Distributed Graph Processing for Machine Learning: The Case for Condor and Spot Instances

Jing Fan   Ce Zhang

Abstract

We study distributed graph processing for machine learning applications in an environment in which the failure of workers is guaranteed to happen frequently. Notable examples of such environments can be found both in the classic grid computing, e.g., Condor, and the emerging cloud computing, e.g., *spot instances* on Amazon EC2. Different from the common practice that running machine learning algorithms with a dedicated cluster, it is still an open research question that how (and whether it is possible) to run machine learning algorithms in this environment.

This report documents our *preliminary* effort towards answering this question by building a prototype system. Ideally, we have two goals: (1) *expressiveness*: this prototype system should allow the user to specify a machine learning system as easy as existing frameworks designed for dedicated clusters; and (2) *efficiency*: the machine learning system specified by the prototype system should be able to run in environments such as Condor efficiently. We report our design of this prototype system that extends the programming model of a popular graph-processing engine, namely GraphLab. Given a user program written in our language extension, we describe the execution model. We validate our prototype system with an emergingly popular machine learning application, namely deep neural network, on *both* Condor and Amazon EC2. We find that our prototype system is able to achieve more than 12TFLOPS on Condor with more than 2.6K cores harvest from the national Open Science Grid (OSG).

1 Introduction

Machine learning has been one emerging area that attracts interests from the community of system research, especially research of distributed systems [3, 2]. Notable example including GraphLab [2], Spark [5], Google's DistBelief [1], and Yahoo's Parameter Server [4, 3]. In this work, we focus on the same goal, that is to build a framework that supports executing machine learning applications in an distributed environment.

It has been a common practice for distributed machine learning systems to be run on dedicated clusters. For example, both Spark and GraphLab has been reported to be able to execute machine learning algorithms on hundreds of machines; DistBelief is able to train deep neural network on 6000 machines; and Parameter Server is able to scale to 1000 machines. Our key observation is that existing frameworks [3, 2, 5, 1, 4] implicitly make *at least one* of the following two assumption about the distributed environment and workload:

1. The coordinators know *a priori* the set of worker nodes, including their configuration, number of worker nodes, and the network topology between these workers. These information are necessary for most existing systems to schedule their workload before execution. For example, Yahoo's Parameter Server uses consistent hashing [3] to allocate resources and workers, and use chain replication to deal worker failures. GraphLab [2] also uses similar approaches.

2. The graph to be processed consists of nodes that are executed with the same level of consistency. For example, in GraphLab, the execution engine can choose from three different consistency levels, and all the nodes in the same graph will follow the same level.

These assumptions are often true in the environment that these systems are designed for, in which the cluster is maintained in a centralized way or leased with cloud-based services (e.g., EC2). However, this assumption does not always hold for a popular environment that often be called *high throughput computation* (HTC) environment. One notable example is Condor, which, when ran on the national open science grid, can easily harvest hundreds of thousands of machine hours per day. Another example is the spot instance on Amazon EC2, which relies on a bidding-based model that could provide much cheaper solution than traditional cloud-based instances. These HTC environments have the characteristics that are different to the above two assumptions when applied to machine learning.

1. The coordinators does not know the set of workers *a priori*. For example, in Condor, the coordinator does not know how many workers will be assigned to it, and what type of machines will be assigned to it. Also, the workers are not guaranteed to be able to communicate with each other. Similar scenario applies to spot instances of EC2, in which the number of workers assigned to the coordinator depends on the bidding price and other bidder, while the configuration of machines are known to the coordinator.

2. Machine learning workload often contains heterogeneous consistency requirement. For example, for some data in machine learning workload, full consistency is not required, while for other data, one might need higher-level of consistency. This observation has also been made in a subset of existing systems, e.g., Parameter Server [3].

We study how to build a GraphLab-like system with these two observations, and study their implications on system design. We first propose a language extension of GraphLab for the user to specify a machine learning algorithm, and use Deep Neural Network (DNN) as an use case. We then study the design decisions we made for both Condor and spot instances of Amazon EC2. We validate our prototype system on standard benchmark data sets.

2 System Design

We describe the design decisions and how they relate to the two observations we made on Condor and spot instances of Amazon EC2. We first describe in more details about these environment, and present the language extension and execution model.

2.1 Condor and Spot Instance of EC2

Figure 1 illustrates the difference between a dedicated cluster, Condor instances, and EC2 instances. We describe their difference as follows.

Dedicated Cluster

Condor

EC2 Spot Instances

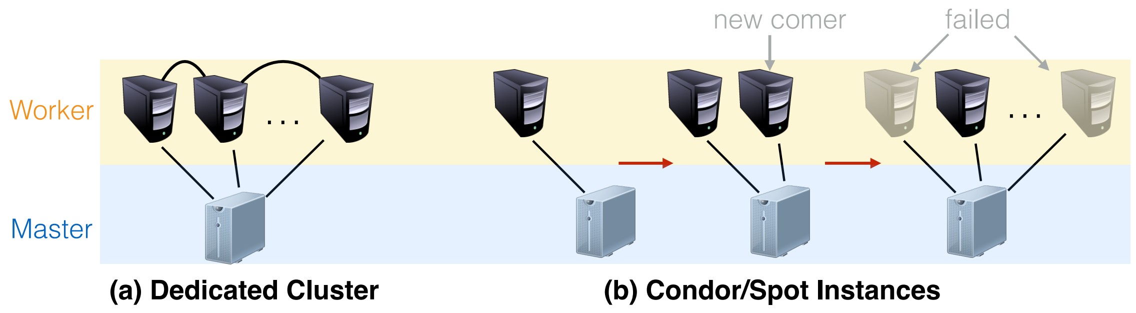

Figure 1: An illustration of (a) dedicated cluster and (b) Condor/Spot-instances. (b) illustrates three states in different time where the worker machines keeps come in and fail.

2.2 Language Extension

2.3 Execution Model

3 Performance Study

References

[1] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng. Large scale distributed deep networks. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1232–1240. 2012.

[2] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 17–30, Berkeley, CA, USA, 2012. USENIX Association.

[3] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, Broomfield, CO, Oct. 2014. USENIX Association.

[4] A. Smola and S. Narayanamurthy. An architecture for parallel topic models. *Proc. VLDB Endow.*, 3(1-2):703–710, Sept. 2010.

[5] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: Sql and rich analytics at scale. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 13–24, New York, NY, USA, 2013. ACM.