IEICE TRANS. INF. & SYST., VOL.E101–D, NO.12 DECEMBER 2018

PAPER Special Section on Parallel and Distributed Computing and Networking

Avoiding Performance Impacts by Re-Replication Workload Shifting in HDFS Based Cloud Storage*

Thanda SHWE[†a)], *Nonmember and* **Masayoshi ARITSUGI**[††b)], *Senior Member*

SUMMARY Data replication in cloud storage systems brings a lot of benefits, such as fault tolerance, data availability, data locality and load balancing both from reliability and performance perspectives. However, each time a datanode fails, data blocks stored on the failed datanode must be restored to maintain replication level. This may be a large burden for the system in which resources are highly utilized with users' application workloads. Although there have been many proposals for replication, the approach of re-replication has not been properly addressed yet. In this paper, we present a deferred re-replication algorithm to dynamically shift the re-replication workload based on current resource utilization status of the system. As workload pattern varies depending on the time of the day, simulation results from synthetic workload demonstrate a large opportunity for minimizing impacts on users' application workloads with the simple algorithm that adjusts re-replication based on current resource utilization. Our approach can reduce performance impacts on users' application workloads while ensuring the same reliability level as default HDFS can provide.

key words: *re-replication, fault tolerance, data reliability, HDFS*

1. Introduction

With the increasing demand of cloud storage year by year, storing, processing and managing a large amount of data efficiently on cloud storage have raised significant concerns especially in maintaining certain level of guarantee for data. Node failures are common in cloud computing infrastructures which are accommodated on commodity servers. As node failures occur frequently, cloud storage file system, such as Google File System [1] and Hadoop Distributed File System [2] employ data replication, i.e., storing a single data block in several different locations for fault tolerance.

Although data replication brings a lot of benefits, such as providing fault tolerance, data locality, data availability and concurrent access to the data for load balancing, each time a datanode fails, data blocks on the failed node must be re-replicated to other nodes to maintain the minimum replication level. If datanodes fail frequently under independent or correlated failure and failed datanodes hold millions of data blocks, the resources needed to support data restoration processes can be high and users' application workloads will compete with re-replication workloads for resources. It may incur performance impacts on users' application workloads, resulting in introduction of QoS violation. Triggering re-replication with slow rate can be a simple solution to solve this problem. However, the ultimate goal of replication is to give full guarantee for fault tolerance, hence triggering re-replication with slow rate can decrease the durability of data. Thus, two opposing trends come into play when re-replication is performed. Speedy re-replication alone can minimize data loss which increases reliability and data durability but performance also degrades. There is a trade-off in performing re-replication.

Servers in data center have high variations in resource usage and resource usage varies in different period of the day [3], [4]. In [3], the authors studied load variations in different types of server workloads for one day period and proved that server workloads have high variability in different period of the day. In [4], average hourly and daily web access logs were studied for 5 weeks and reported that web access pattern tends to follow the similar trend and peak hours are between 11:00 and 17:00.

In view of this, it is necessary to perform the re-replication process effectively based on resource usage status of the whole cluster that allows us to reduce performance impacts on users' application workloads while ensuring the durability of data. The strategy must need to consider carefully both reliability and performance issues.

To address this problem, other studies [5], [6] demonstrated that data block restoration processes created a high load on a small number of nodes and suffered load imbalance in the system. They presented replica reconstruction schemes with the purpose of balancing re-replication jobs among the nodes in the cluster in order to reduce load imbalance during re-replication phase. Balancing only re-replication jobs without considering current running users' application jobs can decrease the performance of the current running users' application jobs. Contrary to their work that only considers balancing re-replication jobs, our previous work [7] considered to balance both users' application workload and re-replication workload during re-replication time as well as scheduled re-replication based on performance status of datanodes and popularity and reliability of data blocks. Although this approach succeeded in balancing all the servers workloads and achieved high re-replication

Manuscript received January 6, 2018.
Manuscript revised May 29, 2018.
Manuscript publicized September 18, 2018.

[†]The author is with Department of Computer Science and Electrical Engineering, Graduate School of Science and Technology, Kumamoto University, Kumamoto-shi, 860–8555 Japan.
[††]The author is with Big Data Science and Technology, Faculty of Advanced Science and Technology, Kumamoto University, Kumamoto-shi, 860–8555 Japan.

*This paper is an extended version of the paper published in the proceedings of 10th IEEE/ACM International Conference on Utility and Cloud Computing.

a) E-mail: thandashwe@gmail.com
b) E-mail: aritsugi@cs.kumamoto-u.ac.jp

DOI: 10.1587/transinf.2018PAP0017

throughput, reduction of re-replication time and probability of overload condition, this re-replication strategy utilized the fact that server utilization in the cluster is under 50% and performed the speedy re-replication by consuming resources more than baseline HDFS. Thus, if the cluster is highly or fully utilized with regular users' workload, speedy re-replication can decrease the performance of users' jobs.

Inspired by our previous work, in this paper, we go a step further and investigate a re-replication scheme called deferred re-replication which adjusts re-replication based on the average utilization of the whole cluster. The main contributions of this paper are as follows:

1. We present deferred re-replication scheme that can minimize performance impacts on users' application workloads while ensuring reliability level as default HDFS re-replication.
2. We develop a simple algorithm that adjusts re-replication based on current resource utilization of the whole cluster and performs re-replication during idle periods.
3. We employ priority based replicas grouping that restores popular and critical data first which consequently improves the performance and reliability of the system.
4. We evaluate the deferred re-replication algorithm on CloudSim with various parameter choices in a 400-nodes cluster. We show that our deferred re-replication algorithm can reduce performance impacts on users' application jobs and maintain the reliability as default HDFS re-replication.

