Resource Efficient Fault-Tolerant Computing Service Framework in Cloud

Moin Hasan
Department of Computer Science and Engineering
SantLongowal Institute of Engineering and Technology, Sangrur, India
mmoinhhasan@gmail.com

Major Singh Goraya
Department of Computer Science and Engineering
SantLongowal Institute of Engineering and Technology, Sangrur, India
mjrsingh@yahoo.com

***Abstract*—Cloud provides a vast range of computing services to the millions of diverse users across the world. Attributed to its large dimensions, complexity, and openness; delivery of services is susceptible to failures in cloud. The services which are to be delivered within the specified deadlines are worst affected in the event of a failure. Therefore, fault tolerance is always consideredas a vital matter of concern in the cloud environment. This paper contributes towards this context and proposes a robust fault tolerance framework in cloud with better resource utilization. The proposed model is tested through extensive simulation experiments. Results are obtained in terms of task delay, service reliability and system throughput.**

Keywords - *cloud computing; fault tolerance;reliability; resource utilization; cooperative computing*

I. INTRODUCTION

Cloud is an internet based computing service paradigm, whereon-demand access to a shared pool of configurable computing resources is empowered globally and conveniently,to be swiftly provisioned and released with minimal management effort [1]. The cloud computing environment is embedded with numerous attractions like *high resource availability*, *on-demand flexibility*, *low upfront cost*, etc. [2]–[4]. Due to its immense benefits, large number of organizations are into the scene as the distinguished cloud providers. *Amazon*, *Google*, *IBM*, and *Microsoft*, etc. are few of them [5], [6]. Services in cloud are delivered to the end users with respect to the software, infrastructure, and platform [1], [7]. Due to the consistent increase in the diverse cloud users, it has to respond a large number of dynamic requests at every instance of time. Every user strives for a reliable and timely response to his/her submitted request. Since, cloud is an internet based environment which uses thousands of commodity servers and possesses a complex architecture, therefore it is always susceptible to faults and failures [3], [8]–[10]. Thus, a stout resource management system is required. Extensive researches in this area have been carried out for many years and plentiful algorithms have been evolved through the time. The Cooperative Computing System (CCS) is one such framework proposed for the grid computing environment [11]. It accentuates on the reliable execution of primary tasks. In the present paper the CCS is customized to execute users' tasks in the cloud computing environment. Tasks of two priorities (Primary and Secondary) are taken into account. The primary tasks are those which are to be executed with highest priority and cannot be suspended. While, the secondary tasks have lower priorities and can be suspended during their executions. In this paper, the secondary tasks are intended to improve the overall resource utilization. They are executed using the concept of On-demand scheduling. Three parameters (task delay, service reliability and CCS throughput) are considered in order to evaluate the performance of the proposed framework. Results are obtained through extensive simulations which show that the proposed framework is very much proficient in order to reliably execute primary tasks in cloud computing environment while increasing the overall resource utilization.

The paper is organized as follows: Section 2 enlists the related work. The proposed framework is explained in section 3. Simulation and obtained results are given in section 4. The paper is concluded in section 5 along with the future scope.

II. RELATED WORKS

Cloud is an environment which simultaneously hosts a large number of requests. These requests are dynamic in nature and correspond to the big-data of workloads [12]. These workloads (applications, jobs, tasks) have to be executed by the available resources in the cloud. Due to the diversity in the end users, the tasks are of heterogeneous priorities [13]. Several researches have been carried out in cloud as well as in other parallel and distributed high performance computing environments considering the priority in input applications. Wu et al. (2012) developed an image processing system based on cloud computing [14]. They did the data distribution for image processing on the basis of priority in data. Anbazhagi et al. (2014) proposed an algorithm for dynamic

task scheduling in IaaS cloud [15]. They first sorted the input tasks according to their priorities and then performed the scheduling operations. Lu et al. (2013) studied the optimization of resource utilization in Hadoop [16]. They also considered priorities in jobs and proposed a non-intrusive slot layering solution for resource utilization.

