Large-scale Data Analysis on the Cloud

Ranieri Baraglia[1], Claudio Lucchese[1], and Gianmarco De Francisci Morales[1,2]

[1] ISTI-CNR, Pisa, Italy
[2] IMT - Institute for Advanced Studies, Lucca, Italy

Abstract. A "data deluge" is currently drowning us. Data sources are everywhere, from Web 2.0 to large scientific experiments, from social networks to sensors. This huge amount of data is a mine hiding valuable information. Nowadays, the extraction of knowledge from such ocean is a competitive advantage for most companies. Cloud computing is the emerging technology for large scale data analysis, providing scalability to thousands of computers, in addition to fault tolerance and cost effectiveness.

The current solutions in the field of cloud computing provide transparent access to large scale computing and storage hardware by exposing two main functionalities to the users: distributed data management and distributed computing. Both the two layers provide simple access and an high level of fault tolerance. The system automatically handles multiple distributed copies of any given file, and it monitors every running job possibly re-executing it in case of failure. On the flip side of the coin, the data level is designed for unstructured data and for parallel scans, while the computing level supports novel and completely different programming models.

We will discuss the emerging technologies by exploring these two layers. In particular, many big players such as Google, Yahoo!, Microsoft, Amazon, already adopted this framework. We will describe every single approach and highlight the strong and weak points of each proposal. Finally, we will discuss the open problems, illustrate some novel algorithms and solutions, and devise a research roadmap in the field of cloud computing.

1 The *"Big Data"* Problem and the *Cloud Computing* Solution

Currently, an incredible *"data deluge"* is drowning the world with data. The quantity of data we need to sift through every day is enormous, just think of the results of a search engine query. They are so many that we are not able to examine all of them, and indeed the competition on web search now focuses on ranking the results. This is just an example of a more general trend.

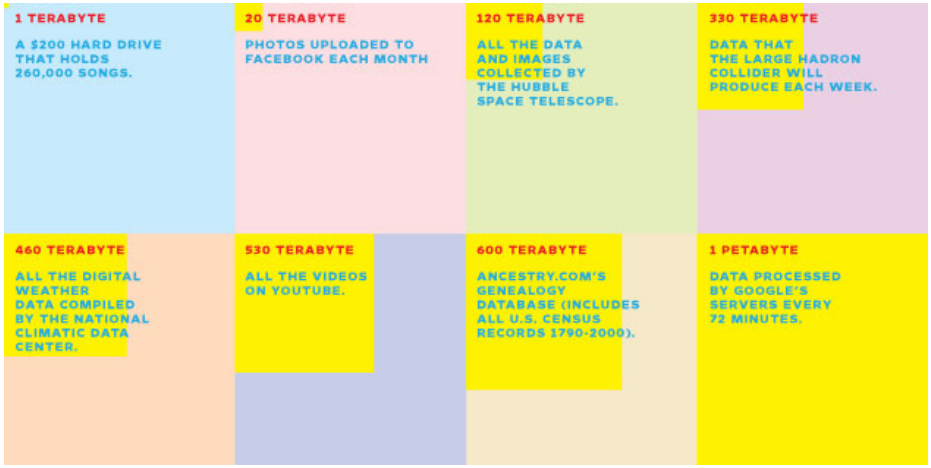

Fig. 1. The Petabyte Age

Data sources are everywhere. Web users produce vast amounts of text, audio and video contents in the so called *Web 2.0*. Relationships and tags in social networks create large graphs spanning millions of vertexes and edges. Scientific experiments are another huge data source. The Large Hadron Collider (LHC) at European Council for Nuclear Research (CERN) is expected to generate around 50 TB of raw data per day. The Hubble telescope captured hundreds of thousands astronomical images, each hundreds of megabytes large. Computational biology experiments like high-throughput genome sequencing produce large quantities of data that require extensive post-processing. In the future, sensors like Radio-Frequency Identification (RFID) tags and Global Positioning System (GPS) receivers will spread everywhere. These sensors will produce petabytes of data just as a result of their sheer numbers, thus starting *"the industrial revolution of data"* [19].

