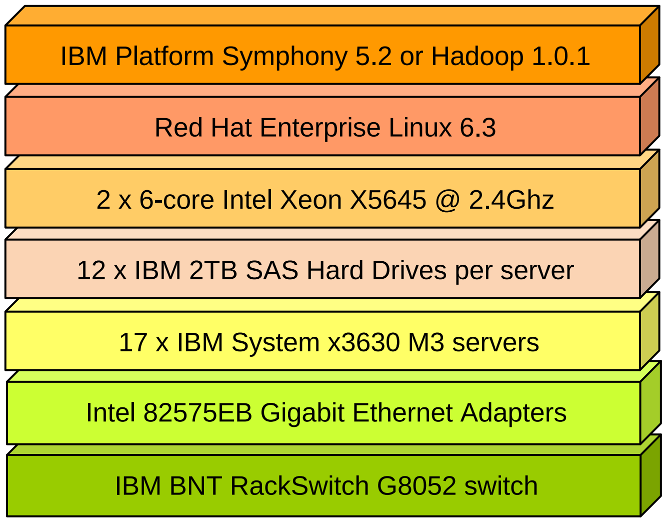
Comparison of IBM Platform Symphony and Apache Hadoop Using Berkeley SWIM

Issue 1.0, 6 November 2012

Technology Stacks Under Test


IBM Platform Symphony 5.2 or Hadoop 1.0.1
Red Hat Enterprise Linux 6.3
2 x 6-core Intel Xeon X5645 @ 2.4Ghz
12 x IBM 2TB SAS Hard Drives per server
17 x IBM System x3630 M3 servers
Intel 82575EB Gigabit Ethernet Adapters
IBM BNT RackSwitch G8052 switch

Key Results

- ➔ In jobs derived from production Hadoop traces at Facebook, IBM® Platform™ Symphony accelerated Hadoop by an average of 7.3x.
- ➔ In this project, the speed advantage of Symphony was most closely related to the shuffle size, with a consistent advantage up to 10KB and a slow decline with increasing shuffle size.
- ➔ Much of the Symphony advantage appears due to better scheduling latency. In a pure corner-case test of scheduling speed, this Symphony configuration outperformed the Hadoop configuration by 74x.
- ➔ These results are indicative only. We used the default settings for the Hadoop core and for Symphony, except where noted, and the two systems used the same HDFS, operating system, JVM, storage, and networks. Nevertheless, it is possible that different settings for Hadoop or Symphony could achieve different results.

NOTE: The tests in this STAC Report are not based on STAC Benchmark specifications. For more information, see the Background section of this report.

STAC
SECURITIES TECHNOLOGY ANALYSIS CENTER

Disclaimer

Contents

Summary

MapReduce, a framework for data-aware distributed computing, is enjoying substantial market uptake.[1] As users move more MapReduce projects from development to operations, they are paying increasing attention to production-related issues. One of these is performance: i.e., how quickly MapReduce jobs can be completed with a given amount of computing resources.

IBM® has created a proprietary implementation of the open-source Hadoop MapReduce run-time that leverages the IBM Platform™ Symphony distributed computing middleware while maintaining application-level compatibility with Apache Hadoop. IBM claims that this implementation of MapReduce run-time components can augment and accelerate various open-source and commercial MapReduce implementations. IBM asked STAC® to compare Apache Hadoop to Apache Hadoop accelerated by IBM Platform Symphony Advanced Edition in the same hardware environment, using an off-the-shelf workload written to the Hadoop MapReduce API. IBM felt that this result would be significant to customers deploying IBM InfoSphere™ BigInsights™ (IBM's flagship big data platform) or other Hadoop-based solutions. This report documents and analyzes the test results and describes the system configuration and test methodology in detail.

There are several publicly available benchmarks written to the Hadoop MapReduce API. For this project, we chose a methodology using the Statistical Workload Injector for MapReduce (SWIM) developed by the University of California at Berkeley.[2] SWIM provides a large set of diverse MapReduce jobs based on production Hadoop traces obtained from Facebook, along with information to enable characterization of each job. The hardware environment in the testbed consisted of 17 compute servers and 1 master server communicating over gigabit Ethernet. We compared Hadoop ver 1.0.1 to Symphony ver 5.2. Both systems used default configurations except where noted.

On average, Symphony performed jobs 7.3 times faster than Hadoop alone. This reduced the total processing time for all jobs by a factor of 6. The advantage enjoyed by Symphony slowly declined as the shuffle size increased beyond 100 kilobytes. IBM's assertion that this performance advantage derives from Symphony's superior scheduling latency was borne out by results from a simple "sleep" benchmark, in which Symphony was 74 times faster than Hadoop.

An important caveat is that results such as these may be sensitive to tuning of the framework software (both Hadoop and Symphony). The two frameworks were deployed on the same HDFS, operating system, JVM, storage, and networks, and the number of map and reduce slots were confirmed to be the same. It is possible that different settings for Hadoop or Symphony could achieve different results. If there are vendors who would like to see this comparison run against Hadoop configured with different settings, or using different Hadoop distributions or versions, we encourage them to contact STAC. IBM has expressed its willingness to support efforts to re-run these tests under such conditions.

[1] One analyst projects growth at a compound annual rate of 58% through 2018: www.marketanalysis.com/?p=279.
[2] https://github.com/SWIMProjectUCB/SWIM/wiki.

1. Background

MapReduce is a programming model to handle large data-processing requirements. It was described by Google in a now famous 2004 paper.[3] While Google only described the technique (rather than releasing any code from its implementation), Yahoo later created and open-sourced its own implementation of MapReduce called Hadoop, which is now gaining adoption in a range of industries.

As organizations move Hadoop into mainstream use, they are paying more attention to production-oriented issues. One of these is performance—that is, how quickly data can be processed with a given amount of computing resources. This speed governs tradeoffs among user-response times, consumption of power and space in the data center, and other common operational concerns.

In 2009, IBM's Platform Computing division created a MapReduce run-time environment that could be deployed together with Apache Hadoop. In this implementation, customers continue to use HDFS, the Hadoop MapReduce API, and various Hadoop utilities, but substantial portions of the Hadoop stack have been re-written, including the JobTracker, TaskTracker and Shuffle services. Having preserved the user- and developer-visible portions of Hadoop, IBM asserts that the implementation is compatible with applications written to Hadoop API's and that higher level big data tools like PIG, HBase and HIVE can run unmodified. The IBM components are based on the mature Platform Symphony grid middleware product and are proprietary software rather than open source. IBM claims that Symphony provides better production-oriented benefits than Hadoop's grid management and scheduling components, and it believes that these will appeal both to its existing Symphony customers and to other Hadoop users. One of those benefits is superior performance. For an independent check on that claim, IBM turned to STAC.

STAC is an independent technology-research firm that specializes in hands-on evaluation of high-performance technologies.[4] For more than six years, STAC has focused exclusively on the demands of financial trading, which is beset by engineering challenges in both "big workloads" (e.g., computation of derivatives risk, analysis of large time-series databases) and "fast workloads" (e.g., ultra-low-latency transformation and distribution of market data, automated trade execution). Toward this end, STAC facilitates the STAC Benchmark Council™, an organization of financial trading organizations and technology vendors (www.STACresearch.com/members) that specifies standard ways to measure the performance of trading solutions using real-world workloads. Vendor members gain a way to issue credible, business-relevant benchmarks. End-user firms get better information, which they can compare to their own systems using standards-based STAC Test Harnesses™ in their own labs.

