CS 6240: Project Proposal

Now that we covered all the basics of MapReduce and practiced our skills, the time has come to solve an interesting data-analysis problem of your choice. You can work in teams of size two 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. (One-person teams would need to be approved by the instructor, but we generally do not recommend 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 has to 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!

Project key events:

- Project proposals are due as stated on Blackboard.
- Each team writes a short overview of their project progress and next steps, which will be due two weeks later. (Details will be announced later.)
- Final project reports will be due another two weeks later. (Details will be announced later.)
- Each team presents the final results of their project orally in class during final exam week. (Details will be announced later.)

The project proposal is worth 5% of your overall homework score. The intermediate progress report is worth 15%, the final report 30%, and the final presentation the remaining 10% of the overall homework score. For project proposal, intermediate, and final report, the usual 1%-per hour late submission policy applies. **For the presentation, please make sure that all team members will attend the entire session and participate in the presentation.**

Please submit your project proposal through the Blackboard assignment submission system and make sure you submit it as a **PDF** file. **Only one team member should submit the proposal.** We make sure that everybody will receive the credit.

Project Proposal Requirements

State the names of all team members. You have some degree of freedom in choosing your project, but you need to make sure of the following:

- You should be working with large data. If the input data is comparably small, your project could still qualify if you are dealing with large intermediate results or have to analyze many versions or combinations of the original small data.

- Choose a problem that requires MapReduce. If everything can be done quickly and easily on your laptop, then you picked the wrong problem.

In the end, we expect one major analysis job per team member, 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., a team of PhD students wants to work on a problem related to their PhD research, the instructor might approve other project tasks. If you are planning to work on tasks not listed in the end of this document, contact the instructor ASAP—at least 5 full days before the proposal submission deadline.)

Your project proposal should have the following content:

- Which data set will you work with? Where is this data set and how will you obtain it?
- Describe in about one paragraph the high-level properties of the data.
    - Example: "The data set 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 data set spans over 40 years."
- Include a few lines (or similar fragment that gives a good idea about the data) of the data set(s) you want to work with.
- What analysis do you want to perform on the data and why do you think this is an interesting problem? In a couple of sentences, describe each main analysis goal and any major sub-tasks that might be needed. Make sure you clearly highlight the major tasks you selected from the list in the end of this document, e.g., by underlining them.
    - Example of a problem with one major task for Web crawl data
        - Analysis task: "We want to compare the average page rank of 'happy' pages against 'sad' pages. Our hypothesis is that happy pages attract more links and hence tend to have a higher PageRank. We will determine if the data supports this hypothesis."
        - Main task: Compute the PageRank for every page.
        - Helper task (this could be a major task, e.g., if sample quality is checked algorithmically by looking for connected components): The data set has 5 billion pages. For initial testing we will need smaller samples. A simple random sample will not work well, hence we need to design a special graph sampling algorithm.
        - Helper task: For each page, we need to determine if it is happy or sad. We are planning to use library XYZ available at ABC.

Important Issues

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

- **You have to think critically about everything you are doing.** Does it make sense the way you do it? For example, assume you are working on PageRank and it turns out that your graph only has 100 nodes and 1000 edges. For such a tiny graph, running a MapReduce implementation of PageRank on 20 machines does not make sense. Similarly, it might turn out that the graph has 1 billion nodes and your PageRank program would not finish in 2 weeks, even with 1000 machines used. Do not just run the program and wait. Make sure you start with smaller data. As you increase data size, try to understand how running time and resource consumption changes. Then estimate what graph size you can realistically handle without spending hundreds of dollars.
- If you are running into any major problems, including "I am spending too much money on AWS", "I really cannot figure this other 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—and tell us why you find it interesting or relevant. This way you will enjoy the project a lot more. For the helper tasks, only mention the major challenges. Do not enumerate every little thing. E.g., if you just need a simple random sample, do not even mention it. If you compute an average of some numbers, do not mention it. Look at the above example to get an idea about the level of detail. The entire proposal should not exceed 3 pages in the formatting style of this document.

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 data set, make sure you can download 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 data sets in any way.

Dataset	Location	Comments
Variety of data sets of different sizes	http://socialcomputing.asu.edu/pages/datasets	
Variety of public data sets on Amazon's AWS	http://aws.amazon.com/datasets/	Some of these data sets might require non-trivial steps to copy them to S3.
On-time performance of flights in the US, published by the Bureau of	http://www.transtats.bts.gov/DL_SelectFiel	One could explore delay patterns, e.g., based on monthly or weekly

Transportation Statistics (This gives access to an extended version of the flight data we used in previous homework assignments.)	ds.asp?Table_ID=236&DB_Short_Name=On-Time	delay for each airport or airport-airline pair. Together with airport location and climate/weather data, it is possible to see if clusters of airports with similar delay patterns correlate with climate zones.
Twitter data (in particular user and follower data)	http://socialcomputing.asu.edu/datasets/Twitter http://an.kaist.ac.kr/traces/WWW2010.html	A classic problem in this domain is link prediction: will two nodes connect sometime in the near future? To solve it, one can train a data mining model on the older part of the data, then test its prediction accuracy on the newer part.
Million Song data set and a few complementary data sets with music ratings and song metadata	http://labrosa.ee.columbia.edu/millionsong/	
Wikipedia data	http://en.wikipedia.org/wiki/Wikipedia:Database_download	
Wikimedia downloads	http://dumps.wikimedia.org/	
TPC-H benchmark	http://www.tpc.org/tpch/	
World Bank data	http://data.worldbank.org/	Many data sets are comparably small. Interesting Big Data challenges could arise from combining 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	http://www.lifecrunch.biz/archives/207	
World-wide reports of cloud measurements	http://cdiac.ornl.gov/ftp/ndp026c/	

Choices of Major Tasks

Be creative. When you look at a data set, think of exciting questions first. Then try to see which of the major tasks below can be used to answer your questions.