The issues raised by large data sets in the context of analytical applications are becoming ever more important as we enter the *"Petabyte Age"*. Figure 1 shows some of the data sizes for today's problems [3]. The data sets are orders of magnitude greater than what fits on a single hard drive, and their management

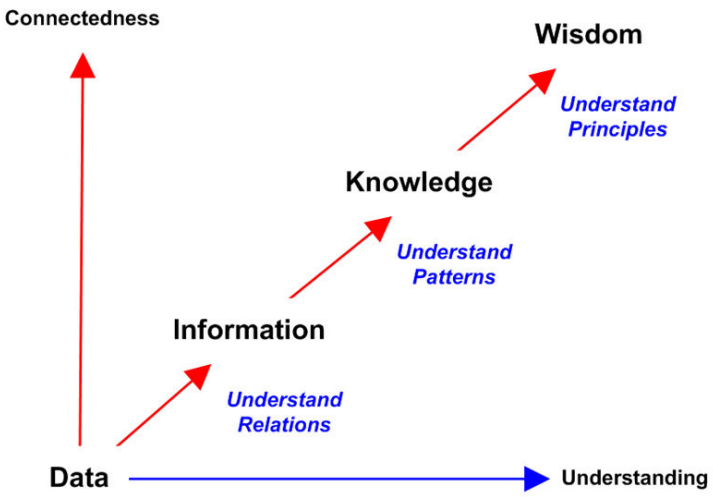

Fig. 2. Data Information Knowledge Wisdom hierarchy

poses a serious challenge. Web companies are currently facing this issue, striving to find efficient solutions. The ability to analyze or manage more data is a distinct competitive advantage. This problem has been labeled in various ways: petabyte scale, web scale or *"big data"*.

But why are we interested in data? It is common belief that data without a model is just noise. Models are used to describe salient features in the data, which can be extracted via data analysis. Figure 2 represents the popular Data Information Knowledge Wisdom (DIKW) hierarchy [36]. In this hierarchy data stands at the lowest level and bears the smallest level of *understanding*. Data needs to be processed and condensed into more connected forms in order to be useful for event comprehension and decision making. Information, knowledge and wisdom are these forms of understanding. Relations and patterns that allow to gain deeper awareness of the process that generated the data, and principles that can guide future decisions.

For data analysis, scaling up of data sets is a *"double edged"* sword. On the one hand, it is an opportunity because *"no data is like more data"*. Deeper insights are possible when more data is available. On the other hand, it is a challenge. Current methodologies are often not suitable to handle very large data sets, so new solutions are needed.

Unfortunately, traditional Database Management System (DBMS) are not able to handle data at such a large scale. Therefore, the only solution to this problem is to exploit parallelism, in terms of a large number of machines exposing huge storage and computing power.

Parallel computing has a long history. It has traditionally focused on *"number crunching"*. Common applications were tightly coupled and CPU intensive (e.g. large simulations or finite element analysis). These systems are notoriously hard to program, fault tolerance is difficult to achieve and scalability is an art. In contrast with this legacy, a new class of parallel systems has emerged: cloud computing.Cloud computing is the result of the convergence of three technologies: grid computing, virtualization and Service Oriented Architecture (SOA). The aim of cloud computing is thus to offer services on a virtualized parallel back-end system. These services are divided in categories according to the resource they offer: Infrastructure as a Service (IaaS) like Amazon's EC2 and S3, Platform as a Service (PaaS) like Google's App Engine and Microsoft's Azure Services Platform, Software as a Service (SaaS) like Salesforce, OnLive and virtually every Web application.

Lately, cloud computing has received a substantial amount of attention from industry, academia and press. As a result, the term "cloud computing" has become a buzzword, overloaded with meanings. Even without a clear definition, there are some properties that we think a cloud computing system should have. Each of the three aforementioned technologies brings some feature to cloud computing. According to SOA principles, a cloud system should be a distributed system, with separate, loosely coupled entities collaborating among each other. Virtualization (not intended just as x86 virtualization but as a general abstraction of computing, storage and communication facilities) provides for location, replication and failure transparency. Finally, grid computing endorses scalability.

Indeed, cloud systems focus on being scale-free, fault tolerant, cost effective and easy to use.

