From TPC-C to Big Data Benchmarks: A Functional Workload Model

Yanpei Chen
Francois Raab
Randy H. Katz

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-174
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-174.html

July 1, 2012

From TPC-C to Big Data Benchmarks: A Functional Workload Model

Yanpei Chen[1], Francois Raab[2], and Randy Katz[3]

[1] Cloudera & UC Berkeley, yanpei@cloudera.com,
[2] InfoSizing, francois@sizing.com,
[3] UC Berkeley, randy@eecs.berkeley.edu,

Abstract. Big data systems help organizations store, manipulate, and derive value from vast amounts of data. Relational database and MapReduce are two, arguably competing implementations of such systems. They are characterized by very large data volumes, diverse unconventional data types and complex data analysis functions. These properties make it challenging to develop big data benchmarks that reflect real life use cases and cover multiple types of implementation options. In this position paper, we combine experiences from the TPC-C benchmark with emerging insights from MapReduce application domains to argue for using a model based on functions of abstraction to construct future benchmarks for big data systems. In particular, this model describes several components of the targeted workloads: the functional goals that the system must achieve, the representative data access patterns, the scheduling and load variations over time, and the computation required to achieve the functional goals. We show that the TPC-C benchmark already applies such a model to benchmarking transactional systems. A similar model can be developed for other big data systems, such as MapReduce, once additional empirical studies are performed. Identifying the functions of abstraction for a big data application domain represents the first step towards building truly representative big data benchmarks.

1 Introduction

Big data is one of the fastest growing fields in data processing today. It cuts across numerous industrial and public service areas, from web commerce to traditional retail, from genomics to geospatial, from social networks to interactive media, and from power grid management to traffic control. New solutions appear at a rapid pace, and the need emerges for an objective method to compare their applicability, efficiency, and cost. In other words, there is a growing need for a set of big data performance benchmarks.

There have been a number of attempts at constructing big data benchmarks [18,20,21,28]. None of them has gained wide recognition and broad usage. It remains an elusive goal to evaluate a wide range of proposed big data solutions. The field of big data performance is in a state where every study and claim employs a different methodology. Results from one publication to the next are

not comparable and often not even closely related, as it was the case for OLTP some twenty years ago and for decision support shortly thereafter.

We recognize the need for a yard-stick to measure the performance of big data systems. The exploding market research and trade publications coverage of the subject indicates that big data is fast spreading within the corporate IT infrastructures, even for non-technology, traditional industry sectors. This new stage in the development of big data present an opportunity to recognize significant application domains as they emerge, in order to define relevant and objective benchmarks.

We argue that the field as a whole has not gained sufficient experience to definitively say "the big data benchmark should be X." Big data remains a new and rapidly changing field. The inherent complexity, diversity, and scale of such systems introduce additional challenges for defining a representative, portable, and scalable benchmark. However, by combining past experience from TPC-style benchmarks and emerging insights from MapReduce deployments, we can at least clarify some key concerns and concepts associated with building standard big data benchmarks.

In this position paper, we draw on the lessons learned from more than two decades of database systems performance measurements, and propose a model that can be extended to big data systems. We discuss properties of good benchmarks and identify the characteristics of big data systems that make it challenging to develop such benchmarks (Section 2). We then present an insider's retrospective on the events that led to the development of TPC-C, the standard yardstick for OLTP performance , and discuss the conceptual models that led to a fully synthetic, yet representative benchmark (Section 3). We formalize two key concepts, *functions of abstraction* and *functional workload models*, and discuss how they enable the construction of representative benchmarks that can translate across different types of systems (Section 4). We then examine some recent cluster traces for MapReduce deployments, and highlight that the process of identifying MapReduce functions of abstraction and functional workload models remains a work in progress (Section 5). Combining the TPC-C and MapReduce insights, we present a vision for a big data benchmark that contains a mix of application domains (i.e., application set), each containing its own functional workload models and functions of abstraction (Section 6).

2 What's New for Benchmarking Big Data

Prior work on computer performance methodology identified the following properties of a good benchmark [22, 25]:

Representative

The benchmark should measure performance using metrics that are relevant to real life application domains, and under conditions that reflect real life computational needs.

Portable

The benchmark should be able to be ported to alternative kinds of systems that can service the computational needs of the same application domain.

Scalable

The benchmark should be able to measure performance for both large and small systems, and for both large and small problem sizes. What are "large" and "small" systems and what are "large" and "small" problems are defined by the specific application domain to be benchmarked.

Simple

The benchmark should be conceptually understandable. What is "simple" depends on the application domains to be benchmarked. The benchmark should try to abstract away details that do not affect performance, and represent case-by-case system configuration or administration choices.

