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 the two most prominent technologies for such systems. Organizations use them to perform complex analysis on diverse and unconventional data types with fast growing data volumes. As more big data systems are deployed, the industry faces the challenge to develop representative benchmarks that can evaluate the capabilities of competing implementations. In this position paper, we argue for building future big data benchmarks using what we call a "functional workload model". This concept draws on combined experiences from standard benchmarks, exemplified by TPC-C. The functional workload model describes the functional goals that the system must achieve, the data access patterns, the load variations over time, and the computation required to achieve the functional goals. Abstracting functional workload models from empirical studies of MapReduce deployments represents the first step towards building truly representative big data benchmarks.

1 Introduction

Big data systems represent one of the fastest growing segments of the computer industry today. They allow organizations to store, manipulate, and analyze large and rapidly growing volumes of data from diverse and unconventional sources. The exploding trade press on big data suggests that it has spread beyond early adopters to traditional industry sectors. As new products appear and vendors issue competing claims, the need emerges for an objective method to compare the applicability, efficiency, and cost of big data solutions. In other words, there is a growing need for a set of standard big data performance benchmarks.

There have been a number of attempts at constructing big data benchmarks [18, 20, 21, 23, 30]. None of them has yet gained wide recognition and usage. The field of big data performance is in a state where results from one publication to the next are not comparable and often not even closely related. This was also the case for online transaction processing (OLTP) some twenty years ago and for decision support shortly thereafter.

In this position paper, we propose and argue for the use of a formal process to develop standard big data benchmarks. This process draws on experiences from successful industry standard benchmarks. We start by summarizing the properties of a good benchmark and we select TPC-C, the standard yardstick for OLTP performance, to illustrate our proposed benchmark development process (Section 2). To that end, we present an insider's retrospective on the development of TPC-C and discuss the process that led to the creation of a fully synthetic, yet representative benchmark (Section 3). Analyzing this process allows us to formalize three key concepts of the paper — *application domains*, the *functional workload model* and *functions of abstraction*; and to discuss how they enable the construction of representative benchmarks that can translate across different types of big data systems (Section 4). We then highlight that the process of identifying MapReduce functional workload models and their functions of abstraction remains bottlenecked on empirical data and outline some of the challenges specific to big data benchmarks (Section 5). Finally, we present a vision for the development of big data benchmarks that would span multiple application domains, each rooted in documented empirical data (Section 6).

2 To Define a Big Data Benchmark

Performance measurement for computer systems is not a new topic, and benchmark properties are well studied. To explore the path that would lead to the definition of a successful big data benchmark, we begin by reviewing pertinent properties of a good benchmark.

2.1 Properties of a good benchmark

The criteria for a good performance benchmark has been the topic of multiple publications [22, 25, 27]. Prior work on the topic 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.
- *Relevant*: The benchmark should focus on measuring technologies that are relevant and prominent in the market and align themselves with an area where demand for performance information is high.
- *Portable*: The benchmark should be fair and portable to competing solutions that target the needs of the same applications.
- *Scalable*: The benchmark should be able to measure performance of systems within a wide range of scale. As technology progresses system scales and their performance capabilities tend to increase. 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 and do not affect performance.

While the above speaks of the properties that a good benchmark should display, it does not address the methodology through which such a benchmark can be constructed. In the following sections we propose such a methodology, illustrate it using a successful standard benchmark, and review how it can be applied to the construction of a big data benchmark.

2.2 Examples of successful benchmark

The field of performance benchmarks is indeed crowded. But 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 the market over the last two decades: SPEC and TPC.

Each organization has published a number of benchmarks with various degree 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. Looking at the most published benchmarks from TPC and SPEC reveals the following:

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

Of the above benchmarks, TPC-C and TPC-D/H were defined using a similar process or abstraction. They can provide useful insight into the creation of a big data benchmark. To further explore and illustrate these concepts we will be examining the story of TPC-C with the goal to formalize the process at the core of its definition.

