This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2494595, IEEE Transactions on Services Computing

Common Pitfalls of Benchmarking Big Data Systems

Gwen Shapira, Yanpei Chen. Cloudera Inc.
{gshapira,yanpei}@cloudera.com

Abstract—It is challenging to get reliable performance benchmarking results. Benchmarking matters because one of the defining characteristics of big data systems is the ability to process large datasets faster. "How large" and "how fast" drive technology choices, purchasing decisions, and cluster operations. Even with the best intentions, performance benchmarking is fraught with pitfalls - easy to get numbers, hard to tell if they are sound.

This paper discusses five common pitfalls drawn from engineering and customer experiences at Cloudera, a leading big data vendor. These pitfalls are: "Comparing Apples to Oranges"- when too many parameters are modified and comparison is impossible, "Not Testing at Scale" - trying to test a big data system by extrapolating from an under-sized test system, "Believing in Miracles" - failing to question suspicious results, "Using Unrealistic Benchmarks" - using workloads far removed from what will realistically be used by customers, and "Communicating Results Poorly" - neglecting to communicate sufficient information for customers to understand and reproduce the results.

These pitfalls offers a behind-the-scenes look at internal engineering and review processes that produces rigorous benchmark results. Readers working on big data in both the industry and in academia can draw lessons from our experience.

Index Terms—Big data, performance, benchmarking, case studies.

1 Introduction

Done poorly, performance benchmarking produces disastrous results. Here are two stories from the authors' early careers.

An engineer ran a benchmark on a proof-of-concept 5-node cluster. Extrapolating the results, the engineer assumed the system will scale linearly and plans for a 50-node cluster to support the required production workloads. The production cluster ran for 30 minues before latency became completely unacceptable. It hit network bottlenecks not revealed at the proof-of-concept scale. As a result, rollout of the production system had to be delayed by a week as the scalability problems were being resolved.

A graduate student ran a Hadoop benchmark without realizing that he accidentally mounted the Hadoop Distributed File System (HDFS) on the departmental network filer. The benchmark promptly took down the filer for all professors, staff, and students at the department. The student received angry e-mails from the system administrators for days following the incident.

These two particular stories reveal how difficult it is to do performance benchmarking in a way that does not disrupt customer-facing, production systems, in a way that represent real-life workloads running there. Despite performance being an increasingly visible aspect of big data systems, there has not yet been many case studies of common benchmarking pitfalls, nor ways to avoid them. In this industry experience paper, we offer a collection of stories that illustrate important principles of conducting performance benchmarking and assessing others' results:

1) Workload and hardware choices should be relevant to the expected use of the product.
2) When modifying a standard benchmark, the modification should be documented and justified.
3) Testing big data means testing the system along multiple dimensions of large scale: Large number of jobs, jobs with large number of tasks, large data size, large clusters, and large nodes.
4) Tests designed to compare systems across a single parameter, e.g., new version of platform, must make sure this parameter was the only change. Changing additional parameters invalidates the comparison.
5) Having a model of expected behavior of the system is mandatory. Otherwise it is impossible to reason about the results.
6) Benchmark results should include enough information to reproduce the result - hardware, configuration, and workload.
7) Make sure any results tables and charts are clear, meaningful, and not misleading.

The stories in this paper come from internal engineering and customer experiences at Cloudera, a leading big data vendor. The pitfalls involve performance benchmarking of different components in the Hadoop ecosystem. This is not a comprehensive categorization of all possible mistakes, our goal is to give readers in both the industry and in academia tools with which they can improve their own work.

2 Comparing Apples to Oranges

We often run two tests, expecting only one parameter to change, while in fact many parameters changed and a comparison is impossible - in other words, we compare apples to oranges.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2494595, IEEE Transactions on Services Computing

Late 2013, the Hadoop community adopted MapReduce 2 (MR2) running on Yet Another Resource Negotiator (YARN) as the default MapReduce execution framework [1], [25]. This change offers functionality improvements over the original MapReduce, or MapReduce 1 (MR1) [12]. Many cluster operators did performance benchmarking on their own when they considered whether to upgrade. They initially reported a performance regression from MR1 to MR2.

What actually happened was that a straightforward comparison ended up comparing two different things, in other words, "comparing apples to oranges". Two issues led to this discrepancy.

One issue was that TeraSort, a limited but frequently used benchmark, changed between MR1 and MR2 [24]. To reflect rule changes in the GraySort benchmark on which it is based, the data generated by the TeraSort included with MR2 is less compressible. A valid comparison would use the same version of TeraSort for both releases, because map output compression is enabled by default as a performance optimization in Cloudera Distribution with Apache Hadoop (CDH). Otherwise, MR1 will have an unfair advantage by using more compressible data (Figure 1).

Another issue was the replacement of "task slots" in MR1 with "containers" in MR2. YARN has several configuration parameters that affected how many containers will be run on each node [5]. A valid comparison would set these configurations such that there is the same degree of parallel processing between MR1 and MR2. Otherwise, depending on whether hardware is over or under-committed, either MR1 or MR2 will have the advantage.

We committed these pitfalls ourselves in the early days of ensuring MR1 and MR2 performance parity. We regularly compared MR1 and MR2 performance on our nightly CDH builds, and the "regression" was caught the very first time we did this comparison. Our MapReduce and Performance Engineering teams collaborated to identify the code changes and understand what makes a valid performance comparison. This effort culminated in MR2 shipped in CDH5.0.0 at performance parity with MR1.

Here are some questions to ask regarding your own performance tests: If you are comparing hardware, are you running identical workloads? If you are comparing software, are you running your workload on identical hardware? Identical data, with identical formats and compression? Did the test procedure or test harnesses change?

