Future Generation Computer Systems 29 (2013) 1808–1815

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Performance comparison under failures of MPI and MapReduce: An analytical approach

Hui Jin [a,*], Xian-He Sun [b]

[a] Parallel Query Group, Oracle, USA

[b] Department of Computer Science, Illinois Institute of Technology, USA

HIGHLIGHTS

- Analytical models are proposed to quantify the impact of failures.
- A numerical study is conducted on both MPI and MapReduce applications under failures.
- The impact of different parameters on the failure-prone performance is investigated.
- Extensive experiments are carried out to examine the accuracy of the proposed models.

ARTICLE INFO

Article history:
Received 1 September 2012
Received in revised form
22 January 2013
Accepted 30 January 2013
Available online 6 March 2013

Keywords:
Fault tolerance
MPI
MapReduce
Checkpoint

ABSTRACT

MPI has been the *de facto* standard of parallel programming for decades. There has been an increasing concern about the reliability of MPI applications in recent years, partially due to the inefficiency of parallel checkpointing. MapReduce is a new programming model originally introduced to handle massive data processing. There are numerous efforts recently that transform classical MPI based scientific applications to MapReduce, due to the merits of easy programming, automatic parallelism, and fault tolerance of MapReduce. However, the stricter synchronization primitive supported by MapReduce also imposes considerable overhead.

While the failure-free performance comparison between MPI and MapReduce has been investigated, there exists little work in comparing the two programming models under failures. In this paper, we propose an analytical approach to quantifying the capabilities of the two programming models to tolerate failures for a comparison. We also carry out extensive numerical analysis to study the impact of different parameters on fault tolerance. This work can be used by the HPC community for various purposes in making critical decisions. For example, it helps algorithm designers to answer the question such as, at which scale should we give up MPI and use MapReduce as the programming model for a better performance under the presence of failures?

1. Introduction

MPI [1] has been the *de facto* standard for parallel programming for decades. MPI utilizes a rich set of communication and synchronization primitives and supports classical scientific applications. While HPC is evolving towards exascale and continues to grow in the scale, a significant challenge facing MPI is its support for fault tolerance. Due to the dependencies among the processes, a failure to single process will soon be propagated to other processes and leads to the collapse of the entire application. Checkpointing is currently the state-of-the-art solution to support MPI fault tolerance. However, its applicability is challenging at extreme scale due to its excessive disk access and limited scalability [2,3].

MapReduce [4] is a newly emerged programming model originated from massive data processing, which has gained much praise from the community due to its ease of programming, automatic parallelism and fault tolerance. Numerous efforts have been made recently to introduce MapReduce as the programming model for scientific computing [5–13]. While the failure-free performance comparison between MPI and MapReduce has been investigated, the performance comparison of the two programming models under failures has been rarely studied. This paper aims to perform a quantitative comparison between the two programming models and answer the questions as follows,

- Given a set of system parameters (i.e., the number of processes, Mean Time Between Failures (MTBF), and failure recovery cost), which programming paradigm should be chosen for a

* Corresponding author. Tel.: +1 650 506 1917.
E-mail addresses: hui.x.jin@oracle.com (H. Jin), sun@iit.edu (X.-H. Sun).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.01.013

better performance? At which scale should we give up MPI and use MapReduce as the programming model for a better performance under the presence of failures?
- When transforming an MPI program to its MapReduce counterpart, how to control the extra overhead such that MapReduce can lead to a performance that is competitive to MPI under failures?
- As a benefit of fault tolerance, MapReduce can provide performance comparable to MPI even with the commodity hardware. How do we make the decision of hardware acquisitions balancing between performance, reliability and cost?

The contribution of this research is three-fold,

- We propose queuing models to predict the performance of both MPI and MapReduce with the consideration of failures, the underlying programming structure and corresponding fault tolerance mechanisms.
- To show how the proposed models can be used for quantitative performance comparison, we present numerical analysis and case studies to answer the questions listed above. The impacts of different parameters on the selection of programming models are also analyzed.
- Extensive simulations and experiments have been conducted based on both synthetic data and failure traces from production systems to verify the accuracy of the proposed performance model and understand the performance implications.

