Languages of Big data

An insights into Big data technologies from a language perspective

Well Big data has quite mature now and day by day some new framework gets added to the Big data Stack. What initially started as Hadoop in 2008, now many other technologies or framework has been added to this stack. Yet predominantly hadoop has still remain one of the driving force behind big data and all other frameworks added later supports its more or less.

Well this paper will not talk about the frameworks related to Big data stack, but will throw some light on the languages to learn in order to be better at processing big data. In that process it will touch upon the various framework and mention a few words about it. But the main flow of this paper will be from prospect of programming language. So in Cognizant terminology language is horizontal whereas the framework becomes the vertical and as we will see below, the horizontal (programming lang.) will cut across many vertical.(framework – Spark, map reduce etc.).

This will address the specific programming languages importance (occasional code examples might exist) in resolving problems related to Big data processing. Some paper in future might cover these topics in details. Check the reference section for more detail understanding of topics mentioned.

1. *UNIX shell scripting - Well ideally although not a programming language in itself but everything seems incomplete without mentioning a word about it. As originally hadoop was based out of Unix (linux) so knowing shell scripting basics will definitely help. In fact hadoop file system commands are itself in some way has a flavour of Unix in it like. [Ref 1]*

    a. *hdfs dfs –ls : to list all the files in a particular directory*
    b. *Creating a chain using unix pipe commnds.eg. to count the words in the HDFS file-<filename>*
       hdfs dfs –cat <filename.dat> | wc –l

    Most probably the jobs that you will run in hadoop will be shell script with hadoop jar or spark submit command in it to submit mapreduce or spark jobs.
    So unix skills comes as top priority.The edge node from where you will do all hadoop related stuff will be a Linux box.

2. *Java - The base code of hadoop (HDFS and mapreduce) is written in java. So in case you want to contribute to Apache Hadoop open source framework or want to understand what's happening under the hood knowing Java will help.*

    But the above point is not strictly applicable if you want to work in Hadoop like in the same way you don't need to know the language(maybe C) that database is written in, similarly you don't need to know Java to work in hadoop but below points will mention why knowing it will give you an upper edge.

First the base processing framework for hadoop – mapreduce must be written in Java. [2, 3, 4]

Below shows a sample mapper example which splits words based on comma separator. Basically all mapper has to extend the Mapper abstract class and implement their own map method for the logic. -

```
public class ProjectionMapper extends Mapper<LongWritable, Text, Text,
                          LongWritable> {
 private Text word = new Text();
 private LongWritable count = new LongWritable();
 @Override
 protected void map(LongWritable key, Text value, Context context)
  throws IOException, InterruptedException {

  String[] split = value.toString().split(",");
  word.set(split[0]);
   if (split.length > 2) {
                            try {
                                count.set(Long.parseLong(split[2]));
                                context.write(word, count);
                                              }
                              catch (NumberFormatException e) {
                                                    }
                           }
                                       }
                                   }
```

Another popular framework cascading which is a library above hadoop to write code in pipeling fashion has to written in Java.Cascading has a nice flow abstraction which makes ETl tasks more logical to code than using raw mapreduce. The other advantage of choosing such kind of framework is you as a user is abstracted away from the underlying execution engine. So like currently Cascading supports – mapreduce, Tez and Spark as execution engine. So the challenges of learning each execution engine is abstracted away from the user. [5]

Most NoSQl database like HBASE, MongoDB has API to work with Java. [6]

In case you use Pig (scripting layer over hadoop), Hive (SQl layer over hadoop) and you are stuck with any functionality which the inbuilt functions of these tools cannot handle then you have to write UDFs (user defined functions) written in Java. [7, 8]

Below is a sample UDF written in Java that extends the UDF class and basically we have to write the logic in the evaluate function.

```
  class SimpleUDFExample extends UDF {

   public Text evaluate(Text input) {
     if(input == null) return null;
      return new Text("Test " + input.toString());
                                    }
                                      }
```