3 Not Testing at Scale

Big data is called big for a reason. Testing small workloads on small clusters and expecting the results to extrapolate to large scale systems simply does not work.

"Scale" for big data systems can mean data scale, concurrency scale (number of jobs and number of tasks per job), cluster scale (number of nodes/racks), or node scale (per node hardware size). Failing to test "at scale" for any of these dimensions can lead to surprising behavior for your production clusters.

It is illustrative to look at another aspect of our efforts to drive MR2 to performance parity with MR1. We wanted to verify that MR2 and MR1 perform at parity when a large number of jobs are running. We ran SWIM [6], which submits many jobs concurrently over hours or even days, simulating the workload logged on actual production clusters. The first runs of SWIM on MR2 revealed a live-lock issue [3] where the jobs would appear as submitted, but none of them would make any progress. Figure 2 shows a web user-interface (UI) screenshot of a YARN Resource Manager that is experiencing live-lock.

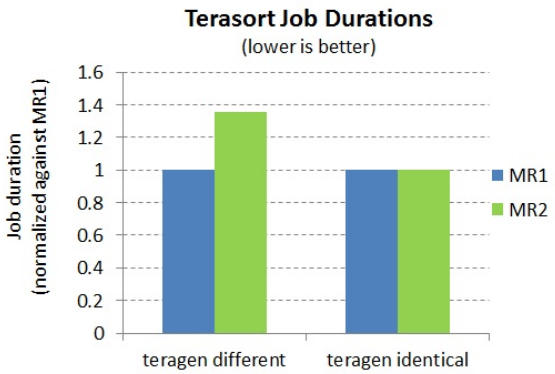


Fig. 1. Terasort performance when the data generation in MR1 and MR2 use different algorithms (left) or the same algorithm (right).

The cause of the live-lock is not straightfoward. Each MR2 job has an Application Master, which is a book-keeping type task that tracks the progress of the entire job. The Application Master still requires a YARN container to run. Without additional configurations, YARN would give all available resources to the Application Masters, leaving no room for the actual tasks. The tasks are behaving normally, but making no progress, i.e., live-lock.

This issue escaped detection in our other scale tests that covered a range of data, cluster, and node scales. The live-lock occurs only when all the containers in a cluster are taken up by Application Masters. On a cluster of non-trivial size, this means hundreds or even thousands of concurrent jobs. SWIM is specifically designed to reveal such issues by replaying production workloads with their original level of concurrency and load variation over time. In this case, we found a critical issue.

4 Believing in Miracles

If something is too good to be true, it is probably not true. We should always have a model of expected system behavior and bottlenecks. This way, we can tell if a performance improvement is reasonable, or too good to be true. Here are some recent "miracles" we debunked.

4.1 Miracle 1: 1000x SQL speedup

A customer reported that Impala [11], a SQL-on-Hadoop system, performs more than 1000x better than their existing relational database manage system (RDBMS). The customer wanted us to help them set up a new cluster to handle their growing production workload.

The 1000x difference is orders of magnitude larger than our own measurements [14], and immediately made us skeptical. Following much discussion, we realized that the customer was comparing very simple queries running on

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2494595, IEEE Transactions on Services Computing

