Amdahl's Law in Big Data Analytics: Alive and Kicking in TPCx-BB (BigBench)

Daniel Richins[†] Tahrina Ahmed[*] Russell Clapp[*] Vijay Janapa Reddi[†‡]

[†]*The University of Texas at Austin* [*]*Stanford University* [*]*Intel* [‡]*Google*

ABSTRACT

Big data, specifically data analytics, is responsible for driving many of consumers' most common online activities, including shopping, web searches, and interactions on social media. In this paper, we present the first (micro)architectural investigation of a new industry-standard, open source benchmark suite directed at big data analytics applications—TPCx-BB (BigBench). Where previous work has usually studied benchmarks which oversimplify big data analytics, our study of BigBench reveals that there is immense diversity among applications, owing to their varied data types, computational paradigms, and analyses. In our analysis, we also make an important discovery generally restricting processor performance in big data. Contrary to conventional wisdom that big data applications lend themselves naturally to parallelism, we discover that they lack sufficient thread-level parallelism (TLP) to fully utilize all cores. In other words, they are constrained by Amdahl's law. While TLP may be limited by various factors, ultimately we find that single-thread performance is as relevant in scale-out workloads as it is in more classical applications. To this end we present core packing: a software and hardware solution that could provide as much as 20% execution speedup for some big data analytics applications.

1. INTRODUCTION

Big data analytics is the application of advanced analytic techniques to large, diverse structured and un-structured data. It empowers users with a granular perspective of complex business operations and customer habits that rarely find their way into traditional data warehouses or standardized reports. Using techniques such as predictive analytics, data mining, statistics, machine learning, and natural language processing, big data analytics enables its users to understand the current state of the business and track complex and continuously evolving behavior such as end-user customer traits.

From an industry perspective, *big data analytics has been oversimplified.* Much previous research has been conducted into big data [1,2,3,4,5,6,7], but this has often taken a broad approach, covering not only data analytics but also media streaming, social networking, real-time services, etc. Often, data analytics is reduced to simple, sample applications intended as demonstrations rather than benchmarks. While not wholly without value, these micro-benchmarks are ultimately not representative of industry. To merit serious study, an analytics benchmark should be characterized by (1) **realism**, the use of applications that are representative of real-world applications, including complexity and size; (2) **comprehensiveness**, or a thorough exercise of functionalities; and (3) **usability**, which ensures reproducibility of studies.

In this work, we study TPCx-BB, which distinguishes itself as a *comprehensive* data analytics benchmark, *representative* of industry workloads, designed to be eminently *usable*. TPCx-BB (hereafter "BigBench") is a recent industry-standard workload developed through collaboration from multiple industry partners. BigBench is especially attractive because it simulates a real-world scenario: a modern retailer with both an online and physical store presence which collects a wealth of data about its customers, competitors, stores, and online reputation. It seeks, through 30 data analytics "queries," to use this data for economic gain. Each query operates on a subset of the data in a unique way: `Q01`, for example, identifies the top products sold together in given stores (structured data), while `Q28` classifies product review sentiment (positive or negative) based on the textual customer reviews (un-structured data) that are logged into the database.

We perform the first comprehensive characterization of BigBench. We analyze BigBench from two key perspectives. First, we undertake a (micro)architectural study of its execution on an enterprise-level cluster. Compared to prior work, which often tend to use only a handful of (micro) benchmarks, the 30 component queries show *diverse behavior* that is masked when they are considered in aggregate. This diversity arises from exercising different Hadoop and Spark capabilities such as MapReduce, machine learning, natural language processing, pure query language queries, and others in various combinations on structured, semi-structured, and un-structured data. Each BigBench query is a complete application, executing multiple operations on various sources of data to produce unique and insightful takeaways. As a result, we see that some queries show great computational diversity (`Q02`, `Q09`, and `Q28`), while others show great memory (`Q06` and `Q19`) or I/O diversity (`Q05` and `Q16`). It is impossible to capture these extremes with only a handful of benchmarks.

Second, we analyze the parallelism characteristics of BigBench to reveal a startling dearth of thread-level parallelism (TLP), in stark contrast to widely held assumptions regarding big data's scale-out potential. The lack of TLP arises from various sources, but its effect is always the same: cores are being left unused by big data. This suggests that relying solely on scale-out resources is insufficient; systems must also be designed to actively monitor TLP and take proactive measures to boost single-thread performance as necessary.

Table 1: Comparison of benchmark quality. Several benchmarks related to big data are compared based on their compliance to three qualities of good benchmark design. A *realistic* benchmark reflects complex real-world scenarios; reliance on micro-kernels (grep, sort, etc.) or computational kernels (k-means, naive Bayes, etc.) undermines benchmark realism. A *comprehensive* benchmark covers all the use cases within a domain; too broad a domain, though, lacks focus and neglects important use cases. A *usable* benchmark has a simple, strong, and robustly repeatable installation and running process. A complex installation interferes with a user's ability to run the benchmark at all. BigBench alone meets all three requirements.

Benchmark	Realism	Comprehensiveness	Usability
BigDataBench [1]	Mostly micro-benchmarks or kernels	Slightly too broad but covers SQL, MapReduce, ML, and user-defined	Very involved setup with extensive configuration
Big Data Benchmarks [8]	Only one non-micro-benchmark	Limited to SQL and user-defined	Out of date; complex installation process
CloudSuite [3]	All workloads are complex and capture real-world use cases	ML, user-defined, MapReduce; only 3 analytics workloads	Formerly very difficult to install; simplified by Docker images
DCBench [5]	Mostly micro-benchmarks or kernels	MapReduce, ML, and user-defined	Out of date; complex installation process
HiBench [7]	Mostly micro-benchmarks or kernels	MapReduce, ML, SQL	Minimal configuration needed; detailed installation instructions
LinkBench [9]	Mimics Facebook computation	Limited to graphs	Needs extensive configuration
YCSB [10]	Creates realistic testing environment for stressing data serving systems	Framework for testing; not a standalone benchmark	Requires integration with real workload; difficulty varies
BigBench [11]	Emulates a retailer running thorough analytics on customer, product, and competitor data	Hadoop and Spark; MapReduce, user-defined, SQL, NLP, ML	Uses Cloudera/HortonWorks; installed by unpacking files; one-line command to run

In response, we propose **core packing** to actively restrict computation to a minimal set CPU cores as a means of proactively enabling Intel Turbo Boost [12]. Our experiments show that Turbo Boost is not normally being applied even when it has the potential to boost performance. Through small architectural and software changes, we estimate that core packing could provide up to 20% speedup in some BigBench queries.

In summary, we make the following key contributions:

- We present the first microarchitectural characterization and system-level analysis of TPCx-BB, an open-source industry-standard big data analytics benchmark.
- Contrary to the assumptions of some previous benchmarks, we reveal that big data analytics is a rich and diverse field with distinct execution characteristics, meriting different foci for study and improvement.
- We experimentally quantify a major bottleneck that arises from limitations on TLP, which indicates that even though parallelism may be abundant in these big data applications, it is still critical to focus on improving scale-up performance in addition to the traditional scale-out systems' research approach.
- We demonstrate the potential of core packing to proactively assist Intel Turbo Boost to improve performance and energy savings in big data analytics.

This paper is organized as follows. Section 2 justifies studying BigBench (TPCx-BB) in an already crowded field from an industry perspective. Section 3 presents the details of BigBench and how it is run. Section 4 describes our experimental setup. In Section 5 we present our (micro)architectural analysis of BigBench and expose the diversity of behaviors among queries that would be hidden by treating them as a single black box. Section 6 exposes the low TLP exhibited by all of the queries. In Section 7 we introduce core packing as a measure to proactively enable Turbo Boost during low-TLP execution. Section 8 presents related work. The paper concludes in Section 9.

2. MOTIVATION

With the wealth of big data-related benchmarks available [1, 3, 5, 7, 8, 9, 10], one might question whether there is any value in studying one more (BigBench). To this we emphatically answer in the affirmative. We outline here three major qualities a benchmark should possess to merit serious study: *realism*, *comprehensiveness*, and *usability*. Table 1 shows how each of several benchmarks compares in compliance to these qualities. Only BigBench captures all three.

Realism As the field of big data has matured, the complexity and realism of the benchmarks has improved. Initial benchmarks focused on micro-benchmarks (e.g. sort, grep, relational queries, etc.) and later benchmarks began incorporating computational kernels (e.g. k-means clustering, naive Bayes, etc.). However, to represent industry-strength workloads, a benchmark must capture complex scenarios.

BigBench relies exclusively on such complex workloads, developed through open discussion with major industry players, including Microsoft, Oracle, Cloudera, and Intel, among others [13]. As a result, BigBench's queries are complex, realistic tasks that represent real-world applications.