The rest of this paper is organized as follows. Section 2 reviews the results of existing related work and introduces the background of MPI and MapReduce from the perspective of programming structure and the failure handling mechanisms. Section 3 presents the performance model to predict the performance of MPI and MapReduce applications. In Section 4, we carry out quantitative analysis to study the impact of different parameters and how the models can be utilized to answer the aforementioned questions. Simulations and experiments are conducted in Section 5 to verify the correctness of the model. We conclude this study in Section 6.

2. Background

2.1. MapReduce for scientific computing

MapReduce is a framework for processing large scale datasets. It mainly consists of two steps, *Map* and *Reduce* [4]. Users specify the Map and Reduce functions and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines.

To benefit scientific computing with the MapReduce programming model, numerous efforts have been elaborated to adapt scientific computing modules into MapReduce. In [5], Tu et al. proposed to implement the longest molecular dynamics (MD) in a MapReduce fashion. CloudBurst was proposed in [7] as a new highly sensitive parallel seed-and-extend read-mapping DNA sequencing algorithm based on MapReduce. The authors of [6,8] transformed high energy physics (HEP) data analysis and *K*-means clustering into MapReduce literature. Kwon et al. showed how a clustering problem can be effectively implemented in a MapReduce-style framework [9]. Bryant demonstrated an interesting case study of implementing sparse matrix multiplication in MapReduce in [10]. In [14], the authors exploited the feasibility of using MapReduce for matrix computation.

Efforts are also dedicated to integrate MapReduce and HPC applications from the perspective of system support. MR–MPI from Sandia National Laboratory is a lightweight library that implements the basic MapReduce functionality on top of MPI runtime systems [11]. The Hadoop community proposes Hamster to make MPI a first-class citizen on a Hadoop cluster [15]. In [12], Hoefler et al. discussed common strategies to implement a MapReduce runtime and the possible optimization on top of MPI. Twister is a runtime that extends MapReduce with iterative operations [16]. In [17], the authors have proposed an approach to running HPC applications on top of MapReduce file systems by bridging the semantic gaps.

2.2. MPI vs. MapReduce: programming structure

MPI is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementors, and users [1]. MPI can be used to implement the applications with ad hoc communications patterns within the process groups.

The performance comparison of MPI and MapReduce is a research topic under intensive study [6,8,14,18,19]. Nevertheless, all these works observed the performance degradation when transforming computing modules from MPI to MapReduce. The possible reasons for the extra overhead include,

1. The inflexibility in the communication/synchronization. MapReduce does not support flexible communication/synchronization among the processes. Some widely used collective MPI operations such as *All-To-All, All-To-One* and *One-To-All* are not supported by MapReduce. The barrier operation can only be placed at the end of the Map phase. Iterative operations are also not supported directly by MapReduce.
2. The inflexibility in job decomposition. The tasks are independent of each other during the Map or Reduce phase, which makes the sharing of some common computation components impossible.
3. The overhead in data manipulation. MapReduce utilizes data intensive file systems such as Google file system [20] or HDFS [21] as the underlying storage systems. These systems adopt multiple replicas to facilitate fault tolerance, which further undermines the performance of MapReduce applications from the perspective of data manipulation.
4. The inflexibility in the intermediate results. The intermediate results must be designed to follow the key/value pattern in MapReduce.
5. Platform-specific overheads. Hadoop is an open-source implementation of MapReduce and has been widely deployed on both research and production systems [22]. Potential overhead may be introduced due to the fact that Hadoop is implemented with JAVA.

This research builds its study on the recognition of MapReduce's disadvantage over MPI, but focuses on the performance study of the two programming paradigms when the impact of failures cannot be ignored. This paper focuses on the computing components that can be transformed to the MapReduce model. The applications that cannot be deployed on MapReduce are outside the scope of this work.

2.3. MPI vs. MapReduce: fault tolerance

From the perspective of reliability, MapReduce differs from MPI in both the impact scope of failures and the failure handling mechanism.

2.3.1. Impact of failures

Different programming models have distinctive capabilities to tolerate failures, depending on whether the impact of failures can