VOL. 14, NO. 1, JANUARY 2019 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

SCHEDULING ALGORITHM FOR CPU-GPU BASED HETEROGENEOUS CLUSTERED ENVIRONMENT USING MAP-REDUCE DATA PROCESSING

Suman Goyat and A. K. Sahoo
Department of Computer Science and Engineering, Sharda University Greater Noida, Uttar Pradesh, India
E-Mail: goyat1606026@gmail.com

ABSTRACT

MapReduce is a popular large-scale data-parallel processing model for analysing and processing large massive data sets. Its success has stimulated several studies of implementing MapReduce on Graphic Processing Unit (GPU). Hadoop's has motivated research interest and has led to different modifications as well as extensions to framework. The Graphics Processing Units (GPU) are widely used in the High-Performance Computing world to enhance job throughput, as its architecture is quite data-parallel friendly. The problem is to find Software and/or hardware solutions that allow firms to discover, evaluate, optimize, and deploy predictive models by analysing big data sources to improve business performance or mitigate risk in major data processing tools. MapReduce has also been widely adopted to solve Big Data problems and in this scenario combination of CPU and GPU will provide huge advantage over the scheme where only CPU is utilised. By default, Hadoop supports some fairly simple scheduling policies e.g. FIFO, fair scheduling, or capacity scheduling. In this paper we will study about MapReduce implementation of big data analysis using heterogeneous CPU-GPU scheduling to improve the performance of the system.

Keywords: MapReduce, data governance, data storage, graphical processing unit, analytics, online processing, privacy and security issues.

1. INTRODUCTION

The recent trends in computer architecture, specifically, the emergence of multi-core CPUs and many-core GPUs, and their integration within a single machine, have given rise to a new class of heterogeneous architectures. Such heterogeneity can be seen within popular desktops, as well as in each node of several of the world's fastest supercomputers. As an example of the former, AMD's recently released Fusion Accelerated Processing Unit integrates CPU and accelerator on a single die and is targeting the desktop and notebook market. As examples of the latter, in the top 500 list three of the top five fastest supercomputers in the world are CPU/GPU clusters. Apart from being used in high end clusters, heterogeneous architectures have been recently introduced in cloud environments Large-scale data processing is tougher than ever before since the size of the data is increasing too fast for hardware resources to keep up. Nowadays, Graphics Processing Units (GPU) is widely used in the High-Performance Computing world to enhance job throughput, as its architecture is quite data-parallel friendly. MapReduce has also been widely adopted to solve Big Data problems [6]. It was originally proposed by Google to pursue a simple and flexible parallel programming paradigm. With MapReduce, users need only write Map and Reduce functions to solve problems in parallel. The underlying programming details, such as how to handle communication among data nodes, are transparent to users. Data affinity across the network and fault tolerance among multiple nodes can be achieved automatically. Heterogeneous systems have received much attention from the parallel software community, though mostly from the programmability and application portability view-point. NVIDIA's CUDA [1], an extension to C, has been popular for programming general purpose computations, although there is a growing interest in OpenCL [2], because of its portability. As GPU technology advances, more powerful GPUs such as NVIDA Kepler have been developed. To leverage the most advanced features of Kepler such as Asynchronous Dual-channel Data Transfer and Hyper Q, we introduce a pipelined workflow to make our system more powerful and efficient. In order to handle Big Data sets, increasing only the computability is not enough. To push our system even further, we use the idea of dynamic schedulingand enhance its ability to process huge amount of data as well as efficiently transfer data between disks and processing units. This paper proposes a scheduling algorithm using Multi-GPU MapReduce implementation for Big-Data Processing.

2. PROBLEM DEFINITION AND APPROACH

Despite much interest in heterogeneous systems, key scheduling challenges associated with them have not received much attention. Particularly, with highly shared clusters (such as those at supercomputer centres) and cloud resources having heterogeneous CPU-GPU nodes, new application scheduling problems are arising. In such shared clusters or cloud environments, high utilization of resources and overall system throughput are important considerations, as opposed to the need for scaling a single application. In this paper, we consider a cluster where each node has a multi-core CPU and a GPU. Our goal is to accelerate a set of applications using the aggregate set of resources in the cluster. We focus on two distinct scheduling problems. Single Node Job Scheduling: In the first problem, we assume that an application can either execute on a single GPU or a multi-core CPU, on any one

