Workload Diversity and Dynamics in Big Data Analytics: Implications to System Designers

Jichuan Chang, Kevin T. Lim, John Byrne, Laura Ramirez, Parthasarathy Ranganathan
Hewlett Packard Labs, Palo Alto

ABSTRACT

The emergence of big data analytics and the need for cost/energy efficient IT infrastructure motivate a new focus on data-centric designs. In this paper, we aim to better understand the design implications of data analytics systems by quantifying workload requirements and runtime dynamics. We examine four workloads representing big data analytics trends for fast decisions, total integration, deep analysis and fresh insights: an archive store, a columnar database enhanced with table compression, an analytics engine with distributed R, and a transaction/analytics hybrid system. These appliations demonstrate diverse resource requirements both within and across workloads as well as load imbalance due to data skew. Our observations suggest several directions to design balanced data analytics systems, including tight integration of heterogeneous, *active* data stores, support for efficient communication and data-centric load balancing.

Categories and Subject Descriptors

C.0 [**General**] System architectures;

C.4 [**Performance of Systems**]

General Terms

Design, Measurement, Performance

Keywords

Big data, analytics system, workload diversity, balanced system designs, system architectures

1. INTRODUCTION

We are entering a data-centric computing era, primarily driven by the need to exploit the vast and fast growing amounts of data. Deep analytics on large sets of diverse data at the fast pace of business operation represents a new set of opportunities. However, satisfying the performance, functionality, and scalability requirements of big data analytics is a challenging task by itself. Solution providers also face relentless pressure to reduce the system's total cost and energy. These factors together motivate new architectures designed and optimized for analytics solutions.

This paper aims to be a first step toward capable and efficient data analytics systems, starting from a more thorough understanding of workloads. We focus on a small yet representative set of emerging applications to cover the important aspects of "big data" analytics:

- **Big**: the vast volumes and fast growth of datasets, requiring cost-effective storage (e.g., HDDs) and scalable solutions (e.g., scale-out architetures);
- **Fast**: the need for low-latency data analytics that can keep pace with business decisions;
- **Total**: the trend toward integration and correlation of multiple, potentially heterogeneous, data sources;
- **Deep**: the use of sophisticated analytics algorithms (e.g., machine learning and statistical analysis);
- **Fresh**: the need for near real-time integration as well as analytics on recently generated data.

Our initial investigation shows that these workloads have *diverse* behaviors caused by their requirements and implementations, while some also demonstrate *dynamic* runtime behaviors due to staged or pipelined execution, program phases and imbalance [15].

For system designers, such workload characteristics suggest several important directions. First, the diversity of analytics solutions reflects a wide spectrum of user requirements. More importantly, it points to a gap which needs solutions that can handle "big, fast, total, deep and fresh" data at the same time. Instead of batch-replicating data across different systems, integrating multiple data sources and their corresponding analytics functions at a finer granularity can potentially fill the gap without sacrificing flexibility and modularity. This architecture, however, requires tight integration of heterogeneous, active data stores and a data-flow driven model for data and computation placement, scheduling and coordination. Second, workload dynamics reflect the heterogeneous nature of different data, data stores and data processing tasks in the system, so a one-size-fit-all design can incur resource utilization inefficiencies. Hardware and software techniques can mitigate such effects by embracing such diversity and dynamics to support better load balancing and efficient data movement.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ASBD'12, June 9, 2012, Portland, Oregon, USA.
Copyright © 2012 ACM 978-1-4503-1439-8/12/04... $10.00

2. WORKLOAD SELECTION

A key challenge in understanding "big data" analytics workloads is the lack of common standards: e.g., what are the operations on, sizes of and types of "big" data? Answers to these questions are often specific to given use-cases and only capture snapshots of a moving target, making it difficult to gain insights on the fundamental workload requirements.

To simplify the task of workload selection and setup, we approach the problem by first identifying the key classes of analytics operations and then carefully selecting their implementations. Specifically, we focus on two information processing paradigms: *relational algebra* and *linear algebra*, covering analytical workloads in databases, key-value stores, information retrieval, machine learning and graph analysis (which can be implemented by relational join or sparse matrix computation).

