Project Info

What a better way of learning doing research than start doing research yourself. For this course, you are expected to propose and work on a topic of your choice. If you are already working in the big data area, you are welcome to bring your expertise in the class and propose work relevant to your experience. The class team (the instructor + the TA) are in your disposal to discuss ideas.

Team Sizes

You can work on a project either individually or in teams up to 3 members. Team creation is strongly encouraged even with people that you have never had coffee with.

Important dates (hard)

Proposal due: Friday April 9th at 23:59.
Project deadline: You should be able to have a presentation and a final report ready by the last Wednesday of the class: Wednesday May 26th. The presentation will be in a mini-workshop format.
Project report deadline: The final write up together with the source code are due Wednesday June 2nd (A week after the final presentation)

Deliverables

For proposal:

A pdf document where you describe the problem you are solving. Why it is important and why it is hard. Who else is working on the area and what is their state of the art? For now it is OK if you cannot beat the state of the art, but you should be able to quantify your difference from it. If you are using a dataset, describe it. How big is it (size in Bytes, number of rows)? What kind of data does it contain? What is the format of the data (text, binary etc)? Do you expect any biases with the data? Print the first 5 lines of the dataset. For projects that involve Machine Learning, write how you are planning to approach the steps of your pipeline: what kind of features are you going to extract? what is your mathematical formulation of the problem, and how do you plan solving it? What is your evaluation metric? I understand that most of the time you are going to work locally with small datasets, but at some time you will have to deploy on Amazon EC2 clusters. What kind of a cluster are you going to need (i.e. NNN instances of type YYY)? For how long? How much do you expect to get charged? Don't forget to include costs associated with storage (EC2/EBS services), data transfers (EC2 mainly), and software platforms (EMR service).
For projects that involve improvements on big data frameworks, what is the problem of the framework? Is it slow (please quantify) or does it have restrictions on its functionality? How are you planning to improve it? How are you planning to evaluate your contribution? What license does it have? Is it Apache2 (so you are free to do whatever you like) or more restrictive? Since you are working on open source projects, include a link to your local fork in your github account (if you don't have one, you can create and use an unrestrictive one for free).

Final deliverables

1. Create a powerpoint presentation so you can present it in class in a mini-conference style setting (please no open office formats… just pptx, ppt, pdf).

2. Also submit a final write-up that resembles a small conference paper. The report should not exceed in size 6 double column pages. You should type it in Latex and use the template and style file from https://www.usenix.org/conferences/author-resources/paper-templates

Indicative Project Ideas

Systems projects

- **Profile Apache Spark**: Although Spark is faster than Map Reduce, it is still a slow execution engine. Unlike the common assumption that for all bottlenecks it is the network (and the Garbage Collector) to blame, there are more components that are expected to slow down execution: Too many serializations and deserializations of the data, too many abstraction layers are just a couple of examples. Also, the networking mechanism that implements shuffling, and is based on RPC calls over TCP/IP adds overheads. In this study, you will come up with 2-3 benchmark programs to quantify the bottlenecks of **shuffling**. We can provide further pointers and support to interested teams.
- **Implement your own Map – Filter -- Reduce (MFR) in either C++ or in Rust**: You are expected to use some of Spark ideas and implement your MFR framework in C++ or Rust (Rust is strongly suggested). If the implementation is in C++, the front-end of the system should be in Python (in that case you have to think how to send code/state written in Python to the back end to run). If the implementation is in Rust, it is OK for the back-end to be in the same language (Rust). You should create an RDD-like class which lazily runs execution upon the detection of an action (reduce in our case. For an easier implementation, it is OK to implement Collect or Count instead of reduce). So, the idea is for a user to create an RDD and write execution recipes: Each map or filter transformation adds a vertex to an execution graph. When a job starts running, the execution graph will run efficiently in each partition of the data: A series of transformations will load data once and will run in cascade before they produce the output for the action.
- **Modify Apache Spark to use serverless execution**: Serverless is a different execution paradigm, when users should not create about cluster management. This is different than the standard practice that you follow in your Spark deployments on AWS: You first create a number of instances, you then set them up, and then you create a Spark installation before running any code in it. In serverless, users submit code to an unknown infrastructure, without worrying about any configuration details. In this project, you should modify Spark Master to use the Kubeless framework of Kubernetes to create executors to run tasks. Executors should also be modified to store their output to a permanent storage system, such as S3, as follow up tasks will have no idea which computer to contact to obtain their input data.
- **Modify Apache Spark for the Exascale era**. In this project, you will re-design a subset of the functionality of the Spark Master by adopting a Peer to Peer architecture. Although Apache Spark has proven its value at scale, its Master-Slave architecture is going to become a bottleneck when datasets become massive. This observation is based on two factors. The first factor is based on the assumption that for the processing of orders of magnitude larger datasets, orders of magnitude larger compute clusters will be necessary. This will involve the need of an overwhelming