Spark (most active ASF project) a popular framework for processing Big data which has recently gained a lot of momentum supports java. Apache Flink has support for java and probably the list goes on. [9, 10, 33]

Sample Spark code that splits words based on separator comma. We use anonymous class in java(new FlatMapFunction)to achieve this.

```
import org.apache.spark.api.java.*;
import org.apache.spark.api.java.function.*;

 JavaSparkContext jsc = new JavaSparkContext(...);
 JavaRDD<String> lines = jsc.textFile("hdfs://...");
 JavaRDD<String> words = lines.flatMap(
  new FlatMapFunction<String, String>() {
   @Override public Iterable<String> call(String s) {
  return Arrays.asList(s.split(", "));
  }
 }
                                              )
```

With support of lambda expression in java 8 the above code can be written in a better way using below.

```
JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(s.split(" ")));
```

Solr used for indexing and search has a Java APi called SolrJ. [11]

Other than for working with Big data, Java knowledge will also help while interfacing with Big data components. For example if you want to expose some data in Big data (say some data stored in Hbase, MongoDb) to user interface/web interface most chances you might develop a restful webservice/servlet which might be written in Java (Jersey etc) [12]

Real time stream engine – Storm, message broker – Kafka has inbuilt support to write code in Java. In fact for both java comes as top priority for using their API [13, 14]

Machine learning tasks using library Mahout has to be written in java although for simple tasks probably CLI commands would be enough. [15, 16]

So Java is definitely a necessity if not must to work with Big data. You can definitely skip it if you have good command in the other languages mentioned.

What in Java will help – Core Java will do with special emphasis on - Collections, OOPs concepts/design basics, libraries related to XML, JSON (Jackson), JDBC, maven build tool

Will give and edge – Servlets, JSP, Restful web service (Jersey), multithreading concepts

3. *Python- Well Python is perhaps the oldest among the 3 languages mentioned. Although it's a dynamic scripting language so might suffer from slight performance when compared to other but it does its jobs quite efficiently. From a learning prospective perhaps Python will be easiest.*

   Python has support for writing mapreduce via framework like dumbo, MRJob. Else you can use the hadoop streaming API to run same. [17]

   It also allows you communicate to HDFS via library called Snakebite. [18]

   Spark, Flink has support for Python. In case you use Python with Spark using RDDs you might suffer a little performance overhead due to conversion of Python process to JVM but good news is if you use Dataframe (added Spark 1.3 onwards) you will have same performance whether you code in Python or the JVM based languages (Java, Scala).

   Below sample shows a python Spark code reading a file in HDFS and counting the number of lines with "ERROR" in it. lambda is like an inline function in Python.

```
text_file = spark.textFile("hdfs://...")
errors = text_file.filter(lambda line: "ERROR" in line)
# Count all the errors
errors.count()
```

   Storm applications can be written in Python using Streameparse. [19]

   But the biggest thing that will help knowing Python is that it will give you potential tools for ML tasks (so called data science) via its many libraries like numpy,pandas,scipy,sckikit-learn etc specifically built for performing data analysis.[20]. Ipython notebook give you a nice UI for doing such activities.[21] So in case you want to start with a small prototype- POC of data analytics knowing Python will help, before you move the ML model for Big data via Spark, mahout,H2O etc

   A sample ML task using sckitlearn, mlatplotlib- below code loads a data set related to digits which is available with scikitlearn. We create a SVM ML model instance (clf) then train that model using all the digits except the last one ([:-1]) then ultimately predict for the last digit using (.predict) [32]

```
from sklearn import datasets
from sklearn import svm

clf = svm.SVC(gamma=0.001, C=100.)
digits = datasets.load_digits()
clf.fit(digits.data[:-1], digits.target[:-1])
```

```
clf.predict(digits.data[-1:])
```

What in Python will help –Python basics will do with special emphasis on – text processing, Collections, OOPs concepts/design basics, libraries related to XML, JSON , web scraping (beautiful scoup), Ipython notebook
Will give an edge – Data analytics library: numpy, pandas, scipy, sckikit-learn. MVC framework : Django