Table 1: Selected workloads

Workload	Requirements	Use-cases	Data store
LazyBase	Big, fast, fresh	Archive/SQL	File
MonetDB/C	Big, fast	TPC-H	Mmap
Presto	Big, fast, deep	PageRank	Memory
T/A-Fusion	Big, fast, total, fresh	TPC-C/H	Mem./file
Hadoop	Big, total	Sort	File

Although conventional databases already support relational algebra operations (e.g., TPC-H queries running on PostgreSQL), they are often inadequate for big data problems due to their performance, scalability or functionality limitations. In this work, we instead consider new and emerging use-cases with more modern implementations.

Table 1 summarizes the high-level characteristics of workloads studies in this paper. Below we provide more details on their use cases and implementations.

LazyBase. LazyBase [1] is a distributed, scalable analytics database. Built on an archive/batch-update model, it can achieve high throughput for both data loading and queries using a pipelined architecture that decouples load/update operations from read-only queries. It also uniquely allows the user to trade data freshness for query latency at runtime. Along the pipeline, data are batch ingested, sorted, indexed and merged into the database, where queries on the data in earlier pipeline stages get more fresh results but can incur longer latencies. Multiple stages of the pipeline can be distributed and executed outside the database server to improve scalability. LazyBase uses DataSeries [9] as the storage engine and PostgreSQL as the SQL query interface. We exercise LazyBase by importing data records and then running SQL queries.

MonetDB/C. MonetDB [2] is an open-source database with columnar store and high-performance hardware-conscious query processing. It uses main memory to improve query speed by memory-mapping needed column tables at run time. Modeling commercial columnar database implementations, our workload further enhances the MonetDB base code with support for scale-out storage and table compression (hence the workload name MonetDB/C). We use Google's Snappy compression library [3] and exercise MonetDB/C with TPC-H queries [4], profiling both the data loading and query execution phases.

Presto. Presto [5] is an extension to the R statistical computing environment [6] with new language primitives (e.g. distributed arrays) and runtime support for distributed execution. Besides scale-out and intra-node parallel execution enhancements, Presto also uses data lineage based incremental processing for continuous analytics and efficient fault-tolerance. Among a wide class of machine learning, graph mining and search algorithms supported by Presto, we examine a power method-based implementation of PageRank. The 100M webpage dataset is represented by a sparse connectivity matrix, scaled to represent the transition probability between webpages. The matrix is block-partitioned for parallel execution, and the block sizes are typically non-uniform due to data skew. During execution, individual worker threads are assigned to data partitions. Workers first load their partitions of the initial matrix into main memory, and then calculate the PageRank eigenvector in an iterative fashion. During each iteration a worker thread gets updated vector values from its peer workers over the network and uses them to calculate the new eigenvector.

T/A-Fusion. This workload represents a trend toward applying real-time analytics on online transactional data in order to reap the benefits of instant business decisions on fresh data. Such transaction/analytics fusion solutions were nearly infeasible in the past, partly due to the performance overheads of frequent data export, transformation and loading (ETL). However, enabled by advances in high-performance scale-out databases utilizing in-memory processing, such solutions have become possible (e.g., [16]). Instead of waiting for commercially-available products for unified transaction/analytics processing, we model a futuristic platform by connecting an in-memory transaction processing database ("TransDB") with a commercial columnar analytics database ("AnalyticsDB"). Similar to CH-benCHmark [17], TransDB is driven by an online transaction processing workload based on modified TPC-C [7], and the updates to the `customer`, `order` and `lineitem` tables are exported to the analytics database. The analytics queries are based on TPC-H [4] queries Q1, Q3, Q6 and Q18, modified to suit the TPC-C schema and generated date/time values. We report the transaction throughputs (in TPS) and analytics query latencies (in seconds), both during and after data export/loading.

