Survey of the Benchmark Systems and Testing Frameworks For Tachyon-Perf

Rong Gu,Qianhao Dong

2014/09/05

0. Introduction

As we want to have a performance framework for Tachyon, we need to consider two aspects in general. One is the testing workloads, it's about the test suites and targets. We have surveyed quite a few famous cloud benchmark systems at the first section of this survey. There, we mostly focused on how they build the test cases to test the cloud storage systems, such as testing Read/Write throughput.

On the other side, the Tachyon-Perf we aim to build is also a testing framework, a framework with extensibility. It should support users easily adding customized testing suites. The testing suites can share the same testing framework and utility. Testing frameworks have been studied a lot in software engineer/testing areas. Most testing frameworks also aim at high extensibility goal. In the second section of this survey, we studied some famous and widely-used testing frameworks related to our project.

Along with introducing the benchmark systems and testing frameworks, we also analyzed the guidance for Tachyon-Perf and the difference between the Tachyon-Perf we want.

1. Benchmark(Workload) Systems

1.1 AMPLab Big Data Benchmark

1.1.1 Introduction

The AMPLab's Big Data Benchmark provides quantitative and qualitative comparisons of five analytic framework systems, which are Redshift, Hive, Shark, Impala and Stinger/Tez. This benchmark is not intended to provide a comprehensive overview of the tested platforms, but it can still do a simple comparison between these systems with the goal that the results are understandable and reproducible.

This benchmark executes a set of relational queries across different data sizes on those five systems, and measures the response time. Since these systems have very different sets of capabilities, the queries is simple enough that most of these systems these can complete, such as scans, aggregations, joins, and UDF's (User Defined Functions). However, the data size can be very large to make full use of those systems.

In detailed, the data set consists of a set of unstructured HTML documents and two SQL tables, and the following show the schemas. The workload consists a set of queries to execute on these data. Certainly, the larger data size means more entries in the data set.

Documents

Unstructured HTML documents

Rankings

Lists websites and their page rank

```
pageURL VARCHAR(300)
pageRank INT
avgDuration INT
```

UserVisits

Stores server logs for each web page

```
sourceIP VARCHAR(116)
destURL VARCHAR(100)
visitDate DATE
adRevenue FLOAT
userAgent VARCHAR(256)
countryCode CHAR(3)
languageCode CHAR(6)
searchWord VARCHAR(32)
duration INT
```

1.1.2 Analysis

For now, the AMPLab's Big Data Benchmark is more like to evaluate the computing performance rather than the I/O throughput. And the benchmark needs to rely on an SQL analytic framework so that it cannot work just on a file system, like Tachyon or HDFS. And when evaluate a file system, we usually don't care the schemas data content.

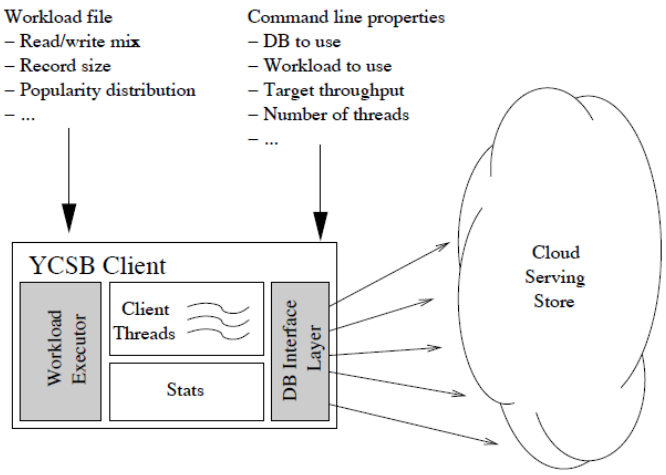
Nevertheless, the difference of the data sizes should be considered in any data-related benchmarks. Besides, the executing time is the best indicator to represent the performance.

1.2 YCSB

1.2.1 Introduction

YCSB (Yahoo Cloud Serving Benchmark) is a benchmark framework for evaluating the performance of the new generation**"key-value" and "cloud" serving stores**. It defines a core set of benchmarks and report results for several widely used system, such as Cassandra, HBase, Yahoo!'s PNUTS. And it is extensible to add benchmarks to a new database system.