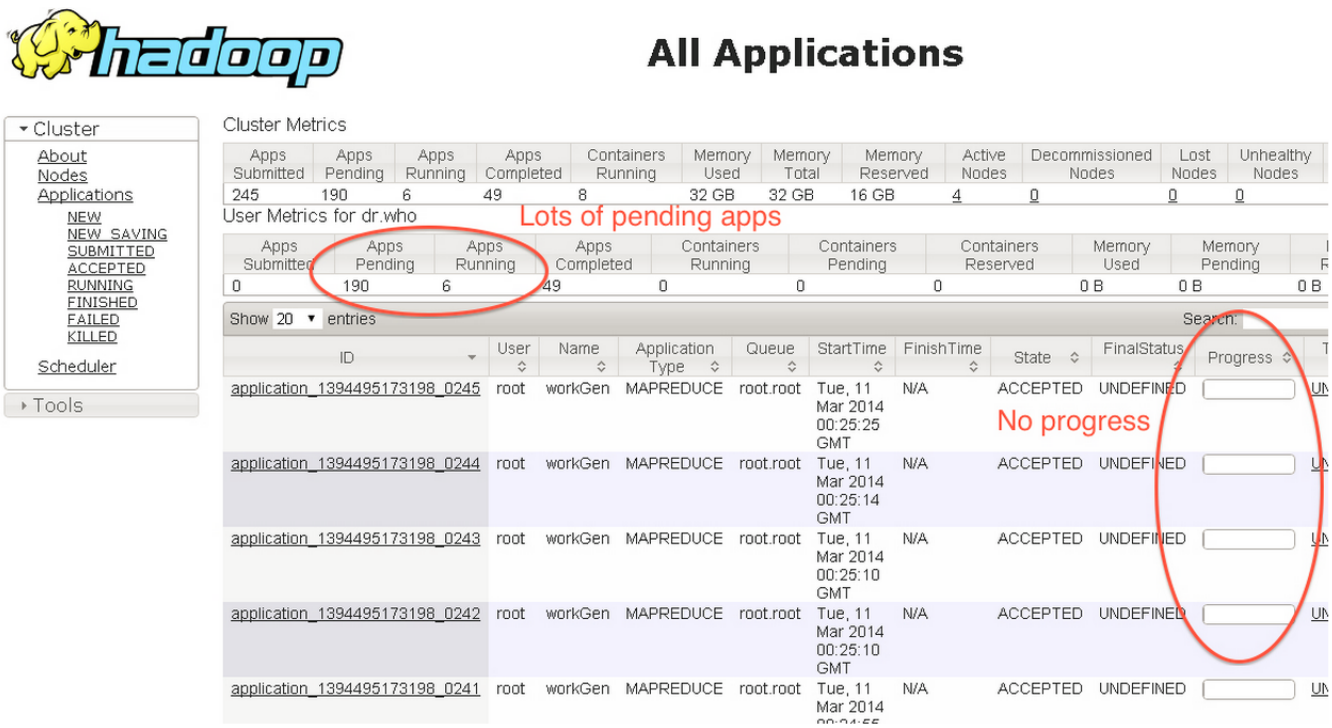


Fig. 2. YARN Resource Manager screenshot of live-lock symptoms.

a proof-of-concept Impala cluster versus complex queries running on a heavily-loaded production RDBMS system.

We helped the customer do an apple-to-apple comparisons, and turns out Impala still has an advantage (average 2x faster and up to 4.5x faster, from [14]). We left the customer with realistic plans for how to grow their data management systems.

4.2 Miracle 2: Indirect writes faster than direct writes

A customer asked us to run several configurations of Sqoop [2], a Hadoop-to-RDBMS connector used to bulk transfer data between the two types of systems. The intent was to find the configuration leading to the best performance of exporting data from Hadoop to RDBMS. Among other tests, we compared the performance of loading data to new partitions through Oracle's direct path writes, to loading the same data through normal inserts.

We expect direct path writes to be significantly faster, since they bypass the busy buffer-cache and redo log subsystems, writing data blocks directly to Oracle's data files. In this test, the normal inserts exercising an indirect write path were 3 times faster than the direct path writes. This suspicious result called for additional investigation.

The investigation revealed that Sqoop was exporting around 50GB of data to an otherwise idle Oracle cluster with over 300GB of memory dedicated to the buffer cache. Loading data into memory in a server with no contention is obviously faster than writing the same data to disk. We explained the results to the customer and recommended repeating the tests on a cluster with realistic workloads.

4.3 Miracle 3: 100x Hadoop sort speedup

A customer asked us for comment on a Hadoop sort benchmark result in the trade press. The result was more than 100x faster than what we found internally.

It turns out that the data size being tested was considerably smaller than the available memory in the cluster. In other words, a knowledgeable operator would be able to configure Hadoop in a way that the sort takes place completely in memory.

This departed from the common practice of configuring sort with data size much greater than total cluster memory. The more-than-100x gap came from the inherent hardware difference between memory and disk IO, rather than a difference between two software systems.

The ability to identify miracles requires us having models of expected performance beyond just a "gut-feeling". These models can come from prior results, or an understanding of where the system bottlenecks should be. Benchmarking without such models would give you a lot of numbers but not a lot of meaning.

5 Using Unrealistic Benchmarks

Unrealistic benchmarks are benchmarks where the workload, hardware, or presentation is chosen without regard of real-life requirements. Rather, these choices intend to inflate the capabilities of benchmarked system under test. Here are some warning signs of a biased benchmark:

5.1 Misleading workloads

Examples of misleading workloads include when someone ran benchmarks on 100GB of data when the system is intended for 100TB data sets, or when a transactional workload is used to test a system with mostly analytical use-cases. Terasort, a very popular benchmark for big

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2494595, IEEE Transactions on Services Computing

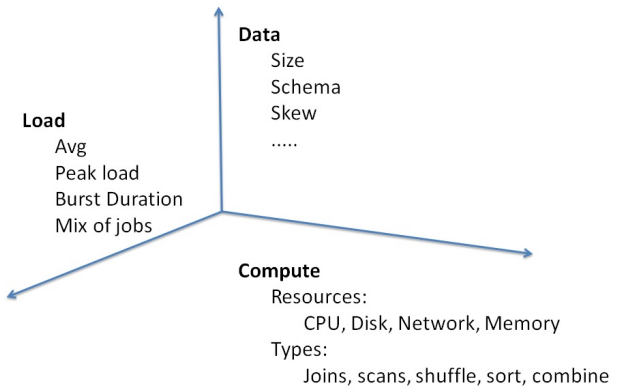


Fig. 3. Different dimensions of a big data workload.

data systems, is also potentially misleading. Terasort has very specific characteristics that stress very specific subsets of the processing subsystem. It is not necessarily a good benchmark to evaluate how the system will scale for all workloads, even though it is a useful first step in comparing different hardware configurations.

An example of how we avoid it at Cloudera: Terasort is only one job in our MapReduce performance benchmarking suite. We run a set of stand-alone, artificial jobs designed to stress in isolation different components of the MapReduce IO and compute pipeline; this suite includes open source jobs such as Terasort, and some jobs written in-house that we consider proprietary assets. We also use an open source tool [6] to replay full customer workloads with a large range of job sizes, types, and arrival patterns. We run both the stand-alone jobs and multi-job workloads under different dimensions of scale beyond just data size (See Section 3).

5.1.1 What makes a representative workload?

Cluster operators often find it challenging to reason about their own workload. If someone has no idea what their production workload looks like, they will have no idea whether the workload captured in a benchmarking study will match their own use case.

Figure 3 is a diagram to help readers characterize their workload. In broad strokes, there are three dimensions - the data characteristics, the compute characteristics, and the load-over-time characteristics [7]. Readers should ask themselves what is the following for their workload:

Data:

- How large is the data?
- What is the data schema, i.e., how do different parts of the data relate to each other?
- Is there any data skew, i.e., whether some data is accessed more frequently than others?
- How is the data represented and stored, i.e, what is the data format or data type?

