

Hadoop Scripting Languages

Domain Specific Languages Pig and Jaql

Johan Uhle, Konstantin Haase
August 31, 2009

Seminar “Map/Reduce Algorithms on Hadoop”
Research Group Information Systems, Alexander Albrecht, Prof. Felix Naumann,
Hasso Plattner Institute, Universität Potsdam, D-14482 Potsdam, Germany,
{johan.uhle, konstantin.haase}@student.hpi.uni-potsdam.de

Abstract. In this paper the domain specific languages Pig and Jaql for Map/Reduce on Hadoop are evaluated regarding their features, ease of programming and performance.

1 Introduction

Working with big data sets has gained growing significance in the past years. This means processing heterogeneous gigabytes or terabytes of plain text by filtering, grouping or applying functions on the whole set or elements. One solution to this problem is using the Map/Reduce paradigm [1], which enables a cluster of computers to process a problem in parallel and thereby increase the processing speed.

Apache Hadoop is an Open Source Implementation of Map/Reduce [2]. It is published under the Apache License and maintained by the Apache Foundation. Its development is mostly driven by Yahoo! employees.

Hadoop programs are written in Java for the Hadoop API. On Hadoop, several domain specific languages try to establish a more abstract approach for utilizing the Map/Reduce paradigm. We have chosen to compare Pig and Jaql regarding their feature set, ease of programming and processing speed.

All source code written for the benchmarks in this paper are published online under Apache License.¹

¹ available at <http://github.com/rkh/hadoop-scripting/>

2 Jaql

Jaql [3] is a high-level scripting language for the JavaScript Object Notation (JSON). It is able to run on Hadoop and break most requests down to Map/Reduce tasks. Jaql heavily borrows from SQL, XQuery, LISP, Pig Latin, JavaScript and Unix Pipes. [4]

Developed mainly inside IBM and with a quiet mailinglist Jaql currently faces a serious lack of documentation and community. The documentation found online is outdated and incomplete in most cases. In addition, Jaql was undergoing major changes in the time of writing this paper.

Even though available, there is no need to write in a strict Map/Reduce pattern. At the point of execution (i.e. the end of a query statement) Jaql transforms the parsed statement into another, equivalent but optimised statement. This step is comparable to a query optimiser in modern database management systems. The optimised query can in turn be transformed back to Jaql code, which is useful for debugging.

Jaql can be extended with user-defined functions, written either in Java or in Jaql itself. However, it is not possible to use Jaql as a general-purpose programming language, as it is not Turing-complete: It lacks both recursion and a universal loop function.

The current Jaql implementation features three modes to run in: In stand-alone mode Hadoop is not used at all and the jobs are not split in Map/Reduce tasks. When using Jaql with Hadoop a so-called mini-cluster can be used, which is managed by Jaql and runs all tasks on one computer in the same process (with one thread per Map/Reduce task). The last option is running Jaql on a traditional, external Hadoop cluster.

3 Programming with Jaql

Listing 1.1. A sample Jaql query

```

1 $source = "http://server/api.php?format=json&query=foo";
2 read(http($source))
3   → transform each $["name"]
4   → write(hdfs("output.dat"));

```

One of the main features of Jaql is the ability to read from and write to different storage types, currently the local file system, the Hadoop Distributed File System, HBase and HTTP. Since Jaql is based on JSON, reading from HTTP allows easy integration of web services like Wikipedia or Flickr, that offer a JSON API.

A typical Jaql request can be seen in listing 1.1. Jaql ships with an interactive shell (read-eval-print loop).

Jaql features two ways of passing parameters to a function. You can write the parameters in brackets after the function name, as known from most other programming languages, i.e. `foo("bar")`. However, Jaql also features a new pipe syntax, more consistent with the way built in Jaql statements are written. The value you “pipe into” a function will be past as the first argument, therefore `foo("bar", "baz")` is equivalent to `"bar" → foo("baz")`.

All values, either arguments or return values, are JSON objects or Jaql functions. Every function in Jaql returns a value. Jaql offers advanced functionality for querying, transforming and aggregating those values. An example can be seen in listing 1.2. Note that operations like `expand` or `group by` will loop through all element of the given array and potentially split this operation on multiple Map/Reduce tasks. Within a loop `$` is referring to the current element.

