M

Microbenchmark

Nicolas Poggi
Databricks Inc., Amsterdam, NL, BarcelonaTech (UPC), Barcelona, Spain

Synonyms

Component benchmark; Functional benchmark; Test

Definition

A microbenchmark is either a program or routine to measure and test the performance of a single component or task. Microbenchmarks are used to measure simple and well-defined quantities such as elapsed time, rate of operations, bandwidth, or latency. Typically, microbenchmarks were associated with the testing of individual software subroutines or lower-level hardware components such as the CPU and for a short period of time. However, in the BigData scope, the term microbenchmarking is broadened to include the cluster – group of networked computers – acting as a single system, as well as the testing of frameworks, algorithms, logical and distributed components, for a longer period and larger data sizes.

Overview

Microbenchmarks constitute the first line of performance testing. Through them, we can ensure the proper and timely functioning of the different individual components that make up our system. The term *micro*, of course, depends on the problem size. In BigData we broaden the concept to cover the testing of large-scale distributed systems and computing frameworks. This chapter presents the historical background, evolution, central ideas, and current key applications of the field concerning BigData.

Historical Background

Origins and CPU-Oriented Benchmarks

Microbenchmarks are closer to both hardware and software testing than to competitive benchmarking, opposed to application-level – macro – benchmarking. For this reason, we can trace microbenchmarking influence to the hardware testing discipline as can be found in Sumne (1974). Furthermore, we can also find influence in the origins of software testing methodology during the 1970s, including works such as Chow (1978). One of the first examples of a microbenchmark clearly distinguishable from software testing is the *Whetstone benchmark* developed during the late 1960s and published later by Curnow and Wichmann (1976).

The *Whetstone benchmark* is considered the first general-purpose benchmark suite that helped

S. Sakr, A. Zomaya (eds.), *Encyclopedia of Big Data Technologies*,
https://doi.org/10.1007/978-3-319-63962-8_111-1

later to set industry standards in computer performance (Longbottom 2017). The first implementations of the benchmark were composed of a set of *ALGOL* and *FORTRAN* routines with the intention of comparing computer architectures and optimizing compilers (Longbottom 2017), in this case, by measuring the number floating point instructions per second of a system. *LINPACK* is another representative microbenchmark example from the late 1970s (Dongarra et al. 2003). *LINPACK* is still in use primarily in its parallel version – *HPL* – in high-performance computing (HPC) environments (A. Petitet 2004) and in the supercomputing Top500 list (TOP500.org 1993).

Another example of a benchmark with a long track of revisions and still in use today is the family of SPEC CPU benchmarks. *SPEC CPU* was standardized and is maintained by the Standard Performance Evaluation Corporation (SPEC 1989), and it is frequently used in official and academic publications (Nair and John 2008; Cantin and Hill 2001). Another early benchmark is UnixBench (1983), in this case, a suite or collection of different benchmarks including those from the Whetstone to graphics card microbenchmarks.

Disk and Network

Unlike HPC and CPU benchmarking, we can characterize BigData – at least the open source-based frameworks, as being GNU/Linux-based, 64-bit x86 architecture deployments over commodity hardware. As in BigData, the main bottlenecks were initially found in the storage and networking layers (Poggi et al. 2014). Due to being readily available, and already part of the system administration routines, the first level of benchmarking in a BigData system typically involves using popular Unix testing tools. In this category, we find the network testing tools such as ping (1983) and traceroute (1987) and disk management utilities including dd (1985) and hdparm (1994) available for most common Linux distributions. After network testing tools, we can find system benchmarks developed by the community, most notably in the disk/file system category: *Bonnie* (bonnie circa 1989), one of the first disk benchmarks with a later 64-bit for *Bonnie++* fork (bonnie++ 1999); *IOzone* another file system benchmark utility (iozone 2002); and Flexible I/O Tester (FIO), which was used in 2012 to set the world record in IOPS (benchmark 2012 IOPS world record) and continues to be popular among storage manufacturers. On networking, *netperf* by Hewlett-Packard (HP) (netperf 1995) has been a popular tool during the late 1990s and early 2000s. Another widely used tool involves the different versions of iperf (2003) still widely used in the field.