Comprehensiveness A comprehensive benchmark must cover all relevant use cases and components—it is not enough to use Hadoop; rather, the various functionalities of the Hadoop ecosystem must be exercised. Hadoop has long since moved beyond just MapReduce: MapReduce, query language (e.g. SQL), machine learning, and user-defined applications are all part of a comprehensive benchmark.

BigBench relies on up-to-date releases from the Hadoop ecosystem, including Apache Spark. Its 30 queries cover a huge variety of use cases for the software: MapReduce, natural language processing, machine learning, query language interaction, and user-defined functions [11].

Usability Many existing benchmarks are not designed for use with industry-standard workflows that integrate with big data package solutions, such as Cloudera and HortonWorks, which is detrimental to repeatability and usability over time. Ideally, installation of a benchmark must be as simple as unpacking the files. Running the benchmark must be just as

easy: with a single command a user should be able to specify important parameters, run the benchmark, and get a score.

BigBench is designed to work seamlessly with industry solutions. It uses Cloudera or HortwonWorks to streamline the installation of Hadoop and related products. BigBench can be run, start to finish, with a single command, while still offering the flexibility to study individual components.

3. BIGBENCH

We present an overview of BigBench. Section 3.1 describes the design and operation of BigBench and explains the different data types that feature in BigBench. Section 3.2 explains the process of running BigBench. Section 3.3 explains how the data volume is controlled.

3.1 A Modern Retailer Implementation

BigBench is a big data workload *specification* to standardize how researchers and industry evaluate big data workloads [14]. The specification is a TPC [11] benchmark ("TPCx-BB"), for which Intel has provided an open source reference implementation [15], based on Hadoop, Hive [16], and Spark [17]. In this paper, we use Intel's open source implementation to conduct our evaluations.

BigBench simulates a modern retailer, with both a physical and online store presence. The benchmark operates on structured, semi-structured, and un-structured data "collected" from stores, online reviews, customers, and competitors through the execution of 30 queries, each extracting some value from the data, e.g. finding the products most frequently purchased together or tracking the product pages that customers viewed before making their online purchases. The 30 queries are comprehensive in that they reflect the end-to-end processing for big data analytics and faithfully capture the typical real-world use-case for a retail business. We omit the complete query descriptions here for brevity. Full details can be found in the TPCx-BB specification [11], but we show high-level query properties in Table 2.

Henceforth, we use the terms "query" and "application" interchangeably, reflecting the fact that each query represents a complete data analytics operation that merits its own study.

Table 2: Primary query data types and operations. The combinations of operation type—MapReduce (MR), machine learning (ML), natural language processing (NLP), user-defined function (UDF), and pure query language (QL)—operation specifics, and data type—structured (Str), semi-structured (Smi), and un-structured (UnS)—give rise to tremendous diversity. Compiled from TPCx-BB specification [11].

Query	Data; Oper.	Query	Data; Oper.	Query	Data; Oper.
Q01	Str; UDF	Q11	Str; QL	Q21	Str; QL
Q02	Smi; MR	Q12	Smi; QL	Q22	Str; QL
Q03	Smi; MR	Q13	Str; QL	Q23	Str; QL
Q04	Smi; MR	Q14	Str; QL	Q24	Str; QL
Q05	Smi; ML	Q15	Str; QL	Q25	Str; ML
Q06	Str; QL	Q16	Str; QL	Q26	Str; ML
Q07	Str; QL	Q17	Str; QL	Q27	UnS; UDF/NLP
Q08	Smi; MR	Q18	UnS; UDF/NLP	Q28	UnS; ML
Q09	Str; QL	Q19	UnS; UDF/NLP	Q29	Str; UDF
Q10	UnS; UDF/NLP	Q20	Str; ML	Q30	Smi; UDF/MR

3.2 Running BigBench

A complete BigBench run consists of four stages, though the score for a run consists of only the latter three stages:

1. **Data Generation** This stage generates the data which will be used in the later processing stages of the benchmark. It is representative of the retailer collecting data from its customers, stores, and competitors. While data generation is an essential part of the benchmark, its operation is not included in the final score.
2. **Load Test** Data generated in the previous stage is imported into the metastore. This tests loading efficiency. This is the first stage that contributes to the benchmark score. Time on an analytics cluster is valuable. Hence, importing data for later analysis must not be too costly.
3. **Power Test** All 30 queries are run in order, each query in isolation. This is representative of *online* analytics, where the goal is to produce results as quickly as possible through sequential execution (i.e. latency matters). It is most useful for detailed microarchitecture-level analysis to guide future processor design, as the behavior of each query can be individually studied.
4. **Throughput Test** Two or more query streams are executed concurrently. The throughput test is representative of *offline* data analytics. Multiple queries are run simultaneously with the goal of maximizing throughput (i.e. latency is unimportant). A query stream is a predetermined ordering of all 30 queries; a larger stream count will almost certainly lead to longer latency, but the throughput test measures throughput rather than latency. Thus, a user may produce a better score by increasing the number of streams.

Official BigBench scores are a measure of the throughput of the three sub-tests: the load, throughput, and power tests. See the appendix for details on how a BigBench run is scored.

3.3 Scalability Knob

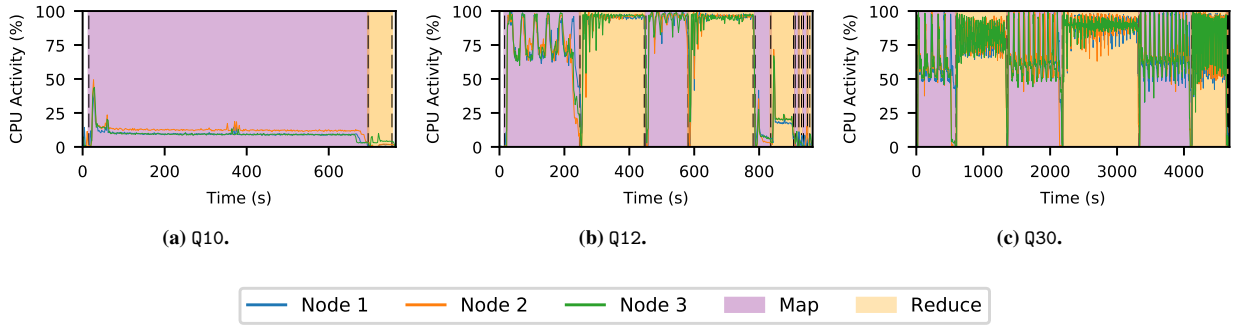
A user can specify the "scale factor" of the data generation phase to control the total volume of data. Each scaling factor unit is equal to $\sim$1 GB of data; hence, a factor of 1000 would produce $\sim$1 TB of input data. Though BigBench supports arbitrary scale factors, there are only seven values for which benchmark results may be officially reported: 1000, 3000, 10,000, 30,000, 100,000, 300,000, and 1,000,000. These data sizes easily scale to fill nearly any cluster.

4. EXPERIMENTAL SETUP

We present here our experimental methodology. Section 4.1 describes our cluster setup. Section 4.2 describes how we aggregate the performance counter data from multiple machines. Section 4.3 explains the different scale factors we use to evaluate the effects of data volume on our conclusions. We recognize the significance of Hadoop parameter tuning; Section 4.4, therefore, discusses our tuning decisions.

4.1 Cluster Configuration

All our experiments are conducted on actual silicon (not simulation). We primarily use two clusters. The first is a 3+1 cluster: 3 "workers" and 1 "master" node. Each of the worker nodes features a single Xeon E5-2699 v3 18-core CPU with



(a) Q10. **(b)** Q12. **(c)** Q30.

Figure 1: CPU activity over time. The activity is shown in aggregate across all the cores on each node. Transitions between map and reduce phases are marked by vertical dashed lines and the phases are colored appropriately.

384 GB of RAM, as detailed in Table 3. Our second cluster is a similarly configured 8+1 cluster which we use to validate our findings more generally (Section 6.3). In Section 7 we briefly use an E5-2699 v4 cluster for illustration purposes.

4.2 Performance Measurement

Given our cluster configuration, we focus our study on the worker nodes. We do not study the master node, as its responsibility is administrative—that is, it coordinates the work of the worker nodes but does no actual computation.

We gather CPU performance counter data using Intel's EMON [18] tool. It is a low-level command line tool that allows us to specify the performance counter events and attributes to monitor in Intel processors.

We aggregate the data from all cores, on each of the three worker nodes, and then average them to obtain a final metric. We justify this decision by observing that the worker nodes exhibit nearly identical execution behavior, as shown in Figure 1. In this figure, we plot the CPU activity of three of the queries as a function of time. For each of the three queries, we show the CPU activity on each of the three worker nodes. For clarity, we shade map and reduce regions in purple and orange, respectively (setup regions are left white).