We explore the possibility of deferring some portions of re-replication to a later time and investigate this deferred re-replication scheme can reduce the performance impacts on the users' application workloads and maintain the same reliability level as the default re-replication scheme. The deferred re-replication performs re-replication based on the observation of average utilization of the whole cluster. In particular, if the resources are under-utilized, re-replication will be performed quickly. Thus, current running users' application jobs do not need to compete with re-replication jobs, resulting in minimizing performance impacts on current running users' jobs. If the resources are highly utilized at the time of re-replication, immediate re-replication is performed for some high priority groups, i.e., for the data blocks which have only one replica and are popular. For other replica groups, re-replication will be deferred by shifting the time at which re-replication is performed. The re-replication scheme observes the utilization status of the cluster periodically in the background and schedules re-replication when the average utilization of the cluster is under pre-defined threshold.

The rest of the paper is organized as follows. The related work is discussed in Sect. 2, and how Hadoop distributed file system does replication and re-replication is presented in Sect. 3. Problem formulation and motivating example is described in Sect. 4. In Sect. 5, we present the design and approach used for deferred re-replication. The simulation results are reported in Sect. 6. Finally, the paper is concluded in Sect. 7.

2. Related Work

In this section, we discuss relevant studies in the following three main categories:

Data re-replication: Most of the studies in data replication has targeted for data locality, reliability and energy efficiency issues. Very limited studies focused on the specific cases of re-replication in HDFS based storage system. Higai et al. [5] proposed replica reconstruction schemes for a single rack cluster that balanced the workloads of replication processes by minimizing the difference of the amount of data transfer of each node during the re-replication phase. In their work, the nodes were connected with ring topology virtually and data blocks were transferred based on this one-directional ring structure. This work succeeded in improvement of replica reconstruction throughput for the single rack cluster and then they continued their work for multiple racks cluster [6]. During data restoration process, data transfer withing a rack was performed based on the one-directional ring structure and inter-rack data transfer was carried out in a round robin manner. The source and destination datanodes were selected to balance re-replication workloads on each datanode. Although these works are the closest studies to our proposed deferred re-replication, their work focused on balancing the load of re-replication jobs, and the impacts on the foreground process was rarely considered. Our deferred re-replication differs from the studies in a way that re-replication jobs are scheduled based on current resource utilization status of the whole cluster and re-replication workloads are allocated to datanodes with the goal of minimizing impacts on the users' application jobs while maintaining reliability guarantees.

Reliability in distributed storage system: As the cloud storage system scales up, assuring and maintaining reliability has become critical concerns. These cloud storage systems are built from commodity servers in which component failures occur in a daily basis. Thus, several studies have addressed the issue of reliability in data storage system. Cidon et al. [8] presented copyset replication that split the datanodes into copysets and stored a single chunk of data only on one copyset. The data loss event occured only when all nodes of same copyset failed simultaneously under correlated failures, resulting in reduction of probability of data loss. They also proposed tiered replication [9] that split the cluster into primary and backup tiers. In that work, they demonstrated that two replicas were enough for protecting data loss under independent node failure while three replicas were enough to protect data under correlated failure and stored the first two replicas on primary tier and the third replica on backup tier without effecting the performance. In [10], Wang et al. proposed a comprehensive re-

liability model that considered not only probability of data loss but also bandwidth allocation in the recovery process. They used proposed reliability model for analyzing reliability and system repair rate for different data layout schemes, namely copyset replication, shifted declustering and random declustering layouts. In [11], Li et al. proposed a novel cost-effective data reliability management mechanism based on proactive replica checking (PRCR) that checked the availability of replicas to maintain reliability. They showed that default three way replication strategy consumed storage space for rarely accessed files. Thus, they proposed a reliability model with the aim of reducing storage cost and demonstrated that wide range of data reliability can be assured with the maximum of two replicas stored in the cloud. Most of these works have focused on developing an analytical model that can give accurate reliability results and the impact of data layout on reliability. Their approaches targeted to evaluate the reliability of the system and to reduce the data loss probability but the aspect of restoring the lost blocks effectively was not studied. In contrast to these works, we consider to restore data blocks effectively in consideration of both reliability of the system and performance impacts on the users' application workloads.

Performance impact in case of data recovery: Replication in HDFS is not only providing fault tolerance but also serving of multiple parallel requests for higher data access rate. Replication improves read performance as it can load-balance read requests across multiple replicas. Thus, some of the studies proposed schemes to recover popular data immediately. Wu et al. [12] proposed Intelligent Data Outsourcing (IDO) that dynamically determined the popularity of data during normal operation state and proactively migrated them from the degraded RAID set to surrogate RAID set in data centers. The proactive migration of popular data blocks was based on the fact that occurrence of background tasks, such as RAID reconstruction, RAID resynchronization, disk scrubbing, and RAID reshape was predictable and proactive migration could improve performance of both foreground and background tasks. The main concepts of this two studies was migration or recovery of popular data proactively by predicting the occurrence of background tasks. However, minimizing performance impacts on users' application jobs while performing data recovery and migration was not addressed. In [13], popularity-based proactive data recovery for HDFS RAID systems(PP) was studied and it was proactive in the sense that popular data were determined before data failure to reduce the preparation time. In case of failure, popular data were recovered first. In the whole, PP intended to reduce the data block reconstruction time in HDFS RAID, consequently reducing execution time of users' requests that accessed the missing data. In HDFS RAID, accessing missing data blocks needs to wait reconstruction of the data blocks to be completed whereas in original HDFS, users' request can be re-directed to another copy. In PP, recovery of data blocks in a single data file was mainly addressed because this is primary role of data reconstruction in HDFS RAID environment. Generally, PP focused on minimizing performance impacts only on specific tasks that requested the missing data but it did not address the impact of massive re-replication workloads on all of normal cluster workloads due to several data nodes failure. In contrast to this, our deferred re-replication pays attention to minimize performance impacts of re-replication workloads on all of users' application workloads in the system during re-replication. Although our deferred re-replication has the same intention with PP in increasing performance by restoring popular data first, PP applied it to reduce the execution time of tasks that will directly access the popular data blocks whereas our deferred re-replication strategy applies it to balance future access load to popular data on many replicas as possible. Hence, our work is not directly comparable as the data block reconstruction objective and context differs significantly.