BigData

> In the beginning there was Sorting (Rabl and Baru 2014)

While preceding the BigData era, we can track the influence of the *sort* benchmark in the original MapReduce paper proposed by Google (Dean and Ghemawat 2004). The sort benchmark led by Jim Gray in an industry-academia collaboration was first introduced in 1985 to measure the input-output (I/O) of a computer system (Anon et al. 1985) as part of the *DebitCredit database* benchmark (Rabl et al. 2014). The sort benchmark was simple: generate and then sort one million, hundred-byte records stored on disk. Since its appearance, different versions – mainly increasing the input size – and the contests (Sortbenchmark.org 1987) organized around the benchmark have greatly influenced the data processing field. The introduction of the sort benchmark in the original MapReduce paper and it's simplicity but also its usefulness of stressing I/O components (Poggi et al. 2014) of a system, sort – in its 1-terabyte format – became the defacto standard in big data benchmarks. TeraSort was later standardized by the Transaction Processing Performance Council (TPC) as TPCx-HS in 2014 (Nambiar et al. 2015) as a way to provide a technically rigorous and comparable version of the algorithm. Since the first releases of Hadoop – the open source implementation of MapReduce in Examples (2016) – the Hadoop examples package has been included with the distribution. Hadoop examples originally included the *Grep* (also listed in Dean and Ghemawat

(2004)) and *WordCount* sample applications as microbenchmarks. The WordCount application has been widely used in educational contexts, becoming one of the first programs a user of Hadoop would implement. *WordCount* is CPU bound (Poggi et al. 2014), in that sense, not as useful as *TeraSort* to stress and benchmark a cluster. Since then, the Hadoop examples has grown to include an increasing number of microbenchmarks and test applications (Noll 2011), most notably the *dfsio* benchmark to measure the Hadoop Distributed File System (HDFS) throughput.

Other notably early Hadoop benchmark included in Hadoop v1 is *GridMix* by Douglas et al. (2007). GridMix simulates multiple users sharing a cluster and includes workloads of cache loads, compression, decompression, and job configuration regarding resource usage. The Hadoop microbenchmark suite designed by Islam et al. (2014) helps with detailed profiling and performance characterization of various HDFS operations. Likewise, the microbenchmark suite designed by Lu et al. (2014) provides detailed profiling and performance characterization of Hadoop RPC over different high-performance networks. Similarly, *HiBench* (Huang et al. 2010) has extended the *dfsio* program to compute the aggregated throughput by disabling the speculative execution of the MapReduce framework. *HiBench* also adds machine learning and web crawling benchmarks to the suite. Similarly, *BigDataBench* from the ICT, Chinese Academy of Sciences (2014) provides a collection of over 33 microbenchmarks in different categories.

As BigData evolves closer to data warehousing and business intelligence (BI), so do the benchmarks. Most likely, the *Wisconsin benchmark* was the first database-like microbenchmark, designed to measure the performance of individual SQL operators (DeWitt and Levine 2008). *MRBench* by Kim et al. (2008) is an early example which provided microbenchmarks in the form of MapReduce jobs derived from *TPC-H* (Transaction Processing Performance Council 2014). The CALDA benchmark proposed by Pavlo et al. (2009) was developed to compare Hadoop against parallel database systems. It included five SQL-based queries which could be implemented to the target system. Both the AMPLab and Hivebench benchmarks are based on CALDA's queries. The Yahoo! Cloud Serving Benchmark (YCSB) (Cooper et al. 2010) is a set of benchmarks for performance evaluations of key/value pair and cloud data-serving systems. YCSB was originally designed to compare Cassandra, HBase, and PNUTS in a cloud setting to a traditional database (MySQL). Over time, YCSB has been extended to support a variety of data systems (YCSB-Web 2017) and become the de facto benchmark for key/value stores. Also, YCSB is still frequently used in technical conferences such as in Poggi and Grier (2017).

Foundations

> Most benchmarks can be classified into two categories: micro- and macro-benchmarks. (Waller 2015)

Performance testing is a subset of performance engineering (Jain 1991b; Testing 2017), where microbenchmarking – or simply benchmarking – is one of the main tools employed to verify reliability and speed of a system. Microbenchmarks are the first line of testing tools for determining the correct functioning and configuration of a node, the interconnect (networking), and then the cluster – a group of networked computers acting as one system. A microbenchmark is then either a program or routine to measure and test the performance of a single component, task, or algorithm.