Comparing the CPU activity across the nodes, we see that in all three queries, the three worker nodes are nearly identical. Much of the time, the nodes' plot lines overlap so well that nodes 1 and 2 cannot be seen. This indicates that all three nodes are executing almost the exact same thing at all times. In retrospect, this finding is unsurprising: Hadoop is designed to evenly distribute comparable work across nodes.

Some regions show noticeable differences between the nodes, for example in Figure 1c at the end of the first map phase. These differences are uncommon and resolve quickly. We speculate that they arise due to inevitable slight work imbalances between the nodes and resolve as soon as a given map or reduce phase has finished. These are distinct from the stragglers identified by Google that they attribute to machines with real performance issues (e.g. failing disks) [19].

Table 3: Experimental setups. The 3+1 cluster has three worker nodes while the 8+1 cluster has eight. We evaluate only the worker nodes, as they bear the burden of computation.

	3+1 Cluster Worker	8+1 Cluster Worker
CPU	1× Xeon E5-2699 v3	2× Xeon E5-2699 v3
Cores	*18*	*18*
Threads	*36*	*36*
Frequency	*2.3 GHz*	*2.3 GHz*
Turbo Freq	*3.6 GHz*	*3.6 GHz*
LLC	*45 MB*	*45 MB*
Memory	384 GB	256 GB
Channels	*4*	*8*
Speed	*DDR4-2133*	*DDR4-2133*
Storage	7× Intel SSD	1× Intel SSD
Network	10 Gbps	10 Gbps
BB Scale Factor	1000	3000

4.3 Data Volume

In most of our experimental evaluation we use a scaling factor of 1000, i.e. a total input size of ∼1 TB. But in order to demonstrate that our findings are generally applicable even at larger scale factors, we also conduct a number of experiments using a scale factor of 3000. Because our primary cluster lacks sufficient disk space for these experiments, we use a cluster from a cloud computing service (Section 6.1). We continue to use 3000 when we turn to a larger cluster (Section 6.3), which we utilize for demonstrating real-world implications of our findings even at larger scales.

4.4 Hadoop Parameter Tuning

Hadoop workloads are notoriously sensitive to parameter tuning—settings such as data compression algorithms, data split sizes, and the overlap of map and reduce phases can strongly influence performance. For our 3+1 clusters, we use light tuning for each query but we do not spend significant time fine-tuning every parameter. We omit tuning when using the cloud computing service due to the time and concomitant cost that would be required for proper tuning. In Section 6 the 8+1 cluster is highly tuned and corroborates the results of the less-tuned 3+1 cluster. We verified our tuning is sufficiently robust that it does not alter our conclusions in this paper.

5. BIGBENCH QUERY ANALYSIS

In this section, we conduct the first deep (micro)architectural characterization of BigBench. In contrast to prior work that has collapsed big data analytics into a small set of applications [3, 5, 8, 20], we show that data analytics applications

Table 4: Microarchitectural characteristics of individual queries. Each row of the table corresponds to a different metric (with repeats in the upper and lower halves of the table). *Hence, each row (distinguished by its color) should be thought of as an independent heat map. The stronger the color of a cell, the higher its value.* These metrics reveal a rich variety of execution behaviors not captured when viewed as a whole, as in Table 5, so we must focus on deeper analysis into individual queries to drive design.

Metric	Q01	Q02	Q03	Q04	Q05	Q06	Q07	Q08	Q09	Q10	Q11	Q12	Q13	Q14	Q15
CPI	1.38	0.757	0.826	0.783	1.52	1.32	1.13	0.903	1.99	1.08	1.36	0.980	1.32	1.46	1.18
Front-End Bound Cycles (%)	13.3	16.1	16.0	16.3	13.2	13.9	13.6	15.8	12.6	15.2	12.9	15.8	13.6	12.6	14.0
Back-End Bound Cycles (%)	60.9	45.8	48.6	47.2	63.2	60.3	57.5	51.2	66.2	46.4	62.5	53.0	60.9	63.6	58.3
Branch Mispredict Rate (%)	1.94	0.819	0.629	0.917	0.409	1.21	1.83	1.12	0.301	2.43	2.08	0.721	1.57	1.24	1.78
L1D MPKI	16.0	9.00	10.2	11.1	10.8	17.0	15.4	11.8	5.69	21.2	17.1	6.51	17.4	12.3	17.2
L2 MPKI	12.4	3.87	3.58	3.85	4.47	12.3	11.4	6.40	3.92	17.1	12.7	4.68	11.7	9.13	11.6
LLC MPKI	2.27	0.731	0.674	0.678	1.21	3.73	1.96	1.19	1.05	1.26	2.43	0.914	2.54	1.96	2.20
Memory Bandwidth (MB/s)	7640	6100	5660	6070	5610	25600	4720	7490	4840	3130	12200	5110	14800	10100	6340
IO Bandwidth (MB/s)	52.1	119	89.8	111	7.75	34.9	36.0	54.7	45.8	21.5	55.0	36.8	19.5	34.5	14.2

Metric	Q16	Q17	Q18	Q19	Q20	Q21	Q22	Q23	Q24	Q25	Q26	Q27	Q28	Q29	Q30
CPI	1.04	1.21	0.885	1.09	1.45	1.06	0.970	1.34	1.38	1.42	1.42	1.61	1.17	1.74	0.786
Front-End Bound Cycles (%)	15.9	13.5	14.1	15.0	12.9	14.5	13.8	13.6	13.3	13.5	13.0	12.1	14.7	12.8	16.4
Back-End Bound Cycles (%)	53.9	59.0	49.1	48.6	62.6	55.6	54.7	60.6	61.5	62.0	62.4	64.1	54.9	64.8	46.9
Branch Mispredict Rate (%)	1.19	2.01	1.64	2.51	1.38	1.70	1.61	1.88	2.03	1.25	0.938	1.93	2.71	0.507	0.689
L1D MPKI	12.2	18.2	14.1	22.7	13.9	15.1	12.6	15.3	16.8	14.3	12.8	16.1	17.9	5.58	8.85
L2 MPKI	9.43	12.3	11.8	17.8	10.2	10.1	9.95	13.2	13.0	10.2	7.32	12.2	15.6	4.46	3.54
LLC MPKI	2.10	2.24	2.16	1.65	2.26	1.75	1.79	2.49	2.31	2.30	1.57	2.49	2.56	1.30	0.705
Memory Bandwidth (MB/s)	13800	5540	12100	2560	9330	8400	9410	15400	10800	11400	4560	6340	4870	6150	6330
IO Bandwidth (MB/s)	274	41.9	23.8	22.9	28.3	115	53.2	27.9	19.9	17.5	21.8	7.91	67.4	26.2	90.7

are rich and varied, necessitating individualized study to adequately understand and address performance issues.

The throughput test (Table 5), which runs multiple streams concurrently, highlights no specific issues. It achieves generally average performance across the board, with the single exception of a high skew toward back end-bound cycles.

Since the throughput test is composed of multiple queries running simultaneously, it obscures a variety of performance bottlenecks unique to subsets of the queries. In contrast, the power test (Table 4), executing queries in isolation, reveals several execution inefficiencies. *Studying the power test serves two important purposes: (1) it exposes performance bottlenecks obscured by the throughput test, and (2) it exposes bottlenecks unique to a latency-sensitive workload.* Thus, by study of the power test, performance issues can be directly addressed, thereby indirectly benefiting the throughput test.

Accordingly, we analyze the power test and explore differences between it and the throughput test. Our analysis is divided into three categories: compute (Section 5.1), memory (Section 5.2), and I/O (Section 5.3). We summarize query heterogeneity via principal component analysis in Section 5.4.

Table 5: Microarchitectural metrics of the throughput test on each of the three worker nodes. These aggregate throughput statistics hide the rich underlying heterogeneity shown in Table 4.

Metric	Node 1	Node 2	Node 3
CPI	1.00	1.00	1.01
Front End-Bound Cycles (%)	15.3	15.3	15.3
Back End-Bound Cycles (%)	53.9	54.0	54.2
Branch Mispredict Rate (%)	0.928	0.922	0.924
L1D MPKI	10.7	10.7	10.8
L2 MPKI	5.85	5.80	5.87
LLC MPKI	1.14	1.13	1.14
Memory Bandwidth (MB/s)	10220	10140	10430
IO Bandwidth (MB/s)	94.6	95.4	95.9

5.1 Compute Heterogeneity

The throughput test shows a stable cycles-per-instruction (CPI) of 1.0 across all three nodes (Table 5). Over 50% of cycles are back end-bound, compared to only 15% for front end-bound cycles. Branch prediction is over 99% accurate, in confirmation of the low fraction of front end-bound cycles.

