Large Scale Pagerank with MapReduce

Shuo-Huan Chang, Yuduo Zhou, Parag Malshe, Hui Li

Abstract— The study of large web graphs is becoming increasing popular in the Cloud. Some large web graph applications have already been implemented with the MapReduce technical, such as Pagerank. However, the MapReduce model is not aware of the inherent random access pattern of web graph which generates lots of network traffic. Thus the efficient processing of large scale Pagerank challenges current MapReduce runtimes. We implemented the Pagerank with different MapReduce runtimes, which includes Twister, DryadLINQ, and Hadoop. By studying the performance of PageRank on different runtimes, we conclude three runtime related optimization strategies for the PageRank.

Index Terms – MapReduce, PageRank, Cloud

Introduction

The PageRank is an already well studied web graph ranking algorithm [1]. It calculates numerical value to each element of a hyperlinked set of web pages, which reflects the probability that the random surfer will access that page. The process of PageRank can be understood as a Markov Chain which needs recursive calculation to converge. An iteration of the algorithm calculates the new access probability for each web page based on values calculated in the previous computation. The iterating will not stop until the difference of all rank values between two iterations is less than a predefined threshold.

MapReduce is a distributed programming technique proposed by Google for large-scale data processing in distributed computing environment. It simplified the implementation of many data parallel applications by removing the burden from programmer such as tasks scheduling, fault tolerance, and messaging.

Implementations

We have implemented the PageRank with Twister, Hadoop, and DryadLINQ on ClueWeb data set with nearly 50 million web pages. We do not create any new PageRank algorithm, but make use of the most common PageRank algorithm [1]. Fig1 shows architecture of MapReduce PageRank. The web graph is spitted into some partitions and each partition is stored as an adjacency matrix (AM) file. Each Map task processes one partition and calculates the partial rank values for some pages. The Reduce task merge all the partial values and generate the global rank values for all the web page.

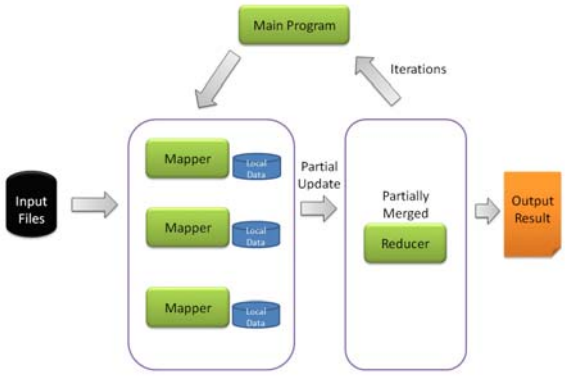

Figure1. Architecture of PageRank MapReduce

Optimization Strategies

We conclude three runtime related optimization strategies for the PageRank. 1) Cache the partition file in the memory during multiple iterations. 2) merge small web graph partitions into large one as long as it fit the

memory size. 3) Local merge before the global merge in Reduce stage.

Twister [2] is an enhanced MapReduce runtime that supports iterative MapReduce computations efficiently. It loads the web graph partition data from disk once, and cache the data in the memory during the multiple iterations of the computation. Hadoop is also a MapReduce framework that supports data-intensive distributed applications. The Hadoop PageRank need reload the web graph partition file from disk to memory each iteration during the computations. Fig2. shows that Twister output performance Hadoop in job run time, as it cache the partition file in the memory during multiple iterations.

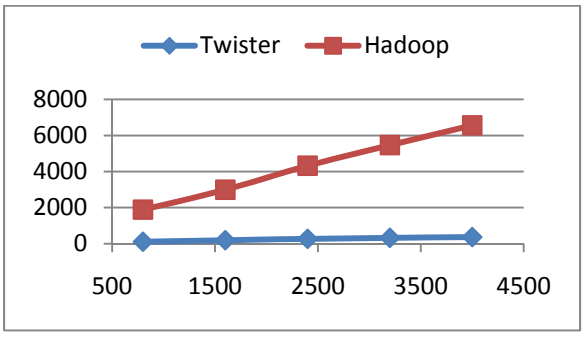

Fig2. Twister PageRank vs. Hadoop PageRank

Dryad is a DAG job execution engine, which contains the MapReduce programming model. DryadLINQ is a compiler which translates LINQ programs to distributed computations. LINQ is an extension to .NET. We implemented PageRank with DryadLINQ on a 32 nodes Windows HPC cluster. As shown in Fig3, when the tasks granularity increase; the job turnaround time decrease greatly. One main reason is that increasing the task granularity can decrease the number of tasks, thus decrease the job scheduling cost.

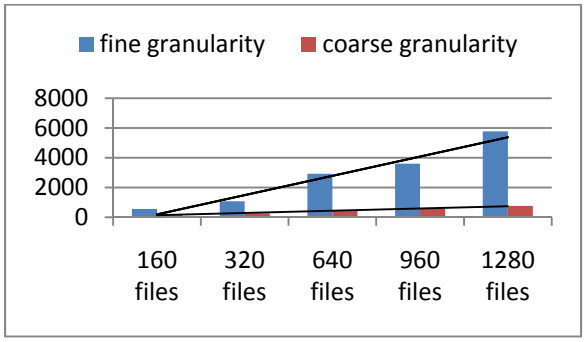

Fig3. DryadLINQ PageRank with different task granularity.

References

[1] http://en.wikipedia.org/wiki/PageRank.

[2] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. Bae, J. Qiu, G. Fox, "Twister: A Runtime for Iterative MapReduce," HPDC, Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing.

[3] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub, "Exploiting the Block Structure of the Web for Computing PageRank," Stanford InfoLab, Technical Report2003.

[4] (2009, The ClueWeb09 Dataset. Available: http://boston.lti.cs.cmu.edu/Data/clueweb09/

[5] M.Isard, M.Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks." Presented at the proceedings of the 2nd ACM SIGOPS