CS 6240: Project

The time has come to solve a larger data-analysis problem. You can work in teams of size **up to four**. The larger the team, the more options you have for addressing interesting problems. Of course, larger teams will have to deliver more results than smaller teams. Please finalize the process of finding team mates ASAP. **We strongly discourage you from working alone.** In our experience you learn more in a team by being able to discuss and collaborate with others, and it tends to be less work per person as data preparation and report writing tasks can be shared.

Each team must create all deliverables from scratch. In particular, it is not allowed to copy another team's code or text and modify it. If you use publicly available code or text, you need to **cite the source** in your report!

Key project events:

- Now: complete team formation and decide about your project data and tasks ASAP.
- This is followed by a progress report about midway into the project, which will discuss preliminary results and next steps.
- Final project reports will be due in the last week of classes, right before Final Exam week.
- Each team presents their project orally in class during Final Exam week.

The intermediate progress report is worth 10%, the final report 20%, and the final presentation the remaining 10% of the overall homework score. For all project deliverables, the usual 1%-per-hour late submission policy applies. **For the presentation, please make sure that all team members will attend all presentations and participate in your team's presentation.**

Project Requirements

Make sure you have addressed all the points below at least 2 weeks before the intermediate report is due:

1. Who are the members of your team (at most four)?
2. Which dataset(s) are you working with? You can choose **any** reasonable data (more info below).
    a. Make sure the data is accessible. To be on the safe side, download it immediately (if possible).
    b. Familiarize yourself with the high-level properties of the data. A good self-check is to try and explain the data to a team mate in a couple of sentences, e.g., "the dataset contains cloud cover reports from hundreds of stations located on land and at sea. It is stored in CSV format. There are about 400 million records, each with 20 attributes describing properties such as latitude, longitude, time, and cloud cover at various altitudes. The dataset spans over 40 years."

3. Choose the project tasks **from the list in the end of this document**. Do not invent your own project tasks. (Exception: students already working on research projects may select challenges related to that research. This requires instructor approval.) **The number of project tasks must match team size**.
    a. You may choose an interesting overarching project goal, which requires tasks from the list in the end of the document for solving it. Here is a hypothetical example for a 2-person team:
        i. Overarching goal: We want to verify if the small-world property holds on Wikipedia. To do so, we need to measure the distances between all pairs of Wikipedia pages.
        ii. Main task (primary): Compute all-pairs shortest path using Spark Scala.
        iii. Main task (secondary): We plan to implement all-pairs shortest path by running single-source shortest path for all possible start pages. With millions of Wikipedia pages, running single-source shortest path for all start pages would be too expensive (quadratic complexity in number of pages). And we also need a small graph for testing and debugging. A simple random sample will not work well, hence we need to design a special graph sampling algorithm.
        iv. Helper task: Write a parser that converts the Wikipedia pages into a graph representation.

You have some degree of freedom in choosing your project, but you need to make sure of the following:

- You should work with "sufficiently" large data. If the input data fits into the memory of a single instance on EMR, your project could still qualify if you are dealing with large intermediate results or are analyzing many versions or combinations of the original small data.
- Choose a problem that requires MapReduce or Spark. If everything can be done quickly and easily on your laptop, then you probably picked the wrong problem.
- As a rule of thumb, if a reasonably efficient single-machine implementation takes at least 15 minutes on your laptop, then your problem is large enough.

In the end, we expect one major project task per team member (but note that harder challenges count as multiple project tasks!), plus several smaller helper jobs per team. **You need to choose the major tasks from the list at the end of this document.**

Under certain circumstances, e.g., advanced students working on a thesis or interested in a more research-oriented challenge, the instructor might approve alternative project tasks. **If you are planning to work on tasks not listed in the end of this document, contact the instructor ASAP—at least 2 weeks before the intermediate report deadline.**

Important Issues

This is not a textbook exercise! You will attempt something that, depending on your choice of data, possibly nobody, including the instructor and TAs, has tried before. Welcome to the real world! You need to be aware of the potential risks:

- **You have to think critically about everything you are doing.** Does your approach make sense? For example, assume you are working on shortest-path and it turns out that your graph only has 100 nodes and 1000 edges. For such a tiny graph, running a MapReduce or Spark implementation of shortest-path on 20 machines does not make sense. Similarly, it might turn out that the graph has 1 billion nodes and your shortest-path program would not finish in 2 weeks, even with 1000 machines used.
- Make sure you carefully monitor your jobs on AWS to avoid high charges.
- If you are running into any major problems, including "I am spending too much money on AWS," "I really cannot figure this API out", or "my team mates are not helping at all", make sure you communicate these issues as soon as possible!

