Big Data

Storage, Sharing, and Security

Edited by Fei Hu

CRC Press
Taylor & Francis Group
AN AUERBACH BOOK

Big Data

Storage, Sharing, and Security

Big Data

Storage, Sharing, and Security

Edited by **Fei Hu**

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN AUERBACH BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

International Standard Book Number-13: 978-1-4987-3487-5 (eBook - PDF)

For Gloria, Edwin & Edward (twins)......

Contents

Preface

Big Data is one of the hottest topics today because of the large-scale data generation and distribution in computing products. It is tightly integrated with other cutting-edge networking technologies, including cloud computing, social networks, Internet of things, and sensor networks. Characteristics of Big Data may be summarized as four Vs, that is, volume (great volume), variety (various modalities), velocity (rapid generation), and value (huge value but very low density). Many countries are paying high attention to this area. As an example, in the United States in March 2012, the Obama Administration announced a US$200 million investment to launch the "Big Data Research and Development Plan," which was a second major scientific and technological development initiative after the "Information Highway" initiative in 1993.

Because Big Data is a relatively new field, there are many challenging issues to be addressed today: (1) *Storage*—How do we aggregate heterogeneous types of data from numerous sources, and then use fast database management technology to store the Big Data? (2) *Sharing*—How do we use cloud computing to share the Big Data among large groups of people? (3) *Security*—How do we protect the privacy of Big Data during the network sharing? This book will cover the above 3S designs, through the detailed description of the concepts and implementations.

This book is unlike any other similar books. Because Big Data is such a new field, there are very few books covering its implementation. Although a few similar books are already published, they are mostly about the basic concepts and society impacts. They are thus not suitable for R&D people. Instead, this book will discuss Big Data management from an R&D perspective.

Targeted Audiences: (1) Industry—company engineers can use this book as a reference for the design of Big Data processing and protection. There are many practical design principles covered in the chapters. (2) Academia—researchers can gain much knowledge on the latest research topics in this area. Graduate students can resolve many issues by reading the chapters. They will gain a good understanding of the status and trend of Big Data management.

Book Architecture: The book consists of two sections:

- *Section I. Big Data management*: In this section we cover the following important topics:

  - *Spatial management*: In many applications and scientific studies, there is a growing need to manage spatial entities and their topological, geometric, or geographic properties. Analyzing such large amounts of spatial data to derive values and guide decision making has become essential to business success and scientific progress.

    - *Data transfer*: A content delivery network with large data centers located around the world requires Big Data transfer for data migration, updates, and backups. As cloud computing becomes common, the capacity of the data centers and both the intranetwork and internetwork of those data centers increase.
    - *Data processing*: Dealing with "Big Data" problems requires a radical change in the philosophy of the organization of information processing. Primarily, the Big Data approach has to modify the underlying computational model to manage uncertainty in the access to information items in a huge nebulous environment.

- *Section II. Big Data Security*: Security is a critical aspect after Big Data is integrated with cloud computing. We will provide technical details on the following aspects:
    - *Security*: To achieve a secure, available, and reliable Big Data cloud-based service, we not only present the state-of-the-art of Big Data cloud-based services, but also a novel architecture to manage reliability, availability, and performance for accessing Big Data services running on the cloud.
    - *Privacy*: We will examine privacy issues in the context of Big Data and potential data mining of that data. Issues are analyzed based on the emerging unique characterizations associated with Big Data: the Big Data Lake, "thing" data, the quantified self, repurposed data, and the generation of knowledge from unstructured communication data, that is, Twitter Tweets. Each of those sets of emerging issues is analyzed in detail for their potential impact on privacy.
    - *Accountability*: Accountability of user data access on a specific application helps in monitoring, controlling, and assessing data usage by the user for the application. Data loss is the main source of leaking information that may possibly compromise the privacy of individual and/or organization. Therefore, the naive question is, "how can data leakages be controlled and detected?" The simple answer to this would be audit logs and effective measures of data usage.

The chapters have detailed technical descriptions of the models, algorithms, and implementations of Big Data management and security aspects. There are also accurate descriptions on the state-of-the-art and future development trends of Big Data applications. Each chapter also includes references for readers' further studies.

