A COMPLETE OVERVIEW ON BIG DATA ANALYTICS

Savitha.C
R and D Engineer
EMIX Technologies India Pvt Ltd
No. 9, 1st Main, Ganganagar,Bangalore
savinish10@gmail.com

Abstract : The reason that is primary of paper is to provide an in-depth analysis of different platforms available for doing big information analytics. This paper surveys different platforms available for big information analytics and assesses the advantages and disadvantages of every among these platforms according to different metrics such as scalability, data I/O rate, fault tolerance, real-time processing, data size supported and task help that is iterative. An in depth description of the pc software frameworks utilized within each of these platforms can be discussed with their strengths and drawbacks besides the apparatus. A number of the critical characteristics described here could possibly help the visitors for making an decision that is informed the best choice of platforms based on their computational requirements.. So that you can provide more insights in to the effectiveness of each of the platform into the context of big information analytics, specific implementation level details of the widely used k-means clustering algorithm on different platforms are also described within the form pseudocode.

Keywords: Big data, MapReduce, graphics processing units, scalability, big data analytics, big data

I .Introduction

The scales of petabyte information flooding daily from internet solutions, social media, astronomy, and biology science, for instance, have driven the shift of data-processing paradigm. Big information refers to a collection of big datasets which will not be processed database that is utilizing is conventional tools et.al (72) . The storage, manipulation, and especially information retrieval of big data have now been widely re-searched and engineered by academia and industry. Google's MapReduce in 30 which leads this shift. It has influenced new means of taking into consideration the programming and design of large systems which are distributed. In contrast to conventional database management systems (DBMSs), MapReduce is outstanding for better ease of use, scalability, and fault-tolerance, but controversial in programming and effectiveness complexity due to the abstraction that is low. Considering that the publication of MapReduce in 2004, there are numerous works focusing on the limits of MapReduce. It is now the most actively investigated and data-processing that is solid that is big. Hadoop in et.al of 7, an implementation that is open-source of, has been extensively utilized outside Google. Following the success of MapReduce, many other data-processing that is big additionally intending at horizontal scalability, simple API, and schema free information have actually emerged. There are three styles being major are developing. One follows the info parallel concept of MapReduce, which employs programming that is low-level user-defined options for general-purpose use, however with more programming that is versatile, also improved performance. For example, Spark in 83 supports iterative and computations which can be interactive. Dryad in et.al 50 provides control that is communication that is okay and user-defined operations in place of necessity Map/Reduce. Another trend takes advantageous asset of the knowledge that is long-time of utilizing abstractions being high-level. Into the data storage space layer, NoSQL (maybe not SQL that is just, such as MongoDB (67) and Cassandra in 53, implement the characteristics of scalability, schema-free, and persistence weighed against traditional data-bases being relational big data applications. Into the data-processing layer, systems with this trend either only develop SQL-like languages together with general execution machines such as for example Hive et.al 75 or build systems from scratch, including storage, execution engine, and development model, such as for instance Dremel in 66. The trend that is remaining on domain-specific issues, e.g. machine learning (ML) and stream data processing. Recently, large-scale ML systems are earnestly researched and developed, since scalability is amongst the bottlenecks now for ML applications. Representative ML systems include the GraphLab that is graph-centric was in 62 and the ML Petuum that is enteric in 81. S4 (68)and Storm (76) are proposed to process data that are stream/real-time e.g., inquiries in search-engines and Tweets in Twit-ter. The subsection that is categorizing that is following paint an overall image of the current non-relational big information systems.

II. Categorization of Big Data Systems

Big information ecosystems are layered software stacks, in-cluding a database that is low-level a data-processing engine based on that.The low-level databases are utilized for data mainte-nance and low-latency questions which are random. In contrast to conventional ones, the generation that is brand new for big data applications are featured with a high scalability, schema-free, persistence, high supply, and easy API. Based on the information which can be real on disk, the sys-tems could be classified as line store, document shop, graph shop, and shop [85] that is key-value. Fig. 1(a) shows systems being representative each category. A database that is column-oriented attribute values belonging to your column that is same, instead of rows. A database that is document-oriented data in documents, with each document assigned an integral that is exclusive to retrieve the doc-ument. A database that is graph that is graph-oriented representing collections of entities (nodes) and their re-lationships (edges) with one another. A database that is key-value data as a pair of key-value pairs, also known as a wide range that is ssociative arranged into rows. It is built to measure up to a size that is big.

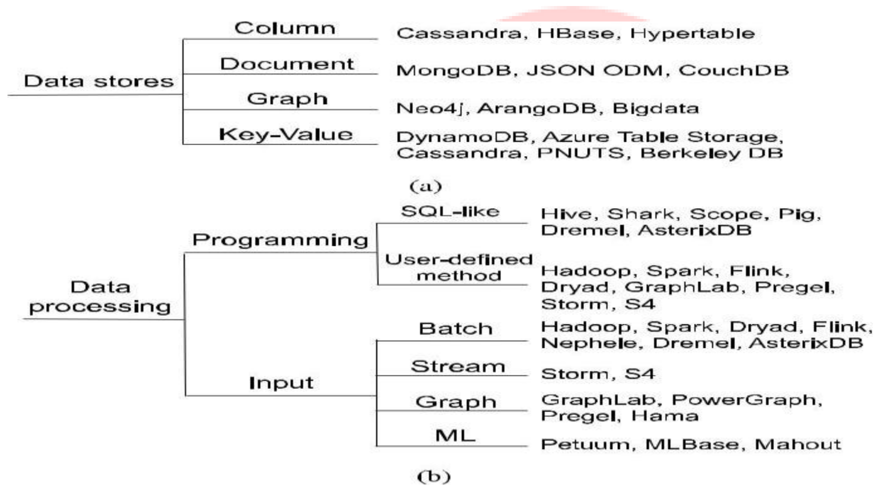
Data stores
Column
Cassandra, HBase, Hypertable
Document
MongoDB, JSON ODM, CouchDB
Graph
Neo4j, ArangoDB, Bigdata
Key-Value
DynamoDB, Azure Table Storage, Cassandra, PNUTS, Berkeley DB
(a)
Data processing
Programming
SQL-like
Hive, Shark, Scope, Pig, Dremel, AsterixDB
User-defined method
Hadoop, Spark, Flink, Dryad, GraphLab, Pregel, Storm, S4
Input
Batch
Hadoop, Spark, Dryad, Flink, Nephele, Dremel, AsterixDB
Stream
Storm, S4
Graph
GraphLab, PowerGraph, Pregel, Hama
ML
Petuum, MLBase, Mahout
(b)

Fig. 1. Categorization of (a) data storage systems and (b) data-processing systems.

Another dimension that can categorize databases is data format, including structured, semi-structured, and unstructured. It chooses exactly how data is interpreted. Un-structured data, such as for instance text messages and videos, is information that has not been arranged into a framework access that is allowing is not hard elements of the data. Organized data could be the opposite, arranged therefore each element may be accessed in several combinations. Semi-structured data lies between the 2, although not organized in to a framework, it can have information that is extra with the information to permit elements of that data become addressed. Traditional databases being relational help organized information, but generation that is brand new such as MongoDB and Cassandra can support structured and semi-structured data, along with unstructured information for some degree.Atop a database is really a data-processing layer make it possible for data experts, developers, and business users to explore and analyze information. As Fig. 1(b) shows, systems in this layer can be classified along two measurements. Accord-ing to your development abstraction, as mentioned earlier, systems like Dremel and AsterixDB utilize high-level declarative that is SQL-like while others use mapReduce-like functions being user-defined. Depending on the input, we categorize current information which can be big as batch, stream, machine and graph learning processing. Batch processing is efficient for processing datasets being large where data are collected, prepared and distributed in batches. Stream processing emphasizes in the velocity of the input that is continual such as for instance user click and request streams on webpages. Data must certanly be processed in an occasion that is disallowing that is certain information together for efficiency. Graph processing operates on graph information structures and conducts model parallelism besides the data parallelism of MapReduce. It usually runs iteratively on the input that is same each iteration accompanied by some synchronisation. Graph processing can help solutions which also are partial ML problems. While, systems like Petuum and Mahout are developed especially for ML algorithms, which are formalized as iterative- convergent programs et.al (81). The goal of the systems is fast and convergence that is efficient obtains the optimality of a function that is objective. Thus, fine-grained fault threshold and strong consistency for other processing systems may not be essential for ML issues.