When deciding about the project, pay particular attention to the data. Make sure the data is available. Read the documentation to understand the data format and if the data set contains the right information you need for your project tasks. When deciding about project tasks, choose something you actually find interesting or care about. This way you will enjoy the project a lot more.

Suggested Approach for Dealing with Project Uncertainty

Think of this as a real-world challenge where ***you*** have to make decisions. Sometimes the best possible program still cannot scale to a given dataset, no matter the number of machines. Sometimes it might scale, but it would need so many machines for the given data that cost becomes prohibitive. **If you have a good reason why you cannot work with the full data set, then it is perfectly fine to work with a smaller version.** In general, the following approach tends to work well:

1. Create a reasonable first version of the program. Test it locally on a very small data set.
2. Think about the running time you would expect on larger data. Do you expect good scalability?
3. Then check how your program actually behaves by increasing data size, e.g., ten-fold, and measuring how this affects running time. Repeat this a few times for larger and larger subsets of the input data.
4. Based on this analysis and experiment, you should be able to (1) determine if you can realistically hope to deal with the full data set using your current approach and (2) identify potential inefficiencies and bottlenecks in your program.
5. Then you can decide on the largest data size for AWS runs and/or focus your efforts on program improvements.

Make sure you document your analysis so that you can include a brief discussion of it in the final report. In the end you want to demonstrate that you are able to carefully analyze the problem and find good solutions based on this analysis.

Data Suggestions

All data suggestions come without any guarantee that the links still work or the data is accessible. (They did work when we tried at some point in the recent past, but things on the Internet have a tendency to change.) When choosing a dataset, make sure you can access it and that there is sufficient documentation to understand the structure and meaning of the data. It is also your responsibility to determine if there are any legal constraints that would prevent you from using the data for your project. **Inclusion of the links in this document does not mean that we endorse the use of these datasets in any way.**

Dataset	Location	Comments
Variety of data sets of different sizes	http://socialcomputing.asu.edu/pages/datasets	
Stanford Large Network Dataset Collection	https://snap.stanford.edu/data/	There are some huge graphs.
Variety of public data sets on Amazon's AWS	http://aws.amazon.com/datasets/	Some of these datasets might require non-trivial steps to copy them to S3.
On-time performance of flights in the US, published by the Bureau of Transportation Statistics	http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time	
Various datasets, including Twitter and DBLP at UIUC	https://wiki.illinois.edu/wiki/display/forward/SoftwareDatasets	
Twitter dump (1 month)	https://wiki.cites.illinois.edu/wiki/display/forward/Dataset-UDI-TwitterCrawl-Aug2012	This data collection contains actual tweets, i.e., the texts posted by users, not just follower information.
Million Song data set	http://labrosa.ee.columbia.edu/millionsong/	There are also a few complementary data sets with music ratings and song metadata.
Wikipedia data	http://en.wikipedia.org/wiki/Wikipedia:Database_download	We can share parser software (no warranty!) that may help you extract graph structure from the raw data.
Wikimedia downloads	http://dumps.wikimedia.org/	
TPC-H benchmark	http://www.tpc.org/tpch/	You can create large benchmarks.

World Bank data	http://data.worldbank.org/	Many data sets are comparably small. Interesting Big Data challenges could arise from joining many of them.
Yahoo! Music ratings	http://webscope.sandbox.yahoo.com/catalog.php?datatype=c	
Other Yahoo! Data collections	http://webscope.sandbox.yahoo.com/catalog.php	
Stack Exchange Data Dump	http://blog.stackexchange.com/category/cc-wiki-dump/	There are many ways to use this for graph analysis, clustering, and prediction problems. It might be necessary to write code to convert from XML to a data format like CSV.
Netflix Prize data set	May not be available any more.	
World-wide reports of cloud measurements	http://cdiac.ornl.gov/ftp/ndp026c/	
Open Library Project	https://openlibrary.org/data	There is data about editions of books and their authors.
Book reviews and ratings	http://www2.informatik.uni-freiburg.de/~cziegler/BX/	This data set could be used in combination with the above Open Library data.

Choices of Major Project Tasks

When you look at a dataset, **think of exciting questions first**. Then try to see which of the major tasks below can be used to answer your questions. Feel free to approach the instructor with suggestions for new project tasks, but do not use them for your project without prior approval..

For all tasks, think carefully about how to make your code as scalable as possible. In MapReduce, decide if you need custom Partitioners and choose keys wisely in order to distribute the work well over many machines. Then analyze your program's scalability experimentally by running it on clusters of different size. Always evaluate critically how many machines you really need. For instance, if a data set is small and the computation cost is low, then you might be better off with a single machine.