We undertook this project because a growing number of Wall Street firms are deploying Hadoop. However, since adoption is still in early stages, the STAC Benchmark Council has not yet settled on benchmarks that are a clear fit for MapReduce. We therefore chose an off-the-shelf MapReduce benchmark that we felt was respectable, for reasons that we detail in the next section.

One could argue that STAC-M3™, the benchmark standard for time-series analysis of tick data, is appropriate for MapReduce. We are aware of a few firms using Hadoop for this workload. However, it is still relatively uncommon, and no Hadoop vendors have stepped up to the STAC-M3 plate so far. A STAC "Big Data" Special Interest Group (SIG) will kick off in 1Q13 to develop such benchmarks (among many other things). Individuals wishing to participate in this process (subject to Council rules) are invited to indicate their interest at www.STACresearch.com/bigsig.

[3] http://research.google.com/archive/mapreduce.html
[4] See http://www.STACresearch.com.

2. Test Methodology

2.1 Overview

For this project, we chose to use the Statistical Workload Injector for MapReduce (SWIM) developed by the University of California at Berkeley.[5] SWIM provides a large set of diverse MapReduce jobs based on production Hadoop traces obtained from Facebook, along with information to enable characterization of each job. In addition, SWIM scales the jobs based on the size of the cluster in order to make it feasible to run benchmarks when thousands of nodes are not available.

SWIM feeds jobs to the SUT[6] in a predetermined fashion that is independent of the SUT's ability to perform the work. Thus, the appropriate way to think of a SWIM workload is not as a multi-job batch process but rather as a series of discrete analytic tasks, where the response time to each job is the key metric. This is the kind of usage pattern one would expect in an environment where multiple users submit interactive requests throughout the day or jobs are triggered automatically in response to incoming events. (Anecdote suggests such use cases are on the rise in a variety of industries.)

We have no idea how the sampled and abstracted 2010 Facebook workload corresponds to the MapReduce workload of any other firm in any given industry (or even Facebook today). However, this workload does have one compelling benefit: it is derived from real, production Hadoop traces.

A caveat to SWIM is that it does not include any analytic processing, so it is not good at representing compute-bound jobs. Nevertheless, SWIM authors have advised us that their tests indicate SWIM workloads perform very similarly to the real workloads from which they were abstracted. If true, this implies that the Hadoop workloads the SWIM team has examined tend to be dominated by I/O.

IBM asserted that Symphony's performance in part comes from its scheduling speed. This was consistent with our findings that Symphony's performance advantage with the Facebook-based SWIM jobs declined when the payloads (particularly the shuffle) became large (see Results). To test this hypothesis directly, we took IBM's suggestion to run a quick "sleep" benchmark, following a precedent established last year for measuring MapReduce scheduling latency.[7] As described later, this test runs the "sleep" example job that comes with Hadoop, which does no work. To be clear, this is not designed to reflect a real-world workload but rather is a corner case designed to focus on a system's scheduling speed.

2.2 SWIM Benchmark

2.2.1 Workload

The SWIM benchmark tools pre-populate HDFS using uniform synthetic data based on samples from production-system traces at Facebook taken in 2010. The tools scale the job sizes based on the number of nodes in the SUT relative to the number of nodes in the Facebook system (3000). The tools replay the workload using synthetic MapReduce jobs.

The full Facebook 2010 file contains over 24,000 jobs. Even after scaling these down to match our cluster size, running all of the jobs would have taken 24 hours (this run time is independent of the SUT, since as we pointed out above, the submit times are independent of SUT speed). Multiplying that by two SUTS

[5] https://github.com/SWIMProjectUCB/SWIM/wiki.
[6] SUT = "System Under Test", or in STAC terminology, "Stack Under Test"
[7] http://www.slideshare.net/cloudera/hadoop-world-2011-hadoop-and-performance-todd-lipcon-yanpei-chen-cloudera

(pure Hadoop and Hadoop with Symphony) and three test runs (to avoid flukes) would have been far beyond our test window. We therefore chose the first *n* jobs that could be completed in a 20-minute test run. As it turned out, *n* was 302.

This set was a diverse combination of input, output, and shuffle sizes, and gaps between submit times. Because, as we will see later, the test results show that the performance advantage to using Symphony depended on size attributes of the jobs, it's important to understand how those attributes were distributed in our test sample and how that relates to the overall set of jobs in the Facebook 2010 SWIM job set (to make sure our sample wasn't biased).

The following series of histograms conveys this visually. (Keep in mind that the charts refer to the down-scaled sizes that were used in the tests. The jobs on which they are based were 3000/17 = 176 times larger.)

Figure 1 shows that input sizes were distributed fairly evenly, with the most common being tens of bytes, but many in the 100KB to 10GB range.

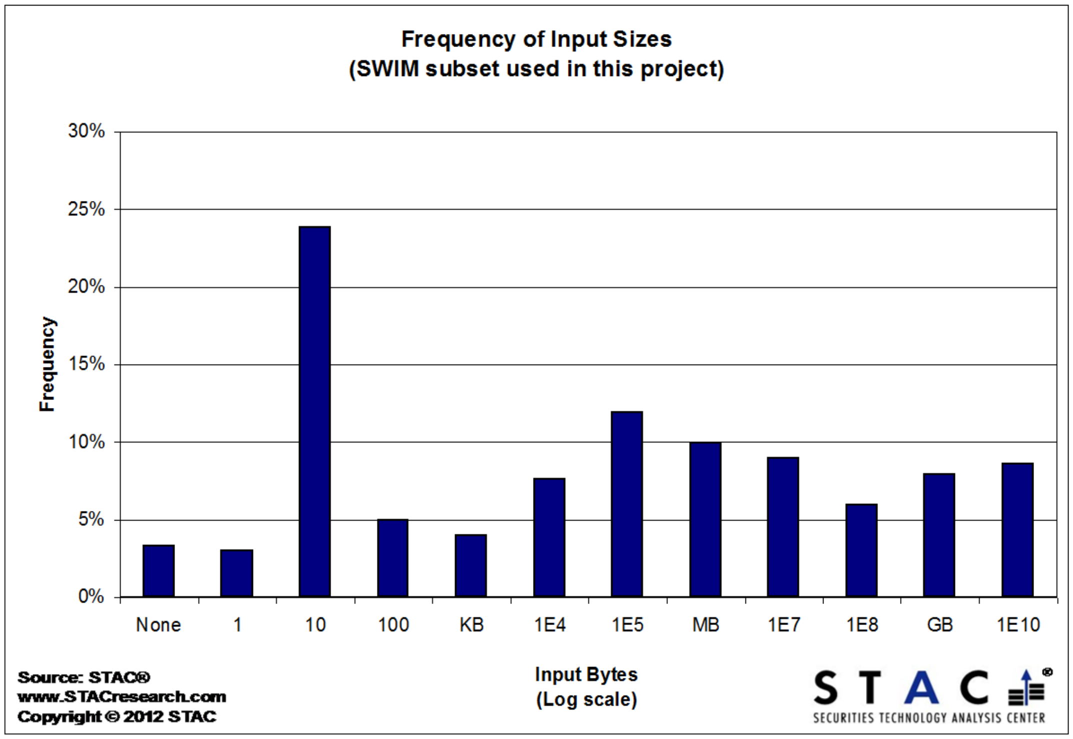


Figure 1

In Figure 2, output sizes are skewed a little lower, with tens of bytes being most common, outputs greater than a megabyte being fairly infrequent, and some jobs with no output at all. These combine to give the distribution of total input and output sizes shown in Figure 3. Jobs with total input and output of 10KB or less (the significance of which will become apparent a bit later) represent 42% of the sample. Figure 4 shows that shuffle sizes tend to be either large or small, with a third of jobs having no shuffle at all and about a third being a shuffle of 100KB or more.