Given these criteria of a good benchmark, there are at least four properties of big data systems that make it challenging to develop big data benchmarks.

System diversity

Big data systems store and manipulate many kinds of data. Such diversity translates to significant, and sometimes mutually exclusive, variations in system design. A design optimized for one application domain may be suboptimal, or even harmful for another. Conversely, it is challenging for a big data benchmark to be representative and portable. The benchmark needs to replicate realistic conditions across a range of application domains, and use metrics that translates across potentially divergent computational needs.

Rapid data evolution

Big data computational needs constantly and rapidly evolve. Such rapid changes reflects the innovations in business, science, and consumer behavior facilitated by knowledge extracted from big data. Consequently, the change in the underlying data likely outpaces the ability to develop schema about the data, or gain design intuition about the systems that manipulate the data. It becomes challenging to ensure that the benchmark tracks such changes both in the data sources and the resulting systems.

System and data scale

Big data systems often involve multi-layered and distributed components, while big data itself often involves multiple sources of different formats. This translates to multiple ways for the system to scale at multiple layers, and the data to scale across multiple sources and formats. Consequently, it becomes challenging for a big data benchmark to correctly scale the benchmarked problem, and correctly measure how the system performs in solving these problems.

System complexity

The multi-layered and distributed nature of big data systems also make it challenging for a big data benchmark to be simple. Any simplifications of big data systems is likely to remain complex in the absolute sense. Further, any simplifications need objective, empirical measurements to verify that system performance is not affected.

Of the four benchmark properties, we argue that the most important are "representative" and "portable". The rest of the paper introduces some benchmark principles that helps achieve these two properties. "Scalability" remains an important challenge to be addressed by future work. "Simple" is likely to be hard to achieve in an absolute sense — for complex systems, "simple" is inherently in conflict with "representative".

3 The Process of Building TPC-C

TPC-C is a good example of a benchmark that has had a substantial impact on technologies and systems. Understanding the origin of this long standing industry yardstick provides important clues toward the definition of a big data benchmark. In this section, we retrace the events that lead to the creation of TPC-C and present the conceptual motivation behind its design.

3.1 The origin of TPC-C

The emergence and rapid growth of On Line Transaction Processing (OLTP) in the early eighties highlights the importance of benchmarking a specific application domain. The field of transaction processing was heating up and the need to satisfy on-line transaction requirements for fast user response times was growing rapidly. CODASYL databases supporting transactional properties were the dominant technology, a status increasingly challenged by relational databases. For instance, version 3 of the Oracle relational database, released in 1983, implemented support for the COMMIT and ROLLBACK functionalities. As competition intensified, the need emerged for an objective measure of performance. In 1985, Jim Gray led an industry-academia group of over twenty members to create a new OLTP benchmark under the name DebitCredit [13].

In the late eighties, relational databases had matured and were fast replacing the CODASYL model. The DebitCredit benchmark, and its derivatives ET1 and TP1, had become de-facto standards. Database system vendors used them to make performance claims, often raising controversies [32]. A single standard was still absent, which led to confusion about the comparability of results. In June of 1988, T. Sawyer and O. Serlin proposed to standardize DebitCredit. Later that year, O. Serlin spearheaded the creation of the Transaction Processing Performance Council (TPC) tasked with creating an industry standard version of DebitCredit [33].

Around this time, Digital Equipment Corporation (DEC) was in the process of developing a new relational database product, code name RdbStar. The development team soon recognized that a performance benchmark would be needed to assess the capabilities of early versions of the new product. DEC's European subsidiary had been conducting a vast survey of database applications across France, England, Italy, Germany, Holland, Denmark and Finland. Production systems at key customer sites had been examined and local support staff interviewed. The survey sought to better understand how databases were used in

the field and which features were most commonly found in production systems. Armed with this data, the RdbStar benchmark development project started with an examination of the many database benchmarks known at the time, including the Wisconsin benchmark [15], AS3AP [34] and the Set Query Benchmark [27].

The approach found to be the most representative of the European survey's findings came from an unpublished benchmark, one developed by the Microelectronics and Computer Consortium (MCC), one of the largest computer industry research and development consortia, based in Austin, TX. Researchers at MCC were working on distributed database technology [14] and had developed a simulator to test various designs. Part of the simulator involved executing OLTP functions inspired by an order processing application. The MCC benchmark was selected by DEC as the starting point for the RdbStar benchmark. Parts of the MCC benchmark were adjusted, as will be discussed later in this paper, and the resulting benchmark became known internally as Order-Entry.