This paper shall concentrate on the study of information layer that is processing. It shall first introduce MapReduce, compare it with DBMSs, and discuss the optimization works for MapReduce. More surveys being detailed MapReduce and Hadoop is seen in (55, 57, 72, and 73). This paper will then overview one other system in the categories discussed above: batch, stream, graph, and device learning processing. The area on batch processing shall cover the 2 programming abstractions: SQL-like and user-defined. The survey will be more from a research that is considerable of view, focusing on distinctive some ideas and model abstractions of different systems, even though many of them may not be popularly utilized now. A quantitative that is fair qualitative contrast of most that available information that is big systems is important. Nevertheless, no standard benchmark suite can be obtained yet. This study may also learn work that is present big data benchmark building that will be challenging and has now perhaps not drawn enough attention due to these benchmarking and assessing pressures of big information systems. Finally, some classes and research that is future will probably be talked about.

III B A I S C M A P R E D U C E F R Am E W O R K

MapReduce is just a development model for processing data that are big large-scale data being distributed systems, intro-duced by Dean and Ghemawat (30,31,32). It's simple and abstracts the information on managing a distributed System, such as for example parallelization, fault-tolerance, information dis-tribution and load balancing. It really is now trusted for the variety of algorithms, including graph that is large-scale, text processing, information mining, device learning, sta-tistical device translation, and many areas et.al 31. There are several source that is available commercial imple-mentations of MapReduce, out of that the most one that's popu-lar Hadoop developed by Apache in et.al 7. This part will discuss the initial framework that is fundamental of and then provide a comparison with DBM

A. MapReduce Framework

The MapReduce model has two phases :Map and Re-duce working on key/value pairs. The Map phase maps the user written functions and input pairs to distributed machines generating intermediate key/value pairs, and the Reduce phase reduces the intermediate pairs to a sin-gle result. The workflow of MapReduce execution is as follows

1) The distributed file system (e.g., Google File System in (42) will first partition the input information right into a set of M splits (e.g., 64 MB in size) and store a few copies of every split on different machines for fault tolerance.

2) The MapReduce library will generate a member of staff and master that is most of the individual program (M Map tasks and R decrease tasks). The master assigns work to worker copies and coordinates all the tasks operating. For locality, the master will attempt to schedule an activity that is map a ma-chine which has a reproduction regarding the correspond-ing input data.

3) A Map worker will scan its regional input partition and generate intermediate key pairs making use of user’s function that is map. The results are stored on local disk and generally are split into R partitions for every Reduce task. The addresses of the outcomes which can be inter-mediate be informed towards the master. The master shall forward those areas towards the Reduce employees.

4) A Reduce worker will see the intermediate re-sults from the local disk of each and every task that is map and then sort the outcomes by their key values. The user's Reduce function will aggre-gate values with all the key that is exact same generate final results kept in a file system that is worldwide.

5) The master will ping every worker occasionally. If any Map or Reduce worker fails, its task will probably be scheduled to some other worker that is available.

6) After all of the tasks finish, the user program will be waked up.

MapReduce has a few benefits which are outstanding Simplicity: it needs code writers don't have any paral lel and/or system experience that is distributed. The sys-tem installation and setup are relatively straightforward Table 1 Comparison of Parallel DBMSs and MapReduce

- Fault threshold: Failures are normal for a com-puter group with huge number of nodes. MapRe-duce can cope with fine-grain failures, reduce the amount of work lost, and doesn't need re-start the task that is working scratch.

- Flexibility: The input data of MapReduce can alternatively have any structure of the schema that is certain.

- Independency: MapReduce can also be storage space system-independent. The storage systems supported include files saved in distributed file system, database query results, data kept in Bigtable structured and[24] input files [32].

- Scalability: MapReduce can scale to tens and thousands of processors.

A. Comparison Between MapReduce and DBMS Before MapReduce, synchronous DBMSs are utilized while the approaches for large-scale information analysis. Fundamentally, all tasks which are mapReduce be written as comparable DBMS tasks through the term that is early of, it provoked strong doubts from Database communities et.al in 29. Comparisons and debates between DBMS and MapReduce have now been shown in a number of articles (32, 33, 55, 70, 74). The debates were toned down until Stonebraker et al. concluded the relationship between DBMSs and MapReduce. They noted that MapReduce is complemen-tary to DBMSs, not a technology that is competing. The aim of DBMS is effectiveness while MapReduce aims at scalability and fault tolerance. The two systems are plainly improv-ing themselves through drawing the counterpart's energy. Works like SCOPE (23)and Dryad (50) all point that is correct method.

IV. Support for Different Dataflow

MapReduce calls for the issue composition to be Map that is strict and actions in a batch processing means. This subsection shall talk about the expansion works on itera-tive, online and dataflow that is streaming according to MapReduce framework.

1) Iterative Dataflow: MapReduce does not support it-erative or recursive straight. Nonetheless, numerous data analy-sis applications, such as information mining, graph analysis and community that is social, require iterative computations. Code writers can manually issue multiple MapReduce jobs to implement programs which are iterative. But, this method causes three performance that is primary. First, unchanged information from iteration to iteration will undoubtedly be re-loaded and reprocessed at each and every iteration, wasting I/O, network bandwidth and CPU. 2nd, some termination conditions, like no production modification between two iterations which can be consecu-tive may itself invoke a MapReduce task. Third, the MapReduce jobs of various iterations have to complete serially. To fix those conditions that are nagging there are a variety of works on extending MapReduce for iterative processing (22, 37, 38, 77, 84).Those works usually need to implement three main extensions of MapReduce: incorporating an iterative programming screen, particularly the screen for termination conditions, caching invariant/ static information in local disk or memory, and modifying task scheduler to be sure data reuse across iterations to aid iterative processing. We shall next introduce some representative works as examples.

HaLoop: A programmer specifies the cycle human anatomy and optionally specifies a termination condition and data being loop-invariant write a HaLoop (22) program. The mas-ter node of HaLoop keeps a mapping from each slave node to the data partitions that this node prepared in the iteration that is previous. The master node will likely then try to assign to the node an activity which consists of information which are cached. HaLoop keeps three forms of caches. First, reducer input cache, caches the consumer specified loop-invariant tables being intermediate. So in later iterations, the reducer can seek out one of the keys into the reducer that is regional cache to get asso-ciated values and pass together utilizing the shuffled key to your user-defined Reduce function. Second, reducer cache that is out-put stores and indexes the most recent out-put that is neighborhood each reducer node to reduce the price of evaluating termination conditions. Third, mapper input cache, caches the input information split that the mapper performs a read that is nonlocal the iteration that is first. Every one of them fits application that is significantly different.

iMapReduce: Compared with HaLoop, the advance-ment of iMapReduce (84) is it allows execution that is asynchronous of iteration. The Map tasks of the iteration may be overlapped utilizing the Reduce tasks associated with iteration that is final. This work assigns a data that is exact same up to a Map task and a Reduce task. Therefore, there's a communication that is one-to-one the Map while the Reduce tasks. The results is supposed to be repaid towards the Map that is matching task the Reduce task creates specific documents. The job scheduler constantly assigns a Map task and its own corresponding Reduce task to the worker that is reduce that is same system resources required. The Map can begin processing the data without waiting for other tasks being map the process is accelerated. iMapReduce proposes the thought of persistent Map and Reduce. For a Map that is task that is persistent all the input information are parsed and processed, the job will wait for results from the decrease tasks and stay triggered again. To implement this, the granularity of information split has to make sure most of the tasks that are persistent at the start based on the task that is avail-able. This will make load balancing challenging.

iHadoop: iHadoop et.al(38) also supports asynchronous Map/Reduce tasks as iMapReduce. However, it makes use of scheduling that is dy-namic of fixed task and node map-ping. It shall not persist tasks for the iteration that is next. Alternatively, the paper reckons that using the scale that is sets which are big the runtime can optimize the duty granularity so your right time and energy to create, destroy, and schedule tasks is in-significant towards the time for you to process the input information. Therefore, iHadoop could help wider kinds of iterative applications.

Twister: Similar to iMapReduce’s Map/ that is Reduce that is twister that is persistent uses very long running Map/Reduce tasks and does not start new Map and Reduce tasks for each and every iteration. It's an runtime that is publish/subscribe that is in-memory communication that is situated data transfer in place of a distributed file system. How-ever, Twister is dependant on the presumption that data sets can fit into the memory that is distributed which can be not necessarily the truth.

