A Fault Tolerant Scheduling Model for Directed Acyclic Graphs in Cloud

Pedro Henrique Di Francia Rosso[1], **Emilio Francesquini**[1]

[1]Centro de Matemática, Computação e Cognição (CMCC)
Universidade Federal do ABC (UFABC)

pedro.rosso@ufabc.edu.br, e.francesquini@ufabc.edu.br

Abstract. *Many High Performance Computing (HPC) and resource intensive applications have been tested and migrated to the Cloud. These applications may have high data input size, which often has a high correlation to execution performance and time. Migration to the Cloud demands adaptation of the fault tolerance (FT) and scheduling approaches. Although those topics are well connected, they are often treated separately. This work proposes a novel integrated scheduling and FT model which takes into account the characteristics of the tasks and the target execution nodes. Preliminary results indicate good potential to improve system reliability and execution makespan of scientific workflows.*

1. Introduction

High Performance Computing (HPC) applications, like Seismic Imaging (geophysics), often have special characteristics, *e.g.*, high resource usage (memory, CPU, etc.). Typically, these applications run in large clusters and dedicated environments, and recently, in Cloud. Failures are one of the problems taken into account in those scenarios, which can be increased when a large number of nodes is employed [2].

Cloud Computing provides a set of shared resources of easy use and management, and advantageous features such as elasticity and on-demand services [7], and is widely employed nowadays. These environments present several challenges which include data migration, cost of replications and reliability [5], and performance. And, like HPC Clusters, Cloud environments are susceptible to failures, and their occurrences tend to become more probable when one considers the total time spent by cloud-based HPC applications. Among others, these reasons make building a fault-tolerant framework, with tools that can predict or detect faults and perform procedures to repair them, more difficult. Moreover, scheduling jobs in Cloud can be also a challenge due to its heterogeneity.

This work proposes a model with the objective of reducing the makespan of scientific applications focusing on how failures impact and how FT can help. This model does so by mixing a diverse set of FT techniques and by choosing these techniques based on the tasks and on the nodes' characteristics.

2. Background and Related Work

There are two major groups of approaches for FT: Reactive and Proactive. The former aims to handle faults after they occur, the latter aims on anticipating faults. Checkpointing, replication and task migration are examples of reactive while self-healing, system rejuvenation and preemptive migration, examples of proactive methods [3].

Some works relate FT and scheduling. A recent survey [6] gathered works that take into account FT as an objective for scheduling. Articles were divided between proactive and reactive approaches. Proactive methods include historical logs, statistical failure prediction, and reliability as a criterion to heuristic, meta-heuristic and classic optimization algorithms. Reactive proposals are categorized into checkpoint (full, coordinated, uncoordinated, incremental, and user level) and replication (active, passive, and primary).

Our proposal employs both reactive proposal categories together, *i.e.*, checkpoint and replication, but doesn't specify which checkpointing method should be used (although user-level and incremental might display better results). We propose the use of an active non-synchronized replication and the determination of the FT approach based on each task characteristics (a single task could employ one or both approaches). In contrast, related works often choose the approach based on machine availability only [1]. Future works include the use of proactive FT methods using, for instance, cloud node monitoring (*e.g.* hardware sensors and network metrics) as components of a reliability coefficient which will be used by the scheduler. Finally, there are some works that do not employ FT, but instead consider how faults and reliability impact Quality of Service (QoS) parameters [4].

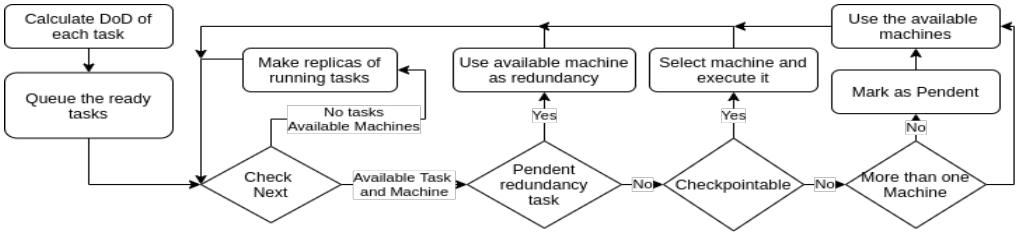
3. Modeling

This work targets heterogeneous (with CPU-GPU nodes) cloud based nodes with a computational power, represented by a computing coefficient, executing an application represented by a set of tasks described as a Directed Acyclic Graph (DAG). Additionally, we assume that each task's input size and the total fixed number of computing nodes is known, and tasks execution times are unknown[1].