In November of 1989, the TPC published its standardized end-to-end version of DebitCredit under the name TPC Benchmark A (TPC-A) [11]. TPC Benchmark B (TPC-B) [12] followed in August 1990, which represented a back-end version of TPC-A. By then, the simple transaction in DebitCredit was starting to come under fire as being too simplistic and not sufficiently exercising the features of mature database products. The TPC issued a request for proposal of a more complex OLTP benchmark. IBM submitted its RAMP-C benchmark and DEC submitted Order-Entry. The TPC selected the DEC benchmark and assigned its author, F. Raab, to lead the creation of the new standard. July 1992 saw the approval and release of the new TPC Benchmark C (TPC-C) [30].

3.2 The abstract makeup of TPC-C

The original Order-Entry benchmark from DEC included two components: a set of database transactions targeting the OLTP application domain, and a set of simple and complex queries targeting the decision support application domain. The TPC adopted the OLTP portion of Order-Entry for the creation of TPC-C. This portion included a controlled mix of five transactions executing against a database of nine tables.

The design of the transactional portion of Order-Entry did not follow the traditional model used for building business applications. The design of an application can be decomposed into four basic elements, as follows:

- Tables: The database tables, the layout of the rows and the correlation between tables.
- Population: The data that populates the tables, the distribution of values and the correlation between the values in different columns of the tables.
- Tansactions: The units of computation against the data in the tables, the distribution of input variables and the interactions between transactions.
- Scheduling: The pacing and mix of transactions.

In the traditional design model, each of these elements implements part of the business functions targeted by the application. The tables would represent the

business context. The population would start with a base set capturing the initial state of the business and evolve as a result of conducting daily business. The transactions would implement the business functions. The scheduling would reflect business activity.

This model results in benchmarks aligned with the business details of the targeted application rather than by more general benchmarking objectives. Most critically, the benchmark elements are specific to a single application and not representative of the whole domain. In contrast, standard benchmarking is a synthetic activity that seeks to be representative of a collection of applications within a domain, and its sole purpose is to gather relevant performance information. Being free of any real business context, the elements of such a benchmark should be abstracted from a representative cross section of the applications within the targeted domain.

To illustrate the concept of using abstractions to design the elements of a benchmark, we take a closer look at how this applies to transactions. The objective is to look at the compute units of multiple applications and to find repetitions or similarities. For instance, in the OLTP application domain, it is common to find user-initiated operations that involve multiple successive database transactions. While these transactions are related through the application's business semantics, they are otherwise independent from the point of view of exercising and measuring the system. Consequently, they should be examined independently during the process of creating a set of abstract database transactions. Consider the following:

```
User-initiated operation
    Database Transaction T1
        Read row from table A
        Update row in table B
        Commit transaction
    Database Transaction T2
        Update row in table A
        Insert row in table C
        Commit transaction
    Database Transaction T3
        Read row from table C
        Update row in table B
        Commit transaction
```

In the above, T1 and T3 are performing similar operations, but on different tables. However, if tables A and C have the same characteristics, T1 and T3 can be viewed as duplicates of the same abstract transaction, one that contains a "read row" followed by an "update row".

Order-Entry abstracted the multitude of real-life transactions from the OLTP application domain to only five abstract transactions. Such a compression resulted in a substantial loss of specificity. However, we argue that the loss is more than outweighed by the gain in the ability to gather relevant performance information across a large portion of the OLTP application domain. The success of the benchmark over the last two decades appears to support this view.

4 Functions of Abstraction and Functional Workload Model

The process of constructing TPC-C highlights two key concepts — *functions of abstraction* and the *functional workload model*. In this section, we explain what they are, and why they are vital to constructing big data benchmarks that target specific application domains while accommodating diverse system implementations.

4.1 Functions of abstraction

Functions of abstraction are units of computation that appear frequently in the application domain being benchmarked. They are expressed in a generic form that is independent of the underlying system implementation. They are units of computation against a data set. Their definition includes the distribution of the data they target and the rules governing the interactions they have with each other. Such a specification is *abstract*, in that it only defines the *functional* goal of the unit of computation, independently of the system level components that might be used to service it. As a result, they can operate against variable and scalable data volumes and can be combined in workloads of desired complexity. Further, one can extract various performance metrics from their combined execution to meet specific benchmarking goals.

TPC-C (i.e., Order-Entry) is articulated around the following five functions of abstraction: a midweight read-write transaction (i.e., New-Order), a lightweight read-write transaction (i.e., Payment), a midweight read-only transaction (i.e., Order-Status), a batch of midweight read-write transactions (i.e., Delivery), and a heavyweight read-only transaction (i.e., Stock-Level) [31]. They are specified in the semantic context, or story-line, of an order processing environment. That context, however, is entirely artificial. Its sole purpose is to allow easy description of the components.