node in the cluster. This is consistent with some of the most recent trends in application development for multi-core and many-core architectures. For instance, OpenCL [2] is becoming a popular programming model for heterogeneous architectures and is device agnostic. Thus, a kernel can be written once and compiled for many devices to produce different binaries. Furthermore, recent research has proposed compilation techniques to dynamically transform CPU into GPU code and vice-versa. For example, MCUDA [10] performs basic transformations of CUDA code into C, Ocelot [4] and PGI CUDA-x86 [11] transform CUDA into optimized x86 compatible code, SWAN [12] performs a CUDA-to-OpenCL transformation, whereas OpenMPC [13] allows OpenMP-to-CUDA translation. In comparison, however, the available support for developing applications for clusters of GPUs is much limited, and the existing GPU benchmarks like Rodinia [14] and Parboil [15] comprise only single-GPU applications.

This paper gives the following contributions:

- Multiple GPUs are utilized to accelerate MapReduce operations.
- Pipelined workflow maximizes the usage of the GPU by overlapping communication and computation. Since we are hiding the traffic behind computation, PMGMR minimizes the overall communication time, thereby increasing the throughput.
- A job scheduler is employed to manage memory and data-flow. Input size is no longer bounded by GPU memory and even CPU memory.
- For Fermi architecture, a GPU operation scheduler is implemented to stabilize performance when GPU operations are issued from multiple CPU threads.
- A configuration optimizer is adopted, which will collect the running result to dynamically adjust the environment settings and ensure a balanced load. As the computation continues, the result will move closer to the ideal curve.

The remainder of this paper is organized as follows: Section 2 introduces necessary GPU architecture and MapReduce framework background. We will discuss our experimental results in Section 3 by comparing our Pipelined system with other similar systems. In Section 5, some related MapReduce implementations are briefly discussed. Finally, the conclusion and future work are given was developed in CUDA targeting Nvidia Fermi and Kepler architectures.

3. GPU BACKGROUND

GPU hardware consists of several streaming multiprocessors (SM), each of which contains multiple computing cores, registers, and a small (e.g. 64KB) but fast on-chip memory called shared memory, accessible to the cores of the SM. The cores of an SM execute in lock-step with each core executing the same instruction at the same time; i.e., in SIMD fashion. The GPU is connected to an off-chip DRAM memory called global memory, with up to a few gigabytes of space to all cores of all SMs. Accesses to global memory is an order of magnitude slower than accesses to registers and shared memory. We refer to this global memory as GPU memory in this paper to differentiate it from CPU main memory. The GPU is connected to the CPU through a bidirectional link, such as PCIe. The GPU will have one or more DMA engines capable of transferring data over this link between CPU and GPU memory. Practically we can see the structure of both GPU and CPU in the terms of the number of cores as in the following figure:

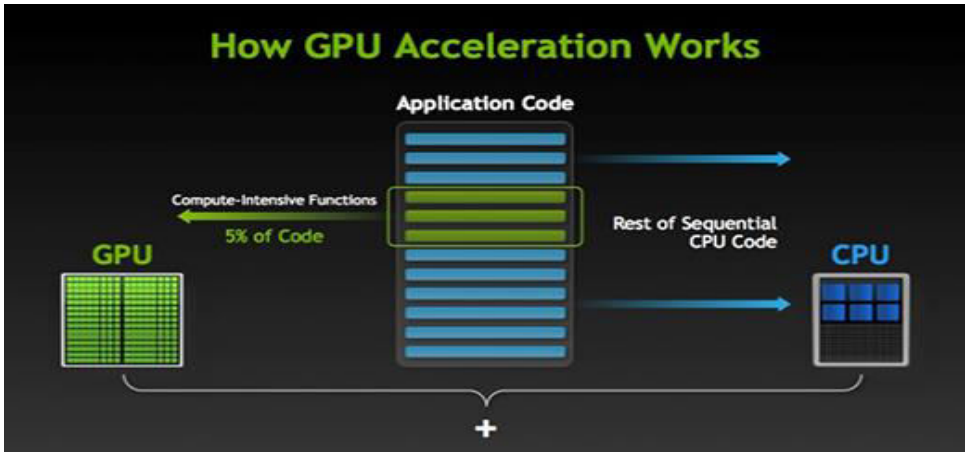


Figure-1. GPU acceleration.