3. HDFS Revisited

Hadoop distributed file system(HDFS) is installed on the clusters to store, process and manage large files. Files on HDFS are divided into multiple data blocks and stored in different datanodes that can access them in parallel, enabling to give faster speed for data processing. HDFS operates with a single namenode and multiple datanodes. The namenode is responsible for metadata management of all the files and periodically checkup of healthy status of datanodes through heartbeat message. Datanodes store and retrieve data blocks as files of their local file system. They report the list of blocks they hold to the namenode and they also send heartbeat message regularly to inform the namenode that they are alive [14]. Figure 1 shows the architecture of HDFS.

HDFS stores multiple copies of single data in different datanodes to protect against data loss. Although one of the ultimate goals of data replication in HDFS is providing fault tolerance, it is also meant for performance issues, such as data locality which means accessible to the nearest data block with the node where computation is performed and load balancing which means increase of read performance by sharing file requests on multiple replicas. HDFS stores three replicas of data by default with rack aware replica placement policy and replication factor can also be config-

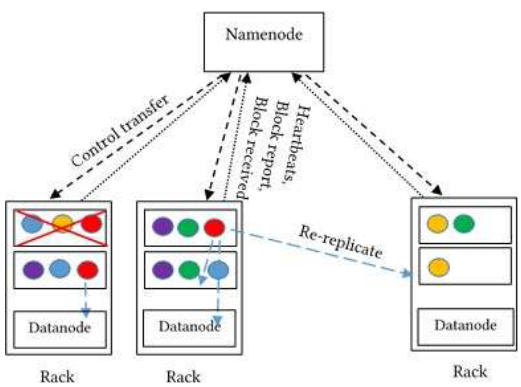

Fig. 1 HDFS Architecture.

ured by the user. The rack aware replica placement policy allocates primary replica on one node in the local rack. The second replica is placed on a node in a different remote rack. The third replica is placed on a different node in the same remote rack. Datanodes do not hold same replicas. Except these two conditions, HDFS selects source and destination datanodes in random manner [14].

Since HDFS clusters can be built on commodity hardware, failure of datanodes occurs frequently. When the namenode detects node failure or detects some of the data blocks that fall under replication level, it restores lost data blocks to remaining datanodes. This data restoration process is called re-replication and there are many potential candidate nodes to be the source and destination datanodes. But HDFS selects source and destination datanodes randomly. HDFS configuration parameters, such as dfs.namenode.replication.work.multiplier.per.iteration and dfs.namenode.replication.max-streams can be adjusted and throttled depending on how much the re-replication process wants to be speed up.

4. Problem Formulation and Motivating Example

The burden incurred by re-replication is significant if the failed datanode holds several thousands of data blocks. Copying data blocks from one node to another during re-replication process can have impacts on cluster normal jobs and keeps the system busy. The re-replication strategy in our previous work [7] consumed more resources than baseline HDFS for its speedy re-replication. This could not be a problem if the cluster is lightly utilized. But if the cluster is fully or highly utilized, it may have impact on the performance of users' application workloads.

We illustrate this issue by a motivating example. Consider a system that consists of 3 nodes as in Fig. 2. Assume that if node(s) failure occurs at time T1 and it may be a single node failure or correlated failure with several nodes. At that time, cluster is lightly utilized with average utilization around 30%. Most of the nodes are lightly utilized and some of the nodes may be over-utilized. Restoring large amount of blocks by consuming high amount of resources could not be significant burden for the system. But if node failure occurs at time T2, at that time cluster is highly utilized with average utilization around 70%. Thus, performing all the re-replication jobs at that time may be a burden for the system. At next time, time T3, the cluster is lightly utilized and deferring re-replication jobs to time T3 will be potential solution to minimize the performance impacts on cluster normal jobs incurred by re-replication jobs. There are several factors to be considered in order to defer the re-replication to next time.

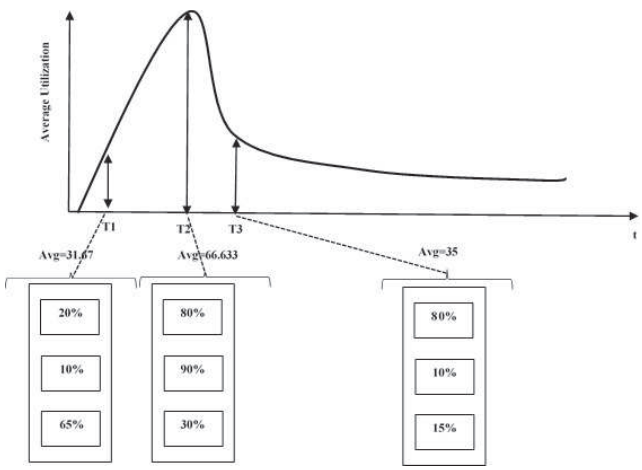

Fig. 2 Re-replication based on average utilization.

