How YARN Opens Doors to Easier Programming Tools for Hadoop 2.0 Users

by **John Lilley** | February 19, 2014

The emergence of YARN for the Hadoop 2.0 platform has opened the door to new tools and applications that promise to allow more companies to reap the benefits of big data in ways never before possible with outcomes possibly never imagined. By separating the problem of cluster resource management from the data processing function, YARN offers a world beyond MapReduce: less-encumbered by complex programming protocols, faster, and at a lower cost.

Yet while many Hadoop applications have migrated and other migrations are in process, most of these applications still cling to the original Hadoop paradigm: MapReduce. That's like putting lipstick on a pig (no pun intended). These programs basically dress up the same functionality without taking advantage of the new capabilities of YARN. Why is YARN important? Some background may help.

Hadoop was first developed in 2005 by Doug Cutting and Mike Carafella with the help and blessing of Yahoo, which to this day runs the largest Hadoop cluster in the world. Hadoop was open-sourced under the auspices of Apache, and major contributors include Hortonworks, Yahoo, Cloudera, and many others. Throughout Hadoop's development, until October 2013 with the release of Hadoop 2.0, MapReduce was *the* computational framework. If you wanted to crunch data under Hadoop, you wrote or generated MapReduce code. Hadoop 2.0 changed that.

Under Hadoop 2.0, MapReduce is but one instance of a YARN application, where YARN has taken center stage as the "operating system" of Hadoop. Because YARN allows *any* application to run on equal footing with MapReduce, it opened the floodgates for a new generation of software applications with these kinds of features:

More programming models.
Because YARN supports any application that can divide itself into parallel tasks, they are no longer shoehorned into the palette of "mappers," "combiners," and "reducers." This in turn supports complex data-flow applications like ETL and ELT, and iterative programs like massively-parallel machine learning and modeling.

Integration of native libraries.
Because YARN has robust support for *any* executable – not limited to MapReduce, and not even limited to Java – application vendors with a large mature code base have a clear path to Hadoop integration.

Support for large reference data.
YARN automatically "localizes" and caches large reference datasets, making them available to all nodes for "data local" processing. This supports legacy functions like address standardization, which require large reference data sets that cannot be accessed from the Hadoop Distributed File System (HDFS) by the legacy libraries.

Despite these innovations, most Hadoop software developers are stuck in the Hadoop 1.0 mindset. They've sacrificed a "bigger leap" to broader availability and greater usability of Hadoop 2.0's powerful resources in exchange for early market entry. The effect for users: Hadoop still has a tall fence around it. Most Hadoop applications still suffer from one or more of these deficiencies:

- They feel like programming tools, exposing too much Java or scripting.

- Their "in Hadoop" software is a small feature subset of their "legacy" software.
- They don't run in Hadoop at all, instead pushing queries through Pig or Hive, and are limited by the volume of data that can be pulled from Hadoop to the "outside."
- They generate MapReduce, which while not a problem in theory, tends to make applications feel like "MapReduce veneers."

Fortunately, ISVs are starting to realize that the power of Hadoop 2.0 lies in enabling applications to run *inside* Hadoop, without the constraints of MapReduce. Vendors like my company, RedPoint Global, as well as Revolution Analytics, Actian, and Talend are starting to create applications that, to greater or lesser extent, feel like more than glossy MapReduce programming veneers.

One of the most exciting developments is a new crop of "visual data-flow design" applications. These applications have been around for years, even decades, in the classic world of ETL, ELT, data quality, and analytic databases. These mature products are used continuously by thousands of *non-programmers* to solve data problems including marketing analytics, fraud detection, clickstream monitoring, replication, and master data management. The accessibility of these solutions to analysts and "data scientists" is critical.

MapReduce Expertise Hard to Come By.
MapReduce is a software framework that developers have been using for years to generate programs for Hadoop. While the popularity of Hadoop has grown—advanced even more thanks to the hype around big data—the number of MapReduce programmers hasn't climbed as fast. The bulk of them can be found in Internet companies and flashy start-ups, and if you're a large company you might have a shot at hiring a few of them. But big demand and low inventory means companies are paying a premium for MapReduce skills.

The reason is that, like many parallel programming models, MapReduce introduces fairly specialized concepts that might be alien—even to seasoned programmers. And thinking in MapReduce isn't necessarily easy. While certain problems can be expressed quite simply, translating a real business problem into MapReduce patterns and idioms, requires experience, training, and insight.