Hadoop. As a comparison point to the other workloads, we also include results of Hadoop sort [8]. Hadoop is a Java implementation of MapReduce. It allows easy programming to support highly parallel analytics on structured or unstructured data, using large clusters of commodity servers.

3. WORKLOAD CHARACTERISTICS

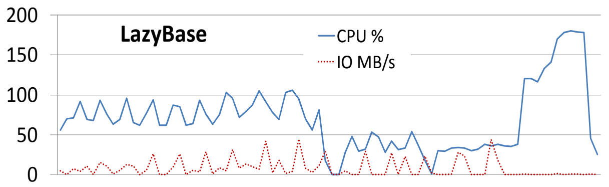
To understand the characteristics of our data analytics workloads, we analyze their runtime statistics collected while running on a cluster of servers. The Presto workload was run on four 2-socket, 6-core 2.66 GHz Xeon X5650 systems (HyperThreading enabled), each with 96 GB of RAM and a 10 GbE NIC. Each system has an SATA SSD drive; however as Presto is an in-memory system, the disk is not utilized beyond the initial data loading. All other workloads were run on up to 6 single-socket, 4-core 2.93 GHz Xeon X5570 systems, each with 6 GB of RAM, a 10Gb/s Ethernet adapter, and a 500 GB hard drive.

Figure 1 and 2 illustrate the resource utilization rates of our workloads over the course of their execution. For long-running workloads such as LazyBase, MonetDB and Presto, we only plot partial utilization traces that represent the key execution phases. The y-axis represents total resource utilization, with the units depending on the data plotted (see the legend of each figure). For example, the LazyBase figure shows the CPU utilization in percent (where 100% refers to a single core being fully utilized), as well as the I/O bandwidth in MB/second.

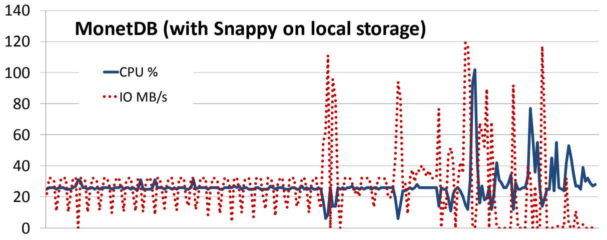
As shown in the figures, there is significant variation in the resource utilization across several workloads. LazyBase tends to be a highly CPU dominated workload. In contrast, MonetDB tends to be primarily I/O dominated throughout the execution time; CPU utilization is generally below 40%. Presto shows significant use of both CPU and network, while Hadoop sort is largely dominated by both IO and network utilization. TransDB as part of the T/A-Fusion workload shows high CPU and network utilization, and low I/O traffic partly due to its in-memory implementation. AnalyticsDB has both high CPU utilization and spikes of network and I/O activity. Although all systems are oriented toward analytics of large data, there is no commonly-shared trend in resource usage among all workloads.

However, certain workloads have commonalities in their design that are reflected by the resource utilization. For example, both Presto and TransDB are designed to be scale-out, in-memory systems. As a result, both systems have high network utilization (for data sharing and movement), and their in-memory design allows them to achieve high CPU utilization. Both MonetDB and the Analytics DB are column-oriented databases, with similar CPU utilizations and brief spikes of I/O utilization.

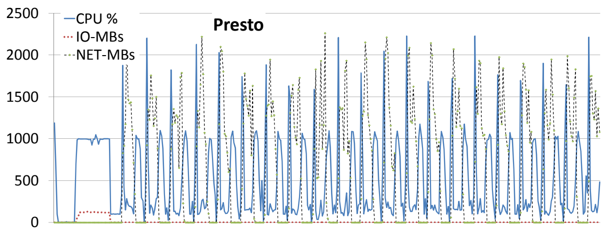
Analyzing each workload more closely, there are distinct changes in the utilization rates across different execution phases. LazyBase shows particularly high CPU utilization during the data importing phase. MonetDB has very steady behavior during the load phase, followed by a sharp spike of I/O utilization after the load has completed (flushing data to disks). Meanwhile, the query phase shows bursty I/O behavior with very high utilization peaks. TransDB maintains high CPU utilization throughout the data loading, and has fairly significant network utilization due to communication between TransDB nodes. The plot for the Analytics DB shows the CPU being fairly well utilized during the queries. Additionally, spikes in network receive traffic and IO write traffic can be seen, which correlate to