Compute:

- What is the hardware bottleneck for the computation done? CPU, memory, disk, or network?
- If the workload is a SQL workload, whether the queries involve joins, scans, filters, group-by's?
- If the workload is MapReduce, whether the jobs need to do a lot of shuffle, sort, combiner operations, are they map-heavy or reduce heavy?
- If the workload is something else, characterize it in terms of the semantics of that processing paradigm.

Load:

- What is the load average?
- How long and how high are bursts in load?
- How do the mix of jobs or queries change over time?
- Are there diurnal patterns?

These questions should get readers started on characterizing their own workload. Answering these questions direct the discussion to other, more complicated, case-by-case characteristics that are also important to capture.

In a real-world example, we start by identifying the primary components of a production workload. If, say, MapReduce, HBase, and Impala are all involved, we need to make sure the test workload combines all of those. Drilling farther in, we may see that most of the MapReduce workload is map-only, with very little data being shuffled or reduced. We may also see that the HBase workload is 75% put and 20% get and 5% scans, and the Impala workload consists of star-schema joins that include one large table and many smaller tables, the results of which will be aggregated by day and month. We make sure our benchmark workload includes this level of details.

The next step is to note the data sizes, and either copy sufficient data from production, or write a small script that will generate synthetic data for the benchmark. It is recommended to note specific data patterns that should be part of the test - for example, if the workload involves sales data, it is likely that some regions and dates have significantly more records than others. This type of skew can impact performance and therefore benchmark results.

The last step is to check characteristics of the load patterns. Start with finding out how many concurrent jobs and queries typically run in production. Then decide whether to test with average load, peak load, expected future peak load, or perhaps the test should increase the load to the point the system breaks in order to find theoretical limits (test to destruction). Since multiple workloads are involved (MapReduce, Impala and HBase), we need to know if those workloads are typically executed together, or if they run during different times. For example, if we run Impala queries mostly during business hours and MapReduce during the night, the test should combine light Impala load with heavy MapReduce load and vice-versa, to simulate expected production conditions.

This type of planning leads to more meaningful results and is well worth the extra effort.

5.2 Premium hardware

Benchmark reports often contain results that come from hardware not typically used in real-life - solid state drives (SSDs) in environments that commonly use hard disk drives (HDDs), or premium SSDs not available in the general market. The Transaction Processing Council - C (TPC-C) [30] benchmark allows the use of hardware that is not available provided that availability dates are published. It is wise to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2494595, IEEE Transactions on Services Computing

check if the hardware choices make results irrelevant for guiding purchase decisions.

An example of how we avoid it at Cloudera: We have explored MapReduce performance for SSDs [20]. We were very conscious of SSD's prevalence in the market compared with HDDs. This prompted us to suggest to our hardware partners to track SSD performance-per-cost, which shows SSDs approaching parity with HDDs, even though the gap in capacity-per-cost remains large.

5.3 Cherry picking queries or jobs

Some reports pick very specific queries out of a standard benchmark, but cannot explain the choice with objective criteria that is relevant to the real-life use cases (or worse, does not disclose that a choice was made).

An example of how we avoid it at Cloudera: Our past Impala performance results [13], [14] used 20 queries derived from the TPC - Decision Support (TPC-DS) [32] benchmark. These queries were chosen based on what our customers observed for business intelligence (BI) use cases. They cover interactive, reporting, and deep analytic use cases. At the time, it was a major improvement over a frequently cited set of five queries [21] that were constructed without empirical backing from actual customer use cases. The 20 queries also represent a step forward from our own early efforts [9] using queries derived from TPC-H [31]. TPC-H is a less demanding benchmark with fewer and less complex queries than TPC-DS, while both are backed by customer surveys from vendors in the TPC Consortium. We have kept the set of 20 queries derived from TPC-DS to help ourselves compare against our own prior results, and we are well aware they are less than the full set of 99 queries in the official TPC-DS. Look for our future reports in this space.

5.4 Questions to ask all benchmark reports

To an extent all commercial and even research benchmarks are suspect of bias, since they are performed by a specific vendor or research group to promote their products or search project. Cluster operators can hold benchmark reports accountable by understanding their own workload and have a conversation about whether a product or research project addresses their specific use case. The following is a list of questions to ask.

- What hardware did you use?
- How was it configured?
- Is it similar to the hardware you are selling?
- Which jobs or queries did you run?
- Why do you think they mimic my workload?
- Were they modified from a well-known spec?
- How did you choose these specific jobs or queries?
- What if the jobs or queries are different?

With these questions, cluster operators force benchmark reports to discuss the limits of their own work.

6 (Mis)Communicating Results

Poor communication detracts from otherwise good performance results. Here at Cloudera, we check all external-facing benchmarking communications for the following:

We select a benchmark that

- Is unbiased (see Section 5),
- Exercise workloads relevant to actual customers, and
- Scales across data size, concurrency level, cluster size, and node size.

We report sufficient information for industry peers to assess the significance of the result, and to reproduce the tests if needed. This requires reporting

- The benchmark used and why we chose it,
- The metrics used and how we measured them,
- The hardware used and the software tuning applied.

These simple guidelines are often neglected in results coming from both industry and academia.

One more aspect of a good benchmarking report is whether the results have been independently verified or audited. The purpose of an independent audit is to have the above checks done by someone other the organization that produced study. Results that passed independent audit are more likely to be communicated clearly and completely.

