Prototyping, Testing, and Evaluating Scaling Incremental Applications and Frameworks in Cloud

Dong Dai
Computer Science Department
Texas Tech University
dong.dai@ttu.edu

Yong Chen
Computer Science Department
Texas Tech University
yong.chen@ttu.edu

Robert B. Ross
Mathematics and Computer Science Division
Argonne National Laboratory
rross@mcs.anl.gov

1. INTRODUCTION

More and more applications in Cloud fall into the incremental computing category where we need to process continuous incoming new data as well as existing historical data, and provide analytics over the entire dataset in a short time. Different from traditional streaming processing [10], the incremental applications usually contain some internal states that will change incrementally while processing new data. These changes can be as simple as manipulating numerical counters, or as complex as multi-stage computations including several iterations. For example, to monitor the popular *#topics* in Twitter, we need to hold an internal state recording the tweets counter of each *#topics*, and then simply increase the counters based on new tweets. A more complex example would be the PageRank algorithm for calculating the importance (page rank value) of each web page in the Internet [13]. In PageRank, the internal state contains the page rank value of each web page, and needs to change each time when a new web page is processed. However, unlike the Twitter example, each new web page will not directly change its own page rank value. It will affect all the neighbors first, then spread the changes in multiple iterations across the overall state, and finally converge to a stable value.

Implementing these incremental applications in a scalable way is a big challenge in current Cloud computing environment. This challenge contains several aspects. First, at the application level, choosing the right algorithm is complicated. Many algorithms usually have both batch versions and incremental or streaming versions, like, PageRank has a Monte-Carlo based method for streaming datasets, which is totally different from the original algorithm [9]. Although these different algorithms may be all correct, their efficiency and performance usually are highly dependent on the problem scenarios and execution environments. In Cloud environment, this requires a flexible but standard environment to test the correctness and evaluate the performance advantages or disadvantages. Second, at the framework level, supporting such large-scale incremental applications is difficult. Although there are many streaming or incremental frameworks existing like S4, Storm, Percolator, Oolong, Domino, Spark-Streaming and many MapReduce extensions [12, 2, 14, 11, 7, 16, 3, 8, 17, 15], none of them has dominated others to be a standard way of writing incremental applications due to their own limitations in different aspects. It would be both interesting and useful to have a flexible but standard environment to test and evaluate these frameworks for different applications. The results not only help developers choose the right solution, but also possibly lead to a better framework or new optimizations for existing frameworks.

2. EXPERIMENTS

First, a broad category of applications are chosen for our evaluations. We consider simple incremental applications like topic trend or keyword monitoring in social network, but more importantly, we will go through the popular machine learning algorithms including linear regression, decision tree, k-means, PageRank, collaborative filter, support vector machine, and neuron networks, etc., to check the right strategy to implement them.

Second, for each of these algorithms, three different scenarios are proposed to evaluate the performance: the batching algorithms on static datasets; the batching algorithms on streaming datasets; and the incremental algorithms on streaming dataset. To show the performance of batch algorithms on static datasets, we will test the popular MapReduce [6] version of these machine learning algorithms based on the open-source implementation from Mahout [1]. However, for other two cases (batching and incremental algorithms on the streaming dataset), the evaluation would be more complex due to more choices are provided. There are MapReduce extensions for incrementally processing[3, 8, 17, 15] that we can use to implement different algorithms, and also incremental or streaming frameworks we can use [12, 2, 14, 11, 7, 16]. Among all these choices, most of the MapReduce extensions are research projects lacking of enough functionalities and optimizations. To ease our efforts, we plan to only select one of them (Twister [8]) as the state of art solution for further comparison. On the other hand. Many incremental or streaming frameworks are mature and ready to use, like S4, Storm, or Spark-Streaming. We will evaluate all the accessible frameworks in our tests. In addition to use these existing incremental frameworks, we also plan to write the MPI implementations as the base-line performance.