GPUs have thousands of cores to process parallel workloads efficiently and CPU have multiple cores may be 8 or 16 as per the configuration of PC.

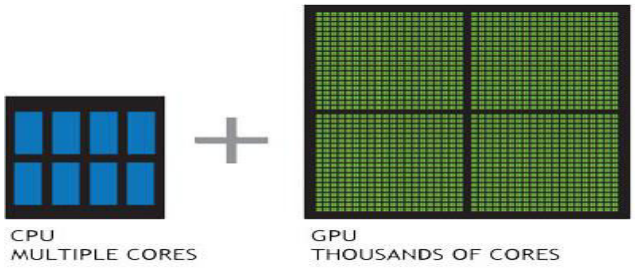


Figure-2. Number of cores in CPU and GPU.

Parallelism appears to be a sustainable way of increasing performance, and there are many applications that display embarrassingly parallel workloads that are perfectly suited for GPUs. However, increased parallelism will only increase the performance of parallel code sections, meaning that the serial part of the code soon becomes the bottleneck. This is often referred to as Amdahl's law [17]. From early academic proof-of-concept papers around the year 2000, the use of GPUs has now become popular in countless industrial applications.

However, the DMA engine can only access CPU memory that has been pinned so that it will not be paged out by the OS. The DMA engines also provide GPU cores direct (but slow) access to CPU memory. The term kernel is used to denote the function that executes on the GPU by a collection of threads in parallel. The programmer configures the kernel to be executed by a given number of GPU threads. These threads are grouped into thread blocks as configured by the programmer. Each thread block is assigned to a SM by a hardware scheduler. Threads of a thread block are further divided into groups of 32, called warps. The threads in a warp execute in lock-step because the cores share the same instruction scheduler. Thread divergence will occur if, on a conditional branch, threads of the same warp take different paths, which can lead to serious performance degradations. Inter-warp divergence does not negatively impact performance.

4. MAP REDUCE OVERVIEW

Map reduce is a programming framework for processing and analysing large data sets in a highly distributed environment. The map function is responsible for filtering and sorting whereas reduce function is responsible for grouping and aggregation operations. A map task can run on any compute node in cluster. Multiple map tasks can run in parallel across the cluster. The output of all the maps will be partitioned and each map will be partitioned and then each partition will be sorted. There will be only one partition for each reduce task. There can be multiple reduce tasks running in parallel on the cluster.

The reduce function will aggregate the information received from map functions. Mapreduce framework supports scalability to a greater extent. Also, Map Reduce achieves a high level of parallel operations and distributed execution over a large number of nodes. In the map reduce model, task is distributed into a multiple job which are assigned to nodes in the network. Reliability is achieved by reassigning any failed node's job to another node. The widely used open source MapReduce implementation is Hadoop. Hadoop uses MapReduce on top of the Hadoop Distributed File System (HDFS) which consists of a Job Tracker. The Job Tracker pushes the work out to an available node in the cluster, striving to keep the work as close to the data as possible. By default, Hadoop supports some fairly simple scheduling policies e.g. FIFO, fair scheduling, or capacity scheduling.

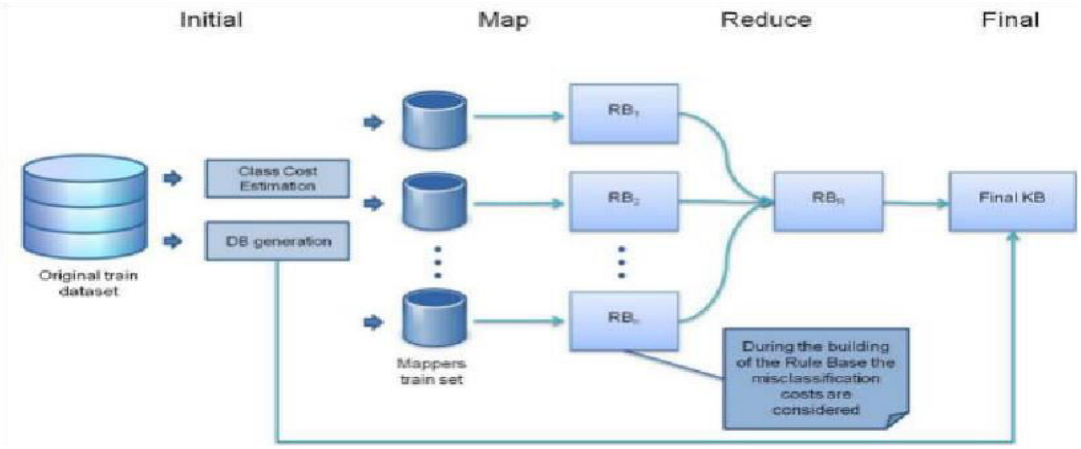


Figure-3. Map reduce flow.

