

# BigDataBench: a Big Data Benchmark Suite from Web Search Engines

Wanling Gao<sup>1</sup>, Yuqing Zhu<sup>1</sup>, Zhen Jia<sup>1</sup>, Chunjie Luo<sup>1</sup>, Lei Wang<sup>1</sup>, Zhiguo Li<sup>2</sup>, Jianfeng Zhan<sup>\*1</sup>, Yong Qi<sup>2</sup>, Yongqiang He<sup>3</sup>, Shiming Gong<sup>4</sup>, Xiaona Li<sup>5</sup>, Shujie Zhang<sup>6</sup>, and Bizhu Qiu<sup>7</sup>

<sup>1</sup>Institute of Computing Technology, Chinese Academy of Sciences,  
{gaowanling, zhuyuqing, jiazhen, luochunjie, wanglei2011, zhanjianfeng}@ict.ac.cn

<sup>2</sup>Xi'an Jiaotong University

<sup>3</sup>Facebook, heyongqiang@fb.com

<sup>4</sup>SouGou, yakergong@gmail.com

<sup>5</sup>Baidu, lixiaona@baidu.com

<sup>6</sup>Huawei, shujie.zhang@huawei.com

<sup>7</sup>Yahoo!, qiubz@yahoo-inc.com

## Abstract

*This paper presents our joint research efforts on big data benchmarking with several industrial partners. Considering the complexity, diversity, workload churns, and rapid evolution of big data systems, we take an incremental approach in big data benchmarking. For the first step, we pay attention to search engines, which are the most important domain in Internet services in terms of the number of page views and daily visitors. However, search engine service providers treat data, applications, and web access logs as business confidentiality, which prevents us from building benchmarks. To overcome those difficulties, with several industry partners, we widely investigated the open source solutions in search engines, and obtained the permission of using anonymous Web access logs. Moreover, with two years' great efforts, we created a semantic search engine named ProfSearch (available from <http://prof.ict.ac.cn>). These efforts pave the path for our big data benchmark suite from search engines—BigDataBench, which is released on the web page (<http://prof.ict.ac.cn/BigDataBench>).*

*We report our detailed analysis of search engine workloads, and present our benchmarking methodology. An innovative data generation methodology and tool are proposed to generate scalable volumes of big data from a small seed of real data, preserving semantics and locality of data.*

*Also, we preliminarily report two case studies using BigDataBench for both system and architecture researches.*

## 1 Introduction

In recent years, more and more data are produced. The roles of people change from passive receptors of information to active creators. People are producing and sharing data continuously. It is reported that 2.5 quintillion bytes of data are created everyday [1]. According to the survey of the global output by IDC, the data are growing exponentially now and the trends will be maintained in the coming years. Big Data are considered as the asset of companies, organizations and even countries. Extracting the big value from Big Data requires enabling big data systems.

As researchers in both academia and industry pay great attention to innovative big data systems and architecture [10] [28] [17] [9] [21], the pressure of evaluating and comparing performance, energy efficiency, and cost-effectiveness of these systems rises [27] [25]. Big data benchmarks are the cornerstone of those efforts.

In a tutorial given at HPCA 2013 [18] [16], we state our position view in big data benchmarking: we should take an incremental and iterative approach in stead of a top-down approach because of the following four reasons: first, there are many classes of big applications with a lack of a scientific classification. Second, even for Internet service workloads, there are many important application domains, e.g.,

<sup>\*</sup>The corresponding author is Jianfeng Zhan.

search engines, social networks, and electronic commerce, though they are mature in terms of both business and technology, however, customers, vendors, or researchers from academia or even different domains of industry do not know enough to make a big data benchmark suite because of the confidential issues. Third, the value of big data drives the emergence of innovative application domains. Fourth, the complexity, diversity, workload churns, and rapid evolution of big data systems indicate that both customers and vendors often have incorrect or outdated assumptions about workload behavior [12].

This paper presents our joint research efforts on big data benchmarking with several industrial partners. After investigating different application domains of Internet services—an important class of big data applications, we pay attention to search engines, which are the most important domain in Internet services in terms of the number of page views and daily visitors. (Social networks, and electronic commerce follow, respectively.) We widely investigated the open source solutions in search engines. Moreover, with two years’ great efforts, we created a semantic search engine named *ProfSearch* (available from <http://prof.ict.ac.cn>). These efforts pave the path for our big data benchmark suite from search engines—*BigDataBench*.