MapIterativeReduce: MapIterativeReduce et.al(77)supports iterative Reduce for reduce-intensive applications such as linear dimension and regression decrease for Microsoft Azure cloud platform. It eliminates the barrier between Map and Reduce by starting reducers plan the data the moment it becomes available from some mappers. The re-sults from the iteration that is final would be fed back into reducers being successive a reducer combines most of the in-put data and creates the end result that is last.

2) Online and Streaming Dataflow: Computations on MapReduce are performed in a pattern that is batch-oriented namely the complete input and production of each and every Map or MapReduce is faster than DBMS, but slower in task exe-cution time. Longer execution time of MapReduce is partly because some implementation certain issues of MapReduce, such as the cost that is start-up of. You can find reasons which are model-related. The layout for the data and it has doing most of the parsing at run time as dining table 1 shows, DBMS does the parsing at loading time and may also re-organize the input information for many optimizations, while MapReduce won’t change. DBMS also has ad-vanced technologies developed for decades, such as for example com-pression, column storage space, or advanced algorithms that are parallel. In addition, MapReduce has to send numerous control messages to synchronize the processing which overhead that is in-creases.

Reusability: MapReduce will not use index or schema. Programmers need to parse the structure of the input files or implement indexes for speedup into the Map and Reduce programs. Besides, users need to offer imple-mentations for simple and operations which are common such as for example selection and projection. Those rule that is custom be difficult to be reused or provided by others and it will be error-prone and suboptimal. Nonetheless, DBMS has schema that is built-in index that may be queried by code writers. DBMS additionally supports operators being numerous high rate of abstractions. Users can easily specify whatever they want through the functional system in the place of ways to get it.

The circumstances where MapReduce is superior to DBMS are et.al 74 to close out, based on the power of MapRe-duce

• Once-analysis data sets. Those data sets are not worth your time and effort of reorganization and indexing in a DBMS. In contrast, DBMS is more suit-able for tasks needing parsing that is repetitive.

• Complex analysis. Dean and Ghemawat et.al 32 point out some situations where in actuality the functions which are map too complicated to be expressed effortlessly in a SQL question, such as for example extracting the outbound links from a collections of HTML documents and aggregating by target document.

• Quick begin analysis. MapReduce implementations are really easy to configure, to program and to run.

• Limited-budget task. Most MapReduce implementations are open-source, while rare open-source DBMSs that are parallel.

A. Communication Optimization

Communication in Hadoop system is implemented by inadequate techniques, such as for example HTTP/RPC. Lu et al. (63) presented DataMPI, that can be an conversation that is efficient for big information computing which includes the conversation and processing of many key-value pairs. To bridge the 2 aspects of higher end com-puting and big information computing and expand MPI to sup-port Hadoop-like big information computing jobs, they abstract what is required regarding the 4D (Dichotomic, Dynamic, Data-centric, and Diversified) bipartite relationship model.

• Dichotomic. The MapReduce as well as other information which may be show that is communications being big between two communicators. The com-munication that is underlying a graph that is bipartite i.e., the procedures are dichotomic and be involved in either the O com-municator or the A communicator.

• Dynamic. Big Data connection features a dynamic characteristic, what this means is the product range that is real is wide of tasks being running each communicator of-ten changes dynamically because of task finish and launch.

• Data-centric. Jim Gray's Laws et.al 48 tell that computations should really be relocated to important computer data, in the place of information to your computations in big information computing. Such concept are situated in lots of information that can easily be popular are big models and systems.

• Diversified. Although plenty that is complete of this can be compared among different information that are big systems may be found, there stay diversities.

They key-value which can be abstract based interaction, which capture the conversation that is important of hadoop-like information computing that is big. Through the standpoint of development, numerous information that is big systems (age.g., Hadoop MapReduce, S4, or HBase) choose key-value pair when the core information representation framework, that is not hard but carries a ability that is strong carry in-formation that is rich. Consequently, this will be a undeniable fact that is offer that is great set based relationship interfaces for big information computing systems and applications though the buffer-to-buffer screen signature that is old-fashioned. Such level that is abstraction that is decrease that is high program-ming complexity in parallel Big Data applications.

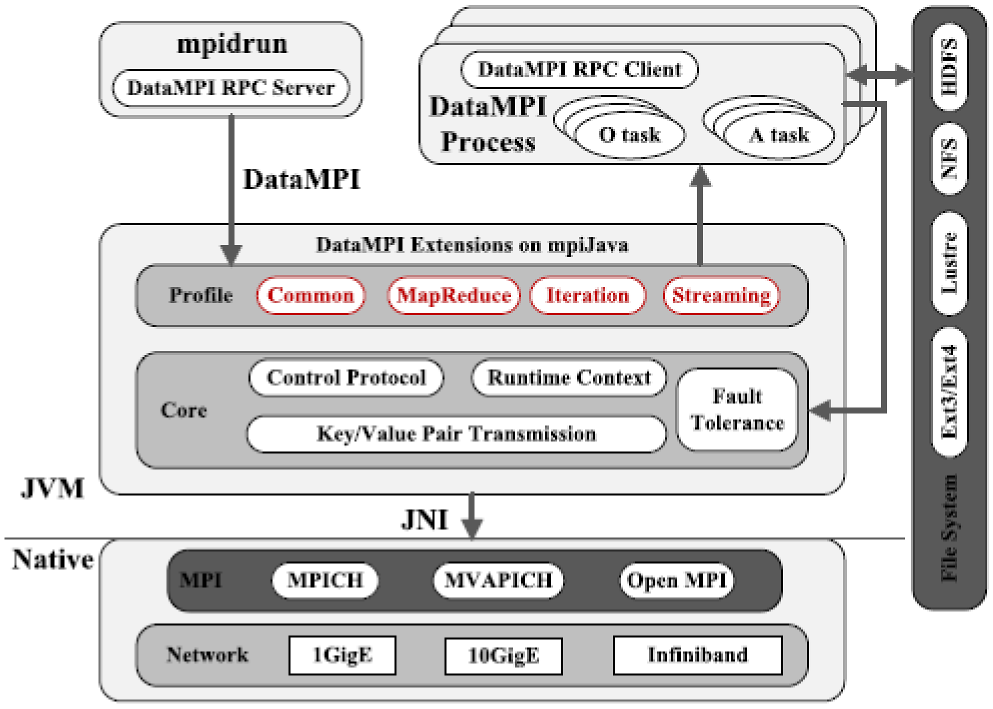

Fig. 3 presents the two-layer architecture of DataMPI. In the JVM layer, DataMPI extends the mpiJava design

The usage of DataMPI specification and so on by powerful procedure management in the Java layer, optimized buffer management by native IO that is direct. The lower layer will be the layer that is indigenous by which JNI is useful for connecting routines that are top are java-based native MPI libraries. Weighed against Hadoop, DataMPI gives a more library and light-weight that is users that can easily be flexible. Liang et al. [58] use BigDataBench.

E. Energy Efficiency Optimization

Energy effectiveness is definitely an topic that is data cen-ters being crucial. Globally, information centers are predicted to eat about US$30 billion worth of electricity 12 months that is per et.al 71. Power and costs that are cooling compared to the IT equipment it supports and also compensate about 42percent of this information facilities costs which are running. There are many magazines for information center power administration. Nevertheless, energy for MapReduce clusters has not drawn attention that will do. It'll be a research that is future that is guaranteeing.

Nowadays there are broadly two ways to reduce steadily the energy cost of MapReduce clusters: one is to power down the employment that is et.al 54 that is low (56); the other is to match the hardware resources for the workload faculties, just like the CPU computing capability (65,79). Both ways are actually a tradeoff between performance and energy.