Thank you for reading this book. We believe that it will help you with the scientific research and engineering design of Big Data systems. We welcome your feedback.

Fei Hu
University of Alabama, Tuscaloosa, Alabama

Editor

Dr. Fei Hu is currently a professor in the Department of Electrical and Computer Engineering at the University of Alabama, Tuscaloosa, Alabama. He earned his PhD degrees at Tongji University (Shanghai, China) in the field of signal processing (in 1999), and at Clarkson University (New York) in electrical and computer engineering (in 2002). He has published over 200 journal/conference papers and books. Dr. Hu's research has been supported by the U.S. National Science Foundation, Cisco, Sprint, and other sources. His research expertise can be summarized as *3S: Security, Signals, Sensors*: (1) *Security*—This deals with overcoming different cyber attacks in a complex wireless or wired network. His current research is focused on cyber-physical system security and medical security issues. (2) *Signals*—This mainly refers to *intelligent signal processing*, that is, using machine learning algorithms to process sensing signals in a smart way to extract patterns (i.e., pattern recognition). (3) *Sensors*—This includes microsensor design and wireless sensor networking issues.

This page intentionally left blank

Contributors

Emad Abd-Elrahman
RST Department
Telecom Sudparis
Evry, France

Ablimit Aji
Analytics Lab
Database Systems
Hewlett Packard Labs
Palo Alto, California

Usamah AlGemili
Department of Computer Science
George Washington University
Washington, DC

Adi Alhudhaif
Department of Computer Science
Prince Sattam bin Abdulaziz University
Al-Kharj, Saudi Arabia

Faisal Alsaby
Department of Computer Science
George Washington University
Washington, DC

Nadia Bennani
INSA-Lyon
LIRIS Department
University of Lyon
Lyon, France

Simon Y. Berkoich
COMStar Computing Technology Institute
and
Department of Computer Science
George Washington University
Washington, DC

Nevil Brownlee
Department of Computer Science
University of Auckland
Auckland, New Zealand

Lionel Brunie
INSA-Lyon
LIRIS Department
University of Lyon
Lyon, France

Thomas Cerqueus
INSA-Lyon
LIRIS Department
University of Lyon
Lyon, France

Ernesto Damiani
Department of Computer Technology
University of Milan
Milan, Italy

Manik Lal Das
Dhirubhai Ambani Institute
of Information and Communication
Technology
Gujarat, India

Vijay Gadepally
MIT Lincoln Laboratory
Lexington, Massachusetts

Ibrahim A. Gomaa
Computer and Systems Department
National Telecommunication Institute
Cairo, Egypt

Fouad Amine Guenane
ENST
Telecom ParisTech
Paris, France

Benjamin Habegger
INSA-Lyon
LIRIS Department
University of Lyon
Lyon, France

Ariel Hamlin
MIT Lincoln Laboratory
Lexington, Massachusetts

Omar Hasan
INSA-Lyon
LIRIS Department
University of Lyon
Lyon, France

Yuh-Jong Hu
Department of Computer Science
National Chengchi University Taipei
Taipei, Taiwan

Rasheed Hussain
Department of Computer Science
and Engineering
Hanyang University
Ansan, South Korea

Jeremy Kepner
MIT Lincoln Laboratory
Lexington, Massachusetts

Donghyun Kim
Department of Mathematics and Physics
North Carolina Central University
Durham, North Carolina

Harald Kosch
Department of Informatics and Mathematics
University of Passau
Passau, Germany

Duoduo Liao
COMStar Computing Technology Institute
Washington, DC

Dongxi Liu
CSIRO
Clayton South Victoria, Australia

Wen-Yu Liu
Department of Computer Science
National Chengchi University Taipei
Taipei, Taiwan

Jianguo Lu
School of Computer Science
University of Windsor
Ontario, Canada

Aniket Mahanti
Department of Computer Science
University of Auckland
Auckland, New Zealand

Michele Nogueira
Department of Informatics
NR2—Federal University of Parana
Curitiba, Brazil

Heekuck Oh
Department of Computer Science
and Engineering
Hanyang University
Ansan, South Korea