In contrast to the throughput test, the power test queries demonstrate strong heterogeneity in execution characteristics (Table 4). The CPI measures show a low of 0.757 in `Q02` (1.32 IPC) and a high of 1.99 in `Q09` (0.503 IPC). The extent of CPI variation suggests that many of the queries present great opportunity for per-core execution improvement.

The only metrics consistent across the power test and throughput test are the front end- and back end-bound cycles. Confirming past findings [21], all queries are strongly back end-bound. The percentage of front end-bound cycles has a small range of 12.1% to 16.3%, while the back end-bound cycles' range is slightly larger at 45.8% to 66.2%. Prior work attributed this phenomenon to cache access latency [3, 20].

All 30 queries are well served by the CPU's branch predictors, with the worst misprediction rate appearing in `Q28` at 2.71%. This agrees with research from Jia, et al. [5] but stands in contrast to that of Kanev, et al. [21]. We note in the latter, though, that the profiled workloads are not restricted to data analytics. See Section 8 for more detailed discussion.

5.2 Memory Heterogeneity

Table 5 shows that the throughput test is well served by the existing cache hierarchy. The number of misses per kilo-instruction (MPKI) rests at only 10.7, 5.9, and 1.1 for the L1D, L2, and LLC caches, respectively. Furthermore, our systems seem to be over-provisioned for the memory bandwidth

requirements of the throughput test. With an average throughput of only 10.4 GB/s, the workload would be adequately served by even a single memory channel. With four channels of DDR4-2133 SDRAM, our systems provides an aggregate bandwidth of 68 GB/s, far in excess of the workload's needs.

In contrast, the isolated queries show strongly heterogeneous cache behavior. The cache miss rates range from 5.58 (Q29), 3.58 (Q03), and 0.674 (Q03) MPKI to 22.7 (Q19), 17.8 (Q19), and 3.73 (Q06) MPKI for the L1D, L2, and LLC, respectively. The throughput test MPKIs fall in the middle of these ranges, as would be expected. The higher miss rates probably merit investigation and remediation.

We note that in some queries, such as Q10, Q19, and Q28, the L2 miss rates are nearly as high as the L1D miss rates, suggesting that the working set sizes of the data exceed the size of both the L1D and L2 caches, a trend not seen in the throughput test. In no case is the LLC miss rate close to the L2 miss rate. Thus, the LLC size (45 MB) seems adequate to capture a meaningful portion of the working set size.

The queries' memory bandwidth utilization is also well below the system capacity. At the low end, Q19 only demands 2560 MB/s average memory bandwidth. Even Q06, with an average bandwidth utilization of 25.6 GB/s, is safely below the total sustainable bandwidth. However, unlike the throughput test, Q06's bandwidth requirements would not be met by a single memory channel, and queries Q11, Q13, Q16, Q19, and Q23 have bandwidth requirements high enough that they could be impeded in older technology systems.

5.3 I/O Heterogeneity

Though we cannot separate disk from network bandwidth usage (both are lumped together as I/O bandwidth), we can examine how usage requirements compare to disk and network capacities. Our seven SSDs provide an aggregate 2.3 GB/s while our 10 Gbps Ethernet translates to 1.25 GB/s.

Although the throughput test bandwidth requirements are an order of magnitude smaller than either disk or network bandwidth capacity, the power test shows that queries demand a range of I/O bandwidth requirements, ranging from 7.75 (Q05) to 274 MB/s (Q16). There is a 35× difference from the lowest to the highest consumer. While our systems provide ample bandwidth, a system setup utilizing older technology, such as spinning hard disks or 1 Gbps Ethernet, would likely impose a performance penalty on the larger consumers, such as Q02, Q04, Q16, and Q21. The throughput test hides this bottleneck. Ferdman et al. concluded that I/O bandwidth was over-provisioned [3] but studied a very limited subset of data analytics applications. The work of Zhou et al. confirms a bandwidth bottleneck for spinning hard disks that does not exist for SSDs [2] but does not consider network bandwidth.

5.4 Heterogeneity Summary

As a summarizing takeaway, we use principal component analysis [22] (PCA) to show the presence of major clusters, which indicates the importance of selecting widely varying and representative applications for study. PCA is a statistical method used to produce uncorrelated variables from a set of possibly correlated variables (Table 4). The smaller the distance between two queries in Figure 2 (i.e. the lower the height of the connecting arc), the more similar the queries

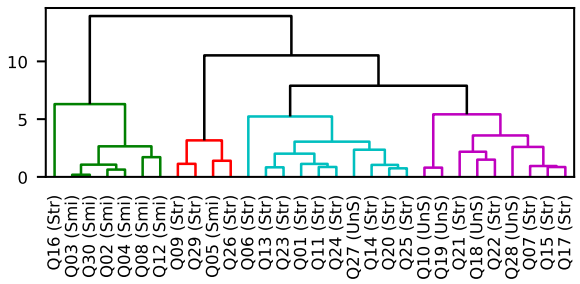


Figure 2: Query similarity dendrogram. For each query we denote the data type on which it primarily operates: structured ("Str"), semi-structured ("Smi"), or un-structured ("UnS").

are. For example, Q03 and Q30 show a very small distance and Table 4 shows very similar metrics for the two queries.

As indicated by the figure's coloring, we have identified four clusters with broadly similar characteristics, two of which have a strong showing of semi-structured (green) and un-structured data queries (purple). The structured data queries dominate the other two clusters. The green cluster queries, all but one of which operate on semi-structured data, are related to users' web click streams. The one semi-structured query outside of the green cluster (Q05, in the red cluster) is more oriented toward customer interest prediction, which generally aligns with the red cluster. The un-structured queries mostly fall in the purple cluster, but there is an equally strong showing of structured queries there. Rather than the kind of data operated upon, these queries are united in that they analyze store sales and correlate those with other categories such as review sentiment, demographics, or store properties. The final cluster, cyan, contains queries that segment products, customers, or stores.

The presence of clusters demonstrates the importance of selecting widely varying and representative applications for study. This analysis can aid future studies that seek to study a subset of BigBench through detailed simulation.

6. AMDAHL'S LAW IS ALIVE AND WELL

From a high level, big data applications are generally viewed as a natural fit for multicore parallel computing. Indeed, Hadoop is frequently touted for its impressive scale-out capabilities, and big data is an intuitive fit for extracting parallelism. Unfortunately, individual applications may not fit the "infinite scalability" model so well.

In this section, we conduct a thread-level parallelism (TLP) study of BigBench execution and demonstrate that not all big data applications are so amenable to scale-out computing as one would hope. We measure the TLP and determine that it is the biggest bottleneck to performance in Section 6.1. The lack of TLP persists even as the data volume increases in Section 6.2. We quantify the performance impact of this finding by scaling the number of cores and operating frequency for BigBench queries in Section 6.3. Then, in Section 6.4, we explore some of the causes of the low thread-level parallelism.

6.1 Limited Thread-Level Parallelism

Though some of the queries showed no major microarchi-

Table 6: TLP for per-query activity (power test) and per-node activity (using a 4-stream throughput test). Higher values indicate greater TLP. Cell shading indicates how little TLP a query exhibits.

Power Test					
Query	**Eff. TLP**	**Query**	**Eff. TLP**	**Query**	**Eff. TLP**
Q01	42.4%	Q11	60.7%	Q21	47.5%
Q02	59.1%	Q12	46.7%	Q22	48.4%
Q03	65.3%	Q13	70.4%	Q23	72.3%
Q04	65.2%	Q14	64.5%	Q24	58.1%
Q05	71.2%	Q15	30.6%	Q25	64.1%
Q06	76.3%	Q16	61.1%	Q26	38.7%
Q07	23.4%	Q17	27.6%	Q27	38.4%
Q08	51.2%	Q18	39.8%	Q28	17.6%
Q09	81.3%	Q19	15.7%	Q29	77.7%
Q10	21.7%	Q20	55.7%	Q30	67.8%
Throughput Test					
Node 1	81.6%	Node 2	81.9%	Node 3	82.5%

tectural bottlenecks (Section 5), there is actually a pervasive system-level problem that affects all the queries and extends even into the throughput test: they all spend a significant amount of time in a halted CPU state—i.e. they have limited thread-level parallelism (TLP). Measuring the total amount of non-halted CPU time gives us a view of how effectively the available cores are being utilized. Though there are various reasons for low CPU utilization, ultimately the result is the same: *these supposedly inherently scalable scale-out workloads are not scaling out to utilize the available cores.*

We measure TLP by taking the ratio of two counters: `CPU_CLK_UNHALTED.REF_TSC` and `TSC`. `TSC` refers to the *Time Stamp Counter* that is incremented every cycle at the nominal clock rate (regardless of DVFS changes). The other counter is an analogous per-core counter that is only incremented when the core is not in a fully halted state. Reasons for halted cycles include processes yielding the core(s) due to blocking I/O operations, synchronization between threads, waiting for input, etc. Thus, TLP is a measure of how many cores are actually being utilized over the full execution.