And, while MapReduce can be written in many different languages—including Python, R, Lisp, C#, C++, and Ruby—most MapReduce programs are implemented in Java since only Java supports the full MapReduce feature set. Some companies have been able to retrain their Java developers and infrastructure engineers on MapReduce and Hadoop. But in doing so, they just increased their employee's market value, and potentially just trained their competitor's new hires.

Of course, MapReduce isn't the only option for processing data at scale using Hadoop. Tools like Pig (a large scale query and analysis system), Hive (a data warehousing application) and others have been available for some time. These tools can express transforms and analysis using more accessible constructs: Hive uses HQL, a language similar to SQL. Pig provides a script language (Pig Latin) to create MapReduce jobs. Business analysts familiar with conventional tools like SQL and SAS should be able to use these tools to write programs to solve large data problems on Hadoop clusters. But, in both cases, there's an additional commitment of time: first, to learn these languages and second, to manage what has morphed from a data analysis problem into a software development task.

The Value of Visual Application Development Tools.
A new generation of "visual design" application development tools could help solve these coding problems. By running as native YARN applications and side-stepping the need for MapReduce, some of these programs eliminate coding altogether. Other tools reduce coding by generating MapReduce code or by generating scripts like Pig. Visual designers are powerful for several reasons:

- Increased level of abstraction: Instead of thinking about classes and methods, users see operations, data, and outcomes.
- Fast "what-if": The drag-and-connect interface supports quick try/observe/adjust cycles.
- Automatic optimization: Scaling and efficiency are built-in.
- High-level palette: High-level constructs like "standardize address", "deduplicate consumers", or "parse names" are often directly on the designer palette.

But so much for theory. How does this look in practice? Here's an illustration that shows how three competing approaches differ:

- MapReduce written in Java
- Pig scripts developed from scratch
- A visually-designed process running a native YARN ETL application. The application is from RedPoint Global, but comparable approaches can be seen in Talend and Actian.

Using these three approaches, we conducted a "Word Count" test on 30,000 files (20 gigabytes) of Project Gutenberg books. This test reads lines of text, breaks them into words, and creates a concordance (list of words and the number of times each occurs). Our Hadoop cluster was small—only four nodes—but was large enough to demonstrate the concepts and tradeoffs. To make the test more realistic, we also required that common punctuation characters be stripped from the text and that the results were sorted descending by count.

Here's what we found:

MapReduce:
Set-up time: While flexible, MapReduce had the longest learning curve and required significant coding skills—both as a Java programmer and a MapReduce specialist—to prepare the test.

Sample MapReduce (small subset of the entire code which totals nearly 150 lines):

```
public static class MapClass
extends Mapper<WordOffset, Text, Text, IntWritable> {
  private final static String delimiters =
      "',./<>?;:\"[]{}-=_+()&*%^#$!@`~ \\|«»¡¢£¤¥¦©¬®¯±¶·¿";
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
  public void map(WordOffset key, Text value, Context context)
    throws IOException, InterruptedException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line, delimiters);
    while (itr.hasMoreTokens()) {
       word.set(itr.nextToken());
       context.write(word, one);
    }
  }
}
```

Sample Pig script without the User Defined Function:

```
SET pig.maxCombinedSplitSize 67108864
SET pig.splitCombination true
A = LOAD '/testdata/pg/*/*/*';
B = FOREACH A GENERATE FLATTEN(TOKENIZE((chararray)$0)) AS word;
C = FOREACH B GENERATE UPPER(word) AS word;
D = GROUP C BY word;
E = FOREACH D GENERATE COUNT(C) AS occurrences, group;
F = ORDER E BY occurrences DESC;
STORE F INTO '/user/cleonardi/pg/pig-count';
```

Performance: It took 3 hours 20 minutes to run the test initially due to the "small files problem" that is familiar to seasoned MapReduce programmers. This problem occurs when reading large collections of small files, because MapReduce's default behavior is to assign a mapper task to each file. This results in a huge number of tasks. To address this issue, we created a custom InputFormat class to read multiple files at once. This reduced our run time to 58 minutes. Then we tuned the split sizes and mapper task limit appropriately, which dropped the run time to about six minutes. Further tweaks, such as adding a combiner to "pre-aggregate" the counts in the mappers and optimizing the counting process using in-memory tables, lowered the time to below three minutes.

Comments: Each performance improvement came at a cost. Overall, nearly a full day of programmer time was spent optimizing the original code.