There are several gold-standards for audit and verification practices established before the rise of big data:

Dedicated auditors

The Transaction Processing Council (TPC) [28] uses dedicated auditors. Each auditor is certified to audit a particular benchmark only after passing a test designed by the working group who initially specified that benchmark [29].

Validation kit and fair-use rules

The Standard Performance Evaluation Corporation (SPEC) [27] uses validation checks built into benchmarking kits, fair-use rules governing how the results should be reported, and review by the SPEC organization, which encompasses many industry peers of the test sponsor.

Peer review

The official Sort Benchmark [26] has new submissions reviewed by past winners. The winners would "hand over the torch" only if new entries are sufficiently rigorous.

There are not yet any widely accepted audit and verification processes for big data. The need for complete and neutral benchmarking results sometimes gets diluted by the need to stand out in the trade press. However, the past year has seen a phenomenal growth in the level of performance knowledge in the broader technical community. Every benchmark report is now scrutinized by industry and academia peers. This increases the need to be rigourous and open about performance benchmarking results.

6.1 A picture in need of 1000 words

Performance reports often use graphs to summarize results. Poor graphs can unintentionally or deliberately mislead readers. We include here an example of a poorly-communicated graph and a better-communicated graph.

Figure 4 comes from one of the author's early work measuring the performance of distributed databases. None of the axes were labeled, the performance metrics are unclear,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2494595, IEEE Transactions on Services Computing

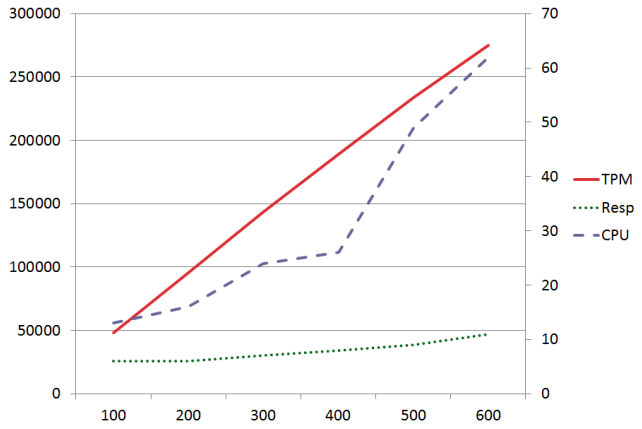


Fig. 4. An example of a poorly-communicated graph.

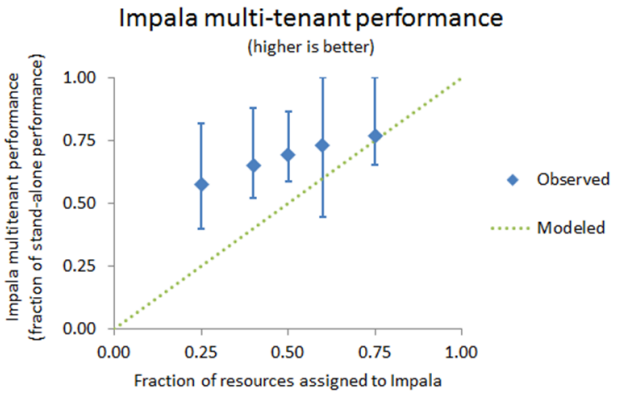


Fig. 5. An example of a better-communicated graph. It still needs a lot of surrounding text for a full explanation.

and the test scenario and test settings are unclear. Even the graph's creator cannot recollect what was being displayed.

Here is what the authors together deciphered. The graph is showing database throughput measured in transactions per minute (TPM), query latency (response time), and CPU utilization of the system. The horizontal axis is likely showing the number of concurrent user or a similar sense of "load". CPU utilization increases under higher load, and the right vertical axis is of the correct numerical range for CPU utilization in percentages. The left vertical axes could be either TPM in number of queries, or response time in milliseconds. There is no way to tell without additional information. Without proper labeling and documentation, every well-done performance benchmarking studies lose their meaning over time.

Figure 5 appears in a recent Cloudera blog [9]. It is a better communicated graph. Without further text, here is what the figure communicates: The graph shows Impala multi-tenant performance, with the metric being a normalized, unitless metric of multi-tenant performance as a fraction of stand-alone performance. This metric has the property that "higher is better". The graph comes from five tests, with Impala receiving an increasing fraction of system resources ranging from 25% to 75%. There is large performance variation as shown by the error bars. There is also a model of desired system behavior, one that suggests Impala should show fraction x of stand-alone performance when given fraction x of system resources.

There is still a lot of information missing from the graph:

What was the workload being tested? It was Impala running concurrently with MapReduce on the same cluster, specifically one MapReduce job concurrent with one Impala query at a time. The cluster is configured to give fraction x of the resources to Impala, with MapReduc receiving the remaining fraction $1 - x$.

What metric is being normalized? Impala query duration when the cluster is executing only the Impala query vs. when the cluster is executing an Impala query with a MapReduce job.

What do the error bars show and why are they so large? Each data point is the arithmetic average of 56 MapReduce job and Impala query combinations. The 56 job-query combinations cover a large range of MapReduce job types and Impala query types, hence the large variation. The error bars themselves represent 25th to 75th percentile range across the job-query combinations.

What fractions of resources were assigned to Impala for the 2nd and 4th markers? It is not immediately clear from the ticker mark intervals on the horizontal axes, but the 2nd and 4th markers represent 40% and 60% of the cluster resourcess assigned to Impala.

What about MapReduce multi-tenant performance? The companion graph for MapReduce multi-tenant performance is Figure 6.

The graph guides the discussion to more interesting topics, such as why should the performance model be as it is, whether the test workload is realistic and useful, and whether the performance is actually good.

