X10 - Enabled MapReduce

Han Dong, Shujia Zhou — University of Maryland Baltimore County
David Grove — IBM

Introduction

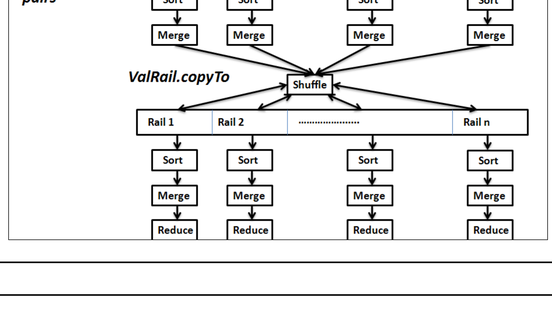
The MapReduce programming model was introduced by Google as an efficient way to support distributed computing and processing large datasets on a large cluster of computing nodes. A notable MapReduce implementation is done in Hadoop, which utilizes its own file system (HDFS) for storing data and sorting data across compute nodes. It consists of a map function for data distribution, a combiner function that merges the intermediate maps, then a partition function that shuffles and sorts the data across different computing nodes. This design is common among other flavors of MapReduce implementations.

Purpose

A key performance factor in the design of MapReduce is the cost of shuffling data across different nodes and the reliance on a third party file system to handle data storage. We decided to address these key challenges with the implementation of MapReduce in the X10 language through a word-count use case. The X10 language supports a Asynchronous Partitioned Global Address Space (APGAS) model, this allows for benefits such as easy-across-node communication, which will also help to avoid the use of third party file systems.

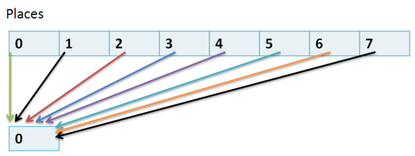
Design of X10 – Enabled MapReduce

HashMap Implementation

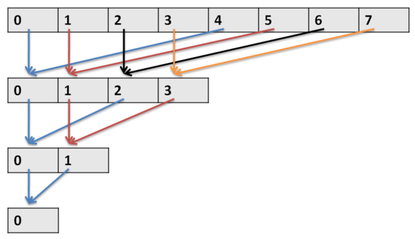


- Stores Key, Value pairs as String, Integer in HashMap data structure.
- Easy to detect duplicates since each element is stored with a hash function that uses Key as input value.
- Utilizes two merging functions to shuffle data across different computing nodes.
- Single Place Merge
- Parallel Reduction Merge
- Difficult to sort HashMap since its using a hash function to store elements. Sorting is done through conversion to ArrayList.