5. SYSTEM MODEL AND CHALLENGES

In this section, we describe our system model, i.e., the nature of the environment and applications we consider. Then, we make several observations about possible approaches and challenges.

A. System model

The CPU and the GPU are connected through a PCIExpress interconnect. Each node communicates with the other nodes in the cluster using an interconnection network. Any given application can either use all cores on the multicore CPU or can primarily use the GPU. It should be noted that GPU intensive applications still require one

CPU thread. Typically, such thread is used to issue GPU calls and does not consume many computing cycles. Therefore, we can assume that all cores in the CPU can be effectively used by other applications scheduled on the same node to use the CPU. In practice, it is possible that some applications can be executed only on the multi-core CPU or primarily on the GPU, but for simplicity, we assume that each application can be executed on either platform. As mentioned in the Introduction, recent research has focused on compilation techniques to dynamically transform CPU into GPU code, and vice-versa. In our scheduling problem, we further assume that no job preemption is performed, and the scheduler operates on batches of jobs. However, batches of jobs enter the system stochastically and are not known a priori. Existing techniques for scheduling across multiple sites [19], [18] and widely used resource managers like SLURM [14] make scheduling decisions based on user inputs regarding execution times and speedups. Similarly, to these approaches, we assume that information about relative speedups/execution times are provided by the user or are available through profiling.

Our implementation supports running the same kernel across both CPUs and GPUs. Kernels can be compiled for available devices prior to or at runtime. Kernels can be run on more than one device in a system. Our scheme allows for implementations that have separate versions of a kernel for each available device and the runtime similarly chooses the correct binary once a device decision has been made. We address occurring problems as difference between homogenous and heterogeneous by using a historical database that contains runtimes for applications on each device and determining a schedule that runs applications on the device in which they will finish first.

Problem can be defined as

- Assigning an application to one of two devices necessitates knowing which device will allow the application to run faster.
- Assign applications to devices to maximize computational throughput while remaining fair to the queue order.
- Running the same kernel across both CPUs and GPUs.
- Allow for implementations that have separate versions of a kernel for each available device.

Steps of the Algorithm In essence, the scheduler we describe implements a greedy algorithm that assigns applications to devices based on a comparison between the predicted times for the application to finish on all available devices. Even if an application would run faster assigned to a particular device, if there are enough applications ahead of it in the queue for that device, it may finish faster assigned to the slower device because that device is free. Our scheduling algorithm is laid out as follows. There are two devices available most applications will run faster when assigned to the GPU. We create a sub-queue for each device, and place applications in those sub-queues from the main queue according to the following rules:

- If a sub queue has applications and the device for that sub-queue becomes free, assign the next application in the sub-queue on that device.
- If one device is busy but the other device is free and the next application in the main queue has not been assigned to that device before, assign it to that device in order to build the database.
- If only one device is free and the next application in the main queue runs slower assigned to that device, estimate how long the other device will be busy using the history database and include other application also scheduled to be assigned to that device in its sub-queue.
- If the next application in the main queue will finish faster by being assigned to the slower device, assign it to that device. If not, put it into the sub-queue for the busy device.
- Continue through the main queue until both devices are busy running applications. As an application finishes, update the historical database with the runtime information, calculating the average runtime and the standard deviation, and repeat the algorithm from the beginning.

The scheduling algorithm described above continues to improve as more data is entered into the historical database, and each application is penalized at most once when it is assigned to a slower device in order to build the database. Because application runtimes are averaged into the previous runtime for a device, outlying points that could be caused by factors unknown to the scheduler (e.g., GPU contention due to video processing associated with the display) are smoothed out over time. Because our scheduler assigns applications to devices that are not necessary optimal for each individual application, we must discuss scheduling fairness. We define a schedule to be fair if each application finishes no later than it would have finished if it were allowed to execute its kernel on its preferred device. In other words, a fair schedule does not penalize an application even if the application is assigned to its non-preferred device. Accordingly, no starvation occurs, and applications are scheduled for a device in queue order. The algorithm presented earlier generates fair schedules except in two cases: if the predicted runtimes are significantly incorrect or when an application is first encountered and is assigned to a slower device.

