for identifying critical components. Subsequent to the component ranking, an optimal fault-tolerance strategy is also proposed to automatically determine the strategy apt for identified critical cloud components. Thus metaheuristic algorithms are used for optimal fault tolerant strategy selection. The simulation results show that by tolerating faults of a minor fraction of the most critical components, the reliability of cloud applications can be greatly improved.

Keywords

Cloud computing, Ranking prediction, Fault tolerance

1. INTRODUCTION

Cloud computing [NIST] is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. All the resources in the cloud are provisioned as services. Cloud computing is becoming extremely popular for its flexibility, scalability and reduced cost. Hence, many leading organizations are deploying their applications in the cloud environment. Cloud applications include lots of components [1] in which most are very complex. Thus building highly reliable and high quality cloud applications has become challenging.

The software systems in the cloud (named as cloud applications) typically involve multiple cloud components communicating with each other. The cloud applications are usually large scale and very complex. But unfortunately, the reliability of the cloud applications is still far from perfect in reality.

Nowadays, the demand for highly reliable cloud applications is becoming unprecedentedly strong. Building highly reliable clouds becomes a critical, challenging, and urgently required research problem. Due to the cost of developing and maintaining redundant components in traditional software reliability engineering, software fault tolerance is usually only employed for critical systems. Different from traditional software systems, there are a lot of redundant resources in the cloud environment, making software fault tolerance a possible approach for building highly reliable cloud applications.

Cloud applications usually involve a large number of components and it is too expensive to provide alternative components for all the cloud components. To reduce the cost so as to develop highly reliable cloud applications within a limited budget, a small set of critical components needs to be identified from the cloud applications. In order to build highly reliable cloud applications, a component ranking framework, named KPCRCloud [2], is proposed.

The two key steps involved here are:

- In KPCRCloud, first a component ranking framework is proposed, to rank the component automatically.
- Metaheuristic algorithms are used to suggest the optimal fault-tolerance strategies for the significant components automatically.

2. RELATED WORK

The increasing popularity of Cloud computing as an attractive alternative to classic information processing systems has increased the importance of reliable and fault tolerant processing.

Andrzej Goscinski, Michael Brock, 2010 [3] proposed the application of the Resources Via Web Services framework (RVWS) to offer higher level abstraction of clouds in the form of a new technology that makes possible the provision of service publication, discovery and selection based on dynamic attributes.

Ghalem Belalem, Said Limam, 2011 [4] proposes a fault tolerant architecture to Cloud Computing that uses an adaptive Checkpoint mechanism to assure that a task running can correctly finish in spite of faults in the nodes in which it is running. The proposed fault tolerant architecture is simultaneously transparent and scalable.

Jose Luis Lucas-Simarro et al. , 2012 [5], presents a modular broker architecture that can work with different scheduling strategies for optimal deployment of virtual services across multiple clouds, based on different optimization criteria, user constraints, and environmental conditions.

Linlin Wu, Saurabh Kumar Garg, Rajkumar Buyya, 2011 [6] proposes innovative admission control and scheduling algorithms for SaaS providers to effectively utilize public cloud resources to maximize profit by minimizing cost and improving customer satisfaction level.

Michael Armbrust et al, 2010 [7] provides simple figures to quantify comparisons between of cloud and conventional computing, and identifying the top technical and non-technical obstacles and opportunities of cloud computing.

Pawel Czarnul, 2012[8] proposes a technique for filtering measured data, in particular to avoid vendor lock-in issues. Also provides a design and results from an engine for simulation of various ranking algorithms in response to streams of prices from various providers.

Sheheryar Malik, Fabrice Huet, 2011 [9] proposed a model in which the system tolerates the faults and makes the decision on the basis of reliability of nodes based on the execution of design diverse variants on multiple virtual machines along with the recovery mechanisms.

Swapna S. Gokhale, Kishor S. Trivedi, 2002 [10] enables the identification of performance and reliability bottlenecks and thus helps to analyze the sensitivity of the performance and reliability predictions to the changes in the parameters. This hierarchical model could be used to assess the impact of workload changes on the performance and reliability of the application.

3. SIGNIFICANT VALUE DETERMINATION

The cloud applications are typically large scale and include a lot of distributed cloud components. To build a highly reliable cloud applications is a challenging and critical research problem. To attack this challenge, a component ranking framework, named FaTCloud is used for building fault-tolerant cloud applications.