Microbenchmarks are used to measure simple and well-defined quantities such as elapsed time, the rate of operations – throughput, bandwidth, or latency. Typical use cases for microbenchmarks involve comparing algorithms (i.e., *sorting*) or the performance of the same software routine tested in different hardware platforms but using the exact same input as in Waller (2015). Furthermore, microbenchmarks often follow a *white box* approach as they are implemented having the specification of the target system or the application's code available.

While there are several classifications of benchmarks, according to Ehliar and Liu (2004), there are two practical approaches to benchmarking: micro and application-level (macro). Furthermore, Ehliar et al. distinguishes between micro and application-level according to the ease of implementation and the testing of one aspect at a time, as in microbenchmarks, to creating a realistic and useful application (macro). However, other authors such as Seltzer et al. (1999) argue that many micro- and macro-benchmarks do not efficiently model real workloads. Thus part of the dangers of relying only on benchmark results as the only basis for high-level decisions.

Microbenchmarks in BigData

Traditionally microbenchmarks were associated with the testing only of a system or lower-level hardware components such as CPU, main memory, and I/O on a single node. In the networking context, including a couple of nodes or networking devices, but in both cases for a short period of time. However, in the BigData scope, the term microbenchmarking is broadened to include the cluster of servers as the system under test and to include the testing of the frameworks or algorithms on which data applications are later built on.

Moreover, microbenchmarks in BigData include the testing of virtualized or logical components such as distributed memory and file systems provided by middleware and layers of abstractions at the software level opposed to the lower-level traditional system benchmarks. Within this context, the metrics of interest are either the total processing power, throughput, or bandwidth of the cluster as a whole, usually for a longer span of total test time which can last for several days (Poggi et al. 2014) and large input data sets to fully stress the total capacity of the system.

Properties of a Good Microbenchmark

On the topic of benchmarking, (Gray 1992) has identified four import criteria for a successful benchmark: relevance, portability, scalable, and simple. In particular, for microbenchmarks, simplicity is the main priority as it must perform a single, well-defined task, with reproducible results. In Gregg (2013), the author makes an emphasis of simplicity being the main advantage of microbenchmarks. In BigData and cloud computing, microbenchmarks also need to be scalable in both weak and strong scaling. Strong scaling refers to keeping the problem size – the data set – fixed but increasing the number of processing elements, typically the number of nodes or containers. In weak scaling, the data set is increased in proportion to the number of processing units. The weak scaling property of the microbenchmark is of particular importance to BigData, as it provides a way to increase the volume of data to be processed or scale factor as in TPC (2017) benchmarks.

In practice, when benchmarking weak scaling properties, the volume of data is also increased without adding more processing capacity, to measure in part the stability of the system and the rate in which performance decreases. Some examples of weak vs. strong scaling can be found in the ALOJA project results (ALOJA-Web 2015). The property of portability in the BigData field is usually covered as most widely used frameworks, i.e., Hadoop and Spark, are based on the Java virtual machine (JVM), which is multi-platform compatible. Furthermore, GNU/Linux, Hadoop, and Spark are open sourced, as well as most of the aforementioned benchmarks. Relevance is the most challenging feature of a microbenchmark. Testing only a single task or component, a microbenchmark often does not represent a real-world application or use case. A microbenchmark can also misrepresent a system or measure an incorrect property.

The Dangers of Microbenchmarks

Several authors have raised the importance and of good practices in benchmarking and about the dangers of using only microbenchmarks for decision-making. In Traeger et al. (2008), authors perform a comprehensive evaluation of over 106 storage benchmark research papers. Traeger et al. (2008) claim that most benchmarks available at the time were flawed and that statistical analysis of results was mostly incomplete. Gregg (2013)

adds to the dangers of benchmarking coining the term "benchmarking sins." The list of *sins* includes casual benchmarking, benchmark faith, testing the wrong thing, using complex tools, cheating, and many statistical and numerical errors while benchmarking. In Jain (1991a), the author expands the list of "common mistakes in benchmarking," including not taking into account the overheads of monitoring tools and not measuring transient performance. Traeger et al. (2008) recommend using a combination of several micro- and at least a macro-level benchmark should be run on the target system to understand better the performance.