6. RELATED WORK

In this section, we compare our work against relevant work on parallel job scheduling. Scheduling of parallel jobs in a cluster or a supercomputing center is a very widely studied problem [8]. Other related works that

enable scheduling for parallel jobs in a distributed cluster are [15], and [7]. These efforts propose scheduling strategies that involve both static and dynamic mechanisms. However, the above works focus only on the homogeneous type of processors. Sabin *et al.*[12] and [13] proposed and evaluated job scheduling strategies multiple sites, i.e. separate clusters with different types of processing nodes. While waiting for a resource, it does not consider the possibility of assigning the job to another resource type. Thus, Torque cannot exploit the flexibility that emerging frameworks like OpenCL will provide. Other related efforts [14] do consider the possibility of scheduling a job onto a different resource type, but are limited in the moldability they consider. Parallel job scheduling with moldability has also been studied [10]. Our work is specific to moldability in heterogeneous environments. Ahmad *et al.*[19] [20] developed a scheduling scheme for jobs with fixed priority on heterogeneous computing systems, where they dynamically compute the precedence for jobs on different resources to make scheduling decisions. However, unlike our work, they consider scheduling for dependent set of tasks represented as directed acyclic graphs (DAG). Scheduling with consideration of resource heterogeneity has also been addressed in the context of grid computing [21], [22], [23], [24]. These efforts, however, differ both in targeting much longer running applications, and in the nature of systems they focus on.