- Reliability: From the reliability perspective, deferring re-replication to later time can increase the probability of data loss. Thus, system must be able to maintain reliability level for any type of failure.
- Load balancing: Deferring re-replication itself cannot get load balancing among the nodes in the system because it can only decide whether the whole system is highly or lightly utilized. Thus, re-replication policy must consider variability of utilization among the nodes and balance both for user application workload and re-replication workload.

As our previous work appropriately solved load imbalance among the nodes, our focus of this paper is to investigate whether shifting re-replication workload to a later time can reduce performance impacts on the users' application workloads without sacrificing reliability of the system.

5. Designing Deferred Re-Replication

A key design decision in deferred re-replication scheme is to defer re-replication based on current utilization of the whole cluster while maintaining the same reliability level as default re-replication. In this section, we motivate this design decision by presenting facts which demonstrate that deferred re-replication can potentially lead to reduced impacts on cluster foreground tasks.

- **Reduce impact on foreground task**: At the time when the cluster is being fully or highly utilized, a node failure will lead to the unavoidable fact that user jobs will compete with re-replication jobs for resource usage. By adjusting re-replication jobs based on the resource usage, impact on foreground task by re-replication can be reduced.
- **Provide reliability as baseline**: In spite of shifting the time of re-replication to later time, deferred re-replication can provide same reliability level as baseline because the high priority group is always re-replicated immediately, while re-replication to other re-replication priority groups can only be deferred.
- **Improve performance**: As data replication in HDFS is more than reliability and is also meant for performance,

deferred re-replication can support performance improvement by re-replicating popular data first.

5.1 Deferred Re-Replication Based on Resource Utilization

The deferred re-replication selectively performs based on average utilization of the whole cluster. In this paper, we propose a simple approach to minimize performance impacts on users' applications jobs while ensuring the reliability level as baseline HDFS. In particular, we use average utilization of the whole cluster to decide the time re-replication is scheduled. Since the re-replication process incurs overhead to the system, it should be activated with careful manner. Thus, in our approach, re-replication for low priority groups will only be triggered whenever the system average utilization is under pre-defined utilization threshold while re-replication for high priority groups will be performed immediately. Our previous work in [7] utilized the fact that average resource utilization of servers in the cluster is under 50% [15], [16] and performed re-replication very quickly by consuming high amount of resources, causing high workload in the system. However, the study [16] also reported that although it is seldom, servers are also utilized near their maximum utilization level. In addition, the study [17] presented that overall peak-to-trough variation of CPU utilization in data center is about 30%. Thus, as a supplementary to our previous work, in this paper, we consider for the situation when the cluster is highly utilized, that means utilized beyond 50% and we set the utilization threshold as 50%. The detailed procedure to re-replicate in balanced manner among the nodes for each priority group follows our previous work [7]. We briefly present our previous work in the next section.

5.2 Our Previously Proposed Proactive Re-Replication Strategy

Our previous work [7] used predicted CPU utilization, predicted disk utilization and popularity of replicas to perform proactive re-replication as we briefly discuss in Sect. 5.2.1. The priority grouping of replicas we employed is discussed in Sect. 5.2.2.

5.2.1 Resource Utilization Prediction

Resource usage of datanodes changes frequently from time to time. Hence, in our previous work, we selected appropriate nodes for re-replication based on predicted CPU and disk utilization information at next time interval in order to avoid the occurrence of overload condition during re-replication phase.

Each datanode sends the current CPU and disk utilization to the namenode through heartbeat message periodically. When a node failure occurs, the namenode predicts future resource utilization of all datanodes in the system using local regression method. Resource utilization prediction based on local regression uses current CPU and disk utilization of each datanode along with historical values and predicts CPU and disk utilization for the next time interval. The re-replication scheduler decides how much re-replication workloads will be allocated to which nodes using predicted CPU and disk utilization in order to balance the server utilization during the re-replication phase and can avoid the occurrence of overload condition.

Table 1 Replicas' priority grouping

Group Category	Data Popularity	Replicas Remaining
G1	High	1
G2	Low	1
G3	High	2
G4	Low	2

5.2.2 Replicas' Priority Grouping

Priority based replicas grouping is generally aimed to consider both reliability of data block and performance degradation of the foreground jobs and proposes scheduling re-replication based on different priority groups. Priority is defined based on the number of current replicas available in the system and popularity of the replicas. The data blocks that need to be re-replicated are grouped into four priority groups as in Table 1.

The reason behind determining one of the attributes, remaining replicas, for priority grouping is quite clear that if there is only one remaining replica, we need to restore the lost replicas immediately in order to prevent data loss. Another attribute, popularity is mainly concerned for the system performance. Restoring popular data first can increase the system performance which is supported by the findings in [13] which showed that increase performance and load balance by recovering popular data firstly. Thus, quick restoration of popular block is important and we consider this issue in our re-replication strategy.

In order to determine whether a data block is popular or unpopular, we employ the approach in [18] to define the threshold value to decide data block's popularity. The average number of accesses $\overline{NOA}$ is used as the threshold. $\overline{NOA}$ can be calculated as $\overline{NOA} = \frac{1}{|H|}\sum_{h\in H} NOA(h)$, where $|\ H\ |$ is the number of records in H that is the access history table, and each record h in H indicates the number of accesses $NOA(h)$ for data block h.

