Performance under Failures of MapReduce Applications

Hui Jin*, Kan Qiao*, Xian-He Sun*, Ying Li*†

*Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois, USA
{hjin6, kqiao, sun}@iit.edu

†College of Computer Science
Communication University of China
Beijing, China
liy@cuc.edu.cn

***Abstract*—The MapReduce programming paradigm is gaining more and more popularity in recent years due to its ability in supporting easy programming, data distribution, as well as fault tolerance. Failure is an unwanted but inevitable fact that all large-scale parallel computing systems have to face with. MapReduce introduces a novel data replication and task re-execution strategy for fault tolerance. This study intends to lead a better understanding of such fault tolerance mechanisms. In particular, we build a stochastic performance model to quantify the impact of failures on MapReduce applications and to investigate its effectiveness under different computing environments. Simulations also have been carried out to verify the accuracy of the proposed model. Our results show that data replication is an effective approach even when failure rate is high, and the task migration mechanism of MapReduce works well in balancing the reliability difference among individual nodes. This work provides a theoretical foundation for optimizing large-scale MapReduce applications, especially when fault tolerance is the concern.**

I. Introduction

MapReduce [1] is a newly emerged programming model originated from large data processing. MapReduce is gaining more and more popularity in recent years due to its merit of easy programming, data distribution and fault tolerance. MapReduce has several advantages over other programming paradigms such as MPI by nature, due to the fact that the map/reduce tasks are independent of each other, which isolates the impact of failures to the single task. Furthermore, MapReduce adopts multiple replicas of each data block and re-executes the failed tasks in the observation of failures, which potentially avoids the data transfer and checkpointing overhead of traditional MPI applications.

The presence of failures is inevitable for large-scale parallel applications. The performance analysis and optimization under failures of traditional HPC applications such as MPI is currently an active research area and has been well studied in the literature. However, little work has been done to investigate the impact of failures on MapReduce applications. Failures are more prevailed for MapReduce applications since it is targeted at commodity hardware based clusters. The problem will be exaggerated when running distributed MapReduce applications under volunteer computing environment, where the interruptions from the host applications are norm to suspend the MapReduce applications [2].

The performance modeling of MapReduce applications with the presence of failures is the key to performance analysis and optimization. There are several challenges for the prediction of the MapReduce application performance. The root cause is the dynamic feature of MapReduce execution. For example, the rescheduling of the failed or slow tasks is decided by the scheduler at runtime with the consideration of data locality.

In this paper, we propose a stochastic model to predict the performance of MapReduce applications under failures. Simulations have been carried out to validate the accuracy of the model. This work can be used to quantify the robustness of MapReduce applications under different system parameters, i.e., the number of processes, the Mean Time Between Failures (MTBF) of each process, failure recovery cost, etc.

The rest of this paper is organized as follows. Section II presents the performance model to predict the MapReduce performance. Simulations are conducted in section III to verify the accuracy of the model. We conclude this study in section IV.

II. Performance Modeling under Failures

A. Assumptions

The MapReduce applications are composed of two phases, map and reduce. There is a barrier between the two phases, which implies that reduce will not start until all the map tasks are completed. There is no synchronization inside each phase. We will model the performance of map phase in this work, which also applies to the reduce phase and should be easy to be extended for the entire MapReduce application.

We assume the inter-arrival times of failures for each process (node) i $(0 \leq i \leq n-1)$ are independent and identically distributed (*iid*) as exponential random variables with parameter λ_i, which is the inverse of MTBF.

The service time of each failure follows a general distribution with mean μ and standard deviation σ. The failures are handled on *First-Come-First-Serve* (FCFS) basis when they are overlapped.

B. MapReduce Performance under Failures

1) Single Task Performance: We term the length of one map/reduce task as γ, which is equivalent to the failure-free execution time. If a failure happens at time $t < \gamma$, an interruption occurs and the task will be re-executed after the failure recovery, which initializes another attempt to finish the task. Such attempts to a successful task execution iterate till the termination condition is met.

Let random variable T denote the time to finish a task of length γ, based on queueing theory, we have

$$E(T) = (e^{\gamma\lambda} - 1)(\frac{1}{\lambda} + \frac{\mu}{1-\mu\lambda}) \qquad (1)$$

Detailed derivation of formula 1 can be found in [3].

2) Worst Performance: In the case of worst performance, the failed tasks will be re-executed on the same node after the failure is recovered. There is no migration and rebalancing among the processes. The MapReduce application will not be completed till the least reliable node finishes all its assigned map tasks. The problem of finding the worst MapReduce performance is equivalent to identify the node with maximum segment execution time.

We list the algorithm to get the worst performance of MapReduce applications as in Algorithm 1.

