A Distributed Fault Tolerant Algorithm for Load Balancing in Cloud Computing Environments

Abderraziq Semmoud [1,*], *Mourad Hakem* [2], and *Badr Benmammar* [1]

[1]Faculty of science, Department of Computer Science, Abou Bakr Belkaid University, Tlemcen, Algeria
[2]DISC Laboratory, Femto-ST, UMR CNRS, Université de Franche-Comté, France

Abstract. Cloud computing is a promising paradigm that provides users with higher computing benefits in terms of cost, availability and flexibility. Nevertheless, with potentially thousands of connected machines, faults become more frequent and may have an adverse effect on the application. Consequently, fault-tolerant load balancing becomes necessary in order to optimize resource utilization while ensuring the reliability of the system. Different approaches have been proposed in the literature for fault tolerance in cloud computing. However, they suffer from several shortcomings: some fault tolerance techniques use task replication which reduce the cloud's efficiency in terms of resource utilization. While other models rely on checkpoint recovery to tolerate failures, resulting in an increase in the mean response time. To address these shortcomings, an efficient and adaptive fault tolerant algorithm for load balancing is proposed. Based on the CloudSim simulator, some series of test-bed scenarios are considered to assess the behavior of the proposed algorithm.

1 Introduction

The cloud is emerging as a wide-scale distributed computing infrastructure that enables coordinated problem solving and resource sharing in today's world that needs information anytime and anywhere. It provides highly secure, scalable, and efficient mechanisms for discovering and negotiating access to computing resources in a transparent manner. Computing resources can thus be shared on a large scale among an infinite number of geographically distributed servers. Even though load balancing is essential to ensure high availability of applications in an increasingly critical environment, failures become inevitable as the number of components in the cloud system increases.

Two kinds of approaches can be adopted to provide fault tolerance: *Proactive* and *Reactive* fault tolerance policies [1]. The principle of the former [2, 3] is to predict failures and take preventive actions to avoid them. The reactive ones allow to recover from failed state when a failure occurs. Reactive fault tolerance techniques are classified into two classes: the first class [4, 5] allows the application to continue execution even if the nodes fail. In the second class [6], the effect of failures is repaired either by re-executing the failed parts of the application or by continuing execution under the assumption that the application will later return to a normal state.

In this work, we design a hybrid failure management mechanism for load balancing in cloud computing environments. This mechanism is based on preemptive migration and replication to proactively take preventive actions before a failure occurs and reactively manage the occurrence of failures. With this, we can effectively reduce resource usage, costs, and network traffic in data centers.

In the following, we summarize the contributions and the novelties of the presented study:
- The proposal of a new distributed algorithm for fault-tolerant load balancing in order to solve the scalability problem.
- The combination of proactive and reactive fault tolerance techniques in an adaptive way.
- Implementing fault tolerance while maintaining the system's load balancing.

The rest of this paper is organized as follows: Section 2 describes some related work on fault tolerance. In Section 3, we present the problem formulation and the proposed fault tolerant algorithm. Section 4 presents the simulation setup and experimental results. In Section 5, we conclude this paper with a summary of the contributions and future works.

2 Related works

In this section, the relevant techniques proposed in the literature to deal with the fault tolerance management are reviewed.

In [7], Chang and al. have presented a fault-tolerant load balancing technique for current web services. In this work, each block within a file is replicated at least three times depending on the access frequency. To reduce the likelihood of data loss, each of the replicas is located on a different server. When the control center detects that

* Abderraziq Semmoud: abderrazak.semmoud@univ-tlemcen.dz

the number of replicas of a block is less than three due to a network problem or a hardware failure, the control server starts immediately a new replication of the block. Fang and al. [8] combine the load balancing and the fault tolerance mechanisms in the distributed stream processing engine (DSPE) to reduce the overall resource use while preserving the system's interactivity, high-throughput, scalability and high availability. The proposed method can handle node failures and a dynamic scenario of data asymmetry based on a data level replication strategy. As the distribution of incoming flows fluctuates, the workload is rebalanced by selectively deactivating data in overloaded nodes and activating their replicas on underloaded ones to minimize migration costs. When a failure occurs, the system activates replicas of the affected data to provide rapid recovery after a failed processing task while keeping the workload balanced.