5.3 Deferred Re-Replication Algorithm

The key insight our deferred re-replication algorithm builds on is that at the time of low resource utilization of the system, re-replication can be performed quickly. If the system is fully or highly utilized, re-replication of high priority groups is performed immediately but re-replication of lower priority groups can be deferred to a later time in order to minimize performance impacts on users' application jobs. To implement this, we simply pick an average uti-

Algorithm 1: Deferred re-replication based on average utilization

```
Require: replicaList: data blocks that are stored in failed node
Require: nodeList: datanodes with their predicted CPU and disk
         utilization
1  Divide replicaList into four priority groups;
2  while replicaList ≠ 0 do
3      Checkaverageutilization();
4      if averageutilization > 50 then
5          Perform re-replication only for groups G1 and G2;
           /* use Algorithm 2 to schedule
              re-replication for each group        */
6          wait for t sample;
7      else
8          Perform re-replication by group order;
           /* use Algorithm 2 to schedule
              re-replication for each group        */
9      end
10 end
```

Algorithm 2: Balancing resource utilization among nodes

```
1  for replica in each priority group do
2      Assign equal number of replicas in each node
3  end
4  Diff = 1;
5  while Diff <= 1 do
6      for each node in nodeList do
7          Calculate MigratingValue;
           /* MigratingValue =
              weightValue * predictedCPU + weightValue *
              predictedDisk + numberofassignedblocks   */
8      end
9      Find max and min MigratingValues;
10     Diff = max.MigratingValue − min.MigratingValue;
11     if Diff > 1 then
12         Add one replica to node of min.MigratingValue;
13         Remove one replica from node of max.MigratingValue;
14     end
15 end
16 Schedule re-replication with number of blocks assigned to each
   node;
```

lization threshold of the whole cluster to decide whether the system is highly or lightly utilized. The re-replication algorithm runs periodically at fixed epochs, and measures average resource utilization. Based on this measurement, it decides whether re-replication is performed or not. Algorithm 1 implements this functionality. Firstly, we record the lost data blocks on failed node as replicaList and record the list of datanodes with their predicted CPU and disk utilization as nodeList. The re-replication scheduling algorithm requires that replicas are categorized into priority groups before scheduling. Thus, we divide the replicas that need to be re-replicated into four priority groups based on popularity of the replicas and number of remaining replicas as we discussed in Sect. 5.2.2. When a node failure occurs, the average utilization of the whole system is checked and if the average utilization is over 50%, we perform re-replication only for high priority groups, groups G1 and G2 as in Line [4-6]. Detailed re-replication scheduling for each priority group among the nodes is carried out by Algorithm 2 which adjusts the numbers of blocks that will be assigned to each node by calculating migrating value of each node based on predicted CPU and disk utilization. Then re-replication is scheduled with number of blocks assigned to each node for each priority group. After performing re-replication for higher priority groups and deferred re-replication waits for specified period of time and check the average utilization again. If the average utilization is under 50%, re-replication for other remaining groups is performed.

5.4 Discussion

As this paper has concentrated on minimizing impacts on cluster normal jobs, decision to schedule re-replication at a certain period of time is performed based on the average utilization of the whole system which is too general to determine utilization status of each node in the system. But we argue that allocating data blocks to nodes which have high utilization variations among nodes is appropriately handled by our previous work and this work mainly focuses on providing solution for the condition when the cluster is highly or fully utilized. In addition, we employ static threshold value to determine whether the system is busy with cluster normal jobs or not. But if the system is occupied with quick variations in utilization, this may result in poor re-replication schedule. If the system keeps high utilization for some period of time, cluster normal jobs which does not have high QoS requirement should be ignored and re-replication should be performed immediately. This kind of strategies could be implemented easily just by looking at the history of utilization.

6. Evaluation

6.1 Simulation Setup

To evaluate the effectiveness of our deferred re-replication, we simulated the cloud computing infrastructure which was composed of 400 homogeneous nodes by using CloudSim [19] and CloudSimEx [20]. The reason behind creating simulation environment that consists of 400 nodes is that the work in [8] showed that there is almost guarantee to occur data loss event if the cluster scales up beyond 300 nodes.

The work in [21] designed and developed HDFS simulator to simulate HDFS environment, such as metadata management on the namenode and heartbeat monitoring mechanism to detect whether the datanode is alive or not. This simulator is event based and mainly targeted for simulation of data durability and data loss condition in HDFS in presence of node failure. As both CloudSim and that simulator are event-based and language of implementation is java, we extracted some of the functions from that simulator and integrated into CloudSim with little adaption.

In order to achieve a realistic simulated environment,

Table 2 Simulation parameters

Parameter	Value
Total Data	750000*64MB
No.of.Datanodes	400
Replication Factor	3
Disk/Ram/CPU Capacity	CPU=2660
	(MIPS= HP ProLiant ML110 G5 server)
	RAM=4GB
	Disk=1TB
Network	1Gbps

the detailed simulation configurations were set as shown in Table 2. We run the simulation for 24-hour period. A cluster with 400 datanodes was created in the simulation environment. For simplicity and ease of evaluation, at the beginning of the simulation, blocks were equally distributed on nodes in the cluster. It is assumed that one data block is the replication element and the block also represents one data file. With above simulation parameter settings, 10 simulation runs were performed. The simulation results give the mean of 10 simulation runs.

We applied synthetic web application workload, proposed by Magalhaes et al. [22]. As distributions and parameters that represent the resource utilization of a single session was presented in that work, we injected this workload into the simulator to analyze the impact of our deferred re-replication algorithm to the reliability and performance of the system. The dynamic arrival pattern of sessions was simulated based on the data provided by the work in [20]. The workload data was obtained from small web site and applied poisson distribution for session arrival.

In addition, in order to simulate as close as possible to realistic environment, we used the following distributions and parameters:

- Zipf distribution to mimic the file access pattern [23].
- Exponential distribution to determine failure's insertion time to the system.
- 1% of node failures in the system except the experiment that is tested for effect of varying percentage of node failures in the system because recent research [24] measured the failure percentage of the cluster and showed the failure percentage was about 0.5% to 1%.