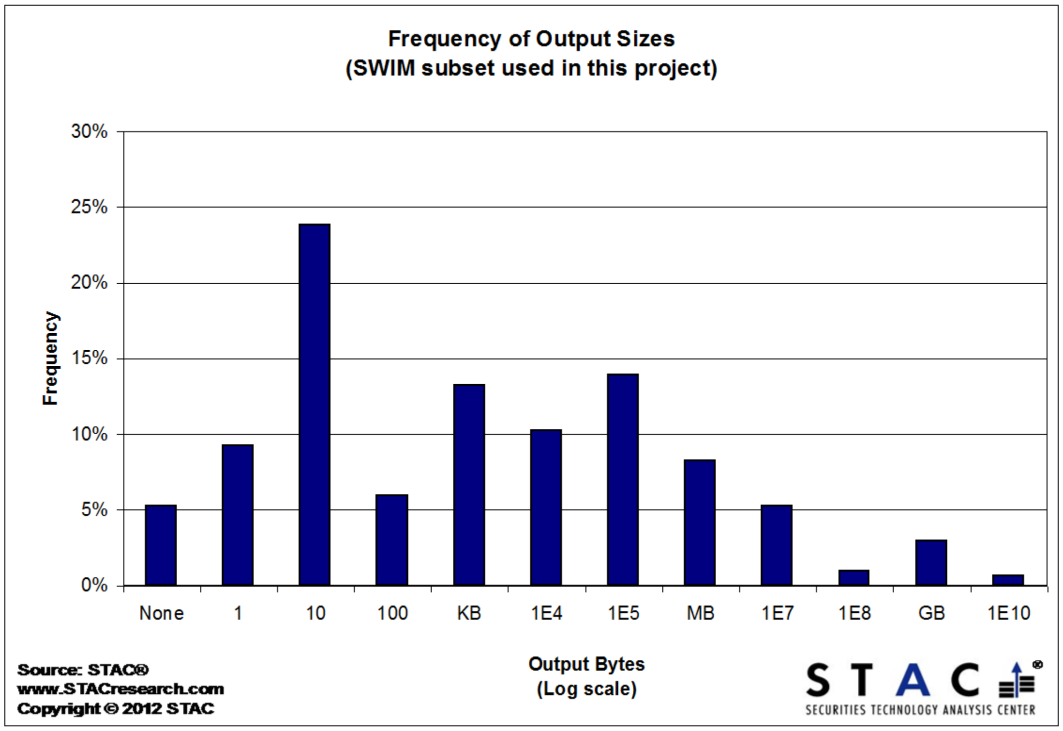


Figure 2

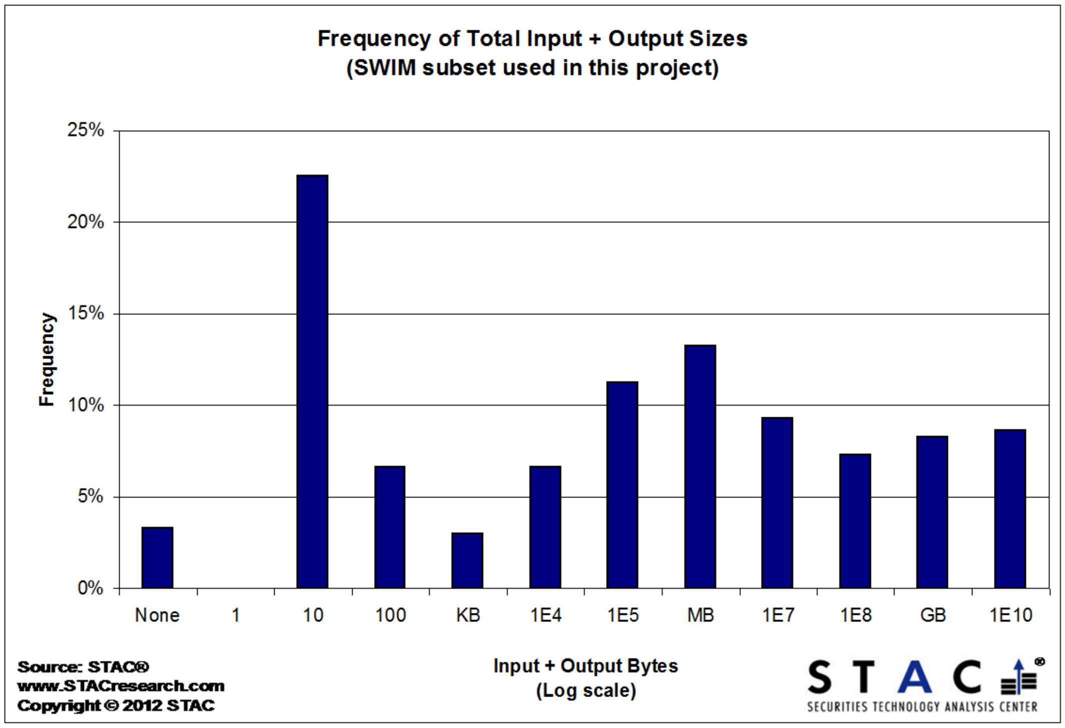


Figure 3

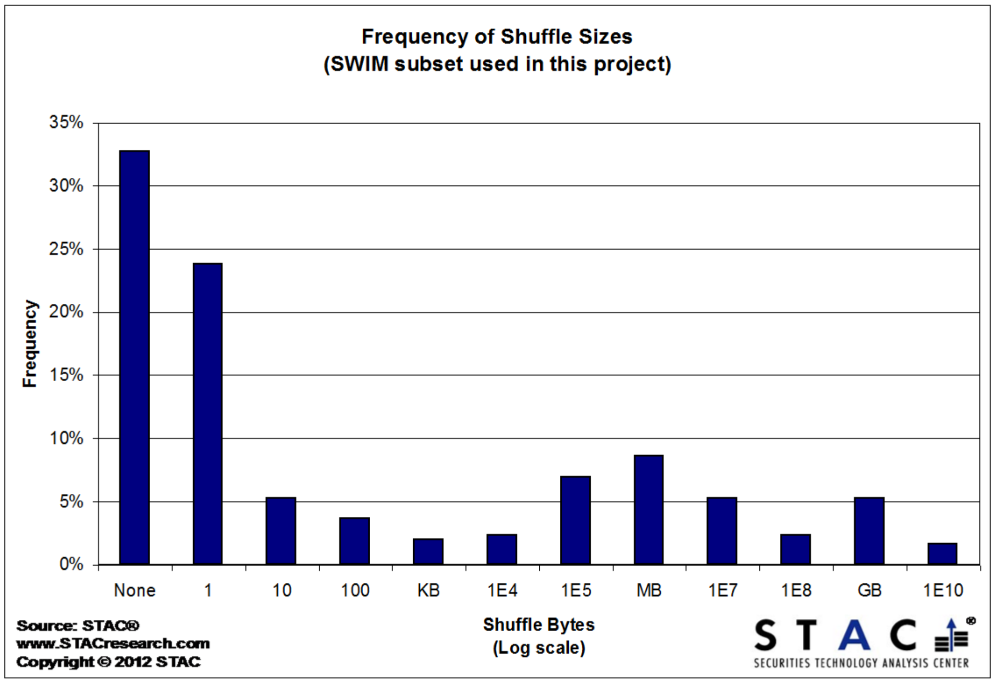


Figure 4

To provide some confidence that our sample of 302 jobs wasn't markedly different from the overall package of 24,000 jobs, Figure 5 shows the distribution of combined input and output sizes for the entire package (after down-scaling), for comparison with Figure 3.