Listing 1.2. Wordcount in Jaql

```

1 read($src)
2   → expand splitArr($, " ")
3   → group by $w = ($) into [$w, count($)];

```

A statement is terminated by a semicolon and will at that point be optimized, as mentioned earlier. However, instead of executing the statement, Jaql is able to return the optimized Jaql code by calling `explain` on the statement.

4 Pig

Pig [5] is a high-level scripting language for data transformation. It is a Hadoop Subproject and in the Apache Incubator since 2007. Just as Hadoop it is mainly developed by Yahoo!, where in early 2009 30 % of all Map/Reduce jobs have been implemented using Pig. [6] It has an active community and a growing ecosystem.

The development of Pig took three key aspects into account: [5]

The *ease of programming* enables the user to create powerful scripts that are easy to write, read and maintain even when working with very large data sets.

Another goal was *optimization*. When writing a Pig script, it is not necessary for the programmer to think within the Map/Reduce paradigm since Pig handles the transformation of the script to the particular Map/Reduce jobs. This also offers opportunities for automatic optimizations made by the Pig compiler. A good explanation about how Pig parses and compiles a Pig script can be found in the paper “Pig Latin: A Not-So-Foreign Language for Data Processing” Chapter 4. [7]

Pig features *extensibility* achieved by user-defined functions (UDF [8]) which are programmed in Java using the Pig interfaces and are called within Pig Latin. Thus Pig has the same feature set as Java Hadoop.

² http://hadoop.apache.org/pig/mailling_lists.html

Pig uses the scripting language Pig Latin [9]. The syntax looks similar to SQL (which may be integrated into Pig additionally [10]), albeit Pig Latin is a data transformation language and therefore is similar to the database query optimiser in modern database management systems.

Pig Latin scripts often do not consist of more than 10 lines of code, whereas user-defined functions in Java tend to be more complex and may need more than 100 lines of code.

5 Programming With Pig

Pig offers an interactive shell called Grunt and a command line tool for running external scripts. Both ways work in local mode (Pig standalone) and on a Hadoop cluster.

Listing 1.3. A typical Pig line of code

```
1 ordered = ORDER words;
```

The code in listing 1.3 shows the typical structure of a pig statement. It is the data transformation of one set (“words”) with an operation (“ORDER”) into a new set (“ordered”). The input data sets are always on the right side of the assignment. The resulting output data set is always on the left side.

Most often Pig Latin Scripts naturally follow the structure of listing 1.4.

Listing 1.4. Pig Latin Script Structure

```
1 Load Data → Manipulate Data → Group Data → Output Data
```

A Pig Latin script always begins with a LOAD statement. The loaded data is manipulated via FILTER, FOREACH GENERATE or DISTINCT statements or with the help of user-defined functions (UDF). Afterwards the data is often grouped. Then the cycle either starts all over again e.g. with additional data being joined or otherwise the resulting data set is written to the output with a STORE statement. Of course this structure may be altered depending on the problem.

Every line is terminated by a semicolon. The pig compiler starts building a logical and physical execution plan once a STORE or debug command is evaluated.

Every data set has a schema determining the structure of its data. These schemas may be defined explicitly by the user in Pig Latin or in an UDF or determined implicitly by Pig. A certain attribute in a data set is accessed either via the corresponding name in the schema or via its position. As already stated Pig has the same possibilities as native Java Hadoop due to the use of UDFs. These functions may implement custom load or store functionality or manipulate data per element or over a whole data set. It is also possible to use UDFs for filtering by deciding if certain data shall be in the resulting data set. The development of UDFs is well documented [8] but requires noteworthy more effort than just writing Pig Latin. Especially defining own schemas can be a hard-to-debug task. The development of UDFs offers common means of Java debugging.

Pig Latin scripts are best to be debugged with the `ILLUSTRATE`, `DESCRIBE` and `DUMP` statements, which give access to the schema and the data. Unfortunately schemas are only to be viewed one level deep and therefore deeper nested data structures are to be debugged more difficult.