The YCSB project comprises two parts. One is the YCSB Client, an extensible workload generator. And another is the Core workloads, a set of workload scenarios to be executed by the generator. The architecture of YCSB Client is shown below.



The loaded data and the workload are both generated by the YCSB Client, then the client will load those data to the database and execute the workload. The workloads can be set to different distributions (Zipfian, Latest or Uniform) and the size can be configured by the users.

According to this architecture, the two components Workload Executor and DB Interface Layer are designed to be extensible that it's easy for users to define a new workload or add a new database.

1.2.2 Analysis

The workloads of YCBS are mainly used for database system, which may not be suitable for file system, but the different distributions and sizes can be corresponded to different access pattern.

Now YCSB Client works in a single-node multi-threaded mode. For multi-node test, one can just start up clients on different servers, each running the same workload. But it's hard to test the performance of the client's locality. In a benchmark for distributed file system, like Tachyon or HDFS, we prefer to run clients on all the worker nodes in order to evaluate both the local and remote throughput.

A great lesson we learnt from YCSB is that a benchmark framework should be extensible. Not only easy to add new workloads, but also easy to be used under different environments.

1.3 HiBench

1.3.1 Introduction

HiBench is a Hadoop Benchmark Suite which contains 9 typical Hadoop workloads (including micro benchmarks, HDFS benchmarks, web search benchmarks, machine learning benchmarks, and data analytics benchmarks). Some of them are just based on Hadoop and some of them need to run on Mahout or Hive.

Since we consider the file system benchmark most on this topic, here we focus on the HDFS Benchmarks named enhanced DFSIO (dfsioe).

The enhanced DFSIO tests the HDFS throughput of the Hadoop cluster by generating a large number of tasks performing writes and reads simultaneously. The input data is automatically generated by its corresponding prepared script and a MapReduce job is adopt to perform the read and write operations. Finally it measures the average I/O rates of each map task, the average throughput of each map task, and the aggregated throughput of HDFS cluster.

1.3.2 Analysis

The enhanced DFSIO of HiBench is a benchmark aimed at testing file system performance, which is the same as our goal on Tachyon. We do the work in the same way. We also launch to a task on each work node of the cluster to measure the performance.

However, HDFS and MapReduce are binded into Hadoop. Therefore the enhanced DFSIO test just adopts a MapReduce job to distribute the testing tasks. However, as Tachyon is a computing-platform independent file system. We just write a light-weighted script and program to handle the distribution and collection work in Tachyon-Perf. This makes the code simpler and the whole test process runs efficient.

1.4 Hadoop HDFS Performance

1.4.1 Introduction

There already exist some benchmarking and testing tools in the Apache Hadoop distribution. The TestDFSIO and NNBench are two HDFS-related benchmarks.

TestDFSIO is a read and write test for HDFS which runs as a MapReduce job. It can be used to do stress test, to discover the performance bottlenecks and to give a first impression of how fast the cluster is in terms of I/O. The enhanced DFSIO we mentioned in the previous section is a variant of this TestDFSIO.

NNBench (NameNode Benchmark) is used for load testing the NameNode. It generates a lot of HDFS-related requests and sends all of them to the NameNode, where those requests are like creating, reading, renaming and deleting files. Again, the NNBench is run as a MapReduce job as well.

1.4.2 Analysis

Like other benchmarks on Hadoop, this test cannot be used to benchmark HDFS in isolation from MapReduce. It needs to distribute the benchmark to the whole cluster. But it’s not well to use the upper layer computing framework since time wasting when startup the job. Someone has tested and found that “by default the NNBench waits 2 minutes before it actually starts”. Moreover, the above benchmarks are part of Hadoop so it’s easy for users to install and use. That’s also one of our reasons to make Tachyon-Perf a part of Tachyon.