Microbenchmarks, however, can excel in the repeatability, portability, speed of execution, and automation of the benchmarks as tests. For these reasons, microbenchmarks are often part of Continuous Integration (CI) development tools to detect performance regressions in new pieces of code.

Microbenchmarks in the Development Cycle

Microbenchmarks are often written as part of software development and testing cycle. When developing new functionality, the programmer may decide – most often not – to write a test that measures the performance of the new code. This process usually occurs alongside the coding of unit or functional code tests. As to performance tests – microbenchmarks – usually take longer to execute than unit tests; the tests might not be run with every iteration of the CI suite. For this reason, the performance tests are likely to be run manually and often is the case in practice that is not run frequently after the new feature is finished and shipped.

A relevant example of microbenchmarks in the development and testing process can be found in the Apache Spark's SQL project source code repository (Apache Spark 2015). The benchmark's folder contains a *base class* to encapsulate adding and running new benchmarks a set of microbenchmarks. The test classes embed the data generation step when needed and output the CPU architecture and time details, as well as the rate of operations achieved. These tests are meant to be run in the developer workstation or CI environment, as they are often single node or even single-threaded – opposed to a full Spark cluster setting. Moreover, most of these tests are marked as *ignored* in the code, which prevents the CI and testing such as *maven* or *sbt* tools to run them by default.

Key Applications

It was mentioned in the Historical Background Section that data processing benchmarks started with the *DebitCredit* database benchmark and specifically the included *sort* test (Rabl and Baru 2014). We can trace the influence of *sort* to the original MapReduce publication (Dean and Ghemawat 2004) and its *TeraSort* variant included in the Hadoop examples and the TPCx-HS standard (Nambiar et al. 2015). *TeraSort* is still frequently used to benchmark new cluster installations.

This section focuses on a selection of microbenchmarks that are still highly relevant to the industry and are part of the BigData Benchmarking Compendium by Ivanov et al. (2015) and the works of Rabl and Baru (2014) and Poggi et al. (2014). Here benchmarks are classified into system, functional, and database categories. Furthermore, several other early key microbenchmarks where mentioned such as Hadoop *GridMix* and *MRBench*. Other benchmarks that have their own section in this publication, such as HiBench, SparkBench, and YCSB, are briefly discussed.

System

Disk and Filesystem Tools

For data processing, in particular, the storage – disk and file systems – is one of the most important resources to carefully design and tune to achieve high performance. Here the focus is on simple GNU/Linux tools that can be used to test that the target system can achieve a stable and desired operation. For input/output (I/O), the metrics of interest are bandwidth measured in megabytes-per-second MB/s, latency in microseconds, and the number of I/O operations per

second or IOPS. IOPS and latency are metrics that are gaining importance (Poggi and Grier 2017) as BigData evolves from batch to more interactive use cases, i.e., streaming, also, to hardware advances such as flash-based storage technologies, e.g., SSDs and NVMe. The task of disk and network testing is usually assigned to system administrators while the functional and database to application roles. This situation leads to different teams not sharing the results or repeating the tests in the future, a situation which can lead to decreased performance over time.

dd (dd 1985) is a disk utility found in most Unix-like operating systems. Its primary purpose is to create, convert, and copy files. However, it can also be used for benchmarking raw disk devices and file systems, as the tool has multiple options and flags both for reading and writing files. Most notably, it contains options to access the raw devices directly and to bypass OS caches. In both cases, it performs only sequential operations – opposed to random R/W. The sample output below is an example of a mounted networked volume from a popular cloud provider (Store 2017):

```
# dd sample output for writing 10GB of data from EBS
10000+0 records in
10000+0 records out
10485760000 bytes (10 GB) copied, 83.3s, 126MB/s

#dd sample output for reading 10GB of data from EBS
10000+0 records in
10000+0 records out
10485760000 bytes (10 GB) copied, 82.8s, 126MB/s
```

It is interesting to note that in this case, the drive wrote and read at the same rate of 126 MB/s. In this case, the volume is a networked-attached SSD. Magnetic drives (HDD) usually present a lower write than reading rate.