In Table 6, we show the TLP (as a percentage of available parallel resources) for the power and throughput tests. In the throughput test, we report this metric for each of the three workers; in the power test, we average the metric across the workers, the same as we did in Section 5.

TLP should ideally sit at or near 100%, indicating that the application is scaling out to all cores. Measures short of 100% indicate that CPU cores are being halted. In the throughput test, for example, various cores are halted nearly 20% of the time, meaning that, effectively, nearly 7 of the 36 CPU cores are doing nothing at any given point in time. Run in isolation (i.e. the power test), the applications show more concerning behavior. They have a wide range of TLPs, ranging from 81.3% in `Q09` down to 15.7% in `Q19`.

6.2 Data Volume Scaling Analysis

We increase the scale factor to see if TLP increases as the data volume increases. Even as the scale factor is increased, we find that TLP remains constrained. We experimented with increasing the scale factor from 1000 to 3000 to determine its effect on TLP. Recall from the experimental setup section (Section 4.3) that our primary cluster lacks sufficient storage to accommodate the larger scale factor; we consequently

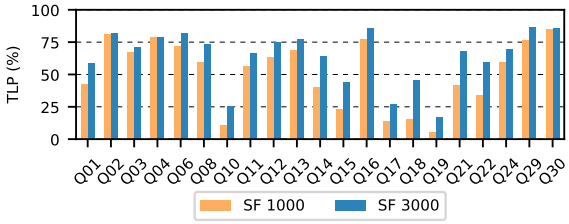


Figure 3: Thread-level parallelism at different scale factors. TLP almost universally rises as the scale factor is increased from 1000 to 3000. Even with the improvements, though, the TLPs are remarkably low, showing that single-thread performance will continue to be an issue.

utilized a cloud compute cluster hosted on Amazon Web Services [23] to examine TLP changes for a large subset of the queries, as shown in Figure 3. There was no contention for resources as we had exclusive access to the physical machines.

Extremely low-TLP queries (e.g. `Q10`, `Q17`, `Q19`) unsurprisingly show the greatest increases in TLP. However, we find that the increase is sublinear and still very low. High-TLP queries are unaffected or show limited impact from the increase in scale factor. Even at the higher scale factor, all queries' TLPs leaves ample room for improvement.

It is likely that with further increases in scale factor we would see a steady rise in TLPs as they approach some upper limit. However, even at a scale factor of 3000, a 3+1 cluster is strained, taking multiple days to finish the power test. A production setup would likely scale out at that point, thereby maintaining the relatively low TLPs that we have seen.

6.3 Core and Frequency Scaling Analysis

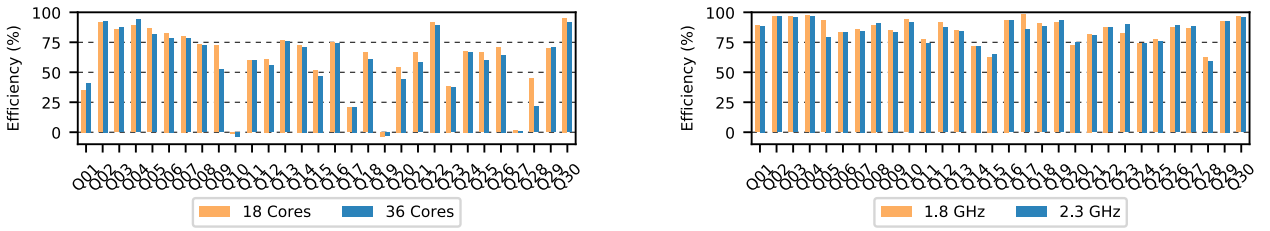
The limited TLP of big data queries manifests even in finely-tuned clusters. To illustrate this, we turn to our 8+1 cluster (Table 3), a distinct setup with carefully tuned parameters for each query. Its results exemplify the real-world impact of limited TLP. Though this careful setup captures 98% of TLP in the throughput test, the low TLP problem persists in the power test. We use a scale factor of 3000.

The effect of adding cores to the cluster can be quantitatively measured. In Figure 4a, we restrict the number of active cores and measure the efficiency of increasing the core count from 9 cores (18 threads) to 18 cores (36 threads) and 36 cores (72 threads). We define efficiency as follows:

$$Efficiency = \frac{RunTime_1 - RunTime_2}{RunTime_1 \cdot (1 - \frac{Resource_1}{Resource_2})}$$

This metric is the ratio of the actual change in running time to the expected change in running time. It quantifies how well an application utilizes increased resources. Most queries benefit from increased core counts, but some are largely unaffected, e.g. `Q27`. The queries with high core-scaling efficiency correlate with the high-TLP queries, e.g. `Q02`, `Q04`, and `Q30`. `Q10` and `Q19` show negative efficiencies, indicating increased running time; this may be attributable to noise or the overhead of managing more threads.

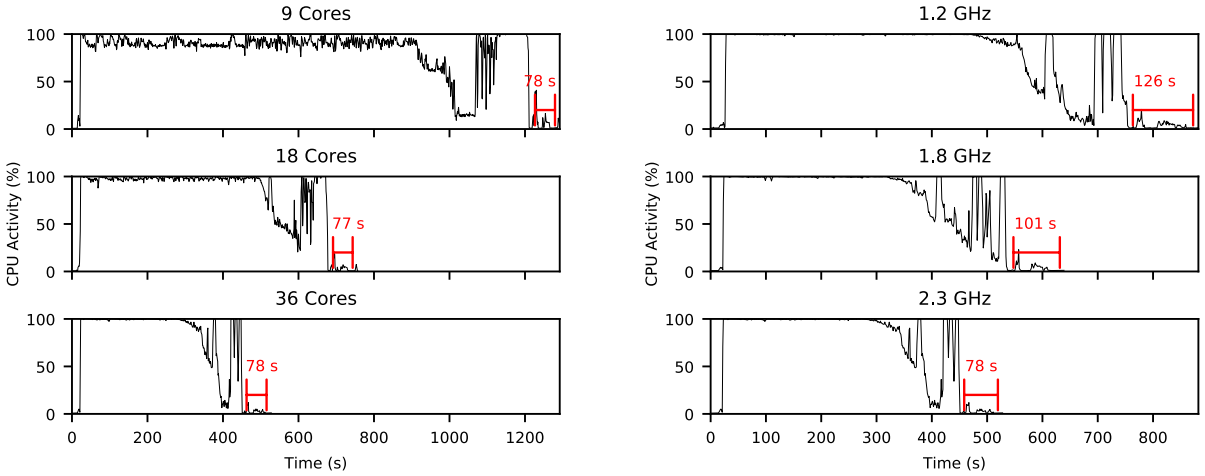
In contrast to Figure 4a, Figure 4b shows frequency scaling efficiency. Unlike core scaling, increasing the frequency always improves performance; additionally, *frequency scaling*



(a) The efficiency of core scaling. The efficiency is computed relative to 9 cores/18 threads at 2.3 GHz. As the number of cores increases, the relative benefit decreases, which is indicative of an insufficiency of TLP.

(b) The efficiency of frequency scaling. The efficiency is computed relative to a base frequency of 1.2 GHz using 18 cores/36 threads. The benefit of the higher frequencies is better than the benefit of more cores.

Figure 4: Core and frequency scaling efficiency on a parameter-tuned 8+1 cluster.



(a) Core scaling. Tail length is largely unaffected by core count.

(b) Frequency scaling. Tail length is very sensitive to frequency.

Figure 5: Core and frequency scaling on `Q06`. The tail is largely unaffected by the core count but is very sensitive to frequency.

is more efficient at improving performance. Even when core scaling efficiency drops below 50% (and even into negative numbers), frequency scaling efficiency remains above 55%.

Consider query `Q17`: going from 9 to 18 cores, it has an efficiency near 20%, meaning that doubling the number of cores only reduces the runtime by 10% (20% of the expected 0.5× run time). Increasing the frequency from 1.20 to 1.80 GHz, though, is nearly 98% efficient. Thus, the 1.5× increase in frequency yields a 1.5× (33%) decrease in run time.