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 origins 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 [33]. 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 [34].

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 empirical 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 [35] and the Set Query Benchmark [29].

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 to be better aligned with the findings of the empirical survey 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) [31].

3.2 The TPC-C application domain

One of the main purposes of a benchmark is to evaluate and contrast the merits of various implementations of the same set of requirements. These requirements are driven from the common elements found in the many use cases [28] that populate broad computational categories such as OLTP, decision support, OLAP, analytics, stream processing or big data. We use the term "application domain" to refer to these computational categories. Specifically, an application domain encapsulates many per-customer use cases. While each use case will likely include some rare and customer-specific computational needs, the application domain focuses on the common computational elements among many similar use cases.

The original Order-Entry benchmark from DEC included two distinct 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 transactional portion of Order-Entry for the creation of its new OLTP benchmark: TPC-C.

An important aspect of the design of the transactional portion of Order-Entry is that it did not follow the model traditionally used for implementing use cases and building business applications. To illustrate this aspect we contrast the two design models.

The design of a business 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.
- *Transactions*: 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 traditional model results in an application that is well aligned with the business details of the targeted use case. As such, it is too specific to be representative of the broader and more generic aspects that characterize a whole application domain.

In contrast, benchmarking is a synthetic activity that seeks to be representative of a whole application domain. Its sole purpose is to gather relevant performance information as it pertains to any application within the targeted domain. Being free of any real business context, the elements of such a benchmark can be abstracted from a representative cross section of the application domain's use cases.

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 the system or measuring its performance. 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 sufficiently similar 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".

During the design of the Order-Entry benchmark, five abstract transactions were selected to encapsulate the activity most commonly found in real-life OLTP application. Such a simplification 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 performance information that are relevant and applicable 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 illustrates two key concepts — *functions of abstraction* and the *functional workload model*. In this section, we explain what they are, and how they form a methodology for constructing benchmarks that target specific application domains while accommodating diverse system implementations.

4.1 Functions of abstraction

The implementations of use cases within a particular application domain are made of computational functions, such as transactions, queries, or MapReduce jobs. As stated above, the design of a benchmark is only concerned with abstracting a cross-section of the most commonly found computational functions. We introduce the concept of *functions of abstraction* as a way of describing these abstracted computational functions. The intent is to capture a *functional* description of "what is being computed" at an abstract level; rather than a more concrete behavioral description of "how the computation is done".

The properties of a function of abstraction are as follows:

- *Generic*: The functional goal of the computation is described in a generic form, independent of the underlying system implementation, its software stack and the hardware behavior that results.
- *Atomic*: A group of transactions, queries, or jobs that must be executed together to serve a meaningful purpose (from a performance standpoint) should be considered as a single function of abstraction and not subdivided.
- *Unique*: Two different sets of transactions, queries, or jobs that serve the same functional goal are two realizations of the same function of abstraction.
- *Data independent*: The same function of abstraction can execute against data with different statistical properties and of different scales. Specifically, the description of the dataset acted upon is separate from the description of the function acting on the data.
- *Interdependent*: Their description includes the rules governing the interactions they have with each other.
- *Composable*: Any subset can be combined to create workloads of various levels of complexity.

TPC-C (i.e., Order-Entry) helps illustrate the concept. The benchmark is articulated around five functions of abstraction: a mid-weight read-write transaction (i.e., New-Order), a light-weight read-write transaction (i.e., Payment), a mid-weight read-only transaction (i.e., Order-Status), a batch of mid-weight read-write transactions (i.e., Delivery), and a heavy-weight read-only transaction (i.e., Stock-Level) [32]. 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.