The work proposed in [9] introduces the adoption of a fault tolerance mechanism that masks the cloud server implementation with cloud selection to avoid network congestion and health monitoring. Since it can preserve the system data availability, the cloud selection is induced to prevent the network traffic. With the proposed framework, a proactive fault tolerance technique is provided and the experimental study revealed reduced overhead and energy consumption. Nirmala et al. [10] propose a new fault tolerant workflow scheduling algorithm for large scale scientific workflow applications that learns replication heuristics in an unsupervised manner. Authors propose a light weight synchronized checkpointing method to identify failed task and estimate replication count using heuristics. Different machine learning algorithms were implemented to predict failures for different type of workflow applications. In the work of Chatterjee et al. [11], a two-level fault tolerant load balancing algorithm called GBFTLB has been proposed. It considers that processors are heterogeneous and uses processor capacity to allocate the load to it. A new communication model is designed for global communication between processors in order to manage the loss of connectivity and minimize communication costs. GBFTLB can operate in asynchronous and synchronous networks and can be applied to systems with arbitrary topologies. The GBFTLB aims to optimize the number of messages and the number of turns required simultaneously.

To minimize the communication cost induced by fault tolerance and deal with dynamic distributed platforms, the authors, in [12], propose a new collaborative checkpoint-rollback recovery scheme which takes partial snapshot of the system by using appropriate coordinations among the network's nodes. They demonstrate throw simulations, that the proposed approach is able to decrease the number of exchanged messages of the coordination process.

As reported above, unlike earlier works, this study investigate the combination of proactive and reactive fault tolerance techniques to deal with unexpected failures during the load balancing process. The objective is to take advantage of each technique in adaptive way to improve the overall performances in terms of system reliability and QoS seen by end users. To address this, we make use of Preemptive Migration and Replication in order to proactively take preventive actions before a failure occurs and then reactively avoid the impact of VM's failure. Moreover, the replication process is performed in a balanced way as we can use replicas to smooth fairly the load in the system.

3 The proposed RPMFT algorithm

Let $G(t)=(VM(t), E(t))$ be a dynamic graph representing the links between VMs. The graph is dynamic in the sense that at every unit of time t, some VMs can enter and leave the system. $VM(t)$ is the set of virtual machines in the system at time t. $E(t)$ is the set of bidirectional links between VMs such as VM_i and VM_j can send messages to each other at time t if and only if $(VM_i, VM_j) \in E(t)$. The system is considered failure-prone where nodes and/or links may fail with a mean time to failure of *mttf*. $N_i(t)$ is the set of VM_i's direct neighbors at time t.

Let $REP_{ik}(t)$ be the set of VMs receiving a the replicas of task T_k from VM_i at time t. The reliability of VM_i at time t is denoted by $REL_i(t)$. The main objective of the proposed fault-tolerant algorithm is to identify the right replication nodes and reduce the number of replications while maximaizing both system's reliablity and tasks' completion rate.

The presented study is essentially an extended version of the STLB algorithm [13], which was designed to tackle the problem of load balancing in cloud computing environments. It differs from the initial version in the way that it takes the fault tolerance requirement into account by combining both proactive and reactive techniques in an adaptive way. The proposed RPMFT (Replication and Preemptive Migration based Fault Tolerance) algorithm adds a powerful part to the initial version by mixing Preemptive Migration and Replication in order to proactively take preventive actions before a failure occurs and reactively avoid the effects of VM's failure. Moreover, the replication is done in balanced way as we can use task replicas to balance the load among the system. The main goal of the proposed algorithm is to ensure fault tolerance while:

- Reducing the number of replications using:
  - A hybridization of proactive and reactive approaches;
  - A reliability factor that depends on an adaptive confidence index.
- Choose the right replication nodes based on neighbors' load.

The feedback loop control mechanism is an essential component of proactive fault tolerance using preemptive migration (Fig. 1), where the application is constantly monitored and analyzed as preventative actions can be taken to avoid imminent application failure by preemptively migrating parts of an application away from nodes that may fail.

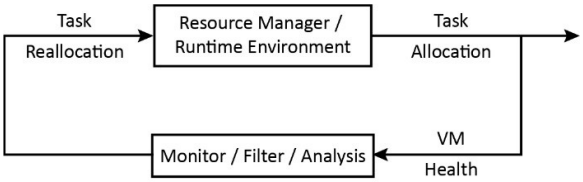


Fig. 1. Feedback-loop control

The reliability factor $REL_i(t)$ represents the health status of VM host. Once it is detected that the reliability factor does not meet an adaptive threshold called *confidence index* $CI_i(t)$, the VM's tasks are reallocated to other VMs. Tasks are migrated to the most underloaded neighbor to balance the load among the nodes in the system.