For priority grouping, we did not fix the number of data blocks belonging to G1 and G2 in the experiments. The data blocks are assigned to the group based on two attributes, namely, popularity and the number of remaining replicas. To determine popularity, we applied Zipf distribution with scale factor 0.8 which is realistic according to the study in [23]. Thus, around 10% of the total simulated data blocks is popular data. However, we found that the second attribute, the number of remaining replicas, depends largely on percentage of failures. It means that the more percentage of failure, the more data blocks will be under this attribute. For example, under 1% of failure, data blocks are rarely seen in groups G1 and G2 but under 10% of failure, about 20% of data blocks that need to be re-replicated are in groups G1 and G2 which will be re-replicated immediately to ensure the reliability of the system.

In all experiments, we compared our deferred re-replication algorithm with both baseline HDFS and Balance that is our previous re-replication strategy [7] which tried to balance resource utilization during re-replication stage. Thus, firstly, we implemented the default baseline HDFS re-replication policy with the following configuration parameters:

(a) Three replicas were allocated with rack aware replica placement policy.

(b) The source and destination datanodes were chosen randomly at the time of re-replication.

(c) The HDFS configuration parameter to control speed of re-replication, dfs.namenode.replication.work.multiplier.per.iteration, was set to its default value, 2. It means that the number of blocks that will be re-replicated at every scheduling interval is 2*number of nodes in the system.

(d) Re-replicate the data blocks first that remains only one block.

6.2 Results

The following sections present results to comprehend the impact of deferred re-replication on the system. We are mainly interested in two aspects: performance impacts of re-replication jobs on the cluster normal jobs and the capacity to assure durability of data blocks.

6.2.1 Performance Impacts on the Users' Jobs

To asses the performance impacts of re-replication jobs on the users' jobs, we define the following metrics.

Slow down: This metric represents performance degradation of users' jobs because of the competition for resources with re-replication jobs. It is calculated by the following equation.

$$slow\ down\ of\ a\ job = \frac{running\ time\ with\ failure}{running\ time\ without\ failure}$$

Completion rate: This metric represents the rate that the tasks are completed by their deadline. It can be calculated by the following equation.

$$completion\ rate = \frac{no.of.jobs\ completed\ by\ deadline}{no.of.submitted\ jobs}$$

Slow Down: In this experiment, in order to investigate the performance impacts on the users' jobs by re-replication jobs, we allocated user job to each node in the system each time node failure occurs and then we examine average slow down of the job. In Fig. 3, we observe that deferred re-replication can reduce performance impacts on users' jobs

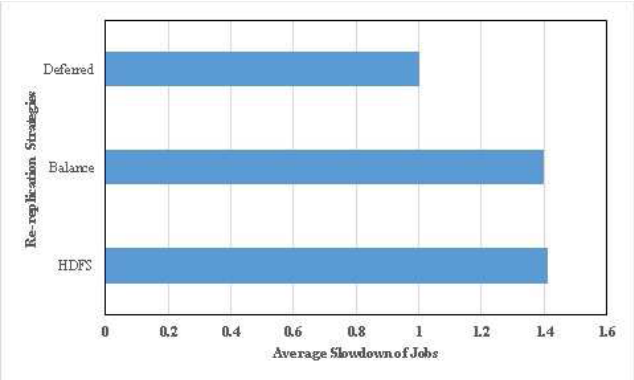

Fig. 3 Slowdown of jobs.

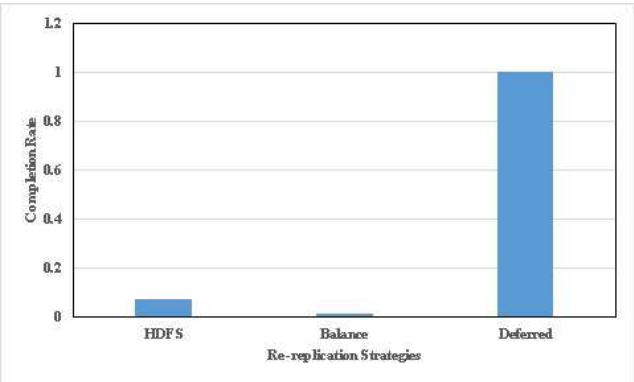

Fig. 4 Completion rate.

by reducing slowdown of jobs according to the equation mentioned above. This is consistent with our expectations. It is because users' jobs do not need to compete with the re-replication jobs for resources.

Completion Rate: In order to satisfy the service level agreement (SLA) imposed on each job, in this experiment, we determine whether our deferred re-replication algorithm can satisfy SLA by ensuring that tasks are completed by their deadlines. We simply set task deadline rounded up to 20% more than their minimum execution time. Figure 4 shows the completion rate the users' jobs submitted during re-replication time. As we can see clearly from this figure, completion rate of users' jobs using deferred re-replication is higher than the other two re-replication schemes. This is because deferred re-replication scheme shifted certain portion of re-replication to a later time based on the average utilization of the whole cluster.

As shown in Figs. 3 and 4, HDFS and Balance each had advantage and disadvantage for the two metrics, while our proposal outperformed them in terms of both of the two metrics.