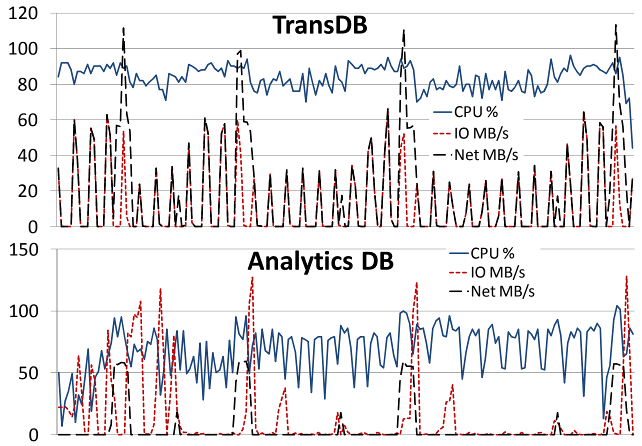
(a) LazyBase (data loading followed by queries)



(b) MonetDB (data loading followed by queries)



(c) Presto (data loading followed by matrix iterations)



(d) T/A Fusion (queries with periodic ETL)

Figure 1: Workload Resource Utilization Traces

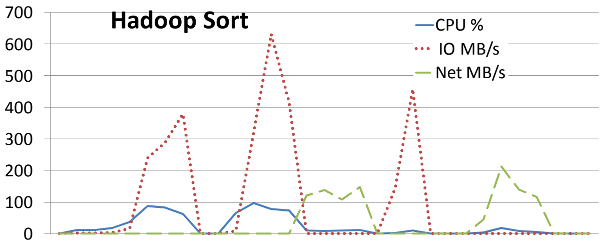


Figure 2: Hadoop Sort resource utilization

the phases where data is exported from TransDB to the Analytics DB.

Furthermore, for certain workloads the distinct resource utilization changes are inherent to the application framework's operation model. Presto utilizes only the CPU during the initial startup phase. Subsequently, there is a clear alternating pattern between the utilization of CPU and network resources. This pattern is representative of the Presto framework; the high CPU segments are during matrix operations, while the very high network segments are triggered by heavy data communication between workers. The Hadoop sort workload has significant I/O and network utilization during specific intervals. Similar to Presto, this behavior clearly represents the Hadoop framework: the initial loading/map phase, writing of intermediate results prior to the shuffle phase, data transmission during the shuffle phase, and finally the writing of results and replicas during the reduce phase.

Table 2: Amdahl's metrics for workloads and phases
("-" indicates one of the resources is not used)

	MIPS vs. MBs	Whole run	Phase 1	Phase 2	Phase 3
LazyBase	CPU/IO	108.3	71.1	44.5	307.8
	CPU/Net	-	-	-	
Presto	CPU/IO	234.3	104.4	-	
	CPU/Net	11.7	-	11.5	
MonetDB /C	CPU/IO	13.1	10.6	20.0	
	CPU/Net	-	-	-	
TransDB	CPU/IO	36.9			
	CPU/Net	20.6			
Analytics DB	CPU/IO	55.4			
	CPU/Net	102.8			
Hadoop Sort	CPU/IO	3.6	3.3	1529	0.5
	CPU/Net	13.1	-	1.9	-

Table 3: Compute-centric metrics

	IPC	LLC MPKI
LazyBase Load	0.82	0.52
LazyBase Query	0.91	0.45
Presto PageRank	0.71	8.19
MonetDB/C Load	1.11	0.51
MonetDB /C Query	1.07	1.53
TransDB TPC-C	0.82	2.52
Analytics DB	1.62	0.40
Hadoop Sort	0.99	2.03

Extending our analysis of workload characteristics, we additionally examine the "Amdahl's rule of thumb" for balanced system designs. The Amdahl's rule (i.e., one instruction per bit of IO, or 8 MIPS per MBs) [10] and recent follow-up work [11] both use high-level metrics to summarize workload resource usage and balance requirements.