amount of TCP connections during execution. In Spark, executors use TCP connections with the driver for heartbeat, stat reporting, job status reporting, and obtaining partition locations for shuffling. This means, that in a cluster configuration with 1000 nodes, in each of which 10 executors run, a driver receives tens of thousands of TCP connections, which result, among others, in huge amounts of threads and expensive context switching. The second factor is the observation that the master program of systems, such as Spark, uses a list to store pointers to dataset partitions, and assign partitions to workers. However, as dataset sizes grow exponentially, those lists quickly result in unnecessary RAM overheads. For instance, a 100PB dataset gets partitioned in 1 Billion partitions of size 128MB. Thus, assuming in the best case that each pointer stores 10Bytes of data, just the partition indices of datasets consume multiples of 10GB. Note, that although today's servers can provide more than 10GB of RAM, this is at the expense of the RAM that is available for processing and cluster orchestration.

- **Modify the Spark Scheduler to leverage stochasticity of ML workloads**. Spark follows the Bulk Synchronous Parallel (BSP) model of execution. That is, a stage waits until ALL tasks of a preceding stage are complete and produce their data. This approach is prone to strangler-related delays, especially when cluster and dataset sizes are large. In this project, you are expected to modify Spark schedulers to allow ML tasks to start their execution without having to wait all tasks from the previous stage to complete. This might result in some initial loss of accuracy, but it will compensate with faster execution times.

Data Projects

- **Transform the million song database in a cloud friendly format:** Unfortunately the million song database is in binary format; so Spark and Hadoop have no idea how to partition it and they end up processing it as a single partition. You should transform each song as an AVRO object and store it as a Parquet file. Although this project is straightforward in coding, you are expected to provide a well-engineered code (complete with unit tests) which should be built with Maven. You should also transform the entire dataset into your new format.
- **Work with public datasets of AWS:**
  - https://registry.opendata.aws/
- **Start working on a big (more than 50GB big) dataset on Kaggle :**
  - **https://www.kaggle.com/datasets**
- **Process some of our own data collections**:
  - NYC taxi dataset
  - Book reviews from Amazon
  - An execution trace on a multi-thousand server cluster of Google
  - Neuroscience data (in collaboration with the group of prof. Papadopouli). The focus here will not be on data discovery, but on the speedup of existing non-linear algorithms. Note that some of these problems are not mainly Big Data problems, but Big Compute problems, and thus the use of Apache Spark might not be ideal. In prior collaborations, students managed to achieve orders of magnitude of Speedup by re-implementing some of those algorithms with OpenMP and Cuda.

- Traces of network routers (in collaboration with the group of prof. Dimitropoulos).

Dataset Sizes

If you choose to analyze a dataset, I generally expect teams to work with sizes of about 20GB-30GB per person (so a 3-member team will work with about 100GB of data).

Exceptions are:

1) *Systems projects*: The main focus of these projects is on programming/performance profiling, and it is OK to experiment with smaller datasets (around 1GB-10GB per team).
2) Projects that require the use of heavy computation (such as the neuroscience project or projects that will develop and train *Neural Networks*) are OK to work with a total of 1-2GB data. Note that projects that use publicly available pre-trained Neural Networks will follow the 20GB-30GB rule per team member.