Table 3 shows the average completion time for all re-replication strategies. In the experiments, we observe that Balance can perform re-replication quickly because it consumed system resources in a greedy manner. The completion time of deferred re-replication largely depends on the

Table 3 Average Completion Time of Re-replication

Re-replication Strategies	Completion Time(Sec)
HDFS	215.4
Balance	34.5
Deferred	7557.6

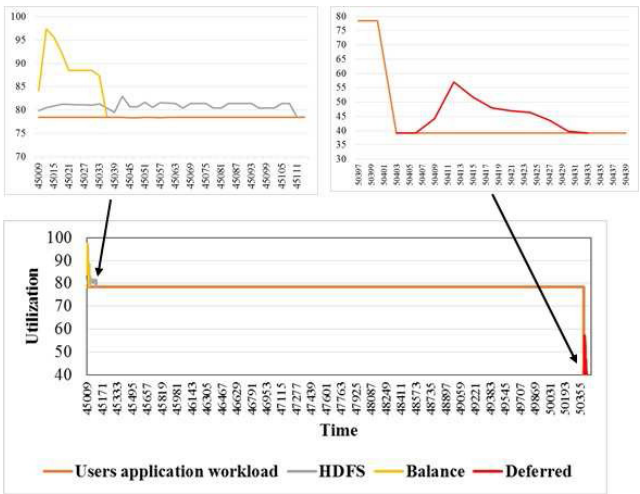

Fig. 5 Utilization of the whole system during re-replication.

resource utilization status of the system. If the system is highly utilized, re-replication will be deferred to later time, resulting in longer completion time. If the system is lightly utilized, completion time of deferred re-replication will be comparable to Balance re-replication. But the experiments in this paper were evaluated only for high utilized condition.

Figure 5 shows the average utilization change of the whole system during re-replication process. In order to see the utilization change during re-replication, we injected node failures at time 45002 that the system was highly utilized with average utilization around 78%. As shown in the figure, although the system was highly utilized, Balance re-replication started immediately by consuming high amount of resources with its maximum value to 97% within short time frame (from 45009 to 45039) and performed the re-replication. As average utilization of the system exceeded the threshold value, deferred re-replication did not re-replicate immediately and checked the resource utilization continuously. The cluster was highly utilized until time frame 50400. At time 50400, system utilization dropped from 78% to 40%. At that time, as system utilization was under 50%, deferred re-replication started the replication process and re-replication was finished at 50433. For HDFS, it did not consume resource utilization as Balance and it took longer time than Balance with resource utilization around 81%.

6.2.2 Reliability

In order to measure the effects of deferred re-replication on the reliability level of the system, we carried out a set of experiments.

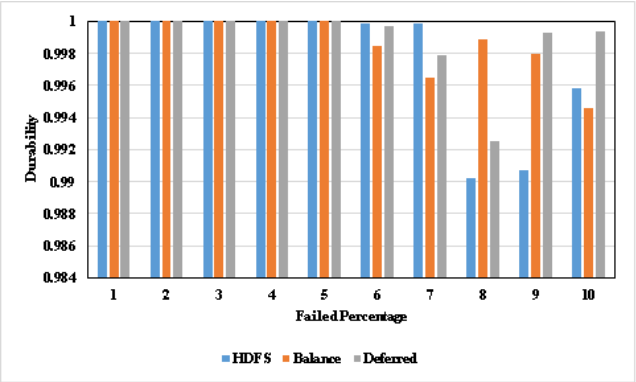

Fig. 6 Durability.

Varying percentage of failure: In this experiment, we vary the failed percentage from 1% to 10%.Although only 1% of node fails in the production cluster [24], we examine the condition for extreme failure case by setting failure percentage of node until 10%. We fix the number of datanodes and number of data blocks on each node. Figure 6 shows the results of our experiment. We surprisingly found that under failure percent from 1 to 5, there is no data loss for all three re-replication schemes. Starting from 6% of node failure, there was data loss in the system. As we can see from Fig. 6, it is difficult to describe that which re-replication strategy is better than other ones because all of three replicas may be lost due to the failure injected to three nodes that holds three replicas. But we observe that deferred re-replication can provide same reliability level as baseline HDFS under failure percentage of 1% to 5% which is realistic failure condition for production clusters.

Varying number of nodes: As the number of datanodes increases, the aggregate rate of datanodes failures would also increase. If too many datanodes fail, the probability of data loss would be high. Thus, we investigate whether deferred re-replication can provide the same reliability level as baseline HDFS and Balance re-replication scheme for varying number of nodes as 200, 400, 600, 800, and 1000 nodes. Here, we set the fail percentage to 1%. We found that both three re-replication scheme does not lose data under these settings and can provide full durability.

Varying Mean Time Between Failure (MTBF): Although we set MTBF(Mean Time between Failure) as 7 hours in our previous experiment, in order to investigate the reliability capability of deferred re-replication, we change the values of MTBF from 1 hour to 5 hours and we injected 1% of data nodes failure [24] to the system. Then, we measured the durability of data blocks for each re-replication strategy. We found that there was no occurrence of data loss event for all re-replication strategies so that full durability can be provided. This is consistent with what we expected for Balance and baseline HDFS. We found that deferred re-replication can also provide full durability although its data blocks restoration process took longer time than baseline HDFS and Balance. This is because deferred re-replication performed immediate re-replication for critical data blocks which remain only the last copy without deferring them to later time.

Varying number of data blocks: In this experiment, we fix the number of fail percent to 1% and the number of nodes to 400 nodes. And then we change the number of blocks hold by each datanode in the system to 100G, 200G, 300G, 400G and 500G. Although we increase the amount of blocks stored in each node, we found that there is no data loss and the system can provide full durability for the data blocks for all three re-replication methods. From this experiment, we report that deferring re-replication to later time can get the same reliability level as baseline HDFS and Balance re-replication approach. From all these experiments for reliability, we confirm that deferred re-replication can provide the same reliability level as other schemes.