As discussed in section I, cloud possesses a few characteristics which mark it as a fault susceptible environment. Some reasons for faults are overloaded servers, network congestion, unreliable hardware and software, human made faults, virus attacks, and natural disasters, etc. [17], [18]. Occurrence of faults ultimately degrades the system reliability if not handled properly. Many researches have targeted this area in their respective works. Qiu et al. (2014) discussed the various approaches (proactive and reactive) to handle the faults [19]. *Self-Healing*, *Preemptive migration*, and *system rejuvenation* are the proactive approaches; while *check-pointing*, *job migration*, and *task replication* are the reactive approaches used for fault tolerance in cloud [20]–[22]. It has been observed that the reactive fault tolerance approaches (mainly task replication) are more advantageous than the proactive ones because of the high resource availability in cloud [19]. Task replication is a technique by which a single task is executed on multiple resource instances. It has two basic variants viz., *active replication* and *passive replication*[23]. A good number of works have been noticed in the literature which use the replication technique. Some of the latest are discussed briefly in this section. Malik and Huet (2011) presented a fault tolerant real-time tasks' execution model in cloud computing [24]. In the presented model the real-time incoming tasks are maintained in an input buffer. Tasks in *first come first serve* (FCFS) manner are then promoted for execution. Each task is replicated on *M* virtual machines, which are embedded with different algorithms for real-time task execution. The result produced by each algorithm is moved further for the acceptance test, where the correctness of the result is verified. The results are then moved to the time checker so as to check whether the result is obtained before deadline or not. If none of the results is obtained before deadline, the task is sent back to input buffer. On the basis of the obtained results, the reliabilities of the corresponding virtual machines are adjusted. Jiang (2011) used the concept of coteries for resource allocation in grids and clouds [25]. Author showed that the use of coteries increases the fault tolerance capability of the system and minimizes the communication overhead. Jhawar et al. (2013) proposed a fault tolerance management system in cloud computing [18]. The fault tolerance is applied at the virtualization layer directly rather than at the application being deployed, by replicating the whole virtual instance. Faults are detected by a run-time monitoring system which uses heartbeat protocol. The primary component periodically sends a liveness request to all the replicated backups. A timer is maintained for each request. If the replicated backup fails to respond *N* liveness requests within a predefined time, it is considered to be failed. Chen et al. (2015) proposed an energy-efficient and fault-tolerant framework for data storage and processing in dynamic clouds [26]. For the purpose, they integrated the concept of *k-out-of-n* mechanism from distributed computing into cloud computing. In order to store and process data, two functions are developed, namely *AllocateData()* and *ProcessData()* respectively. The methodology first separates the storage requests and processing requests and passes them to their respective functions. The probability of operation failure is then estimated, on the basis of which the expected transmission time is computed. After that, the *k-out-of-n* mechanism is applied and the resources are finally allocated. Wang et al. (2015) presented a fault tolerant scheduling mechanism for real-time tasks in virtualized clouds [27]. They utilized the primary backup approach for the fault tolerance. In the proposed mechanism the users' tasks are queued in an input buffer and then transferred to the scheduler, which has three basic components viz. *resource controller*, *backup copy controller*, and *real-time controller*. As per the agreement of these components, each task is scheduled on two different virtual machines lying in different hosts. One host is called as the *primary host* and the other is called as the *backup host*. At the arrival of a new task, the two hosts are vertically scaled up in order to provide a new virtual instance to the arrived task. Similarly, in case of task departure, the hosts are vertically scaled down. Some other works using task replication are given in [28]–[30].

Though, task replication promises high service reliability, but it may cause underutilization of the computing resources unless optimized [31]. Several works in the literature consider this issue, they are briefly given as follows: Qin and Jiang (2006) proposed a fault-tolerant scheduling algorithm for real-time heterogeneous systems [32]. The authors considered the real-time tasks to be precedence constrained and used the primary backup approach to schedule them. In order to improve the throughput of the system, the backup copy of a task is overlapped with its successors' primary copies. Di and Wang (2013) proposed an error-tolerant algorithm to allocate resources and execute tasks with deadline constraints in cloud system [33]. Tasks are executed on the basis of ratio of algorithmic generated execution time and user given deadline. For the sake of resource utilization, the concept of *VM multiplexing* is used. Calheiros and Buyya (2013) proposed a fault-tolerant execution framework for scientific workflows with deadline constraints using task replication [34]. To avoid the resource wastage, the idle time of the provisioned resources are utilized to replicate the tasks.

A. *Cooperative Computing System (CCS)*

CCS is a fault-tolerant task execution framework in the grid computing environment, which emphasizes on the reliable execution of the primary tasks. Whenever a primary task arrives, a central resource manager designates a calculated number of computing resources for the execution of the arrived task. This designated assemblage of resources is called the CCS. The primary task is then scheduled on one of the resources in CCS. This resource is nominated as the ACTIVE resource. The remaining resources in the CCS are nominated as ACTIVE STANDBY resources. The ACTIVE STANDBY resources are intended to deliver execution backup in case of any fault. CCS is a repairable parallel system, meaning thereby if a resource in CCS fails, the resource manager supplants it with a new similar resource from the grid. It keeps the CCS reliable throughout the execution of the primary task. If an ACTIVE resource in CCS fails, the primary task is migrated to one of the ACTIVE STANBY resources in the CCS. This ACTIVE STANDBY resource is then nominated as the ACTIVE resource. The failed ACTIVE resource is replaced by a new similar resource which is subsequently nominated as the ACTIVE STANDBY resource. CCS provides highly reliable service to the primary tasks. The probability of primary task suspension in CCS is extremely small. To let it happen, all the resources in CCS will have to fail, which is an extremely sporadic event. The mathematical aspects (system state representation, CCS size evaluation, execution overhead, etc.) of CCS are not elucidated in this paper. The reader is therefore advised to refer [11] for the comprehensive analysis of CCS.