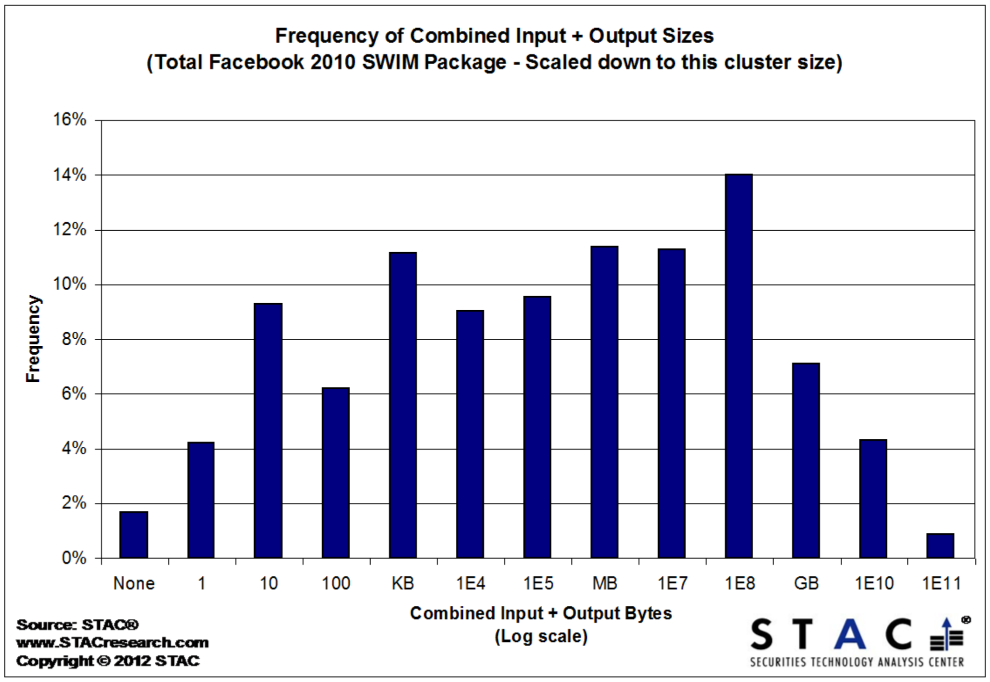


Figure 5

While it's clear that 10-byte input/output jobs were over-represented in our sample, the results show that what's important with respect to the question of bias is the proportion of jobs that were 10KB or less. As it happens, these jobs made up 42% of the entire scaled-down SWIM package, which is exactly the same proportion as in the sample used for the tests, as noted above.

The results also show that when the shuffle size reaches about 10KB, shuffle size is a better predictor of the relative performance advantage than is the combined input and output size. Figure 6 plots the shuffle-size frequency for the entire package, for comparison with Figure 4. While our sample is higher in jobs with byte-sized shuffles and lower in 100- and 1000-byte shuffles, it is very similar in the proportion of shuffle sizes above and below 100KB, which turns out to be the most salient division. 70% of our sample is below 100KB, which is the same proportion for the total job set.

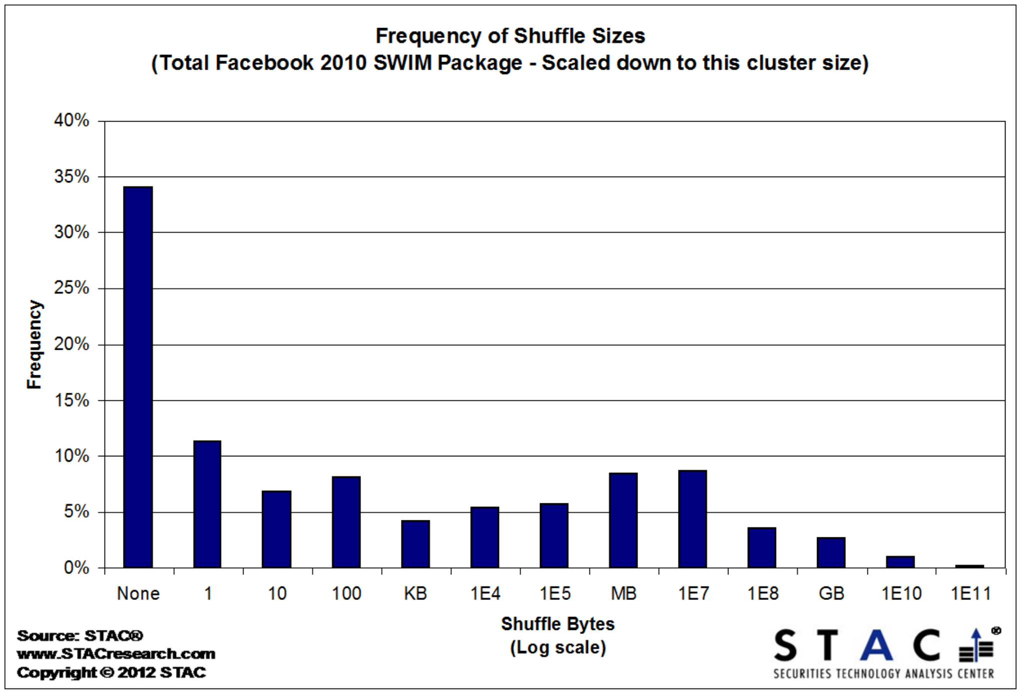


Figure 6

The foregoing figures show the input, output, and shuffle sizes independently of each other. But as we indicated earlier, the SWIM package includes a diversity of size *combinations*. To complete our characterization of the jobs used in this test, Figure 7 captures the correlation of shuffle size with the combined input and output size. It is immediately evident that the workload is dominated by two cases: those where the shuffle increases linearly with input+output (in fact, 1:1), and those where there is no shuffle at all.

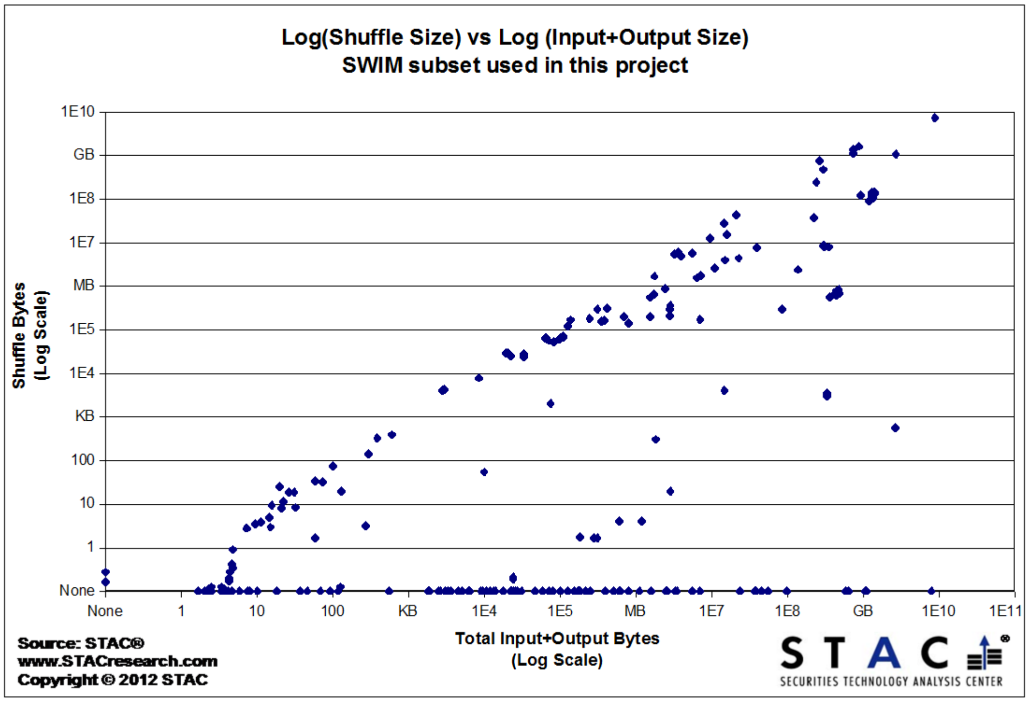


Figure 7

2.2.2 Test Prep

STAC created a set of orchestration scripts that started and stopped all required SUT components, ran the benchmark, and collected results.