In addition to preemptive migration, a hybrid based approach is designed by involving characteristics of both active and passive replication. In this technique, one replica processes the requests received from clients. However, another replica handles the client request when the VM containing the primary replica becomes unreliable.

The proposed model handles faults that may occur in all available virtual machines. It tolerates faults based on reliability and load assessment to reschedule tasks to the most reliable virtual machines while preserving load balancing. As described in Algorithm 1, once a new task is received, it is replicated to the least loaded neighboring node that meets the following condition:

C: $(VM_j \in N_i(t)) \wedge (REL_j(t) > CI_j(t))$

Algorithm 1

```
01.  while running
02.    if receive new task T_k then
03.      REP_ik(t) = ∅
04.      Replicate T_k on VM_j satisfying condition C
05.      REP_ik(t) = REP_ik(t) ∪ {VM_j}
06.    end if
07.    if REL_i(t) < CI_i(t) then
08.      for T_m ∈ VM_i tasks set do
09.        if |REP_im(t)| = 0 then
10.          Migrate T_m on VM_j satisfying condition C
11.        end if
12.      end for
13.    end if
14.  end while
```

When we deal with fault tolerance, a policy should be used to identify when to take actions (proactively or reactively) to ensure system reliability and scalability. In this work, a reliability factor $REL_i(t)$ is used to achieve this goal. This factor is calculated based on hardware and software satus as well as the history of the machine state. Indeed, a VM relocates tasks to its neighbors only if its reliability factor is less than a confidence index threshold $CI_i(t)$.

4 Performance evaluation

The performance of the proposed algorithm is evaluated with CloudSim [14] using series of experiments. The study is conducted on sixteen data centers each consisting of five physical machines. Tasks arrive randomly with exponentially distributed service times. The simulation parameters for scenario 1 and scenario 2 are presented in Table 1.

Table 1. Simulation parameters.

	Length of tasks	**Number of VMs**	**Number of tasks**
Scenario 1	104 - 8 × 104 MI	100	200
Scenario 2	104 - 8 × 104 MI	100	1200

Three metrics are used to evaluate algorithm performance: the Overhead, the Completion Rate, and the Average CPU Utilization. The fault tolerance overhead is computed as follows:

$$Overhead_{FTSTLB} = \frac{L_{RPMFT}}{L_{FFLB}} \quad (1)$$

where L_{RPMFT} is the latency achieved by the fault tolerant algorithm, and L_{FFLB} denotes the latency achieved by the fault-free version of the algorithm.
The *Completion Rate* is the number of completed tasks when failures occur, while the *Average CPU Utilization* is the third metric used to assess the algorithm's performances which is defined as:

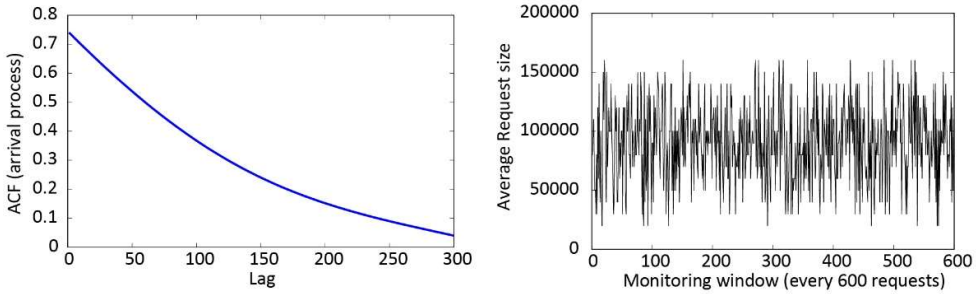
$$ACU = \frac{\sum_{task \in RT} CU_{task}}{|RT|} \quad (2)$$

where RT is the set of received tasks and CU_{task} is the amount of CPU usage when processing a *task*.

The time-to-failure of the machines is represented with a mathematical model that describes the probability of failures occurring over time which is called Weibull distribution. The Weibull failure rate function, $\lambda(t)$, is given by:

$$\lambda(t) = \frac{\beta}{\eta}\left(\frac{t-\gamma}{\eta}\right)^{\beta-1}, t \geq \gamma \geq 0, \beta > 0; \eta > 0 \quad (3)$$

where β is the shape parameter, η is the scale, and γ is the location parameter.



(a) ACF for arrival process (b) Average request size.