For each of the above functions of abstraction, the functional goal is defined in terms of a set of data manipulation operations. The targeted data is defined through the distribution of values used as input variables. Their interactions is governed by the transactional properties of atomicity, consistentency, isolation, and durability (i.e., the ACID properties). Defined as such, the TPC-C transactions achieve the goals of functions of abstraction as follows:

- The underlying system could be a relational database, a traditional file system, a CODASYL database, or an extension of the Apache Hadoop implementation of MapReduce that provides transactional capabilities.
- The volume and distribution of the data set on which the transactions operate is specified separately.
- Benchmarks of various complexities can be created by using various combinations and mixes of the defined functions of abstraction. While TPC-C involves the combination of all five transactions, the Payment transaction can be run by itself to implement the TPC-A (i.e., Debit-Credit) benchmark.

– Multiple performance metrics can be extracted by measuring various aspects of the resulting execution. The TPC-C standard tpm-C metric (number of New-Order transactions executed per minute) is one of them. Many other metrics are possible, e.g., the 99th percentile latency (response time) of the Payment transactions.

Once defined, the functions of abstraction can be combined with a specified scheduling and with the definition of table structures and populations to form a functional workload model, which we explain next.

4.2 Functional Workload Model

The *functional workload model* seeks to capture the representative load that a system needs to service in the context of a selected application domain. It describes in a system-independent manner the compute patterns (i.e., functions of abstraction) as well as their scheduling and the data set they act upon. Describing the scheduling involves specifying one or more load patterns to be applied to the system. This definition is in terms of execution frequency, distribution and arrival rate of each individual function of abstraction. Describing the data set acted upon involves specifying its structure, inter-dependence, initial size and contents, and how it evolves over the course of the workload's execution. These descriptions are of a synthetic nature in that they are limited to the essential characteristics of the functional goals of the selected application domain.

The simplicity and lack of duplication at the core of the definition of functions of abstraction are just as critical for the design of the other components (i.e., scheduling and data set) of the functional workload model. This is exemplified in TPC-C as follows:

– There are functions of abstractions in the form of five transactions.
– There is a randomized arrival pattern for the transactions.
– There is an inter-dependence between the transactions. In particular, every New-Order will be accompanied by a Payment, and every group of ten New-Order transactions will produce one Delivery, one Order-Status, and one Stock-Level transaction [31].
– There are specified structures, inter-dependencies, contents, sizes, and growth rates for the data set, materialized in nine tables (i.e., Warehouse, District, Customer, History, Order, New-Order, Order-Line, Stock, and Item [30]).

A major shortcoming of several recent benchmarking attempts is the lack of any clear workload model, let alone a functional workload model as defined here. The resulting *micro benchmarks* measure system performance using one stand-alone compute unit at a time [6, 9, 26, 28]. They are lacking the functional view that is essential to benchmarking the diverse and rapidly changing big data systems aimed at servicing emerging application domains, as we explain next.

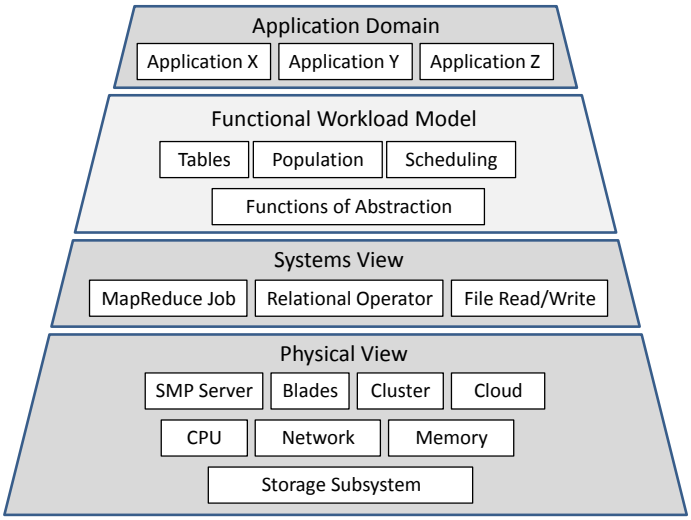

Fig. 1. The conceptual relations between application domains, functional workload models, functions of abstraction, and the system and physical views.

4.3 Functional benchmarks essential for big data

The ultimate goal of benchmarking is to facilitate claims such as "System X has been engineered to have good performance for the application domain Y." Evaluating big data systems is no exception and, as such, requires an approach based on the same level of abstraction that brought success to existing benchmarking efforts, such as that for OLTP and decision support.

The functional view explained previously mirrors the view detailed in early computer performance literature [23]. The functional view enables a large range of similarly targeted systems to be compared, because such an abstraction level has been intentionally constructed to be independent of system implementation choices. In particular, the functional description of TPC-C does not preclude an OLTP system to be built on top of, say, the Hadoop distributed file system, and its performance compared against a traditional RDBMS system. Such comparisons are essential for decisions that commit significant resources to alternate kinds of big data systems, e.g., for assigning design priorities, purchasing equipment, or selecting technologies for deployment.