4. *Scala - The most recent among the 3 and becoming one of the popular languages in Big data circle. Scala is based out of JVM and has both OOPs and functional style support. Best part is it supports all the libraries of Java so you get the best of both world. Since it takes features of both object oriented features (from Java) and functional principles (from Haskell etc) its like learn one to enjoy both. As a result it makes the language more challenging from a learning prospective but it definitely worth the challenge. Moreover to make things work in big data using Scala you don't quite need to understand all language features.*

   Hadoop => Cascading => Scalding =>Scala allows you to write programs in Hadoop in really concise and faster way. [22].Recently Scalding has support for REPL to see things in action after you type the command. The flow above mentions the various abstractions above hadoop. (Scalding a DSL for scala above Cascading which in turn is a framework above Hadoop)

   A sample scalding job code snippet that takes 2 files as input and creates fields for them like the first file (input 1) is made to have 4 fields('id, 'email, 'language, 'location) based on separator comma. Ultimately we join the two files based on fields 'user_id -> 'id and group by product id and get the locations related to each product. The final output is written to output file (output argument). Good thing is that scalding has support to run in local mode to test your solutions.

```
val input1 = TextLine( args( "input1" ) )
val input2 = TextLine( args( "input2" ) )
val output = Tsv( args( "output" ) )
val usersInput = input1.read.mapTo( 'line -> ('id, 'email, 'language,
                'location) ) { te: TupleEntry =>
val split = te.getString( "line" ).split("\t");
(split( 0 ), split( 1 ), split( 2 ), split( 3 ))
}
```

```
        val transactionsInput = input2.read.mapTo( 'line ->
      ('transaction_id,'product_id, 'user_id, 'purchase_amount,
              'item_description) ) { te: TupleEntry =>

    val split = te.getString( "line" ).split("\t");

  (split( 0 ), split( 1 ), split( 2 ), split( 3 ), split( 4 ))

      }


val joinedBranch =  transactionsInput

  .joinWithSmaller('user_id -> 'id, usersInput)

  .project('product_id, 'location)

  .unique('product_id, 'location)

  .groupBy('product_id) {group => group.size(Set('location))}

  .write(output)
```

Spark is built in Scala so it has full support for Scala. Has a nice REPL for Scala to see commands getting executed instantly. In case you want to add some extension to already existing libraries in Spark, Scala would be needed. [23 – Check how StatCounter class is extended]

The good thing about Scala its helps to build prototype quickly in Spark using the REPL to see instant command output. Same logic in Spark can be implemented in Scala much concisely compared to java as in java we have to write anonymous classes.

Scala also has nice language features which helps to build DSL (domain specific language) for other tasks. [24]

What in Scala will help –Scala basics will do with special emphasis on – text processing, Collections, OOPs/functional programming concepts/design pattern, traits, case classes, pattern matching etc.
Will give an edge – sbt, Akka, Play, Spray framework. Scalaz library

5. *SQL – Although not a language in the class of languages described above, but big data processing without SQL is quite unimaginable so it deserves a mention here.*
   Hive was the original SQl engine above Hadoop and has metadata related to it in Hive metastore. [25]

But now we do have many choice of SQl over hadoop and these include below. All these does not uses mapreduce as the execution engine so come with little boost in performance and good part it can access the Hive metasore so as a result can access all your hive table–

Impala – A product from Cloudera it claims to process data really fast and does not use mapreduce. [26, 27]

Drill -Apache Drill is an open source, low-latency query engine for Hadoop that delivers secure, interactive SQL analytics at petabyte scale. With the ability to discover schemas on-the-fly, Drill is has capabilities on data stored in multiple formats in files or NoSQL databases. [28]

Spark SQl – Run on Spark backend so really fast and supports federated data stores like Drill i.e. data store in Relation database, NoSql, HDFS etc. [29, 30]

