DSC 102
Systems for Scalable Analytics

Arun Kumar

Review 2

Time tip: Roughly 45sec to 1min per 1pt
Q1) [4pts] Suppose you want to build a deep net for a prediction task. You decide to explore neural architectures that have up to 4 layers, with each layer having a possible 500 or 1000 neurons. You use SGD for training and tune 3 hyper-parameters: learning rate, regularization, and batch size. Suppose you try 4 values for each hyper-parameter using the grid search method.

What is the total number of models trained?

4 layers with 2 # neurons each: $2 \times 2 \times 2 \times 2$
3 hyppars with 4 values each: $4 \times 4 \times 4$
So, total combos of models: $2^4 \times 4^3 = 1024$
Q2) You are given a large $n \times d$ matrix on HDFS that is uniformly sharded column-wise across $k$ workers.

A. **[7pts]** Write a MapReduce job (pseudocode or precise prose) to obtain the row sums vector of the matrix (aka rowSums in R). It should be scalable along the number of columns.

B. **[3pts]** What is the minimum possible communication cost (in big O notation of $n$, $d$, and $k$) to get the row sums vector using MapReduce?

A. **InputSplit:** Same as the column-wise shards given.

**Map():** Do a vector sum of all the columns in the input shard. Emit the partial rowSums vector as value with a single global dummy key.

**Reduce():** Gets an iterator with all partial rowSums vectors from mappers; vector sum of all of them; emit that as final output with no key.

B. $O(nk)$ because each of the $k$ workers send a state of size $O(n)$; note that this cost is independent of $d$. 
(Advanced; Optional) Q3) [12pts] Given a large n x d matrix on HDFS propose a sharding and write a MapReduce job to obtain its columnar sums vector (aka `colSums` in R). The layout and job should both be scalable along both the number of rows and number of columns.

InputSplit: Tile-wise sharding with tile shape as a fixed known hyperparameter (e.g., 1000 x 1000) and one tile per shard. So, each shard has (tile-row, tile-col) indices associated with it in the whole matrix.

Map(): Do a `colSums` over all the rows in the input tile/shard. Emit this partial vector as value with the tile’s tile-col index value as the key.

Reduce(): Key is a tile-col index in the matrix. Gets an iterator with all partial `colSums` vectors from mappers that got a tile that had that tile-col index. Vector sum of all of the partial vectors; emit that as final output value with of `ColSums` for all the columns that tile-col as the key.

So, the output is itself written out in a sharded form by multiple reducers.
Q4) [10pts] You are given a large table Ratings (RID, Stars, Date, MovieID, UserID) from a recommender system stored on HDFS. You want to know the average rating (Stars) for each movie. Propose a sharding and write a MapReduce job (pseudocode or precise prose) to obtain the above. It should be scalable along the both the number of tuples and the number of movies.

InputSplit: Tuple-wise sharding, with multiple tuples per shard but shards are not so big as to not fit in a mapper’s memory.

Map(): Create a hash table data structure with key being MovieID and value being the 2-tuple (partial sum of Stars, partial count). Go through rows in the input shard to populate this hash table. At the end, for each record in hash table do an emit with key being its MovieID and value being its 2-tuple.

Reduce(): Key is a MovieID value. Gets an iterator with all partial 2-tuples of sums and counts. Vector sum of all of them to get global Sum and global count of that MovieID; divide sum by count to get average; emit that as final output value with with that MovieID as the key.

So, the output is itself written out in a sharded form by multiple reducers.
Q5) Consider the same table Ratings from Q4. Suppose you are now given it is a tuple-wise sharded file on HDFS. You aim to binarize the Stars attribute as part of your ML data preparation to cast the recommendation/ratings prediction task as binary classification.

**A. [7pts]** First you want to see the histogram of Stars values. Write a MapReduce job (pseudocode or precise prose) to get this. It should be scalable along the number of tuples.

No histogram bins or widths given; so, just get count of each unique value

InputSplit: Same as the tuple-wise shards given, with multiple tuples per shard but shards are not so big as to not fit in a mapper’s memory.

Map(): Create a hash table data structure with key being Stars and value being the partial count. Go through rows in the input shard to populate this hash table. At the end, emit this hash table as value with a single global dummy key.

Reduce(): Gets an iterator with all partial hash tables from mappers; unify the hash tables into one based on the keys to add up the partial counts by key/each Stars value—this gives you the full histogram. Emit that as the final output with no key.
Q5) Consider the same table Ratings from Q4. Suppose you are now given it is a tuple-wise sharded file on HDFS. You aim to binarize the Stars attribute as part of your ML data preparation to cast the recommendation/ratings prediction task as binary classification.

B. [6pts] Now based on the histogram you decide to binarize the Stars attribute with a threshold of 3.5, i.e., Stars >= 3.5 is set to +1; Stars < 3.5 is set to -1. Write a MapReduce job (pseudocode or precise prose) to do this. It should be scalable along the number of tuples.

This is a Map-only job because no global agg. is needed.

InputSplit: Same as part A.