To reduce the cost so as to develop highly reliable cloud applications within a limited budget, a small set of critical components needs to be identified from the cloud applications. The critical components are are identified by determining the significant value. Kernel Principal Component Ranking named KPCRCloud approach is expected to have better accuracy in selecting the significant values for identifying critical components. By tolerating faults of a small part of the most important cloud components, the cloud application reliability can be greatly improved. Based on this idea, FaTCloud is proposed to identify the most significant components and suggests the optimal fault-tolerance strategies for these significant components automatically. FaTCloud can be employed by designers of cloud applications to design more reliable and robust cloud applications efficiently and effectively.

Initially two ranking algorithms are proposed to identify significant components from the huge amount of cloud components. Then, an optimal fault-tolerance strategy selection algorithm to determine the most suitable fault-tolerance strategy for each significant component is presented. The following Fig.1 shows the system architecture of the fault-tolerance framework [14] (named FaTCloud), and it includes two divisions:

1. **Ranking**
2. **Optimal fault-tolerance selection.**

The procedures of FaTCloud are as follows:

1. A component graph is built for the cloud application based on the component invocation relationships.

2. Significance values of the cloud components are calculated by employing component ranking algorithms. Based on the significance values, the components can be ranked.

3. Most significant components in the cloud application are identified based on the ranking results.

4. The performance of various fault-tolerance strategy candidates is calculated and the most suitable fault-tolerance strategy is selected for each significant component.

5. The component ranking results and the selected fault-tolerance strategies for the significant components are returned to the system designer for building a reliable cloud application.

