Computing Infrastructure for Big Data Processing

Ling LIU
Distributed Data Intensive Systems Lab,
School of Computer Science, Georgia Institute of Technology, USA
lingliu@cc.gatech.edu

Abstract. With computing systems transforming from single-processor devices to the ubiquitous and networked devices and the datacenter-scale computing in the cloud, the parallelism has become ubiquitous at many levels. At micro level, parallelisms are being explored from the underlying circuits, to pipelining and instruction level parallelism on multi-cores or many cores on a chip as well as in a machine. From macro level, parallelisms are being promoted from multiple machines on a rack, many racks in a data center, to the globally shared infrastructure of the Internet. With the push of big data, we are entering a new era of parallel computing driven by novel and ground breaking research innovation on elastic parallelism and scalability. In this article, we will give an overview of computing infrastructure for big data processing, focusing on architectural, storage and networking challenges of supporting big data analysis. We will briefly discuss emerging computing infrastructure and technologies that are promising for improving data parallelism, task parallelism and encouraging vertical and horizontal computation parallelism.

Keywords. Big data, Cloud computing, Data analytics, Elastic scalability, Heterogeneous computing, GPU, PCM, Big data processing

1 Introduction

Digital data have become a torrent engulfing every area of business, science and engineering disciplines, gushing into every economy, every organization and every user of digital technology. In the age of big data, deriving values and insights from big data using rich analytics becomes an important differentiating capability for competitiveness, success and leadership in every field.

The term "big data" refers to large, diverse, complex, longitudinal, or distributed data sets generated from instruments, sensors, Internet transactions, email, video, click streams, and all other digital sources available today and in the future. Many also use the term “big data” to refer to data that is too large, too dispersed, and too unstructured to be handled using conventional hardware and software facilities [1]. For example, the massive scale of graph data easily overwhelms memory and computation resources on commodity servers. There are two inescapable contests in the big data space to which big data technology has to respond.

The first contest is the *diversity* of data. As we record more data, we end up having different formats of data to manage. About 20% is relational [1], but we also have text, video, image, emails, Twitter feeds, Facebook profiles, social graphs and time series data pumped out from sensors. A popular characterization of big data is by *volume*, *velocity* and *variety* [1]. **Volume** describes the relative size of data to the processing capability. Today a large number may be 10 terabytes. In 12 months 50 terabytes may constitute big data if we follow Moore’s Law. Overcoming the volume issue requires both technologies that store vast amounts of data in a scalable fashion and technologies that use distributed approaches to querying and deriving actionable information and insights from the big data. **Velocity** describes the frequency at which data is generated, captured and shared. The velocity of large data streams from a vast range of devices and click streams not

Received Month dd, yyyy; accepted Month dd, yyyy

E-mail: lingliu@cc.gatech.edu

only create requirements for greater real-time use cases, but also power the ability to parse text, detect sentiment, and identify new patterns. Real-time analytics require fast matching and immediate feedback loops based on alignment with geo location data, social media, user history and current sentiment. **Variety** refers to the proliferation of data types from social, machine to machine, and mobile sources in addition to traditional transactional data. Data no longer fits into neat, easy to consume structures. The addition of unstructured data such as speech, text, and language increasingly complicate the ability to categorize data. Such diversity of data not only requires elastic storage, transport, access and processing methods but also calls for new approaches to deriving deeper insights and values from big data.

The second contest is the *richness of analytics*. As more digital data and information technology penetrate into science and engineering fields, we are confronted with richer analytics to perform. In addition to SQL queries, we also have time series analysis, statistical analysis, geo-spatial analysis, graph analysis, sentiment analysis, entity extraction, and so forth. Not surprisingly, the platforms and tools for big data analytics today are too few to make rich analytics scale to big data and yet too complex to make them plug and play.