Pig:
Set-up time: Learning Pig was fairly easy. It was pretty natural to create the coding for this test. However to make a common adjustment in the code—changing the set of whitespace separators to include punctuation—required the addition of a "User defined function" or UDF which had to be written in Java. Pig is generally easy enough to use by people who aren't professional programmers but who know how to write scripting languages like JavaScript or Visual Basic.

Performance: The results were not stellar: run time was close to 15 minutes.

Comments: While coding took less time, Java programming was ultimately required to meet the test requirements.

YARN-enabled ETL/ELT designer:
Set-up time: The tool is designed to have a shorter learning curve than even Pig scripting. Dragging tools like "Delimited Input", "Summarize" and "Tokenize" from the palette and configuring them is designed to be discoverable and intuitive, and the resulting diagram has a one-to-one correspondence between icons and operations. There's no need for coding or learning a language like Java or Pig. The visual design covers the input file format, tokenizing and counting steps. The resulting data flow graph contains seven icons along with a grouping construct that shows what executes "inside" Hadoop. Each icon represents a step in the data transformation.

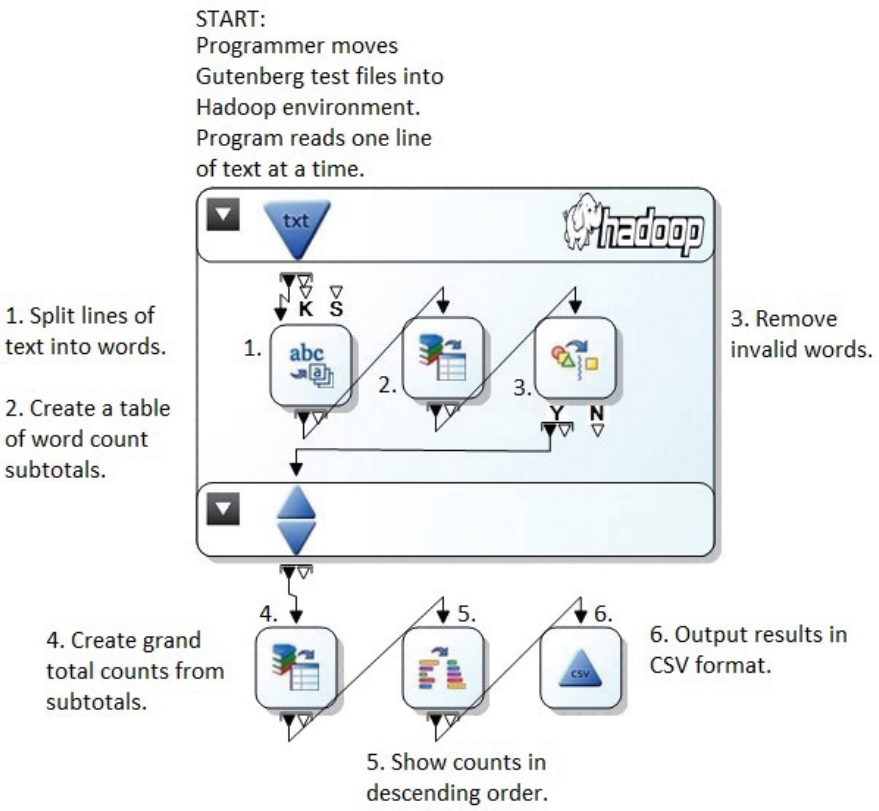
Performance: The run time for this data flow is just over three minutes with no tuning.

Comments: Because there is no code to manage, and editing is done visually, running "what if" scenarios is quick for non-programmers. Once the data flow is

designed, it can be stored and saved for later use. In addition, the logic can be captured into a "macro" for sharing and reuse between multiple data flows.

While this Project Gutenberg exercise may not be a "real world" benchmark—counting words isn't likely to be the problem you want to solve—it was instructive in terms of the comparative productivity of MapReduce versus a tool that is optimized for YARN. Companies that really want to reap the benefits of Hadoop 2.0 need to bypass code-intensive approaches and look at a new breed of development tools that solve problems using a model suitable for data analysts who know how to do typical SQL queries and who don't have Java or MapReduce expertise. With Hadoop 2.0, tools that leverage YARN's less-restrictive execution model have a chance to flourish and bring clear productivity gains to a broader base of businesses than ever before.

YARN-enabled ETL/ELT designer:



John Lilley is chief architect for RedPoint Global, Inc. He has been developing high-performance software for 20 years with an emphasis on data integration, data quality and marketing analytics. He founded DataLever in 1999 and became leader of the RedPoint Global Data Management engineering team in 2011. He can be reached at: www.linkedin.com/in/johnelilley.