We report our detailed analysis of search engine workloads, and present our benchmarking methodology. An innovative data generation methodology and tool are proposed to generate scalable volumes of big data from a small seed of real data, preserving semantics and locality of data. Also, we preliminarily report two case studies using *BigDataBench* for both system and architecture researches. We gained two insights from the observations: first, the peak data processing rates of big data systems are both applications and data volumes dependent, and hence tuning peak performance must consider different application scenarios. Second, some architectural events, e. g., cache and TLB behaviors, are tending towards stability only on condition that the data volume increases to a certain extent. This observation has a significant implication for simulation-based architecture researches since large-scale simulation is time-consuming.

The rest of the paper is organized as follows. Section 2 presents the detailed analysis of search engine workloads. Section 3 summarizes our benchmarking methodology and its result—*BigDataBench*. Section 4 introduces how to generate scalable volumes of data. In Section 5, we report two case studies using *BigDataBench* for both system and architecture researches. Finally, we draw the conclusion in Section 6.

## 2 Workload analysis of search engines

Our big data benchmarking work is based on two practises: investigation of the open source solutions in common search engines, backed by several industry partners, and our experience of building a semantic search engine named *ProfSearch*.

*ProfSearch* is a Chinese semantics search engine used to search scientists or professionals from different disciplines. Now it has collected publicly available information of 251,564 researchers across 260 universities and institutes. Different from common web search engines, for a scholar, *ProfSearch* can extract different categories of information from many data sources, for example, his/her research interests and educational background.

We have used various state-of-art algorithms or techniques to access, analyze, store big data and offer search services as follows [19]:

*Crawler workloads.* We use Scrapy [7], a widely used open source web scraping framework written in Python, to build our web crawler.

*Analysis workloads.* We have used various machine learning and data mining techniques to classify and cluster web pages downloaded by *Crawler*, extract structured data from unstructured web pages, analyze the semantics and topics of these pages. These analysis workloads contain state-of-art algorithms and learning models, such as Support Vector Machine (SVM) [14], Naive Bayes, K-means, Hidden Markov Model (HMM) [23], Conditional Random Fields (CRFs) [20], Latent Semantic Analysis (LSA) [15] and Latent Dirichlet Allocation (LDA) [11].

SVM and Naive Bayes are widely used classification algorithms in data mining. We use SVM to classify whether a web page is a home page or not, and Naive Bayes to classify professionals to a certain category according to his/her research interests. Different from classification which is a supervised learning process, clustering is unsupervised without manual annotation. We use *K-means* to cluster papers to find the similarity among scientists’ papers.

HMM and CRFs are statistical sequence modeling method, often applied in speech, handwriting, gesture recognition, part-of-speech tagging, and bioinformatics. Whereas an ordinary classifier predicts a label for a single sample without regard to “neighboring” samples, these models can take context into account. We use HMM for Chinese segmentation which divides the text of Chinese into meaningful tokens and part-of-speech tagging, and CRFs for information extraction which automatically extracts structured data from web pages.

LSA uses a mathematical technique called singular value decomposition (SVD) to identify patterns in the relationships between the terms and concepts contained in an unstructured collection of text. It uncovers the underlying latent semantic structure of word usage in a body of text. LDA

is an example of a topic model. It is a generative model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. We have used LSA and LDA to analyze what research fields are similar to other's interests and who has similar interests with the others.

*Store and management workloads.* In ProfSearch, there is large scale of unstructured, semi-structured and structured data to be stored. The original unstructured web pages are stored using HDFS, the distributed file system in Hadoop [4]. Large intermediate data generated in the analysis process are semi-structured, and we use Hive [5] to store them for further analysis. The structured data extracted from the web are stored in MySQL.

*Web service workloads.* To offer web search service, we use Sphinx [8], a widely used free software, to index documents and provide results for query. Sphinx implements inverted index. It is an index data structure storing a mapping from content to its locations in a database file, or in a document or a set of documents. Sphinx uses slightly modified BM25 function to rank matching documents according to their relevance to a given search query. BM25 is based on the probabilistic retrieval framework and represents state-of-art TF-IDF-like retrieval functions. Additionally, we use Apache Tomcat as the web server, and Memcached [6], a distributed memory object caching system, to speed up the search service.

### 3 Benchmarking Methodology and Decisions

This section presents our methodology and decisions on assembling *BigDataBench*. We state how we choose the representative applications and generate scalable volumes of data preserving semantics and locality of data. And we also give a summary of *BigDataBench*.