We envision that the rigorous demands that big data places on networks, storage and servers make it more attractive to outsource big data analytics (the hassle and expense) to the cloud. Furthermore, the explosive interest in big data will help the domain scientists and every enterprise and organization to realize that they can analyze all their data in the same way as large companies like Walmart, Amazon do from business optimization to real time analysis of domain specific data as well as customer data without huge up-front ownership cost for supporting big clusters, big administration, big programs, and so forth. We argue that developing the systematic approaches to mining big data for intelligence and optimization are the key to push the big data ecosystem to develop new systems and software that let domain scientists, business owners, organizations of all sizes plow through massive datasets faster and more efficiently.

In this article, we will give an overview of computing infrastructure for big data processing, focusing on *architectural, storage and networking challenges of supporting big data analysis*. Instead of casing through known or standard cloud computing infrastructure, delivery models for big data storage and processing, we accentuate on rethinking the computing stack for big data processing. We will emphasize on emerging computing infrastructure and technologies that are promising for improving data parallelism, task parallelism and encouraging vertical and horizontal computation parallelism. By **data parallelism**, we refer to the simultaneous execution of the same function across the element of a dataset. By **task parallelism**, we refer to simultaneous execution of many different functions across the same or different datasets. By **vertical and horizontal computation parallelism** we refer to the architectural innovation, software innovation and algorithm innovation for promoting data parallelism and task parallelism in big data processing.

The rest of the paper is organized as follows: We first give an overview of three promising architectural technologies, emphasizing on their sweet spots for big data processing by exploiting parallelisms. Then we discuss key issues in dealing with big data that exceeds the capacity of existing hardware or the processing capability of existing data storage and computation software by reviewing a selection of alternative data partitioning models in terms of their effectiveness for big data processing.

2 CPUs for Big Data Processing

A GPU is a highly parallel computing device designed for the task of graphics rendering [2]. Despite its graphics root, GPU (Graphics Processing Unit) has evolved to become a more general processor with hundreds of cores and more powerful than CPU for the execution of data-parallel, arithmetic (versus memory) intensive applications in which the same operations are carried out on many elements of data in parallel. The reason for the large performance gap between many-core GPUs and general purpose multi-core CPUs lies in the fundamental architectural design of the two processors. A CPU is designed to optimize for sequential code performance. It uses sophisticated control logic to allow instructions from a single thread of execution to execute in parallel or even out of sequential order while maintaining the appearance of sequential execution. It provides large cache memories to reduce the instruction and data access latencies of applications. In contrast, the GPU is designed to optimize for the execution of massive number of threads. The hardware takes advantage of a large number of execution threads to find work to do when some of them are waiting for long-latency memory accesses, thus minimizing the control logic required for each execution thread. Small cache memories are provided to help control the bandwidth requirements of these applications so that multiple threads that access the same memory data do not need to all go to DRAM. As a result, much more chip area is dedicated to the floating-point calculations.

GPUs are especially well-suited to address problems that can be expressed as data-parallel computations with a high ratio of arithmetic operations to memory operations. Many applications that process large data sets can use a data-parallel programming model to speed up simulations. For example, image and media processing applications such as video encoding and decoding, image scaling, and pattern recognition very naturally map image blocks and pixels to parallel processing threads, Example applications include video processing, machine learning, 3D medical imaging, computational finance, computational biology.

GPU is designed as a numeric computing engine and it will not perform well on some tasks. Therefore, most applications will use both CPUs and GPUs, executing the sequential parts on the CPU and numeric intensive parts on the GPUs. A known challenge to joint CPU-GPU execution of an application is the significant overhead of memory transfers between the host CPU and the GPU. In general, for smaller sized datasets, the overhead of time spent in sending data to the GPU and bringing it back neutralizes any performance benefit obtained by computing on the GPU. Figure 1 shows CPU and GPU interconnection. Also GPUs offer best performance gains when all the computing resources, processing cores and memory, are maximally utilized. Thus, it is best to analyze data sizes to determine which jobs to offload to the GPU. Figure 2 shows a performance comparison of block nested loop (BNL) and GPU-based Nested Loop (GNL) for skyline query computation using two types of datasets: anti-correlated and independent datasets with varying dimensionality (*d*) and size of datasets (*N*).