The SWIM "GenerateReplayScript" program was used to generate job replay scripts. This program generates individual job run scripts and a main *run-jobs-all.sh* script in a *scriptsTest* directory to run all 24,442 jobs created from the Facebook trace. The *run-jobs-all.sh* runs the individual job scripts according to the timing from the trace. We modified this script to run only the first 302 jobs, as explained above.

For this test, the recommended parameters from the SWIM benchmark tools were used with one exception: the cluster size (parameter (highlighted in red, below) was changed from the default of 16 to 17, to match the number of nodes in the SUT. GenerateReplayScript uses this parameter to scale the workload down from the number of nodes in the original Facebook cluster (3000).

```
java GenerateReplayScript \
       FB-2010_samples_24_times_1hr_0.tsv \
       3000 \
       17 \
       67108864 \
       6000\
       scriptsTest\
       workGenInput\
       workGenOutputTest\
       67108864\
       workGenLogs\
       hadoop\
       WorkGen.jar\
       '/opt/sym/hadoop/conf/workGenKeyValue_conf.xsl'
```

The synthetic input, output, and shuffle data was created in HDFS using a Java class (HDFSWrite.java compiled to HDFSWrite.jar) and configuration files provided by the SWIM benchmark tools. The generated data consisted of random characters and was generated once for all test runs and used by both Hadoop and Symphony.

The MapReduce tasks are encoded in the WorkGen.java class provided with the SWIM benchmark tools. This class uses the Java MapReduce API to encapsulate the map function, the reduce function and code to run the entire job. The map function reads from input files and writes random data into intermediate output files based on parameters from the original Facebook traces. The reduce function also writes random data into output files based on parameters provided by the SWIM tools. The same WorkGen.jar file was used with both Symphony and Hadoop test runs.

STAC verified that the source code for the SWIM tools matched a fresh copy downloaded from github and that the same job logic (WorkGen.jar) was used in the Symphony and Hadoop runs. STAC also verified that the parameters used in SWIM configuration files matched those recommended in the SWIM tools package.

2.2.3 Procedure

We tested each system three times in succession to account for variability.

The following was the procedure for each test run with Hadoop:

1) Verify cluster servers are cleaned up from previous tests
2) Start HDFS
3) Start Hadoop map reduce services
4) Run SWIM test by invoking “run-jobs-all.sh” using Hadoop
5) Wait for SWIM test to complete
6) Save off test specific results and logs
7) Stop Hadoop map reduce services
8) Stop HDFS

The following was the procedure for each test run with Symphony:

1) Verify cluster servers are cleaned up from previous tests
2) Start HDFS
3) Start Symphony map reduce services
4) Run SWIM test by invoking “run-jobs-all.sh” using Symphony
5) Wait for SWIM test to complete
6) Save off test specific results and logs
7) Stop Symphony map reduce services
8) Stop HDFS

2.2.4 Time synchronization

STAC determined by code inspection that the SWIM code obtained all timestamps for a given job on the same master node by creating two Java Date objects and subtracting the difference. Thus no inter-server synchronization was required.

2.3 Sleep Benchmark

2.3.1 Overview

The tester specifies the number of mappers and reducers and the amount of time that each should sleep. Each task uses no CPU, memory, disk, or network I/O.

2.3.2 Test Setup

STAC created a set of automation scripts that managed starting and stopping all required SUT components, running the sleep job and collecting results. All sleep tests were run on the same test setup as the SWIM tests. For both Hadoop and Symphony, the sleep test was executed using the sleep example provided in the hadoop-examples-1.0.1.jar file from the Apache Hadoop package.

Following the precedent mentioned above, each map and reduce activity was configured to sleep for 1 millisecond. Via command line, the sleep job was configured to initiate 5000 mappers and one reducer.

STAC wrapped the sleep command with the Linux “time” utility in order to measure the time required to complete the full sleep job.

STAC verified that the same jar file was used for Symphony and Hadoop sleep tests.

2.3.3 Test Procedures

We tested each system three times in succession to account for variability.

The following was the procedure for each test run with Hadoop:

1) Verify cluster servers are cleaned up from previous tests
2) Start HDFS
3) Start Hadoop map reduce services
4) Run SLEEP test with Hadoop using the parameters “-mt 1 -rt 1 -m 5000 -r 1” and using the Linux “time” utility to track the elapsed time of the test.
5) Wait for SLEEP test to complete
6) Save off the elapsed time of the test.
7) Stop Hadoop map reduce services
8) Stop HDFS

The following was the procedure for each test run with Symphony:

1) Verify cluster servers are cleaned up from previous tests
2) Start HDFS

3) Start Symphony map reduce services
4) Run SLEEP test with Symphony using the parameters "-mt 1 -rt 1 -m 5000 -r 1" and using the Linux "time" utility to track the elapsed time of the test.
5) Wait for SLEEP test to complete
6) Save off the elapsed time of the test.
7) Stop Sympony map reduce services
8) Stop HDFS
9) Verify that no errors occurred during test runs and that all nodes are operational.

2.4 Limitations

As with any benchmark project, this one had limitations:

- Relevance of workloads. As discussed in the Test Methdology overview, there are many advantages to the diverse Facebook-based SWIM workload, but its relevance to the MapReduce workloads of any given reader is unknown.

- No compute. The SWIM jobs do not run any analytics. They merely move data back and forth. This constrains their relevance to those jobs that are primarily I/O-bound. Fortunately for the benchmark, the real world is full of those.

- Sensitivity to tuning. Hadoop and Symphony were deployed on the same HDFS, operating system, JVM, storage, and networks, and the number of map and reduces slots were the same. The settings for Hadoop and Symphony were default except as noted. It is possible that different settings for Hadoop or Symphony could achieve different results.

3. System Specifications

3.1 Overview

Figure 8 shows the SUT setup. Seventeen (17) IBM servers were the compute nodes and one (1) server was the master. STAC verified that all servers had a common hardware and software configuration. Each compute node had twelve (12) hard drives. Each node was connected by 1 Gigabit Ethernet which was used for all network connectivity between servers in the cluster. The IBM Platform Symphony and Apache Hadoop software was installed on the servers and the Hadoop HDFS was used in all test runs.