However, prior work only examines the balance ratio between compute and IO bandwidths, measured over the whole workload execution. Here we extend the concept to a class of metrics for balanced system design. Specifically, Table 2 includes a new metric to measure the ratio between compute and network bandwidths, relevant to scale-out and communication-heavy workloads.

Kozyrakis et al. [11] also consider intra-workload variance using both average and standard deviations (for normal distributions); however, the `Average+2Sigma` values reported for Bing, Hotmail, and Cosmos, are 10-100 times lower than the maximum, indicating much higher variance than those of normal distributions. To address this issue, we also report metrics for different execution phases.

Table 2 clearly illustrates both inter- and intra-workload diversity. Comparing ratios in the "whole run" column, the compute-to-IO ratios range from 3.6 (Hadoop Sort) to 234.3 (Presto), while the compute-to-Network ratios have a relatively smaller range (i.e., one order of magnitude difference). For workloads consisting of multiple phases, the difference can be as high as 3000x (Hadoop CPU/IO phase 2 vs. phase 3).

In comparison, Table 3 shows a more compute-centric view of low-level hardware behavior, using traditional metrics of average instructions per cycle (IPC) and last-level cache misses per kilo-instruction (LLC MPKI). We collect performance counter statistics using Oprofile for both kernel and user-level code. Except for the Analytics DB, most workloads have relatively low IPC (~1) and reasonable LLC miss rates. Comparing Table 3 with the detailed utilization traces (Figures 1 and 2) highlights the need for in-depth analysis for these big data analytics workloads. For example, although the Presto IPC is only 0.71— indicating poor utilization of processor microarchitectural resources—the previous figure shows high CPU utilization during the matrix operation phases. The combination suggests a mismatch between Presto's compute requirement and the underlining processor implementation, potentially caused by the high memory-to-compute ratio and poor reference locality of sparse matrix operations (MPKI=8.19). We also note that the variance of workload metrics observed by the processor and memory system can be significantly smaller than those system-level metrics reported in Table 2.

4. EARLY OBSERVATIONS

Most of these modern workloads do not need sophisticated parameter tuning (except for Hadoop, where we choose the best performing task and file chunk sizes). Although our current data are still early results, potentially requiring future work to fine-tune and further understand these workloads, nevertheless we can make some interesting observations.

4.1 Implications of Diversity

First, the workload diversity, manifested as both inter- and intra-workload Amdahl's metric variation, indicates huge potential for improved efficiency via balanced system designs. Because of the diverse resource requirements, one-size-fit-all designs typically cannot satisfy the different resource balance ratios for a collection of analytics workloads. The resulting imbalance leads to either resource over-provisioning (hence wasted cost/energy) or bottlenecks that increase the execution time and consumed energy. This observation is also reflected in the recent trend toward systems optimized for important workloads, e.g., memcached [12], key-value stores [13] and web 2.0 workloads [14], which seek to regain the balance between compute, IO and network by matching the requirement of a specific workload.

Exploiting intra-workload diversity can provide additional cost and energy efficiency benefits. Systems can adapt to execution phases either *statically* or *dynamically*. Based on profiling or application-level information, a static approach can employ heterogeneous, phase-optimized components working together to accomplish the entire analytics task. On the other hand, a dynamically balanced system can detect phase behaviors and adjust resource provisioning at runtime (e.g., through DVFS or PowerNap [20]) to match workload requirements. Below we use two examples to show the potential benefits of offloading.

4.2 Changing the balance ratios

One approach toward balanced system designs is to alter the workload's resource requirement through data or compute transformations such as indexing, approximation and compression. Such transformations can trade compute resource for I/O or network to match the existing resource provisioning ratios.