7. Conclusion

We presented our deferred re-replication scheme that can provide reducing performance impacts on users' application jobs without sacrificing reliability guarantees. The current work was based on our previous work which achieved servers' utilization balance during re-replication phase. It was extended to take into consideration time varying workloads and triggered re-replication on the average utilization of the whole system. This simple extension enables us to reduce impact on cluster normal jobs by the re-replication jobs while ensuring reliability. Our results indicate that deferred re-replication achieved significant minimization of performance impacts on the cluster jobs while ensuring the same reliability level as the baseline. We conclude that deferred re-replication based on resource usage of the whole cluster can be an effective way of reducing impacts on cluster normal jobs which have high QoS requirement. In the future work, we plan to consider for controlling re-replication rate based on continuous arrival of workloads at different times. Additionally, it is oriented towards consideration of node heterogeneity with different performance values and thermal balancing which is important metric for green cloud computing.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," Proceedings of the 19th ACM Symposim on Operating Systems Principles, 2003.
[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies, pp.1–10, 2010.
[3] P. Bohrer, E.N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, Power aware computing, Kluwer Academic Publishers Norwell, MA, USA, 2002.
[4] A. Abraham and V. Ramos, "Web usage mining using artificial ant colony clustering and linear genetic programming," Proceedings of the 2003 Congress on Evolutionary Computation, pp.1384–1391, 2003.
[5] A. Higai, A. Takefusa, H. Nakada, and M. Oguchi, "A study of effective replica reconstruction schemes at node deletion for hdfs," Pro-

ceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp.512–521, 2014.
[6] A. Higai, A. Takefusa, H. Nakada, and M. Oguchi, "A study of replica reconstruction schemes for multi-rack hdfs clusters," Proceedings of the IEEE/ACM 7th International Conference on Utility and Cloud Computing, pp.196–203, 2014.
[7] T. Shwe and M. Aritsugi, "Proactive re-replication strategy in hdfs based cloud data center," Proceedings of the 10th IEEE/ACM International Conference on Utility and Cloud Computing, pp.121–130, 2017.
[8] A. Cidon, S.M. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum, "Copysets:reducing the frequency of data loss in cloud storage," Proceedings of the 2013 USENIX Annual Technical Conference, pp.37–48, 2013.
[9] A. Cidon, R. Escriva, S. Katti, M. Rosenblum, and E. Sirer, "Tiered replication: a cost-effective alternative to full cluster geo-replication," Proceedings of the 2015 USENIX Annual Technical Conference, pp.31–43, 2015.
[10] J. Wang, H. Wub, and R. Wand, "A new reliability model in replication based big data storage systems," Parallel and Distributed Computing, vol.108, pp.14–27, 2017.
[11] W. Li, Y. Yang, J. Chen, and D. Yuan, "A cost-effective mechanism for cloud data reliability management based on proactive replica checking," Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp.564–571, 2012.
[12] S. Wu, H. Jiang, and B. Mao, "Proactive data migration for improved storage availability in large-scale data centers," IEEE Trans. Comput., vol.64, no.9, pp.2637–2651, 2015.
[13] S. Wu, W. Zhu, B. Mao, and K.-C. Li, "PP: popularity-based proactive data recovery for hdfs raid systems," Future Generation Computer Systems, vol.86, pp.1146–1153, 2018.
[14] T. White, Hadoop: The Definitive Guide, 3rd. ed., O' Reilly Media, Inc, 2012.
[15] D. Meisner, B.T. Gold, and T.F. Wenisch, "Powernap: eliminating server idle power," Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems, 2009.
[16] L.A. Barroso and U. Hölzle, "The case for energy-proportional computing," Computer, vol.40, no.12, pp.33–37, 2007.
[17] C. Kilcioglu, J.M. Rao, A. Kannan, and R.P. McAfee, "Usage patterns and the economics of the public cloud," Proceedings of the 26 th International Conference on World Wide Web, pp.83–91, 2017.
[18] M. Tang, B.-S. Lee, X. Tang, and C.-K. Yeo, "The impact of data replication on job scheduling performance in the data grid," Future Generation Computer Systems, vol.22, no.3, pp.254–268, 2006.
[19] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F.D. Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software-Practice and Experience, vol.41, no.1, pp.23–50, 2011.
[20] N. Grozev and R. Buyya, "Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments," The Computer Journal, vol.58, no.1, pp.1–22, 2015.
[21] C. Debians, P.A.T. Togores, and F. Karakusoglu, "Hdfs replication simulator," https://github/peteratt/HDFS-Replication-Simulator, 2012.
[22] D. Magalhaes, R.N. Calheiros, R. Buyya, and D.G. Gomes, "Workload modeling for resource usage analysis and simulation in cloud computing," Journal of Computers and Electrical Engineering, vol.47, no.C, pp.69–81, 2015.
[23] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: evidence and implications," Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies, 1999.
[24] R.J. Chansler, "Data availability and durability with the hadoop distributed file system," The USENIX Magazine, vol.37, no.1, pp.16–22, 2012.

Thanda Shwe received her B.E. and M.E. degrees in Information Technology from Mandalay Technological University, Myanmar in 2004 and Yangon Technological University, Myanmar in 2006, respectively. From 2009 to 2014, she was with the Department of Information Technology, Yangon Technological University, Myanmar. She is now Ph.D. candidate at Kumamoto University, Japan.

Masayoshi Aritsugi received his B.E. and D.E. degrees in computer science and communication engineering from Kyushu University, Japan, in 1991 and 1996, respectively. From 1996 to 2007, he was with the Department of Computer Science, Gunma University, Japan. Since 2007, he has been a Professor at Kumamoto University, Japan. His research interests include database systems and parallel/distributed data processing. He is a senior member of IPSJ, and a member of ACM, IEEE, and DBSJ.