Daniel E. O'Leary
University of Southern California
Los Angeles, California

Albert Reuther
MIT Lincoln Laboratory
Lexington, Massachusetts

Jun Sakuma
Department of Computer Science
University of Tsukuba
Tsukuba, Japan

Nabil Schear
MIT Lincoln Laboratory
Lexington, Massachusetts

Ahmed Serhrouchni
ENST
Telecom ParisTech
Paris, France

Emily Shen
MIT Lincoln Laboratory
Lexington, Massachusetts

Junggab Son
Department of Mathematics and Physics
North Carolina Central University
Durham, North Carolina

Mayank Varia
Boston University
Boston, Massachusetts

Dong Wang
Department of Computer Science
and Engineering
University of Notre Dame
Notre Dame, Indiana

Fusheng Wang
Department of Biomedical Informatics
and
Department of Computer Science
Stony Brook University
Stony Brook, New York

Shenlu Wang
School of Computer Science and Engineering
University of New South Wales
Sydney, Australia

Yan Wang
School of Information
Central University
of Finance and Economics
Beijing, China

J. Gerard Wolff
CognitionResearch.org
Menai Bridge, United Kingdom

Sophia Yakoubov
MIT Lincoln Laboratory
Lexington, Massachusetts

Maryam Yammahi
Department of Computer Science
George Washington University
Washington, DC

and

College of Information
Technology
United Arab Emirates University
Al Ain, United Arab Emirates

Arkady Yerukhimovich
MIT Lincoln Laboratory
Lexington, Massachusetts

Se-young Yu
Department of Computer Science
University of Auckland
Auckland, New Zealand

John Zic
CSIRO
Clayton South Victoria, Australia

I

BIG DATA MANAGEMENT: STORAGE, SHARING, AND PROCESSING

Chapter 1

Challenges and Approaches in Spatial Big Data Management

Ablimit Aji

Fusheng Wang

CONTENTS

1.1 Introduction

Advancements in computer technology and the rapid growth of the Internet have brought many changes to society. More recently, the Big Data paradigm has disrupted many industries ranging from agriculture to retail business, and fundamentally changed how businesses operate and make decisions at large. The rise of Big Data can be attributed to two main reasons:

First, high volumes of data generated and collected from *devices*. The rapid improvement of high-resolution data acquisition technologies and sensor networks have enabled us to capture large amounts of data at an unprecedented scale and rate. For example, the GeoEye-1 satellite has the highest resolution of any commercial imaging system and is able to collect images with a ground resolution of 0.41 m in the panchromatic or black and white mode [1]; the Sloan Digital Sky Survey (SDSS), with a rate of about 200 GB per night, has amassed more than 140 TB of information [5]; and the modern medical imaging scanners can capture the micro-anatomical tissue details at the billion pixel resolution [13].

Second, traces of *human* activity and crowd-sourcing efforts facilitated by the Internet. The proliferation of cost-effective and ubiquitous positioning technologies, mobile devices, and sensors have enabled us to collect massive amounts of spatial information of human and wildlife activity. For example, FourthSquare—a popular local search and discovery service—allow users to *check-in* at more than 60 million venues, and so far has more than 6 billion check-ins [2]. Driven by the business potential, more and more businesses are providing services that are *location-aware*. At the same time, the Internet has made remote collaboration so easy that, now, a crowd can even generate a free mapping of the world autonomously. OpenStreetMap [3] is a large collaborative mapping project, which is generated by users around the globe, and it has more than two million registered users as of this writing.

In many applications and scientific studies, there is a growing need to manage spatial entities and their topological, geometric, or geographic properties. Analyzing such large amounts of spatial data to derive values and guide decision making have become essential to business success and scientific progress. For example, location-based social networks (LBSNs) are utilizing large amounts of user location information to provide geo-marketing and recommendation services. Social scientists are relying on such data to study dynamics of social systems and understand human behavior. Epidemiologists are combining such spatial data with public health data to study the patterns of disease outbreak and spread. In all those domains, spatial Big Data analytics infrastructure is a key enabler.

Over the last decade, the Big Data technology stack and the software ecosystem has evolved to cope with most common use cases. However, modern data-intensive spatial applications require a different approach to be able to handle unique requirements of spatial Big Data.

In the rest of this chapter, first we provide examples of data-intensive spatial applications, and discuss the unique challenges that are common to them. Then, we present major research efforts, data-intensive computing techniques, and software systems that are intended to address these challenges. Lastly, we conclude the chapter with a discussion on future outlook of this area.