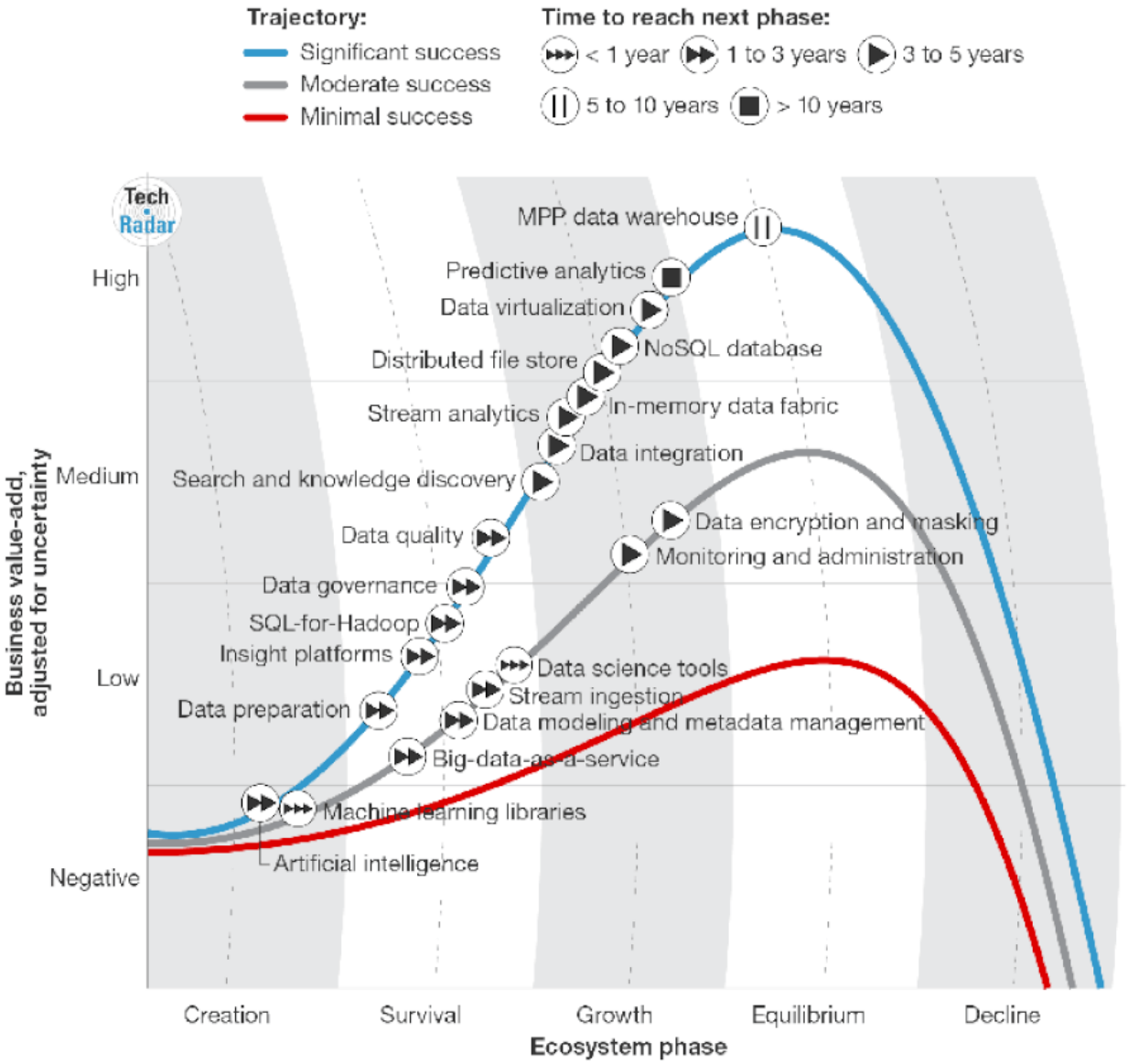
7. CONCLUSIONS

Cluster of heterogeneous nodes comprising multi-core CPUs as well as many-core GPUs. Specifically, we have addressed two scheduling problems: one targeting single-node (first problem) and the other targeting multi-node (second problem) applications.We have shown that our scheduling policies outperform a blind round robin scheduler and approximate the performances of an ideal scheduler that involves an impractical exhaustive exploration of all possible schedules. Relative Speedup based policy with Conservative assignment (RSC), and Adaptive Shortest Job First policy (ASJF) can be proposed. Our proposed schemes are based on different amount of information expected from the user and focus on different tradeoffs between application wait and completion times. This approach improves the overall throughput by up to 42% when compared to the best of two existing schemes in the literature, TORQUE and MCT.

Appendix 1

Recent advancement on Big Data Processing

As the big data analytics market rapidly expands to include mainstream customers, which technologies are most in demand and promise the most growth potential? The answers can be found in *TechRadar: Big Data, Q1 2016*, a new Forrester Research report evaluating the maturity and trajectory of 22 technologies across the entire data life cycle. The winners all contribute to real-time, predictive, and integrated insights, what big data customers want now.



Here is my take on the 10 hottest big data technologies based on Forrester's analysis:

a) **Predictive analytics:** software and/or hardware solutions that allow firms to discover, evaluate, optimize, and deploy predictive models by analyzing big data sources to improve business performance or mitigate risk.

b) **NoSQL databases:** key-value, document, and graph databases.

c) **Search and knowledge discovery:** tools and technologies to support self-service extraction of information and new insights from large repositories of unstructured and structured data that resides in multiple sources such as file systems, databases, streams, APIs, and other platforms and applications.

d) **Stream analytics:** software that can filter, aggregate, enrich, and analyze a high throughput of data from multiple disparate live data sources and in any data format.

e) **In-memory data fabric:** provides low-latency access and processing of large quantities of data by distributing data across the dynamic random-access memory (DRAM), Flash, or SSD of a distributed computer system.

f) **Distributed file stores:** a computer network where data is stored on more than one node, often in a replicated fashion, for redundancy and performance.

g) **Data virtualization:** a technology that delivers information from various data sources, including big data sources such as Hadoop and distributed data stores in real-time and near-real time.

h) Data integration: tools for data orchestration across solutions such as Amazon Elastic MapReduce (EMR), Apache Hive, Apache Pig, Apache Spark, MapReduce, Couchbase, Hadoop, and MongoDB.

i) Data preparation: software that eases the burden of sourcing, shaping, cleansing, and sharing diverse and messy data sets to accelerate data's usefulness for analytics.

j) Data quality: products that conduct data cleansing and enrichment on large, high-velocity data sets, using parallel operations on distributed data stores and databases.