For all challenges below, a "reasonable" program gets you up to 90% of the score. To get 100%, find a clever solution, discuss why you believe your solution will be hard to beat, and/or include some relevant analysis (e.g., of total network traffic). For example, a careful choice of partitioning granularity or

number of Reduce tasks, together with an analysis or experiments justifying that choice, would be a good way to get the remaining 10%.

~~Crossed-out tasks cannot be selected. We left them there so that you see more examples of what would have been an acceptable task, had we not already done it in the homework.~~

Single-source shortest path (MapReduce or Spark): If you use Spark, you must implement the algorithm "from scratch," i.e., without use of special graph libraries like GraphX. For a clever solution, explore optimizations to improve the performance, e.g., to reduce the number of useless computations performed. Compare the performance with these optimizations versus the performance without them.

All-pairs shortest path (MapReduce or Spark): If you use Spark, you must implement the algorithm "from scratch," i.e., without use of special graph libraries like GraphX. For a clever solution, explore optimizations to improve the performance, e.g., to reduce the number of useless computations performed. Compare the performance with these optimizations versus the performance without them. This counts as two project tasks.

Graph sampling (MapReduce or Spark): If you use Spark, you must implement the algorithm "from scratch," i.e., without use of special graph libraries like GraphX. Sample nodes and edges from a large graph to generate smaller data to be used for one or more of the other graph analysis tasks. This cannot use simple random sampling, but should be informed by the specific requirements of the graph analysis ptask. (Simple random sampling tends to destroy a lot of the graph connections, heavily affecting PageRank, shortest path and other computations.) Also, to generate good test data for some algorithms, additional constraints might need to be met. For instance, for PageRank we would want both dangling and non-dangling nodes. *In addition to the sampling job itself, you also need to write a parallel program to check the connectedness of the sampled graph* by finding all connected components and outputting the size of each component (number of nodes, number of edges).

Interesting structures in a graph (MapReduce or Spark): Find interesting patterns in a large graph, e.g., cycles of length k>3. A cycle of length k is a set of k edges (x_1,x_2), (x_2,x_3),…, (x_{k-1},x_k), (x_k,x_1), where all x_i are distinct. For a clever solution, design an efficient algorithm for finding matching edges or add more constraints, e.g., on properties of node or edge annotations. Instead of cycles, you can also look for stars or other patterns of edges connected to each other in some special way. Be careful about the size of intermediate results—find a creative way to deal with this issue!

Graph statistics (MapReduce or Spark): If you use Spark, you must implement the algorithm "from scratch," i.e., without use of special graph libraries like GraphX. Find the largest cycle in a graph. Or find the diameter of the graph, i.e., the longest shortest path between any two nodes in the graph. This counts as two project tasks.

K-means clustering (MapReduce or Spark): If you use Spark, you must implement the algorithm "from scratch," i.e., without use of special machine learning libraries like SparkML. Compute K-means clusters for a variety of values of K and use some common cluster quality measure to find the best value for K. Implement two versions of the K-means algorithm: (1) the distributed K-means clustering algorithm that

uses multiple tasks for each iteration, and (2) the local K-means algorithm that computes an entire clustering for a single K in a single task. For approach (2), parallel computation can be achieved by exploring many starting configurations and different values of K concurrently. This counts as two project tasks for MapReduce and as four project tasks for Spark.

Hierarchical agglomerative clustering (MapReduce or Spark): If you use Spark, you must implement the algorithm "from scratch," i.e., without use of special machine learning libraries like SparkML. Choose one of the cluster distance metrics, e.g., single-link, and think carefully about effective parallelization and minimizing computation cost. This counts as two project tasks.

Measure speedup (Spark) for two non-trivial queries of the TPC-H benchmark. (TPC benchmarks come with their own data generator so that data sets of a desired size can be generated.) Explain the reasons for the observed high or low speedup.

Dense matrix product H-V (MapReduce or Spark): Implement the algorithm for dense matrices that partitions the left matrix horizontally and the right one vertically. Think about the right granularity of partitioning. For the Spark version, do not use the existing linear algebra library.

Dense matrix product V-H (MapReduce or Spark): Implement the algorithm for dense matrices that partitions the left matrix vertically and the right one horizontally. Think about the right granularity of partitioning. For the Spark version, do not use the existing linear algebra library.

Dense matrix product B-B (MapReduce or Spark): Implement the algorithm for dense matrices that partitions both input matrices into blocks. Think about the right granularity of partitioning. For the Spark version, do not use the existing linear algebra library. This counts as two project tasks.