2 Technology Overview

Large scale data challenges have spurred a large number of projects on data oriented cloud computing systems. Most of these projects feature important industrial partners alongside academic institutions. Big internet companies are the most interested in finding novel methods to manage and use data. Hence, it is not surprising that Google, Yahoo!, Microsoft and few others are leading the trend in this area.

We propose a general architecture of cloud computing systems, that captures the commonalities in the form of a multi-layered stack. Table 1 reports some of the most popular cloud computing softwares classified according to the proposed multi-layered stack architecture.

In the lowest layer, the coordination layer, we find two implementations of a consensus algorithm. Chubby [5] is a distributed implementation of Paxos [26] and Zookeeper is Hadoop's re-implementation in Java. They are centralized services for maintaining configuration information, naming, providing distributed synchronization and group services. All these kinds of services are used by distributed applications.

Layers	Google	Yahoo!	Microsoft	Others
High Level Languages	Sawzall	Pig Latin	DryadLINQ	Hive Cascading
Computation	MapReduce	Hadoop	Dryad	
Data Abstraction	BigTable	HBase PNUTS		Cassandra Voldemort
Distributed Data	GFS	HDFS	Cosmos	Dynamo
Coordination	Chubby	Zookeeper		

Table 1. Major cloud software stacks

On the next level, the distributed data layer presents different kinds of data storages. A common feature in this layer is to avoid full POSIX semantic in favor of simpler ones. Furthermore, consistency is somewhat relaxed for the sake of performance.

Google File System (GFS) [17], Hadoop Distributed File System (HDFS) and Cosmos [6] are all distributed file systems geared towards data streaming. They are not general purpose file systems. For example, in HDFS files can only be appended but not modified. They use large blocks of 64 MB or more, which are replicated for fault tolerance. GFS uses a master-slave architecture where the master is responsible for metadata operations while the slaves (chunkservers) are used for data. Cosmos is Microsoft's internal file system for cloud computing. Dynamo [14] is a storage service used at Amazon. It has a peer-to-peer (P2P) architecture that allows to store a key-value pairs and achieves very low latency access to data.

BigTable [7] and HBase [4] are non-relational databases. They are actually multidimensional, sparse, sorted maps, useful for semi-structured or non structured data. As in many other similar systems, access to data is provided via primary key only, but each key can have more "columns". These systems provide efficient read/write access built on top of their respective file system (GFS and HDFS) with support for versioning, timestamps and single row transactions. Google Earth, Google Analytics and many other user-facing services use this kind of storage.

PNUTS [12] is a similar storage service developed by Yahoo! outside the Hadoop project. Cassandra [16] is a Facebook project (now open-sourced into Apache incubator) that aims to build a system with a BigTable-like interface on a Dynamo-style infrastructure. Voldemort [28] is an open-source non relational database built by LinkedIn. It is basically a large persistent Distributed Hash Table (DHT). This area has been very fruitful and there are many other products, but all of them share in some way the features sketched in these paragraphs: distributed architecture, non relational model, no ACID guarantees, limited relational operations (no join).

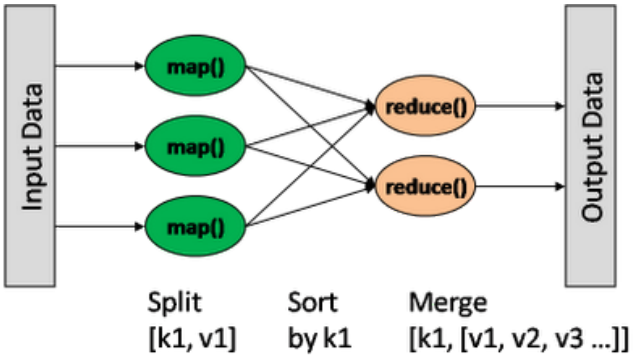

Fig. 3. The MapReduce programming model

In the computation layer we find paradigms for large scale data intensive computing. They are mainly dataflow paradigms with support for automated parallelization. We can recognize the same pattern found in previous layers also here: trade off generality for performance.

MapReduce (MR) [13] is a distributed computing engine inspired by concepts of functional languages. More specifically, MR is based on two higher order functions: Map and Reduce. The Map function reads the input as a list of key-value pairs and applies a User Defined Function (UDF) to each pair. The result is a second list of intermediate key-value pairs. This list is sorted and grouped by key and used as input to the Reduce function. The Reduce function applies a second UDF to every intermediate key with all its associated values to produce the final result. The two phases are non overlapping as detailed in Figure 3.