1) Energy Down Nodes: The motivation of (54) and (56) is the fact that CPU that is utilization that is average is low. Both 2) Match Hardware: Mashayekhy et al. [65] propose energy-aware MapReduce scheduling algorithms while satisfying the answer that is ongoing contract instead of minimiz-ing time that is operating. It generates utilization of the truth that same Map or Reduce tasks consume variant energy and time that is running various nodes with this combined group that is underlying. The algorithms first profile the tasks' energy and time that is running on several types of nodes, then build an schedule that is optimized future

execution the maximum amount of jobs have to run sometimes, such as for example spam detection. The scheduler will find the nodes with low power price along with sat-isfying the clear answer that is agreement that is ongoing purchase to accomplish the jobs. Outcomes reveal that the algorithms can acquire near optimal solutions with significant energy cost benefits when compared with schedulers intending at minimizing the time that is right is operating. Wirtz and Ge [79] adjust the processor regularity on the basis of the working jobs which are working calculation requirements. It compares three poli-cies: 1) same regularity for all processors

2) maximum processor regularity within the map and Reduce func-tions and frequency that is minimal, and so the compu-tations utilizes fast cores and I/O uses sluggish ones; and 3) regularity set to bound the performance loss in only a value individual specified. This setting makes use of CPUMiser on each node. CPUMiser will gather performance countertop information for CPU task then adjust the regularity precisely. Results show that smart frequency establishing can enhance the energy effectiveness, nonetheless, the amount that is famous of be determined by the feature that is work-load. A work that is recent evaluates the performance and power footprint of Hadoop on both real and digital teams, thinking about the hadoop that is conventional col-locating information and computing, plus the model that is alternatngi of these papers head to two guidelines. Reference [54] makes use of a subset of the nodes for the MapReduce task and abilities down others, while [56] uses all the nodes to initial complete the ongoing work, then powers down all the nodes after the task finishes. Lang and Patel [54] define a covering subset (CS), and alter the HDFS (Hadoops file system) to ensure that one or more reproduction of each and every offered information block is within the subset that is addressing. In that case your nodes perhaps not in the set are disabled without affecting the possibility associated with information even though the execution of this work. The outcomes show that disabling nodes in many situations that are complete energy expense while the decrease quantity is dependent upon the applying. The outcomes additionally expose a celebration that is energy that is operating tendency even though the number of disabled nodes increases. Leverich and Kozyrakis [56] tries in order to avoid pitfalls of [54], namely HDFS modification, operating time enhance and storage overprovision. It proposes a technique called All-In Strategy (AIS) that starts all of the nodes to perform the job as soon as possible and after that abilities down the unit that is entire there is absolutely no task. So the reaction time degra-dation regarding the ongoing work is predictable, in line with the time for it to power the hardware up and OS. Results show that for long and calculation complex MapReduce jobs, AIS outperforms than CS in response time and energy preserving.;

3) Match Hardware: Mashayekhy et al. (65) propose energy-aware MapReduce scheduling algorithms while satisfying the solution that is ongoing agreement in the place of minimiz-ing time that is running. It generates use of the truth that same Map or Reduce tasks consume variant power and time that is running different nodes of this blended group that is underlying. The algorithms first profile the tasks' power and time that is running on various kinds of nodes, then construct an routine that is optimized future execution the maximum amount of jobs need to run sporadically, such as for instance spam detection. The scheduler will find the nodes with low power price as well as sat-isfying the answer that is agreement that is ongoing order to accomplish the jobs. Outcomes reveal that the algorithms can obtain near optimal solutions with significant power cost savings compared to schedulers planning at minimizing enough time that is correct is running. Wirtz and Ge in 79) adjust the processor regularity in line with the working jobs that are working computation needs. It compares three poli-cies: 1) exact same regularity for many processors; 2) maximum processor regularity into the map and Reduce func-tions and minimal regularity otherwise, therefore the compu-tations utilizes fast cores and I/O makes use of sluggish ones; and 3) regularity set to bound the performance loss in only a value individual specified. This environment employs CPUMiser on each node. CPUMiser will collect performance countertop information for CPU task and then adjust the regularity precisely. Outcomes reveal that smart frequency establishing can enhance the energy effectiveness, nevertheless, the particular level that is famous of be determined by the feature that is work-load. A work that is recent evaluates the performance and power footprint of Hadoop on both genuine and electronic groups, considering the hadoop that is conventional col-locating information and computing, in addition to the model that is alternating

4) Parallel Database Systems

AsterixDB: AsterixDB (3, 17) is just a research study in-volving scientists at UC Irvine as well as UC San Diego and UC Riverside, which aims to build a infor-mation that is system that is scalable support for the storage, querying, and analysis of huge collections of semi-struc-tured nested information items, with a new declarative query language (AQL).

The consumer type of AsterixDB is made of two components that are core the Asterix Data Model (ADM) additionally the question language (AQL) targeting information being semi-structured. ADM supports a number that is wide of data for-mats, and carries out AsterixDB data query and storage processing. Every person ADM data instance is typed, self-describing and saved in a dataset. The datasets of ADM could be indexed, partitioned over multiple hosts in a cluster, and replicated to attain scalability or avail-ability. Datasets could have connected schema information that describes the core content of these circumstances.
Asterix Query Language (AQL) may be the question language for AsterixDB, that is used to access and manipulate Asterix Data. AQL borrows from XQuery and Jaql the
programmer-friendly declarative syntax and is compara-ble to those languages when it comes to expressive energy. AQL is designed to cleanly match and manage the info structuring constructs of ADM. So that it omits numerous XML-specific and features being document-specific.
AsterixDB uses a scalable motor that is parallel Hy-racks in (20) to process inquiries. AQL queries are put together into Hyracks Jobs for execution, that are in the shape of DAGs contains Operators and Connectors. Each oper-ator presents a AQL operation and is responsible for loading partitions of input information and creating output data partitions, while each connector redistributes the output partitions and makes input partitions for the operator that is next.Operators in Hyracks have a three-part specification, provided right here.
Operator Descriptors. Every operator is built as an implementation of the Operator Descriptor program.
Operator Strategies. Hyracks allows an operator to spell it out the different phases involved in its assessment in an amount that is high Tasks. Each activity of a operator really represents a couple of parallel tasks to be planned on the machines within the cluster. Hyracks also includes a collection of pre-existing operators and connectors, for instance, File Readers/Writers and Mappers operators and M:N Hash-Partitioner connectors.

V. S t r e a m , g r a p h , a n d m a c hi n e l e a rn i n g p r o c e s s i n g s y s t e m s

This section is about the other three categories of big data processing systems. Although general-purpose sys-tems could also be used of these applications, specific sys-tems can leverage domain features more effortlessly, so efficiency, programmability, and correctness are nor-mally improved. Storm and S4 will soon be discussed as repre-sentatives of stream processing. Graph systems will introduce GraphLab and Pregel, which can be useful for ML issues. Petuum and MLbase are specially de-signed for ML algorithms.
A. Stream Processing Systems
Data stream applications such as the search that is real-time social support systems need scalable flow processing systems operating at high data rates instead of the long-latency batch processing of MapReduce-like systems. This sec-tion shall introduce Storm and S4 systems specific for flow processing.
Storm: Storm in 76 is an open-sourced flow that is distributed system developed by Twitter. You can easily process unbounded streams of data, such as for instance a hundred million tweets a day. For real-time processing and computation that is con-tinuous Storm operates more efficiently than batch processing. More than 60 organizations are experimenting or using with Storm.
The Storm model consists channels of tuples flowing through topologies defined by users as being a Thrift in et.al 12 ob-ject. Thrift's cross language services be sure that any languages could be used to produce a topology. The vertices of a topology express computations and sides are data movement. There are two types of vertices, spouts and bolts. Spouts will be the way to obtain data flow, and bolts process the tuples and pass them to your downstream bolts. The topology might have rounds.S4: Different from Storm, S4 in (68) uses a decentralized and architecture that is symmetric ease. There is absolutely no node that is central the S4 system. S4 design hails from a mix of MapReduce and the Actors model.
A stream of S4 is described as a series of events in the form of (K, A) where K is just a key that is tuple-valued and A may be the attribute, for instance, a (word, count) event in a word look counting problem.

The occasions are transmitted through Processing Elements (PEs) which perform some computations published by designers. A PE is distin-guished from other PEs by four elements: its function-ality defined, the kinds of events it consumes, the feature that is keyed those occasions, additionally the value associated with the characteristic. PE is defined by users and S4 system initiates one PE per each key that is exclusive the flow. The state of a PE is not accessible by other PEs.

The PEs operate in logical hosts, called Processing Nodes (PNs). PNs would be mapped to nodes being real the interaction layer. The PNs are accountable for listen-ing the occasions, executing operations regarding the activities, dis-patching occasions, and output that is emitting. The interaction layer uses ZooKeeper too to coordinate between nodes. The work that is main of layer is to manage cluster, recover failure, and map nodes.

S4's fault tolerance is through checkpoints. It does not help guaranteed in full processing for tuples like exactly what Storm does.

Graph Processing Systems

To especially support graph that is large-scale on clusters, parallel systems, such as GraphLab [62] and Pregel [64], are proposed. They execute graph models and information parallelly. Some of ML algorithms concentrate on the also dependencies of information. Its natural to make use of graph struc-tures to abstract those dependencies. So they are essen-tially computations that are graph-structured.