6 Cluster Setup

All benchmarks were measured on the same cluster consisting of ten nodes. Unfortunately our nodes did vary in hardware widely, especially in CPU and memory. Two of the nodes were two upto four times faster than the others. This caused some variation in performance tests when running with a small number of map or reduce tasks, since the workload of the faster nodes varied depending on how many tasks they have got assigned. In our opinion a homogenous cluster would be more suited for benchmarking. We used Hadoop version 0.18.3 since neither Pig nor Jaql are compatible with the current version of Hadoop (0.20). We ran the latest version of Pig 0.3.0 and a development snapshot of the upcoming Jaql 0.4³. During the Markov Chain benchmarks we also tried Jaql 0.3 and both release candidates for Jaql 0.4.

7 Scenario: Wordcount

In the following wordcount scenario the words in a document are counted by number of occurrence. This scenario fits the purpose of comparing the different approaches very well, because counting words is a classic Map/Reduce task. Furthermore programming a wordcount script is fairly easy and reduces the risk of us using a bad implementation for the benchmark.

³ available at <http://github.com/rkh/jaql>

8 Benchmark: Wordcount

We have been running our Hadoop Java, Pig and Jaql scripts against a series of Wikipedia articles with sizes between 10 Mb and 5000 Mb and 4 runs for each size.

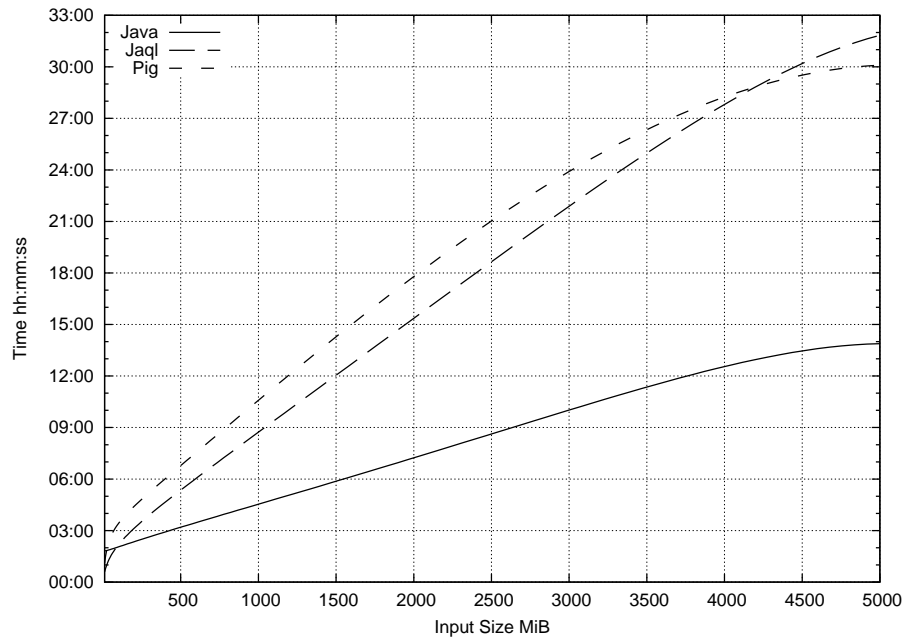


Fig. 1. Wordcount performance benchmark results

It can be observed that Java performs better by an order of magnitude, compared to Pig and Jaql, which do not differ much from each other. Apparently all do scale well, since execution time is growing slower than input size for all three implementations.

9 Scenario: Markov Chain

A Markov Chain is a logical construct for generating random texts, which appear to have a more or less sane content.⁴ Texts are generated using a function that returns a random word following the given phrase in a predefined input text. One way to implement fast access to the input text is by creating an index of

⁴ Indeed, there is a lot more one can do with Markov Chains. [11]

all phrases and their successors in the input text. In our Map/Reduce example we use articles from Wikipedia, which we split into tuples and return those as a SQL statement to be stored inside a database.

10 Programming Markov Chain with Jaql

Our final Jaql script for generating a Markov Chain index can be seen in listing 1.6. In line 42 the script will call the self function \$markov, defined in line 19–25, on an array of strings. Each string represents a line from the input file. All lines will be replaced with the results of \$markovLine in line 21, which are tuples with four words each. Then identical tuples will be aggregated into an SQL expression, containing the tuple itself and number of appearances.