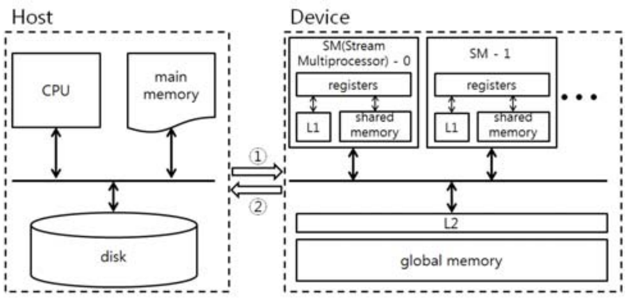
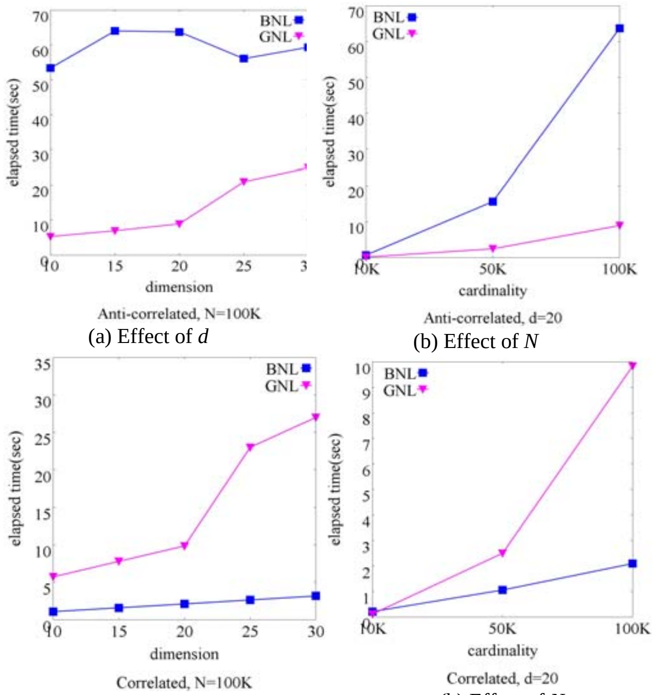


Fig. 1 GPU and CPU interconnetion



(a) Effect of *d* (b) Effect of *N*

(a) Effect of *d* (b) Effect of *N*

Fig. 2. Elapsed time for anti-correlated and correlated datasets

By design, GNL aims to drastically reduce the cost of skyline dominance tests by parallelizing the CPU-based dominance test itself using GPUs. GNL can determine dominance relationship by performing multiple dominance tests in parallel. It is noteworthy that GNL is orthogonal to existing pruning efforts, and can be used by all existing algorithms [17] after their own pruning steps to speed up the performance of their dominance tests.

Recent research on tightly integrating a CPU and a GPU on a single chip [4] shows that it can significantly reduce power and performance overhead introduced by the communication of CPUs and GPUs over long electrical connections, without sacrificing process power [5].

3 Heterogeneous Many Core Computing

Many-core computing devices have large numbers of processors (cores) on a single chip. Such configurations are attractive because they can achieve a greater performance (calculations per second) for a given amount of electrical power than their single-core equivalents. With dual-core and quad-core CPUs now commonplace, processor development is heading forward along the route of hundreds to thousands of cores. In addition, multi-core chips mixed with simultaneous multithreading, memory-on-chip, and special-purpose "heterogeneous" cores promise further performance and efficiency gains, especially in processing multimedia, recognition and networking applications.

A many-core system on a die can be an asymmetric system with a few large cores to deliver higher single-thread performance, and a large number of small cores. A heterogeneous system consists of general purpose cores (GP) and special purpose cores (SP), each core having local cache memory, and all cores connected together with an on-die interconnection network. Example of special purpose cores are those for hardware acceleration, e.g. graphics engines. Although a many-core system will deliver higher compute throughput than a multi-core system for the same die size and in the same power envelope, it may be difficult to harvest the performance of thousand cores, according to Amdahl’s Law [6,7], which states that the parallel speedup is limited by the serial code in a program. If the serial percentage in a program is large, then parallel speedup saturates with small number of cores. Thus, only data parallelism of a single application is insufficient to harvest the performance of a many-core system [7]. We need to exploit and utilize task level and application level parallelism to maximize many core systems for big data processing.