Map(): For each row in the input shard, copy its tuple and edits Stars value as per thresholding rule given, and emit it as value with the same tuple ID as key as in the input.
Q5) Consider the same table Ratings from Q4. Suppose you are now given it is a tuple-wise sharded file on HDFS. You aim to binarize the Stars attribute as part of your ML data preparation to cast the recommendation/ratings prediction task as binary classification.

C. [6pts] Suppose the table was ETL’d into Spark and queryable using SparkSQL. Now do both A and B above as a single SQL query each.

**SQL query for part A:**
```
SELECT Stars, COUNT(*) FROM Ratings GROUP BY Stars
```

**SQL query for part B:**
```
CREATE TABLE BinarizedRatings AS
    ((SELECT RID, 1, Date, MovieID, UserID
    FROM Ratings WHERE Stars >= 3.5)
    UNION
    (SELECT RID, -1, Date, MovieID, UserID
    FROM Ratings WHERE Stars < 3.5))
```
Q6) [4pts] Consider a company exploring a full move to AWS for their large-scale data analytics workloads and figuring out the economics of this move.

The capital cost of getting an on-premise cluster is $300mil. The operational cost of running and maintaining the on-premise cluster is $1mil per month. The quote from AWS on the monthly cost of running their workloads on the cloud is $5mil per month. Assume neither of these monthly costs change over time.

For how long will the cloud-native approach beat the cost of the on-premise approach?

Say, in $z$ months
Cloud-native cost = 5$z$ mil vs On-prem cost = $300$mil + 1mil * $z$
So, $z = 75$ months
Q7) [6pts] You are given a large-scale data analytics workload that exhibits sub-linear speedup on a multi-worker cluster. Its completion time with 4 workers is 80 time units. Its speedup curve from 1 to 20 workers is a straight line with a slope of 0.75. What is the workload’s completion time on 20 workers? Note that the speedup curve's origin is at (1 worker, 1x speedup), i.e., there is no intercept.

Time on 4 workers = 80; so time on 1 worker = 80 * (1+3*0.75)

So, time on 20 workers = 80 * (1+3*0.75) / (1+19*0.75) = 17.05 time units
Q8) [6pts] Consider this task graph, a forest of \( n \) pairwise dependent tasks. The work times of each task are also as listed. Note that \( n \) and \( k \) are positive integer variables, and \( n \) is a multiple of \( e \). The parameters \((a, b, c, d, e)\) are \((1, 3, 5, 2, 8)\) respectively. You are given a homogeneous cluster with \( n/2 \) nodes. What is the lowest possible completion time when using pure task-parallelism? Note that BSP or hybrid approaches are not allowed.

First visualize task lengths on graph

2n tasks but only \( n/2 \) workers
Degree of parallelism from graph is only \( n \)
Goal is to pack stuff onto \( n/2 \) workers as tightly as possible to avoid idle times

First batch: we have \( n/2 \) tasks each of length \( k \) & \( 3k \); just assign one of each to each worker; so, each worker runs exactly for \( 4k \); no idle times at all!

Next batch: we have \( n/2 \) tasks each of length \( 5k \) & \( 2k \); just assign one of each to each worker; so, each worker runs exactly for \( 7k \); no idle times again!

Overall, \( 4k + 7k = 11k \) with no idle times anywhere! Mathematically impossible to do better than this. :)
**Q9) [5pts]** You are given a database with instances of two relations \( R(A,B) \) and \( S(B,C,D,E) \), wherein \( B \) is a primary key in \( S \) and foreign key in \( R \). The set of values of \( B \) in both relations are identical. All attributes in this database are of integer datatype (4 bytes each). All tables are stored in column store format without compression on disk. The number of pages on disk of \( R \) and \( S \) are 28 million and 4 million, respectively.

What is the rough disk I/O cost (in pages) of the following query? Exclude output write costs. Assume the DRAM cache is initially empty.

\[
\text{SELECT MAX(C) FROM S;}
\]

Only \( C \) is needed; since it is column store: \((1/4)*4\text{mil} = 1\text{mil}\)
Q10) [5pts] You are given a large float64 matrix $M$ of dimensions 25 million x 10 million stored in tiled layout without compression. The tiles are squares of dimensions 2000 x 2000. They stride along both dimensions by 2000 cells starting from top left. Assume a tile’s data fits exactly on one page on disk.

What is the rough minimum disk I/O cost (in pages) of the following linear algebra computation? Exclude output write costs. Assume the DRAM cache is initially empty.

**Summation of $M[1 : 5,000,000][2,000,001 : 4,000,000]$**

Note that $M[i:j][k:l]$ means only rows $i$ to $j$ and columns $k$ to $l$ (ends included) are read. Row/column indices start from 1.

Only some of the tiles are needed. Since hyppar of tile shape is 2000 x 2000 and selection indices on both rows and columns neatly align with tile boundaries, the actual number of tiles to read for summation is only:

$$= (5\text{mil} / 2000) * ((4\text{mil} - 2\text{mil}) / 2000) = 2.5\text{mil}$$