III. PROPOSED WORK

The CCS proposed in [11] for grid computing environment assumes that the secondary tasks are being executed on ACTIVE STANDBY resources in the background. The authors did not contemplate the execution of secondary tasks into the simulation. In the present paper the CCS is first customized for the service provisioning in a cloud computing environment rather than grid computing environment. Secondly, the assumption that the secondary tasks are being executed in background is eliminated. The performance of CCS is examined through extensive simulations by executing both primary as well as secondary tasks in the cloud datacenter.

A. *System Model*

In this paper the cloud datacenter is considered as a set D of n physical hosts connected to the datacenter manager (DM), following the widely used star topology. The *DM* contains each and every information regarding all the hosts in the datacenter. Each physical host H is considered as a set of preconfigured virtual machines.

Mathematically,

$$D = \{H_1, H_2, H_3, \dots\dots\dots, H_n\} \quad (1)$$

$$H_i = \{V_{i1}, V_{i2}, V_{i3}, \dots\dots\dots, V_{im}\} \quad (2)$$

Each virtual machine is modelled as follows:

$$V_{ij} = (c, m, s, avl) \quad (3)$$

Where; c denotes the computational power in *million instructions per second* (MIPS), m denotes the memory in *giga bytes* (GB), s denotes the storage capacity in *giga bytes* (GB), and *avl* denotes the availability state (1 for available and 0 for unavailable) of the virtual machine V_{ij}. Now, each arrived task T is modelled as follows:

$$T = (T_s, T_{pr}) \quad (4)$$

Where, T_s denotes the task size (in MI) and T_{pr} denotes task priority (primary or secondary).

B. *FTCSF*

The block diagram of *FTCSF* framework is shown in Fig. 1. Before discussing the execution methodology, let's have a look at the basic components of the framework shown in table 1.The user inputs the task to the *TM*, where it is received by the *TR*. The *TR* then transfers the received task to the *TE* in FCFS manner. Inside *TE*, the size and priority of the task is examined so that appropriate resource(s) can be allocated. After the examination, if the task priority is primary, it is promoted to the *TS*, else it is transferred to the waiting queue. The *TS* calculates the size k of the CCS resource group G on the basis of various task parameters. In this paper, the CCS size is calculated similar to that in paper [11]. However, CCS is a set G of k similar virtual machines fulfilling the task requirements. In is to be worth noted that the k virtual machines must be located on different hosts so that their failure rates should be mutually independent.

Table 1. Brief description of different framework components

Component	Meaning	Description
DM	Datacenter Manager	Manages all the resources in the datacenter
TM	Task Manager	Manages the overall execution of the arrived tasks
TR	Task Receiver	Receives the incoming tasks and stores them into a buffer
TE	Task Examiner	Examines the size and priority of the received tasks
WQ	Waiting Queue	Stores the tasks waiting for execution
TS	Task Scheduler	Schedules the received tasks on the suitable resources
RG	Report Generator	Generates the simulation results for the executed tasks

Mathematically,

$$G = \{V_1, V_2, \dots, V_k\} \;\; provided;\; host(V_1) \neq host(V_2) \neq \cdots \neq host(V_k) \qquad (5)$$

Now, the availability state of the CCS group Avl_G at time t is given as follows:

$$Avl_G(t) = avl_{V_1}(t) \vee avl_{V_2}(t) \vee \dots \vee avl_{V_k}(t) \qquad (6)$$

Where, ∨ is the logical OR operator. Equation(6) shows that the *G* will remain alive if at least one of the virtual machines in it is available. The steps of group formation are given in algorithm 1. Now, after the group formation, the primary task is scheduled on one of the resources in it and the resource is nominated as *active*. The remaining resources in *G* are nominated as *stand-by*. The *TS* then searches the waiting queue for the secondary tasks which can be scheduled on the *stand-by* resources in *G* and does the scheduling if found. The CCS group is kept alive until all the tasks in it complete. After the completion of all the tasks in *G*, it is destructed and the *RG* generates the corresponding results. The steps of task execution are given in algorithm 2.

CCS is a fault tolerant framework for the primary task. *Active* resource in *G* periodically sends the execution state of the primary task to the *stand-by* resources. Whenever an *active* resource in fails, the primary task is migrated to one of the *stand-by* resources in *G* and resumed the execution from the last updated state. In this paper the methodology of task migration is slightly modified. If a primary task needs to be migrated, the group *G* is searched for an idle *stand-by* resource. If such a resource is found, the primary task is migrated to this resource. Doing this elude the possible preemption of the secondary task from execution, which consequently improves the overall performance of the system. In case there is no idle *stand-by* resource available in the group, the primary task is then migrated to any *stand-by* resource by suspending the secondary task being executed on this resource. On the other hand, whenever a *stand-by* resource fails, the secondary task (if being executed on the failed resource) is simply suspended. In the meantime, the *RM* repairs *G* by replacing the failed resource by a similar one from the datacenter.

