Map-Reduce Algorithms and Analysis

Chandan Mazumdar

Matrix Vector Multiplication

- **Problem:** We have an n×n matrix M, whose element in row i and column j will be denoted mij and a vector v of length n whose jth element is vj

  Then the matrix-vector product is the vector x of length n, whose ith element xi is given by

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

MR Solution

- **Case 1: Vector v fits in main memory**

  Assuming that n is large but not that large that vector v cannot fit in main memory.

- **Storage:** The matrix M and the vector v each will be stored in a file of the DFS. Assuming the row-column coordinates of each matrix element will be discoverable, either from its position in the file, or because it is stored with explicit coordinates, as a triple (i, j, mij).
- v is first read into the main memory entirely and are made available to all instances of the Map function

Map and Reduce functions

- **The Map Function:** It is applied on one element of M.

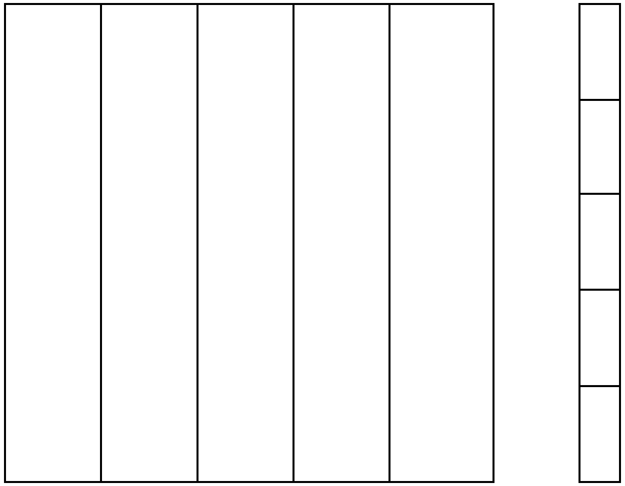
  Each Map task will operate on a chunk of the matrix M. From each matrix element mij it produces the key-value pair <i, mij.vj>.

- **The Reduce Function**: The Reduce function simply sums all the values associated with a given key i. The result will be a pair <i, xi>.

Too large vector

- **Case 2: Vector v too large to fit entirely in main memory.**
  In this case a large number of disk access are required as pieces of vector are moved to mian memory to multiply components by elements of the matrix.

- **Storage:** Divide the matrix into vertical stripes of equal width and divide the vector into an equal number of horizontal stripes, of the same height
- The ith stripe of the matrix multiplies only components from the ith stripe of the vector
- We can divide the matrix into one file for each stripe, and do the same for the vector.
- Each Map task is assigned a chunk from one of the stripes of the matrix and gets the entire corresponding stripe of the vector.



Division of a matrix and vector into five stripes

Distributed grep

- **Problem:** Grep command outputs each line that match a given pattern in a given collection of files.
- It’s serial algorithm, works by inspecting each line of text for the pattern
- For huge files MR can be used
- The file will be stored in chunks in HDFS

MR solution

- **Map function:** The map function emits a line if it matches a supplied pattern.

```
map(String key, String value):
  // key: pattern
  // value: document collection
  for each line l in value that contains key:
    EmitIntermediate(key,l);
```

MR Solution

- **Reduce function:** The reduce function is an identity function that just copies the supplied intermediate data to the output.

```
reduce(String key, Iterator values):
// key: pattern
// values: a list of lines that contain the pattern
    String result = null;
    for each line v in values:
    result = result.append(values+ "newline");
    Emit(AsString(result));
```

Count of URL Access Frequency

- **Problem:** Count the frequency with which each URL is accessed in the log of webpage requests.
-
- Similar to the problem of WordCount , instead of counting frequency of words, we count the same for URLs.

- **Input**: Log of webpage requests

MR Solution

- **Map Function :**The map function processes logs of web page requests and outputs

  <URL, 1>.

- **Reduce Function**: The reduce function adds together all values for the same URL and emits a <URL, total count> pair.

Map and Reduce functions

```
map(String key, String value):
        // key: document name
        // value: document contents
          for each URLu in value:
          EmitIntermediate(u, "1");


reduce(String key, Iterator values):
        // key: an URL
        // values: a list of counts
           int result = 0;
           for each v in values:
           result += ParseInt(v);
           Emit(AsString(result));
```

Reverse Web-Link

- **Problem:** For each webpage, we want to find out all the pages that has a direct link to it and hence we find the reverse structure of the webpage

  target → list of sources.