Translating back to the earlier list, properties of functions of abstraction apply to TPC-C as follows:

– *Generic*: The functional goal is defined in terms of a set of data manipulation operations. 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.
– *Atomic*: Each transaction involves multiple data manipulation operations that operate as a whole.
– *Unique*: The five transactions serve five different functional goals.
– *Data independent*: The targeted data is defined separately through the distribution of values used as input variables. The data volume and schema is likewise specified separately.
– *Interdependent*: Their interactions is governed by the transactional properties of atomicity, consistency, isolation, and durability (i.e., the ACID properties).
– *Composable*: Workloads of various complexities can be created by using various combinations and mixes of the defined transactions. The TPC-C workload involves the combination of all five transactions, while the Payment transaction run by itself would become the TPC-A (i.e., Debit-Credit) workload.

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* captures in an implementation-independent (i.e., functional) manner the load that the system needs to service. This load is designed to be representative of the demands put on the system by an average use case within the application domain. The functional workload model includes three components - the functions of abstraction, their load pattern, and the data set they act upon.

The load pattern applied to the system is specified in terms of the execution frequency, distribution and arrival rate of each individual function of abstraction. In defining the load pattern, functions of abstraction can be combined to form coordinated groups with interdependencies.

The data set acted upon is specified in terms of its structure, inter-dependence between data elements, initial size and contents, and how it evolves over the course of the workload's execution.

The definition of these three components is limited to the essential functional goals of the particular application domain. The simplicity and lack of duplication that governs the definition of functions of abstraction must also be applied when specifying the load pattern and the data set that completes the functional workload model.

Again, TPC-C helps illustrate the concepts involved in the functional workload model:

- There are functions of abstractions in the form of five transactions.
- The load pattern involves a randomized arrival of transactions controlled by a weighted selection criteria and a random inter-arrival delay [32].
- 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 [32].
- There are specified structures, inter-dependencies, contents, initial 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 [31]).

In contrast, a major shortcoming of some of the recent big data *micro benchmark* proposals [6, 9, 26, 30] is the lack of any clear workload model, let alone a functional workload model as defined here. The resulting benchmarks measure system performance using one stand-alone compute unit at a time. They are lacking the functional view that is essential to benchmarking the diverse and rapidly changing big data solutions aimed at servicing emerging application domains, as we explain next.

4.3 Functional benchmarks essential for big data

We advocate the functional view for big data benchmarks, as illustrated by the *Functional Workload Model* layer in Figure 1.

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 from being built on top of, say, the Hadoop distributed file system, and its performance compared against a relational database system.

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 added, deprecated, involved in a different load pattern or performed on a data sets with different characteristics. Thus, functions of abstraction form an essential part of a scalable and evolving benchmark model.

Figure 1 also shows the *Systems View* and *Physical View*. In Section 5.4, we will explain the pros and cons of these alternate approaches using some examples of early MapReduce benchmarks. We will also discuss these approaches in the context of a general purpose big data benchmark.

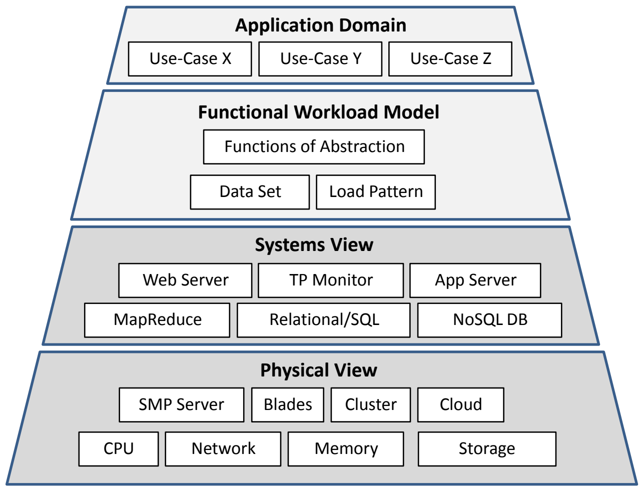

Fig. 1. The conceptual relations between application domains, functional workload models, functions of abstraction, and the system and physical views.

