PARALLEL PROGRAMMING IN BIG DATA ENVIRONMENT

Govindaraju G N¹, B K Raghavendra²

¹Research Scholar, VTU, Dept. of Computer Science & Engineering, KSSEM, Bengaluru, India
²Professor & Head, Dept. of Computer Science & Engineering, KSSEM, Bengaluru, India

ABSTRACT:

Cloud Computing is a emerging new approach for parallel data processing, most of the organizations have started to integrate frameworks for parallel data processing in their product. Making it easy for customers to access these services and to deploy their programs. With the Big Data technology the explosion and profusion of available data in a wide range of application domains rise up new challenges and opportunities in science, engineering and business. One major challenge is how to take advantage of the unprecedented scale of data typically of heterogeneous nature in order to acquire further insights and knowledge for improving the quality of the offered services. To exploit this new resource need to be scale up and scale out both our infrastructures and standard techniques. Our society is already having rich data, but the question remains is whether or not we have the conceptual tools to handle it more efficiently. In this paper we analyse the techniques and challenges for efficient parallel data processing. Big Data is the next frontier for innovation, competition, and productivity.

Keywords: Parallel programming, Big Data, MapReduce, Programming models.

[1] INTRODUCTION

Big Data is typically defines need for new techniques and tools in order to be able to process it. In order to use big data, we need programs which span multiple physical and virtual machines working together in order to process all of the data in a reasonable span of time. Getting programs on multiple machines to work together in an efficient way, so that each program knows which components of the data to process, and then being able to put the results from all of the machines together to make sense of a large pool of data that uses special programming techniques. Since it is typically much faster for programs to access data stored locally instead of over a network, the distribution of data across a cluster and how those machines are networked together are also important considerations which must be made when thinking about big data problems.

The uses of big data are almost as varied as they are large. Prominent examples including social media network, analyze their members' data to learn more about them and connect them with content and advertising relevant to their interests, or search engines looking at the relationship between queries and results to give better answers to users’ questions. Two of the largest sources of data in large quantities are transactional data, including everything from stock prices to bank data to individual merchants' purchase histories; and sensor data, much of it coming from what is commonly referred to as the Internet of Things (IoT).One of the best known methods for turning raw data into useful information is by what is known as MapReduce. MapReduce is a method for taking a large data set and performing computations on it across multiple computers, in parallel. It serves as a model for
how program, and is often used to refer to the actual implementation of this model. In essence, MapReduce consists of two parts. The Map function does sorting and filtering, taking data and placing it inside of categories so that it can be analysed. The Reduce function provides a summary of this data by combining it all together. While largely credited to research which took place at Google, MapReduce is now a generic term and refers to a general model used by many technologies.

[2] TOOLS USED TO ANALYSE BIG DATA

The most powerful and established tool for analysing big data is known as Apache Hadoop. Apache Hadoop is a framework for storing and processing data[1] in a large scale, and it is completely open source. Hadoop can run on commodity hardware, making it easy to use with an existing data center, or even to conduct analysis in the cloud. Hadoop is broken into four main parts:

- The Hadoop [5]Distributed File System (HDFS), which is a distributed file system designed for very high aggregate bandwidth
- YARN, a platform for managing Hadoop’s resources and scheduling programs which will run on the Hadoop infrastructure
- MapReduce, as described above, a model for doing big data processing
- And a common set of libraries for other modules to use.

Other tool which has been receiving a lot of attention recently is Apache Spark. The main selling point of Spark is that it stores much of the data for processing in memory, as opposed to that of disk, and using which we can get much faster result for certain kind of analysis. Depending on the operation, analysts may see results a hundred times faster or more. Spark can use the Hadoop Distributed File System, but it is also capable of working with other data stores, like Apache Cassandra or OpenStack Swift. It's also fairly easy to run Spark on a single local machine, making testing and development easier. There are countless open source solutions for working with big data, many of them specialized to provide optimal features and performance for a specific niche or for specific hardware configurations. And as big data continues to grow in size and importance, the list of open source tools for working with it will certainly continue to grow alongside.

[3] PARALLEL PROGRAMMING

Most of the data comes from software logs, cameras, microphones, RFID readers, wireless sensor networks and so on. These machines generate high speed data and their production rates will grow exponentially with Moore’s Law. Storing this data is cheap, and it can be mined for valuable information. In this context, there is some good news for parallel programming. Data analysis software parallelizes fairly naturally. In fact, software written in SQL has been running in parallel for more than 20 years. But with “Big Data” now becoming a reality, more programmers are interested in building programs on the parallel model and they often find SQL an unfamiliar and restrictive way to wrangle data and write code. The biggest game-changer to come along is MapReduce, the [3]parallel programming framework that has gained prominence thanks to its use at web search companies.

