November 19, 2010

SnowFlock-Aware Hadoop Implementation

(http://www.cs.toronto.edu/~mjulia/CSC2231Project/SnowFlockAwareHadoop.html)

Progress Report II: Shahan Khatchadourian and Julia Rubin

To recall, our main objective is to enhance MapReduce jobs with the SnowFlock (Kaleidoscope) cloning mechanisms as described in the following two scenarios:

(1) Create clones as part of the Map task, before the user-defined Map functions are called; destroy them after they finish transferring their output to the Map task.

(2) Create clones as part of the Reduce task, before the user-defined Reduce functions are called; destroy them after they finish transferring their output to the Reduce task.

We prototyped the first scenario using forked threads to simulate clones. That is, we create a new thread after the record reader has generated tuples and before these tuples are processed by the user-defined map functions. Each new thread (or clone, in the future), processes a fixed number of map functions. This number is controlled by a configuration parameter and is currently set to 3. The execution results are passed back to the original, owning, thread over the network. Intermediate key/value pairs are not written to disk by map clones, but rather by the owning thread in order to preserve MapReduce's fault-tolerance.

This simple simulation technique allowed us to quickly prototype and validate our approach, since it did not require modifying Hadoop code nor interacting with SnowFlock APIs. Our next step is to replace threading by SnowFlock cloning, and also to introduce cloning into the Reduce phase. Our implementation will need to monitor the number of requested clones and ensure that this number is restricted to within cluster constraints. This can potentially be done at a SnowFlock or Job Configuration level.

Since, at this stage of implementation, cloning is conducted within a user-defined MapReduce job, thus, allowing all thread/clone handling to be accomplished without modifying the Hadoop code itself, we plan to investigate further whether embedding the cloning into the Hadoop infrastructure can significantly influence the results of our evaluation. A significant result of placing cloning within Hadoop code is that existing job code can be run without modification. On the other hand, we expect few code modifications would be required to existing job code even if SnowFlock APIs are exposed to the job developer.

A consequence of cloning is the introduction of network overhead, even for data-local map tasks, because the data is to be transferred from the spawn clones back to their "owning" task. We intend to investigate under which scenarios this overhead is justified, and can be considered small comparing to the gains earned by multiplexing processing power as the result of the impromptu clones creation.

For the evaluation, we plan to measure the performance of stock Hadoop, configured to run one map and one reduce task. Then, from these map and reduce task, we will spawn various number of clones (up to the maximum of different map and reduce keys, respectively) and measure the performance of each such configuration. We will choose those configurations that perform the best and pre-configure Stock Hadoop to run the same number of map and reduce task, but without any cloning. We will compare the performance to the version that runs with cloning. We also plan to repeat this process with more than one map and reduce tasks in the initial settings. To evaluate the overhead of cloning, we will measure the performance of an application running with a fixed number of nodes, vs. the same application running spawning the same number of clones.

We have identified several applications and workloads to be used for performance measurements, in particular, the publicly available benchmark data and applications introduced in [1]. However, the smallest data-set used in these benchmarks is 53.5G – we do not have such amount of free space available. Thus, we will use a subset of this data-set. Additional applications that we plan to use are canonical MapReduce applications, such as AggregateWordCount and Grep, which are provided as part of the Hadoop distribution.

References

[1] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data, New York, NY, USA, 2009, pp. 165-178.