Forrester's TechRadar methodology evaluates the potential success of each technology and all 10 above are projected to have "significant success." In addition, each technology is placed in a specific maturity phase-from creation to decline-based on the level of development of its technology ecosystem. The first 8 technologies above are considered to be in the Growth stage and the last 2 in the Survival stage.

Forrester also estimates the time it will take the technology to get to the next stage and predictive analytics is the only one with a ">10 years" designation, expected to "deliver high business value in late Growth through Equilibrium phase for a long time." Technologies #2 to #8 above are all expected to reach the next phase in 3 to 5 years and the last 2 technologies are expected to move from the Survival to the Growth phase in 1-3 years.

Finally, Forrester provides for each technology an assessment of its business value-add, adjusted for uncertainty. This is based not only on potential impact but also on feedback and evidence from implementations and market reputation. Says Forrester: "If the technology and its ecosystem are at an early stage of development, we have to assume that its potential for damage and disruption is higher than that of a better-known technology." The first 2 technologies in the list above are rated as "high" business value-add, the next 2 as "medium," and all the rest "low," no doubt because of their emerging status and lack of maturity.

Why did I add to the list of hottest technologies two that are still in the Survival phase-data preparation and data quality? In the same report, Forrester also provides the following data from its Q4 2015 survey of 63 big data vendors:

What is the level of customer interest in each of the following capabilities? (% answering "very high")

Data preparation and discovery	52%
Data integration	48%
Advanced analytics	46%
Customeranalytics	46%
Data security	38%
In-memory computing	37%

While Forrester predicts that a few standalone vendors of data preparation will survive, it believes this is "an essential capability for achieving democratization of data," or rather, its analysis, letting data scientists spend more time on modeling and discovering insights and allowing more business users to have fun with data mining. Data Quality includes data security from the table above, in addition to other features ensuring decisions are based on reliable and accurate data. Forrester "expects that data quality will have significant success in the coming years as firms formalize a data certification process. Data certification efforts seek to guarantee that data meets expected standards for quality; security; and regulatory compliance supporting business decision-making, business performance, and business processes."

"Big Data" as a topic of conversation has reached mainstream audiences probably far more than any other technology buzzword before it. That did not help the discussion of this amorphous term, defined for the masses as "the planet's nervous system" or as "Hadoop" for technical audiences. Forrester's report helps clarify the term, defining big data as the ecosystem of 22 technologies, each with its specific benefits for enterprises and, through them, consumers.

Big data, specifically one its attributes, big volume, has recently gave rise to a new general topic of discussion, Artificial Intelligence. The availability of very large data sets is one of the reasons Deep Learning, a sub-set of AI, has been in the limelight, from identifying Internet cats to beating a Go champion. In its turn, AI may lead to the emergence of new tools for collecting and analyzing data.

Says Forrester: "In addition to more data and more computing power, we now have expanded analytic techniques like deep learning and semantic services for context that make artificial intelligence an ideal tool to solve a wider array of business problems. As a result, Forrester is seeing a number of new companies offering tools and services that attempt to support applications and processes with machines that mimic some aspects of human intelligence."

Prediction is difficult, especially about the future, but it's a (relatively) safe bet that the race to mimic elements of human intelligence, led by Google, Facebook, Baidu, Amazon, IBM, and Microsoft, all with very deep pockets, will change what we mean by "big data" in the very near future.

REFERENCES

[1] J.Gummaraju, L. Morichetti, M. Houston, B. Sander, B. R. Gaster and B. Zheng. 2010. Twin peaks: a software platform for heterogeneous computing on general-purpose and graphics processors. In 19th Conference on Parallel Architectures and Compilation Techniques.pp. 205-216, Vienna, Austria.

[2] S. Hong and H. Kim. 2010. An integrated gpu power and performance model. In Proceedings of the 37th

annual international symposium on Computer architecture, ISCA '10, pages 280–289, New York, NY, USA.

[3] V. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, *et al*. 2010. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. In: 37th International Symposium on Computer Architecture.pp. 451-460.

[4] J. Meng and K. Skadron. 2009. Performance modeling and automatic ghost zone optimization for iterative stencil loops on gpus. In: Proceedings of the 23rd international conference on Supercomputing, ICS '09, pp. 256-265.

[5] V. J. Jim´enez, L. Vilanova, I. Gelado, M. Gil, G. Fursin and N. Navarro. 2009. Predictive runtime code scheduling for heterogeneous architectures. In: 4th Conference on High Performance Embedded Architectures and Compilers.pp. 19-33.