1.2 Big Spatial Data and Applications

The rapid growth of spatial data is driven not only by conventional applications, but also by emerging scientific applications and large internet services that have become data-intensive and compute-intensive.

1.2.1 Spatial analytics for derived scientific data

With the rapid improvement of data acquisition technologies such as high-resolution tissue slide scanners and remote sensing instruments, it has become more efficient to capture extremely large spatial data to support scientific research. For example, digital pathology imaging has become an emerging field in the past decade, where examination of high-resolution images of tissue specimens enables novel, more effective ways of screening for disease, classifying disease states, understanding its progression, and evaluating the efficacy of therapeutic strategies. In clinical environment, medical professionals have been relying on the manual judgment from pathologists—a process inherently subject to human bias—to diagnose, and understand the disease condition.

Today, *in silico* pathology image analysis offers a means of rapidly carrying out quantitative, reproducible measurements of micro-anatomical features in high-resolution pathology images and large image datasets. Medical professionals and researchers can use computer algorithms to calculate the distribution of certain cell types, and perform associative analysis with other data such as patient genetic composition and clinical treatment.

Figure 1.1 shows a protocol for *in silico* pathology image analysis pipeline. From left to the right, the sub-figures represent: glass slides, high-resolution image scanning, whole slide images, and automated image analysis. The first three steps are data acquisition processes that are mostly done in a pathology laboratory environment, and the final step is where the computerized analysis is performed. In the image analysis step, regions of micro-anatomical objects (millions per image) such as nuclei and cells are computed through image segmentation algorithms, represented with their boundaries, and image features are extracted from these objects. Exploring the results of such analysis involves complex queries such as spatial cross-matching, overlay of multiple sets of spatial objects, spatial proximity computations between objects, and queries for global spatial pattern discovery. These queries often involve billions of spatial objects and heavy geometric computations.

Scientific simulation also generates large amounts of spatial data. Scientists often use models to simulate natural phenomena, and analyze the simulation process and data. For example, earth science uses simulation models to help predict the ground motion during earthquakes. Ground motion is modeled with an octree-based hexahedral mesh, using soil density as input. Simulation tools calculate the propagation of seismic waves through the Earth by approximating the solution to the wave equation at each mesh node. During each time step, for each node in the mesh, the simulator calculates the node velocity in spatial directions, and records those information to the primary storage. The simulation result is a spatio temporal earthquake data set describing the ground velocity response [6]. As the scale of the experiment increases, the resulting dataset also increases, and scientists often struggle to query and manage such large amounts of spatio temporal data in an efficient and cost-effective manner.

1.2.2 GIS and social media applications

Volunteered geographic information (VGI) further enriched global information system (GIS) world with massive amounts of user-generated geographical and social data. VGI is a special case of the larger Internet phenomenon—user-generated content—in the GIS domain. Everyday Internet users can provide, modify, and share geographical data using interactive online

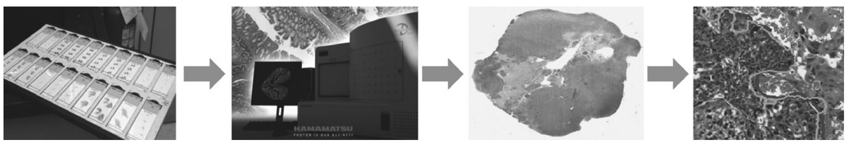


Figure 1.1: Derived spatial data in pathology image analysis.

services such as OpenStreetMap [3], Wikimapia, GoogleMap, GoogleEarth, and Microsoft's Virtual Earth. The spatial information needs to be constantly analyzed and corroborated to track changes, and understand the current status. Most often, a spatial database system is used to perform such analysis.

Recently, the explosive growth of social media applications contributed massive amounts of user-generated geographic information in the form of tweets, status updates, check-ins, Waze, and traffic reports. Furthermore, if such geospatial information is not available, automated geo tagging/coding tools can infer and assign an approximate location to those contents. Analysis of such large amounts of data has implications for many applications—both commercial and academic. In [11] authors have used the geospatial information to investigate the relationship between the geographic location of protestors attending demonstrations in the 2013 Vinegar protests in Brazil and the geographic location of users that tweeted the protests. Another example is location-based targeted advertising [24] and recommendation [18]. Those online services and GIS systems are backed by conventional spatial database systems that are optimized for different application requirements.