Therefore, we conclude that *big data, which is, by design, supposed to be massively parallel and scalable, still demands high single-thread performance from modern processors.*

6.4 Low TLP Causes

There are various explanations for the TLP problem. `Q10` has one of the lowest TLPs of all the queries, evidenced by both Table 6 and Figure 1a. When we examine the per-core activity of `Q10` (not shown), we find that only a handful of cores are active, maintaining near-100% activity, while the remainder of the cores are essentially halted. In contrast, a per-core breakdown of most queries shows all cores doing effectively the same thing, with core activity plots very similar to the full CPU activity plot (e.g. Figure 1c). In other words, `Q10` naturally lacks the parallelism to utilize all available cores. `Q10` performs Natural Language Processing (NLP) [13], similar to `Q10` , `Q18` , `Q19` and `Q27`, all of which also suffer from low TLP (Table 6).

In other queries, the work may be fairly well spread across cores and yet still show low TLP. We can understand the limited TLP in these queries by looking at `Q06` as a case study (Figure 5). In Figure 5a, we vary the core count from 9 (18 threads) to 18 (36 threads) and 36 (72 threads), plotting the TLP timelines. As the figure shows, the first phase of the application scales well with the increasing core count. The final phase, however—the "tail" region—is nearly unchanged by the core count. Most of the queries exhibit these tail regions, where a relatively small amount of work has to be completed. These tails are not limited to query termination, though, as most queries are composed of multiple map and reduce phases, each of which can exhibit a tail.

In contrast, consider Figure 5b, which shows the effects of frequency scaling on `Q06`. Moving from 1.2 GHz to 1.8 GHz and 2.3 GHz, not only does the first phase benefit (as it did with additional cores), but the tail also shrinks from 126 to 78 seconds. Thus, even though `Q06` is largely TLP-rich, it has one distinct phase with almost no thread-level parallelism.

An additional likely TLP limiter is resource contention. Periodic dips in TLP (Figure 1) tended to correlate with some

I/O related system calls in our experiments (not shown).

7. WORKING WITH AMDAHL'S LAW

Our findings on the lack of TLP suggest that warehouse-scale computing requires a more nuanced approach than simply pursuing scale-out computing. Scale-out resources offer no benefit when queries encounter TLP-limited regions of execution. Indeed, as dictated by Amdahl's law, the speedup of these queries will be increasingly dominated by single-thread performance as the highly parallel regions grow ever faster.

We propose that proactive measures can and should be taken to identify TLP-limited periods of execution and to actively boost single-thread performance. In this section we explore Intel's Turbo Boost technology [12] as one target for proactive management. Though Turbo Boost already works to take advantage of slack in thermal and power margins, we show in Section 7.1 that the software architecture and Turbo Boost hardware constraints combine to undermine its effectiveness. Therefore, in Section 7.2 we propose an architectural enhancement to Turbo Boost that can work in concert with software to maximize Turbo Boost efficacy. We evaluate this proposal for big data analytics in Section 7.3.

7.1 Turbo Boost Limits in the Hadoop System

Turbo Boost is an Intel technology that utilizes slack in the CPU's power and thermal margins to drive operating frequency above its nominal maximum value [12]. Turbo Boost is limited, however, by the number of active cores on a CPU. To achieve the maximum Turbo Boost frequency in large core count systems, nearly all the cores have to be disabled. The Xeon E5-2699 v4 (the successor to the E5-2699 v3 that we used), for example, nominally runs at 2.20 GHz and has a maximum Turbo Boost frequency of 3.60 GHz [24]. With all its cores enabled, however, the E5-2699 v4 has a much lower Turbo Boost ceiling of 2.80 GHz [25]. It can only achieve the 3.60 GHz frequency with all but two cores disabled.

Hardware Limits As we show in Figure 6, most Turbo Boost (i.e. frequency gains) for server-class systems cannot be realized unless most of the cores have been disabled. With all 22 cores active, the E5-2699 v4 Turbo Boost ceiling is capped at 2.80 GHz, a far cry from 3.60 GHz. Only once 14 of the 22 cores are disabled does the Turbo Boost ceiling begin to rise beyond 2.80 GHz, and only when 20 cores have been disabled does the ceiling finally reach 3.60 GHz. With every disabled core, additional power and thermal slack is created, yet the Turbo Boost ceiling usually remains unaffected.

Software Limits Within the Hadoop ecosystem, Turbo Boost is severely handicapped: in combination with kernel scheduling policies, the multitudinous threads launched by Java and Hadoop's suite of tools are spread out to as many cores as possible. Unfortunately, the threads' activity is too limited to justify the use of additional cores. Even a severely TLP-constrained query (e.g. `Q10`)—which ought to enjoy a very high Turbo Boost ceiling—runs barely over 2.80 GHz.

Undermining Performance These hardware and software limits together work to constrain the performance of data analytics applications. Figure 7 shows for `Q10` the number of cores with one or more Linux threads that are "Running" [26] at any given time as well as the computing time consumed by *all* threads. During query execution, all the

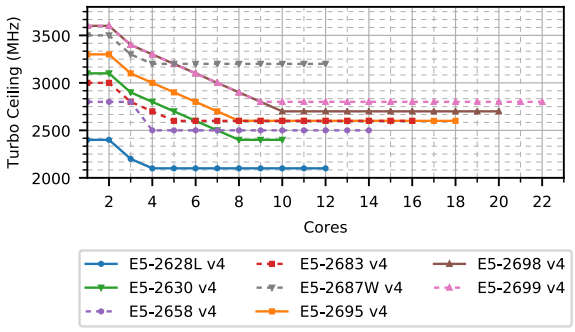


Figure 6: Turbo Boost ceiling as a function of core count. As CPU core counts have increased, the available Turbo Boost ceiling has become relatively constrained. Only when most cores are disabled can Turbo Boost become most effective.

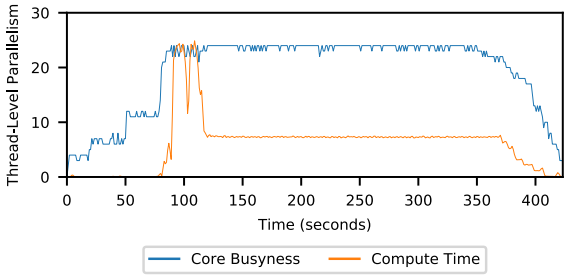


Figure 7: CPU busyness and total compute time. Busy cores are those with at least one Linux thread in the "Running" state. Computation time is a measure of how much time is spent on the cores divided by the wall clock time; thus, it is a measure of how many cores are needed. It is equivalent to TLP.

cores are occupied by running threads, which limits Turbo Boost's ceiling. Simultaneously, though, the computing time falls far short of the capacity of the busy cores. *Thus we see that the operating system and Hadoop ecosystem (with its many threads) work together to undermine the ability of Turbo Boost to accelerate TLP-limited regions.*

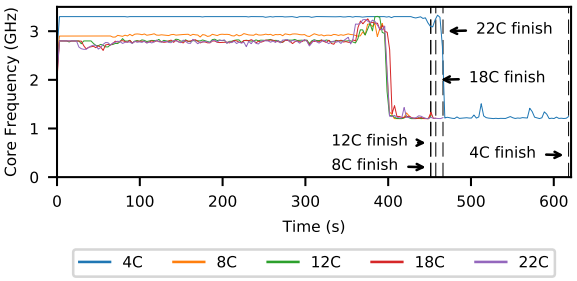
7.2 Core Packing (Proactive Turbo Boosting)

We propose **core packing** to enable *proactive* Turbo Boost management: the profusion of running threads currently spread across many cores should be collapsed onto the minimum set of cores needed to accommodate them. This change would serve to proactively facilitate Turbo Boost.

By forcing the threads onto fewer cores (by explicitly disabling cores), we can induce a higher frequency and shorter running time. We use `Q10` as a case study, due to its particularly low TLP. For the following experiments, we use a single-socket Broadwell-based (instead of Haswell-based) 3+1 cluster of Xeon E5-2699 v4 CPUs. We use the Broadwell CPUs because of their higher core count (22 instead of 18).[1]

In Figure 8, we show the operating frequencies and running

[1]In executing `Q10`, the Haswell CPUs achieve an average operating frequency slightly above 2.80 GHz. With the four extra cores in the Broadwell CPUs, the higher Turbo Boost ceiling is out of reach, and these CPUs almost never rise above 2.80 GHz.



(a) Effect of scaling the number of cores on Q10.

Q10	Time	Energy	Rel. Time	Rel. Energy
4 Cores	632 s	24.6 kJ	1.308×	1.028×
8 Cores	466 s	22.2 kJ	0.965×	0.929×
12 Cores	467 s	23.3 kJ	0.967×	0.976×
18 Cores	473 s	23.8 kJ	0.979×	0.994×
22 Cores	483 s	23.9 kJ	1.000×	1.000×

(b) Benefits when running with fewer cores.