The FT model is based on three basic reactive approaches: checkpointing, replication and task migration, with the lemma: "*Checkpointing when viable, replication whenever possible and migration if needed*". The scheduling model is based on a priority queue, in which the priority consists of the Degree of Dependency (DoD) of each task. DoD is given by cumulative number of direct and indirect dependents of a task. At each cycle, the scheduler peeks the head of the queue, and evaluates the task based on FT factors. Task's data input size determines if the task will use checkpoint or redundancy when compared to a predetermined threshold, that can be either user defined or calculated by a statistical method. If a task is deemed not checkpointable, the scheduler will try to create redundancy using replication across multiple machines. If no additional machine is available, it will execute the task on a single machine but tag it as "pending redundancy". As soon as a machine becomes available, tagged tasks are replicated. As a way to improve FT, if there are available machines but no tasks left execute, the scheduler replicates already running tasks. This approach also reduces the overhead for running checkpointable tasks since only one of the replicas will actively be performing checkpoints. Future works include system monitoring, which will affect the machines that will be selected to run the task, specially in the case where a task should be replicated but there is only one machine available. We also plan to evaluate the use of cloud spot machines for the improvement of tasks redundancy. Figure 1 shows the basic flowchart of the model.

[1]To assess our proposal, we developed a simulator that includes all the parameters above as well as several others (checkpoint threshold, number of redundancies, etc.). The source code of the simulator is available at: https://github.com/PedrooHR/FTScheduler-Sim/tree/ERAD2020.

Figure 1. Flowchart of the scheduler

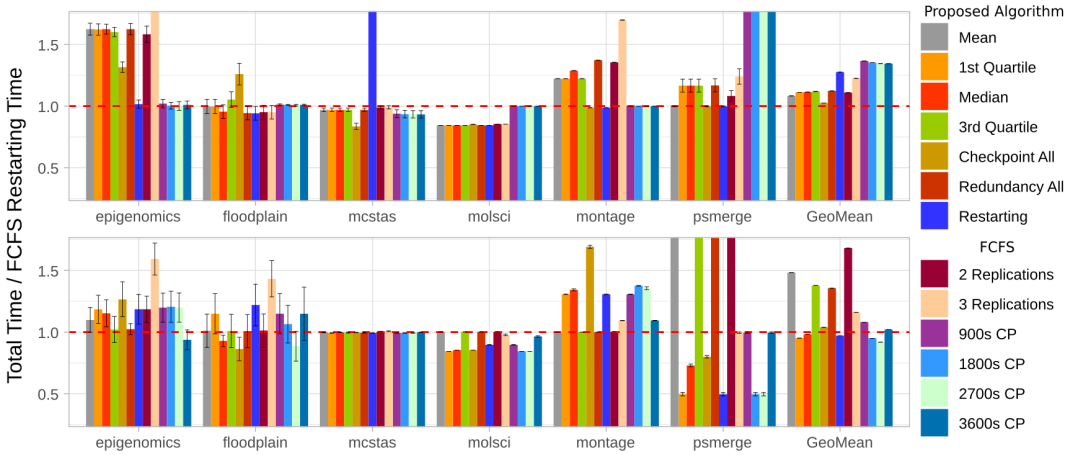


4. Experimental Results

The evaluation of our approach was done using simulation. Where failures are modeled after a Weibull distribution (modeled by a mean time between failures of 10 hours) [8], when a machine fails, another is provided. To determine the basic checkpoint interval we look at input data size, where $interval = \max\{600, TaskTime/(InputSize/40)\}$ seconds, where input size is given in MB, and 40 is the throughput of general purpose HDD of Amazon in MB/s AWS[2].