The Map and Reduce function are purely functional and thus without side effects. This is the reason why they are easily parallelizable. Furthermore, fault tolerance is easily achieved by just re-executing the failed function. The programming interface is easy to use and does not allow any explicit control of parallelism. Even though the paradigm is not general purpose, many interesting algorithms can be implemented on it. The most paradigmatic application is building the inverted index for Google's search engine. Simplistically, the crawled and filtered web documents are read from GFS, and for every word the couple $\langle word, doc\_id \rangle$ is emitted in the Map phase. The Reduce phase needs just to sort all the document identifiers associated with the same word $\langle word, [doc\_id1, doc\_id2, ...] \rangle$ to create the corresponding posting list.

Hadoop [4] is a Java implementation of MapReduce, which is roughly equivalent to Google's version, even though there are many reports about the performance superiority of the latter [18]. Like the original MR, it uses a master-slave architecture.

The Job Tracker is the master to which client applications submit MR jobs. It pushes tasks (job splits) out to available slave nodes on which a Task Tracker runs. The slave nodes are also HDFS chunkservers, and the Job Tracker knows

which node contains the data. The Job Tracker thus strives to keep the jobs as close to the data as possible. With a rack-aware filesystem, if the work cannot be hosted on the actual node where the data resides, priority is given to nodes in the same rack. This reduces network traffic on the main backbone network for the Map phase.

Mappers write intermediate values locally on disk. Each reducer in turn pulls the data from various remote disks via HTTP. The partitions are already sorted by key by the mappers, so the reducer just merge-sorts the different partitions to bring the same keys together. These two phases are called shuffle and sort and are also the most expensive in terms of I/O operations. In the last phase the reducer can finally apply the Reduce function and write the output to HDFS.

Dryad [21] is Microsoft's alternative to MapReduce. Program specification is done by building a Direct Acyclic Graph (DAG) whose vertexes are operations and whose edges are data channels.

At the last level we find high level interfaces to these computing systems. Sawzall [35] is a high level, parallel data processing scripting language built on top of MR. Its intended target are filter and aggregation scripts of record sets, akin to the AWK scripting language. It forces the programmer to think one record at a time, and allows the system to massively parallelize the computation without any programming effort.

Pig Latin [33] is a more sophisticated language for general data manipulation. It is an interpreted imperative language which is able to perform filtering, aggregation, joining and other complex transformations. DryadLINQ [40] is a set of language extensions and a corresponding compiler. The compiler trasforms Language Integrated Query (LINQ) expressions into Dryad plans. Hive [38] is a Facebook project to build a data warehousing system on top of Hadoop. Finally, Cascading [10] is a Java API for creating complex and fault tolerant data processing workflows on top of Hadoop.

2.1 Case Studies

Data analysis researchers have seen in cloud computing the perfect tool to run their computations on huge data sets. In the literature, there are many examples of successful applications of the cloud paradigm in different areas of data analysis.

Information retrieval has been the classical area of application for cloud technologies. Indexing is a representative application of this field, as the original purpose of MR was to build Google's index for web search. In order to take advantage of increased hardware and input sizes, novel adaptations of single-pass indexing have also been proposed [29].

Pairwise document similarity is a common tool for a variety of problems such as clustering and cross-document coreference resolution. When the corpus is large, MR is convenient because it allows to efficiently decompose the computation[15].

Frequent itemset mining is commonly used for query recomendation. When the data set size is huge, both the memory use and the computational cost can be prohibitively expensive [27].

Machine learning has been a fertile ground for cloud computing applications. For instance, MR has been employed for the prediction of user rating of movies, based on accumulated rating data [8]. In general, many widely used machine learning algorithms can be expressed in terms of MR [9].

Graph analysis is a common and useful task. Graphs are ubiquitous and can be used to represent a number of real world structures, e.g. networks, roads and relationships. The size of the graphs of interest has been rapidly increasing in recent years. Estimating the graph diameter for graphs with billions of nodes (e.g. the Web) is a challenging task, which can benefit from cloud computing [22].