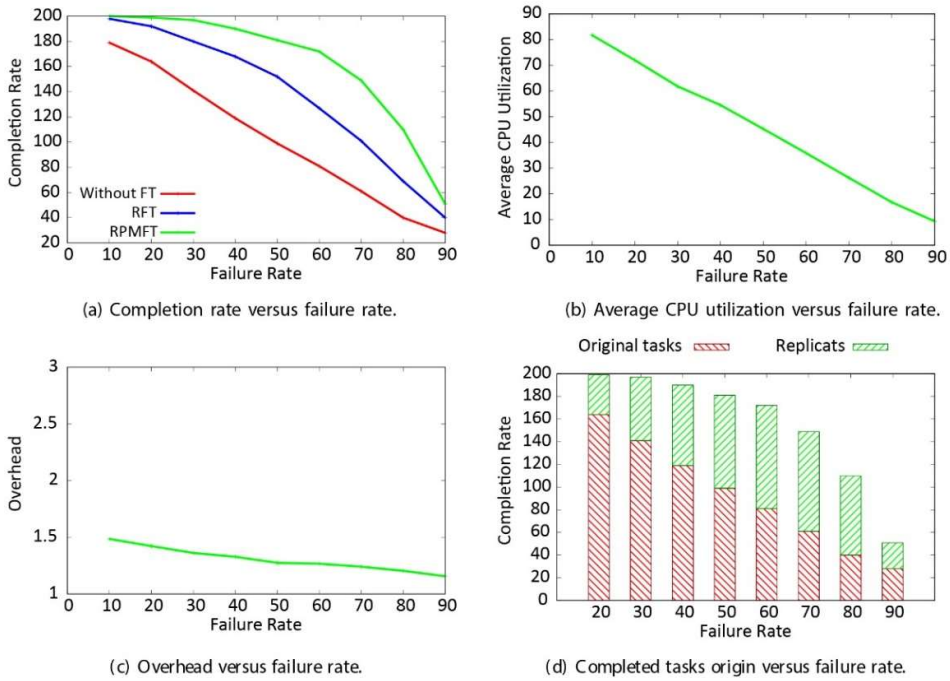
Fig. 2. ACF for arrival process used in the simulation (a), and (b) Average request size for every 600 requests for tasks with random length.

4.1. Experiment results and analysis

The RPMFT algorithm is simulated to explore its performance to achieve more completion rate and less overhead and average CPU utilization. The proposed algorithm was compared to the replication fault tolerance (RFT) approach in terms of completion rate.

Fig. 2(a) presents the autocorrelation function (ACF) of tasks arrival. Fig. 2(b) shows the requests size window every 600 requests for Scenario 1 and Scenario 2.

Fig. 3(a) shows a significant improvement in completion rate. However, the performance of all algorithms converge to the same completion rate when the failure rate is close to 100%. On average, we have 18% improvement of completion rate compared with the replication algorithm. The Average CPU utilization of RPMFT algorithm is higher as shown in Fig. 3(b) which is not surprising since we need more resources for task replications. We readily observe from Fig. 3(c) that the overhead decreases slightly together with the number of failures. This is due to the fact that the fault tolerance overhead is already absorbed by the increase of failure rate, which leads to a reduction in the number of completed tasks and thus to a decrease of the induced overhead. Fig. 3(d) reveals that the RPMFT algorithm reported an important number of completed tasks using replicas, especially for failure rates equal to 60%, 70% and 80%.



(a) Completion rate versus failure rate. (b) Average CPU utilization versus failure rate.

(c) Overhead versus failure rate. (d) Completed tasks origin versus failure rate.

Fig. 3. Completion rate (a), Average CPU utilization (b), Overhead (c) vs. failure rate, and (d) Completed tasks origin vs. failure rate for a set of 200 tasks with random length.