To understand the parallel software, let’s look at what the computer industry has already accomplished. The branch of parallel research that has had the most success in the field is parallel databases. Rather than requiring the programmer to unravel an algorithm into separate threads to be run on separate cores, parallel databases let them chop up the input data tables into pieces, and pump each piece through the same single-machine program on each processor. This “parallel dataflow” model makes programming a parallel machine as easy as programming a single machine. And it works on “shared-nothing” clusters of computers in a data center: The machines involved can communicate via simple streams of data messages, without a need for an expensive shared RAM or disk infrastructure.

SQL provides a higher-level language that is more flexible and optimizable, but less familiar to many programmers. MapReduce largely asks programmers to write traditional code, in languages like C, Java, Python and Perl. In addition to its familiar syntax, MapReduce allows programs to be written to and read from traditional files in a file system, rather than requiring database schema definitions. MapReduce is such a compelling entryway into parallel programming that it is being used to nurture a new generation of parallel programmers. Every Berkeley computer science undergraduate now learns MapReduce, and other schools have undertaken similar programs. Industry is eagerly supporting these efforts.
[4] PROGRAMMING MODELS FOR BIG DATA

The large volume of data that Internet services work with has led to interest in parallel processing on commodity hardware. There is a sense among some Big Data leaders that the infrastructure challenge has largely been met. But as the volumes of data swell into exabyte territory for more and more organizations, the biggest challenge is going to be devising ways to mine the data and make sense of it all, or at least in part to turn data into knowledge, and knowledge into wisdom. The leading example is Google, which uses its MapReduce framework[8] to process over 20 peta bytes of data per day.

The Apache Hadoop is used to analyse massive amounts of data without necessarily indulging in expensive proprietary hardware or software. However, adoption of Hadoop alone isn’t necessarily helping businesses make smarter decisions or discover completely new facts. The power of scalable infrastructure needs to be supplemented with nifty data mining and machine learning tools, better visualization of results, and easier ways to track and analyse the findings over a period of time. Besides, there is the entire realm of real-time analytics, which is beyond the batch oriented nature of Hadoop. Pig, Sawzall, Microsoft’s Dryad, and others functional languages are in development. They can be classified by terms like high throughput computing (HTC) or many-task computing (MTC), depending on the amount of data and the number of tasks involved in the computation. Although these systems differ in design, the programming models they provide share similar objectives, namely hiding the hassle of parallel programming, fault tolerance and execution optimizations from the developer. Developers can typically continue to write sequential programs.

The processing framework then takes care of distributing the program among the available nodes and executes each instance of the program on the appropriate fragment of data. Conceptually, many of the Big Data analysis can be thought of as single program multiple data (SPMD) algorithms or a collection thereof. These SPMDs can be implemented using different parallelization techniques such as threads, MPI, MapReduce, and mash-up or workflow technologies, yielding different performance[9] and usability characteristics. Most techniques try to explore an “almost embarrassingly parallel” style of parallelism. In this case, the parallel independent sets of data lead to independent “maps” (processing), which are followed by a reduction (e.g., to give histograms in particle physics, or aggregated queries in web searches). The excellent quality of service (QoS) and ease of programming provided by the MapReduce programming model has gained itself a lot of traction for this type of problem. However, the architectural and performance limitations of the current MapReduce architectures make their use questionable for many applications (e.g., machine learning algorithms need iterative closely coupled computations). More general workflow or dataflow paradigm (which is seen in Dryad and MapReduce extensions) is always valuable, and we explore such solutions in the following paragraphs.

[5] RUNTIME ENVIRONMENTS FOR BIG DATA

High level languages (i.e., for parallel programming) have been a holy grail for computer science research, but lately researchers made a lot of progress in the area of runtime environments. There is much similarity between parallel and distributed run times, with both supporting messaging with different properties (several such choices are presented in Fig. 1, for different hardware and software models). The hardware support of parallelism/concurrency varies from shared memory multicore, closely coupled clusters, and higher-latency (possibly lower bandwidth) distributed systems.
Fig. 1 Combinations of processes/threads and intercommunication mechanisms. (a) MPI is long running processes with Rendezvous for message exchange/synchronization. (b) Yahoo’s Hadoop uses short running processes communicating via disk and tracking processes. (c) Microsoft’s Dryad uses short running processes communicating via pipes, disk or shared memory between cores. (d) Web Services send irregular point-to-point messages between short or long running services.

The Fig. 1 shows combinations of processes/threads and intercommunication mechanism. (a) MPI is long running processes with Rendezvous for message exchange/synchronization. (b) Yahoo’s Hadoop uses short running processes communicating via disk and tracking processes. (c) Microsoft’s Dryad uses short running processes communicating via pipes, disk or shared memory between cores. (d) Web Services send irregular point-to-point messages between short or long running services coordination (communication/synchronization) of the different execution units vary from threads (with shared memory on cores), MPI (between cores or nodes of a cluster), workflow or mash-ups linking services together, and the new generation of data intensive programming systems typified by Hadoop (implementing MapReduce) or Dryad.