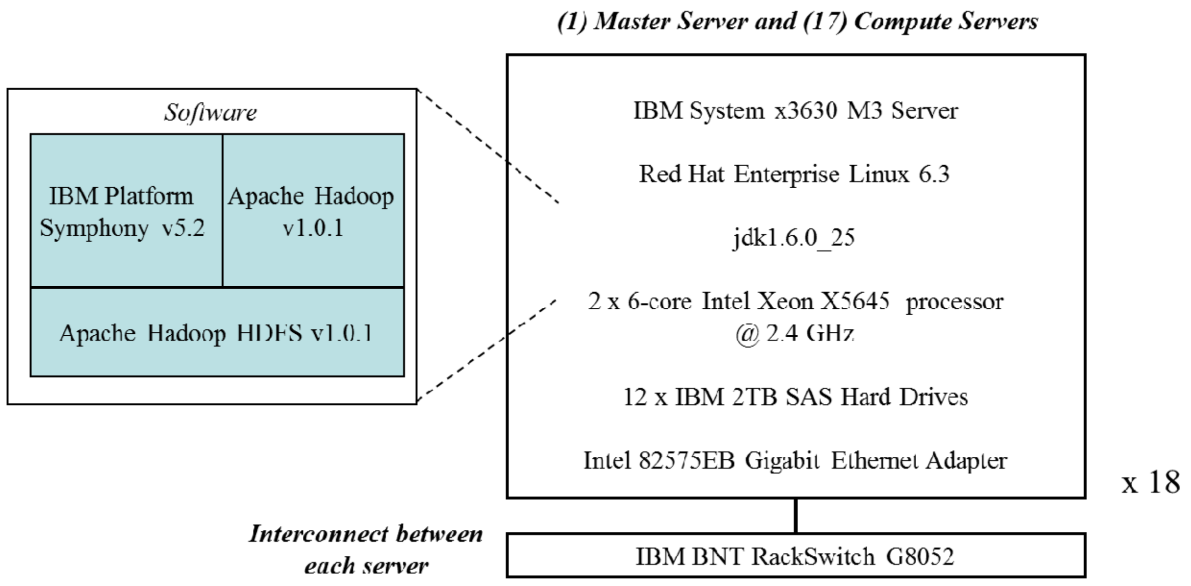

(1) Master Server and (17) Compute Servers
Software
IBM Platform Symphony v5.2
Apache Hadoop v1.0.1
Apache Hadoop HDFS v1.0.1
IBM System x3630 M3 Server
Red Hat Enterprise Linux 6.3
jdk1.6.0_25
2 x 6-core Intel Xeon X5645 processor @ 2.4 GHz
12 x IBM 2TB SAS Hard Drives
Intel 82575EB Gigabit Ethernet Adapter
x 18
Interconnect between each server
IBM BNT RackSwitch G8052

Figure 8: SUT Setup

Hadoop was configured so that each compute node had 8 mappers and 4 reducers, or 136 mappers and 68 reducers across the entire cluster, for a total of 204 slots. The total of mappers and reducers mapped 1:1 to the number of CPU cores.

Symphony allocates map and reduce tasks dynamically, so explicit task configuration was not required. The system was configured for 12 slots per node, or 204 in total, the same number of slots as the Hadoop configuration. We verified by inspection during sample runs that each server had no more than 12 map and reduce tasks running.

STAC verified that the same JVM tunings were applied to map reduce tasks on Symphony and Hadoop.

3.2 Master Node Configuration

3.2.1 Servers

Vendor Model	IBM System x3630 M3
Processors	2 x 6-core Intel Xeon X5645 @ 2.4Ghz

Cache	32kB L1, 256kB L2, 12288kB L3
Intel QPI	5.86 GT/s
Hyperthreading	Enabled
Memory	12 x 8192MB DIMMs, 1067 Mhz
Operating System	Red Hat Enterprise Linux 6.3 (Linux 2.6.32-279.el6.x86_64)
BIOS	Vendor: IBM Corp. Version: HSE121AUS-1.10 Release Date: 06/14/2012
Rack Units	2

3.2.2 Network Interfaces

Ethernet NIC (for HDFS, task scheduling and test orchestration)	Intel 82575EB Gigabit Ethernet Adapter
Driver	igb v3.2.10-k
Firmware	2.2-5

3.3 Compute Node Configuration

3.3.1 Servers

Vendor Model	IBM System x3630 M3
Processors	2 x 6-core Intel Xeon X5645 @ 2.4Ghz
Cache	32kB L1, 256kB L2, 12288kB L3
Intel QPI	5.86 GT/s
Hyperthreading	Enabled
Memory	12 x 8192MB DIMMs, 1067 Mhz
Operating System	Red Hat Enterprise Linux 6.3 (Linux 2.6.32-279.el6.x86_64)
BIOS	Vendor: IBM Corp. Version: HSE121AUS-1.10 Release Date: 06/14/2012
Rack Units	2
Storage	HDFS - 12 x IBM 2TB 7.2K 6Gbps NL SAS 3.5" HS HDD Other – 1 x 298GB

3.3.2 Network Interfaces

Ethernet NIC (for HDFS, task scheduling and test orchestration)	Intel 82575EB Gigabit Ethernet Adapter
Driver	Igb v3.2.10-k
Firmware	2.2-5

3.3.3 Software configuration

HDFS	
Hadoop	Version 1.0.1
System Configuration	See Appendix A
Java Virtual Machine	jdk1.6.0_25
JVM config for org.apache.hadoop.hdfs.server.namenode.NameNode, org.apache.hadoop.hdfs.server.namenode.SecondaryNameNode, org.apache.hadoop.hdfs.server.datanode.DataNode	-Xmx1000m

Hadoop	
Hadoop	Version 1.0.1
Java Virtual Machine	jdk1.6.0_25
JVM config for org.apache.hadoop.mapred.Child	-Xms1024m -Xmx1024m -XX:+UseParallelGC -XX:+DisableExplicitGC
JVM config for org.apache.hadoop.mapred.TaskTracker org.apache.hadoop.mapred.JobTracker	-Xmx1000m

Symphony	
Symphony	Version 5.2
Java Virtual Machine	jdk1.6.0_25
JVM config for com.platform.mapreduce.MRServiceJava.MRService	-Xms1024m -Xmx1024m -XX:+UseParallelGC -XX:+DisableExplicitGC
JVM config for com.platform.perf.purger.MainApp	-Xms64m -Xmx512m
JVM config for main app, com.platform.perf.dataloader.main,	-Xms64m -Xmx1024m

3.4 Network Interconnects

Ethernet switch used for HDFS, task scheduling and test orchestration	IBM BNT RackSwitch G8052

3.5 Benchmark software

SWIM	SWIM GitHub package at https://github.com/SWIMProjectUCB/SWIM , version - “a918466719dccc2250f80d6e54ada92ae1c2dc62”
SLEEP	Hadoop 1.0.1 package - /opt/sym/hadoop-1.0.1/hadoop-examples-1.0.1.jar

4. Results

4.1 SWIM Results

4.1.1 Headline results

Table 1 compares statistics for the job durations using Hadoop and Symphony. The mean job duration for Hadoop was approximately 34 seconds and for Symphony was 6 seconds, approximately 6 times shorter.

	Table 1 - JOB DURATION (seconds)		
	MIN	MEAN	MAX
Hadoop	31	34	184
Symphony	3	6	127

For each job, we also computed the ratio of the Hadoop job duration to the Symphony job duration. This equals the ratio of Symphony speed to Hadoop speed. Table 2 summarizes the statistics for this ratio, and Figure 9 shows them graphically. The average ratio was 7.3x.