The functional view also allows the benchmark to scale and evolve. This ability comes from the fact that functions of abstraction are specifically constructed to be independent of each other, and of the characteristics of the data sets they act upon. Thus, functions of abstraction can remain relatively fixed as the size of the data set is scaled. Further, as each application domain evolves, functions of abstraction can be deprecated or added, appear with changed frequencies or se-

quences, or performed on data sets with different characteristics. Thus, functions of abstraction forms an essential part of a scalable and evolving benchmark.

A common but incorrect approach is to benchmark systems using *physical* abstractions that describe hardware behavior, as illustrated by the *Physical View* layer in Figure 1. This view describes the CPU, memory, disk, and network activities during workload execution. Benchmarking systems at this level allows the identification of potential hardware bottlenecks. However, the physical level behavior changes upon hardware, software, or even configuration changes in the system. Thus, benchmarks that prescribe physical behavior precludes any kind of performance comparison between different systems — the differences between systems alter the physical behavior, making the results from physical benchmark executions on two systems non-comparable. Prior work on computer performance in general have already critiqued this approach [23]. Studies on Internet workloads have further phrased this concern as the "shaping problem" [29].

Another alternative is to benchmark systems using the *systems* view [10, 16], as illustrated by the *Systems View* layer in Figure 1. This approach captures system behavior at the natural, highest level semantic boundaries in the underlying system. For TCP-C, the systems view breaks down each function of abstraction into relational operators (systems view for RDBMS) or read/write operations (systems view for file systems). For MapReduce, this translates to breaking down each function of abstraction into component jobs and describe the per job characteristics. The systems view does enable performance comparison across hardware, software, and configuration changes. However, it requires a new systems view to be developed for each style of system. More importantly, the systems view precludes comparisons between two different kinds of systems servicing the same goals, e.g., between a MapReduce system and a RDBMS that service the functionally equivalent enterprise data warehouse workload. This would be a major shortcoming for big data benchmarks that seek to accommodate different styles of systems.

Hence, we advocate the functional view for big data benchmarks, as illustrated by the *Functional Worload Model* layer in Figure 1. It enables comparison between diverse styles of systems that service the same functional goals, while allowing the benchmark to scale and evolve as each application domain emerges.

5 Benchmark Lessons from Hadoop MapReduce

The field of big data encompasses many types of systems. Two important examples are relational databases and MapReduce. This section serves as the counterpart to Section 3. It discusses the state of limited empirical knowledge about how organizations use big data, which hinders the identification of representative functions of abstraction (Section 5.1). However, the empirical data hints toward the identification of some application domains, each of which subject to its own functional workload model (Section 5.2). Further, we argue that the effort to build a big data benchmark should go beyond MapReduce-specific benchmarks, some of which are deficient even as MapReduce-only benchmarks (Section 5.3).

5.1 Towards functions of abstraction for MapReduce

The widespread deployment of MapReduce is a relatively recent phenomenon compared with the established nature of systems that fall under TPC-C. As such, only within the last year did researchers undertake systematic comparison of MapReduce deployments within multiple industrial sectors [17]. The latest empirical insights indicate that the effort to extract Hadoop MapReduce functions of abstraction remains a work in progress. As the functional workload models capture the mixes of functions of abstraction, the lack of the latter also implies the lack of the former. Two bottlenecks are that more system deployments need to be surveyed and that information regarding the functional computation goals remains to be collected.

The limited nature of existing empirical data is due to the fact that MapReduce became accepted for business-critical computations only recently. The data in [17], while unprecedented for MapReduce, is limited to seven workloads. This is far from the breadth of the OLTP survey that preceded TPC-C. A key result from [17] is the diversity of observed behavior. This result indicates that we should survey more system deployments to systematically understand both common and outlier behavior. Even if functions of abstraction are extracted from the current, limited survey, there is no guarantee that these functions of abstraction would be representative of a majority of MapReduce deployments.

The lack of direct information regarding functional computation goals is due to the fact that current logging tools in the Apache Hadoop implementation of MapReduce collect only system-level information. Specifically, the analysis in [17] identified common MapReduce jobs using abstractions that are inherently tied to the map and reduce computational paradigm (i.e., input, shuffle, output data sizes, job durations, map and reduce task times). While such a systems-view has already led to some MapReduce-specific performance tools [10], this view becomes insufficient for constructing functions of abstractions that encompass both MapReduce and other systems.