The following is a list we use to check our own graphs.

- Does the graph need a title, or is one unnecessary based on surrounding text?
- If the graph shows multiple data series, is a legend displayed or included in the graph caption?
- Are the graph axes labeled? Do the labels include appropriate units?
- Is there one or several performance metrics being graphed?
- If there is a single performance metric graphed, is it on the vertical axes?
- As big data performance is variable from measurement to measurement, are error bars necessary?
- If a line or curve is drawn connecting two markers, is it reasonable to extrapolate across a range of unmeasured settings?
- If there is a model of desirable behavior, is the model also shown on the graph?

Big data systems have evolved to the point where the meaning of performance can be complex, and the number of relevant metrics can be large. This is especially true when we consider different big data processing engines not as stand-alone components, but as concurrently active frameworks sharing resources on the same cluster. Thus, we should make every effort to ensure clear communication.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2494595, IEEE Transactions on Services Computing

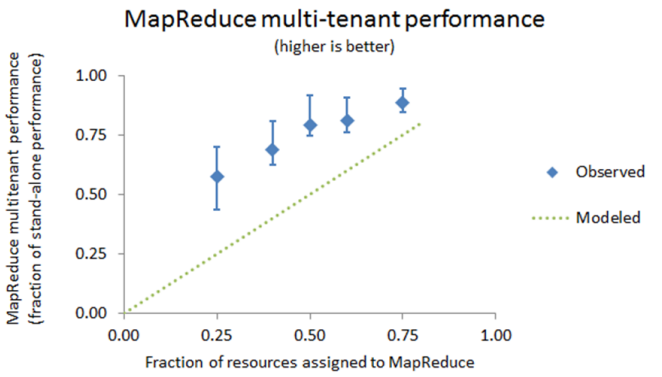


Fig. 6. Companion graph to Figure 5, showing MapReduce multi-tenant performance.

6.2 Following our own advice - Miracle checking

Earlier we highlighted the need to check any miracle results for their validity. In Figure 5, the fact that multi-tenant performance turned out better than modeled is an immediate warning sign for a possible "miracle" result. Since Impala and MapReduce were concurrently active for the multi-tenant scenario, the results would be reasonable if MapReduce multi-tenant performance suffered and was lower than modeled. The opposite happened, and the companion MapReduce multi-tenant performance also exceeded our model (Figure 6). This was indeed a "miracle" result worth understanding.

Two factors caused this result. First, our test scenarios run through 56 pairs of concurrent MapReduce jobs and Impala queries, one pair at a time. For any given pair, either the MapReduce job or Impala query would complete first. Thereafter, the remaining MapReduce job or Impala query would receive the entire cluster's resources. In other words, our test procedure systematically skewed the results in favor of being better than the model.

Another reason is the statistical multiplexing of hardware resource demands. This is a subtle effect of multi-tenant processing. For our tests, a MapReduce job and an Impala query need different hardware resources at different times. The resource demands are frequently not overlapping, i.e., statistically multiplexed. This multiplexing happens due to the range of processing covered in the 56 job-query pairs and the different design of the MapReduce and Impala processing engines. In other words, the cluster hardware is better utilized when there are different kinds of processing present on the system.

Understanding the cause of this "miracle" result helped us improve our test scenario. Our latest multi-tenant workloads run many concurrent Impala queries and MapReduce jobs, so that the system resources are fully utilized regardless of statistical multiplexing. Also, we run continuous streams of MapReduce jobs and Impala queries, such that for the duration of measurement, there will always be two different frameworks competing for resources.

7 Practitioner Use of Benchmarks

There are few cases when a big data practitioner would need to run a benchmark:

- Validating an existing system following a system upgrade or migration
- Compare between technologies for a new system
- Assessing the impact of workload changes

In our experience, benchmarks are used in different ways in each scenario.

When upgrading or migrating an existing system, benchmarks validate whether the new infrastructure delivers expected performance. It is key to ensure apples-to-apples comparisons between different setups.

The new infrastructure should be validated with the existing workload. If the workload includes batch jobs, simply replicating data to the new system and running the batch jobs is all that is required. If the workload is more interactive, then a load-generation harness such as HP LoadRunner [18] or Apache JMeter [4] is often used.

In some cases, the specific production workload cannot be replicated in the new environment. In those cases, it is very common to choose an industry standard benchmark to try to emulate the production workload.

When trying to compare technologies for a newly designed system, insist on full disclosure, and make sure the benchmarks used are a good substitute for the workload planned for the cluster. Specifically, ensure the benchmark report makes apples-to-apples comparisons against competing technologies.

Some common benchmarks used include: Terasort and SWIM [6] for MapReduce, TPC-DS [32] and TPC-H [31] for SQL-on-Hadoop, and YCSB [33] for NoSQL key-value stores. Depends on the workload planned for the cluster, they may or may not be appropriate.

The gold standard for validating results is independent audit. Some commericial vendors who use industry standard benchmark show such results. An alternative to independent audit is to try to reproduce the reported results on a pre-production environment. We have seen cases where a published performance result cannot be reproduced on identical trial systems provided by cluster operators.

When running a home-grown benchmark kit based on real workloads, independent audit is nearly impossible and reproducing the result may simply reproduce built-in errors. There, a good practice is to compare the measured performance to published results of similar systems, review the differences, and see whether the performance differences can be explained with a reasonable model. We discussed some examples of reviewing differences in Section 2 and the importance of performance models in Section 4.