1.3 Challenges and Requirements

Modern data-intensive spatial analytics applications are different from conventional applications in several aspects. They involve the following:

- *Large volumes of multidimensional data*: Conventional warehousing applications deal with data generated from business transactions. As a result, the underlying data (such as numbers and strings) tend to be relatively *simple* and *flat*. However, this is not the case for the spatial applications which deal with massive amounts of geometry shapes and spatial objects. For example, a typical whole slide pathology contains more than 20 billion pixels, millions of objects, and 100 million derived image features. A single study may involve thousands of images analyzed with dozens of algorithms—with varying parameters—to generate many different result sets to be compared and consolidated, at the scale of tens of terabytes. A moderate-size healthcare operation can routinely generate thousands of whole slide images per day, leading to petabytes of analytical results per year. A single 3D pathology image could come from a thousand slices and take 1 TB storage, containing several millions to 10 millions of derived 3D surface objects.
- *High computation complexity*: Most spatial queries involve multidimensional geometric computations that are often compute-intensive. While spatial filtering through minimum bounding rectangles (MBRs) can be accelerated through spatial access methods, spatial refinements such as polygon intersection verification are highly expensive operations. For example, spatial join queries such as spatial cross-matching or spatial overlay can be very expensive to process. This is mainly due to the polynomial complexity of many geometric computation methods. Such compute-intensive geometric computation, combined with the large volumes of Big Data requires a high-performance solution.
- *Complex spatial queries*: Spatial queries are complex to express in current spatial data analytics systems. Most scientific researchers and spatial application developers are often interested in running queries that involve complex spatial relationships such as nearest neighbor query, and spatial pattern queries. Such queries are not well supported in current spatial database systems. Frequently, users are forced to write database user-defined functions to be able to perform the required operations. SQL—structured query language—has gained tremendous momentum in the relational database field

and become the de facto standard for querying the data. While most spatial queries can be expressed in SQL, due to the structural differences in the programming model, efficient SQL-based spatial queries are often hard to write and requires considerable optimization efforts.

A major requirement for the spatial analytics systems is fast query response. Scientific research or analytics in general, is an iterative and exploratory process in which large amounts of data can be generated quickly for the initial prototyping and validation. This requires a scalable architecture that can query spatial data on a large scale. Another requirement is to support queries on a cost-effective architecture such as commodity clusters or cloud environments. Meanwhile, scientific researchers and application developers often prefer expressive query languages over programming API, without worrying about how the queries are translated, optimized, and executed. With the rapid improvement of instrument resolutions, the increased accuracy of data analysis methods, and the massive scale of observed data, complex spatial queries have become increasingly compute-intensive and data-intensive.

1.4 Spatial Big Data Systems and Techniques

Two mainstream approaches for large-scale data analysis are parallel database systems [15] and MapReduce-based systems [14]. Both approaches share certain common design elements: they both employ a shared-nothing architecture [25], and deployed on a cluster of independent nodes via a high-speed interconnecting network; both achieve parallelism by partitioning the data and processing the query in parallel on each partition.

However, parallel database approach has major limitations on managing and querying spatial data at massive scale. Parallel database management systems (DBMSs) tend to reduce the I/O bottleneck through partitioning of data on multiple parallel disks and are not optimized for computational-intensive operations such as spatial and geometric computations. Partitioned parallel DBMS architecture often lacks effective spatial partitioning to balance data and task loads across database partitions. While it is possible to induce a spatial partitioning, fixed grid tiling, for example, and map such partitioning to one dimensional attribute distribution key, such an approach fails to handle boundary objects for accurate query processing. Scaling out spatial queries through a parallel database infrastructure is possible while being costly, and such approach is explored in [27,28]. More recently, Spark [31] has emerged as a new data processing framework for handling iterative and interactive workloads.