Presto – It came out of Facebook. Good thing about it is that it can query different sources other than hadoop to give a consolidated view. [31]

Conclusion –

So above points highlights the importance of the 3 languages in Big data. Although with the advent of new tools like Informatica Big data Edition, Talend big data, IBM Big Data Insights, Microsoft Azure HDInsights, Splunk probably you might just not need these languages that much to make things work. But they do not have the cutting edge updates happening in Big data tech space instantly. They do make updates to their tools based on what's happening in open source world but will lag a little bit from the open source community. Moreover these tools comes with their own licensing cost and these have some drawback of not supporting all the big data technologies stack that is present in the open source market. But they do have a huge plus point that of developer productivity.

There were few other languages which is also used in big data circle (maybe not as above 3) which definitely needs a mention are - R , JavaScript, Go and Clojure. Go among this is the most recent language and came out of Google and has potential to be used in building distributed systems.

But to develop applications with the current trend in big data space- the 3 languages mentioned above are still the market leaders – Java, python Scala (in no specific order though).

Note- All these is based on following the Big data technology landscape and reading various articles related to same and also by observing few projects patterns in my current client. Different customers have different requirements and so might use different technologies/languages, but in case any of project uses the 3 leading hadoop distribution- Cloudera, Hortonworks, MapR then chances are most probably applications will be based out of these.

References –

1. **https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html**
2. **http://research.google.com/archive/mapreduce.html**
3. **http://hadooptutorials.co.in/tutorials/mapreduce/advanced-map-reduce-examples-2.html**
4. **https://github.com/awanninger/map-reduce/tree/master/Programs**
5. **http://www.cascading.org/**
6. https://dzone.com/articles/handling-big-data-hbase-part-4
7. https://pig.apache.org/docs/r0.9.1/udf.html
8. https://blogs.oracle.com/datawarehousing/entry/three_little_hive_udfs_part
9. http://blog.cloudera.com/blog/2014/04/making-apache-spark-easier-to-use-in-java-with-java-8/
10. https://github.com/apache/spark/blob/master/examples/src/main/java/org/apache/spark/examples/JavaPageRank.java
11. **https://wiki.apache.org/solr/Solrj**
12. https://github.com/sonalgoyal/crux
13. http://hortonworks.com/blog/storm-kafka-together-real-time-data-refinery/
14. http://zdatainc.com/2014/07/real-time-streaming-apache-storm-apache-kafka/
15. https://github.com/apache/mahout/tree/master/examples/src/main/java/org/apache/mahout/classifier/sgd/bankmarketing
16. https://mahout.apache.org/users/classification/bankmarketing-example.html
17. **http://blog.matthewrathbone.com/2013/11/17/python-map-reduce-on-hadoop---a-beginners-tutorial.html**
18. **https://labs.spotify.com/2013/05/07/snakebite/**
19. **https://github.com/Parsely/streamparse**
20. **https://github.com/jakevdp/sklearn_pycon2013**
21. **http://nbviewer.ipython.org/github/ipython-books/cookbook-code/blob/master/notebook**
22. **https://github.com/twitter/scalding**
23. **https://github.com/sryza/aas/blob/master/ch02-intro/src/main/scala/com/cloudera/datascience/intro/RunIntro.scala**
24. http://blog.cloudera.com/blog/2015/08/how-apache-spark-scala-and-functional-programming-made-hard-problems-easy-at-barclays/
25. **http://www.semantikoz.com/blog/the-free-apache-hive-book/**
26. **http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala.html**
27. **http://www.cloudera.com/content/www/en-us/resources/aboutcloudera/cloudera-impala-ebook.html**
28. **https://www.mapr.com/products/apache-drill**
29. **http://spark.apache.org/docs/latest/sql-programming-guide.html**
30. **https://databricks-training.s3.amazonaws.com/data-exploration-using-spark-sql.html**
31. **https://prestodb.io/**

32. http://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html#example-classification-plot-digits-classification-py
33. https://flink.apache.org/