5 Extending these Concepts to MapReduce

For the functions of abstractions concept to be useful, it must be applicable to different types of big data systems. Two important examples are relational databases and MapReduce. Identifying functions of abstraction for big data is currently bottlenecked on limited empirical knowledge. However, emerging empirical data hints toward the identification of some application domains, each with its own functional workload model. This section discusses some benchmark lessons drawn from MapReduce and generally applicable to big data.

5.1 Towards functions of abstraction for big data

MapReduce and big data represent relatively new and rapidly expanding computing paradigms. The latest empirical insights [16] indicate that the effort to extract Hadoop MapReduce functions of abstraction remains a work in progress. The data in that study, 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 [16] is the diversity of observed behavior. This result indicates that we should survey more system deployments to 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 big data deployments.

A key shortcoming in the data from [16] is the lack of direct information regarding functional computation goals. This 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 [16] 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 extracting functions of abstractions related to big data application domains.

A good starting point to identify functions of abstraction would be to capture the data query or workflow text at MapReduce 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 woud involve collecting the experiences of bid 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 Emerging big data application domains

The data in [16] allows us to speculate on the emerging 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 [16] 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 diurnal workload patterns, identified by visual inspection, and 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 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 [16], all three application domains appear in all big data deployments. While interactive analytics carries the most weight in terms of the number of jobs, they are all good candidates for a targeted big data benchmark, provided that they are confirmed by either trace analysis or user surveys of additional big data deployments.

5.3 Challenges highlighted by MapReduce survey

The MapReduce survey in [16] also served to highlight properties of big data systems that represent new challenges in the development of big data benchmarks. They can be summarized as follows:

– *System diversity*: Big data systems tend to host multiple use cases from divergent application domains. Such diversity translates to significant, and sometimes mutually exclusive, variations in the design of big data systems. A good benchmark for big data needs to replicate realistic conditions across a range of application domains, and use metrics that translates across potentially divergent computational needs. Thus, it may be challenging for a big data benchmark to be representative and portable.
– *Rapid data evolution*: Big data systems use cases constantly and rapidly evolve. This reflects the innovations in business, science, and consumer behavior facilitated by knowledge extracted from big data. This change is often rooted in the underlying data set and likely outpaces the ability to develop a representative data set as part of the functional workload model. The challenge is to ensure that the benchmark keeps sufficient pace with such changes to remain relevant.
– *System and data scale*: Big data systems often involve multiple, distributed components, while big data itself often involves multiple sources of different formats. This translates to multiple ways for the system and the data to scale. Consequently, it is challenging for a big data benchmark to be truely scalable and adequately capture the multi-dimentional scaling paradigm of big data systems.
– *System complexity*: The 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 fairly complex in the absolute sense. The process of simplifications will need to be supported by objective and

empirical measurements to verify that all significant performance factors are captured by the benchmark.

5.4 Surveying MapReduce-specific benchmarks

The success of MapReduce greatly helped raise the profile of big data. The application domains currently dominated by MapReduce should be an important part of big data benchmarks. Some MapReduce benchmarks also help highlight limited approaches for building a general purpose big data benchmark.

The list below discusses these approaches, along with the corresponding MapReduce-specific benchmarks, and why they make it hard to achieve the desirable benchmark properties summarized in Section 2.