* Abderraziq Semmoud: abderrazak.semmoud@univ-tlemcen.dz

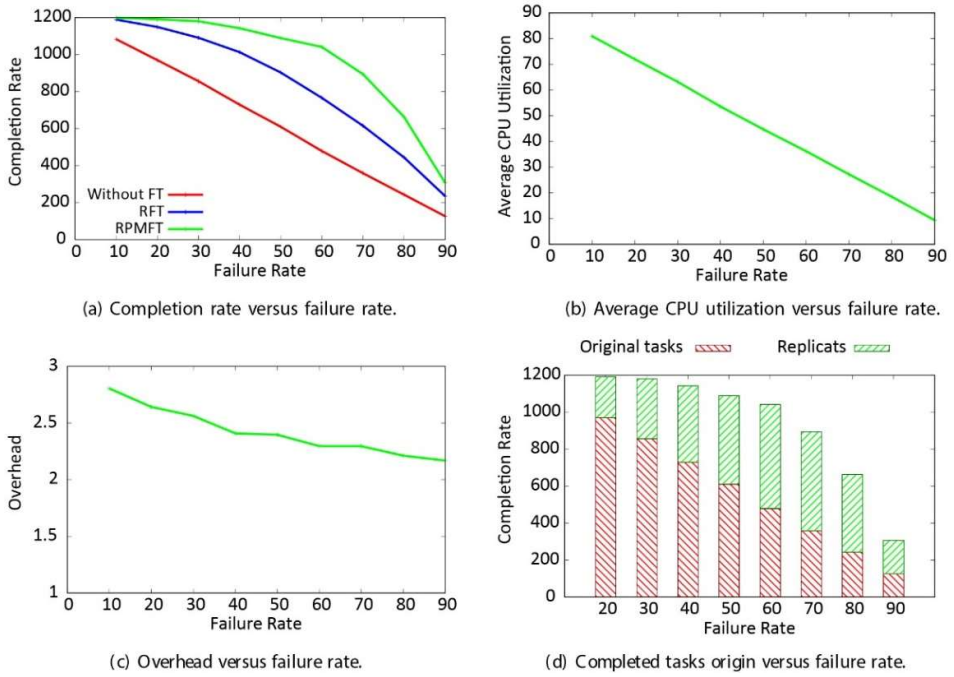


Fig. 4. Completion rate (a), Average CPU utilization (b), Overhead (c) vs. failure rate,and (d) Completed tasks origin vs. failure rate for a set of 1200 tasks with random length.

In the second scenario plotted in Fig. 4, as the failure rate increases, the completion rate decreases notably for the RFT approach that uses replication as the unique criterion for fault tolerance. The performance of our proposal is significantly better. For instance, Fig. 4(a) shows that the completion rate has been improved by 17%, and Fig. 4(d) reveals also good performances in terms of the number of completed tasks. This can be explained by the use of preemptive migration for fault tolerance only if the reliability of VMs is near a prescribed level of confidence index.

5 Conclusion

We have proposed in this paper a new distributed and adaptive fault tolerant algorithm for load balancing in cloud computing environments. Our proposal is made upon a combination of proactive and reactive fault tolerance techniques in an adaptive way. It aims to minimize the number of replications while maintaining system reliability. To this end, a reliability factor is used to decide when the migration of task-replicas should be done. Based on CloudSim simulator, it was shown that when it comes to fault tolerance and load balancing, our algorithm exhibits better performances than its direct rival RFT in all the tested scenarios. In future works, it would be interesting to add an artificial intelligence technique to improve the decision made when predicting failures and choosing the right nodes for task replication.

References

1. A. Semmoud, M. Hakem,B. Benmammar, IJHPCN, **15**, 233-248 (2019)
2. B. Ray, A. Saha, S. Khatua, S. Roy, IEEE T Cloud Comp. (2020)
3. L. Guan, H. Chen, L. Lin, IEEE Access, **9**, 21522-21531 (2021)
4. R. Chen, Y. Yao, P. Wang, K. Zhang, Z. Wang, H. Guan, B. Zang, H. Chen, IEEE T Parall. Distr., **29**, 1621-1635, (2017)
5. L. Aranda, A. Sánchez-Macián, J. Maestro, IEEE T Vlsi. Syst., **28**, 1336-1340 (2020)
6. J. Nakamura, Y. Kim, Y. Katayama, T. Masuzawa, Concurrency Pract. Ex., **33**, e5647 (2021)
7. H. Chang, Y. Chang, S. Hsiao, J Syst. Software, **93**, 102-109 (2014)
8. J. Fang, P. Chao, R. Zhang, X. Zhou, World Wide Web, **22**, 2471-2496 (2019)
9. T. Tamilvizhi,B. Parvathavarthini, Cluster Comput., **22**, 10425-10438 (2019)
10. A. Setlur, S. Nirmala, H. Singh, S. Khoriya, J Parallel Distr. Com., **136**, 14-28 (2020)
11. M. Chatterjee, A. Mitra, S. Setua, S. Roy, Comput. Electr. Eng., **81**, 106517 (2020).
12. J. Nakamura, Y. Kim, Y. Katayama, T. Masuzawa, Concurrency Pract. Ex., **33**, e5647 (2021)
13. A. Semmoud, M. Hakem, B. Benmammar, J. Charr, Concurrency Pract. Ex., **32**, e5652 (2020)
14. R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, R. Buyya, Softw. Pract. Exp., **41**, 23-50 (2011)