Using compression as an example, Figure 3 shows the execution time of MonetDB/C for data loading and query phases, normalized against a no-compression, local-storage baseline, with different compression options. For example, local versus R4 represents local compression/storage versus 4 remote storage nodes with compression offload (emulated by the remote nodes, each with one active processor core); GZ and SP represent gzip and Snappy compression libraries, respectively.

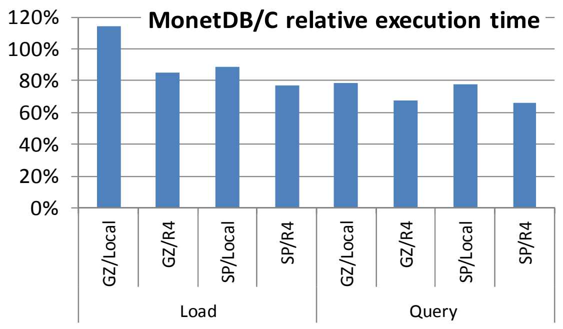
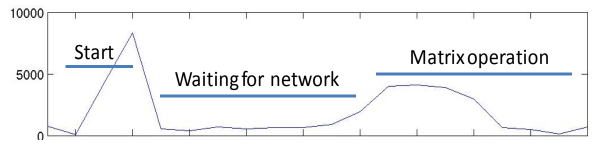
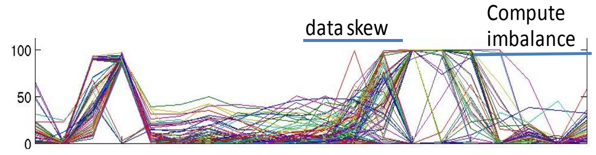


Figure 3: MonetDB compression options/impacts



(a) Aggregated CPU utilization



(b) Per-core utilization traces

Figure 4: Presto CPU utilization (one iteration)

On our testbed cluster, we observe that MonetDB with compression usually runs faster (by up to 30%) than the baseline, except when loading with gzip on the local node. In this case, the compute overheads from gzip compression outweigh the savings from reduced IO transfers. In addition, for a given algorithm, compression offloading to remote storage can usually further improve performance. Finally, the selection of compression algorithm and implementation (gzip vs. Snappy) can also impact performance, especially for the data loading phase, which involves compute-heavy compression operations. These data are collected using general-purpose cores, and active data stores with hardware accelerated compression could potentially achieve even higher performance and efficiency benefits.

Similarly, we also experimented with offloading for LazyBase. By distributing the data-centric processing stages (ingest or sort) to another node, we can reduce the data loading time by up to 25%.

4.3 Support for Tight Integration

We also observe two workloads with heavy network usage: Presto as well as T/A-Fusion using TransDB and Analytics DB. Presto and TransDB's in-memory implementations eliminate the IO bottlenecks but at the same time require matching network bandwidths for data movement. The 10Gb/s Ethernet is sufficient for TransDB, mainly because TPC-C transactions have relatively small payloads. However, this is not the case for Presto, which has HPC-like heavy network usage. As shown in Figure 4(a), the bandwidth limitation can often delay the completion of CPU tasks as they wait for the input vector to be updated.

Based on the function type, data reduction ratio and data dependency, offloading data processing tasks can either decrease or increase network bandwidth requirement. Purely shipping control and code should have modest network requirement, and certain offload functions can further reduce the amount of data moved by first applying in-place filtering and aggregation. Offloading is likely to reduce network traffic for such workloads. On the other hand, compared to local compression and sorting, offloading the two functions to remote data stores can increase network traffic and instead motivates the need for high-bandwidth energy-efficient networks. To achieve optimal designs, a holistic approach is needed to make intelligent tradeoffs between the compute/IO efficiencies of offloading and the potential overheads of data placement/movement.

