**Head: Fast Track Apache Spark**

Subhead: 6 lessons learned to get a quick start on productivity

By: Tobi Bosede

My [Strata NYC 2017](https://conferences.oreilly.com/strata/strata-ny/public/schedule/detail/61259) talk, about big data analysis of futures trades, is based on research done under the limited funding conditions of academia. This meant that I did not have an infrastructure team, therefore I had to set up a Spark environment myself. I was analyzing futures order books from the Chicago Mercantile Exchange (CME) spanning May 2, 2016 to November 18, 2016. The CME data included extended hours trading with the follwowing fields: instrument name, maturity, date, time stamp, price, and quantity. Futures were comprised of 21 financial instruments spanning six markets - foreign exchange, metal, energy, index, bond, and agriculture Trades were recorded roughly every half second. In the process of doing this research, I learned a lot of lessons. I want to help you avoid making the mistakes I did so you can start making an immediate impact in your organization with Spark. Here are the six lessons I learned:

1. You don’t need a database or data warehouse. It is common for Spark setups to use Apache Hadoop’s distributed file system (HDFS) and Hive for querying, but you can use text files and other accepted file formats in local directories if you don’t want to go through the hassle of setting up a database or warehouse. When I worked at Sprint they had an on premise cluster setup and stored data in HDFS which provided great exposure for me to learn new technologies. However, in my research I had already been reading CSVs directly into Spark using the spark-csv package which has since then been merged into the main spark project because of the fundamental capability the package provides.

2. You don’t need a cluster of machines. You can hit the ground running using your local machine or a single server. This is also helpful in that you do not have to consider what cluster manager to install - YARN or Mesos. You can simply use the standalone cluster manager that comes with Spark. Just make sure that, if you use one machine, it has multiple cores and enough memory to cache your data. In general, any time you build a distributed system you should first start with one machine. One of my mentors once told me that software engineering is like math. To build something, it is often useful to start at n=0 as in an inductive proof and then generalize from there. That is one analogy that I can relate to!

3. Use a notebook. Don’t bother trying to configure an IDE or using the shell to write applications. Of course, using the shell is great if you are submitting an application or doing some basic coding. I used the shell when I was learning Spark to run some of the examples that come with Spark and follow along with tutorials. And in theory having all the features of an IDE might be a reflexive thing many programmers want for any coding they do. However, the pain of configuring an IDE with Spark is not worth the investment. A friend of mine once told me he had spent time struggling to setup an IDE for programming in Spark in vain. If only I had met him before then. I would have been able to share that all the programming I did in Spark for my research was either in the shell or a Jupyter notebook. The latter being used the majority of the time and so and IDE was completely unecessary. Although, every once in a while I would go old school and just use vi, a command line editor, to code.

4. Don’t know Scala? Start learning Spark in the language you *do* know - whether it be Java, Python, or R. In Spark versions 2.0+, a lot of additional support was added for R, namely in the form of SparkR and sparklyr. A lot of people are intimidated by the Spark learning curve, but language does not have to be a barrier into entry. One of the Spark core developers’ mantra is to democatize data science so it is not just for the engineer, but for the scientist, for the analyst. Making APIs available in so many languages is a deliberate effort.

5. Use DataFrames instead of RDDs (Resilient Distributed Datasets) for ease of use. There has been extensive work done between Spark versions 1.6 and 2.0 to make functionality that has traditionally been done using RDDs easier to do with DataFrames. Also, DataFrames are efficient across languages. So if you’re more comfortable with Java, Python, or R, there is no performance loss suffered by not switching to Scala. (This reinforces lesson #4 above.) I am on the Spark users mailing list and have suggested various improvements to the community. In general I try to keep abreast of latest developments in te Spark community not only through the list serv, but also by reading onine documentation and published research papers.

6. Avoid partial actions. A partial action is an action that does not allow the DAG (Directed Acyclic Graph) for a Spark job to be fully evaluated. Thus, transformations along the way do not complete. For example, one issue that plagued me for some time was why computation was slow on data I cached. Eventually, I realized by looking at the web UI that only some of the data was cached. However, I still did not know why all the data was not being cached. In fact, I knew that since cache() was a transformation, I had to call an action before it would be evaluated. The code I was running was data=df.cache().show(). However, because show() is defaulted to 20, and my data had more than 20 rows, show() was a partial action. I should have been running data=df.cache().count(), as count() will always act as a full action. So be wary of partial actions. Thanks to [Holden Karau](https://www.amazon.com/Holden-Karau/e/B00G4JS6IO/ref=ntt_dp_epwbk_0) for pointing this one out to me.

These tips highlight Spark’s ability to deliver serious gains in productivity despite limited user computing capability. There is definitely an ideal Spark setup for each organization’s particular needs. One or all of the following will most likely be necessary once there is buy in from stakeholders to create a robust analytic system: expanding to a cluster setup, building a data warehouse, and utilizing an infrastructure team. My aim with this post is twofold: 1) simply to suggest ways to make it easier to create a proof-of-concept Spark environment that justifies stakeholder investment, 2) provide some pointers if you decide that a bare bones Spark setup is best for you.