Although the many-core architecture with hundreds to thousands of small cores can deliver unprecedented compute performance in an affordable power envelope, fine grain system power management, an optimized on-die-network, and efficient memory technology are vital. Furthermore, how to make storage and system software as well as application developments to fully benefit from the potential of hundreds to thousands of cores poses interesting research challenges for big data processing and big data technology innovation.

4 Persistent Memory for Big data storage

Persistent memory, also called Phase change memory (PCM), is an emerging non-volatile memory technology pioneered by Intel, Numonyx, Samsung, IBM and others, as a low cost, more reliable, faster and better alternative to flash memory [8,9].

In contrast to DRAM, non-volatile memory can retain information without power and has fast access times, providing both huge power savings and the potential for much faster data transfer. PCM combines the best attributes of NOR, NAND and RAM within a single chip, including bit-alterable, non-volatile, fast read speed, fast write/erase speed and good scalability [8]:

- **Bit alterable:** unlike flash memory, stored information can be switched from one to zero or zero to one without a separate erase step.
- **Non-volatile:** like NOR flash and NAND flash, PCM does not require a constant power supply to retain information.
- **Read performance**: PCM features fast random access times, Similar to RAM and NOR flash memory, which enables the execution of code directly from the memory, without an intermediate copy to RAM. The read latency of PCM is comparable to single bit per cell NOR flash, and the read bandwidth can match DRAM.
- **Write/erase performance**: PCM achieves write throughput speeds faster than NAND and with lower latency. This will deliver significant write performance improvement over NOR and NAND flash, When combined with bit-alterable (no separate erase step).

Scalability: PCM is immune to the charge storage scaling issue, because it does not store charge (electrons). As the memory cell shrinks on flash, the number of electrons stored on the floating gate shrinks. In contrast, both NOR and NAND rely on floating gate memory structures, which are difficult to shrink.

Up to now, all extant architectures assume that directly CPU-accessible memory is volatile. From the history of computing, a persistent storage system, including the file system and virtual memory system, has assumed to base on the characteristics of moving-head disks, devices with a very high access latency organized into fixed-sized blocks of thousands of bytes. These characteristics run deep at every level of the software stack. The PCM technologies are fundamentally different both in their low access time and their fine-grained (byte-level) access. With persistent memory, architectures and software design and optimizations will need to respond accordingly because widespread use of non-volatile memory will make the distinction between main memory and bulk memory disappearing. This may require redesign of the operating system and file system to take advantage of non-volatile memory technologies.

Although for now, the base memory technology is still the biggest wild card for most of the big data processing technologies, it is important to start rethinking of big data computing in the context of emerging architecture and storage technologies.

5 Exploiting Parallelism and Scalability

Designing a scalable system for analyzing, processing and mining huge real world datasets has become one of the challenging problems facing both systems researchers and data management researchers. For example, many big graphs such as Web graphs, social network graphs, protein interaction graphs are particularly challenging to handle, because they cannot be readily decomposed into small parts that could be processed in parallel [13,14].