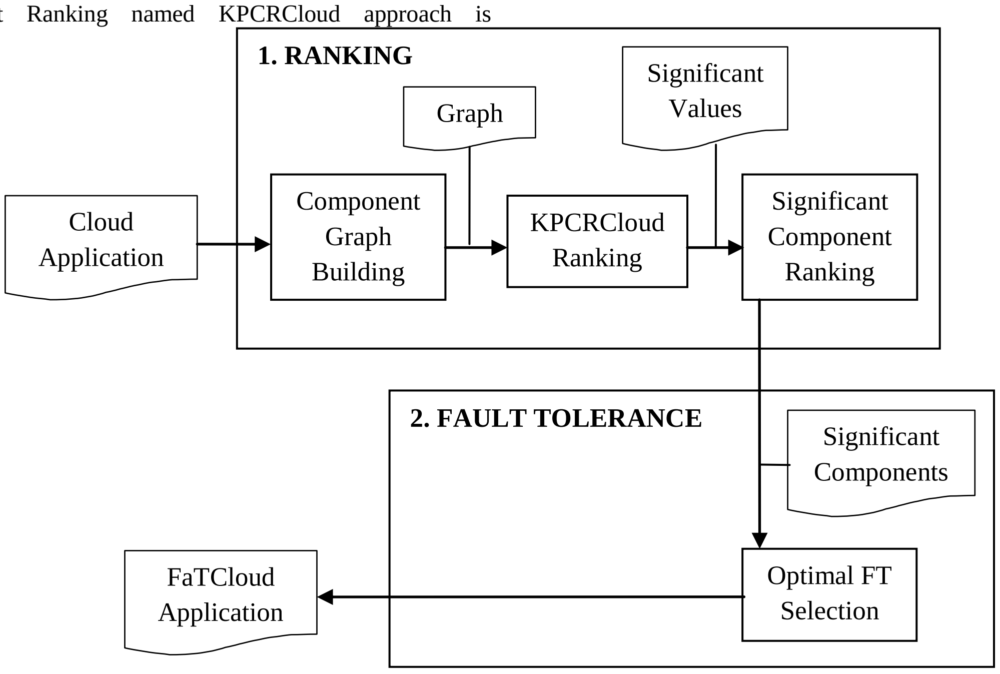

Fig 1. System Architecture

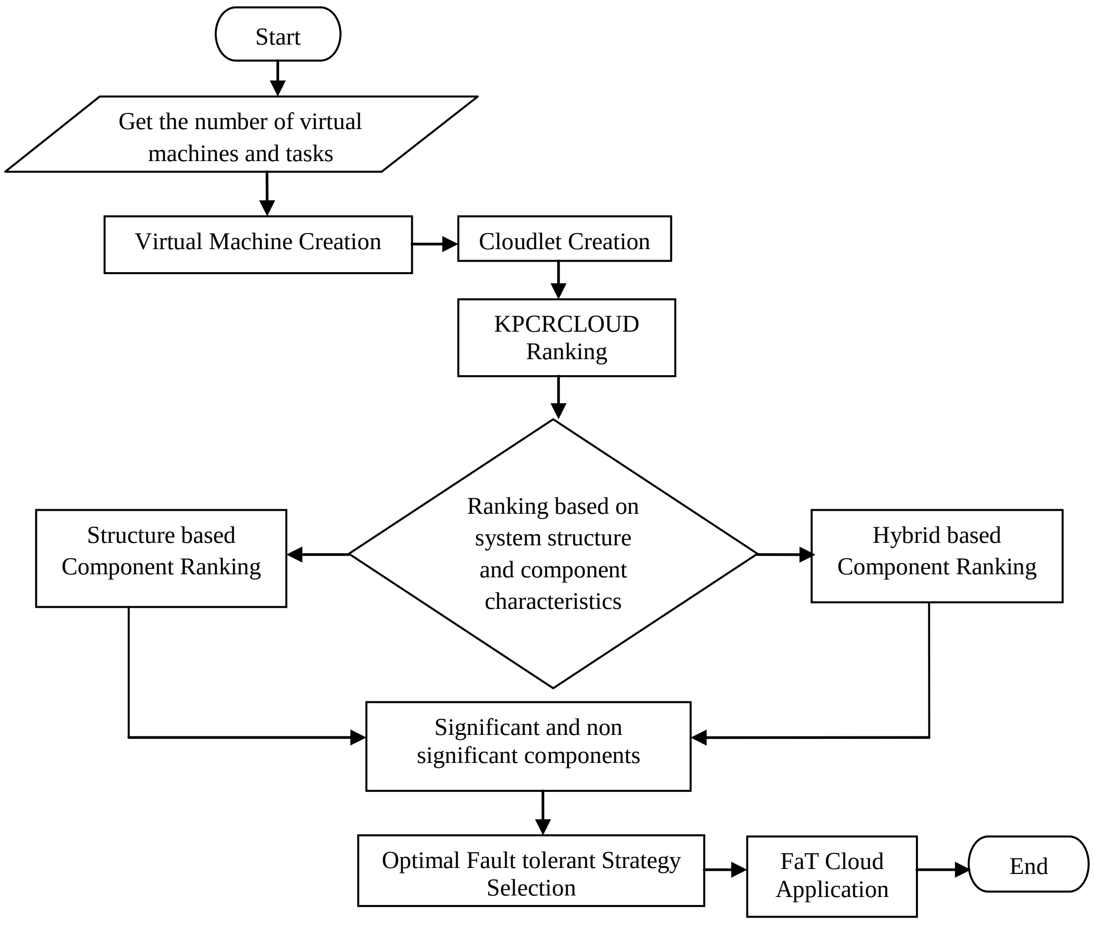
Start
Get the number of virtual machines and tasks
Virtual Machine Creation
Cloudlet Creation
KPCRCLOUD Ranking
Ranking based on system structure and component characteristics
Structure based Component Ranking
Hybrid based Component Ranking
Significant and non significant components
Optimal Fault tolerant Strategy Selection
FaT Cloud Application
End

Fig 2. Flow Chart

4. COMPONENT RANKING IN CLOUD APPLICATION

In Cloud environment, an application proceeds with the cloudlet creation. A simulation toolkit is used that enables modelling and simulation of Cloud computing systems and application provisioning environments.

The objective of significant component ranking algorithms is to measure the importance of cloud components based on the available information. Significant component ranking includes three steps:

4.1 Component Graph Building

A cloud application can be modeled in the form of an undirected weighted graph G, where a node c_i in the graph represents a component and a directed edge eij from node c_i to node c_j represents a component invocation relationship, i.e., c_i invokes c_j. Each node c_i in the graph G has a nonnegative significance value $V(c_i)$, which is in the range of (0,1). Each edge eij in the graph has a nonnegative weight value $W(e_{ij})$, which is in the range of [0,1]. The weight value of an edge e_{ij} can be calculated by

$$W(e_{ij}) = \frac{frq_{ij}}{\sum_{j=1}^{n} frq_{ij}} \qquad (1)$$

Where, frq_{ij} is the invocation frequency of component c_j by component c_i, n is the number of components and $frq_{ij} = 0$, if component c_i does not invoke c_j.