Co-clustering is a data mining technique which allows simultaneous clustering of the rows and columns of a matrix. Co-clustering searches for sub-matrices of rows and columns that are interrelated. Even though powerful, co-clustering is not practical to apply on large matrices with several millions of rows and columns. A distributed co-clustering solution that uses Hadoop has been proposed to address this issue [34].

A number of different graph mining tasks can be unified via a generalization of matrix-vector multiplication. These tasks can benefit from a single highly optimized implementation of the multiplication. Starting from this observation, a library built on top of Hadoop has been proposed as a way to perform easily and efficiently large scale graph mining operations [23].

Cloud technologies have also been applied to other fields, such as scientific simulations of earthquakes [8], collaborative web filtering [31] and bioinformatics [37].

3 Open Problems

The systems described in the previous sections are fully functional and commonly used in production environment. Nonetheless, many research efforts have been made in order to improve and evolve them. Most of the efforts have focused on MapReduce, also because of the availability and success of Hadoop. The research in this area can be divided in three categories: computational models, programming paradigm enrichments and online analytics.

3.1 Computational Models

The design of efficient algorithms through the MapReduce paradigm, is still black magic left to the intuition and experience of skilled programmers. In order to have a rigorous approach, that goes beyond the trial-and-error process, some theoretical framework must be devised. Indeed, lots of efforts are being delivered towards the definition of a computational model for MapReduce.

A computational model would allow to accurate estimate the cost incurred by a MapReduce algorithm. This is not a trivial task, since the performance of a MapReduce algorithm is affected by several interdependent factors. These include: (1) disk access at the beginning of the MapReduce job, between the Map

and the Reduce stages, and between different MapReduce jobs, (2) communication costs, mainly between the Map and the Reduce stage, and (3) parallel computation and load imbalance.

Some interesting models were proposed by [2] and [24]. Both of them force a few assumptions on the availability of memory, storage, cpu and on the communication costs between the Map and the Reduce stage. These models are applied to well known problems, such as sorting, database joins, or string processing. Both of the two can be profitably used to improve the design of novel MapReduce algorithms.

Finally, [25] developed an interesting functional model of MapReduce based on Haskell. Indeed the MapReduce paradigm is borrowed from functional programming languages. Their analysis shows various interesting points. First, the authors disambiguate the type definition of MR which is very vague in the original paper. They decompose MR in three phases highlighting the shuffling and grouping operation that is usually performed silently by the system.

Then, they analyze the combiner function of MR and find that it is actually useless. If it defines a function that is logically different from the reducer there is no guarantee of correctness. Otherwise there is no need to define two separate functions for the same functionality.

The three studies mentioned above, highlight some of the limitations of the current implementations of the MapReduce paradigm. Some of these weaknesses are given by the need to store data on disk after each Map or Reduce, or to wait every Map task to be completed before starting the Reduce stage, or the redundancy in the combiner tasks. Improved models are needed both to better characterize MapReduce jobs and to identify the possible improvements to the MapReduce framework itself.

3.2 Programming Paradigm Enrichments

A number of extensions to the base MR system have been developed. These works differentiate from the models above because their result is a working system. Most of these works focus on extending MR towards the database area.

[39] proposes an extension to MR in order to simplify the implementation of relational operators. Namely, the implementation of join, complex, multi-table select and set operations. The normal MR workflow is extended with a third final Merge phase. This function takes in input two different key-value pair lists and outputs a third key-value pair list. The model assumes that the Reduce produces key-value pairs that are then fed to the Merge. The signature of the functions are as follows:

$$\begin{aligned} map: & \quad (k_1, v_1)_\alpha & \rightarrow [(k_2, v_2)]_\alpha \\ reduce: & \quad (k_2, [v_2])_\alpha & \rightarrow (k_2, [v_3])_\alpha \\ merge: & \ ((k_2, [v_3])_\alpha, (k_3, [v_4])_\beta) & \rightarrow [(k_4, v_5)]_\gamma \end{aligned}$$

where α, β, γ represent data lineages, k is a key and v is a value. The lineage is used to distinguish the source of the data, a necessary feature for joins.