Algorithm 1 MapReduce Worst Performance

```
$maxE = 0$ //Maximum expected task execution time.
for each node $n_i$ do
    $E_i(T) = (e^{\gamma\lambda_i} - 1)(\frac{1}{\lambda_i} + \frac{\mu_i}{1-\mu_i\lambda_i})$
    if $E_i(T) > maxE$ then
        $maxE = E_i(T)$
    end if
end for //find the maximum expected execution time.
return $E(\mathcal{T}_{mr}^{worst}) = maxE \times k$ // k is the number of tasks assigned to each node.
```

3) Best Performance: In the case of best performance, the failed tasks have data replicas on other healthy nodes and can be re-executed without additional data migration.

Let $U_i(t) = \lfloor \frac{t}{E_i(T)} \rfloor$ denote the expected number of map tasks complicated by node i in a time span of t.

The execution time of a MapReduce application is essentially the minimum t that satisfies,

$$\sum_{i=0}^{n-1} U_i(t) \geq n \times k \qquad (2)$$

We first iterate all the nodes and identify the minimum and maximum expected segment execution time, $minE$ and $maxE$, respectively. The best MapReduce performance must fall within the scope of $[minE \times k, maxE \times k]$, which is used as the initial scope for a binary search. In each iteration of the binary search, we calculate the number of tasks that can be completed at time $t = (minE + maxE)/2$. The search terminates till the search scope is small enough.

III. Simulation Results

A. Simulation Setup

The simulation is based on the synthetic data generated from the assumed distributions. Failure arrivals of each node are generated with the exponential distribution. Lognormal distribution is a good approximation for failure recovery and is used to generate the failure recovery log [4].

We have implemented a MapReduce simulator to emulate the execution of MapReduce applications. The MapReduce scheduler was implemented to mimic Hadoop, an open source MapReduce implementation [5].

To measure the performance of MapReduce task migration and evaluate the accuracy of the model, we have set up seven environments (A-G). Table I summarizes the statistics on the nodes of each environment.

TABLE I: Environments and the Statistics

Environment	Mean MTBF	CoV of MTBF	Mean of μ	CoV of μ
A	512 Hours	0.5	0.2 Hour	0.5
B	8192 Hours	0.5	0.2 Hour	0.5
C	512 Hours	1	0.2 Hour	0.5
D	512 Hours	0.5	2 Hours	0.5
E	512 Hours	0.5	2 Hours	1
F	512 Hours	1	2 Hours	0.5
G	512 Hours	1	2 Hours	1

B. Experimental Results

Figure 1 verifies the MapReduce performance model applications with 16384 processes. The model presents satisfactory accuracy for all the environments. All the simulated execution times fall within the predicted best and worst performance. Most simulated performance is close to the predicted best performance, which implies that the task migration works well to balance the reliability difference among individual nodes. We also observe that MapReduce with 3 replicas presents better performance than that of 1 replica, this is because the higher number of replicas reduce the chances of data migration due to failures.

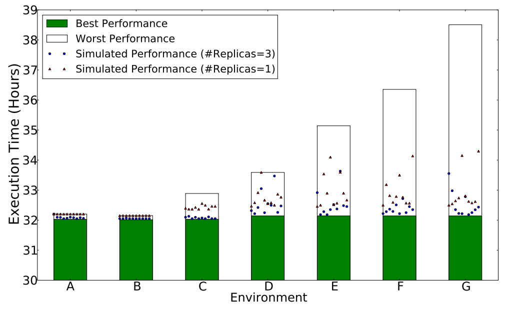

Fig. 1: Model Verification (# Processes=16384)

IV. Conclusions

We have derived a stochastic model to predict the performance of MapReduce applications under failures. The simulation results confirm the accuracy of the model. We also observe that the task migration mechanism of MapReduce works well to mitigate the impact of failures.

In the future, we plan to investigate further in performance analysis and optimization, along the line of performance comparison of MapReduce and its MPI counterparts, determining the optimal number of replicas and block size, etc. We are also interested in evaluating the performance of MapReduce under volunteer distributed computing environment.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[2] H. Lin, X. Ma, J. S. Archuleta, W. chun Feng, M. K. Gardner, and Z. Zhang, "MOON: MapReduce On Opportunistic eNvironments," in *HPDC 2010*.
[3] H. Jin, Y. Chen, H. Zhu, and X.-H. Sun, "Optimizing HPC Fault-tolerant Environment: An Analytical Approach," in *ICPP 2010*.
[4] B. Schroeder and G. A. Gibson, "A Large-scale Study of Failures in High-Performance Computing Systems," in *DSN 2006*.
[5] "The Hadoop Project Website," *http://hadoop.apache.org/*.