However, the NNBench mentions another side of the file system benchmark, which is the metadata test. Now Tachyon-Perf supports write and read test on each work node, and this mainly tests the I/O throughput. However, to evaluate the performance of a distributed file system, it’s important to measure the performance of metadata request, which is like the

NNBench. So we also consider to support a connection test to test the connect and metadata operation. It will hold a set of Tachyon Client and send metadata requests to the Tachyon Master, and measure the correctness and response time.

1.5 Spark-Perf

1.5.1 Introduction

Spark-Perf is a framework for repeatedly running a suite of performance tests for the Spark cluster computing framework. Now it supports 4 suites, which are Spark core tests, PySpark tests, Spark Streaming tests and MLlib tests.

The Spark core tests evaluate the performance of aggregate, sort and count operations in scala and PySpark tests the same but in python. The Spark Streaming tests are used to test the streaming performance. And MLlib tests consists of a set of machine learning algorithm tests.

Before running tests, user should configure options in a script file. When running, all the configured suites will run sequently and generate a result file.

1.5.2 Analysis

The test suite of Spark-Perf has a simple process logic, preparation, executing and outputting. This should be suitable for our framework so that it's extensible to add new test suite.

However, it's hard to add suite to Spark-Perf. Now each suite of Spark-Perf is separate and complex. Furthermore, there is no detailed documentation about how to add a test suite to Spark-Perf.

Spark-Perf is single-node and the test running distributed on the Spark cluster. For our performance framework on Tachyon, it should be distributed to support multi-client mode.

Spark-Perf uses sbt build tools and has the same configuration style of Spark, which means user can easily start testing without modifying the system environment. With this, we consider our framework to use the same environment of Tachyon.

2. Testing Frameworks

2.1 The DaCapo Testing Framework

2.1.1 Introduction

This benchmark suite is intended to be a tool for Java benchmarking. It is widely used by the programming language, memory management and computer architecture communities. It consists of a set of open source, real world applications with non-trivial memory loads.

The latest released suite consists of a set of benchmarks, which are shown below as an example. All of them support three data sizes (small, default, large), and some of them can be configured to be multi-threaded.

antlr A parser generator and translator generator.
bloat A bytecode-level optimization and analysis tool for Java.
chart A graph plotting toolkit and pdf renderer.
eclipse An integrated development environment (IDE).
fop An output-independent print formatter.
hsqldb An SQL relational database engine written in Java.
jython A python interpreter written in Java.
luindex A text indexing tool.
lusearch A text search tool.
pmd A source code analyzer for Java.
xalan An XSLT processor for transforming XML documents.

The benchmarks in the DaCapo suite are packaged into a jar file, and run via a benchmark harness which simplifies the execution of the benchmark and provides hooks to user-specified callbacks, warm-up iterations, validation etc. Following is the detailed work process:

- ✧ Before executing the first (warmup) iteration of a benchmark, the test data (if any) is extracted from the dacapo jar file, into a scratch directory, and any per-benchmark preparation takes place.
- ✧ During each iteration of the benchmark, some optional pre-iteration work takes place, then the pre-iteration user callback is invoked and the body of the benchmark is executed.
- ✧ When completes, the post-iteration user callback is invoked, and the per-benchmark validations take place. Some post-iteration work is then done (e.g. deleting output files), and the next iteration is executed.

Furthermore, the DaCapo also supports that user to write own benchmark instead of just defining some callbacks. If a user wants to plug in a new benchmark, he/she just needs to implement these components of a benchmark:

1. The benchmark program itself.
2. Input data to the benchmark
3. A configuration file
4. The benchmark-specific harness class.
5. Two targets in the build.xml ant script.

The build.xml should contain targets that a) download the source from its home server, and b) build it from source, and package its test data, harness etc into the suite. Test data should be zipped into a file called <benchmark>.zip, which will be automatically extracted into the scratch directory at runtime.

The benchmark harness is conventionally called *<Benchmark>Harness,* and is in the package *dacapo.<benchmark>*. This class must extend the abstract dacapo.Benchmark class, and many benchmarks simply need to provide a trivial constructor, and an iterate method that

performs one iteration of the script. The methods of the Benchmark class, in conjunction with the configuration file, perform most of the tasks required by a typical benchmark.