During development we have created multiple version of this script. Unfortunately, we were not able to run any version on our Hadoop cluster with more than one map task⁵, therefore we did only run benchmarks for Pig and Java.

11 Programming Markov Chain with Pig

Listing 1.5. Markov Chain in Pig Latin

```

1 wikipedia = LOAD 'wikipedia' USING PigStorage('\t') AS (row:chararray);
2 tuples = FOREACH wikipedia GENERATE FLATTEN(splitsuc.SPLITSUC(*));
3 grouped = GROUP tuples by (word1, word2, word3, word4) PARALLEL 16;
4 grouped_counted = FOREACH grouped GENERATE group, COUNT(tuples);
5 STORE grouped_counted INTO 'wikipedia.sql' USING splitsuc.STORESSQL();

```

For the implementation two UDFs had to be programmed.

- The first one is `SPLITSUC()`, which is called within a `FOREACH` loop, processing each row of the input. It is splitting the rows into 4-word tuples with a tuple containing every word with its three predecessors.
- The other one is `STORESQL()`, that is transforming the 4-tuples into SQL Multi-INSERT statements ready to be imported into a database. The number of output files is determined by the number of reduce jobs.

These UDFs each are about 80 lines of code and consumed most of the development time needed for implementing the algorithm. Writing the Pig Latin code was a fairly easy task, since the necessary steps were straight-forward:

1. Split into 4-tuples
2. Group by 4-tuples
3. Count the occurrences of each 4-tuple

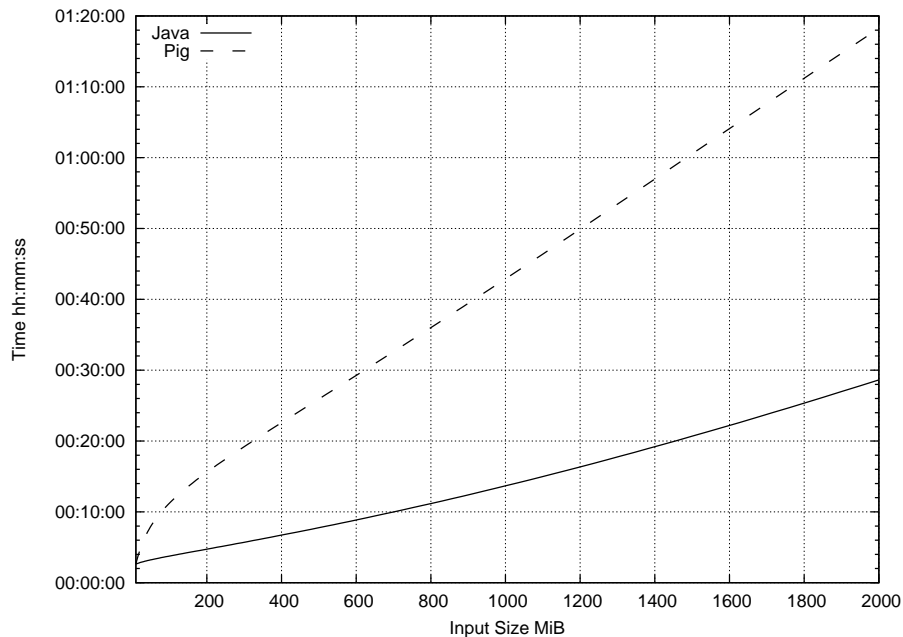


Fig. 2. Markov Chain benchmark results

12 Benchmark: Markov Chain

We have been running a Hadoop Java and a Pig implementation on the cluster against a series of Wikipedia articles with a size of 10 Mb to 2000 Mb. To minimise benchmark errors we have done 4 runs for each size.

It is clearly visible, that Pig is slower than the native Java implementation. We were not able to run our Jaql implementation due to the problems mentioned before.

13 Conclusion

Pig and Jaql both succeed in the task of abstracting from the Map/Reduce pattern towards the actual data processing tasks.

Pig Latin is limited in expressiveness and a fallback to UDFs is often needed, whereas Jaql provides a richer feature set being extendable with pure Jaql.