GraphLab: GraphLab in 62 adopts a model that is vertex-centric computations will operate on each vertex. The ab-straction of GraphLab includes the info graph, upgrade sync and function procedure. The info graph manages user-defined information, including model parameters, algorithm state and analytical information. Update functions are user-defined computations changing the info of a vertex and its own vertices which can be adjacent sides in the information graph. Those functions will return the modified data while the vertices that require to be modified by the change functions within the iterations which are future. Sync operations are used to maintain worldwide statistics describing data kept within the data graph, for example, worldwide convergence estimators.

Take the PageRank that is popular issue, which re-cursively describes the ranking of a webpage, for example, each vertex regarding the data graph represents a webpage stor-ing the ranking and every edge represents a web link keeping the weight associated with website link. The revision function shall calculate the ranking of a vertex in line with the weighted links to its neighbor vertices. The neighbor vertices will soon be sched-uled to the queue waiting for future enhance if the vertex that is current by greater than a limit.

For storage space, the information graph is partitioned accord-ing to domain knowledge that is specific some graph parti-tion heuristics. Each partition will be a file that is split a distributed storage space system, such as HDFS. You will have a meta-graph storing the connectivity file and structure places of these partitions. In line with the meta-graph together with true quantity of physical machines, a fast balanced distributed loading can be executed. Then Vertices will simultaneously be executed on clusters. The GraphLab execution engine supports execution that is fully asynchronous vertices and also supports vertex priorities. It requires the graph that is whole system state to reside in RAM. Each vertex is connected with a reader and a writer lock. Each device only runs updates on neighborhood vertices after finishing lock acquisition and data syn-chronization. The synchronization and acquisition are pipelined for various vertices for each machine to re-duce latency.

The fault tolerance of GraphLab is implemented utilizing distributed checkpoint mechanism. The mecha-nism is made completely asynchronously on the basis of the Chandy-Lamport snapshot. It incrementally constructs a snapshot without suspending execution. The machine is supposed to be recovered through the last checkpoint in case of a failure.GraphLab has nature support for a number of ML algorithm properties.

It supports: 1) graph parallelfor expressing data dependencies of ML algorithm;asynchronous iterative computation for quick conver-gence; dynamic calculation for prioritizing computa-tions on parameters requiring more iterations to converge; and serializability for ensuring that all par-allel executions have comparable sequential execution to permit ML experts focus on algorithm design.

PowerGraph [43] is the variation that is subsequent of. It can efficiently process graphs being natural graphs with power-law distribution of connectivity. Those graphs can cause load imbalance issues because of that the few popular vertices could be with many edges, while lots that is big of vertices are in fact with few edges.

Pregel: the objective of Pregel [64] is build a sys-tem which can implement graph that is arbitrary in a large-scale distributed environment with fault toler-ance. We will introduce Pregel through comparisons with GraphLab.

Just like GraphLab, Pregel additionally utilizes the approach that is vertex-centric. Each vertex is of a value that is user-defined. Each side is associated with a value, a source, and a target vertex identifier. User-defined functions can change the consistant state of vertices and edges.

There are two main differences which can be outstanding GraphLab and Pregel. First, the functions that are update GraphLab can access a vertex, its connected edges and its neighbor vertices. But, The Pregel functions can only access a vertex and its particular edges. The neighbors state will be sent to the vertex through messages. The big event can receive messages sent to a vertex in the previous it-eration, modify the vertex state and its particular outbound edges, deliver messages to other vertices which will be gotten in next iteration and graph topology that is even mutate. 2nd, the graph and ML computation include a sequence of iterations. GraghLab supports the iteration that is asynchronous while Pregel uses the barrier synchronization. Pregel is inspired by Valiant’s Bulk Synchronous Parallel (BSP) model. The framework invoke the same user-defined function to each vertex and execute them in synchronous, and then computation is syn-chronized after every iteration for instance the increment of this global iteration index during an iteration. Therefore, the model doesn't have to think about information deadlock or race problems.

The fault tolerance of Pregel normally implemented by checkpointing. The worker devices helps you to save their state of the partitions to persistent storage space plus the master device helps you to save the aggregator values at the start of each iteration.

Apache Hama [9] is an source that is open motivated by Pregel. Hama realizes BSP on the HDFS, along with the Dryad engine from Microsoft.

Machine Learning Processing Systems

The scale of ML dilemmas are in-creasing exponentially which will be so called “Big Model on Big Data [81] to explore the value of big data, while increasing the ac-curacy of ML models. Therefore, scalability is one of the bottlenecks for ML research [36]. That is highly challenging because to style, implement, and debug distributed ML systems need ML specialists to address cluster development de-tails, such as for example race deadlock and condition, as the
developing of complex mathematics models and algorithms. Most of conventional processing that is parallel, such as for instance MapReduce, don't have normal support for processing ML problems on clusters. To help ease the use of ML algorithms to issues that are industrial-scale numerous systems designed for ML happen developed.

While many of ML problems are represented as graphs, there's also algorithms like topic modeling that aren't inefficient or easy to be represented in that way. Besides, the execution correctness of asynchro-nous graph-based model is[81] that is confusing. This subsection will introduce systems that are ML-centric.

Petuum: Petuum [81] could be the state-of-the-art distributed ML framework, built on an principle that is ML-centric. It for-malizes ML algorithms as iterative-convergent programs. Compared to GraphLab, Spark as well as other platforms which support partial solutions for ML dilemmas, it can support a wider spectrum of ML algorithms. Besides, Petuum considers data parallel and model synchronous execution of an ML program, which can exploit the 3 unique properties of ML algorithm: mistake threshold, powerful structural dependency and convergence that is non-uniform. So it converges more proficiently than other platforms [51].

The Petuum system comprises three elements: scheduler, employees, and parameter host. The 3 functions which are user-defined are schedule, push, and pull. The scheduler allows model parallelism by control-ling which parameters to update by each worker accord-ing towards the routine function that is user-defined. This allows ML programs to investigate dependency that is structural and select independent parameters for synchronous updates in case of parallelization mistake and non-convergence. The routine function additionally allows to consider the convergence that is non-uniform of ML through prioritizing parame-ters that needs more iterations to converge. GraphLab system has also considered this issue. Petuum uses pi-pelining to overlap the routine with worker execution. The scheduler is also accountable for central aggregation through pull function if needed.

The employees in Petuum are accountable to get the parameters and run function push that is synchronous change. The parameters will synchronize with the automatically parameter sever using a distributed provided memory API.
The parameter host provides access that is international parameters via the provided memory API. Based on the concept that ML algorithms are often tolerant to minor errors, the parameter server implements stale Parallel that is synchro-nous(consistency model. Therefore community synchronization and interaction overheads are re-duced somewhat while maintaining the convergence guarantees.
Petuum doesn't specify a information abstraction, so any information storage space system might be utilized. Fault tolerance is accomplished by checkpoint-and-restart, but it is currently suitable for up to hundreds of machines.
Another popular distributed ML platforms is MLbase MLbase is composed of three elements: MLlib, MLI, and ML Optimizer. MLlib has been mentioned in Section IV-A3. This is a distributed ML that is low-level library against Spark runtime. It had been an element of the MLbase project and it is now supported by the Spark com-munity. The library includes algorithms that are typical classification, regression, an such like. MLI is atop MLlib, which presents ML that is high-level development. The layer that is greatest is ML Optimizer. It solves a search issue within the algorithms contained in MLI and MLlib for a most readily useful model that is applicable. Users specify ML dilemmas by MLbase task language that is declarative.

Mahout [10] is an open-sourced ML that is scalable collection. It offers a host for building scalable algo-rithms, many ML algorithm implementations on Hadoop, and brand new algorithms on Spark too. All have actually their very own in-house distributed ML systems besides these open-sourced systems, organizations like Amazon, Facebook, and Yahoo.

V I . B I G DA T A B E N C H M A R K

Both quantitatively and qualitatively while more and more big information processing systems are proposed and implemented, it is critical to fairly assess those systems. How-ever, the growth of standard information which can be big lags behind due to the complexity, diversity, and var-iability of workloads. An easy range besides, the application piles work-loads constructed on cover. Most internet businesses tend to keep their data and applications confi-dential, steering clear of the building [78] that is benchmark. This area will first introduce the most recent and benchmark that is diverse BigDataBench and then talk about other rooms focusing on certain applications.

Wang et al. [78] suggest that big data benchmarking needs the following demands:

Measuring and comparing data that are big and architecture.

Being data-centric.

Diverse and representative workloads.