One of the technologies that made big data analytics popular and accessible to enterprises of all sizes is MapReduce [22] (and its open-source Hadoop [9] implementation). The MapReduce programming model has become popular because a programmer can harness the processing power of a cluster of commodity hardware for very large parallel tasks in a simple way. The programmer only needs to write the logic of a Map function and Reduce function. The MapReduce framework divides the program execution across a cluster of computer nodes into a Map and Reduce stage, separated by the transfer of data between machines in the cluster, called the Shuffle stage. In the Map stage, each Mapper machine in the cluster executes a Map function on a distinct region of the input data. The Map execution produces the Map output of key-value records locally at each machine. The records for any given key, which are spread out on many machines, are aggregated at each Reducer for the Reduce stage by remotely reading from the Mappers. An important challenge for scaling MapReduce enabled data analytics in the cloud is to manage multiple virtual MapReduce clusters, corresponding to a diverse set of analytic jobs, executing concurrently on a shared physical infrastructure. Each MapReduce job generates different loads on the shared physical infrastructure: (a) computation load: number and size of each VM (CPU, memory), (b) storage load: amount of input, output and intermediate data, and (c) network load: traffic generated during the map, shuffle and reduce phases. The network load is of special concern with MapReduce as large amounts of traffic can be generated in the shuffle phase when the output of map tasks is transferred to reduce tasks. As each reduce task needs to read the output of *all* map tasks [22], a sudden explosion of network traffic can significantly deteriorate cloud performance. For example, one way to enhance the performance of data analytic jobs on MapReduce cloud is to devise smart partitioning and placement optimization that can minimize or reduce inter-VM (virtual machines) network traffic for MapReduce workloads. We argue that by improving data locality for both Map and Reduce phases of an analytic job, one can effectively reduce the network distance between storage and compute nodes for both map and reduce processing. For instance, the VMs executing the reduce tasks should be close to the map-task VMs which generate the intermediate data used as reduce input. This improved data locality can reduce both job execution times and cumulative network traffic of virtual MapReduce clusters, because network transfer time is often the bottleneck of total execution time for an analytic job. While map locality is well understood and implemented in MapReduce systems, reduce locality has surprisingly received little attention in spite of its significant potential impact on performance. Figure 3 shows our preliminary study [19] on the impact of improved reduce locality for a Sort workload. It shows the Hadoop task execution timelines for a 10 GB dataset in a 2-rack 20-node physical cluster1, where 20 Hadoop VMs were placed without and with reduce locality. We see reduce locality resulted in a significantly shorter shuffle phase, leading a reduction of total job runtime by 4x.

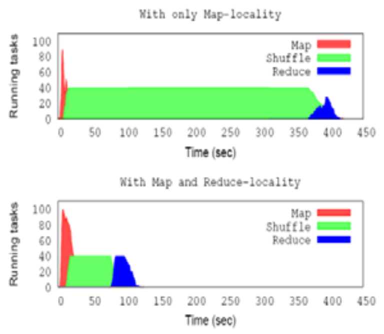


Fig.3 Impact of Reduce locality

(Timeline plotted by Hadoop's job_history_summary**)**

Although MapReduce is a good fit for a wide array of large scale computing problems [12,13], there are a fair amount of datasets and the corresponding computing problems and analytic algorithms, which cannot be readily decomposed into small parts that could be processed in parallel efficiently. For example, many have argued that datasets with high degree of correlation such as graphs, often lack of data-parallelism, and thus render MapReduce inefficient for computing on such graphs and can lead to suboptimal performance and usability issues [13,14].

We argue that unlike traditional MapReduce, where data is placed independently of their inherent correlation and independent of how the data will be processed by the analytic jobs, to enable elastic scalability of data analytic jobs over complex datasets with high degree of correlations, such as graph datasets, the correlation semantics should be taken into accounted when exploiting data parallelism by data partitioning and partition placement.

To better illustrate our design ideas and partitioning algorithms, we will use real-world graphs as examples to introduce our data partitioning and computation partitioning techniques. We model a direct (sparse) graph $G=(V,E)$. We associate a value with each vertex $v \in V$, and each edge $e = (u,v) \in E$. We refer to u as the *source* of edge e and v as the *destination* of edge e. Similarly, we refer to e as the outgoing edge or out-edge of vertex u and the incoming edge or in-edge of vertex v. We assume a computer with limited memory (DRAM) capacity such that the graph structure, edge values and vertex values do not fit into memory. In the context of big data, it implies that the amount of memory is only a small fraction of the memory required for storing the complete graph. We also assume that there is enough memory to contain the edges and their associated values of any single vertex in the graph. We first examine hash-based graph partitioning methods and analyze their pros and cons in terms of computation efficiency.

To better understand how to design the most effective hash partitioning algorithm, we first analyze the following three basic hash partitioning methods:

(i) Forward hash partition: We partition a graph into N partitions, each consists of a set of vertices and the outgoing edges of those vertices.