In this way, the edge e_{ij} has a larger weight value if component c_j is invoked more frequently by component c_i compared with other components invoked by c_i.

4.2 Significant Component Ranking

Based upon the component graph, the two component ranking algorithms are proposed namely, using KPCRCloud and FaTCloud Algorithms. FaTCloud algorithms are further used as FaTCloud1 and FaTCloud2. The first approach employs the system structure information (i.e., the component invocation relationships and frequencies) for making component ranking. The second approach not only considers the system structure, but also considers the component characteristics (i.e., critical components or noncritical components) for making component ranking. Further, KPCRCloud is used for learning preference relations and to perform significant value calculation with accuracy.

4.2.1 Structure-Based Component Ranking

In a cloud application, some components are frequently invoked by a lot of other components. These components are considered to be more important, since their failures will have greater impact on the system compared with other components. Intuitively, the significant components in a cloud application are the ones which have many invocation links coming in from the other important components. An algorithm is proposed to calculate the significance values of the cloud components employing the component invocation relationships and frequencies.

The procedure of this structure-based component ranking algorithm is shown in the following steps:

1. Randomly assign initial numerical scores between 0 and 1 to the components in the graph.
2. Compute the significance value for a component c_i by:

$$V(c_i) = \frac{1-d}{n} + d \sum_{k \epsilon N(c_i)} V(c_k) W(e_{ki}) \qquad (2)$$

Where, n is the number of components $N(c_i)$ is a set of components that invoke component c_i.
The parameter d ($0 \le d \le 1$) is employed to adjust the significance values derived from other components, so that the significance value of c_i is composed of the basic value of itself (i.e., $0 \le d \le n$) and the derived values from the components that invoked c_i.
3. Repeat the computation until all significance values become stable.

4.2.2 Hybrid Based Component Ranking

In order to rank the components as accurate as possible, a hybrid component ranking approach is proposed, which considers both the system structure as well as the component characteristics as follow:

1. Randomly assign initial numerical scores between 0 and 1 to the components in the graph. Divide the components in the graph into two component sets, critical components C and noncritical components NC, employing the prior knowledge provided by the system designers.
2. If a component c_i is a critical component (ci ϵ C), compute the significance value for component c_i by

$$V(c_i) = (1-d)\frac{\beta}{|C|} + d \sum_{k \epsilon N(c_i)} V(c_k) W(e_{ki}) \quad (3)$$

and if a component c_i is a noncritical component (c_i ϵ NC), compute the significance value for component c_i by

$$V(c_i) = (1-d)\frac{1-\beta}{|NC|} + d \sum_{k \epsilon N(c_i)} V(c_k) W(e_{ki}) \quad (4)$$

Where, the parameter β, ranges from ($\frac{|C|}{n} \le \beta \le 1$), is employed to determine how much the hybrid approach relies on the critical components and the noncritical components.

4.2.3 Kernel Principal Component Ranking Algorithm

A ranking function is a function *f: Q→ R* mapping the instance-label pair *q* to a real value representing the relevance of the label *y* with respect to the instance *x*. The aim of our ranking task is to find a label ranking function *f: Q→ R* such that the ranking induced by the function for any instance *x* ∈ *X* is a good prediction for the true preference relation.