Short running threads can be spawned up in the context of persistent data in memory and have modest overhead. Short running processes (i.e., implemented as stateless services) are seen in Dryad and Hadoop. Also, various runtime platforms implement different patterns of operation. In Iteration-based platforms, the results of one stage are iterated many times. This is typical of most MPI style algorithms. In Pipelining-based platforms, the results of one stage (e.g., Map or Reduce operations) are forwarded to another. This is functional parallelism typical of workflow applications. An important ambiguity in parallel/distributed programming models/runtime comes from the fact that today both the parallel MPI style parallelism and the distributed Hadoop/Dryad/Web Service/Workflow models are implemented by messaging. This is motivated by the fact that messaging avoids errors seen in shared memory thread synchronization. MPI is a perfect example of runtimes crossing different application characteristics. MPI gives excellent performance and ease of programming for MapReduce, as it has elegant support for general reductions. However, it does not have the fault tolerance and flexibility of Hadoop or Dryad. Further MPI is designed for local computing; if the data is stored in a compute node’s memory, that node’s CPU is responsible for computing it. Hadoop and Dryad combine this idea with the notion of taking the computing to the data. A (non-comprehensive) presentation of technologies in use today for Big Data processing[10] is presented in Fig. 2.

[6] MAPREDUCE AND HADOOP

MapReduce (MR) emerged as an important programming model for large-scale data parallel applications. The MapReduce model popularized by Google is attractive for ad-hoc parallel processing of arbitrary data, and is today seen as an important programming model for large-scale data-parallel applications such as web indexing, data mining and scientific simulations, as it provides a simple model through which users can express relatively sophisticated distributed programs. MapReduce breaks a computation into small tasks that run in parallel on multiple machines, and scales easily to very large clusters of inexpensive commodity computers.

![Diagram of Big Data analysis tools and frameworks](image)

The Fig. 2 shows an ecosystem of Big Data analysis tools and frameworks. A MR program consists only of two functions, called Map and Reduce, written by a user to process key/value data pairs. The input data set is stored in a collection of partitions in a distributed file system deployed on each node in the cluster. The program is then injected into a distributed processing framework and executed in a manner to be described. The Map function reads a set of “records” from an input file, does some filtering and/or transformations, and then outputs a set of intermediate records in the form of new key/value pairs. As the Map function produces these output records, a “split” function partitions the records into R disjoint buckets by applying
a function to the key of each output record. This split function is typically a hash function, though any deterministic function will suffice. Each map bucket is written to the processing node’s local disk. The Map function terminates having produced \( R \) output files, one for each bucket.

In general, there are multiple instances of the Map function running on different nodes of a compute cluster. The term instance is used to refer to a unique running invocation of either the Map or Reduce function. Each Map instance is assigned a distinct portion of the input file by the MR scheduler to process. If there are \( M \) such distinct portions of the input file, then there are \( R \) files on disk storage for each of the \( M \) Map tasks, for a total of \( M \times R \) files \( F_{i,j} \), where \( 1 \leq i \leq M \), \( 1 \leq j \leq R \). The key observation is that all Map instances use the same hash function; thus, all output records with the same hash value are stored in the same output file. The second phase of a MR program executes \( R \) instances of the Reduce program. The input for each Reduce instance \( R_j \) consists of the files \( F_{i,j} \), \( 1 \leq j \leq M \). These files are transferred over the network from the Map nodes’ local disks. Again, all output records from the Map phase with the same hash value are consumed by the same Reduce instance, regardless of which Map instance produced the data. Each Reduce instance processes or combines the records assigned to it in some way, and then writes records to an output file (in the distributed file system), which forms part of the computation’s final output.

The input data set exists as a collection of one or more partitions in the distributed file system. It is the job of the MR scheduler to decide how many Map instances to run and how to allocate them to available nodes. Likewise, the scheduler must also decide on the number and location of nodes running Reduce instances. The MR central controller is responsible for coordinating the system activities on each node. A MR program finishes execution once the final result is written as new files in the distributed file system.

A key benefit of Map Reduce is that it automatically handles failures, hiding the complexity of fault-tolerance from the programmer. If a node crashes, MapReduce automatically reruns its tasks on a different machine. Similarly, if a node is available but is performing poorly, a condition called a straggler, MapReduce runs a speculative copy of its on another machine to finish the computation faster. Without this mechanism (known as “speculative execution” not to be confused still with speculative execution at the OS or hardware level for branch prediction), a job would be as slow as the misbehaving task. In fact, Google has noted that in their implementation speculative execution can improve job response times by 44%.