Covering software that is representative.

State-of-the-art strategies.

Usability.

To fulfill the requirements above, they provide BigDataBench. It really is presently probably the most diverse and repre-sentative data which can be big suite when compared with other proposed suites that are ordinarily for particular applica-tions or software stacks. They pay attention to investigat-ing workloads in three most application that is important based on widely acceptable metrics—the number of web page views and day-to-day visitors, including internet search engine, business, and community that is social. To consider workload candidates, they make a tradeoff between selecting various kinds of applications: including online services, offline analytics, and analytics which can be real-time. The data types include structured, semi-structured, and information being un-structured. The benchmarks are oriented to analytics which can be differ-ent, such as Hadoop, Spark, MPI, Hive, Hbase, and MySQL. The summary of BigDataBench is presented in dining table 3.

HiBench [49] is just a standard suite for Hadoop MapReduce developed by researchers from Intel. This standard contains four types of workloads: Micro Benchmarks (the Sort, WordCount and TeraSort work-loads), Web Search (the Nutch Indexing and PageRank workloads), Machine

Learning (the Bayesian Classifica-tion and K-means Clustering workloads), and HDFS Benchmark. The inputs of the workloads are either information sets of fixed size or scalable and data sets that are synthetic. The fixed-size information sets may either be straight removed from genuine information sets (age.g., the Wikipedia page-to-page website link database employed by the PageRank workload, and a subset associated with the Wikipedia dump data set used by the Bayesian classi-fication workload) or generated according to real data sets (e.g., in the Nutch Indexing workload, the input are the 2.4 million website pages generated according to an Wikipedia that is in-house). The synthetic information sets may either be ran-domly generated (age.g., the inputs used by the 3 workloads of Micro Benchmarks are randomly generated using the programs in the circulation that is hadoop or cre-ated with a couple statistic distributions (e.g., the input for the K-means Clustering workload).

There are various other open-sourced data which can be big rooms. However, like HiBench, they're primarily for spe-cific applications. Yahoo! Cloud Serving Benchmark (YCSB) [27] and LinkBench [15] include online service workloads for cloud computing and community that is social correspondingly. The information being big from AMP Lab [6] is for real-time analysis system. All those benchmarks are not as diverse as BigDataBench

.

VI I. O P E N R E S E A R C H I S S U E S

This paper has offered a survey of present data that are big systems. Some future that is possible guidelines will bediscussed in this part.

Novel Data Model: though there are mature big information storage systems now, due to the fact scale of information grows rapidly, and the raw data comes from more different sources, it's still a challenge that is excellent information storage. The systems need to process data of different structures, therefore the price of normalizing or formatting information that are massive be extremely expensive and unsatisfactory. Consequently, a information that are versatile must certanly be extremely important.

Novel Processing Model: the genuine amount of classes of big information workloads additionally keeps increasing. More parallel algorithms are developed to extract in-formation that is valuable big data. Such a situation presents outstanding challenge for present general function big data systems, they have been created while they just consider the existing workloads when. Furthermore, different classes of applications also have various dependence on re-sources, building a scalable system model with effective resource administration should really be a very problem that is impor-tant. Besides, the look of numerous data which can be big systems aiming at particular kinds of applications makes it hard for businesses with dif-ferent forms of workloads to consolidate them using one cluster

Big Data Benchmark: you will find currently some rooms which are bench-mark for big data, in which BigData-Bench discussed in this paper may be the state-of-the-art one. Benchmarks need to be developed based on the area of application too, in case of biasing systems that are particular. Re-searchers should expand the standard suite with increased popular utilized applications, and promote the suite become publicly used, which will gain the investigation work of big information system optimization and processing that is new development.

High Performance Computing: Supercomputers offer high performance when it comes to calculation, memory interaction and access. Classic programming that is parallel, such as for instance MPI, OpenMP, CUDA, and OpenCL, have performance benefits on the conventional big data pro-cessing tools, such as for instance MapReduce. Big data communities have already resorted to HPC for higher data processing efficiency and gratification that is real-time such as for instance DataMPI. The conjuncture of HPC and data being big draw more attentions.

Energy Efficiency: Since energy is increasingly more an concern that is essential big computing clusters, mini-mizing the vitality usage for every big information job is a critical issue which hasn't been commonly investigated. This paper discussed some work that is associated power ef-ficiency optimizations considering MapReduce. Big information processing systems with natively supported energy efficient features will undoubtedly be an appealing way that is future.

Large-Scale Machine Learning: Distributed ML systems on clusters are critical to using advanced ML algo-rithms at industrial scales for data analysis. The big ML models now can reach huge amounts of parameters with mas-sive quantity of data. ML models have distinct features, such as for example calculation dependencies and conver-gence that is uneven. We have discussed a few systems being representative large-scale ML. But, this considerable research direction continues to be in its very early phase. More features of ML must be supported and discovered. Moreover, common formalism for data/model parallelism is established being a guidance for future system development.

Other directions like large-scale computation debug-ging, domain-specific information processing, and general public assessment that is avail-able, are all promising research areas, deserving further exploration and development

CONCLUSION

Big data-processing systems happen widely researched by industry and academia. A study was presented with by this paper of the systems. On the basis of the processing paradigm, we categorize those functional systems into batch, stream, graph, and machine learning processing.. According to the deficiencies, we discussed the extensions and optimizations for MapReduce platform, including help for versatile dataflows, efficient data access and communication, parameter tuning, along with energy. We then surveyed other batch processing systems, including systems which can be general-purpose, Nephele/PACT and Spark. SQL-like systems tangled up in this paper are Hive, Shark, SCOPE, AsterixDB, and Dremel. For stream systems which are processing Storm and S4 are introduced as representatives. Scalability is amongst the ML algorithms bottlenecks. We then talked about how graphcentric systems like Pregel and GraphLab, and ML-centric systems like Petuum, parallelize the graph and ML model, as well as their characteristics which are distinctive. Future research opportunities are discussed at the final end associated with study.