Due to both computational intensity and data intensity of spatial workloads, large-scale parallelization often holds the key to achieving high-performance spatial queries. As the cloud-based cluster computing technology gets mature and economically scalable, MapReduce-based systems offer an alternative solution for data and compute-intensive spatial analytics at large scale. Meanwhile, parallel processing of queries rely on effective data partitioning to scale. Considering that spatial workloads are often compute-intensive [7,22], how to utilize hardware accelerators for query co-processing is a very promising technique as modern computer systems are embracing heterogeneous architecture that combines graphics processing unit (GPU) and CPU [12]. In the rest of the chapter, we elaborate each of these techniques in greater detail, and summarize state-of-the-art approaches and systems.

1.4.1 MapReduce-based spatial query processing

MapReduce is a very scalable parallel processing framework that is designed to process flat unstructured data. However, it is not particularly well suited to process multidimensional spatial objects, and several systems have emerged over the past few years to fill this gap. Well-known systems and prototypes include HadoopGIS [8–10], SpatialHadoop [16,17], Parallel Secondo [20,21], and GIS tools for Hadoop [4,30]. These systems are based on the open source implementation of MapReduce—Hadoop, and provides similar analytics functionality. However, they differ in implementation details and architecture: HadoopGIS and SpatialHadoop are pure MapReduce-based query evaluation systems; Parallel Secondo is a hybrid system that combines a database engine with MapReduce; and GIS tools for Hadoop is a functional extension of Hive [26] with user-defined functions.

MapReduce relies on the partitioning of data to process them in parallel, and it is the key for a high-performance system. In the context of large-scale spatial data analytics, an intuitive approach is to partition the dataset based on the spatial attribute, and assign spatial objects to partitioned regions (or tiles). Consequently, generated tiles form a parallelization unit that can be processed independently in parallel. A MapReduce-based spatial query processing system takes advantage of such partitioning to achieve high performance. Algorithm 1.1 illustrates a general design framework for such systems, and all the above-mentioned systems follow this framework while implementation details may vary.

Algorithm 1.1: Typical workflow of spatial query processing on MapReduce

```
1 A. Data/space partitioning
2 B. Data storage of partitioned data on HDFS
3 for tile in input_collection do
4     Indexing building for objects in the tile
5     Tile based spatial querying processing
6 E. Boundary object handling
7 G. Data aggregation
8 H. Result storage on HDFS
```

Initially the dataset is spatially partitioned to generate tiles as shown in step A. In step B, spatial objects are assigned unique tile identifiers (UIDs), merged, and stored into Hadoop Distributed File System (HDFS). Step C is for pre-processing queries, which could be queries that do global index-based filtering. Step D does tile-based spatial query processing independently and parallelized across a large number of cluster nodes. Step E provides handling of boundary objects that arise from the partitioning. Step F is for post-query processing, and step G performs data aggregation. Finally, the query results are persisted to HDFS, which can be input to the next query operator.

Following such framework, spatial queries such as spatial join query, spatial range query, and nearest neighbor query can be implemented efficiently. Reference implementations are provided in HadoopGIS, SpatialHadoop, and Parallel Secondo.

1.4.2 *Effective spatial data partitioning*

Spatial data partitioning is an essential initial step to define, generate, and represent partitioned data. Effective data partitioning is critical for task parallelization, load balancing, and directly affects system performance. Generally a space-oriented partitioning can be applied to generate data partitions, and the concept is illustrated in Figure 1.2 in which the spatial data is partitioned into uniform grids.

However, there are several problems with this approach: (1) As spatial objects (e.g., polygons and polylines) are extent, regular grid-based spatial partitioning would undesirably

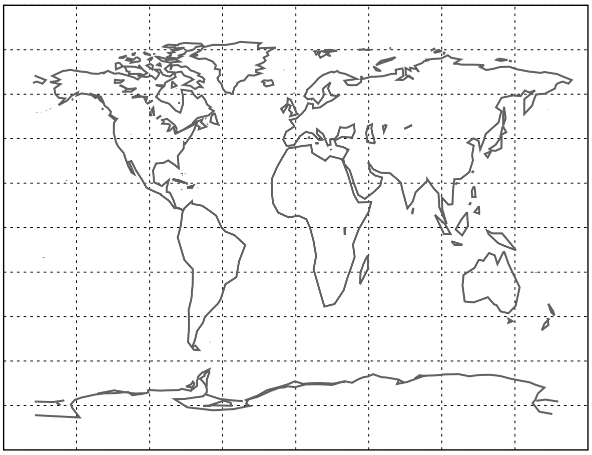


Figure 1.2: An example of spatial data partitioning.