(ii) Reverse hash partition: We partition a graph into N partitions, each consists of a set of vertices and the incoming edges of those vertices.

(iii) Bi-direction hash partition: We partition a graph into N partitions, each consists of a set of vertices and all edges of those vertices (bi-direction, both in-edges and out-edges). Sometimes we refer to bi-direction hash partition simply as hash partition when no confusion occurs.

In all these hash partitioning methods, edges are sorted by their source vertex [14] and assignment of a vertex to a partition depends solely on the vertex ID. The default partitioning function is $hash(ID) mod\ N$. The distribution of partitions to worker machines can be done by the master node where a partition to machine index is maintained. Let d be the number of worker machines. Typically, $d << N$. The clean separation of partitions from worker machines, it enables the system to improve both load balance and fault tolerance: Thus, when one worker machine is overloaded or fails to respond, we can easily reassign some partitions or the unfinished partition to another worker machine to achieve better load balance and higher fault tolerance. Pregel [13] has demonstrated that by forward hash partitioning of big graphs, it can scale the computation of PageRank, Shortest Paths, Bipartitie Matching and a Semi-Clustering algorithm using a clusters of machines. GraphChi [14] has shown that the reverse hash partitioning method is more I/O efficient for certain types of graph computations as in-edges are ordered by source, the search of out-edges of a given vertex can be done in a fixed number (d-1) of inter-machine communications. In comparison, vertex hash partition, also referred to as bi-direction hash partitioning in this article, incurs more memory consumption compared to forward and reverse hash partitioning method due to the amount of edges included in each partition. At the same time, bi-direction hash partitioning offers better computational efficiency for certain graph computations, such as computing vertex value based on the counts or weights of both its in-edges and its out-edges – finding the list of two-hop friends in a social network.

Interestingly, for certain types of graph queries, all the above baseline hash partitioning methods are no longer effective due to the amount of inter-partition communications incurred for computing the results of queries. Fig.1 shows an example RDF graph extracted from the LUBM benchmark [15]. Each edge represents a subject-predicate-object triple with source vertex as subject and destination vertex as object and edge represents the predicate linking subject and object.

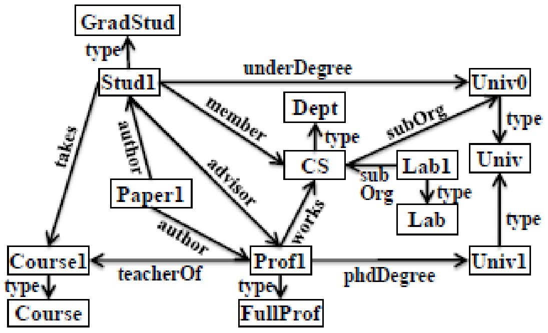


Figure 1 An Example RDF Graph

Fig.2 shows three example RDF queries expressed in SPARQL [16], a SQL-like standard query language for RDF, recommended by W3C.

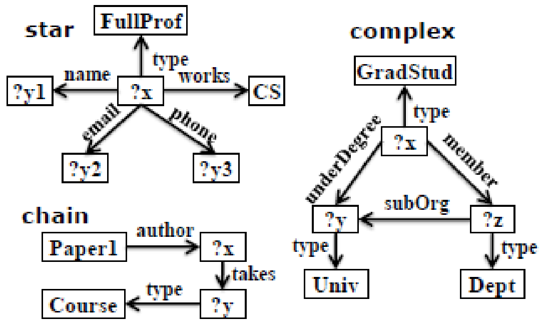


Figure 2 SPARQL Query Graphs

Most SPARQL queries consist of triple patterns, which are similar to RDF triples except that in each triple pattern, the subject, predicate and object may be a variable. A triple pattern is said to match a subgraph of the RDF data when the terms in the subgraph may be substituted for the variables. Processing a SPARQL query *Q* involves graph pattern matching and the result of *Q* is a set of sub-graphs of the big RDF graph, which match the triples patterns of *Q*. SPARQL queries can be categorized into three types in terms of their join characteristics: star, chain and complex queries.