Sparse matrix product H-V, V-H, or B-B (MapReduce or Spark): Implement any of the above matrix product algorithms, but this time supporting *sparse* matrix representation. This means that only non-empty cells should be stored in the partitions. Think about the right granularity of partitioning. For the Spark version, do not use the existing linear algebra library. For horizontal-vertical and vertical-horizontal, this counts as two project tasks, if you consider input in a block-oriented format (not just triples of row index, column index, and value). For block-partitioning, this counts as three project tasks.

Classification and prediction ensembles using existing data mining libraries (MapReduce): Train an ensemble classification or prediction model consisting of individual models from an existing library such as Weka. More precisely, you can use Weka libraries for training and prediction of *individual* models, but you must code the framework for ensemble training and prediction, and for efficient exploration of the best model parameters. Then use your ensemble model to make predictions and report its accuracy. You may apply this solution to a Kaggle competition. This counts as two project tasks.

Classification and prediction ensembles using your own local data mining algorithm implementation (MapReduce): Train an ensemble classification or prediction model consisting of individual models for which you provide your own non-parallel in-memory model training and testing functionality. To avoid spending too much time on details not related to MapReduce, simplify your task by assuming all data

attributes are real numbers. This way you do not have to deal with different data types. Then use your ensemble model to make predictions and report its accuracy. You may apply this solution to a Kaggle competition. This counts as three project tasks.

Classification and prediction in Spark MLlib: Train the most accurate model possible using (1) a single model, e.g., decision tree, and (2) and ensemble, e.g., Random Forest or boosted trees. Explore different model parameters, try to address memory problems, and report a thorough comparison of prediction quality versus running time. It is acceptable to save money by performing model-parameter exploration on your local machine, then to train and predict on AWS only for a few most promising models. You may apply this solution to a Kaggle competition.

Frequent itemset mining (MapReduce or Spark): Implement a parallel version of the Apriori frequent itemset mining algorithm. Do not use existing libraries for frequent itemset mining. This counts as three project tasks if you include the Apriori pruning step, which avoids counting for candidates with an infrequent subset.

Decision trees (MapReduce): Write a program to train a single decision tree on a huge data set in parallel. Do not use existing libraries for trees. To avoid spending too much time on details not related to MapReduce, simplify your task by assuming all data attributes are real numbers. This way you do not have to deal with different data types. This counts as four project tasks.

Nearest neighbor classifier (MapReduce): Implement the nearest-neighbor classification algorithm to predict the output for a large test set in parallel. Do not use existing libraries for nearest-neighbor classification. However, you can use HBase as a nearest-neighbor index. (If you use HBase, then this counts as four project tasks.) Or you can implement another clever algorithm for finding the nearest neighbors of a given test record. If you do not use HBase, a clever partitioning for the case when both training and test data do not fit entirely into memory, e.g., using a 1-Bucket style solution that splits both datasets, counts as two project tasks. Notice that indexing for KNN is a little tricky. Consider a dataset of numbers, where we want to find the k numbers closest to a given number x. Assume we store the set of numbers in order (HBase does this for the appropriate choice of key.) How far from x could those k nearest neighbors be? If we can guess that all those neighbors are within distance 20, then we can run a range query for range [x-20, x+20]. HBase supports efficient key-range retrieval, but we need to estimate the range. If we pick it too large, then too much data is retrieved; if we pick it too small, then fewer than k neighbors are found, and we have to request a larger range. Things get more challenging when the data has more than one dimension. You could index each dimension separately, later combining the separately retrieved results in a clever way. Or think about way to convert a multi-dimensional key to a 1-dimensional sorted order.

HBase as index (MapReduce or Spark): Use HBase as an index for an equi-join implementation: store one input relation in HBase, then use a MapReduce or Spark job that scans through the other input relation and looks for matches in the HBase table(s). Compare the performance and scalability of this approach against hash+shuffle (Reduce-side join). This counts as four project tasks.

Theta-Joins 1-Bucket (MapReduce or Spark): Implement the 1-Bucket algorithm and determine the best partitioning for minimizing running time on at least three types of problems: one where input size exceeds output size (by at least a factor of 10), one where input and output size are similar, and one where output size is at least 10 times greater than input size. This counts as two project tasks.

Theta-Joins M-Bucket (MapReduce or Spark): Implement an algorithm that partitions the join attribute domain into M buckets, e.g., based on approximate quantiles, and determines which bucket combinations must be covered. (Those are the candidate regions.) Then design a covering algorithm that minimizes input duplication while covering all candidate regions. This counts as four project tasks.

Deliverables

1. Nothing to be submitted, but work on the project ASAP to be ready for the intermediate report deadline.
2. **Make sure that you sign up as a group on Blackboard.**