#### 3.1 Following An Incremental Approach

For the first step, we investigate application domains. We single out the most important application domain. Firstly, we pay attention to Internet services, and rank main application domains according to a widely acceptable metric—the number of page views and daily visitors. We investigate the top sites listed in Alexa [3], of which the rank of the sites is calculated using a combination of average daily visitors and page views. We find the search engine is the mostly popular application domain, 40% of the top sites are search engines, such as Google, Bing and etc. Data from the Internet study [2] also prove the popularity of search engines, which shows that 92% of online adults use search engines to find information on the web. So we focus on the search engine firstly.

For the second step, we choose typical workloads from Web search engines as candidates of our *BigDataBench*.

Considering the workload churns and emerging workloads domains, we believe that the mature big data bench-

mark suites will take a long way to go. We will continuously add more representative applications and remove the out-of-data applications.

#### 3.2 Methodology of Generating Big Data

Big data benchmarks require big data set as the inputs to drive their workloads. Chen et al.[12] found that the observed workload behaviors, shown in their collected large-scale, long-term MapReduce workloads traces from Cloud-era and Facebook, do not fit well-known statistical distributions. Our previous work [26] collected three search engine companies' traces and also found that the frequently used distributions cannot capture the key characteristics of real data. Moreover, there is a large performance gap between running search engines with real data and randomly generated data [26]. In this case, only real data can reflect the real system behaviors, and hence the real life data is preferred [12] in big data benchmarks. However, it is a big challenge to obtain real big data as follows: Firstly, most of end user do not own real big data whereas Internet service companies who own the real life big data would not like to share big data for commercial confidentiality and user privacies; secondly, even though big data is openly available, downloading terabyte scale data is too costly to be acceptable.

Based on the above reasons, we would like to generate synthetic data preserving key characteristics of real data. There are two key characteristics we must consider semantic and locality. Semantic characteristic reflects the insightful meaning of real life data. Locality reflects the data access patterns. We investigate the real life data with the purpose of getting a semantic model and a locality model. We find the real search engine query terms follow zipf's law[26]. We then generate the synthetic query trace on the basis of the real search query terms we have gotten and let the query terms follow zipf's law. The temporal locality can be reflected by using reuse distance. We calculate each real term's reuse distance and generate the synthetic data according to the real terms' reuse distance. The details can be found at Section 4.

#### 3.3 Considering Variety of Workloads

In addition to three "V" of big data: volumes, velocity, and variety, our previous work [16] showed that diversity of workloads must be considered in big data benchmarking since they have different characteristics in term of computation, memory, and I/O access patterns. As we discussed in Section 2, a search engine involves in many important workloads, and we must choose typical workload for assembling *BigDataBench*.

#### 3.4 Summary of *BigDataBench*

After the analysis of the common search engines backed by several industry partners and our semantics search engine-ProfSearch, we choose the following workloads for *BigDataBench* at present: *Sort*, *Grep*, *WordCount*, *Naive*

*Bayes* and *SVM* for their representative algorithms and diversities of computing and I/O access patterns [13]. We also choose the search service including a back-end search server and a front-end Web server. Our future work will add more workloads.

## 4 Scalable Data Generation Tool

The data generation implementation of *BigDataBench* includes two parts. One is generating user requests, and the other is generating input data of *BigDataBench*.

### 4.1 Request Generation [26]

The key characteristics of a search workload trace are query sequences and timing sequences [26]. Query sequences depict the contents in each request. Query contents are determined by the semantic model that characterizes the frequencies of terms and the combinations of terms constituting a request. The timing sequences depict the issuing intervals of requests, from which we can compute the fluctuation of query requests. Meanwhile, measuring the temporal locality of a series of queries is similar with measuring the temporal locality of a series of memory accesses. Stack distance is an effective way to analyze the temporal locality of a series of requests. Stack distance, also called reuse distance, depicts the number of different queries between a recurring query. we have obtained permission to use three real workload traces, one from SoGou and the other two from two of the largest search service providers in China.

We generate synthetic query rate according to a real query trace. Supposing that we have an one day real-life trace, we divide the whole trace to several sections by one hour or one minute so that it can reflect the user’s behaviors more accurately. So we generate scalable volumes of request traces preserving timing, semantics, and locality model. The details of those models can be found at our previous work [26].