Let us define $\mathbb{R}^Q$ = *{f: Q→* $\mathbb{R}$*}* and let H ⊆ $\mathbb{R}^Q$ be the hypothesis space of possible ranking functions. To measure how well a hypothesis *f* ⊆ $\mathcal{H}$ is able to predict the preference relations P_x for all instances *x* ∈ *X*, we consider the following cost function that captures the amount of incorrectly predicted pairs of the relevant training data points:

$$d(f,\mathcal{T}) = \frac{1}{2}\sum_{i,j=1}^{n} W_{ij} \mid sign(s_i - s_j) - sign(f(q_i) - f(q_j)) \mid \qquad (5)$$

where sign(•) denotes the signum function

$$sign(r) = \begin{cases} 1, if\ r > 0 \\ -1, if\ r \le 0 \end{cases}$$

The use of cost functions like Equation leads to intractable optimization problems, therefore, we consider the following least squares approximation, which regresses the differences $(s_i - s_j)$ with $f(q_i) - f(q_j)$ of the relevant training datapoints q_i and q_j

$$c(f,\mathcal{T}) = \frac{1}{2}\sum_{i,j=1}^{n} W_{ij}\,((s_i - s_j) - (f(q_i) - f(q_j)))^2 \ (6)$$

In the above described setting we assume that every instance-label pair has an associated score.

5. OPTIMAL FAULT-TOLERANCE STRATEGY SELECTION

5.1 Fault -Tolerant Strategies

Software Fault Tolerance hence System reliability can be improved by employing functionally equivalent components to tolerate component failures. There are many strategies in which three are exclusively included here. They are:

- *Recovery Blocks (RB)*

  RB [3] is a means of structuring redundant program modules, where standby components will be invoked sequentially. Failure probability f of a recovery block can be calculated by:

$$f = \prod_{i=1}^{n} f_i \qquad (7)$$

- *N- Version Programming (NVP)*

  NVP [11] is multi-version programming where versions are independently generated. Failure probability f of a recovery block can be calculated by:

$$f = \sum_{i=\frac{n+1}{2}}^{n} F(i) \qquad (8)$$

- *Parallel Strategy*

  Invokes all the n functional equivalent components in parallel and the first returned response will be employed as the final result. Failure probability f of a recovery block can be calculated by:

$$f = \prod_{i=1}^{n} f_i$$

- *ACO*

  Ant colony optimization algorithm (ACO) [5] is a probabilistic technique and is a member of the ant colony algorithms family, in swarm intelligence methods, and it constitutes some metaheuristic optimizations. It can be reduced to finding good paths through graphs. In general, the ant moves from state to state with probability

$$p_{xy}^{k} = \frac{(T_{xy}^{\alpha})(\eta_{xy}^{\beta})}{\sum_{y \epsilon allowed_y}(T_{xy}^{\alpha})(\eta_{xy}^{\beta})} \qquad (9)$$

  Where (T_{xy}^{α}) is the amount of pheromone deposited for transition, $0 \le \alpha$ is a parameter to control the influence of T_{xy}, η_{xy},is the desirability of state transition *xy* (*a priori* knowledge, typically $1/d_{xy}$, where *d* is the distance). T_{xy}, η_{xy} and represent the attractiveness and trail level for the other possible state transitions.

- *BC Strategy*

  The Artificial Bee Colony (ABC) [13] is an optimization algorithm based on the intelligent foraging behaviour of honey bee swarm. The main steps of the algorithm are given below:

1. Initial food sources are produced for all employed bees.

2. REPEAT
   - Each employed bee goes to a food source and then evaluates its nectar amount and dances in the hive
   - Each onlooker watches the dance of employed bees and chooses one of their sources depending on the dances, and then goes to that source.

- Abandoned food sources are determined and are replaced with the new food sources discovered by scouts.
- The best food source found so far is registered.

3. UNTIL (requirements are met)

5.2 Strategy Selection

Designer must specify the constraints for each component to apply the fault tolerance strategy. Here two constraints are considered. They are Response time and cost. The optimal fault-tolerance strategy selection problem for a cloud component with user constraints formulated mathematically as:

Minimize: $f = \sum_{i=1}^{m} f_i \times x_i$ (10)

Subject to: $\sum_{i=1}^{m} s_i \times x_i \leq u_1,$
$\sum_{i=1}^{m} t_i \times x_i \leq u_2,$
$\sum_{i=1}^{m} x_i = 1,$
$x_i \in \{0,1\}$

Where,
f_i is the failure probability of the strategy candidates
s_i is the cost of the strategy candidates and
t_i is the response time of the strategy candidates
m is the number of fault tolerance strategy candidates
u_1 is the user constraints for cost
u_2 is the user constraints for response time
x_i is set to 1 if the ith candidate is selected for the component and 0 otherwise.
Need to calculate the cost, response time, and the aggregated failure probability values of different fault-tolerance strategy candidates. Then, the following Algorithm is designed to select the optimal candidate.