Flexible I/O Tester (FIO) (benchmark 2012 IOPS world record): FIO is the most widely used tool by drive manufacturers to measure device specifications. It was originally written by Jens Axboe to automate the testing of Linux schedulers and I/O subsystem. Since then it has been ported to several operating systems outside GNU/Linux including BSD, Solaris, and Windows. FIO allows testing file systems or raw devices directly. Also to do sequential and random read and write tests. Multiple-file reading and writing are also possible, as well as selecting the number of jobs, I/O queue depth, I/O library, and request (payload) size.

In addition to *dd* and *FIO*, which are single-node tools, cluster-oriented tools are needed to measure cluster-aggregated bandwidth for reading and writing files. For example, for Hadoop's HDFS, the *dfsio* microbenchmark is often used.

Networking

For networking, the most widely used tool is *iperf* (iperf 2003). *Iperf* can be started in client or server mode to measure the bandwidth and latency between two nodes. It has the limitation of only accepting connections from one client. To benchmark all of the nodes at the same time – and stressing the switching infrastructure – several servers and clients need to be started pointing to different ports. Using this technique, it allows to test the whole cluster, but the aggregated metrics need to be calculated separately from the individual *iperf* reports.

Functional

Hadoop Examples and HiBench

On the BigData functional level, the most widely used benchmarks include the examples of the Hadoop examples package included in most Hadoop distributions. The examples have evolved from including only *grep* and *WordCount* to over 15 sample jobs to test a specific feature of the cluster or to aid in Hadoop's framework development. Running Hadoop examples is discussed in depth in Noll (2011) blog. Also, benchmarking platforms such as ALOJA by Poggi et al. (2014) enables to easily run a selection of tests in a target cluster, as well as providing results of benchmarks ran in several architectures. In Hadoop versions 2.7 and higher, the example applications have been split into *MapReduce* and *common* packages. The continuously growing list of supported tests can be found at the examples repository (2018). Below is a list of the most representative benchmarks included and some of the extensions found in the first version of *HiBench* (Huang et al. 2010). *HiBench* started as a convenient package and extension to the Hadoop examples

but has evolved into a multi-framework and custom benchmarking suite.

- **TeraSort** sorts 1TB of data generated by the TeraGen program distributed with Hadoop. TeraSort is widely used as a reference in research papers as well as in BigData competitions. TeraSort is I/O and CPU intensive.
- **Wordcount** counts the number of word occurrences in large text files. It is distributed with Hadoop and used in many MapReduce learning books. It is CPU bound.
- **Sort** uses the MapReduce framework to sort the input directory into the output directory, being predominately I/O and shuffle intensive.
- **DFSIO** (or Enhanced DFSIO in HiBench) is an I/O-intensive benchmark to measure throughput in HDFS using MapReduce. It features separate read and write benchmarks.
- **Pagerank**, an implementation of Google's web PageRanking algorithm. It crawls Wikipedia sample pages.

Additional benchmarks included in HiBench:

- **Bayes** Bayesian Machine Learning classification using the Mahout library. The input of this benchmark is extracted from a subset of the Wikipedia dump.
- **K-means** Mahout's implementation of the k-means algorithm for knowledge discovery and data mining.
- **HiveBench** the OLAP-style Join and Aggregation queries are adapted from the CALDA benchmark (Pavlo et al. 2009). HiveBench has the goal to test the Hive SQL performance.

SparkBench

SparkBench (Li et al. 2015), developed by IBM, is a comprehensive Spark-specific benchmarking suite. The purpose of SparkBench is to analyze tradeoffs between different configurations in Spark and to guide system tuning. It is composed of over ten workloads in four categories including machine learning, graph processing, streaming, and SQL queries. SparkBench reports two metrics: *job execution time* (seconds) and *data process rate* (MB/second).