Star queries are the most common, have only one join variable, which is the subject of all the triple patterns involved, and consist of subject-subject joins (e.g., the upper-left example in Fig.2).

Chain queries consist of subject-object joins (i.e., the subject of a triple pattern is joined to the object of another triple pattern) and their triple patterns are connected one by one like a chain (e.g., the bottom-left example in Fig.2).

Complex queries refer to the remaining queries that are combination of star and chain queries, which include combinations of star and chain queries (e.g., the right query graph in Fig.2).

Clearly, star queries can be processed efficiently in parallel if we partition the RDF graph using the forward hash partition and bi-direction hash partition methods because no inter-partition communication cost will be incurred. However, all three basic hash partitioning techniques are inefficient for processing non-star queries. This is because inter-partition processing includes the transfer of intermediate results across multiple servers, the communication cost can be very high when the size of intermediate results is large. Also, because we use Hadoop MapReduce to join the intermediate results and it has to pay the initialization overhead for each Hadoop job, the query processing may be unnecessarily slow even though the size of intermediate results is small. In our experiments, Hadoop MapReduce requires about 10 seconds to initialize one Hadoop job.

There are other challenges in the context of exploiting data parallelism through data partitioning. For example, we need to ensure load balance in terms of partition to machine assignment. The more semantics are incorporated into the data partitioning algorithms, the harder to devise a good load balanced scheme for assigning partition to machine. To make the matter worse, the assignment of partitions to machines should also minimize the communication cost across different machines as network I/O is known to be expensive in cloud based data centers.

Big data analytics hold the promise of revealing insights hidden previously by big data that is too costly to process. Hadoop MapReduce is not a panacea or a magic bullet to solve or address all big data processing problems. We need a transformative and forward looking approach to developing alternative and complimentary big data computing and programming models that can achieve and maximize data parallelism, task parallelism and vertical and horizontal computation parallelism for all types of datasets. For example, in addition to extend Hadoop MapReduce, we should also devote research and engineering efforts to new programming models that are alternative and complimentary to Hadoop MapReduce programming model for processing and analyzing big datasets with varying degree of correlations, complexity in querying, manipulation and analysis, and for providing real-time analytics in a plug and play delivery model. In addition, we need new programming models that can generalize many domain specific analytic algorithms and problem-specific learning kernels into programmable programming models with easy to use interfaces for domain scientists and business owners and enterprises of all sizes. Ultimately, delivering big data analytics as a service (DAaaS) will transform the service computing infrastructure today into a more user-friendly and more value-drive big data computing eco-system of the future.

6 Conclusion

We are entering a big data technology era with exciting opportunities and research challenges: computer chips are evolving from multi-core to many cores (thousands of cores), memory technology is evolving from gigabytes (GB) to terabytes (TB), with the promise of persistent memory, storage technology is evolving from terabytes to petabytes, and server technology is evolving from cluster computing to rack based computing and networking technology is evolving to co-existence of TCP-IP with many other alternative channels of inter-device communication channels.

In this article, we have given a brief overview of the emerging computing infrastructure for big data processing. We focus on architectural, storage and networking challenges for supporting big data analysis, especially those that are improving data parallelism, task parallelism vertical and horizontal computation parallelism. We encourage the data and systems research community to rethinking of how to support big data processing by leveraging all these emerging architectural, storage and networking advances with innovations, transformative scientific methodologies, and efficient engineering solutions.

In the past 30 years, data was primarily used to record and report business and scientific events, and in the next 30 years data will be used also to derive new insights, to influence business and scientific events, and to speed up and advance scientific discovery. We conjecture that deriving values and insights from big data using rich analytics will be an important differentiating capability for competitiveness, success and leadership in every field.

Acknowledgements The author acknowledges discussions and collaborations with many of her current collaborators in big data research and cloud computing research from DiSL, Georgia Institute of Technology, and other institutions in US and Asia. This work is partially support from grants by NSF CISE SaTC program, NetSE program, an IBM faculty award, and Intel Science and Technology Center on Cloud Computing.