Algorithm: Optimal FT Strategy Selection

```
Input: $s_i$, $t_i$, and $f_i$ values of candidates; user constraints $u_1$ and $u_2$
Output: Optimal candidate index $\rho$; $m$: number of candidates
for ($i = 1$; $i <= m$; $i$++) do
        if ($s_i \leq u_1$ && $t_i \leq u_2$) then $v_i = f_i$;
        end
end
if no candidate meet user constraints then
        Throw exception;
end
Select $v_x$ which has minimal value from all the $v_i$;
$\rho = x$;
```

6. RESULTS AND ANALYSIS

A simulation tool named CloudSim has been used for simulating this project. The CloudSim toolkit supports both system and behavior modeling of Cloud system components such as data centers, virtual machines (VMs) and resource provisioning policies.

In the FaTCloud1 and FaTCloud2 approaches, the parameter d balances the significance value derived from the other components and the basic value of the component itself. In this experiment, the component ranks are fairly stable when the parameter d is changed from 0.75 to 0.95.

6.1 Performance Metrics And Test Methods

Probability Failure parametric measure is taken for consideration in order to evaluate the performance of the approach. Random walk is employed to simulate the invocation behavior in cloud applications. To start a random walk, a node in the invocation graph is randomly selected as the start node. A very small stop rate is used for the random walk to guarantee the invocation coverage of all nodes in the graph. In this experiment, 10,000 invocation sequences are generated for each setting of 100 nodes.

Table 1. Performance Comparison based on Failure Probability between No FT and FaTCloud1

Number of tasks	System Failure	
	NoFT	FaTCloud1
0	0.045	0.020
1	0.045	0.030
2	0.048	0.035
3	0.055	0.048
4	0.055	0.055
5	0.070	0.055

FaTCloud1 and FaTCloud2 fault-tolerance mechanisms are applied on these invocation sequences and the average results are evaluated based on the probability failure. The FaTCloud1 and FaTCloud2 approaches are compared with NoFT scenario and the results are evaluated.

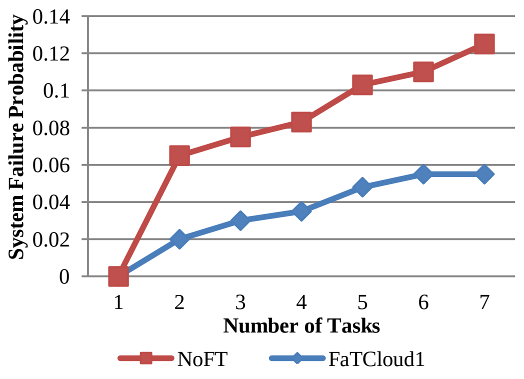

Fig 3: Performance Comparison based on Failure Probability between NoFT and FaTCloud1

Fig. 3 and Table 1 show the simulation results of number of tasks and system failure probabilities for NoFT and FaTCloud1. The number of tasks is varied from 0 to 14 and the respective system failure probabilities are obtained. The comparison is evaluated between FaTCloud1 and NoFT scenarios.

It is inferred from the graph that with the increase in the number of tasks, the system failure probabilities also increases linearly. Fig. 4 and Table 2 show the simulation results of number of tasks and system failure probabilities for NoFT and FaTCloud1.

The maximum system failure probability of the FaTCloud1 approach is 0.077% where as the NoFT scenario attains the maximum failure probabilities of 0.127 %. Thus, FaTCloud1 approach performs well under all tasks.

It is clearly observed from the graph and table that the when the number of tasks is lesser, both FaTCloud2 and NoFT approaches attain nearly the similar system failure probabilities.

Table 2. Performance Comparison based on Failure Probability between FTCloud2 and No FT

Number of Tasks	System Failure	
	NoFT	FTCloud2
0	0.020	0.015
1	0.021	0.025
2	0.024	0.030
3	0.038	0.041
4	0.039	0.052
5	0.050	0.052

For instance, when the number of tasks is 6, the system failure probability attained by both the approaches is nearly 0.052%.