[6] C.-K. Luk, S. Hong, and H. Kim. 2009.Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In: 42nd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 45-55.

[7] C. Augonnet, S. Thibault, R. Namyst and P. Wacrenier. 2009.StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. In Euro-Par Conference on Parallel Processing.pp. 863-874.

[8] J. Meng and K. Skadron. 2009. Performance modeling and automatic ghost zone optimization for iterative stencil loops on gpus. In: Proceedings of the 23rd international conference on Supercomputing, ICS '09.pp. 256-265.

[9] D. G. Feitelson, L. Rudolph and U. Schwiegelshohn. 2004. Parallel job scheduling - a status report. in JSSPP. pp. 1-16.

[10] A. C. Dusseau, R. H. Arpaci and D. E. Culler. 1996. Effective distributed scheduling of parallel workloads. SIGMETRICS Perform. Eval. Rev. 24: 25-36.

[11] V. K. Naik, M. S. Squillante and S. K. Setia. 1993. Performance analysis of job scheduling policies in parallel supercomputing environments. in Supercomputing '93, New York, NY, USA. pp. 824-833.

[12] G. Sabin, R. Kettimuthu, A. Rajan and P. Sadayappan. 2003. Scheduling of parallel jobs in a heterogeneous multi-site environment. In: Proceeding of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing. pp. 87-104.

[13] G. Sabin, V. Sahasrabudhe and P. Sadayappan. 2005. Assessment and enhancement of meta-schedulers for multi-site job sharing. in HPDC '05, Washington, DC, USA. pp. 144-153.

[14] R. K. *et al*. 2004. Single-isa heterogeneous multi-core architectures for multithreaded workload performance. in ISCA '04. Washington, DC, USA: IEEE Computer Society. pp. 64-75.

[15] M. Becchi and P. Crowley. 2006. Dynamic thread assignment on heterogeneous multiprocessor architectures. in CF '06, New York, NY, USA. pp. 29-40.

[16] SLURM: A highly scalable resource manager. https://computing.llnl.gov/linux/slurm/slurm.html.

[17] W. Jiang and G. Agrawal. 2011. Mate-cg: A mapreduce-like framework for accelerating data-intensive computations on heterogeneous clusters. Ohio State University Computer Science Dept., Tech. Rep.

[18] Torque Resource Manager. http://www.clusterresources.com/ products/torque-resource-manager.php.

[19] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen and R. F. Freund. 1999. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: Proceedings of the Eighth Heterogeneous Computing Workshop, ser. HCW '99. Washington, DC, USA: IEEE Computer Society. p. 30 [Online]. Available: http://dl.acm.org/citation.cfm?id=795690.797893

[20] A. B. Downey and D. G. Feitelson. 1999. The elusive goal of workload characterization. SIGMETRICS Perform. Eval. Rev. 26: 14-29. [Online]. Available: http://doi.acm.org/10.1145/309746.309750

[21] R. F. t. Freund. 1998. Scheduling resources in multi-user, heterogeneous, computing environments with smartnet. In: Proceedings of the Seventh Heterogeneous Computing Workshop, ser. HCW '98. Washington, DC, USA: IEEE Computer Society. p. 3

[Online]. Available: http://dl.acm.org/citation.cfm?id=795689.797878

[22] Ahmad M. Dhodhi and R. Ul-Mustafa. 1998. DPS: Dynamic priority scheduling heuristic for heterogeneous computing systems. IEE Proceedings - Computers and Digital Techniques. 145(6): 411-418. [Online]. Available: http://link.aip.org/link/?ICE/145/411/1

[23] Y. S. Kee, H. Casanova and A. Chien. 2004. Realistic modeling and synthesis of resources for computational grids. in SC ’04. pp. 54-63.

[24] Y. S. Kee, K. Yocum, A. A. Chien and H. Casanova. 2006. Improving grid resource allocation via integrated selection and binding. in SC ’06.

[25] F. Berman *et al.* 2005. New grid scheduling and rescheduling methods in the grads project. International Journal of Parallel Programming. pp. 209-229.

[26] S. K. Garg, R. Buyya, and H. J. Siegel. 2009. Scheduling parallel applications on utility grids: Time and cost trade-off management. in ACSC ’09.pp. 139-147.