Benchmark

Jet 0.4 vs Spark and Flink Batch Benchmark

Comparison

Hazelcast Jet 0.4
Apache Flink 1.2.0
Spark 2.1.1

Benchmarks

Word Count – Total size of input file is given in parentheses.

- 1 million distinct words (64GB)
- 1 million distinct words (640GB)
- 10 million distinct words (73.5GB)
- 100 million distinct words (82.8GB)

All data sets are distributed across all 10 nodes evenly. Each file contains several lines, with each line containing 20 words. Words are all numeric, starting from 0 to the maximum distinct number. So a typical file would look like:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
...
........................... 999998 999999 1000000
```

All source code is available here: https://github.com/hazelcast/big-data-benchmark

Test Environment

10 Servers running on AWS:

AWS Instance Type	c3.8xlarge
Physical Processor	Intel Xeon E5-2680 v2 2.8
vCPU	32
RAM	60 GiB
Storage	2 x 320 SSD
Network	10 Gigabit
Java	Oracle 1.8.0_121

JVM Heap per node	32 GB
Hadoop	Hadoop 2.7.3

Hazelcast Jet 0.4

32 GB JVM Heap per Node, 1600 partitions:

Data Set	Duration (secs)	Throughput (MB/s)
1m distinct 64GB	32.97	1,987.75
1m distinct 640GB	306.9	2,135.42
10m distinct 73.5GB	40.31	1,867.13
100m distinct 82.8GB	115.49	728.83

Hazelcast Jet 0.4 with IMap

The word count benchmark using IMap is the same as the previous benchmark, except that the data is read from and written to a distributed IMap instead of HDFS. Many Jet users can benefit from the distributed data structures embedded in Jet.
The only difference between the DAGs of the two benchmarks are the source and sink vertices.
Jet is one of the most efficient ways to do bulk import of data into a IMap from different sources: the IMap is populated by reading the lines from the very same file in HDFS and putting them into map by another Jet job. The time this job takes is recorded in the ***Load to IMap*** column.

On Heap

32 GB JVM Heap per Node, 1600 partitions:

Data Set	Load to IMap (secs)	Duration (secs)	Throughput (MB/s)	Notes
1m distinct 64GB	53.30	8.48	7,728.3	
1m distinct 640GB	–	–	–	Dataset too big to fit in cluster memory.
10m distinct 73.5GB	56.50	18.54	4,059.55	
100m distinct 82.8GB	63.79	149.27	563.90	

Off Heap

32 GB JVM Heap 20 GB Offheap per Node, 1600 partitions:

Data Set	Load to IMap (secs)	Duration (secs)	Throughput (MB/s)	Notes
1m distinct 64GB	52.34	8.31	7,886.40	

1m distinct 640GB	–	–	–	Dataset too big to fit in cluster memory.
10m distinct 73.5GB	50.20	12.91	5,829.90	
100m distinct 82.8GB	57.49	75.97	1,107.97	

Apache Flink 1.2.0

On Heap

Configuration:

```
taskmanager.heap.mb: 32768
taskmanager.numberOfTaskSlots: 32
taskmanager.network.numberOfBuffers: 40960
taskmanager.memory.preallocate: true
env.java.opts.taskmanager: -XX:+PrintGC -XX:+PrintGCTimeStamps -Xloggc:{{flink_home}}/log/flink.gc.log
```

Data Set	Duration (secs)	Throughput (MB/s)
1m distinct 64GB	136.20	481.17
1m distinct 640GB	842.63	777.76
10m distinct 73.5GB	246.73	305.05
100m distinct 82.8GB	260.67	322.91

Off Heap

32 GB JVM Heap/Offheap per Node
Configuration:

```
taskmanager.heap.mb: 32768
taskmanager.memory.off-heap: true
taskmanager.numberOfTaskSlots: 32
taskmanager.network.numberOfBuffers: 40960
taskmanager.memory.preallocate: true
env.java.opts.taskmanager: -XX:+PrintGC -XX:+PrintGCTimeStamps -Xloggc:{{flink_home}}/log/flink.gc.log
```

Data Set	Duration (secs)	Throughput (MB/s)
1m distinct 64GB	106.58	614.90
1m distinct 640GB	837.57	782.45
10m distinct 73.5GB	188.53	399.21

100m distinct 82.8GB	212.80	395.55

Apache Spark 2.1.1

On Heap

Configuration:

```
spark.driver.memory 16g
spark.executor.memory 32g
spark.executor.cores 32
spark.executor.extraJavaOptions -XX:+PrintGC -XX:+PrintGCTimeStamps -Xloggc:{{spark_home}}/logs/spark.gc.log
```

Data Set	Duration (secs)	Throughput (MB/s)	Notes
1m distinct 64GB	77.17	849.24	
1m distinct 640GB	730.94	896.60	
10m distinct 73.5GB	937.80	80.26	Heavy GC pauses
100m distinct 82.8GB	–	–	Did not run test as 10m took more than 15 minutes.

Off Heap

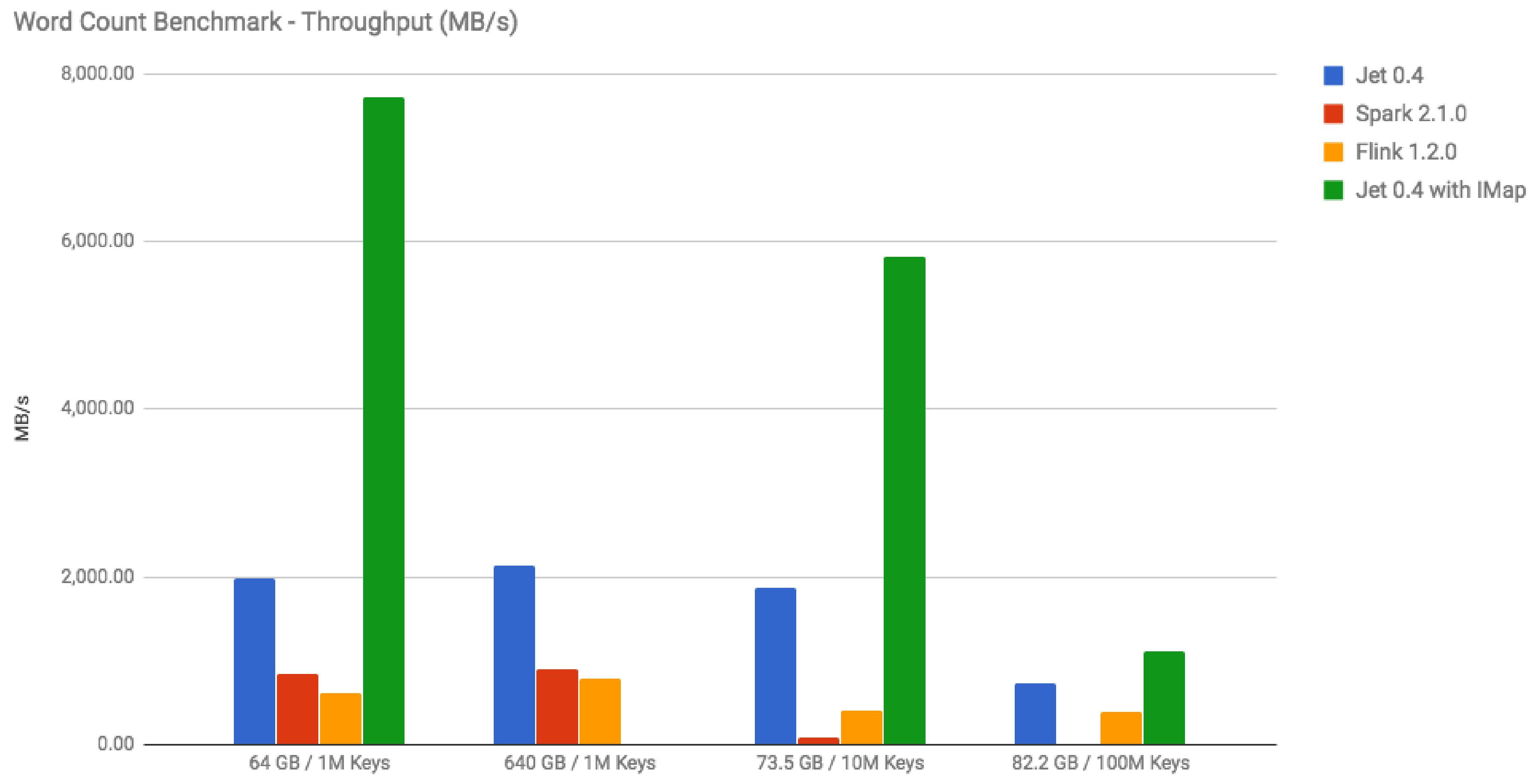
20GB Off Heap
Configuration:

```
spark.driver.memory 16g
spark.executor.memory 32g
spark.executor.cores 32
spark.memory.offHeap.enabled true
spark.memory.offHeap.size 20g
spark.executor.extraJavaOptions -XX:+PrintGC -XX:+PrintGCTimeStamps -Xloggc:{{spark_home}}/logs/spark.gc.log
```

Data Set	Duration (secs)	Throughput (MB/s)	Notes
1m distinct 64GB	86.56	757.12	
1m distinct 640GB	–	–	Didn't run test since no difference to without off heap is anticipated.
10m distinct 73.5GB	1558.42	48.30	
100m distinct 82.8GB	–	–	Did not run test as 10m took more than 26 minutes.

Results Summary

Word Count is the classic Big Data sample app and is often used to compare performance between systems. Jet 0.4 is faster than all other frameworks. Moreover, when the data set is available in embedded distributed Map Jet is almost 4x faster compared to starting from HDFS.


Word Count Benchmark - Throughput (MB/s)
MB/s
64 GB / 1M Keys
640 GB / 1M Keys
73.5 GB / 10M Keys
82.2 GB / 100M Keys
Jet 0.4
Spark 2.1.0
Flink 1.2.0
Jet 0.4 with IMap