- **Input**: Set of web pages. We want the list of source → target links through which it is possible to navigate from the source webpage to target webpage.

Map

- **Map function:** The map function outputs<target, source> pairs for each link to a targetURL found in a page named source.

map(URL key, String value):
// key: source URL
// value: contents of source URL
for each target URL u in value:
EmitIntermediate(u, key);

Reduce

- **Reduce function:** The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair:<target, list(source)>

```
reduce(String key, Iterator values):
    // key: target URL
    // values: a list of source URLs
        List<URL> result = null;
        for each source URL v in values:
        AddToList(result, v);
        Emit(target, result);
```

Inverted Index

- **Problem:** Normal indexing occurs as :

  document → list of words in it

- But for query search on basis of multiple words, an inverted index is more useful

- Output

  word → list of documents

Map

- **Map function:** The map function parses each document, and emits a sequence of <word, document ID>pairs.

map(String key, String value):

// key: document name/ID

// value: document contents

for each word w in value:

EmitIntermediate(w,key);

Reduce

- **Reduce function:** this function accepts all pairs for a given word, sorts the corresponding document IDs and emits a<word,*list*(document ID)> pair. The set of all output pairs forms a simple inverted index.

```
reduce(String key, Iterator values):
    // key: a word
    // values: a list of documents for the key
        List<(word, values)> invertIdx;
        for each v in values:
          v= sort(v);
        invertIdx.add(key, v);
```

Selection – by Map-Reduce

- A Relational Algebra operation
- Apply a condition C to each tuple in the relation and produce as output only those tuples that satisfy C. The result of this selection is denoted $\sigma_C(R)$.

MR Solution

- Selections do not need the full power of MapReduce
  - Done most conveniently in the map portion alone
  - Also possible in reduce portion alone

- The **Map Function**: For each tuple t in R, test if it satisfies C. If so, produce the key-value pair (t, t). That is, both the key and value are t.

- The **Reduce Function**: The Reduce function is the identity. It simply passes each key-value pair to the output.

MR Solution

- Note that the output is not exactly a relation, because it has key-value pairs.

- However, a relation can be obtained by using only the value components (or only the key components) of the output.

Projection

- For some subset S of the attributes of the relation, produce from each tuple only the components for the attributes in S. The result of this projection is denoted $\pi_S(R)$.

- Projection is performed similarly to selection,
  - because projection may cause the same tuple to appear several times, Reduce function must eliminate duplicates.

MR Solution

- The **Map Function**: For each tuple t in R, construct a tuple t′ by eliminating from t those components whose attributes are not in S. Output the key-value pair (t′, t′).

- The **Reduce Function**: For each key t′ produced by any of the Map tasks, there will be one or more key-value pairs (t′, t′). The Reduce function turns (t′, [t′, t′, . . . , t′]) into (t′, t′), so it produces exactly one pair (t′, t′) for this key t′.

MR Solution

- The Reduce operation is duplicate elimination.
-
- This operation is associative and commutative, so a combiner associated with each Map task can eliminate duplicates produced locally.

- However, the Reduce tasks are still needed to eliminate two identical tuples coming from different Map tasks.

Matrix Multiplication

- M is a matrix with element m_{ij} in row i and column j
- N is a matrix with element n_{jk} in row j and column k
- product P = MN is the matrix P with element p_{ik} in row i and column k, where

$$p_{ik} = \sum_{j} m_{ij} n_{jk}$$

- It is required that the number of columns of M equals the number of rows of N, so the sum over j makes sense.

Storage

- A matrix is considered a relation with three attributes: the row number, the column number, and the value in that row and column.
- Matrix M as a relation M(I, J, V), with tuples (i, j, m_{ij})
- Matrix N as a relation N(J,K,W), with tuples (j, k, n_{jk})
- Very Good representation for Very large very Sparse Matrix
  - Think of the Web structure with 10 billion pages and each page with 10 out-links on average : only 1 in a billion entry is 1.
- For very large dense matrix, multiplication may be computationally infeasible and impractical.