Single Place Merge



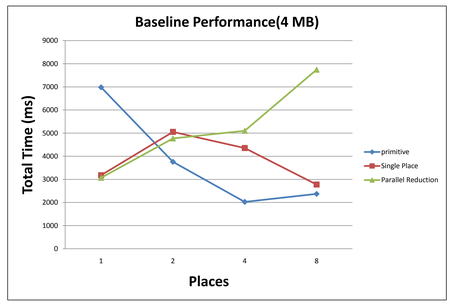
Parallel Reduction Merge

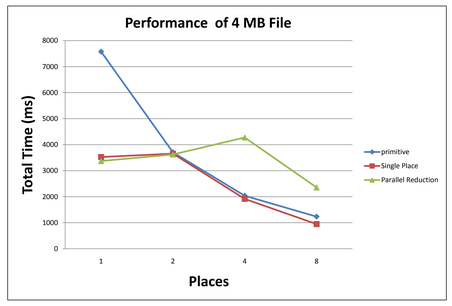


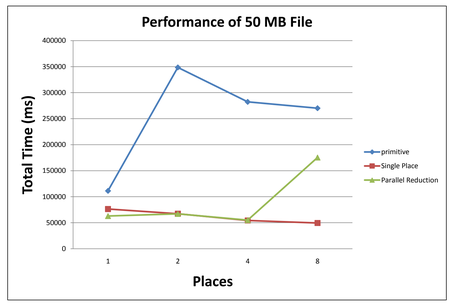
Rail Implementation

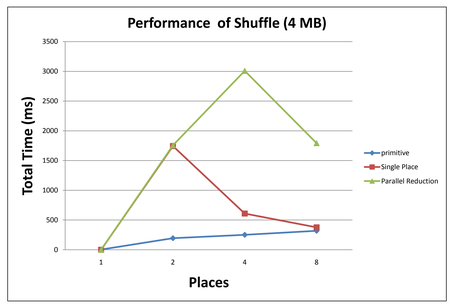
- Stores Key, Value pairs as 1) String, Integer pairs and 2) String, LinkedList<Integer> pairs in Rail.
- Rail can be converted to ValRail; which has built in functions (copyTo) to transport data to remote *places*.
- Motivation behind linked list was to copy head of list to remote *place*, then access rest of body through remote memory accesses.
- Motivation behind primitive String, Integer Rail was to simplify data movement.
- Utilizes a merge sort algorithm to sort Rail.

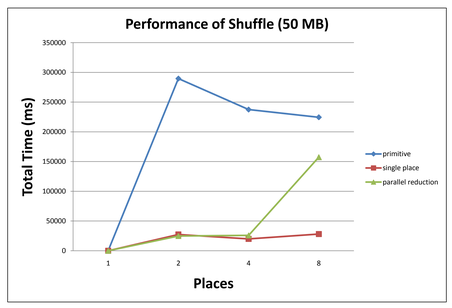
Results of X10 – Enabled MapReduce











Analysis

- Ran tests on IBM Watson lab servers running x10 version 2.0.4. Each blade has two Quad Core AMD processors (8 cores total).

- We were not able to run tests with the Rail<String, LinkedList<Integer>> data structure since it was not possible to pass head node of linked list to a remote place; subsequently accessing rest of the data through references in the list. In order to do shuffling successfully with a linked list, each node in the linked list had to be copied over. This introduces redundant data copying and its implementation would be similar to utilizing a HashMap instead.

- **Baseline case is run on only one blade, from one place up to eight. Demonstrates the infeasibility of utilizing parallel reduction merge to shuffle data. Poor performance can be attributed to the fact that it is a recursive function and it also performs additional copying of data that is redundant.**

- The rest of the tests were run across four computing nodes.
- **The performance of processing the 4MB file indicates that both the primitive String, Integer implementation with a Rail and the single place merge with HashMap are scalable to 8 cores. Demonstrates benefit of utilizing X10's libraries to do across node communication.**

- **The performance of processing the 50 MB file indicates that single place merge of HashMap is still scalable to 8 cores. The primitive String, Integer implementation of Rail actually performed the worst.**
- Based on the shuffling data, it can be seen that the transportation of data across different computing nodes is a major factor in the overall performance.
- **The copying of data by utilizing the *at(Place)* place shifting operator seems to perform better than the *ValRail.copyTo* operator. This is primarily because we were not able to access a mutable object from a remote *place*.** In order to utilize the copyTo function, we had to convert the GrowableRail into a ValRail and preprocess the borders of the ValRail to specify the ranges to be copied; this resulted in very complex code, redundant data copying and also a waste of memory. Whereas the HashMap implementation had no redundant data copying, each data was appended to a pre-existing HashMap

Summary

- We have investigated the X10 implementation of MapReduce with three different methods and indentified both the benefits and difficulty of using x10 to implement data shuffling. **While the x10 libraries provide very robust functions for easy transportation of data, the lack of remote memory accesses may be a good design decision to reduce the overuse of pointers. However, this design decision makes the shuffling of data across places time-consuming---a potential performance bottleneck in an X10-enabled MapReduce Implementation.** Although the APGAS model provides us with very powerful tools to do across node computing, it could provide us with better features such as the ability to build a mutable data structure in one place and transport its contents to a data structure in another place.

Vocabulary

•**Activity:** Light weight thread within a place

•**Place:** Collection of mutable objects and activities that operate on those objects with its own allocated heap memory

•**Partitioned Global Address Space (PGAS):** Each place is allocated with its address space, they do not overlap. Allows remote access of objects in one place by activities of another place.

•**Map:** handles the parsing and storing of words into the data structures local to each *place*.

•**Shuffle/Merge:** handles the distribution of data from local to remote *places*.

•**Sort:** handles the final sorting of the data structure and removes duplicates.