### 4.2 Input Data Generation

Our data generation tool is based on small-scale real data. We firstly analyze the characteristics of the small-scale data and then expand the data scale maintaining the characteristics. To preserve the characteristics of real-world data, we analyze the real data from semantic and locality.

We parse the real data and get class information, word information and the length of documents. Class information includes all the classes that occurred, the number of documents in each class, and the number of all the documents. Word information includes all the words that occurred, the number of a word occurred in each class, and the number of words in each class. Basing on those information, we can compute the probability distribution of classes, the probability distribution of words in each class, and the probability distribution of document length in each class. We ex-

pand the real data to big data by keeping those probability distributions unchanged.

## 5 Case Studies

In order to evaluate *BigDataBench*, we do some case studies in this section. First, the data generation is the most important part of Big Data Benchmarking, so we compare our generated data with real data to evaluate the validity of our generated data. Second, we present two case studies running *BigDataBench* for system and architecture researches respectively.

In the first experiment, we deploy a distribution of Hadoop [4] over a 2-node cluster (one master and one slave node). In the latter two experiments, we deploy a distribution of Hadoop over a 15-node cluster (one master and fourteen slave nodes). All the nodes in the clusters have the same configuration. Each node has two Xeon E5645 processors equipped with 16 GB memory and 8 TB disk. The detailed configuration for each node is listed in Table 2. The operating system is Centos 5.5 with Linux kernel 2.6.34. The Hadoop distribution is 1.0.2 with Java version of 1.6. The configuration of Hadoop is 18 map slots and 18 reduce slots for each slave node, and the Java opts for the task tracker child processes are 512 MB. In order to make the results credible, we repeat each experiment three times and report the average value.

CPU Type		Intel CPU Core	
Intel ®Xeon E5645		6 cores@2.40G	
L1 DCache	L1 ICache	L2 Cache	L3 Cache
6 × 32 KB	6 × 32 KB	6 × 256 KB	12MB

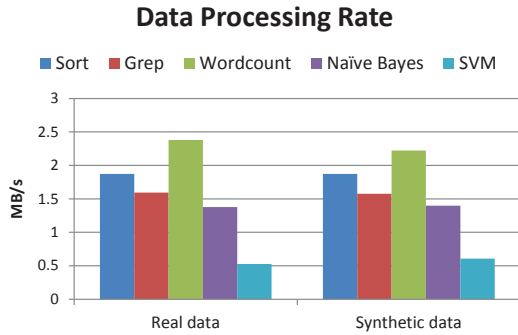
Table 1. Details of node configuration

### 5.1 Comparing Generated Data with Real Data

To evaluate the correctness of our methodology, we conduct an experiment to compare the *BigDataBench*’s generated data with the real data running five *BigDataBench* workloads. The real data is a 146-MB wikipedia data while the 146-MB synthetic data is generated by our tool using 7MB of the wikipedia as the seed. The selected five workloads are Sort application, Grep application, Word-count application, Naive Bayes application and SVM application from *BigDataBench*. We collect application-level and architectural-level monitor statistics when running the workloads. We use a user-perceived performance metric—data processing rate to evaluate the system processing capability [22] under different data set. For each application, the metric of data processing rate is defined as the input data size dividing by the application running time. For example, the running time of Sort with 100GB input is 2487 second, and then the data processed per second of Sort at 100GB

is 41.6MB/s. Architectural statistics are collected by using the *perf* tool. We investigate the following metric: L3 cache misses per 1000 instructions, L2 cache misses per 1000 instructions, L1 Instruction misses per 1000 instructions, L1 data misses per 1000 instructions, Instruction TLB (Translation Lookaside Buffer) misses per 1000 instructions and data TLB misses per 1000 instructions. We have not got the value of L1 data cache misses as it can be overlapped by out-of-order executions.

Figure 1, Figure 2 separately reports the data processing rates, cache and TLB behaviors of running two kinds of data with regard to different workloads. From Figure 1, we can see that for the synthetic and real data, the data processing rates of the workloads are close and the deviation of two data sets with the same workload is less than 12.9% (SVM), and the deviation of the two data sets are about 1% for Sort, Grep, Naive Bayes and 7% for Wordcount, which implied that the data processing behaviour is consistent between the generated data and the real data. From Figure 2, we can see that the cache and TLB behaviors for the real and synthetic data are close and the deviation of two data sets with the same workload is less than 2 instructions per 1000 instructions, which implied that the memory access behaviour is consistent between the generated data and the real data. So we can say that for the system or architecture research using the generated data of *BigDataBench* can achieve the similar workload behaviours with that of the real data.