References

1. Mckinsey Big data: The next frontier for innovation, competition, and productivity, 2011. http://www.mckinsey.com/insights/mgi/research/technology_and_innovation/big_data_the_next_frontier_for_innovation.
2. Graphics Processing Unit (GPU). Wikipedia 2012. http://en.wikipedia.org/wiki/Graphics_processing_unit
3. Chris Gregg and Kim Hazelwood. “Where is the Data? Why You Cannot Debate CPU vs. GPU Performance Without the Answer”, ISPASS 2011. http://www.cs.virginia.edu/kim/docs/ispass11.pdf
4. Nam Sung Kim, Stark C. Draper, Shi-Ting Zhou, Sumeet Katariya, Hamid Reza Ghasemi, Taejoon Park: Analyzing the Impact of Joint Optimization of Cell Size, Redundancy, and ECC on Low-Voltage SRAM Array Total Area. IEEE Trans. VLSI Syst.

20(12): 2333-2337 (2012)

5. Syed Zohaib Gilani, Nam Sung Kim, Michael J. Schulte: Power-efficient computing for compute-intensive GPGPU applications. PACT 2012: 445-446
6. Tim Mattson. “The Future of Many Core Computing:A tale of two processors”, Intel Labs, 2010.
7. Shekhar Borkar. “Thousand Core Chips – A Technology Perspective”. ACM DAC 2007.
8. Wikipedia. “Phase-change memory (PCM)”, 2012.
9. CRA. 21st Century Computer Architecture. http://cra.org/ccc/docs/init/21stcenturyarchitecturewhitepaper.pdf.
10. NRC. “The Future of Computing Performance: Game Over or Next Level?”, http://www.nap.edu/catalog.php?record_id=12980.
11. NSF. “*Advanced Computing Infrastructure: Vision and Strategic Plan”, 2012. http://*www.nsf.gov/pubs/2012/nsf12051/nsf12051.pdf.
12. Yahoo WebScope. Yahoo! Altavista web page hyperlink connectivity graph, circa 2002, 2012. http://webgraph.sandbox.yahoo.com.
13. Grzegorz Malewicz, et.al. “Pregel: A System for Large Scale Graph Processing”. SIGMOD 2010.
14. Aapo Kyrola et.al. “GraphChi: Large Scale graph Computation on Just a PC”. OSDI 2012.
15. Y. Guo, Z. Pan, and J. Heflin, “LUBM: A benchmark for OWL knowledge base systems,” Web Semant., vol. 3, no. 2-3, pp. 158–182, Oct. 2005.
16. “SPARQL Query Language for RDF,” http://www.w3.org/TR/rdf-sparqlquery/.
17. Wonik Choi, Ling Liu and Boseon Yu . “Multi-Criteria Decision Making with Skyline Computation”, In Proceedings of the 13th IEEE International Conference on Information Reuse and Integration, Aug 8-10, 2012.
18. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. pages 137–150, December 2004.
19. Balaji Palanisamy, Aameek Singh, Ling Liu, Bhushan Jain. ``Purlieus: Locality-aware Resource Allocation for MapReduce in a Cloud", ACM/IEEE International Conference on SuperComputing (SC2011), Seattle WA, Nov. 12-18, 2011.

Ling Liu is a full professor in the School of CS at Georgia Institute of Technology. She directs the research programs in Distributed Data Intensive Systems Lab (DiSL) in the areas of cloud computing, Big data and big data analytics, distributed computing. She has published over 300 International journal and conference papers, and supervised more than 20 Ph.D. dissertations. She is a recipient of 2012 IEEE Computer Society Technical Achievement Award and a co-editor-in-chief of the 5 volumes of Encyclopedia of Database Systems (Springer 2010). She is the Editor in Chief of IEEE Transactions on Service Computing, on the editorial board of over a dozen international journals, and has served as general chair and PC chairs of many international conferences. Her current research is supported by the U.S. National Science Foundation (NSF) and industrial companies such as IBM and Intel.