As we have described, for the framework evaluation, we only deploy accessible *mature* open-source frameworks (i.e., Twister, S4, Storm, Spark-Streaming) and our own framework called Domino [7] into different standardized Cloud environments built from the Chameleon [4] and CloudLab [5]. To test the scalability, we need to deploy them in different sizes.

Note that although there are already many comparisons among different incremental frameworks, researchers and developers from different communities usually disagree with each other about these results. In most cases, these disagreements come from misconfiguration or non-optimal deployment of specific frameworks. In our experiments, we plan to work with different communities closely to ensure we carry out the nearly optimal deployment for these frameworks.

The most critical performance measurement of incremental applications is how fast they can absorb the incremental inputs. To detect how fast the applications can absorb the incoming data, in our evaluations, we will give burst external inputs instead of slow streams. This extreme situation shows us the potential of an incremental framework. Specifically, for different evaluations, we will create a large static datasets simulating the whole dataset and small datasets simulating the streams. These changing datasets will be written into the incremental framework as fast as possible, in order to trigger the applications. We will also run the MapReduce applications on both the entire input dataset and the smaller changing datasets. Running MapReduce only on the smaller changing datasets (streams) actually shows its best performance since they process only the updated data.

3. REQUIREMENTS

According to our plan, there will be several requirements on the Cloud platforms.

First, since we aim at providing performance test, the ability of *quickly deploying different Cloud environments* will be critical to us. Deploying large-scale private Cloud in public Cloud platform is always a time-consuming job. Allowing developers to make customized images will be a huge benefit. Furthermore, if the platforms provide deployment tools like Docker and the relevant management services, the evaluation process will be easier too.

Second, *the performance tests require a consistent service level* guarantee. As we intend to evaluate the performance of different frameworks, the deployed Cloud should keep a consistent service level with the same resources. Providing a way to users to guarantee this will be a benefit.

Third, the platform should have *a good scalability*, which is an important factor to our experiments. It will be a huge benefit to scale out to test the performance under real-world workloads.

4. CONCLUSION

In this project, we propose a detailed testing and evaluation on the scaling incremental applications and existing processing frameworks in Cloud environment. As we have described, this type of applications is more and more important in real-world scenarios. The results of this project not only help application developers to choose right implementation strategies for their applications, but also help the framework developers to improve their productivity.

References

[1] Mahout project. In *http://mahout.apache.org*.

[2] Storm project. In *http://storm-project.net/*.

[3] Y. Bu, B. Howe, M. Balazinska, and M. Ernst. Haloop: Efficient iterative data processing on large clusters. In *Proceedings of the VLDB Endowment*, volume 3. VLDB Endowment, 2010.

[4] Chameleon. http://www.chameleoncloud.org/.

[5] CloudLab. http://www.cloudlab.us/.

[6] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] D. Dong, Z. Xuehai, K. Dries, R. Robert, and C. Yong. Domino: An incremental computing framework in cloud with eventual synchronization. In *The 23th ACM International Symposium on High Performance Parallel and Distributed Computing (HPDC'14)*, 2014.

[8] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010.

[9] A. Goel. Algorithms for distributed stream processing. Technical report, Stanford University, 2012.

[10] U. Jaeger and J. Obermaier. Parallel event detection in active database systems: The heart of the matter. *Active, Real-Time, and Temporal Database Systems*, pages 159–175, 1999.

[11] C. Mitchell, R. Power, and J. Li. Oolong: asynchronous distributed applications made easy. In *Proceedings of the Asia-Pacific Workshop on Systems*, page 11. ACM, 2012.

[12] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*. IEEE, 2010.

[13] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. *Standford InfoLab*, 1999.

[14] D. Peng and F. Dabek. Large-scale incremental processing using distributed transactions and notifications. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*. USENIX Association, 2010.

[15] H. Yang, A. Dasdan, R. Hsiao, and D. Parker. Mapreduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of Data*, pages 1029–1040. ACM, 2007.

[16] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, pages 10–10. USENIX Association, 2012.

[17] Y. Zhang, Q. Gao, L. Gao, and C. Wang. Priter: A distributed framework for prioritized iterative computations. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 13. ACM, 2011.