**Figure 1. The Data Processing Rates of Real Data and Synthetic Data.**

## 5.2 Using *BigDataBench* for System Evaluation

In this section, we use *BigDataBench* to evaluate the performance of the cluster system mentioned above. As Rajaraman explained [24], for big data application, inferior algorithms beat better than sophisticated algorithms because of the computing overhead. We choose five applications that use simple algorithms from *BigDataBench*: *WordCount*, *Naive Bayes*, *Grep*, *SVM* and *Sort* whose computa-

tion complexities slightly vary from  $O(n)$  to  $O(n * \log_2 n)$ . Similar with the above case study, we still use data processing rate to evaluate the system processing capability.

Figure 3 reports the data processing rates for different data volumes with regard to different workloads. Please note that the same data is employed for all five workloads. Therefore, as shown in the figure, the system under test have a threshold with regards to data processing rate for *Grep*, *WordCount*, *Naive Bayes* and *SVM* workloads. The cluster system can only be fully loaded on condition that the data volume exceeds the threshold. From the figure, we can observe that the thresholds for the four workloads are between 100GB and 1TB. In comparison, *Sort* has global data access requirements. From Figure 3, we can find that there exist an inflexion point. After this point, the data processing rate decreases. Thus, there must exist a data size on which the data processing rate is optimal for the cluster system and the workload. This data size should fall between 10GB and 1TB. The reason for the existence of the inflexion point is that *Sort* requires the global transfer of the whole data set. Besides, data must be transferred for processing. Thus, after the inflexion point, the data processing waits for the data transfer, which can possibly saturate the I/O and the network. The larger the data volume, the more data to be transferred. Given the same bandwidth and I/O throughput, the longer time is needed for data transfer, thus the longer the data processing must wait for the data transfer. This finally results in the slope of the curve in Figure 3 for the sort workload.

Different data volume thresholds and inflexion point (for some workloads) are found in different workloads of close computation complexity. Meanwhile, the peak data processing rates have large performance gaps among different applications. These are the key characteristics of *BigDataBench*, indicating so-called peak performance are both application and data volume dependent.

## 5.3 Using *BigDataBench* for Architecture Research

We also collect the micro-architectural statistics to analyze the architectural metrics on experiments with different data scales. Architectural statistics are collected by using the *perf* tool. We analyze the micro-architecture from the perspective of cache access efficiency. There is a complex memory hierarchy for modern processor. So we investigate the following metric: L3 cache misses per 1000 instructions, L2 cache misses per 1000 instructions, L1 Instruction misses per 1000 instructions, instruction TLB (Translation Lookaside Buffer) misses per 1000 instructions and data TLB misses per 1000 instructions. We have not got the value of L1 data cache misses as it can be overlapped by out-of-order executions. Thus the chosen metrics basically cover the complete memory hierarchy of the processor.

## Comparison-Architecture

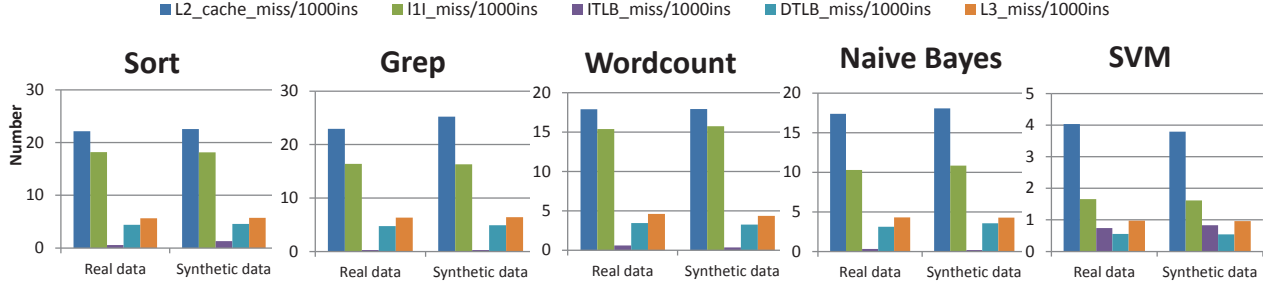


Figure 2. Cache and TLB Behaviors of Real Data and Synthetic Data.