Figure 8: Attainable benefits when TLP is low.

times of Q10 with various core counts. The 22-core line (Figure 8a), the baseline machine configuration, shows that Q10 executes at 2.80 GHz for most of its operation and finishes in 483 seconds (Figure 8b). Recall from Table 6 and Figure 1a that Q10 is very TLP-deficient. Therefore, as we decrease the core count (using the operating system to explicitly disable cores) we see the query finish more quickly. For the 12- and 18-core runs, the operating frequency remains at 2.80 GHz, so the speedup is likely attributable to the decreased overhead of managing so many threads. When the core count drops to 8, the operating frequency increases to 2.90 GHz, and at 4 cores, it jumps to 3.30 GHz. While the 4-core configuration severely degrades the running time (632 seconds—a 31% slowdown), the 8-core configuration completes in only 466 seconds—a 3.5% speedup. Q10 has a high-TLP region at the beginning of its execution, which offsets the higher frequency of the 4-core configuration; the 8-core configuration is able to meet the demands of the high-TLP operations while still providing a frequency boost. We also note that the 8-core configuration reduces total energy consumption by 7.1%.

Thus, if Turbo Boost can be carefully orchestrated (i.e. controlled and efficiently executed), there is tremendous opportunity for increased performance and energy savings. To this end, we propose a modified architecture combined with software support to facilitate core packing for proactive Turbo Boosting without sacrificing scale-out performance.

Architecture Support The Turbo Boost ceiling of current many-core server architectures, like the Xeon E5-2699 v4, does not increase until most of the cores are halted. We propose a smoother transition from the base to the maximum Turbo Boost ceiling: allow the ceiling to increase at smaller increments for every core that is halted. We believe this design change should be achievable, as we do not propose that the maximum frequency or the starting frequency be changed; rather, we propose that the steps between frequencies be smaller but uniformly applied for every disabled core.

Table 7: Estimated application speedups on a Xeon E5-2699 v3 based on our analytical model. Because no current hardware implements our proposed changes, it is not possible to measure actual speedups.

Query	Speedup	Query	Speedup	Query	Speedup
Q01	15.8%	Q11	9.2%	Q21	13.2%
Q02	13.1%	Q12	14.7%	Q22	13.9%
Q03	11.1%	Q13	8.0%	Q23	7.7%
Q04	11.3%	Q14	7.4%	Q24	9.6%
Q05	7.2%	Q15	14.1%	Q25	8.6%
Q06	6.2%	Q16	11.5%	Q26	16.9%
Q07	19.7%	Q17	19.2%	Q27	14.5%
Q08	14.4%	Q18	15.2%	Q28	13.7%
Q09	4.7%	Q19	18.9%	Q29	6.6%
Q10	15.1%	Q20	10.3%	Q30	10.1%

Software Support Using hardware counters, the operating system or even runtime environment should actively monitor the TLP. Whenever it dips low enough that a core could be safely disabled, the operating system should proactively halt that core until the TLP again rises. Current scheduling policies try to distribute threads onto as many cores as are available; we have seen the folly in this approach, as it diminishes Turbo Boost's effectiveness.

7.3 Core Packing Analysis

To evaluate the benefits of core packing, we show in Figure 9 just how large the *measured* benefits could be. Using all 22 cores during the high-TLP region at the beginning of execution (Figure 9a) and then dropping down to 4 cores when TLP is severely limited (Figure 9b), from the time spent in the high- and low-TLP regions we estimate that we could achieve a 13.5% speedup and 30% energy reduction.

Q10's simplicity allows us to simply and effectively mimic and measure core packing behavior on a real system. Because of its simplicity, we were able to run Q10 with two different core configurations that mimic the behavior of core packing. But we have to use a model to evaluate the other, more complex queries. We base this model on the assumption that the two supporting mechanisms we recommended earlier can be implemented in a real system. If we assume (1) that each query can execute at 3.60 GHz when only one core is active and (2) that the Turbo Boost ceiling is a linear function of the number of active cores, then we can estimate the operating frequency (the average Turbo Boosted frequency):

$$freq_{TB} = freq_0 + (1 - TLP) \cdot (freq_{Max} - freq_0) \qquad (1)$$

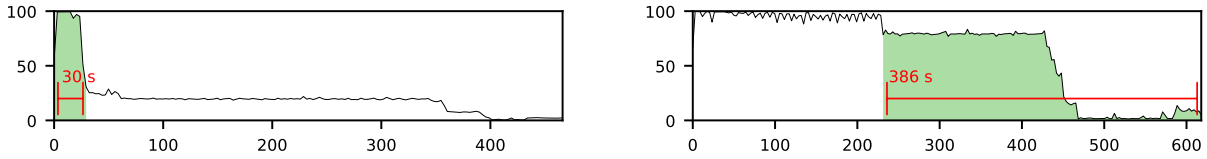
Here, $freq_0$ is the base average frequency of the application (measured at close to 2.80 GHz for each query) on the Haswell-based Xeon E5-2699 v3 and $freq_{Max}$ is the maximum Turbo Boost frequency (3.60 GHz).

We estimate the performance improvement as follows:

$$speedup = \frac{freq_{TB} - freq_0}{freq_0} \cdot efficiency_{freq} \qquad (2)$$

Here, $efficiency_{freq}$ is the efficiency of frequency scaling, which we reported earlier (Section 6.3).

Table 7 shows our estimated performance gains using Equations 1 and 2. Our model predicts a 15.1% speedup for Q10, which is very close to the speedup we estimated from execution profiles on varying core counts. Based on this model,



(a) Q10 CPU activity with 22 cores. In the shaded region, the use of all 22 cores is ideal due to the high TLP.

(b) Q10 CPU activity with 4 cores. In the shaded region, the use of only 4 cores is ideal due to the low TLP.

Figure 9: Core scaling in Q10. The TLP varies over the course of the application, so the number of cores should ideally be adjusted as well. We show only two configurations here, but a deployment-ready solution would vary the core count even more carefully. We use the shaded regions in computing the attainable speedup in a deployment-ready solution.

we estimate that the queries could gain a minimum of 4.7% speedup (Q09) and a maximum of 19.7% speedup (Q07).

In summary, core packing would actively monitor instantaneous TLP and adapt the system's performance accordingly. Core packing would proactively force the shutdown of lightly active cores, to consolidate the work onto fewer more active cores, and thereby yield substantial performance improvements while requiring *zero* changes to the application code.

8. RELATED WORK

Though previous work has researched the execution characteristics of big data applications, that work has been based on benchmarks that rely on micro-benchmarks [1, 3, 5, 7, 8]. Our work studies BigBench, which is unique in thoroughly exploring the subfield of big data analytics and relying solely on industry-backed, fully realized applications. In this section, we compare our work to that based on other benchmarks.

Architectural Studies Introducing BigDataBench, Wang et al. concluded that data caches perform well for big data applications [1], in agreement with the DCBench paper by Jia et al. [5] and the memory characterization from Dimitrov et al. [4]. Lotfi-Kamran et al. suggested that the LLC is excessive and that its area should be partially reclaimed for additional cores [27]. We find that data analytics cache behavior is more nuanced than past papers have revealed: the merit of the L2 cache varies between workloads and the MPKIs at all cache levels is very dependent upon the application. Ferdman et al. [3] and Yasin et al. [20] demonstrated that cache access latency causes back end stalling, which we confirm.

Meanwhile, Kanev et al. found instruction footprint in the caches to be one of the major contributors to stalls, reporting front end stalls in 15-30% of cycles [21]. This contrasts our work, where we have found front end stalls impacting only 12-16% of cycles. We conjecture that this is due to Kanev's work including applications beyond data analytics.

System-Level Studies Kanev et al. profiled a Google warehouse-scale computer, looking at both microarchitecture and software trends [21]. They identified a "datacenter tax" consisting of common subroutine overheads for which they suggest specialized accelerators. We concur that single-thread acceleration is necessary, though we propose a more general solution in augmenting Turbo Boost through core packing. Cochran et al. proposed setting thread affinities to control power consumption [28]. We use a comparable technique to achieve both energy and performance improvement.

Prior work has noted the low CPU utilization in big data workloads [3, 6, 13, 21, 29, 30, 31, 32] and has attributed it to resource contention, long disk latency, lack of memory-level parallelism, etc. Our work confirms that CPUs are being under-utilized, but we treat this as a parallelism issue and seek to improve single-thread performance.