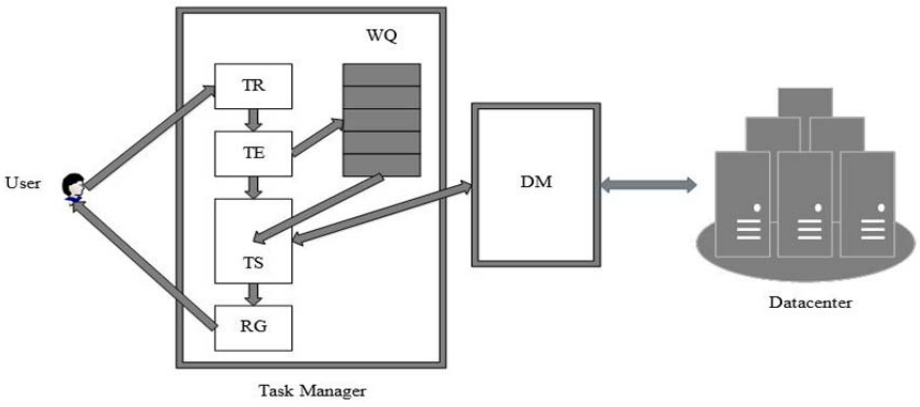

Figure 1. Block diagram of FTCSF

Algorithm 1
Input: Primary task T

1. G ← {}
2. k ← size(G)
3. count ← 0
4. For each host H in D
    a. For each V in H
        i. If capability(V) = requirement(T)
            1. G ← {V}
            2. Exit for
        ii. End if
    b. End for
    c. Increment count
    d. If count = k
        i. Exit for
5. End for
6. Return G

Algorithm 2
Input: Task T

1. TR ← T
2. TE ← T
    a. T_s ← getSize(T)
    b. T_{pr} ← getPriority(T)
    c. If T_{pr} = primary
        i. Go to step 3
    d. Else
        i. WQ ← T
    e. End If
3. TS ← T
4. G ← formGroup(T)
5. scheduleTask(T)
6. nominateResources(G) //Nominate resources as active and stand-by
7. If taskState(T) = running
    a. For each idle stand-by V in G
        i. search(WQ)
        ii. If secondary task T' found
        iii. scheduleTask(T')
    b. End for
8. Else //Task state is completed
    a. If all the tasks in G completed
        i. destructGroup(G)
    b. Else
        i. Wait for all tasks to be completed
        ii. Go to step 8(a)
    c. End If
9. End If
10. Generate Report

IV. SIMULATION AND RESULTS

The proposed FTCSF framework is simulated by executing 1000 primary tasks by varying the mean task duration (MTD) and the resource failure/repair rate. The performance is evaluated in terms of task delay, service reliability, and CCS group throughput. The simulation variables are shown in table 2.

Table 2: Simulation variables details

Simulation Variable	Values Taken
Mean Task Duration	30, 60, and 90 (minutes)
Failure/ Repair Rate	2, 4, and 6 (per hour)

A. Task Delay

Average task delay (in minutes) for primary, secondary, as well as total tasks is calculated. The results with respect to mean task duration are shown in Fig. 2, Fig. 3 and Fig. 4 at respective resource failure/repair rate equal to 2, 4 and 6 per hour. It is clearly noticeable from the graph in Fig. 2 that the average delay in primary tasks at resource failure/repair rate equal to 2 per hour is below 0.5 minutes for all mean task durations. The maximum delay observed in primary tasks is 1.5 minutes approximately for mean task duration equal to 90 minutes at resource failure/repair rate equal to 6 per hour as shown in Fig. 4. It shows that presented framework completes the primary tasks with negligible delay.

B. Service Reliability

Reliability of a system is defined as the probability by the virtue of which the system accomplishes its purpose adequately for a specified period under the provided working conditions. In this paper the service reliability of the CCS group for primary task, secondary task, as well overall tasks is calculated. The results are shown in Fig. 5, Fig. 6 and Fig. 7 with respect to mean task duration at resource failure/repair rate equal to 2, 4 and 6 per hour respectively. From the graphs it can be noticed that the service reliability for primary tasks is touching 1. It means that the FTCSF is highly reliable for the execution of primary tasks. The service reliability for secondary tasks is almost 0.5 and the overall service reliability is almost 0.6. One thing to be noticed here is that the service reliability is almost similar with respect to each value of mean task duration irrespective of the resource failure/repair rate.