A good starting point to define functions of abstraction within MapReduce deployments would be to capture the data query or workflow text at Hadoop extensions such as Hive [2], Pig [4], HBase [1], Oozie [3], or Sqoop [5]. The hope is that the analysis of a large collection of such query or workflow texts would mirror the empirical survey that led to the TPC-C functions of abstraction. A complementary effort is to systematically collect the experiences of human data scientists and big data systems administrators. A collection of such first-hand experiences should offer insights on what are the common big data business goals and the ensuing computational needs. The emergence of enterprise MapReduce vendors with a broad customer base helps expedite such efforts.

5.2 Speculation on application domains within MapReduce

The data in [17] allows us to speculate on the big data application domains that are addressed by the MapReduce deployments surveyed, notwithstanding the

limits outlined in Section 5.1. In the following, we describe the characteristics of these application domains.

A leading application domain is *flexible latency analytics*, for which MapReduce was originally designed [19]. Flexible latency analytics is indicated by the presence of some jobs with input and output data sets that are orders of magnitude larger than for other jobs, up to the "full" data set. This application domain has previously been called "batch analytics". However, as with other application domains such as decision support, the batch nature is due to the limited capabilities of early systems. Low latency is desirable but not yet essential; hence "flexible latency". The data in [17] indicates that different deployments perform vastly different kinds of analytics, suggesting that the application domain likely involves functions of abstraction with a wide range of characteristics.

Another application domain is *interactive analytics*. Evidence suggesting interactive analytics include (1) diurnal workload patterns, identified by visual inspection, and (2) the presence across all workloads of frameworks such as Hive and Pig, one of whose design goals was ease of use by human analysts familiar with SQL. The presence of this application domain is confirmed by human data scientists and systems administrators [8]. Low computational latency would be a major requirement. It is likely that this application domain is broader than online analytical processing (OLAP), since the analytics typically involve unstructured data, and some analyses are specifically performed to explore and identify possible data schema. The functional workload model is likely to contain a dynamic mix of functions of abstraction, with a large amount of noise and burstiness overlaid on a daily diurnal pattern.

Yet another application domain is *semi-streaming analytics*. Streaming analytics describes continuous computation processes, which often update time-aggregation metrics. For MapReduce, a common substitute for truly streaming analytics is to setup automated jobs that regularly operate on recent data, e.g., compute click-rate statistics for a social network with a job every five minutes. Since "recent" data is intentionally smaller than "historical" data, we expect functions of abstraction for this application domain to run on relatively small and uniformly sized subset of data. The functional workload model is likely to involve a steady mix of these functions of abstraction.

According to the seven deployments surveyed in [17], all three application domains appear in all deployments. While interactive analytics carries the most weight in terms of the number of jobs, they are all good candidates for inclusion in a big data benchmark, provided that they are confirmed by either empirical or human surveys of additional big data deployments.

5.3 MapReduce-specific benchmarks and their shortcomings

The success of MapReduce greatly helped raised the profile of big data. While the application domains currently dominated by MapReduce should be a part of big data benchmarks, the existing MapReduce benchmarks are not representative of all big data systems. The following examines some existing MapReduce benchmarks, some of which are deficient even as MapReduce-only benchmarks.

Some tools measure stand-alone MapReduce jobs [6, 9, 26, 28]. These tools lack the true concept of a multi-job *workload*. They are inherently limited to measuring only a narrow sliver of the full range of cluster behavior, as a real life cluster hardly ever runs one job at a time or just a handful of specific jobs. Such tools are insufficient for quantifying even just MapReduce performance.

Some tools do have a workload perspective, but assume more of a physical view [7]. They seek to reproduce the exact breakdown of jobs into tasks, the exact placement of tasks on machines, and the exact scheduling of task execution. This is a physical view because one cannot compare two MapReduce systems that, for example, have different algorithms for translating jobs into tasks, placing tasks onto machines, scheduling tasks to be executed differently, or even the same MapReduce software system that operates on two clusters of different sizes. Further, the attempt to reproduce a large amount of execution details introduces scalability issues for both the tracing tool and the trace replay tool [8, 24]. Such physical view is also insufficient for quantifying MapReduce performance, let alone the rest of big data.

Some tools have a systems-view of MapReduce workloads [10]. They are sufficient for measuring MapReduce performance, and are already used by Cloudera and its partners. However, as the systems view is specific for MapReduce, they are also insufficient for a more general big data benchmark.

Hence, identifying the functions of abstraction for MapReduce application domains is an important step for building a part of a general big data benchmark.

6 Recommendations for Big Data Benchmark

The concepts of functions of abstraction, functional workload model, and application domains help us develop a vision for a possible big data benchmark. This vision is captured in Figure 2.