Table 2 - JOB DURATION RATIOS (Hadoop duration/Symphony duration)			
MIN	MEAN	MEDIAN	MAX
1.4x	7.3x	7.8x	9.9x

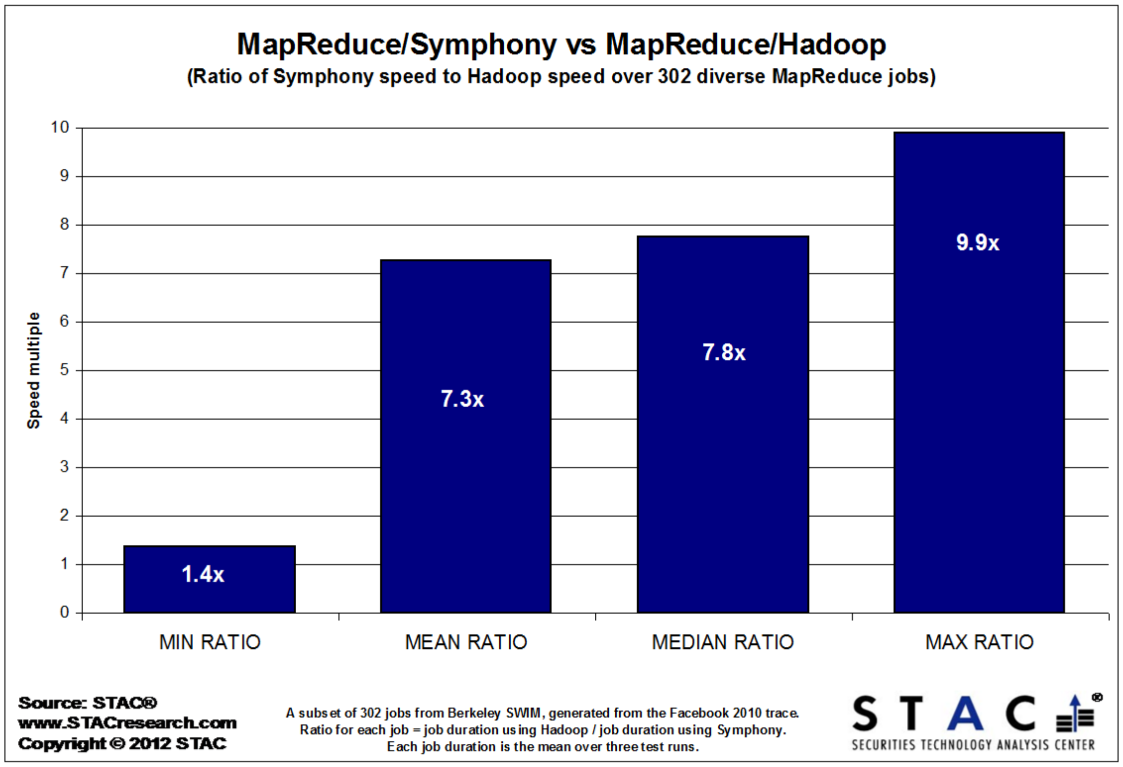


Figure 9

In the worst case, Symphony performed about 40% faster than Hadoop. In the best case, it performed 890% better (9.9x). In the next section, we analyze what differentiates these jobs.

4.1.2 Sensitivity of the Symphony advantage

Figure 10 plots Symphony's speed advantage for each job (Hadoop job duration divided by Symphony job duration) against the total bytes input and output. The advantage seems to hold steady until about 10KB, at which point a clear negative correlation develops. The correlation also becomes looser at that point.

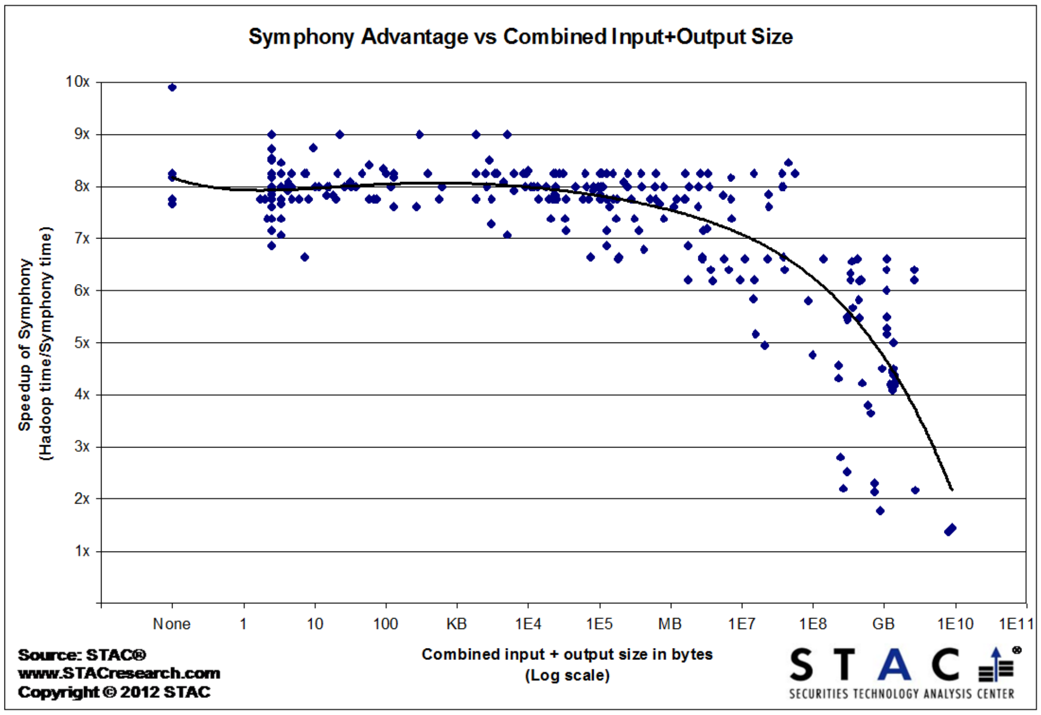


Figure 10

Recall from Figure 7, however, that there is a very strong correlation of shuffle size and combined input+output size, when shuffle is non-zero. So Figure 11 plots the Symphony advantage versus shuffle size. Here we see the same general shape of relationship, but with the opposite tightness: up to 10KB of shuffle size, the correlation is looser than the correlation with input/output size, but over 10KB, shuffle size seems to be a better predictor than input/output (and better than input or output alone, which are not shown).

The bottom line (short of using regressions to develop a full-blown performance model) is that on this cluster, Symphony enjoyed a roughly 8x advantage until either the combined input/output size or the shuffle size exceeded 10KB. After that point, shuffle size was the best predictor of relative performance.

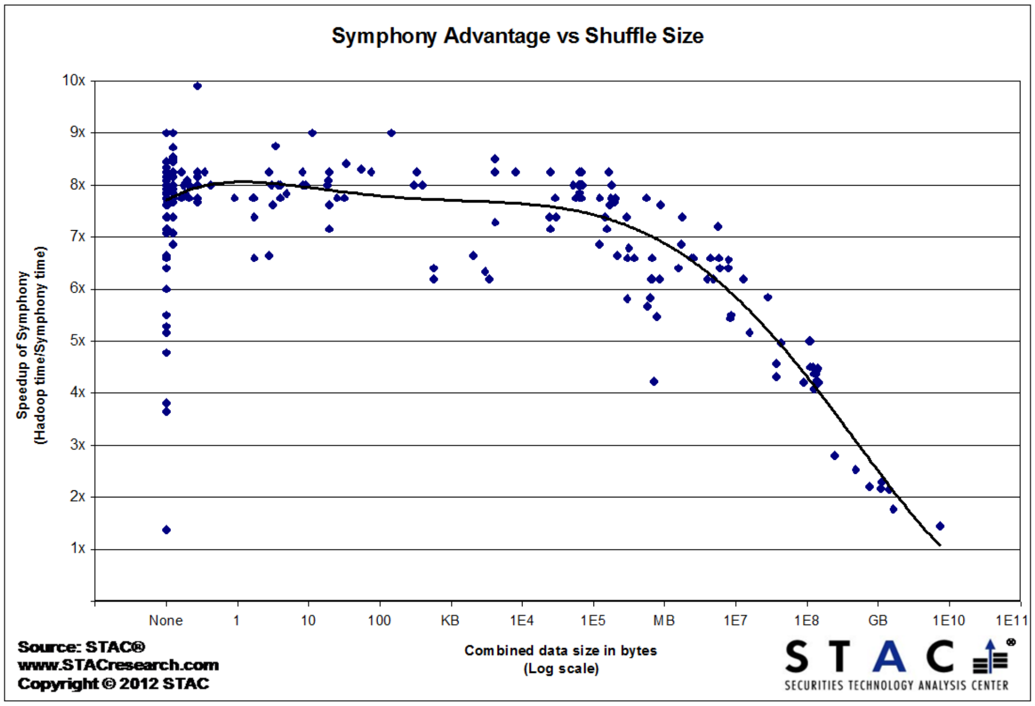


Figure 11

4.2 Sleep-Test Results

The negative relationship between Symphony's speed advantage and the shuffle size of the job (above 10KB) is consistent with IBM's contention that Symphony enjoys a significant advantage in scheduling latency. When most of the job time is I/O, improvements scheduling have a lower relative impact.

