Memcached Integration with Twister

Saliya Ekanayake, Jerome Mitchell, Yiming Sun, Judy Qiu

sekanaya@cs.indiana.edu, jeromitc@umail.iu.edu, yimsun@cs.indiana.edu, xqiu@indiana.edu

***Abstract*— MapReduce programming model has proven its value in the domain of large scale data analysis since the introduction by Google. Twister extends this model efficiently with its support for iterative MapReduce with streaming data transfer via publish/subscribe messaging. This efficiency is challenged recently with applications requiring large chunks of data transfer. We propose having a unified memory access layer across Twister nodes instead of a publish/subscribe broker as a viable solution. The implementation integrates the open source Memcached, a distributed memory caching system, with Twister. We expect to get comparable performance results on this approach against current Twister implementation. We also plan to compare performance on our implementation with another proposed solution, which uses RPC messaging with Twister.**

I. INTRODUCTION

The advent of data-intensive applications, such as Web page ranking, log analysis, galaxy image clustering, ocean simulation, and biological data analysis, has led to invention of highly scalable distributed computation frameworks. Google pioneered this with their implementation of MapReduce [1] framework, which supports map and reduce model of computation. Its success in handling large scale data analysis made the open source community at Apache to build an open source version named Apache Hadoop [2].Simple MapReduce model is not sufficient for many scientific applications as most of them require iterations over MapReduce. Twister [3], an iterative MapReduce framework developed by the Pervasive Technology Institute's SALSA group at Indiana University, addresses this aspect explicitly. Also, it has gained performance over comparable frameworks like Hadoop and DryadLINQ [4] as a reason of its streaming data transfer via publish/subscribe messaging. This approach to messaging, however, has shown delays when transferring large chunks of data. Our solution to this challenge is to integrate a unified memory access layer across Twister nodes to handle the large data transfer. This eliminates the need of a messaging broker and is expected to gain comparable performance against current implementation of Twister.

We propose our solution as a poster describing the integration of Memcached [5, 6, 7, and 8]; a distributed memory caching system, with Twister and how our approach can support the scalability required for a high throughput of data deluge applications. The rest of the proposal presents an introduction to Memcached followed by the evaluation plan of the proposed solution. We have also presented a set of known work relating to this issue at the end.

II. MEMCACHED

Memcached is an open source high-performance distributed memory object caching system. It is adopted by variety of vendors like Flickr, Twitter, Wordpress, etc. The idea of Memcached is to provide a unified layer of memory across a set of machines (Figure 1). The memory layer simply enables storing and retrieving key value pairs. This makes it more suitable for Twister as the communication often results in key value pairs. Once integrated (Figure 2), Twister will be able to access objects seamlessly across its compute nodes giving better performance over its current architecture (Figure 3).

III. VALIDATION AND EVALUATION METHODS:

In order to determine the scalability of a messaging infrastructure for various loads, a guideline must be undertaken.

1. The effectiveness of the infrastructure
2. The time taken for messages to be delivered
3. The memory and CPU utilization of the architecture

We also intend to compare and contrast using the above criteria with Twister's current implementation and the alternative solution of using RPC messaging with Twister. We explain two of the key applications of which we intend to compare the performance.

PageRank algorithm calculates the numerical value of each Web page on the World Wide Web, which reflects the probability of a surfer accessing that particular page. 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 (δ) is less than a predefined threshold, where δ is the vector distance between the page access probabilities in Nth iteration and those in $(N+1)^{th}$ iteration.

Biological sequence alignment algorithm using Smith-Waterman is of classic all-pairs nature that performs local alignment for each pair of input sequences. This requires data transfer across each node while performing the

computation thus giving us good grounds for performance comparison.

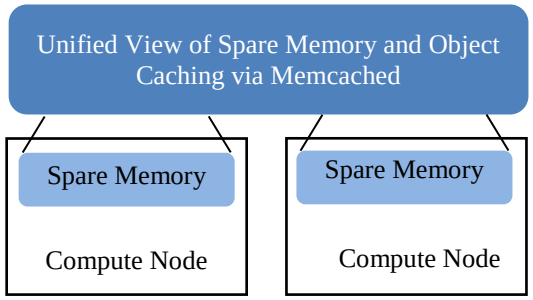

FIGURE 1: MEMCACHED VIEW OF SPARE MEMORY

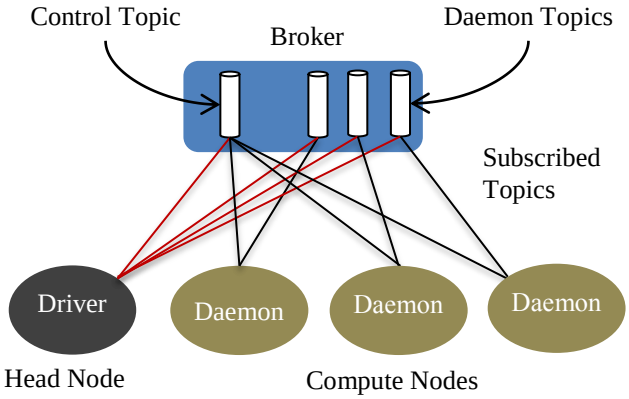

FIGURE 2: TWISTER ARCHITECTURE

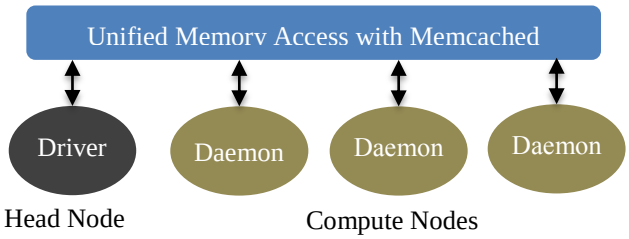

FIGURE 3: PROPOSED MEMCACHED INTEGRATION

IV. RELATED WORK

There are two proposed solutions to overcome the bottleneck of data transfer via publish/subscribe at present. One is to have a separate peer-to-peer data transfer via TCP in Twister. Thus, the messaging broker would only handle control messages leaving the burden of data transfer to compute nodes themselves. The other approach is to use RPC messaging between nodes to handle communication. This is currently under development in parallel with our solution by another group of students at Indiana University.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107-113, 2008.

[2] *Apache Hadoop*. Available: http://hadoop.apache.org/

[3] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, Geoffrey Fox, Twister: A Runtime for Iterative MapReduce," The First International Workshop on MapReduce and its Applications (MAPREDUCE'10) - HPDC2010

[4] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda, and C. J., "DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language," in *Symposium on Operating System Design and Implementation (OSDI)*, 2008.

[5] Deris M et. al, 2003, High Service Reliability For Cluster Server Systems, Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'03).

[6] Fitzpatrick B., 2004, Distributed Caching with Memcached, Linux journal.

[7] Shen H.H., Chen S.M., Zheng W.M., and Shi S.M., 2001, "A communication model for Data Availability on Server clusters", Proc. Int'l. Symposium on Distributed Computing and Application, 2001.

[8] Cvetanovic Z., 2004, Performance Analysis Tools for Large-scale Linux Clusters, 0-7803-8694-9, IEEE 2004