Database

Besides hardware and software testing, BigData benchmarking is heavily influenced by data processing and databases in general. Moreover, currently it is sometimes difficult to differentiate between a BigData system such as Spark with its SQL library and a massively parallel processing (MPP) database. Over time, what previously were considered functional (Chen et al. 2012) or even application-level benchmarks such as database benchmarks such as the *DebitCredit* (Anon et al. 1985) and the *Winsconsin* benchmark (DeWitt and Levine 2008; Chen et al. 2012); by extracting specific queries, they can also be considered microbenchmarks. An early example of this fusion is the *CALDA* benchmark, a simple benchmark used initially to compare Hadoop to a distributed database. Over time, CALDA (and its variants) became the de facto benchmark for the early SQL-on-Hadoop (Poggi et al. 2016) systems such as Hive and Spark, with the later AMPLab (AMPLab 2013) adaptations of CALDA.

CALDA – Pavlo's Benchmark and Adaptations

Pavlo's benchmark (Pavlo et al. 2009; Stonebraker et al. 2010) consists of five tasks defined as SQL query testing scans, joins, aggregations, and a user-defined function (UDF). The benchmark was developed to specifically compare the capabilities of Hadoop with those of commercial parallel relational database management systems (RDBMS) but was used as a base to also compare newer emerging frameworks such as Hive and Spark.

AMP Lab BigData Benchmark extended CALDA to measure the analytical capabilities of BigData warehousing solutions in general. The hosted benchmark results currently provide quantitative and qualitative comparisons of five data warehouse systems: RedShift, Hive, Stinger/Tez, Shark, and Impala. It supports different data sizes to scale also in cluster size.

YCSB

The Yahoo! Cloud Serving Benchmark (YCSB) (Cooper et al. 2010; Patil et al. 2011) is a set of benchmarks for performance evaluations of key/value pair and cloud data-serving systems.

YCSB was originally designed to compare Cassanda, HBase, and PNUTS in a cloud setting to a traditional database (MySQL). Over time, YCSB has been extended to support a variety of data systems (YCSB-Web 2017) also including MongoDB and Riak and has become the de facto benchmark for key/value stores. Also, YCSB is still frequently used in technical conferences such as in Poggi and Grier (2017).

Concluding Remarks

As final words, there is a fine distinction between micro- and macro-level benchmarks. What initially was presented as a complex, application-like benchmark, over time, as its behavior is understood and technology advances, microbenchmarks are usually extracted from the application, as is the case with BigData microbenchmarks, especially in the database-like properties of the systems. Microbenchmarks today are still the first line of performance testing during development and setting up a production system. Furthermore, the benchmarking field is constantly evolving, and microbenchmarking inheriting functionality from macro-benchmarks and expanding with technology.