[7] PIG AND HIVE

During the 1970s, the database research community engaged in a contentious debate whether a program to access data in a DBMS should be written either by 1) stating what one wants, rather than presenting an algorithm for how to get it (the Relational model), or 2) presenting an algorithm for data access (the Codasyl case). In the end, the former view prevailed and the last 30 years is a testament to the value of relational database systems. Programs in high-level languages, such as SQL, are easier to write, easier to modify, and easier for a new person to understand. Codasyl was criticized for being similarly to an assembly language for DBMS access.

MapReduce-like programming is today seen somewhat analogous to Codasyl programming: the developer has to write algorithms in a low-level language in order to perform record-level manipulation. However, evidence from the MapReduce community suggests that there is widespread sharing of MapReduce code fragments to do common tasks, such as joining data sets. To alleviate the burden of having to re-implement repetitive tasks, the MapReduce community is migrating high-level languages on top of the current interface to move such functionality into the run time. Pig and Hive are two notable projects in this direction.

[8] CHALLENGES

Technology and industry are currently undergoing a profound transformation: large-scale, diverse data sets and streams[6] (derived from sensors, the web, transactions, or complex simulations) present a huge opportunity for data-driven decision making. [4]Besides the massive volume of data, “Big Data” will come in a variety of data formats (e.g., sensor data, text, audio, and video), origin, quality, and so forth. The data are created at an ever-increasing rate, making the subject of velocity (i.e., the time window in which the data will need to be processed) crucial in order to arrive at actionable information in a timely manner. Moreover, as the data come from different sources of different quality and trustworthiness, another crucial aspect of data analytics is to assess the veracity of
an analysis result, i.e., its correctness and credibility. Furthermore, the visualization and interactive analysis of huge, changing data sets still presents numerous challenges for data analysts.

Recent advances in computer technology and processing paradigms have created new tools and technologies at the forefront of “Big Data”. On top of such tools, the “four V’s” put pressure on developers to become comfortable with new programming paradigms. And the domain is continuously attracting significant attention in society, industry, and science. Novel statistical and mathematical algorithms, prediction techniques, and modelling methods, new approaches for data collection and integration, data analysis[7] and compression, enhanced technologies for processing and sharing data and information, as well as novel languages for the declarative specification and automatic optimization and parallelization of complex data analysis programs are needed to simultaneously cope with the volume, velocity, variety, and veracity aspects of data analytics. This capability will enable a paradigm shift in scientific and commercial applications.

Advances in information processing, integration, signal processing, machine learning, data mining, compression, and visualization will open up new ways of extracting useful, reliable, and verified information in a timely fashion from huge and diverse data sets. The “NoSQL” approach brings with it a range of issues. Integrating Big Data of various media types and providing low latency and high velocity analytics with trustworthy information is a major challenge not met by existing data management systems. Big Data analytics systems must be able to ingest data from various media types, in particular audio and video streams, at increasing speeds, while at the same time already enabling the analysis of the data in a continuous fashion. Scalable, easy-to-use database or data analysis systems and new algorithms and analysis paradigms must be developed that address such different aspects and requirements simultaneously.

Some examples are scalable online analysis, computational linguistics, statistics, and machine learning algorithms. The specification and automatic optimization of data analysis programs that include iterations and complex user-defined functions in a scalable way for a variety of hardware platforms such as SIMD, clusters, or many core CPUs, as well as complex hardware architectures, such as NUMA still presents many research challenges. Other examples of challenges are new declarative languages and methods for the scalable processing and optimization of complex data analysis programs, such as active learning techniques or interactive entity linking techniques as part of computational linguistics, which must consider latency as a hard constraint, while ensuring a certain degree of trustworthiness in the case of contradicting, missing, or incomplete information, even in resource-constrained environments.

[9] CONCLUSIONS

With Cloud Computing emerging as a promising new approach for ad-hoc parallel data processing, major companies have started to integrate frameworks for parallel data processing in their product portfolio, making it easy for customers to access these services and to deploy their programs. There are various mechanisms that generate Big Data in daily life, which is a potential gold mine for understanding access patterns and increasing revenue. The explosion and profusion of available data in wide range of application domains rise up new challenges and opportunities in a different disciplines ranging from science, engineering and Technology. One major challenge is how to take advantage of the unprecedented scale of data typically of heterogeneous nature in order to acquire further insights and knowledge for improving the quality of the offered services. To exploit this new resource, we need to scale up and scale out both our infrastructures and standard techniques. In this paper we analysed opportunities and challenges for efficient parallel data processing. Big Data is the next frontier for innovation, competition, and productivity.

REFERENCES


[10] Parallel Data Processing with MapReduce: A Survey, SIGMOD Record, December 2011 (Vol. 40, No. 4)