## Architecture-Analysis

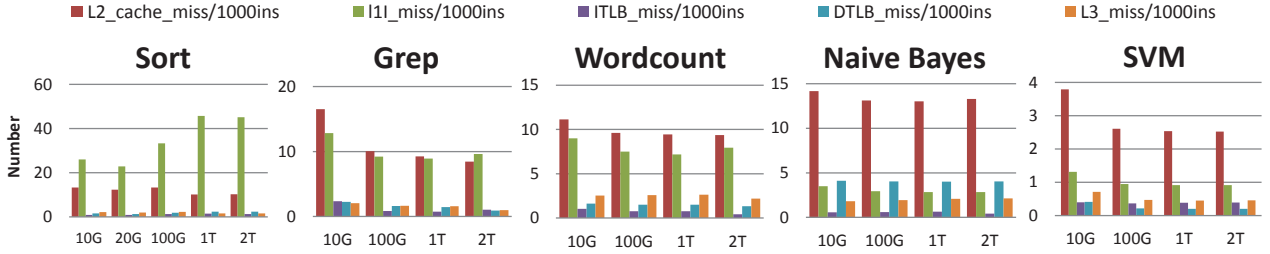


Figure 4. Cache and TLB Behaviors of Data Analysis Applications.

The cache and TLB behaviors of the system under five data analysis workloads from *BigDataBench* workloads are shown in Figure 4. As a comparison, we also benchmark a search server of search engine, which uses the distribution version *Nutch 1.1*. The search server receives user requests, searches in the index, and then sends the corresponding items to front end. In this benchmarking effort, we change the index size from 2 GB to 8 GB, and the corresponding segment size from 4.4 GB to 17.6 GB. The cache and TLB behaviors of the search server are shown in Figure 5.

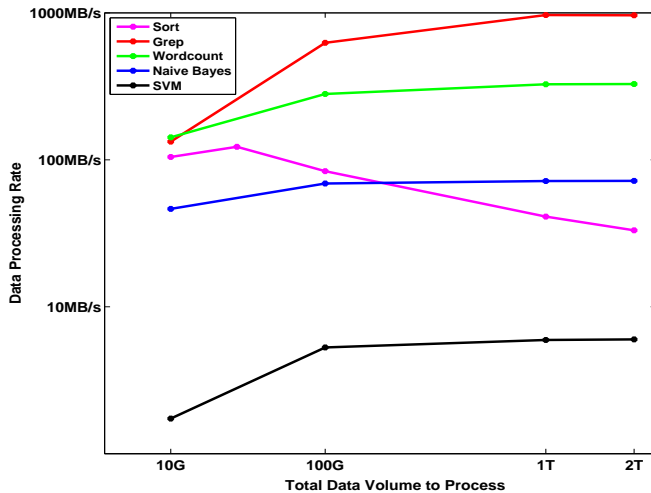
From Figure 4 and Figure 5, we can find that for different application, the cache and TLB behavior data have different trend with increasing data volumes. For example, for L1 Instruction misses per 1000 instructions, the number of *Sort* increases with the volume of data input while the number of *Grep* decreases. However, for different (five) applications, the numbers reach stable values when the data volume increases to a certain extent.

In sum, we conclude that some architectural events are tending towards stability only on condition that the data volume increases to a certain extent. This observation has a significant implication for architecture researches since simulation is time-consuming. We will perform further investigation.

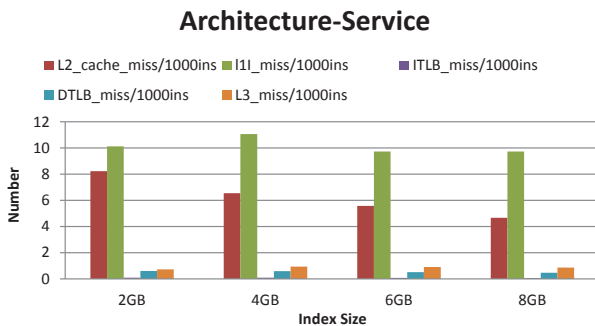
## 6 Conclusions

In this paper, we present our joint research efforts with several industrial partners on big data benchmarking from Web search engines: the most important domain in Internet services in terms of the number of page views and daily visitors.