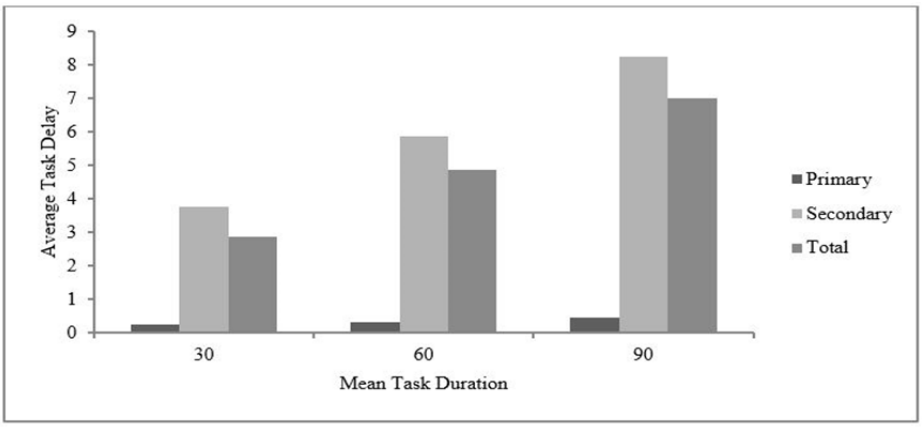

Figure 2. Average task delay at failure/repair rate = 2 per hour

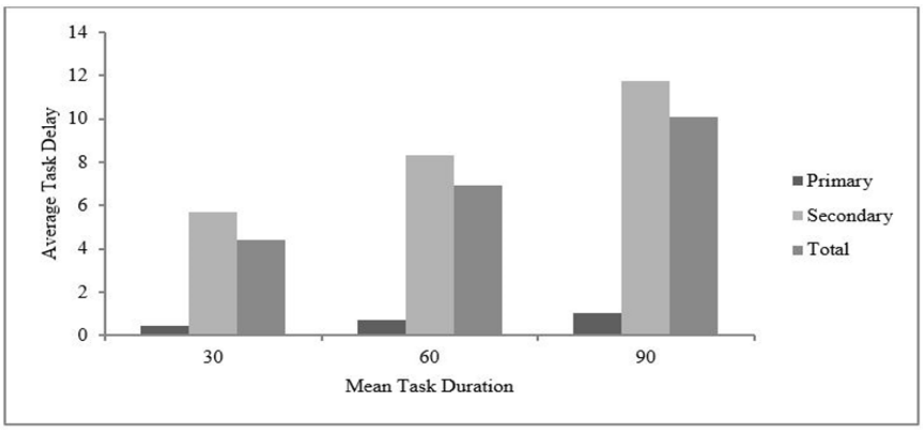

Figure 3. Average task delay at failure/repair rate = 4 per hour

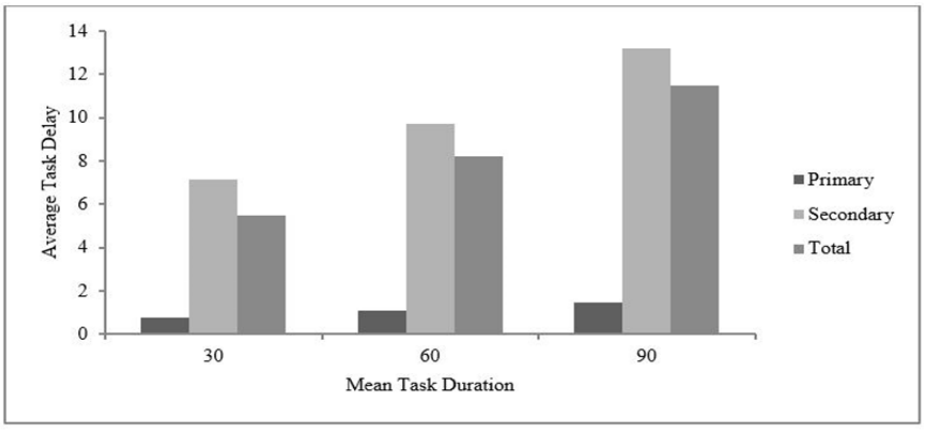

Figure 4. Average task delay at failure/repair rate = 6 per hour

This is because, in this paper, the resource failure rate and repair rate are set to be equal. However, a difference in resource failure and repair rate may alter the value of reliability. The other thing to be noticed is that the overall service reliability is closer to that of secondary tasks. It is due to the fact that number of primary tasks being executed in a CCS group cannot be more than 1. However, the number of secondary tasks at any instance can reach up to (*k-1*), where *k* is the total number of resources in *G*. Therefore, the value of overall service reliability shifts towards the value of that for secondary tasks rather that the primary tasks.