Big data encompasses many application domains. Hence a big data benchmark also encompasses multiple application domains. OLTP is one domain. If confirmed by further survey, other possible domains are flexible latency analytics, interactive analytics, and semi-streaming analytics. There may be other application domains yet to be identified. The criteria for including an application domain is that (1) a trace-based or human-based survey indicates that the application domain is important to the big data needs of a large range of enterprises, and (2) sufficient empirical traces are available to allow functions of abstraction and functional workload models to be extracted.

Within each application domain, there are multiple functions of abstraction, extracted from empirical traces and defined in the fashion outlined in Section 4.1. While each specific system deployment may include a different set of functions of abstraction, the benchmark should include the common functions of abstraction from across all system deployments within the application domain. What is "common" needs to be supported by empirical traces.

There is also a representative functional workload model, extracted from empirical traces and defined in the fashion outlined in Section 4.2. Each specific

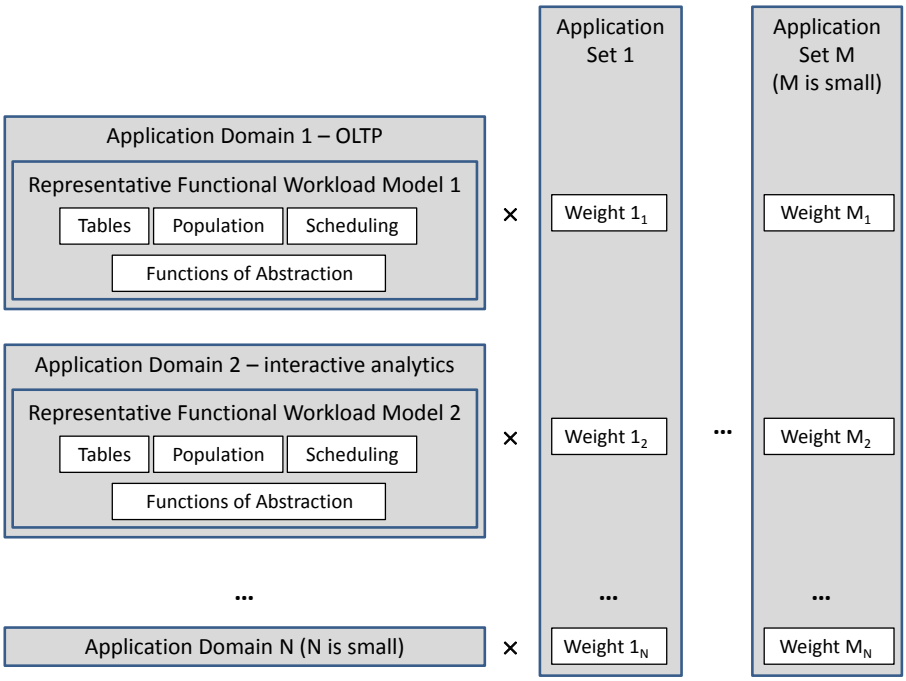

Fig. 2. A vision for a big data benchmark.

system deployment or application may include a different functional workload model, i.e., different mix of functions of abstraction, organization of data sets, and workload arrival patterns. The benchmark should include a single representative functional workload model for each application domain, i.e., a functional workload model that is not specific to any one application, greatly simplified, and yet typical of the entire application domain. The details of this representative functional workload model need to be supported by empirical traces.

Enterprises with different business needs would prioritize different application domains. Hence, each application domain would be given a different numerical weight in terms of the relative amounts of computation generated. As typical enterprises can be classified into a small number of categories, we speculate that their big data needs can likewise be classified. Hence, there would be several different sets of numerical weights, each corresponding to the big data needs of a particular type of enterprise. The actual numerical weights of each such application sets need to be supported by empirical traces across multiple application domains at each type of enterprise.

The traces and survey used to support the selection of functional workload models and numerical weights should be made public. Doing so allows the benchmark to establish scientific credibility, defend against charges that it is not repre-

sentative of real life conditions, and align with the business needs of enterprises seeking to derive value from big data.

Further, the hierarchical structure allows the coverage of diverse application domains and diverse business-driven priorities among application domains. At the same time, comparability between results is enhanced by limiting the weights of application domains to a fixed small set, and for each application domain, limiting the functional workload model to a single representative one.

It should also be noted that we could include TPC-C, and its successor, TPC-E, as the functional workload model for the OLTP application domain of a big data benchmark.

One issue unaddressed by this framework is to balance the need for a comparable benchmark against the need for the benchmark to keep up with evolution in how enterprises use big data. The choice within the TPC community is to keep the benchmarks static. This choice has successfully allowed the TPC style benchmarks to remain comparable across several decades, but has also led to some criticism that the benchmarks are stale. A different choice made by the SPEC community is to evolve its benchmarks regularly. This ensures that the benchmarks reflect the latest ways in which systems are operated, but limits comparable benchmark results to only those within the same version of each benchmark. The framework we proposed allows for either choice. Big data itself is likely to evolve over time. The pace of this evolution remains to be seen.