REFERENCES

[1] A. Alexandrov et al., "Massively parallel data analysis with PACTs on Nephele," PVLDB, vol. 3, no. 2 pp. 1625–1628, 2010. doi: 10.14778/1920841.1921056.
[2] A. Alexandrov et al., "The Stratosphere platform for big data analytics," Int. J. Very Large Data Bases, vol. 23, no. 6, pp. 939–964, 2014.
[3] S. Alsubaiee1 et al., "Asterixdb: A scalable, open source bdms," Proc. VLDB Endowment, vol. 7, no. 14, 2014.
[4] P. Alvaro et al., "Boom analytics: Exploring data-centric, declarative programming for the cloud," in Proc. 5th Eur. Conf. Comput. Syst., Paris, France, Apr. 13–16, 2010, pp. 223–236, doi: 10.1145/1755913.1755937.
[5] AMPLab at UC Berkeley. MLbase, 2013. [Online] Available: http://www.mlbase.org
[6] AMPLab, UC Berkeley. Big Data Benchmark, 2014. [Online] Available: https://amplab.cs.berkeley.edu/benchmark/
[7] Apache. Hadoop, 2014. [Online] Available: http://hadoop.apache.org
[8] Apache. Flink, 2015. [Online] Available: http://flink.apache.org
[9] Apache. Hama, 2015. [Online] Available: https://hama.apache.org
[10] Apache. Mahout, 2015. [Online] Available: http://mahout.apache.org
[11] Apache. Tez, 2015. [Online] Available: http://tez.apache.org
[12] Apache. Thrift, 2015. [Online] Available: https://thrift.apache.org
[13] Apache. Yarn, 2015. [Online] Available: https://hadoop.apache.org/docs/current/ hadoop-yarn/hadoop-yarn-site/YARN.html
[14] Apache. Zookeeper, 2015. [Online] Available: https://zookeeper.apache.org
[15] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan, "Linkbench: A database benchmark based on the facebook social graph," K. A. Ross, D. Srivastava, and D. Papadias, eds., in Proc. ACM SIGMOD Int. Conf. Manage. Data, New York, NY, USA, Jun. 22–27, 2013, pp. 1185–1196, doi: 10.1145/2463676.2465296.
[16] D. Battre´ et al., "Nephele/PACTs: A programming model and execution framework for web-scale analytical processing," in Proc. 1st ACM Symp. Cloud Comput., Jun. 10–11, 2010, Indianapolis, IN, USA, pp. 119–130, doi: 10.1145/1807128.1807148.
[17] A. Behm et al., "ASTERIX: Towards a scalable, semistructured data platform for evolving-world models," Distrib. Parallel Databases, vol. 29, no. 3, pp. 185–216, 2011, doi: 10.1007/s10619-011-7082-y.
[18] B. Behzad et al., "Taming parallel I/O complexity with auto-tuning," in Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal., Denver, CO, USA, Nov. 17–21, 2013, p. 68, doi: 10.1145/2503210.2503278.
[19] B. Behzad, S. Byna, S. M. Wild, Prabhat, and M. Snir, "Improving parallel I/O autotuning with performance modeling," in Proc. 23rd Int. Symp. HPDC, Vancouver, BC, Canada, Jun. 23–27, 2014, pp. 253–256, doi: 10.1145/2600212.2600708.
[20] V. R. Borkar et al., "Hyracks: A flexible and extensible foundation for data-intensive computing," in Proc. 27th Int. Conf. Data Eng., Hannover, Germany, Apr. 11–16, 2011, pp. 1151–1162, doi: 10.1109/ICDE.2011.5767921. [21] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," Comput. Networks, vol. 30 no. 1–7, pp. 107–117, 1998, doi: 10.1016/S0169-7552 (98)00110-X.
[22] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "HaLoop: Efficient iterative data processing on large clusters," PVLDB, vol. 3, no. 1, pp. 285–296, 2010, doi 10.14778/1920841.1920881.
[23] R. Chaiken et al., "Scope: Easy and efficient parallel processing of massive data sets," Proc. VLDB Endow., vol. 1, no. 2, pp. 1265–1276, Aug. 2008, doi: 10.14778/1454159.1454166.
[24] F. Chang et al., "Bigtable: A distributed storage system for structured data (awarded best paper!)," in B. N. Bershad and J. C. Mogul, eds., Proc. 7th Symp. Operating Syst. Design Implementation, Seattle, WA, USA, Nov. 6–8, 2006, pp. 205–218, USENIX Association. [Online] Available: http://www.usenix.org/events/osdi06/tech/ chang.html
[25] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce online," in Proc. 7th USENIX Symp. Netw. Syst. Design Implementation, San Jose, CA, USA, Apr. 28–30, 2010, pp. 313–328. [Online] Available: http://www.usenix.org/events/ nsdi10/tech/full_papers/condie.pdf
[26] T. Condie et al., "Online aggregation and continuous query support in MapReduce," in Proc. ACM SIGMOD Int. Conf. Manage. Data, Indianapolis, IN, USA, Jun. 6–10, 2010, pp. 1115–1118, doi: 10.1145/1807167.1807295.
[27] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in J. M. Hellerstein, S. Chaudhuri, and M. Rosenblum, eds., Proc. 1st ACM Symp. Cloud Comput., Indianapolis, IN, USA, Jun. 10–11, 2010, pp. 143–154, doi 10.1145/1807128.1807152.
[28] D. Dahiphale et al., "An advanced mapreduce: Cloud mapreduce, enhancements and applications," IEEE Trans. Netw. Serv. Manage., vol. 11, no. 1, pp. 101–115, 2014, doi: 10.1109/TNSM.2014.031714.130407.

[29] D. J. DeWitt and M. Stonebraker, MapReduce: A Major Step Backwards, 2008. [Online] Available: https://homes.cs.washington.edu/billhowe/mapreduce_a_major_step_backwards.html
[30] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in Proc. 6th Symp. Operating Syst. Design Implementation, San Francisco, CA, USA, Dec. 6–8, 2004, pp. 137–150. USENIX Association. [Online] Available: http://www.usenix.org/events/osdi04/tech/dean.html
[31] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008, doi: 10.1145/1327452.1327492.
[32] J. Dean and S. Ghemawat, "MapReduce: A flexible data processing tool," Commun. ACM, vol. 53, no. 1, pp. 72–77, 2010, doi: 10.1145/1629175.1629198.
[33] D. DeWitt, MapReduce: A Major Step Backwards, 2008. [Online] Available: http://homes.cs.washington.edu/billhowe/mapreduce_a_major_step_backwards.html
[34] J. Dittrich et al., "Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing)," PVLDB, vol. 3, no. 1, pp. 518–529, 2010,
doi: 10.14778/1920841.1920908.
[35] J. Dittrich et al., "Only aggressive elephants are fast elephants," PVLDB, vol. 5, no. 11, pp. 1591–1602, 2012.
[36] P. Domingos, "A few useful things to know about machine learning," Commun. ACM, vol. 55, no. 10, pp. 78–87, 2012, doi: 10.1145/2347736.2347755.
[37] J. Ekanayake et al., "Twister: A runtime for iterative MapReduce," in Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput., Chicago, IL, USA, Jun. 21–25, 2010, pp. 810–818, doi: 10.1145/1851476.1851593.
[38] E. Elnikety, T. Elsayed, and H. E. Ramadan, "iHadoop: Asynchronous iterations for MapReduce," in Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci., Athens, Greece, Nov. 29–Dec. 1, 2011, pp. 81–90, doi: 10.1109/CloudCom.2011.21.
[39] M. Y. Eltabakh et al., "CoHadoop: Flexible data placement and its exploitation in Hadoop," PVLDB, vol. 4, no. 9, pp. 575–585, 2011, doi: 10.14778/2002938.2002943.
[40] E. Feller, L. Ramakrishnan, and C. Morin, "Performance and energy efficiency of big data applications in cloud environments: A Hadoop case study," J. Parallel Distrib. Comput., 2015, doi: 10.1016/j.jpdc.2015.01.001.
[41] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo, "SPADE: The systems declarative stream processing engine," in Proc. ACM SIGMOD Int. Conf. Manage. Data, Vancouver, BC, Canada, Jun. 10–12, 2008, pp. 1123–1134, doi: 10.1145/1376616.1376729.
[42] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in M. L. Scott and L. L. Peterson, eds., in Proc. 19th ACM Symp. Operating Syst. Principles, Bolton Landing, NY, USA, Oct. 19–22, 2003,
pp. 29–43, doi: 10.1145/945445.945450.
[43] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in Proc. 10th USENIX Symp. Operating Syst. Design Implementation, Hollywood, CA, USA, Oct. 8–10, 2012, pp. 17–30.
[44] Y. He et al., "Rcfile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems," in Proc. 27th Int. Conf. Data Eng., Hannover, Germany, Apr. 11–16, 2011, pp. 1199–1208, doi: 10.1109/ICDE.2011.5767933. [45] A. Heise, A. Rheinla¨nder, M. Leich, U. Leser, and F. Naumann, "Meteor/sopremo: An extensible query language and operator model," in Proc. Workshop End-to-end Manage. Big Data, Istanbul, Turkey, 2012.
[46] H. Herodotou and S. Babu, "Profiling, what-if analysis, and cost-based optimization of MapReduce programs," PVLDB, vol. 4, no. 11, pp. 1111–1122, 2011. [Online] Available: http://www.vldb.org/pvldb/vol4/ p1111-herodotou.pdf
[47] H. Herodotou et al., "Starfish: A self-tuning system for big data analytics," in Proc. 5th Biennial Conf. Innovative Data Syst. Research, Asilomar, CA, USA, Jan. 9–12, 2011, pp. 261–272. [Online] Available: www. cidrdb.org; http://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper36.pdf
[48] T. Hey, S. Tansley, and K. M. Tolle, Eds., The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft Res., 2009. [Online]
Available: http://research.microsoft.com/enus/collaboration/fourthparadigm/
[49] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in Proc. IEEE 26th Int. Conf., 2010, pp. 41–51.
[50] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in Proc. 2nd ACM SIGOPS/EuroSys Eur. Conf. Comput. Syst., pp. 59–72, New York, NY, USA, 2007, doi: 10.1145/1272996.1273005.
[51] I. King M. R. Lyu, J. Zeng, and H. Yang, "A comparison of lasso-type algorithms on distributed parallel machine learning platforms," in Proc. Workshop Distrib. Mach. Learning Matrix Comput., Montreal, QC, Canada, 2014.
[52] V. Kumar, H. Andrade, B. Gedik, and K.-L. Wu, "DEDUCE: At the intersection of MapReduce and stream processing," in Proc. 13th Int. Conf. Extending Database Technol., Lausanne, Switzerland, Mar. 22–26, 2010, vol. 426, pp. 657–662, doi: 10.1145/1739041.1739120.
[53] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," Operating Syst. Rev., vol. 44, no. 2, pp. 35–40, 2010, doi: 10.1145/ 1773912.1773922.
[54] W. Lang and J. M. Patel, "Energy management for MapReduce clusters," PVLDB, vol. 3, no. 1, pp. 129–139, 2010. [Online] Available: http://www.comp.nus.edu.sg/vldb2010/proceedings/files/papers/ R11.pdf
[55] K.-H. Lee et al., "Parallel data processing with MapReduce: A survey," SIGMOD Rec., vol. 40, no. 4, pp. 11–20, 2011, doi:10.1145/2094114.2094118.
[56] J. Leverich and C. Kozyrakis, "On the energy (in)efficiency of hadoop clusters," Operating Syst. Rev., vol. 44, no. 1, pp. 61–65, 2010, doi: 10.1145/1740390.1740405.
[57] F. Li, B. C. Ooi, M. Tamer O¨ zsu, and S. Wu, "Distributed data management using mapreduce," ACM Comput. Surv., vol. 46, no. 3, p. 31, 2014, doi: 10.1145/2503009.
[58] F. Liang, C. Feng, X. Lu, and Z. Xu, Performance Benefits of DataMPI: A Case Study with BigDataBench, Lecture Notes in Computer Science, vol. 8807. Springer Int. Publishing, 2014, doi 10.1007/978-3-319- 13021-7_9.
[59] H. Lim, H. Herodotou, and S. Babu, "Stubby: A transformation-based optimizer for MapReduce workflows," PVLDB, vol. 5, no. 11, pp. 1196–1207, 2012, doi: 10.14778/ 2350229.2350239.
[60] Y. Lin, D. Agrawal, C. Chen, B. C. Ooi, and S. Wu, "Llama: Leveraging columnar storage for scalable join processing in the MapReduce framework," in Proc. ACM SIGMOD Int. Conf. Manage. Data, Athens, Greece, Jun. 12–16, 2011, pp. 961–972, doi: 10.1145/1989323.1989424.
[61] B. T. Loo et al., "Implementing declarative overlays," in Proc. 20th ACM Symp. Operating Syst. Principles, Brighton, U.K., Oct. 23–26, 2005, pp. 75–90, doi: 10.1145/ 1095810.1095818.
[62] Y. Low et al., "Distributed graphlab: A framework for machine learning in the cloud," PVLDB, vol. 5, no. 8, pp. 716–727, 2012. [Online] Available: http://vldb.org/pvldb/vol5/p716_yuchenglow_vldb2012.pdf
[63] X. Lu, F. Liang, B. Wang, L. Zha, and Z. Xu, "DataMPI: Extending MPI to Hadoop-like big data computing," in Proc. IEEE 28th Int. Parallel Distrib. Process. Symp., Phoenix, AZ, USA, May 19–23, 2014, pp. 829–838, doi: 10.1109/IPDPS.2014.90.
[64] G. Malewicz et al., "Pregel: A system for large-scale graph processing," in A. K. Elmagarmid and D. Agrawal, Eds., in Proc. ACM SIGMOD Int. Conf. Manage. Data, Indianapolis, IN, USA, Jun. 6–10, 2010, pp. 135–146, doi: 10.1145/1807167.1807184.
[65] L. Mashayekhy, M. M. Nejad, D. Grosu, D. Lu, and W. Shi, "Energy-aware scheduling of MapReduce jobs," in Proc. IEEE Int.