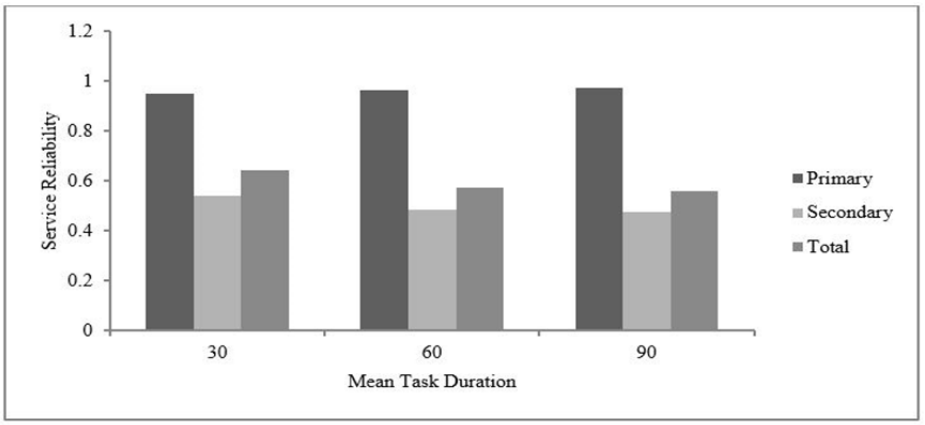

Figure 5. Service Reliability at failure/repair rate = 2 per hour

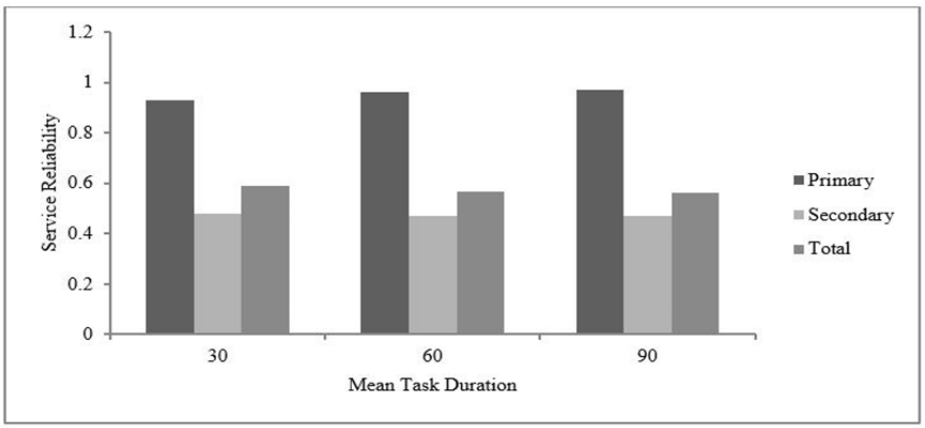

Figure 6. Service Reliability at failure/repair rate = 4 per hour

C. Throughput

For a CCS group *G*, the number of tasks executed by it per unit of time is called its throughput. In this paper the throughput of *G* is calculated as the number of tasks executed in it per hour. Obtained results are shown in Fig. 8. It can be noticed that the throughput in not much affected by the variation in resource failure/repair rate. It is also