To summarize, in this paper we introduce several concepts essential for building a big data benchmark — functions of abstraction, functional workload model, and application domains. A big data benchmark built around these concepts can be representative, portable, scalable, and relatively simple. The next step in the process of building a big data benchmark would be to survey additional system deployments to identify the actual application domains worthy of inclusion in the benchmark, and within each application domain, identify the representative functional workload model and its functions of abstraction.

References

1. Apache HBase. `http://hbase.apache.org/`.
2. Apache Hive. `http://hive.apache.org/`.
3. Apache Oozie. `http://incubator.apache.org/oozie/`.
4. Apache Pig. `http://pig.apache.org/`.
5. Apache Sqoop. `http://sqoop.apache.org/`.
6. Gridmix. `HADOOP-HOME/mapred/src/benchmarks/gridmix` in Hadoop 0.21.0 onwards.
7. Gridmix3. `HADOOP-HOME/mapred/src/contrib/gridmixinHadoop0.21.0onwards`.
8. Personal conversation with data scientists and cluster operators at Facebook.
9. Sort benchmark home page. `http://sortbenchmark.org/`.
10. SWIM - Statistical Workload Injector for MapReduce. `http://github.com/SWIMProjectUCB/SWIM/wiki`.
11. TPC Benchmark A Standard Specification Revision 2.0. `http://www.tpc.org/tpca/spec/tpca_current.pdf`, 1994.

12. TPC Benchmark B Standard Specification Revision 2.0. `http://www.tpc.org/tpca/spec/tpcb_current.pdf`, 1994.
13. Anon et al. A measure of transaction porcessing power. *Datamation*, 1985.
14. L. Belady and C. Richter. The MCC Software Technology Program. *SIGSOFT*, 10, 1985.
15. D. Bitton, D. DeWitt, and C. Turbyfill. Benchmarking database systems: A systematic approach. In *VLDB 1983*.
16. Y. Chen. *Workload-driven design and evaluation of large-scale data-centric systems*. PhD thesis, UC Berkeley, 2012.
17. Y. Chen, S. Alspaugh, and R. Katz. Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads. In *VLDB 2012*.
18. B. Cooper et al. Benchmarking cloud serving systems with ycsb. In *SOCC 2010*.
19. J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI 2004*.
20. Z. Fadika, E. Dede, M. Govindaraju, and L. Ramakrishnan. Benchmarking mapreduce implementations for application usage scenarios. In *GRID 2011*.
21. M. Ferdman et al. Clearing the clouds, a study of emerging scale-out workloads on modern hardware. In *ASPLOS 2012*.
22. D. Ferrari. *Computer systems performance evaluation*. Prentice-Hall, 1978.
23. D. Ferrari. Characterizing a workload for the comparison of interactive services. *Managing Requirements Knowledge, International Workshop on*, 0, 1979.
24. B. D. Gowda. HiBench: A Representative and Comprehensive Hadoop Benchmark Suite. In *Presentations of WBDB 2012*.
25. J. Gray. The Benchmark Handbook For Database and Transaction Processing Systems - Introduction. In J. Gray, editor, *The Benchmark Handbook For Database and Transaction Processing Systems*. Morgan Kaufmann Publishers, 1993.
26. S. Huang et al. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *ICDEW 2010*.
27. P. O'Neil. A set query benchmark for large databases. In *Conference of the Computer Measurement Group 1989*.
28. A. Pavlo et al. A comparison of approaches to large-scale data analysis. In *SIGMOD 2009*.
29. V. Paxson and S. Floyd. Why we don't know how to simulate the internet. In *Proceedings of the 29th conference on Winter simulation*, 1997.
30. F. Raab. TPC-C - The Standard Benchmark for Online Transaction Processing (OLTP). In J. Gray, editor, *The Benchmark Handbook For Database and Transaction Processing Systems*. Morgan Kaufmann Publishers, 1993.
31. F. Raab, W. Kohler, and A. Shah. Overview of the TPC Benchmark C: The Order-Entry Benchmark. `www.tpc.org/tpcc/detail.asp`.
32. O. Serlin. IBM, DEC disagree on DebitCredit results. *FT Systems News*, 63, 1988.
33. O. Serlin. The History of DebitCredit and the TPC. In J. Gray, editor, *The Benchmark Handbook For Database and Transaction Processing Systems*. Morgan Kaufmann Publishers, 1993.
34. C. Turbyfill, C. Orji, and D. Bitton. As3ap: A comparative relational database benchmark. In *COMPCON 1989*.