Exercise

Q1) [2pts] Which of the following model families has complex non-sequential data access patterns for training?

A. Tree-based methods
B. GLMs trained with BGD
C. K-Means clustering
D. Both A and B
E. All of A to C
F. None of the above
Exercise

Q2) [2pts] Which of the following ML model families is XGBoost primarily designed for?

A. GLMs
B. Tree-based methods
C. Deep learning
D. Bayesian networks
E. All of A to D
F. None of the above
Q3) [2pts] Which of the following ML systems was designed for the Hadoop/MapReduce backend?

A. MADlib
B. Spark ML/MLlib
C. Mahout
D. TensorFlow
E. Dask
F. None of the above
Q4) [2pts] Which function in the RDBMS UDA API roughly corresponds to the role of Reduce in MapReduce?

A. Initialize
B. Transition
C. Merge
D. Finalize
E. Both A and C
F. None of the above
Q5) [6pts] Briefly discuss 2 advantages of in-RDBMS ML over Parameter Server.

Many correct answers possible, e.g., any of these are okay:

a) Manageability in terms of governance, security, and authentication is easier vs handling exported data files on separate clusters

b) Some native ETL support with SQL vs no support for ETL

c) UDA is reproducible vs uncontrollable asynchronous update orders and race conditions leading to unreproducibility

d) More suitable for small clusters due to low communication overheads
Exercise

Q6) [6pts] Briefly discuss 2 advantages of Parameter Server over Spark ML.

Q7) [6pts] Briefly discuss 2 disadvantages of in-RDBMS ML over Spark ML.

Just like Q5, many correct answers possible based on the pros and cons discussed in the lecture slides/videos. Left as homework for you to synthesize the answers. :)
Exercise

Q8) [6pts] Briefly discuss 2 reasons why SGD has become the optimization procedure of choice in large-scale ML.

a) Better convergence efficiency: SGD performs orders of magnitude more updates to the model parameters compared to batch methods. So, one can often converge to a good ML accuracy in fewer epochs.

b) Works well for highly non-convex loss functions too, especially DL, which is common in large-scale ML. Batch methods fail to generalize well for DL.

c) (Also okay) Easier to implement than batch methods; so, many more robust and scalable implementations exist for SGD than batch methods for ML users to adopt.
Exercise

Q9) [8pts] Briefly discuss 2 systems-level advances made in XGBoost to improve scalability and/or efficiency of the training process and why they are effective.

a) Column blocks-based partitioning. Enables better use of multi-core/multi-node parallelism to reduce runtime.
b) Cache-aware updates for g and h stats. Fewer cache spills/memory stalls keeps CPU busier and reduce runtime.
c) Sharding of column blocks. Enables better scaling to disk-resident data without thrashing.
d) (Also okay) Pre-sorted columns. Reduces time complexity by amortizing this cost for computing stats during splits.
Q10) [10pts] Briefly explain the core abstractions of 2 ML systems that were designed primarily to mitigate the developability concerns of large-scale ML.

Any of MADlib/Bismarck, Hadoop-MapReduce, Spark, GraphLab, or Parameter Server are okay.
See the respective lecture slides/videos for their abstractions, the 4-function UDA API of MADlib/Bismarck, the 2-function MapReduce API of Hadoop, the RDD API of Spark, the 3-function API of GraphLab and 2-function API + consistency choices of Parameter Server.
Q11) [10pts] Write pseudocode for a MapReduce job to compute the column-wise sums of a given large matrix. Make sure to explain your assumption on how the dataset is stored/sharded to begin with.

Many correct answers are possible depending on how the data is sharded: row-wise, column-wise, or tiled.

Suppose the matrix is sharded row-wise. Then each Mapper gets as input a data partition with multiple rows. Let Vector be a vector data type.