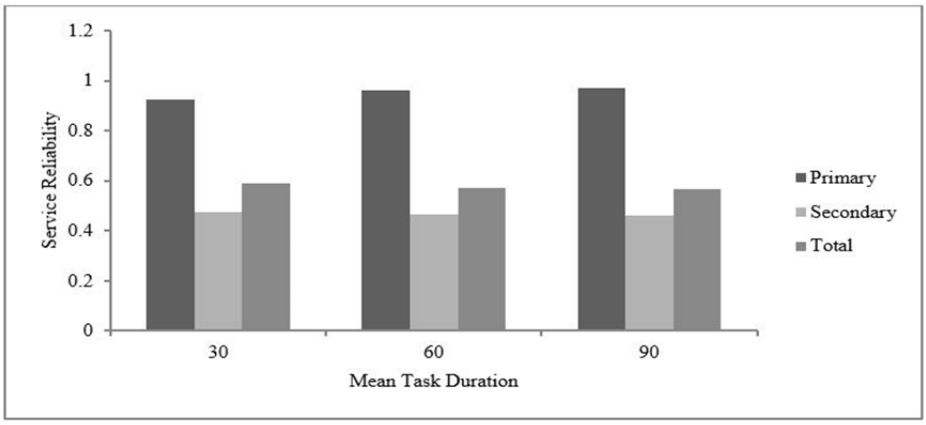

Figure 7. Service Reliability at failure/repair rate = 6 per hour

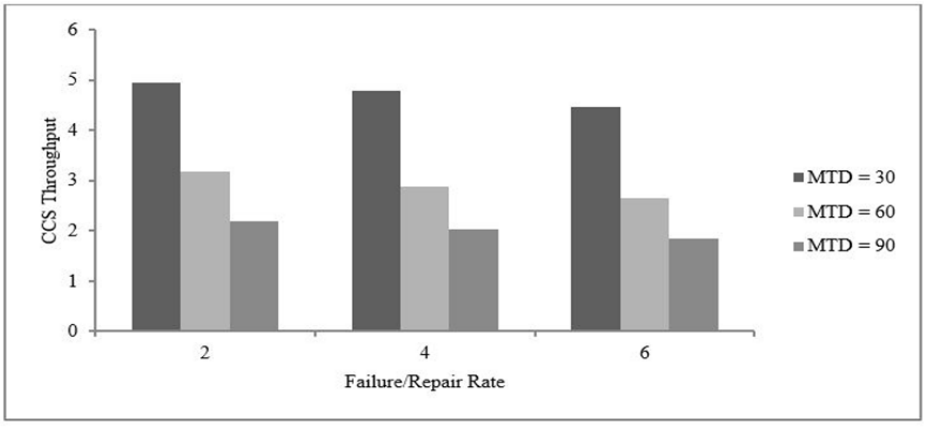

Figure 8. Throughput of CCS group *G*

because of the same rate of resource failure and repair. However, difference in rate may cause the throughput to be affected. Moreover, the throughput decreases with increase in the mean task duration, which is an obvious fact.

V. CONCLUSIONS

Fault tolerance is one of the major research areas in cloud computing. Plentiful research has been done and still being carried out. Cloud computing is not just confined to a single cloud now. Multi-clouds, collaborative clouds, etc. are now coming into the sight. This changing in trends is one the reasons for keeping the research alive in this arena. In this paper a reliable framework is presented to execute primary tasks in a cloud datacenter. In order to achieve the desired goal, the concept of CCS is utilized. To optimize the resource utilization in CCS group, secondary tasks are also executed as per the concept of on-demand scheduling. Simulation results show that the presented framework is a promising one in terms of the considered parameters. The proposed framework is not compared with any existing framework in this paper, this aspect will be taken in future. Moreover, the optimization of group formation with respect to its size and communication overhead is also left for the future.

ACKNOWLEDGMENT

This work is carried out under the research fellowship provided by SantLongowal Institute of Engineering and Technology, Sangrur, India.

REFERENCES

[1] P. Mell and T. Grance, "The NIST definition of cloud computing recommendations of the National Institute of Standards and Technology," Natl. Inst. Stand. Technol. Inf. Technol. Lab., vol. 145, p. 7, 2011.

[2] B. K. Rani, B. P. Rani, and A. V. Babu, "Cloud Computing and Inter-Clouds – Types, Topologies and Research Issues," Procedia Comput. Sci., vol. 50, pp. 24–29, 2015.

[3] M. A. Vouk, "Cloud Computing – Issues, Research and Implementations," vol. 16, no. 4, pp. 235–246, 2008.

[4] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, "A view of cloud computing," Commun. ACM, vol. 53, no. 4, pp. 50–58, 2010.

[5] H. Shen and G. Liu, "An efficient and trustworthy resource sharing platform for collaborative cloud computing," IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 4, pp. 862–875, 2014.

[6] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," Proc. 10th Annu. Conf. Internet Meas. - IMC '10, p. 1, 2010.

[7] D. Puthal, B. P. S. Sahoo, S. Mishra, and S. Swain, "Cloud Computing Features, Issues, and Challenges: A Big Picture," 2015 Int. Conf. Comput. Intell. Networks, pp. 116–123, 2015.

[8] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?," in USENIX Symposium on Internet Technologies and Systems, 2003, pp. 1–15.

[9] M. Ali, S. U. Khan, and A. V. Vasilakos, "Security in cloud computing: opportunities and challenges," Inf. Sci. (Ny)., vol. 305, pp. 357–383, 2015.

[10] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," J. Internet Serv. Appl., vol. 1, no. 1, pp. 7–18, 2010.

[11] M. S. Goraya and L. Kaur, "Fault tolerance task execution through cooperative computing in grid," Parallel Process. Lett., vol. 23, no. 1, pp. 1–20, 2013.

[12] F. Zhang, J. Cao, W. Tan, S. U. Khan, K. Li, and A. Y. Zomaya, "Evolutionary scheduling of dynamic multitasking workloads for big-data analytics in elastic cloud," IEEE Trans. Emerg. Top. Comput., vol. 2, no. 3, pp. 338–351, 2014.

[13] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," Futur. Gener. Comput. Syst., vol. 48, pp. 1–18, 2015.

[14] T. Y. Wu, C. Y. Chen, L. S. Kuo, W. T. Lee, and H. C. Chao, "Cloud-based image processing system with priority-based data distribution mechanism," Comput. Commun., vol. 35, no. 15, pp. 1809–1818, 2012.

[15] Anbazhagi, L. Tamilselvan, and Shakkeera, "QoS based dynamic task scheduling in IaaS cloud," in 2014 International Conference on Recent Trends in Information Technology, 2014, pp. 1–8.

[16] P. Lu, Y. C. Lee, and A. Y. Zomaya, "Non-intrusive slot layering in Hadoop," Proc. - 13th IEEE/ACM Int. Symp. Clust. Cloud, Grid Comput. CCGrid 2013, pp. 253–260, 2013.