The time consuming part of developing in Pig is writing UDFs. If a library of commonly used UDFs is present (i.e. Pig's Open Source UDF Repository Piggy-Bank [12] or company specific in-house libraries) a Pig Latin script is composed in

⁵ read: with no more than 64 MB of input data

very short time. For a lot of common tasks, native Pig Latin statements already are powerful enough and UDFs aren't needed.

Furthermore we would prefer Pig for "easy" querying tasks, since basic functionality is quickly accessible and well documented. But we think, that prototyping of more complex problems is faster done in Jaql than in Pig.

Since Pig and Jaql are still in development improvements may be seen, especially in performance, where both aim on competing with native Java.

While Pig seems to be production-ready Jaql currently appears to be more of a research project.

In conclusion, Pig and Jaql both ease programming for Hadoop by allowing rapid development, compared to Hadoop's Java interface, but they are both in an early stage of development and cannot yet fully compete with pure Java Hadoop performance.

References

1. Sanjay Ghemawat, Jeffrey Dean: MapReduce: Simplified Data Processing on Large Clusters. <http://labs.google.com/papers/mapreduce.html> (2004)
2. Apache Software Foundation: Apache Hadoop. <http://hadoop.apache.org/> (2009)
3. IBM Corp.: Jaql. <http://jaql.org> (2009)
4. K.Beyer, V. Ercegovac: Jaql: A JSON Query Language. <http://code.google.com/p/jaql/wiki/JaqlOverview> (2009)
5. Apache Software Foundation: Pig. <http://hadoop.apache.org/pig> (2009)
6. The Register: Yahoo breeds Pig that talks elephant. http://www.theregister.co.uk/2009/05/06/yahoo_and_pig/ (2009)
7. Olston, C., B. Reed, U. Srivastava, R.K., Tomkins, A.: Pig latin: A not-so-foreign language for data processing. In: SIGMOD '08: Proceedings of the ACM SIGMOD International Conference on Management of Data, Vancouver, Canada (jun 2008)
8. Apache.org Wiki: User-Defined Function Manual. <http://wiki.apache.org/pig/UDFManual> (2009)
9. Apache Software Foundation: Pig Latin Manual. <http://hadoop.apache.org/pig/docs/r0.2.0/piglatin.html> (2009)
10. Apache Software Foundation: SQL interface for Pig. <https://issues.apache.org/jira/browse/PIG-824> (2009)
11. Wikipedia: Markov Chain. http://en.wikipedia.org/wiki/Markov_Chain (2009)
12. Apache Software Foundation: Piggy Bank - User Defined Pig Functions. <http://wiki.apache.org/pig/PiggyBank> (2009)
13. White, T., Romano, R.: Hadoop: The Definitive Guide. O'Reilly Media (2009)

APPENDIX

Listing 1.6. Markov Chain in Jaql

```

1 registerFunction("strSplit",
2   "com.acme.extensions.expr.SplitIterExpr");
3 registerFunction("splitArr",
4   "com.acme.extensions.expr.SplitIterExpr");
5
6 $markovEntry = fn($words, $pos) (
7   $words
8     → slice($pos - 3, $pos)
9     → transform serialize($)
10    → strJoin(",_")
11 );
12
13 $markovLine = fn($line) (
14   $words = $line → strSplit("_"),
15   range(3, count($words) - 1)
16     → transform $markovEntry($words, $)
17 );
18
19 $markov = fn($lines) (
20   $lines
21     → expand each $line $markovLine($line)
22     → group by $w = ($) into
23       "INSERT_Into_jaqltest.splitsuc_VALUES_(" +
24       $w + ",_" + serialize(count($)) + ");";
25 );
26
27 // mini cluster
28 // hdfsShell("-copyFromLocal input corpera/current.dat") * 0;w
29
30 // real cluster
31 $source = "corpera/current.dat";
32
33 read({
34   type: "hdfs",
35   location: $source,
36   inoptions: {
37     format: "org.apache.hadoop.mapred.TextInputFormat",
38     converter: "com.acme.extensions.data.FromLineConverter"
39   }
40 })
41   → $markov()
42   → write({type: "hdfs", location: "jaql_output"});

```