Our benchmarking methodology includes three parts: first, we follow an incremental approach to assemble big data benchmarks, and then an innovative data generation methodology and tool are proposed to generate scalable volumes of big data from a small seed of real data, preserving semantics and locality of data. Finally, we consider diversity of workloads in addition to volumes, velocity and variety of data. Also, we preliminarily report an experiment to verify correctness of the data generation methodology of *BigDataBench* and two case studies using *BigDataBench* for both system and architecture researches. We gained two insights from the observations: first, the peak data processing rates of big data systems are both applications and data volumes dependent, and hence tuning peak performance must consider different application scenarios. Second, some architectural events, e. g., cache and TLB behaviors, are tending towards stability only on condition that the data volume increases to a certain extent. This observation has a significant implication for simulation-based



**Figure 3. Data Processing Rates for Different Data Volumes to Process.**



**Figure 5. Cache and TLB Behaviors of the Nutch Server.**

architecture researches since large-scale simulation is time-consuming.

## 7 Acknowledgements

We are very grateful to anonymous reviewers. This work is supported by the Chinese 973 project (Grant No.2011CB302502), the Hi-Tech Research and Development (863) Program of China (Grant No.2011AA01A203, No.2013AA01A213), the NSFC project (Grant No.60933003, No.61202075) and the BNSF project (Grant No.4133081).

## References

- [1] <http://www-01.ibm.com/software/data/bigdata/>.
- [2] <http://searchenginewatch.com/article/2101282/Who-Uses-Search-Engines-97-of-Adult-U.S.-Internet-Users-Study>.
- [3] Alexa website. <http://www.alexa.com/topsites/global>.
- [4] Hadoop home page. <http://hadoop.apache.org/>.
- [5] Hive home page. <http://hive.apache.org/>.
- [6] memcached home page. <http://memcached.org/>.
- [7] scrapy home page. <http://scrapy.org>.
- [8] sphinxsearch home page. <http://sphinxsearch.com/>.
- [9] G. A. Big data benchmarking-data model proposa. In *First Workshop on Big Data Benchmarking, San Jose, California*, 2012.
- [10] L. Barroso et al. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 4(1):1–108, 2009.
- [11] D. M. Blei et al. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.
- [12] Y. Chen. *We Don't Know Enough to make a Big Data Benchmark Suite*. Workshop on Big Data Benchmarking, 2012.
- [13] Z. Chen et al. Characterizing os behavior of scale-out data center workloads. *The Seventh Annual Workshop on the Interaction amongst Virtualization, Operating Systems and Computer Architecture (WIVOSCA 2013)*, 2013.
- [14] C. Cortes et al. Support vector machine. *Machine learning*, 20(3):273–297, 1995.
- [15] S. Deerwester et al.. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.
- [16] Z. Jia et al. *The Implications of Diverse and Scalable Data Sets in Benchmarking Big Data Systems*. Second Workshop on Big Data Benchmarking, 2012.
- [17] M. Ferdman et al. Clearing the clouds: A study of emerging workloads on modern hardware. *Architectural Support for Programming Languages and Operating Systems*, 2012.
- [18] W. Gao et al. A benchmark suite for big data systems. In *The 19th IEEE International Symposium on High Performance Computer Architecture(HPCA 2013) Tutorial* <http://prof.ict.ac.cn/HPCA/index.html>.
- [19] Z. Jia et al. Characterizing data analysis workloads in data centers. In *Workload Characterization (IISWC), 2013 IEEE International Symposium on*. IEEE.
- [20] J. Lafferty et al. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.
- [21] P. Lotfi-Kamran et al. Scale-out processors. In *Proceedings of the 39th International Symposium on Computer Architecture*, pages 500–511. IEEE Press, 2012.
- [22] C. Luo et al. Cloudrank-d: benchmarking and ranking cloud computing systems for data processing applications. *Frontiers of Computer Science*, 6(4):347–362, 2012.
- [23] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- [24] A. Rajaraman. More data usually beats better algorithms. *Datawocky Blog*, 2008.
- [25] L. Wang et al. In cloud, can scientific communities benefit from the economies of scale? *Parallel and Distributed Systems, IEEE Transactions on*, 23(2):296–303, 2012.
- [26] H. Xi et al. Characterization of real workloads of web search engines. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, volume 11, pages 15–25. IEEE, 2011.
- [27] J. Zhan et al. Cost-aware cooperative resource provisioning for heterogeneous workloads in data centers. *Computers, IEEE Transactions on (Volume: PP, Issue: 99)*.
- [28] J. Zhan et al. High volume computing: Identifying and characterizing throughput oriented workloads in data centers. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1712–1721. IEEE, 2012.