MR Solution

- **First stage**:
  - The **Map Function**: For each of the M matrix element m_{ij}, produce the key value pair (j, (M, i, m_{ij})). Likewise, for each N matrix element n_{jk}, produce the key value pair (j, (N, k, njk)). Note that M and N in the values are not the matrices themselves. Rather they are names of the matrices or better, a bit indicating whether the element comes from M or N.
  - The **Reduce Function**: For each key j, examine its list of associated values. For each value that comes from M, say ((M, i, mij)), and each value that comes from N, say ((N, k, njk)), produce a key-value pair with key equal to (i, k) and value equal to the product of these elements, mij*njk.

MR Solution

- **Second Stage:** perform a grouping and aggregation by another MapReduce operation.

  - The **Map Function**: This function is just the identity. That is, for every input element with key (i, k) and value v, produce exactly this key-value pair.

  - The **Reduce Function**: For each key (i, k), produce the sum of the list of values associated with this key. The result is a pair ((i, k), v), where v is the value of the element in row i and column k of the matrix P = MN.

MM with one MR

- It may be required to use only a single MapReduce pass to perform matrix multiplication P = MN.
- Possible if we put more work into the two functions.
- Use the Map function to create the sets of matrix elements that are needed to compute each element of the answer P.
- An element of M or N contributes to many elements of the result.
  - one input element will be turned into many key-value pairs.
  - The keys will be pairs (i, k), where i is a row of M and k is a column of N.

Map Function

- For each element m_{ij} of M, produce all the key-value pairs
  - $((i, k), (M, j, m_{ij}))$ for k = 1, 2, . . ., up to the number of columns of N.

- For each element n_{jk} of N, produce all the key-value pairs
  - $((i, k), (N, j, n_{jk}))$ for i = 1, 2, . . ., up to the number of rows of M.

- M and N are really bits to tell which of the two relations a value comes from.

Reduce Function

- Each key (i, k) will have an associated list with all the values (M, j,mij) and (N, j, njk), for all possible values of j.
- The Reduce function connects the two values on the list that have the same value of j, for each j.
- Sort by j the values that begin with M and sort by j the values that begin with N, in separate lists.
- The j-th values on each list having their third components, m_{ij} and n_{jk} extracted and multiplied.
- These products are summed and the result is paired with (i, k) in the output of the Reduce function.

Factors affecting the efficiency of MR

- Computational time
- Data Movement time / bandwidth
- Storage Plan
- No. of Maps
- No. of Reduces

Coordination by Master Node

- Task Status : (idle, in-progress, completed)
- Idle tasks get scheduled as workers become available
- When a map task completes, it sends the Master the location and sizes of its r intermediate files, one for each reduce task.
- Master pushes this info to the reducers
- Master pings workers periodically to detect failures

Dealing with failures

- Map worker failure
  - Map tasks completed or in progress at worker are set to idle
  - Idle tasks eventually re-scheduled to other workers
- Reduce worker failure
  - Only in-progress tasks are reset to idle
  - Idle reduce tasks are restarted on other workers
- Master failure
  - MR task is aborted and client notified

Partition function

- Want to control how keys get partitioned
  - The set of keys that go to a single reduce worker
- System uses a default partition function
  - Hash(key) % r
- Sometimes useful to override the hash function
  - E.g., hash(hostname(URL)) % r ensures URLs from a host end up in the same output file

Distributed Sort

- **Problem:** Sort the records on basis of a key.
- **Map Function:** The map function extracts the key from each record, and emits a <key, record> pair.
- **Reduce Function:** The reduce function emits all pairs unchanged. This function is based on partitioning facilities and ordering properties.
- **Partitioning Function :** The users of MapReduce specify the number of reduce tasks/output files that they desire (R) and data gets partitioned across these tasks using a partitioning function on the intermediate key.
- A default partitioning function is provided that uses hashing (e.g. "hash(key) mod R").

- **Ordering Guarantees:** We guarantee that within a given partition, the intermediate key/value pairs are processed in increasing key order.
- This ordering guarantee makes it easy to generate a sorted output file per partition.
- This is useful when the output file format needs to support efficient random access lookups by key, or users of the output find it convenientto have the data sorted.

Extensions to MR

- Workflow Systems
  - Clustera
  - Hyracks

- Recursive tasks are usually handled by iterative MR

- Pregel

THANK YOU !