Another attempt to integrate database feature into MR is proposed by [1]. Their HadoopDB is an architectural hybrid between Hadoop and a DBMS (PostgreSQL). The goal is to achieve the flexibility and fault tolerance of Hadoop and the performance and query interface of a DBMS. The architecture of HadoopDB comprises single instances of PostgreSQL as the data storage (instead of HDFS), and Hadoop as the communication layer

HadoopDB is an interesting attempt to join distributed systems and databases. Nonetheless, we argue that reusing principles of database research rather than its artifacts should be the preferred method.

3.3 Online Analytics

A different research direction aims to enable online analytics on large scale data. This gives substantial competitive advantages in adapting to changes, and the ability to process stream data. Systems like MR are essentially batch systems, while BigTable provides low latency but is just a lookup table. To date there is no system that provides low latency general querying.

[32] describe an approach to interactive analysis of web-scale data. The scenario entails interactive processing of a single negotiated query over static data. Their idea is to split the analysis phase in two: an offline and an online phase. In the former, the user submits a template to the system. This template is basically a parameterized query plan. The system examines the template, optimizes it and computes all the necessary auxiliary data structures to speed up query evaluation. In the process it may negotiate restrictions on the template. In the online phase the user instantiates the template by binding the parameters into the template. The system computes the final answer using the auxiliary structures in "real-time".

The offline phase works in a batch environment with large computing resources (i.e. a MR-style system). The online phase may alternatively be run on a single workstation, if enough data reduction happens in the offline phase. The authors focus on parameterized filters as examples and show how to optimize the query plan for the 2-phase split. They try to push all the parameter-free operations in the offline phase, and create indexes or materialized views at phase junctures. Various types of indexing approaches are then evaluated, together with other background optimization techniques. This work looks promising and there are still questions to be answered like what kind of templates are amenable to interactivity, how to help the user building the query template and how to introduce approximate query processing techniques (e.g. online aggregation [20]).

On this last issue we find significant a work by [11]. They modify Hadoop in order to pipeline data between operators, support online aggregation and continuous queries. They name their system Hadoop Online Prototype (HOP). To implement pipelining the simple pull interface between Reduce and Map has to be modified into a hybrid push/pull interface. The mappers push data to reducers in order to speed up the shuffle and sort phase. The pushed data is

treated ad tentative to retain fault tolerance: if one of the entities involved in the transfer fails the data is simply discarded. For this reason mappers also write the data to disk as in normal MR, and the pull interface is retained.

Online aggregation is supported by simply applying the reduce function to all the pipelined data received so far. This aggregation can be triggered on specific events (e.g. 50% of the mappers completed). The result is a snapshot of the computation and can be accessed via HDFS. For online aggregation the problem is that the output of Reduce on 50% of the input is not directly related to the final output. This means that every snapshot must be recomputed from scratch, using considerable computing resources. This problem can be alleviated for Reduce functions that are declared to be distributive, associative or algebraic aggregate. This line of research is extremely promising and some of the shortcomings of this work are directly addressable. For example we can enable overlapping the computation for a Reduce-to-Map pipelining. Moreover, we could use semantic clues to specify properties about the functions or the input. This could easily be implemented using Java annotations.

4 A research agenda

Cloud computing is an emerging technology in the data analysis field, used to capitalize on very large data sets. "There is no data like more data" is a famous motto that epitomizes the opportunity to extract significant information by exploiting huge volumes of data. Information represents a competitive advantage for companies operating in the information society, an advantage that is all the greater the sooner it is achieved. In the limit, online or real-time analytics will become an invaluable support for decision making. To date, cloud computing technologies have been successfully employed for batch processing. Adapting these systems to online analytics is a challenging problem, mainly because of the fundamental design choices that favor throughput over latency [30]. This research field is new and rapidly evolving. As a result, the initial efforts are not organic. We feel the need of providing a coherent research agenda in this field.

Many data analysis algorithms spanning different application areas have been proposed for cloud systems. So far, speedup and scalability results have been encouraging. Still, it is not clear in the research community which problems are a good match for cloud systems. More importantly, the ingredients and recipes for building a successful algorithm are still hidden. A library of *"algorithmic design patterns"* would enable faster and easier adoption of cloud systems. Even though there have been attempts to address these issues, current research efforts in this area have been mostly "one shot". The results are promising but their scope and generality is often limited to the specific problem at hand. These efforts also lack a coherent vision and a systematic approach. This "high entropy" level is typical of new research areas whose agenda is not yet clear. One of the main shortcomings is that the available models for cloud systems are at an embryonal stage of development. This hampers the comparison of algorithms and the evaluation of their properties from a theoretical point of view.