2.1.2 Analysis

On one side, the DaCapo Benchmark is not suitable for a distributed environment yet. Also, it's still a bit complicate to add a new benchmark. In most situations, we cannot add a new benchmark by just defining some callbacks. And in Tachyon testing cases, some actions like warm-up iterations is not a must.

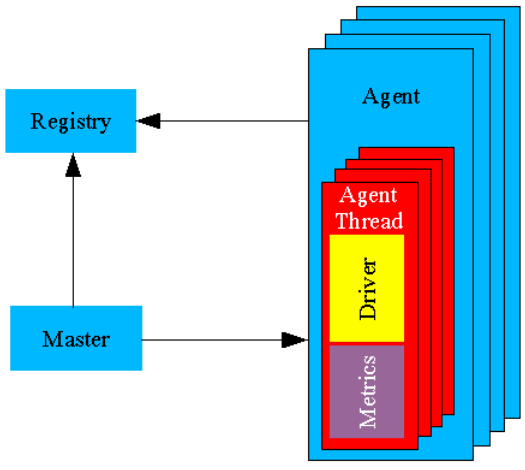
On the other side, we also learned a lot from DaCapo. The first thing is the abstract of the testing workflow. It's preparation, running, and collecting results. Also, the preparation can be performed by setting configurations. Another important thing we find that extending the abstract class is the straightest way to do the extension in DaCapo. We are also consider to adopt this style in Tachyon-Perf to make the framework be extensible .

2.2 Faban Harness and Benchmark Framework

2.2.1 Introduction

Faban is a free and open source performance workload generating and executing framework. It has two major components, the Faban Driver Framework and the Faban Harness.

Faban Driver Framework is an API-based framework and component model, which can help users develop new benchmarks rapidly. The major components of the driver are shown as below. The Registry registers all the agents to the Master, and the Benchmark is a grouping of one or more drivers. The Master starts all the agents and each agent starts a set of threads which run the benchmark.



Faban Harness is a tool to automate running of server benchmarks. It also serves as a container to host benchmarks allowing new benchmarks to be deployed in a rapid manner. The above Drive Framework provides a way to define the workload logic, but it still needs to glue the benchmark to the harness.

To add a new benchmark, first need to implement a benchmark class based on the Driver Framework, where three files is needed, the benchmark class, the configuration file and the submission form. Then, package the benchmark to a jar file by the ant build script. Finally, configure the Harness so that the benchmark can be detected by Harness to run.

2.2.2 Analysis

Faban is too general here. It maybe extensible enough to add any benchmarks. But one needs to handle and familiar with the Faban Driver Framework and the Faban Harness. It is even more complex than the DaCapo. Moreover, Faban uses the Ant build tools and we want Tachyon-Perf to use the Maven which is used in Tachyon.

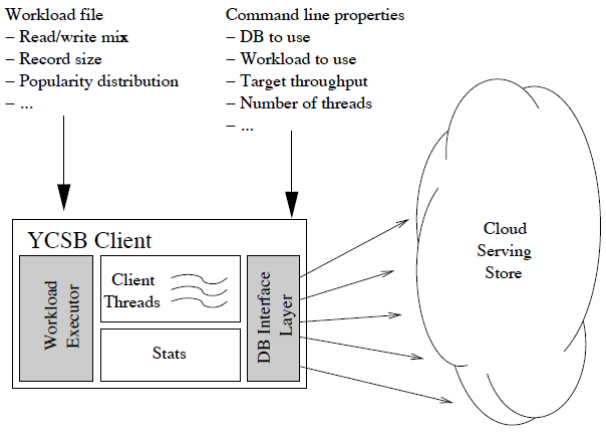
We indeed need a distributed framework, but we want to make the framework both simple and extensible. What we consider is that the management component only takes charge of starting clients and collecting metrics, while the benchmark component is easy to extend. The lesson we learned from Faban is that we can make Tachyon-Perf as a master/slave architecture.

2.3 YCSB

2.3.1 Introduction

YCSB (Yahoo Cloud Serving Benchmark) is a benchmark framework for evaluating the performance of the new generation**"key-value" and "cloud" serving stores**. It defines a core set of benchmarks and report results for several widely used system, such as Cassandra, HBase, Yahoo!'s PNUTS. And it is extensible to add benchmarks to a new database system.