8 Related Work

8.1 Qualities of a good benchmark

The criteria for a good performance benchmark have been the topic of decades of publications [15], [16], [19]. Prior work has identified the following essential properties:

- Representative: The benchmark should measure performance under real life environments and use metrics that are relevant to real life applications.
- Portable: The benchmark should be fair and portable to competing solutions that target the needs of the same applications.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2494595, IEEE Transactions on Services Computing

Benchmark	Publications
SPECjbb (2000 - 2005)	1,050
TPC-C	760
SPEC SFS	730
SPECweb (96 - 2009)	700
TPC-D/H	650

TABLE 1
Benchmark Result Publications

- Scalable: The benchmark should measure the performance of systems within a wide range of scale. As technology progresses, systems increase in scale and performance capabilities. The benchmark should be able to accommodate for that increase.
- Verifiable: The benchmark should prescribe repeatable measurements that produce the same results and can be independently verified.
- Simple: The conceptual elements of the benchmark should be reduced to a minimum and made easily understandable. The benchmark should also abstract away details that represent case-by-case configurations or system administration choices that do not affect performance.

Selecting a benchmark with the above qualities is a first step towards addressing many of the pitfalls identified: non-representative benchmarks lead to the unrealistic benchmarks pitfall, non-portable benchmarks make it easier to commit the comparing apples-to-oranges pitfall, non-scalable benchmarks lead to the not testing at scale pitfall, non-verifiable benchmarks make it easier to believe in miracles, non-simple benchmarks make it easier to miscommunicate results.

Unfortunately, the crowded field of emerging big data benchmarks often fall short on the "representative" characteristic. The two most critical shortcomings we see are (1) failing to capture a multi-job, multi-query workload and (2) failing to provide empirical evidence to justify the choice of jobs, queries, and data that are included in the benchmark. Our prior work [8] contains a critique of several recent big data benchmarks.

8.2 Successful benchmarks and their making

A few benchmarks have reached the level of active industry standards. When it comes to benchmarks measuring complete or end-to-end systems, two organizations have dominated: SPEC and TPC.

Each organization has published a number of benchmarks with various degrees of success. One criteria for success is the level at which the benchmark is being used by various organizations. While internal use is difficult to quantify, external publication of benchmark results is easy to tally and represents a clear success criteria. Table 1 shows the most published benchmarks from TPC [28] and SPEC [27].

Of these benchmarks, TPC-C and TPC-D/H followed a similar process of finding representative customer workloads that provide insight regarding how to create a big data benchmark. Little has been written about the insider's views of the benchmarks definition process. The making of TPC-C is published only recently [10].

The key ideas from this process are:

- Ground the benchmark based on empirical survey of customer use cases, in TPC-Cs case a survey of hundreds of customers across multiple countries.
- Develop abstract functions, datasets, and execution scheduling models that cover common characteristics across use cases without being burdened by the specific quirks of any single use case.
- Specify the benchmark in a technology agnostic fashion to ensure the benchmark is portable.
- Specify the benchmark with special attention to how should the benchmark scale the functions, datasets, and execution scheduling.
- Build the benchmark execution harness with special attention to how the harness can scale without adding overhead.
- Ensure the benchmark behaves deterministically, or at least within statistical bounds, so that the benchmark can be rigorously audited.

The authors are involved in ongoing efforts to develop the TPC Decision Support (TPC-DS) benchmark for big data. These considerations present some of the hardest technical challenges, especially because the benchmark has to ensure the results have technical merit despite competing commericial interests from different test sponsors.

8.3 Parametric vs empirical models

A more theoretical consideration brought about by big data concerns what kind of models benchmarks should employ to generate the load and the data. The traditional approach is to use analytical models with a small number of parameters. For example, a common parametric model for arrival patterns is the Poisson or memoryless arrival model, used previously to generate network traffic [22]. A common parametric model for data patterns is the Zipf or long-tail frequency model, used for populating synthetic databases [17].

This approach works less well for big data, because the complex, diverse, non-stationary nature of the customer workloads make it hard to capture representative behavior using simple statistical processes with a small number of parameters. An alternative is to use empirical models, where the workload traces *are* the model. One can think of empirical models as models with an infinite number of parameters.

Recent work has started shifting towards empirical models, for example, showing that TELNET and FTP session arrivals approximate Poisson processes whose average arrival rates are empirical constants that change at the hourly or finer granularity [22]. A recent and successful MapReduce benchmark uses a fully empirical model, with the benchmark test workload being a statistical sample of the original historical workload trace [6].

The shift is an interesting one, because it illustrates that big data benchmarks sometimes need to compromise simplicity (favoring analytic models) to achieve representativeness (favoring empirical models). Furthermore, both kinds of models will fail to completely capture complex, non-stationary behavior [23].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2494595, IEEE Transactions on Services Computing

Once people reading benchmark results and people producing benchmark results get past the basic pitfalls discussed earlier, they would confront deeper technical challenges such as the choice of benchmark models, and whether that helps or hinders understanding system behavior in real-life.

9 Conclusion

Performance benchmarking is a challenging task. When done well, benchmarks can guide ourselves as well as the community. Cloudera is a leading vendor in big data, and we make special effort to ensure our performance studies are fair, rigorous, and thus useful to ourselves and our customers. The stories here show that even with good intent and best practices, performance benchmarking is fraught with challenges. Anyone can make benchmarking errors, everyone can learn from them, and everyone can benefit from reviewing their own work.

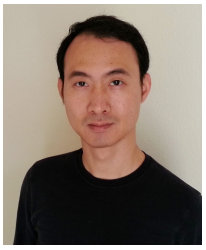
Acknowledgments

We would like to thank our colleagues at Cloudera who helped with various studies cited in this paper, especially Sandy Ryza, Karthik Kambatla, Jeff Bean, Justin Erickson, David Rorke, Dileep Kumar, and Arun Singla.