Cross-References

- Component Benchmark
- SparkBench
- TPC-xHS
- YCSB

References

ALOJA-Web (2015) Aloja home page. https://aloja.bsc.es/
AMPLab (2013) https://amplab.cs.berkeley.edu/benchmark/
Anon EA, Bitton D, Brown M, Catell R, Ceri S, Chou T, DeWitt D, Gawlick D, Garcia-Molina H, Good B, Gray J, Homan P, Jolls B, Lukes T, Lazowska E, Nauman J, Pong M, Spector A, Trieber K, Sammer H, Serlin O, Stonebraker M, Reuter A, Weinberger P (1985) A measure of transaction processing power. Datamation 31(7):112–118. http://dl.acm.org/citation.cfm?id=13900.18159
Apache Spark (2015) Apache spark repository – benchmarks. https://github.com/apache/spark/tree/master/sql/core/src/test/scala/org/apache/spark/sql/execution/benchmark
benchmark (2012 IOPS world record) F (2005) https://en.wikipedia.org/wiki/Jens_Axboe
bonnie++ (1999) https://en.wikipedia.org/wiki/Bonnie++
bonnie (circa 1989) https://www.tbray.org/ongoing/When/200x/2004/11/16/Bonnie64
Cantin JF, Hill MD (2001) Cache performance for selected spec cpu2000 benchmarks. ACM SIGARCH Comput Archit News 29:13–18
Chen Y, Raab F, Katz R (2012) From TPC-C to big data benchmarks: a functional workload model. In: WBDB
Chow TS (1978) Testing software design modeled by finite-state machines. IEEE Trans Softw Eng SE-4(3):178–187. https://doi.org/10.1109/TSE.1978.231496
Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010) Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM symposium on cloud computing, SoCC 2010, Indianapolis, 10–11 June 2010, pp 143–154
Curnow HJ, Wichmann BA (1976) A synthetic benchmark. OLW Comput J 19(1):43–49
dd (1985) https://en.wikipedia.org/wiki/Dd_(Unix)
Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Proceedings of the 6th conference on symposium on operating systems design and implementation (OSDI'04). USENIX Association
DeWitt DJ, Levine C (2008) Not just correct, but correct and fast: a look at one of Jim Gray's contributions to database system performance. SIGMOD Rec 37(2):45–49. https://doi.org/10.1145/1379387.1379403
Dongarra JJ, Luszczek P, Petitet A (2003) The linpack benchmark: past, present, and future. Concurr Comput Pract Exp 15:2003
Douglas C et al (2007) Grid-mix. https://hadoop.apache.org/docs/r1.2.1/gridmix.html
Ehliar A, Liu D (2004) Benchmarking network processors. In: Swedish system-on-chip conference
Examples H (2016) https://apache.googlesource.com/hadoop-common/+/release-0.1.0/src/examples/org/apache/hadoop/examples
examples repository H (2018) https://github.com/apache/hadoop/tree/trunk/hadoop-mapreduce-project/hadoop-mapreduce-examples/src/main/java/org/apache/hadoop/examples
Gray J (1992) Benchmark handbook: for database and transaction processing systems. Morgan Kaufmann Publishers Inc., San Francisco
Gregg B (2013) Systems performance: enterprise and the cloud. Always learning, Prentice Hall. https://books.google.de/books?id=xQdvAQAAQBAJ
hdparm (1994) https://sourceforge.net/projects/hdparm/

Huang S, Huang J, Dai J, Xie T, Huang B (2010) The HiBench benchmark suite: characterization of the MapReduce-based data analysis. In: 22nd international conference on data engineering workshops, pp 41–51. https://doi.org/10.1109/icdew.2010.5452747

ICT, Chinese Academy of Sciences (2014) BigDataBench 3.1. http://prof.ict.ac.cn/BigDataBench/

iozone (2002) https://en.wikipedia.org/wiki/IOzone

iperf (2003) https://en.wikipedia.org/wiki/Iperf

Islam NS, Lu X, Wasi-ur Rahman M, Jose J, Panda DKD (2014) A Micro-benchmark suite for evaluating HDFS operations on modern clusters. Springer, Berlin/Heidelberg, pp 129–147. https://doi.org/10.1007/978-3-642-53974-9_12

Ivanov T, Rabl T, Poess M, Queralt A, Poelman J, Poggi N, Buell J (2015) Big data benchmark compendium. In: TPCTC, pp 135–155

Jain R (1991a) The art of computer systems performance analysis – techniques for experimental design, measurement, simulation, and modeling. Wiley professional computing. Wiley-Interscience, New York

Jain R (1991b) The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling. Wiley-Interscience, New York

Kim K, Jeon K, Han H, Kim SG, Jung H, Yeom HY (2008) Mrbench: a benchmark for mapreduce framework. In: 14th international conference on parallel and distributed systems, ICPADS 2008, Melbourne, 8–10 Dec 2008, pp 11–18

Li M, Tan J, Wang Y, Zhang L, Salapura V (2015) Sparkbench: a comprehensive benchmarking suite for in memory data analytic platform spark. In: Proceedings of the 12th ACM international conference on computing frontiers, CF'15. ACM, New York, pp 53:1–53:8

Longbottom R (2017) Whetstone benchmark history and results. https://www.roylongbottom.org.uk/whetstone.htm

Lu X, Wasi-ur Rahman M, Islam NS, Panda DKD (2014) A micro-benchmark suite for evaluating Hadoop RPC on high-performance networks. Springer International Publishing, Cham, pp 32–42. https://doi.org/10.1007/978-3-319-10596-3_3

Nair AA, John LK (2008) Simulation points for SPEC CPU 2006. In: IEEE international conference on computer design, pp 397-403. ISSN:1063-6404