Scale-Up Performance Appuswamy et al. recognized that scale-up is often more critical than scale-out performance [30]. We show that even in scale-out scenarios, scale-up performance is critical. Frequency boosting has previously been proposed to mitigate the bottlenecks of Amdahl's law and to balance power efficiency [33, 34, 35]. We propose a modified architecture where more incremental frequency improvements can be utilized by an active software system.

9. CONCLUSION

BigBench is a realistic big data analytics benchmark and a deep study of its behavior reveals that architectural characteristics and bottlenecks vary among queries. By far, though, the worst bottleneck is the startlingly low thread-level parallelism present in all applications, including even (in some environments) the throughput test. Particularly in the power test of BigBench, this low TLP wastes so many cycles as to be equivalent to leaving multiple cores unused at all times.

The great diversity of big data analytics demands that any future study consider a wide variety of applications. Any analysis or solution that derives from too narrow a field of applications will likely have questionable applicability. Conversely, the diversity also indicates that there is not likely to be a panacea to address all performance issues. As in any mature field, innovative solutions will be those that consider the multitudinous applications and adapt to their idiosyncrasies.

BigBench's TLP paucity violates commonly held beliefs regarding big data, namely that it is arbitrarily parallelizable and that scale-out is the only needed solution. Amdahl's law dictates that as parallelizable regions become faster, the single-threaded (or thread-limited) regions will increasingly dominate compute time. Without minimizing the importance of scale-out computing, this work has demonstrated that scale-up improvements continue to be eminently relevant. Future work will need to focus on the single-threaded bottlenecks that afflict big data or risk leaving untapped a major opportunity for performance improvement.

Acknowledgements

We thank Bhaskar Gowda from Intel for his help with TPCx-BB. We are also grateful to the reviewers for their constructive

feedback. The majority of the work was done while Daniel Richins and Tahrina Ahmed were at Intel and Vijay Janapa Reddi was at The University of Texas at Austin. In addition, the work was sponsored by various sources of funding from Intel. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of their industry affiliations.

APPENDIX

A BigBench score is computed from the load (T_{LD}), throughput (T_{TT}), and power test (T_{PT}) throughputs:

$$T_{LD} = 0.1 \cdot T_{Load}$$

$$T_{PT} = M \cdot \sqrt[M]{\prod_{i=1}^{i=M} Q(i)}$$

$$T_{TT} = \frac{1}{n} \cdot T_{T_{put}}$$

where T_{Load}, $Q(i)$, and $T_{T_{put}}$ are the running times of the load test, query i in the power test, and the throughput test, respectively, in seconds. M is the number of queries (30) and n is the number of streams in the throughput test. T_{LD} is multiplied by 0.1 to reduce its weight in the final score.

The final score (at scale factor SF) is computed as

$$BBQpm@SF = \frac{SF \cdot 60 \cdot M}{T_{LD} \cdot \sqrt{T_{PT} \cdot T_{TT}}}$$

where SF represents the scale factor and the factor of 60 converts from second to minutes.

A. REFERENCES

[1] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, *et al.*, "BigDataBench: a Big Data Benchmark Suite from Internet Services," in *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 488–499, IEEE, 2014.

[2] R. Zhou, M. Liu, and T. Li, "Characterizing the Efficiency of Data Deduplication for Big Data Storage Management," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 98–108, IEEE, 2013.

[3] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pp. 37–48, ACM, 2012.

[4] M. Dimitrov, K. Kumar, P. Lu, V. Viswanathan, and T. Willhalm, "Memory System Characterization of Big Data Workloads," in *Big Data, 2013 IEEE International Conference on*, pp. 15–22, Oct 2013.

[5] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo, "Characterizing Data Analysis Workloads in Data Centers," in *IEEE International Symposium on Workload Characterization (IISWC)*, Sept 2013.

[6] Z. Ren, X. Xu, J. Wan, W. Shi, and M. Zhou, "Workload Characterization on a Production Hadoop Cluster: A Case Study on Taobao," in *Workload Characterization (IISWC), 2012 IEEE International Symposium on*, pp. 3–13, IEEE, 2012.

[7] S. Huang, J. Huang, Y. Liu, L. Yi, and J. Dai, "Hibench: A Representative and Comprehensive Hadoop Benchmark Suite," in *ICDE Workshop*, 2010.

[8] AMPLab, "Big Data Benchmark." `https://amplab.cs.berkeley.edu/benchmark/`.

[9] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan, "LinkBench: A Database Benchmark Based on the Facebook Social Graph," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, (New York, NY, USA), ACM, 2013.

[10] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pp. 143–154, ACM, 2010.

[11] Transaction Processing Performance Council, "TPC Express Big Bench TPCx-BB Standard Specification Version 1.1.0." `https://www.tpc.org/tpc_documents_current_versions/pdf/tpcx-bb_v1.1.0.pdf`.

[12] Intel Corp., "Intel® Turbo Boost Technology 2.0." `http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html`.

[13] C. Baru, M. Bhandarkar, C. Curino, M. Danisch, M. Frank, B. Gowda, H.-A. Jacobsen, H. Jie, D. Kumar, R. Nambiar, M. Poess, F. Raab, T. Rabl, N. Ravi, K. Sachs, S. Sen, L. Yi, and C. Youn, *Discussion of BigBench: A Proposed Industry Standard Performance Benchmark for Big Data*, pp. 44–63. 2014.

[14] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen, "BigBench: Towards an Industry Standard Benchmark for Big Data Analytics," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp. 1197–1208, ACM, 2013.

[15] "GitHub - intel-hadoop/Big-Data-Benchmark-for-Big-Bench: Big Bench Workload Development." `https://github.com/intel-hadoop/Big-Data-Benchmark-for-Big-Bench`.

[16] "Apache Hive." `https://hive.apache.org`.

[17] "Apache Spark - Lightning-Fast Cluster Computing." `https://spark.apache.org`.

[18] Intel Corp., "Intel® EMON User's Guide (PDF)." `https://software.intel.com/en-us/emon-user-guide`.

[19] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, 2008.

[20] A. Yasin, Y. Ben-Asher, and A. Mendelson, "Deep-dive Analysis of the Data Analytics Workload in CloudSuite," in *IEEE International Symposium on Workload Characterization (IISWC)*, pp. 202–211, Oct 2014.

[21] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a Warehouse-scale Computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ISCA '15, ACM, 2015.

[22] R. Bro and A. K. Smilde, "Principal component analysis," *Analytical Methods*, vol. 6, no. 9, 2014.

[23] "Amazon Web Services (AWS) - Cloud Computing Services." `https://aws.amazon.com/`.

[24] Intel Corp., "Intel® Xeon® Processor E5-2699 v4 (55M Cache, 2.20 GHz) Product Specifications." `http://ark.intel.com/products/91317/Intel-Xeon-Processor-E5-2699-v4-55M-Cache-2_20-GHz`.

[25] WikiChip, "Xeon E5-2699 v4 - Intel - WikiChip." `https://en.wikichip.org/wiki/intel/xeon_e5/e5-2699_v4`.

[26] "proc(5) - Linux manual page." `http://man7.org/linux/man-pages/man5/proc.5.html`.

[27] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, *et al.*, "Scale-Out Processors," in *ACM SIGARCH Computer Architecture News*, vol. 40, pp. 500–511, IEEE Computer Society, 2012.

[28] Cochran, Ryan and Hankendi, Can and Coskun, Ayse K and Reda, Sherief, "Pack & Cap: adaptive DVFS and thread packing under power caps," in *Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*, ACM, 2011.

[29] Z. Jia, R. Zhou, C. Zhu, L. Wang, W. Gao, Y. Shi, J. Zhan, and L. Zhang, *The Implications of Diverse Applications and Scalable Data Sets in Benchmarking Big Data Systems*, pp. 44–59. Springer, 2014.

[30] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron, "Scale-up vs Scale-out for Hadoop: Time to rethink?," in

Proceedings of the Annual Symposium on Cloud Computing, SOCC, 2013.

[31] T. Ivanov and M.-G. Beer, "Evaluating Hive and Spark SQL with BigBench," *CoRR*, vol. abs/1512.08417, 2015.

[32] E. Anderson and J. Tucek, "Efficiency Matters!," *SIGOPS Oper. Syst. Rev.*, vol. 44, Mar. 2010.

[33] M. Annavaram, E. Grochowski, and J. Shen, "Mitigating Amdahl's Law Through EPI Throttling," in *32nd International Symposium on Computer Architecture (ISCA'05)*, IEEE, 2005.

[34] E. Grochowski, R. Ronen, J. Shen, and H. Wang, "Best of Both Latency and Throughput," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.*, IEEE, 2004.

[35] J. Li and J. F. Martinez, "Power-Performance Implications of Thread-level Parallelism on Chip Multiprocessors," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, IEEE, 2005.