Furthermore, up to now industry has been leading the research in cloud computing. Academia has now to follow and "catch-up". Nonetheless this fact also has positive implications. The research results are easily adapted to industrial environments because the gap between production and research systems is small.

In this work, we have illustrated three main directions deserving great efforts from the research community. First, it is necessary to selectively and coherently integrate into cloud systems *principles* from database research, as opposed to its *artifacts*. There is a strong need for the development of a comprehensive approach to integration, by picking relevant principles from the database literature, and adapting them to the new context.

Second, a novel computational model for cloud computing is needed. One the one hand, this will provide the necessary tools for the analysis of algorithms and systems. On the other hand, this will be able to provide a common reference ground for the comparison of the available solutions, algorithms and paradigms.

Finally, given the relevance of interactive and online analytics, new approximate techniques can be designed, such as sampling, by trading time with precision. Providing confidence intervals for approximate answers is an important feature, missing from current systems, that needs to be explored. Also, skewness of the data distribution, must be taken into account, since it often leads to *straggler* tasks that slow down the entire job.

We believe that in the next future, a number of extensions to MapReduce and to the MapReduce paradigm will be proposed in order to support online analytics and other common computation patterns. This will conceivably lead to a new paradigm, an evolution beyond the original MapReduce.

In conclusion, large-scale data analysis and cloud computing are becoming a meeting point for the distributed systems and database communities. Cross-contamination of research areas is a great stimulus for scientific advancement. This field raises new scientific and technological challenges that will step forward the state-of-art, and that will transform the way we interact with the ever-changing Web.

Bibliography

[1] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. In *Proceedings of the VLDB Endowment*, volume 2, pages 922–933, August 2009.

[2] F.N. Afrati and J.D. Ullman. A New Computation Model for Rack-Based Computing. Submitted to PODS 2010: Symposium on Principles of Database Systems, 2009.

[3] C. Anderson. The Petabyte Age: Because more isn't just more—more is different. *Wired*, 16(07), July 2008.

[4] Apache Software Foundation. Hadoop: A framework for running applications on large clusters built of commodity hardware, 2006.

[5] Mike Burrows. The Chubby lock service for loosely-coupled distributed systems. In *OSDI '06: Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, pages 335–350, November 2006.

[6] R. Chaiken, B. Jenkins, P.Å. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: Easy and efficient parallel processing of massive data sets. In *Proceedings of the VLDB Endowment*, volume 1, pages 1265–1276. VLDB Endowment, August 2008.

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. BigTable: A distributed storage system for structured data. In *OSDI '06: Proceedings of the 7th Symposium on Operating systems Design and Implementation*, pages 205–218, November 2006.

[8] S. Chen and S.W. Schlosser. Map-Reduce Meets Wider Varieties of Applications. Technical Report IRP-TR-08-05, Intel Research, Pittsburgh, May 2008.

[9] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-Reduce for Machine Learning on Multicore. In *Advances in Neural Information Processing Systems*, volume 19, pages 281–288. MIT Press, 2007.

[10] Concurrent Inc. Cascading: An API for executing workflows on Hadoop, January 2008.

[11] T. Condie, N. Conway, P. Alvaro, J.M. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce Online. Technical Report UCB/EECS-2009-136, University of California, Berkeley, October 2009.

[12] B.F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: Yahoo!'s hosted data serving platform. In *Proceedings of the VLDB Endowment*, volume 1, pages 1277–1288. VLDB Endowment, August 2008.

[13] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI '04: Proceedings of the 6th Symposium on Opearting Systems Design and Implementation*, pages 137–150. USENIX Association, December 2004.

[14] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *SOSP '07: Proceedings of 21st ACM SIGOPS symposium on Operating systems principles*, pages 205–220. ACM, October 2007.