Nambiar R, Poess M, Dey A, Cao P, Magdon-Ismail T, Qi Ren D, Bond A (2015) Introducing TPCx-HS: the first industry standard for benchmarking big data systems. Springer International Publishing, Cham, pp 1–12. https://doi.org/10.1007/978-3-319-15350-6_1

netperf (1995) https://www.cup.hp.com/netperf/NetperfPage.html

Noll M (2011) Benchmarking and stress-testing a hadoop cluster. http://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfsio-nnbench-mrbench/

Patil S, Polte M, Ren K, Tantisiriroj W, Xiao L, López J, Gibson G, Fuchs A, Rinaldi B (2011) YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In: ACM symposium on cloud computing in conjunction with SOSP 2011, SOCC'11, Cascais, 26–28 Oct 2011, p 9

Pavlo A, Paulson E, Rasin A, Abadi DJ, DeWitt DJ, Madden S, Stonebraker M (2009) A comparison of approaches to large-scale data analysis. In: SIGMOD, pp 165–178

Petitet JDAC A, Whaley RC (2004) Hpl – a portable implementation of the high-performance linpack benchmark for distributed-memory computers. ICL – UTK Computer Science Department Retrieved 22 Sept 2016

ping (1983) https://en.wikipedia.org/wiki/Ping_(networking_utility)

Poggi N, Grier D (2017) Evaluating NVMe drives for accelerating HBase. Dataworks Summit (Hadoop Summit) San Jose

Poggi N, Carrera D, Vujic N, Blakeley J et al (2014) ALOJA: a systematic study of hadoop deployment variables to enable automated characterization of cost-effectiveness. In: IEEE international conference on big data, big data 2014, Washington, DC, 27–30 Oct

Poggi N, Berral JL, Fenech T, Carrera D, Blakeley J, Minhas UF, Vujic N (2016) The state of sql-on-hadoop in the cloud. In: 2016 IEEE international conference on big data (big data), pp 1432–1443. https://doi.org/10.1109/BigData.2016.7840751

Rabl T, Poess M, Baru CK, Jacobsen H-A (2014) Specifying big data benchmarks – First workshop, WBDB 2012, San Jose, 8–9 May 2012, and Second workshop, WBDB 2012, Pune, India, 17–18 Dec 2012, Revised selected papers. Lecture notes in computer science, vol 8163. Springer (2014). https://doi.org/10.1007/978-3-642-53974-9

Rabl T, Baru C (2014) Big data benchmarking tutorial. In: IEEE international conference on big data, big data 2014, Washington, DC, 27–30 Oct

Seltzer M, Krinsky D, Smith K, Zhang X (1999) The case for application-specific benchmarking. In: Proceedings of the seventh workshop on hot topics in operating systems, HOTOS'99. IEEE Computer Society, Washington, DC, pp 102

Sortbenchmarkorg (1987) http://sortbenchmark.org/

SPEC (1989) https://www.spec.org/

Stonebraker M, Abadi DJ, DeWitt DJ, Madden S, Paulson E, Pavlo A, Rasin A (2010) MapReduce and parallel DBMSs: friends or foes? Commun ACM 53(1):64–71

Store AEB (2017) https://aws.amazon.com/ebs/

Sumne FH (1974) Measurement techniques in computer hardware design. State of the Art Report No 18, pp 367–390

Testing SP (2017) https://en.wikipedia.org/wiki/Software_performance_testing

TOP500org (1993) https://www.top500.org/project/top500_description/

TPC (2017) https://www.tpc.org/

traceroute (1987) https://en.wikipedia.org/wiki/Traceroute

Traeger A, Zadok E, Joukov N, Wright CP (2008) A nine year study of file system and storage benchmarking. Trans Storage 4(2):5:1–5:56. https://doi.org/10.1145/1367829.1367831

Transaction Processing Performance Council (2014) TPC benchmark H – standard specification. Version 2.17.1
UnixBench (1983) https://github.com/kdlucas/byte-unixbench
Waller J (2015) Performance benchmarking of application monitoring frameworks. BoD – books on demand
YCSB-Web (2017) Ycsb code repository. https://github.com/brianfrankcooper/YCSB