When working with large data, it is important to understand if a task is feasible. Run it first on a small sample. Then gradually increase data size. Make sure you carefully monitor your jobs on AWS to avoid high charges.

For all tasks, think carefully about how to make your code as scalable as possible. 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 tasks a "reasonable" program gets you up to 90% of the score. To get beyond that, find a clever solution or discuss/analyze why you believe your solution will be hard to beat.

Notice that in addition to the major tasks, you will usually also need to solve minor "helper" tasks, e.g., to prepare or clean the data. We point out a few of them below. Depending on the major tasks and data you choose, the effort for minor tasks might vary. This work will also be counted toward your project score.

~~**PageRank**: this would be a major task, but PageRank is not allowed as an option this time.~~

Single-source shortest path: For a clever solution, you can 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.

Graph sampling and sample quality check: 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. Simple random sampling will usually not work well, because it might create nodes that are not connected to each other. Furthermore, 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. To check the structure of the sampled graph, find all connected components and output the size of each component (number of nodes, number of edges).

Interesting structures in a graph: Find interesting patterns in a large graph, e.g., cycles of length > 2. 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. (For example, in Assignment 3 we used a condition on flight

arrival and departure time.) Instead of cycles, you can also look for stars or other patterns of edges connected to each other in some special way.

Graph statistics: Find the largest cycle in a graph. And also find the diameter of the graph, i.e., the longest shortest path between any two nodes in the graph.

Hive: Use Hive for a task that requires a join, e.g., using a query and data from the TPC-H benchmark. Compare the performance to your own implementation in plain (Java) MapReduce. Or compare to an implementation in PigLatin.

K-means clustering: 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 machines for each iteration, and (2) the local K-means algorithm that computes the entire clustering for a single K on a single machine. For approach (2), parallel computation can be achieved by exploring many values of K concurrently. This can be applied to many data sets, e.g., to find clusters of airports with similar monthly delay patterns (as computed in Assignment 4).

Hierarchical agglomerative clustering: Choose one of the cluster distance metrics, e.g., single-link, and think carefully about minimizing computation cost and effective parallelization.

Pig Optimizer exploration: If you are familiar with database optimization, try to determine what the Pig optimizer can do. (This is somewhat similar to Assignment 3, but now you take a more comprehensive approach). Choose a query with multiple joins, some groupings, selections and projections. Think of several equivalent PigLatin programs that vary the join order and at what time projections and selections are applied. Make sure you understand which of these plans you expect to be significantly better than others. Then run them and see if Pig shows the expected outcome. You can also explicitly turn some of Pig's optimization off and see if that makes a difference for some of the programs. Suggestions:

- Come up with a query that joins at least 3 data sets (this could be the same data set participating three times in the joins) and also has some selections and projections.
- Think about different execution plans for this query, focusing on the order in which the operators are applied (see Assignment 3 for a simple example). Make sure the problem is more complex than for Assignment 3.
- While exploring different execution plans, think about which one you expect to do better than another one and why that is.
- Make a list of optimizations the Pig optimizer might possibly apply, e.g., regarding join order and applying selections/projections early. Then go back to the different execution plans and see which one of them would be improved by that optimization. In particular, try to find pairs (P1, P2) of execution plans that have the following properties: (1) without optimization, P1 would be much cheaper than P2; but (2) with optimization of type XYZ (e.g., automatically pushing selections "down"), P1 and P2 would have about the same cost. Then you can measure the

performance for these (P1, P2) pairs in Pig, allowing you to infer if the corresponding optimization was applied.

Measure speedup and scaleup, e.g., for TPC-H. (TPC benchmarks come with their own data generator so that data sets of a desired size can be generated.) Think about how to meaningfully measure scaleup.

Matrix multiplication: Support multiplications of the type $A \cdot A^T$ (matrix with its own transpose) and $A \cdot B$ (two arbitrary matrices). The program should support a sparse representation where only non-empty cells are given in an input file.

Classification and Prediction using existing data mining libraries: Train an ensemble classification or prediction model consisting of individual models from an existing package such as Weka. In particular, you can use Weka libraries for training and prediction of *individual* models, but you have to 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.

Frequent itemset mining: Implement a parallel version of the Apriori frequent itemset mining algorithm. Do not use existing libraries for frequent itemset mining.

Decision trees: 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.

Nearest neighbor classifier: 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. Or you can implement another clever algorithm for finding the nearest neighbors of a given test record.

HBase as index: Use HBase as an index for an equi-join implementation. Compare the performance and scalability for competing join implementations, e.g., nested loops, index nested loops, and sort-merge join.

Deliverables

1. The project proposal as discussed above. (1 PDF file)