[15] Tamer Elsayed, Jimmy Lin, and Douglas W. Oard. Pairwise document similarity in large collections with MapReduce. In *HLT '08: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies*, pages 265–268. ACL, June 2008.

[16] Facebook. Cassandra: A highly scalable, eventually consistent, distributed, structured key-value store, August 2008.

[17] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *SOSP '03: Proceedings of the 19th ACM symposium on Operating systems principles*, pages 29–43. ACM, October 2003.

[18] Google Blog. Sorting 1PB with MapReduce, November 2008.

[19] Joseph M. Hellerstein. Programming a Parallel Future. Technical Report UCB/EECS-2008-144, University of California, Berkeley, November 2008.

[20] Joseph M. Hellerstein, Peter J. Haas, and Helen J Wang. Online aggregation. In *SIGMOD '97: Proceedings of the 23rd ACM SIGMOD international conference on Management of data*, pages 171–182. ACM, May 1997.

[21] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 59–72. ACM, March 2007.

[22] U. Kang, C. Tsourakakis, A.P. Appel, C. Faloutsos, and J. Leskovec. HADI: Fast diameter estimation and mining in massive graphs with Hadoop. Technical Report CMU-ML-08-117, Carnegie Mellon University, Pittsburgh, December 2008.

[23] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. PEGASUS: A Peta-Scale Graph Mining System Implementation and Observations. In *ICDM '09: Proceedings of the 9th IEEE International Conference on Data Mining*, pages 229–238. IEEE Computer Society, December 2009.

[24] H. Karloff, S. Suri, and S. Vassilvitskii. A Model of Computation for MapReduce. In *SODA '10: Symposium on Discrete Algorithms*. ACM, January 2010.

[25] Ralf Lammel. Google's MapReduce programming model – Revisited. *Science of Computer Programming*, 70(1):1–30, January 2008.

[26] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[27] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. PFP: Parallel FP-Growth for Query Recommendation. In *RecSys '08: Proceedings of the 2nd ACM conference on Recommender Systems*, pages 107–114. ACM, October 2008.

[28] LinkedIn. Project Voldemort: A distributed database, June 2009.

[29] Richard M. C. McCreadie, Craig Macdonald, and Iadh Ounis. On single-pass indexing with MapReduce. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 742–743. ACM, July 2009.

[30] Marshall Kirk McKusick and Sean Quinlan. GFS: Evolution on Fast-forward. *Queue*, 7(7):10–20, August 2009.

[31] Michael G. Noll and Christoph Meinel. Building a Scalable Collaborative Web Filter with Free and Open Source Software. In *SITIS '08: Proceedings of the 4th IEEE International Conference on Signal Image Technology and Internet Based Systems*, pages 563–571. IEEE Computer Society, November 2008.

[32] C. Olston, E. Bortnikov, K. Elmeleegy, F. Junqueira, and B. Reed. Interactive Analysis of Web-Scale Data. In *CIDR '09: Proceedings of the 4th Conference on Innovative Data Systems Research*, January 2009.

[33] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig Latin: A not-so-foreign language for data processing. In *SIGMOD '08: Proceedings of the 34th ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, June 2008.

[34] Spiros Papadimitriou and Jimeng Sun. DisCo: Distributed Co-clustering with Map-Reduce: A Case Study towards Petabyte-Scale End-to-End Mining. In *ICDM '08: Proceedings of the 8th IEEE International Conference on Data Mining*, pages 512–521. IEEE Computer Society, December 2008.

[35] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming*, 13(4):277–298, October 2005.

[36] Jennifer Rowley. The wisdom hierarchy: representations of the DIKW hierarchy. *Journal of Information Science*, 33(2):163–180, April 2007.

[37] M.C. Schatz. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25(11):1363, June 2009.

[38] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: A warehousing solution over a map-reduce framework. In *Proceedings of the VLDB Endowment*, volume 2, pages 1626–1629. VLDB Endowment, August 2009.

[39] H. Yang, A. Dasdan, R.L. Hsiao, and D.S. Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *SIGMOD '07: Proceedings of the 33rd ACM SIGMOD international conference on Management of data*, pages 1029–1040. ACM, June 2007.

[40] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P.K. Gunda, and J. Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI '08: Proceedings of the 8th Symposium on Operating System Design and Implementation*, December 2008.