References

[1] Apache Software Foundation, "Apache Hadoop NextGen MapReduce (YARN)," http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.
[2] Apache Software Foundation, "Apache Sqoop," http://sqoop.apache.org.
[3] Apache Software Foundation, "With Fair Scheduler, cluster can logjam when all resources are consumed by AMs," Apache Jira YARN-1913, https://issues.apache.org/jira/browse/YARN-1913.
[4] Apache Software Foundation, "Apache JMeter," http://jmeter.apache.org/.
[5] J. Bean, "Apache Hadoop YARN: Avoiding 6 Time-Consuming 'Gotchas'," Cloudera Developer Blog, 2014, http://blog.cloudera.com/blog/2014/04/apache-hadoop-yarn-avoiding-6-time-consuming-gotchas/.
[6] Y. Chen, S. Alspaugh, A. Ganapathi, R. Griffith, and R. Katz, "Statistical workload injector for mapreduce," https://github.com/SWIMProjectUCB/SWIM/wiki.
[7] Y. Chen, S. Alspaugh, and R. Katz, "Interactive Query Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads," in *VLDB 2012*.
[8] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The Case for Evaluating MapReduce Performance Using Workload Suites," in *MASCOTS 2011*.
[9] Y. Chen, P. Gokhale, and A. Singla, "Configuring Impala and MapReduce for Multi-tenant Performance," Cloudera Developer Blog, 2013, http://blog.cloudera.com/blog/2013/06/configuring-impala-and-mapreduce-for-multi-tenant-performance/.
[10] Y. Chen, F. Raab, and R. Katz, "From tpc-c to big data benchmarks: A functional workload model," *Lecture Notes on Computer Science*, vol. 8163, 2014.
[11] Cloudera Inc., "Cloudera Impala," http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html.
[12] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI 2004*.
[13] J. Erickson, M. Kornacker, and D. Kumar, "New SQL Choices in the Apache Hadoop Ecosystem: Why Impala Continues to Lead," Cloudera Developer Blog, 2014, http://blog.cloudera.com/blog/2014/05/new-sql-choices-in-the-apache-hadoop-ecosystem-why-impala-continues-to-lead.
[14] J. Erickson, G. Rahn, M. Kornacker, and Y. Chen, "Impala Performance Update: Now Reaching DBMS-Class Speed," Cloudera Developer Blog, 2014, http://blog.cloudera.com/blog/2014/01/impala-performance-dbms-class-speed/.
[15] D. Ferrari, *Computer systems performance evaluation*. Prentice-Hall, 1978.
[16] J. Gray, "The Benchmark Handbook For Database and Transaction Processing Systems - Introduction," in *The Benchmark Handbook For Database and Transaction Processing Systems*, J. Gray, Ed. Morgan Kaufmann Publishers, 1993.
[17] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger, "Quickly generating billion-record synthetic databases," in *SIGMOD 1994*.
[18] Hewlett-Packard, "HP LoadRunner," http://www8.hp.com/us/en/software-solutions/loadrunner-load-testing/.
[19] K. Huppler, "The art of building a good benchmark," in *TPC Technical Conference 2009*.
[20] K. Kambatla and Y. Chen, "The Truth About MapReduce Performance on SSDs," in *LISA 2014*.
[21] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A Comparison of Approaches to Large-scale Data Analysis," in *SIGMOD 2009*.
[22] V. Paxson and S. Floyd, "Wide area traffic: the failure of poisson modeling," *Networking, IEEE/ACM Transactions on*, vol. 3, no. 3, Jun 1995.
[23] V. Paxson, "Empirically derived analytic models of wide-area tcp connections," *IEEE/ACM Trans. Netw.*, vol. 2, no. 4, Aug. 1994.
[24] S. Ryza, "Getting MapReduce 2 Up to Speed," Cloudera Developer Blog, 2014, http://blog.cloudera.com/blog/2014/02/getting-mapreduce-2-up-to-speed/.
[25] The Apache Software Foundation Blog, "The Apache Software Foundation Announces Apache Hadoop 2," https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces48.
[26] The Sort Benchmark, http://sortbenchmark.org/.
[27] The Standard Performance Evaluation Corporation, http://www.spec.org/.
[28] Transaction Processing Council, www.tpc.org.
[29] Transaction Processing Council Auditors, http://www.tpc.org/information/who/whoweare.asp#auditors.
[30] Transaction Processing Council, "TPC-C Benchmark," http://www.tpc.org/tpcc/.
[31] Transaction Processing Council, "TPC-H Benchmark," http://www.tpc.org/tpch/.
[32] Transaction Processing Council, "TPC-DS Benchmark," http://www.tpc.org/tpcds/.
[33] YCSB Community, "Yahoo! Cloud Serving Benchmark," https://github.com/brianfrankcooper/YCSB/.

Gwen Shapira is a Software Engineer on the Platform Engineering team at Cloudera, working on data ingest products. She has 15 years of experience working with customers to design scalable data architectures. She specializes in migrating data warehouses to Hadoop, integrating Hadoop with relational databases, building scalable data processing pipelines, and scaling complex data analysis algorithms.



Yanpei Chen is a software engineer on the Performance Engineering team at Cloudera. He works on multiple Hadoop ecosystem components including MapReduce, Impala, Solr, HBase, and Hive, because someone has to make sure the entire Hadoop ecosystem performs well together. His work involves internal engineering optimizations and external competitive benchmarking and customer support.