- Not having a true functional workload model. Bechmarks in this category focus on measuring stand-alone MapReduce jobs [6, 9, 26, 30]. They are inherently limited to measuring 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. This prevents the benchmark from achieving the "representative" property.
- Adopting a physical view of benchmarking. This category includes the Gridmix3 [7] Bechmark. It seeks 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 similar to other physical view benchmarks that reproduce CPU, memory, disk, and network activities. While useful for comparing hardware components, one cannot use physical view benchmarks to compare, for example, two MapReduce systems that have different scheduling algorithms or operate on data of different compression formats. Further, the attempt to reproduce a large amount of execution details introduces scalability issues for the benchmark execution tool [8,24]. "Portable", "scalable", and "verifiable" properties would be hard to achieve.
- Adopting a systems view of benchmarking. This view is adopted by the SWIM [10] benchmark. This approach captures system behavior at the natural, highest level semantic boundaries in the underlying system. For MapReduce, this translates to MapReduce-specific, per job characteristics such as the input and output data to the map() and reduce() functions. The systems view does allow many desirable benchmark properties to be achieved, and SWIM is already used by leading big data platform vendors. However, the systems view for MapReduce is not "portable" to other big data solutions. For example, the map() and reduce() abstractions do not directly translate to traditional RDBMS systems. Hence, the systems view is also insufficient for a general big data benchmark.

The functional view advocated in this paper specifically seeks to go beyond these limits. It aims to enable comparison between diverse styles of systems that service the same functional goals, but have different system architectures and exhibit different physical behaviors.

6 Vision 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.

Big data encompasses many *application domains*. OLTP is one domain. If confirmed by further survey, other possible domains are OLAP, flexible latency analytics, interactive analytics, and semi-streaming analytics. There may be other application domains yet to be identified. The criteria for identifying an application domain should be that a trace-based or user-based survey indicates that the application domain is important to the big data needs of a large range of enterprises, and that 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. The benchmark should include the functions of abstraction representing the common traces 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 system deployment or application will likely include a different 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.

The traces and survey used to support the selection of functions of abstraction and functional workload models should be made public. Doing so allows the benchmark to establish scientific credibility, defend against charges that it is not representative of real life conditions, and align with the business needs of enterprises seeking to derive value from big data.

Good first steps toward realizing the ideas in this paper include the Big-Bench benchmark [23], which includes English descriptions of what could be expanded into functions of abstractions for some Teradata use cases, and the CH-benchmark [17], which aims to combine the OLTP and OLAP application domains.

7 Summary and Future Work

In this paper we summarized the properties of a good benchmark and highlighted the need for a formal process to build a benchmark displaying these properties. We studied the creation of TPC-C as an example of such a process and formalized it by introducing several essential concepts — application domains, functions of abstraction, and the functional workload model. We studied the results of published surveys of big data systems as a first step toward defining application domains and functions of abstractions specific to big data, with the ultimate goal of creating a set of widely accepted and frequently used big data benchmarks.

The next step in the process of building the first standard big data benchmark would be to survey additional system deployments to identify the most prominent big data application domain and within this application domain to identify the representative functional workload model and its functions of abstraction. In the future, we should also consider combining multiple big data benchmarks to represent systems that increasingly host use cases from multiple application domains.

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, S. Alspaugh, and R. Katz. Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads. In *VLDB 2012*.
17. R. Cole et al. The mixed workload ch-benchmark. In *DBTest 2011*.
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 et al. 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. A. Ghazal et al. Bigbench: towards an industry standard benchmark for big data analytics. In *SIGMOD 2013*.
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. K. Huppler. The art of building a good benchmark. In *TPC Technical Conference 2009*.
28. I. Jacobson et al. *Object-Oriented Software Engineering - A Use Case Driven Approach*. Addison-Wesley, 1992.
29. P. O'Neil. A set query benchmark for large databases. In *Conference of the Computer Measurement Group 1989*.
30. A. Pavlo et al. A comparison of approaches to large-scale data analysis. In *SIGMOD 2009*.
31. 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.
32. F. Raab, W. Kohler, and A. Shah. Overview of the TPC Benchmark C: The Order-Entry Benchmark. `www.tpc.org/tpcc/detail.asp`.
33. O. Serlin. IBM, DEC disagree on DebitCredit results. *FT Systems News*, 63, 1988.
34. 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.
35. C. Turbyfill, C. Orji, and D. Bitton. As3ap: A comparative relational database benchmark. In *COMPCON 1989*.