As input DAGs, we used the standard Pegasus Workflows[3] which are a variety of real application workflows with different characteristics. Experimental results were done using ten random scenarios (pairs of faults and machines) using different FT approaches. Figure 2 shows the relation between the makespan of each approach with FCFS Restarting model (restart from beggining) as baseline. First plot shows the results for the original workflows, while second plot shows the results for the workflows scaled by ten times (size and time). Last column of each plot shows the geometric mean of the workflows for each parameter. Error bars represent data confidence intervals.

Figure 2. Results for Original Workflows (top) and Scaled Workflows (bottom)



Plots shows mixed results for our proposal (ranging from best to worst) on execution performance for the proposed algorithm. This occurs primarily because of the variety of the characteristics in the workflows (for instance, *montage* has 11s and 13MB average task time and size, while *psmerge* have almost 3h and 4TB). When analysing the proposed algorithm performance, it is clear that different FT approaches yield different results. In general, the simplest approach of restart on failure provides better results because

[2]https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html
[3]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.

some workflows are so short that the probability of failures is very low. To evaluate our approach on a scenario in which tasks last longer (which would be the case in HPC scenarios), we scaled the execution times and data usage. Results can be seen in the second plot on Figure 2. As tasks take longer to execute, the probability of a failure occursduring execution is also higher. Additionally, our approach assumes there is a high correlation between data usage and execution time, however only `psmerge` has this characteristic.

5. Conclusions

Scheduling and Fault Tolerance are two areas of paramount importance for HPC and Cloud Computing. The definition of a single scheduling algorithm might not be desirable since it might hinder an efficient fault tolerance approach (as constraints and objectives can drastically change according to the job in execution). This work presents a novel algorithm that, although still in development, takes into account scheduling and FT at the same time using a mix of FT techniques which are chosen based on each task's characteristics.

Simulations show that choosing the FT method based on the tasks can impact makespan, and that the choices are influenced by each workflow characteristics. Future works include the addition of failure probabilities as a choice factor in the scheduler, improved load balancing and task placement optimization to reduce data transfers, evaluate DAG metrics (e.g: diameter, communication ratio, etc.) to build correlations between workflows and FT approaches, and evaluate the cost of regular and spot machines.

Acknowledgments

The authors are grateful to the Center of Petroleum Studies (CEPETRO-Unicamp/Brazil) and PETROBRAS S/A for the support to this work as part of BRCloud Project.

References

[1] Mohammed Amoon. Adaptive framework for reliable cloud computing environment. *IEEE Access*, 4:9469–9478, 2016.

[2] James Elliott, Kishor Kharbas, David Fiala, Frank Mueller, Kurt Ferreira, and Christian Engelmann. Combining partial redundancy and checkpointing for HPC. In *32nd Intl. Conference on Distributed Computing Systems*, pages 615–626. IEEE, 2012.

[3] Moin Hasan and Major Singh Goraya. Fault tolerance in cloud computing environment: A systematic survey. *Computers in Industry*, 99:156–172, 2018.

[4] Vahideh Hayyolalam and Ali Asghar Pourhaji Kazem. A systematic literature review on qos-aware service composition and selection in cloud environment. *Journal of Network and Computer Applications*, 110:52–74, 2018.

[5] Yashpalsinh Jadeja and Kirit Modi. Cloud computing-concepts, architecture and challenges. In *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, pages 877–880. IEEE, 2012.

[6] Chesta Kathpal and Ritu Garg. Survey on fault-tolerance-aware scheduling in cloud computing. In *Information and Communication Technology for Competitive Strategies*, pages 275–283. Springer, 2019.

[7] Peter Mell, Tim Grance, et al. The NIST definition of cloud computing. 2011.

[8] Devesh Tiwari, Saurabh Gupta, and Sudharshan S Vazhkudai. Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 25–36. IEEE, 2014.