Beyond support for efficient data movement, tight integration can further address skew-induced load imbalance. As shown in Figure 4(b), the per-core CPU utilization traces are initially synchronized when a new iteration starts, which corresponds to the initial spike of CPU utilization in Figure 4(a), but the following data movement operations can take different amount of time to finish, mainly because the block partitions have different sizes (imperfect load balancing). Similarly, the run lengths of matrix operations also differ because the sizes of their matrix partitions are very different. These two factors together reduce the average CPU utilization as shown in the second half of Figure 4(b).

To address the issue of data skew in analytics workloads, an ideal load balancer should consider both compute and data induced costs, including data partitioning, data locality as well as the I/O and communication overheads for data migration. We envision future solutions that not only integrate data access and analytics functions to support "big, fast, total, deep and fresh" analytics, but also can manage data placement, function offloading, task scheduling and coordination/synchronization holistically in order to better match the inherent data-flow requirements between various subsystems and data stores.

5. CONCLUSIONS

We are entering an exciting data-centric era with significant needs for big data analytics. To build cost- and energy-efficient solutions for big data analytics, emerging workloads designed and optimized for these tasks must first be studied in depth.

In this paper, we examine several representative workloads, each fitting into one or more aspects of the "big, fast, total, deep, fresh" data analytics paradigm. Early results in workload characterization have shown important implications for system designers. First, there is significant inter- and intra- workload diversity in terms of usage patterns and resource requirements, favoring solutions with balanced subsystems that are specifically optimized for important workloads or execution phases to achieve optimal efficiency. Second, the dynamics of the workloads—due to either execution phases, change of resource requirements or data-induced load imbalance—motivate a holistic approach that can tightly integrate heterogeneous subsystems and efficiently manage the various data and compute components across the system. Our preliminary examination of potential designs, including remote storage and compression offloading, indicates strong potential for novel architectures targeted at big data analytics systems.

ACKNOWLEDGMENTS

The authors gratefully acknowledge Brad Morrey and Craig Soules for help with the LazyBase workload, Indrajit Roy and Shivaram Venkataraman for help with the Presto workload, Shimin Chen for suggestions and help on the schema and queries for T/A-Fusion workload, and Mitch Wright, Siamak Tavallaei and Dave Carlisle for helpful discussions.

REFERENCES

[1] J. Cipar, et al., LazyBase: Trading Freshness for Performance in a Scalable Database. EuroSys 2012.

[2] MonetDB. http://www.monetdb.org. 2012

[3] Google Snappy. http://code.google.com/p/snappy.

[4] TPC Council. TPC-H. http://www.tpc.org/tpch/.

[5] S. Venkataraman, et al., Using R for Iterative and Incremental Processing. Draft under submission.

[6] The R project for statistical computing. www.r-project.org

[7] TPC Council. TPC-C. http://www.tpc.org/tpcc/.

[8] Apache Hadoop. http://wiki.apache.org/hadoop/Sort.

[9] E. Anderson. Capture, Conversion, and Analysis of an Intense NFS Workload. FAST 2009.

[10] J. Gray and P. Shenoy. Rules of Thumb in Data Engineering. TechReport MS-T R-99-100. 2000.

[11] C. Kozyrakis, et al. Server Engineering Insights for Large-Scale Online Services. IEEE Micro, vol. 30(4), July/Aug. 2010.

[12] M. Berezecki et al. Many-Core Key-Value Store. IGCC 2011.

[13] D. Andersen, et al. FAWN: a Fast Array of Wimpy Nodes. SOSP 2009.

[14] K. Lim, et al. Understanding and designing new server architectures for emerging warehouse-computing environments. ISCA 2008.

[15] Y. Kwon et al. A Study of Skew in MapReduce Applications. OpenCirrus Summit, 2011.

[16] A. Kemper and T. Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. ICDE 2011.

[17] R. Cole, F. Funke, et al. The Mixed Workload CH-benCHmark. DBTest 2011.

[18] D. Meisner, B. Gold and T. Wenisch. PowerNap: Eliminating Server Idle Power. ASPLOS 2009.