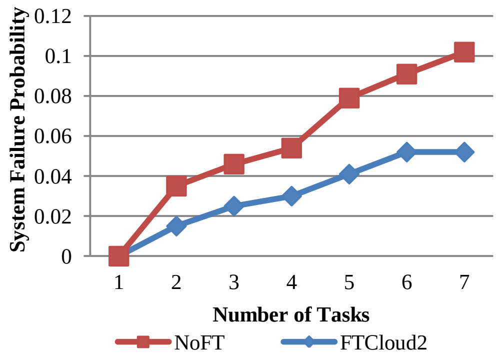

Fig 4: Performance Comparison based on Failure Probability between FaTCloud2 and No FT

But, when the number of tasks is increased, the proposed FaTCloud2 approach outperforms the NoFT scenario. The maximum system failure probability attained by FaTCloud2 is 0.1% where as that obtained by NoFT scenario is 0.127%. Likewise for the comparison between NoFT and FaTCloud2.

7. CONCLUSION

Concluding, FaTCloud, the component ranking framework for fault-tolerant cloud applications is simulated. In the proposed component ranking algorithms, the significance value of a component is determined by incorporating KPCRCloud Ranking algorithm. After identifying the significant components, an optimal fault-tolerance strategy selection algorithm is proposed to provide optimal fault-tolerance strategies such as ABC and AOC to the significant components automatically, based on the user constraints. The experimental results show that FaTCloud1 and FaTCloud2 approaches outperform other approaches and the Kernel Principal Component Ranking approach is proposed to have better accuracy in selecting the significant values for identifying critical components.

REFERENCES

[1] Avizienis, 1995, “The Methodology of N-Version Programming,” Software Fault Tolerance, M.R. Lyu, ed., pp. 23-46, Wiley.

[2] Colorni, M. Dorigo et V. Maniezzo, 1991, Distributed Optimization by Ant Colonies, Proceedings Of Ecal91 - European Conference On Artificial Life, Paris, France, Elsevier Publishing, 134-142.

[3] Andrzej Goscinski, Michael Brock, 2010, “Toward dynamic and attribute based publication, discovery and selection for cloud computing”, *Future Generation Computer Systems*, pp. 947-970.

[4] D. Karaboga, 2005 An Idea Based On Honey Bee Swarm for Numerical Optimization, Technical Report-TR06,Erciyes University, Engineering Faculty, Computer Engineering Department.

[5] Jose Luis Lucas-Simarro, Rafael Moreno-Vozmediano, Ruben S. Montero, Ignacio M. Llorente, 2012, “Scheduling strategies for optimal service deployment acrossmultiple clouds”, *Future Generation Computer Systems*, pp.1431–1441.

[6] Linlin Wu, Saurabh Kumar Garg, Rajkumar Buyya, 2011, “SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments ”, *Journal of Computer and System Sciences*, pp. 1280–1299.

[7] Michael Armbrust et al., 2010, “A View of Cloud Computing,” Comm. ACM, vol. 53, no. 4, pp. 50-58.

[8] Pawel Czarnul, 2012,“An Evaluation Engine for Dynamic Ranking of Cloud Providers”, *Informatica 37*, pp. 123–130.

[9] Sheheryar Malik, Fabrice Huet, 2011, “Adaptive Fault Tolerance in Real Time Cloud Computing”, *in 2011 IEEE World Congress on Services*, pp. 280-287.

[10] Swapna.S. Gokhale and K.S. Trivedi, 2002, “Reliability Prediction and Sensitivity Analysis Based on Software Architecture,” *Proc. Int’l Symp. Software Reliability Eng. (ISSRE ’02)*, pp. 64-78.

[11] M. Dorigo, 1992, Optimization, Learning and Natural Algorithms (*in Italian*), Ph.D. thesis, DEI, Politecnico di Milano, Italy, pp.140

[12] Randell B. and Xu J., 1995 “The Evolution of the Recovery Block Concept,” Software Fault Tolerance, M.R. Lyu, ed., pp. 1-21, Wiley.

[13] Tom Heskes et.al, 2009, Kernel Principal Component Ranking:Robust Ranking on Noisy Data, Institute for Computing and Information Sciences, Radboud University Nijmegen.

[14] Zibin Zheng et al. 2012, “Component Ranking for Fault-Tolerant Cloud Applications”, IEEE Transactions On Services Computing, Vol. 5, No. 4, pp. 540-550.