[17] A. Abid, M. T. Khemakhem, S. Marzouk, M. Ben Jemaa, T. Monteil, and K. Drira, "Toward antifragile cloud computing infrastructures," Procedia Comput. Sci., vol. 32, pp. 850–855, 2014.

[18] R. Jhawar, V. Piuri, and M. Santambrogio, "Fault tolerance management in cloud computing: a system-level perspective," IEEE Syst. J., vol. 7, no. 2, pp. 288–297, 2013.

[19] W. Qiu, Z. Zheng, X. Wang, X. Yang, and M. R. Lyu, "Reliability-based design optimization for cloud migration," IEEE Trans. Serv. Comput., vol. 7, no. 2, pp. 223–236, 2014.

[20] M. N. Cheraghlou, A. Khadem-Zadeh, and M. Haghparast, "A survey of fault tolerance architecture in cloud computing," J. Netw. Comput. Appl., vol. 61, pp. 81–92, 2016.

[21] S. M. A. Ataallah, S. M. Nassar, and E. E. Hemayed, "Fault tolerance in cloud computing - Survey," in 11th International Computer Engineering Conference, 2015, pp. 241–245.

[22] Z. Xie, H. Sun, and K. Saluja, "A survey of software fault tolerance techniques," Univ. Wisconsin-Madison/Department …, 2006.

[23] W. Zhao, P. M. Melliar-Smith, and L. E. Moser, "Fault tolerance middleware for cloud computing," in Proceedings IEEE 3rd International Conference on Cloud Computing, 2010, pp. 67–74.

[24] S. Malik and F. Huet, "Adaptive fault tolerance in real time cloud computing," in Proceedings - IEEE World Congress on Services, 2011, pp. 280–287.

[25] J. R. Jiang, "Nondominated local coteries for resource allocation in grids and clouds," Inf. Process. Lett., vol. 111, no. 8, pp. 379–384, 2011.

[26] C.-A. Chen, M. Won, R. Stoleru, and G. G. Xie, "Energy-efficient fault-tolerant data storage and processing in mobile cloud," IEEE Trans. Cloud Comput., vol. 3, no. 1, pp. 28–41, 2015.

[27] J. Wang, W. Bao, X. Zhu, T. Yang, and Y. Xiang, "FESTAL: fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized cloud," IEEE Trans. Comput., vol. 64, no. 9, pp. 2545–2558, 2015.

[28] M. R. Arunya and M. Karthick, "An adaptive fault tolerance management in cloud computing using replication and fragmentation," Int. J. Softw. Hardw. Reserach Eng., vol. 1, no. 4, pp. 79–81, 2013.

[29] M. Dobber, R. Van Der Mei, and G. Koole, "Dynamic Load Balancing and Job Replication in a Global-Scale Grid Environment: A Comparison," IEEE Trans. Parallel Distrib. Syst., vol. 20, no. 2, pp. 1–12, 2009.

[30] M. Vardhan, A. Goel, A. Verma, and D. S. Kushwaha, "A Dynamic Fault Tolerant Threshold based Replication Mechanism in Distributed Environment," Procedia Technol., vol. 6, pp. 188–195, 2012.

[31] W. Ahmed and Y. W. Wu, "A survey on reliability in distributed systems," J. Comput. Syst. Sci., vol. 79, no. 8, pp. 1243–1255, 2013.

[32] X. Qin and H. Jiang, "A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems," Parallel Comput., vol. 32, no. 5–6, pp. 331–356, 2006.

[33] S. Di and C. L. Wang, "Error-tolerant resource allocation and payment minimization for cloud system," IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 6, pp. 1097–1106, 2013.

[34] R. N. Calheiros, R. Buyya, and S. Member, "Meeting Deadlines of Scientific Workflows in Public Clouds with Tasks Replication," IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 7, pp. 1787–1796, 2013.

AUTHORS PROFILE

Moin Hasan received the B.E. degree in computer science and engineering from Sant Longowal Institute of Engineering and Technology, Sangrur, India, in 2012 and the master's degree in computer science and engineering from Mangalayatan University, Aligarh, India, in 2014. He then worked as a Research Scholar in the Department of Computer Science and Engineering, Sant Longowal Institute of Engineering and Technology, Sangrur, India, until February 2017. His areas of research interest include scheduling, load balancing and fault tolerance in parallel and distributed systems, including grid and cloud computing.

Major S. Goraya received the B.E. degree in computer science and engineering from Sant Longowal Institute of Engineering and Technology, Sangrur, India, in 1997 and the master's and PhD degrees in computer science and engineering from Punjabi University, Patiala, India, in 2003 and 2013, respectively. He is currently working as Associate Professor in the Department of Computer Science and Engineering, Sant Longowal Institute of Engineering and Technology, Sangrur, India. His research interests include grid computing, cloud computing, and distributed computing.