The YCSB project comprises two parts. One is the YCSB Client, an extensible workload generator. And another is the Core workloads, a set of workload scenarios to be executed by the generator. All the workloads can be configured to different size and distributions. The architecture of YCSB Client is shown below.



As a benchmark framework, YCSB supports to add new workloads and new databases. A new workload can be a variant of current CoreWorkload, which can be implemented by

creating a new parameter file with new values for the read/write mix, request distribution, etc.

Another way to add a new workload is to create a new java class which extends from the *com.yahoo.ycsb.Workload,* then override the *init(),cleanup(),doInsert()* and *doTransaction()* methods, and specify the "workload" property of the YCSB Client to the added class.

Adding a new database needs to create a new java class as well. The new class extends the *com.yahoo.ycsb.DB* and then overrides the *init(), read(), scan(), insert(), update() and delete()* methods if necessary. Also, specify the "db" property of the YCSB Client.

2.3.2 Analysis

YCSB is designed to testing on database systems, so it may not be used on a distributed file system. But the way to add a new workload/database looks simple, just creating a new class and specifying the property.

YCSB measures the response time for all the benchmarks since it runs on database systems. But in benchmarks running on file system, they may have different metrics, e.g. throughput or response time. That makes us consider not only the extension of benchmarks, but also the extension of reports. Even though YCSB is designed to be extensible, some users have found that the usage of YCSB is kind of complicated.

2.4 Others

Here, we introduce some other testing frameworks we've learned during survey. They are not very close to our Tachyon-Perf here.

2.4.1 Grinder

The Grinder is a JavaTM load testing framework that makes it easy to run a distributed test using many load injector machines. Its test scripts are written using a dynamic scripting language, like Jython, and specify the tests to run. So it's easy to run any distributed test written by the script language.

2.4.2 Perf4J

Perf4J is a set of utilities for calculating and displaying performance statistics for Java code. It's similar to log4j but more customized. It's able to expose performance statistics as JMX attributes, expose performance graphs in a web application, and extensible to add user-defined metrics.

2.4.3 Caliper

Caliper is Google's open-source framework for writing, running and viewing the results of JavaMicrobenchmarks. It's simple to use and create own benchmarks, where an example is:

And the run

```
public static class Benchmark1 {
  @Benchmark void timeNanoTime(int reps) {
    for (int i = 0; i < reps; i++) {
      System.nanoTime();
    }
  }
}
```

command is:

```
$ CLASSPATH=build/classes/test caliper tutorial.Tutorial.Benchmark1
```

3. Reference

[1] AMPLab Big Data Benchmark: https://amplab.cs.berkeley.edu/benchmark/

[2] Cooper B F, Silberstein A, Tam E, et al. Benchmarking cloud serving systems with YCSB[C]//Proceedings of the 1st ACM symposium on Cloud computing. ACM, 2010: 143-154.

[3] YCSB Wiki: https://github.com/brianfrankcooper/YCSB/wiki

[4] HiBench: https://github.com/intel-hadoop/HiBench

[5] Hadoop Benchmark tools: http://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfsio-nnbench-mrbench/

[6] DFSIO: https://support.pivotal.io/hc/en-us/articles/200864057-Running-DFSIO-mapreduce-benchmark-test

[7] Blackburn S M, Garner R, Hoffmann C, et al. The DaCapo benchmarks: Java benchmarking development and analysis[C]//ACM SIGPLAN Notices. ACM, 2006, 41(10): 169-190.

[8]DaCapo:http:/www.dacapobench.org/

[9]Faban: http://www.faban.org/

[10] Cooper B F, Silberstein A, Tam E, et al. Benchmarking cloud serving systems with YCSB[C]//Proceedings of the 1st ACM symposium on Cloud computing(SoCC). ACM, 2010: 143-154.

[11]Grinder,http://grinder.sourceforge.net/

[12]Perf4J, http://perf4j.codehaus.org/

[13]Caliper, https://code.google.com/p/caliper/