To measure the scheduling advantage directly, we took IBM's suggestion to run a simple sleep test, as described in Section 2.3. As Table 3 demonstrates, Symphony outperformed Hadoop in this benchmark by 74x. Hadoop took 912 seconds to complete the benchmark, while Symphony took 12 seconds.

Table 3 - RESPONSE TIMES FOR SLEEP TEST

	Hadoop		Platform Symphony	
Run	**Time to execute 5000 map tasks (sec)**	**Tasks per second**	**Time to execute 5000 map tasks (sec)**	**Tasks per second**
1	911.62	5.48	11.35	440.53
2	911.72	5.48	13.37	373.97
3	911.65	5.48	12.33	405.52
Mean	**911.66**	**5.48**	**12.35**	**406.67**

5. Vendor Commentary

IBM provided the following comments on this report (the views expressed are those of IBM):

These results are a significant independent validation that low-latency scheduling is important to achieving performance goals for a variety of real-world MapReduce workloads. In addition to the impressive results achieved (a 7.3x average decrease in job run-time and a 6x total reduction in the sum of job run-times) we wanted to point out an important corollary. While not explicitly measured in this benchmark, the nature of the workload and results clearly show that by using IBM Platform Symphony, considerably less hardware could have supported the same simulated Facebook workload without compromising service levels. We believe that readers who study these results closely will appreciate the enormous gains in efficiency and potential financial benefits. Deploying large scale grids is expensive, and by demonstrating performance gains on diverse MapReduce workloads that range between 40% and 890% it is safe to say that Platform Symphony offers significant opportunities for cost reduction for many customers.

Many in the financial services community already know IBM Platform Symphony as a high-performance, multi-tenant grid manager. It has proven itself to be effective at managing high-performance SOA workloads. For this and other reasons, it often supports risk analytics in investment banking where speed, agility and cost-efficiency are of critical importance.

From IBM's perspective, Hadoop MapReduce is just another grid workload that can be optimized through smarter scheduling and resource sharing. The ability to consolidate and manage diverse applications on a shared grid is something we view as important. Platform Symphony uniquely supports multiple types of workloads concurrently on the same shared grid, so the efficiency gains are cumulative. Faster processing, dynamic loaning and borrowing at run-time, and the ability to share infrastructure between applications and departments all improve performance and reduce cost. (This benchmark validates only the first of these three assertions.)

IBM Platform Symphony supports several Hadoop distributions, including IBM InfoSphere BigInsights, Cloudera CDH, and the open-source Apache distribution. (IBM encourages customers looking for the most developer- and user-friendly solution for complex, large scale analytics to consider IBM BigInsights, which provides tools like BigSheets and JAQL. See http://www-01.ibm.com/software/data/infosphere/biginsights/ *for more information.)*

Whatever the Hadoop environment, IBM Platform Symphony provides an excellent opportunity to boost performance and significantly reduce costs by taking advantage of low-latency scheduling and savings through sophisticated resource sharing on a flexible, multi-tenant grid.

6. Appendix A

The following Hadoop configuration files differed from their defaults as indicated:

core-site.xml

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?><!-- Put site-specific
property overrides in this file. --><configuration>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/opt/sym/hadoop-1.0.1/temp</value>
    </property>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://r1n1:8020/</value>
    </property>
</configuration>
```

hdfs-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
   <property>
        <name>dfs.replication</name>
        <value>1</value>
   </property>
   <property>
        <name>dfs.name.dir</name>
        <value>/opt/sym/hadoop-1.0.1/name/</value>
   </property>
   <property>
        <name>dfs.data.dir</name>

<value>/hadoop/sdb/data,/hadoop/sdc/data,/hadoop/sdd/data,/hadoop/sde/data,/hadoop/sdf
/data,/hadoop/sdg/data,/hadoop/sdh/data,/hadoop/sdi/data,/hadoop/sdj/data,/hadoop/sdk/
data,/hadoop/sdl/data,/hadoop/sdm/data</value>
   </property>
   <property>
        <name>dfs.hosts.exclude</name>
        <value>/opt/sym/hadoop-1.0.1/conf/excludes</value>
   </property>

</configuration>
```

mapred-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
   <property>
        <name>mapred.job.tracker</name>
        <value>r1n1:9001</value>
   </property>
    <property>
        <name>mapred.tasktracker.map.tasks.maximum</name>
        <value>8</value>
        <description>The maximum number of map tasks that will be run simultaneously
by a task tracker.
        </description>
    </property>
    <property>
        <name>mapred.tasktracker.reduce.tasks.maximum</name>
        <value>4</value>
        <description>The maximum number of reduce tasks that will be run
simultaneously by a task tracker.
        </description>
    </property>
    <property>
        <name>mapred.child.java.opts</name>
        <!--<value>-Xmx512m</value>-->
        <value>-Xms1024m -Xmx1024m -XX:+UseParallelGC -XX:+DisableExplicitGC</value>
    </property>
    <property>
        <name>mapred.local.dir</name>

<value>/hadoop/sdb/local,/hadoop/sdc/local,/hadoop/sdd/local,/hadoop/sde/local,/hadoop
/sdf/local,/hadoop/sdg/local,/hadoop/sdh/local,/hadoop/sdi/local,/hadoop/sdj/local,/ha
doop/sdk/local,/hadoop/sdl/local,/hadoop/sdm/local</value>
    </property>
<!--<property>
  <name>mapred.job.reuse.jvm.num.tasks</name>
  <value>-1</value>
</property>
-->
 <property>
        <name>mapreduce.jobtracker.staging.root.dir</name>
        <value>/user</value>
    </property>

</configuration>
```

About STAC

STAC is a technology-research firm that facilitates the STAC Benchmark Council™ (www.STACresearch.com/council), an organization of leading trading organizations and vendors that specifies standard ways to measure the performance of trading solutions. STAC Benchmarks cover workloads in market data, analytics, and trade execution.

STAC helps end-user firms relate the performance of new technologies to that of their existing systems by supplying them with STAC Benchmark reports as well as standards-based STAC Test Harnesses™ for rapid execution of STAC Benchmarks in their own labs. End users do not disclose their results. Some STAC Benchmark results from vendor-driven projects are made available to the public, while those in the STAC Vault™ are reserved for qualified members of the Council (see www.STACresearch.com/vault).

STAC also conducts customized research into the latest technology stacks for capital markets firms and vendors. STAC provides performance measurement services, advanced tools, and simulated trading environments in STAC Labs.

To be notified when new STAC Reports become available, please sign up for free at www.STACresearch.com.

STAC
SECURITIES TECHNOLOGY ANALYSIS CENTER