**map** (String dataFile, String shard)
- Parse shard into rows Vector[]
- Vector partialSum = [0]
- For each row in rows:
  - partialSum += rows
- emit (partialSum)

**reduce** (Iterator partialSums)
- Vector colSums = [0]
- For each vec in partialSums:
  - colSums += vec
- emit (colSums)
Q11) [10pts] Write pseudocode for a MapReduce job to compute the column-wise sums of a given large matrix. Make sure to explain your assumption on how the dataset is stored/sharded to begin with.

Suppose the matrix is sharded column-wise. Then each Mapper gets as input a data partition with multiple columns, with all rows per column.

map (String dataFile, String shard)
   Parse shard into cols Vector[]
   (start,end) = Get min and max column IDs from shard
   Vector colSums = []
   For each col in cols:
      colSums.append(cols.sum())
   emit ((start,colSums))

reduce (Iterator sumStructs)
   Vector allColSums = []
   Sort sumStructs by start
   For each struct in sumStructs:
      allColSums.append(struct.colSums)
   emit (allColSums)
Exercise

Q11) [10pts] Write pseudocode for a MapReduce job to compute the column-wise sums of a given large matrix. Make sure to explain your assumption on how the dataset is stored/sharded to begin with.

Suppose the matrix is sharded column-wise. Then each Mapper gets as input a data partition with multiple columns, with all rows per column. If we do not mind the column sums being output as a sharded file, a Map-only job suffices. That is, the reduce-side concatenation is not really needed.

```java
map (String dataFile, String shard)
    Parse shard into cols Vector[]
    (start,end) = Get min and max column IDs from shard
    Vector colSums = []
    For each col in cols:
        colSums.append(cols.sum())
    emit ((start,end),colSums)
```
Q11) [10pts] Write pseudocode for a MapReduce job to compute the column-wise sums of a given large matrix. Make sure to explain your assumption on how the dataset is stored/sharded to begin with.

Suppose the matrix is sharded as tiles with each tile having some rows and some columns. Assume the tile lengths and widths are fixed across the matrix. Let each Mapper get as input a data partition with one tile. The output is a sharded file.

```java
map (String dataFile, String shard) {
    Parse shard into rows Vector[]
    (start,end) = Get min and max column IDs from shard
    Vector partialSum = []
    For each row in rows:
        partialSum += rows
    emit ((start,end),partialSum)
}

reduce (Pair<Int> IDs, Iterator partSums) {
    Vector partSums = [0]
    For each vec in partSums:
        colSums += vec
    emit (IDs, colSums)
}
```
Q12) [10pts] Assume you are given a large matrix stored as a table with rows as tuples and columns as attributes. Write pseudocode for a UDA to compute the column-wise sums of the matrix. Make sure to explain your aggregation state.

We are already given rows are tuples and columns are attributes. So, unlike the MapReduce question there is only one possible approach.

**Agg. State:** Vector partialSums [ ] of length the same as the table’s arity

- **initialize():** partialSums = [0]
- **transition(Tuple t):**
  
  For i = 0 to arity - 1:
  
  partialSums[i] += t[i]

- **merge(Vector partialSums, Vector v):**
  
  Return partialSums + v

- **finalize():** Return partialSums
Q13) [6pts] Suppose you are using SGD to train an ML model on a large dataset. You are given that the shuffle step to randomize the data order takes 5min, while running an epoch of SGD takes 2min. Running SGD with shuffles before every epoch takes 20 epochs to converge, while running SGD with only one shuffle upfront takes 30 epochs to reach the same accuracy. Which approach among the above two is faster from a total runtime standpoint?

Shuffle-Always takes $20 \times (5\text{min} + 2\text{min}) = 140\text{min}$
Shuffle-Once takes $5\text{min} + 30 \times 2\text{min} = 65\text{min}$
Shuffle-Once is faster overall.
Q14) True or False [2pts each]:

A. It is always possible for Parameter Server to be faster than a single-node scalable system for SGD if the former is given enough worker nodes. **False. Communication costs of PS are non-trivial and can make it slower.**

B. XGBoost makes no algorithmic modifications to the standard GBDT algorithm published before that. **False.**

C. Model averaging-based SGD can work reasonably well for training convex models such as GLMs. **True.**