MapReduce: Data Distribution for Reduce

Graduate Group #10
Team member 1 (Heena Dave, 810917421, *hdave@kent.edu*)
Team member 2 (Bhavy Bhut, 810917417, *bbhut@kent.edu*)

1. Introduction

Hadoop is based on master-worker architecture. Master node is responsible for job scheduling and worker node runs tasks assigned by master node. Scheduling in master node is based on a hash function in hadoop architecture. By default, hash function is *Hash (Hashcode (intermediate key) mod number of Reducers)*. The default hash function is effective and provides load balance for uniformly distributed data. However, this hash function failed to achieve effective load balance for non-uniform data. For example, in basic Word Count Program, basic words (a, an, the, etc.) appear more frequently than other words so it imposes more work on corresponding reducers. Consider an example shown in below figure for letter count example.

a	b	c	a	b	d	c	a	B	d	a	a	a	b	c	e

Hash function load	a: 6, d: 2	b: 4, e: 1	c: 3
Optimal load	a: 6	b: 4, d: 2	c: 3, e: 1

Above figure illustrates example of key assignment in MapReduce, it also depicts scenario of hash function load balancing and optimal load balancing for three reduce workers. Hash function assigns keys *a* and *d* to first machine, *b* and *e* to second machine and only c to third machine. As result first machine has maximum load of 8 while third machine has only load of 3. As shown in optimal load, hash function load balancing is not effective.

As in web index data, search engines and social sites like google, facebook, twitter has more visitors than other sites. Now, if we want to calculate the number of times a website is visited then as per default hash function reducers with search engines and social sites has more data to process than other nodes. It will increase overall processing time as few reducers will sit idle until all the reducers produces final output value.

2. Project Description

MapReduce programming model performs tasks based on dividing work among two functions: Map and Reduce. The map function generates intermediate key-value pair while the reduce function performs the final output. For example, for URL-count example, map function generates key-value pairs where each URL being the key and value will be one for each key. Then, the reduce function performs the count operation. Here, between Map and Reduce function calls, there are some sorting operations done to sort all the values of the same key together so that one reduce function can take all the intermediate key-value pairs belonging to same key.

The problem with this technique is, in real, the data is not evenly distributed. Hence, the data distribution for reduce function is not efficient in a case where there are some URLs being hit a lot and some URLs being hit only a few times. In this case, the reduce function handling URLs being hit more will be busy calculating the final count value and the reduce function handling URLs being hit rarely will be free as it will finish calculating the final value faster.

The project is about distributing data for reduce function effectively so that no reduce function takes a lot of time to calculate the final value while other reduce functions are idle.

Challenges to tackle this problem is that master node needs to distribute data to reduce workers in even manner to achieve effective load balancing. As the data for the same key is distributed between multiple reduce functions, there is one final reduce operation must be performed to calculate final value from previous reduce functions' values. The data distribution should be effective enough in such a way that even there is one more reduce function to be performed, the performance should still be efficient than the original MapReduce functionality.

We are two members working on this project: Heena Dave will be working on Algorithm design and load balancing and Project Report while Bhavy Bhut will work on Program Creation and Performance comparison between original MapReduce and data distribution of Reduce function.

3. Background

- Related papers (or surveys for graduate teams)
  - Hesham A. Hefny, Mohamed Helmy Khafagy, Ahmed M Wahdan, "Comparative Study Load Balance Algorithms for Map Reduce environment", International Journal of Applied Information Systems (IJAIS) – ISBN : 2249-0868 Foundation of Computer Science FCS, New York, USA
  - Yanfang Le, Jiangchuan Liu, Funda Ergun, Dan Wang, "Online Load Balancing for MapReduce with Skewed Data Input", ISBN 978-1-4799-3360
  - FENG LI, BENG CHIN OOI, M. TAMER ÖZSU, SAI WU, "Distributed Data Management Using MapReduce", ACM Computing Surveys
  - Jeffrey Dean, Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", Google, Inc.
- Software Tools
  - Library: Apache Hadoop which we use as a distributed data storage and Yarn for job scheduling
  - Java: Java Sdk 6 or higher
  - IDE: Eclipse IDE for source code

    - DBMS: not required
- Required hardware
    - Hardware requirement for Slaves

Workload pattern	**Storage**	**Processor** (1.8 GHz or higher recommended)	**Memory (GB)**	**Network**
Balanced	Based on input data	4	8	1 GB onboard
Compute intensive		8	16	
Storage heavy		4	8	

    - Hardware requirement for Master (Namenode)

Workload pattern	**Storage**	**Processor** (1.8 GHz or higher recommended)	**Memory (GB)**	**Network**
Balanced	Based on input data	4	8	1 GB onboard
Compute intensive		4	8	
Storage heavy		4	8	

- Related programming skills
    - Basic understanding of Hadoop File System
    - Should be aware of MapReduce Framework for functionality of this project
    - Object oriented programming
    - Distributed Environment

4. Problem Definition

- Formal (mathematical) definitions of problems

  The problem with MapReduce is that the data for Reduce function is not evenly distributed and it is distributed based on the key values. On the other hand, the data in real world application is non-uniform so some reduce function will take a lot of time in calculating the final value whereas others will finish faster and sit idle until all the reduce functions are done with the calculations.

  The project is about making a programming model in which the Map function will act similar to MapReduce programming model but the data distribution for Reduce function will differ and the data distribution will be more uniform considering the performance improvement.

  For example, in letter count, the vowels a, e, i, o and u are referred more frequently than the consonants. So, the reducers handling vowels will take a lot more time than the ones handling consonant. Now, even if a reducer calculates count of 4-5 consonants together but still in some cases the data distribution will not be even.

- Challenges of tackling the problems

  Although, the data distribution is an important factor, when the values of same key is distributed amongst different reducers, there is a final reducer call is needed to calculate final value from all the calculated values of reducers handling the same key. This extra function call will also take some amount of time and it needs to be considered in performance. Hence, the data distribution should occur only in some cases where there is a huge amount of difference between numbers of values for the same key.

- A brief summary of general solutions in your project

  The data distribution for Reducer should be more uniform so that all the reducer functions finish their tasks almost simultaneously and no function will sit idle for long time. This will increase performance as well as utilize the resources efficiently. To do so, an algorithm should be designed in a way that will calculate total number of keys, total number of values and a ratio of total values / total reducers. Based on this ratio, reducer function will handle key-value pairs with closest ratio.

  For example, there are five keys for word count:

  A: 10, B: 20, C: 30, D: 40, E: 1500

  And there are two reducer functions available which can calculate minimum value for each key then the normal MapReduce will work as follows:

  Reducer 1: A: 10, B: 20, C: 30, D: 40 ➔ total keys=4 and total values= 100

  Reducer 2: E: 1500 ➔ total key = 1 and total values = 1500

  It is not optimal as Reducer 1 will finish its task faster even though it handles 4 keys while Reducer 2 will take more time than Reducer 1.

  The algorithm should be designed in a way that,

  Total values = 1600

  Total keys = 5

  There are 2 Reducers so each reducer should take around 800 values to calculate.

  Therefore,

  Reducer 1: A: 10, B: 20, C: 30, D: 40, E: 700

  Reducer 2: E: 800

  Final Reducer ➔ Calculate minimum of Reducer 1 key= E value and Reducer 2 key= E value.