Congress Big Data, Anchorage, AK, USA, Jun. 27–Jul. 2, 2014, pp. 32–39, doi: 10.1109/BigData.Congress.2014.15.
[66] S. Melnik et al., "Dremel: Interactive analysis of web-scale datasets," Proc. VLDB Endowment, vol. 3, no. 1/2, pp. 330–339, 2010.
[67] MongoDB Inc. Mongodb, 2016. [Online] Available: https://www.mongodb.com
[68] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in W. Fan, W. Hsu, G. I. Webb, B. Liu, C. Zhang, D. Gunopulos, and X. Wu, eds., in Proc. 10th IEEE Int. Conf. Data Mining Workshops, Sydney, Australia, Dec. 13, 2010, pp. 170–177, doi: 10.1109/ICDMW.2010.172.
[69] D. A. Patterson, "Technical perspective: The data center is the computer," Commun. ACM, vol. 51, no. 1, p. 105, 2008, doi: 10.1145/1327452.1327491.
[70] A. Pavlo et al., "A comparison of approaches to large-scale data analysis," in Proc. ACM SIGMOD Int. Conf. Manage. Data, Providence,
RI, USA, Jun. 29–Jul. 2, 2009, pp. 165–178, 2009, doi: 10.1145/1559845.1559865.
[71] M. Piszczalski, Locating Data Centers in an Energy-Constrained World, May 2012. [Online] Available: http://pro.gigaom.com/ 2012/05/locating-data-centers-in-an-energyconstrained- world/
[72] I. Polato, R. Re´, A. Goldman, and F. Kon, "A comprehensive view of hadoop research—A systematic literature review," J. Netw. Comput. Appl., vol. 46, pp. 1–25, 2014, doi: 10.1016/j.jnca.2014.07.022.
[73] S. Sakr, A. Liu, and A. G. Fayoumi, "The family of MapReduce and large-scale data processing systems," ACM Comput. Surv., vol. 46, no. 1, p. 11, 2013,doi: 10.1145/2522968.2522979.
[74] M. Stonebraker et al., "MapReduce and parallel DBMSs: Friends or foes?" Commun. ACM, vol. 53, no. 1, pp. 64–71, 2010, doi: 10.1145/1629175.1629197.
[75] A. Thusoo et al., "Hive—A petabyte scale data warehouse using Hadoop," in Proc. 26th Int. Conf. Data Eng., Long Beach, CA, USA, Mar. 1–6, 2010, pp. 996–1005, doi: 10.1109/ICDE.2010.5447738.
[76] A. Toshniwal et al., "Storm@twitter," inProc. SIGMOD Int. Conf. Manage. Data, 2014, pp. 147–156, doi: 10.1145/2588555.2595641.
[77] R. Tudoran, A. Costan, and G. Antoniu, "Mapiterativereduce: A framework for reduction-intensive data processing on azure clouds," in Proc. 3rd Int. Workshop MapReduce Appl. Date, 2012, New York, NY, USA, pp. 9–16, doi: 10.1145/2287016.2287019.
[78] L. Wang et al., "Bigdatabench: A big data benchmark suite from internet services," in Proc. 20th IEEE Int. Symp. High Perform. Comput. Architecture, Orlando, FL, USA, Feb. 15–19, 2014, pp. 488–499, doi: 10.1109/HPCA.2014.6835958.
[79] T. Wirtz and R. Ge, "Improving MapReduce energy efficiency for computation intensive workloads," in Proc. Int. Green Comput. Conf. Workshops, Orlando, FL, USA, Jul. 25–28, 2011, pp. 1–8, doi:10.1109/IGCC.2011.6008564.
[80] R. S. Xin et al., "Shark: Sql and rich analytics at scale," in Proc. ACM SIGMOD Int. Conf. Manage. Fata, 2013, pp. 13–24.
[81] E. P. Xing et al., "Petuum: A new platform for distributed machine learning on big data," in Proc. 21th ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining, Sydney, NSW, Australia, Aug. 10–13, 2015, pp. 1335–1344, doi: 10.1145/ 2783258.2783323.
[82] Y. Yu et al., "Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language," in Proc. 8th USENIX Conf. Operating Syst. Design Implementation, 2008, Berkeley, CA, USA, pp. 1–14. [Online] Available: http://dl. acm.org/citation.cfm?id = 1855741.1855742
[83] M. Zaharia, N. M. Mosharaf Chowdhury, M. Franklin, S. Shenker, and I. Stoica, Spark: Cluster Computing With Working Sets, Tech. Rep. UCB/EECS-2010-53, Dept. Electr. Eng. Comput. Syst., Univ. California, Berkeley, May 2010. [Online] Available: http://www.eecs. berkeley.edu/Pubs/TechRpts/2010/EECS- 2010-53.html
[84] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "iMapReduce: A distributed computing framework for iterative computation," J. Grid Comput., 10 vol. 1, pp. 47–68, 2012, doi: 10.1007/s10723-012-9204-9.
[85] I. A. T. Hashema, I. Yaqooba, N. B. Anuara, S. Mokhtara, A. Gania, and S. U. Khanb, "The rise of "big data" on cloud computing: Review and open research issues," Inf. Syst., vol. 47, pp. 98–115, Jan. 2015, doi: 10.1016/ j.is.2014.07.006.