produce objects spanning multiple cell grids, which need to be replicated and post-processed. If such objects account for a considerable fraction of the dataset, the overall query performance would suffer from such boundary handling overhead. (2) Fixed grid partitioning is skew-averse, whereas data in most real-world spatial applications are inherently highly skewed. In such case, it is very likely that parallel processing nodes assigned to process those dense regions will become the *stragglers*, and the overall query processing efficiency will suffer [23].

The boundary problem can be addressed in two different ways—*multi-assignment, single-join* (MASJ) and *single-assignment, multi-join* (SAMJ) [19,32]. In MASJ approach, each boundary object is replicated to each tile that overlaps with the object. During the query processing phase, each partition is processed only once without considering the boundary objects. Then a *de-duplication* step is initiated to remove the redundancies that resulted from the replication. However, in SAMJ approach, each boundary object is only assigned to one tile. Therefore, during the query processing phase, each tile is processed multiple times to account for the boundary objects.

Both approaches introduce extra query processing overhead. In MASJ, the replication of boundary objects incurs extra storage cost and computation cost. In SAMJ, however, only extra computation cost is incurred by processing the same partition multiple times. Hadoop-GIS and SpatialHadoop takes the MASJ approach and modify, query processing steps account for replicated objects. However, depending on the application requirement, such design choice can be re-evaluated and modified to achieve better performance.

The data skew problem can be mitigated through skew-aware partitioning approaches that can create balanced partitions. HadoopGIS uses a multi-step approach named *SATO* which can partition a geospatial dataset into *balanced regions* while *minimizing the number of boundary objects*. SATO represents the four main steps in this framework for spatial data partitioning: **S**ample, **A**nalyze, **T**ear, and **O**ptimize. First, a small fraction of the dataset is sampled to identify overall global data distribution with potential dense regions. The sampled data is analyzed with a *partition analyzer* that produces a coarse partition scheme in which each partition region is expected to contain roughly equal amounts of spatial objects. Later, these coarse partition regions are further processed with a partitioning component that *tears* the regions into more granular partitions that are much less skewed. Finally, generated partition meta-information is aggregated to produce multilevel partition indexes and additional partition statistics that are used to *optimize* spatial queries.

SpatialHadoop also creates balanced partitions in similar manner. Specifically, an appropriate partition size is estimated, and rectangular boundaries are generated according to such partition parameter. Then, a MapReduce job is initiated to create spatial partitions that corresponds to such configuration. An $R+$-Tree-based partitioning is used to ensure the partition size constraint.

1.4.3 Query co-processing with GPU and CPU

Most spatial queries are compute-intensive [7,22], as they involve geometric computations on complex multidimensional spatial objects. While spatial filtering through MBRs can be accelerated through spatial access methods, spatial refinements such as polygon intersection verification are highly expensive operations. For example, spatial join queries such as spatial cross-matching or overlaying multiple sets of spatial objects on an image or map can be very expensive to process.

GPUs have been successfully utilized in numerous applications that require high-performance computation. Mainstream general purpose GPUs come with hundreds of cores, and can run thousands of threads in parallel. Compared to the multi-core computer systems (dozens of cores), GPUs can scale to large number of threads in a cost-effective manner. In the coming years, such heterogeneous parallel architecture will become dominant, and software systems must fully exploit such heterogeneity to deliver performance growth [12].

In most cases, spatial algorithms are designed for executing on the CPUs, and the branch-intensive nature of CPU-based algorithms require the algorithm to be rewritten for GPUs for satisfactory performance. For such need, PixelBox is proposed in [29]. PixelBox is an algorithm specifically designed for accelerating cross-matching queries on the GPUs. It first transforms the vector-based geometry representation into raster representation using a pixelization method, and performs operations on such representations in parallel. The pixelization method reduces the geometry calculation problem into simple pixel position checking problem, and it is very suitable for execution on GPUs. Since testing the position of one pixel is totally independent of another, it can parallelize the computation by having multiple threads process the pixels in parallel. Furthermore, since the position of different pixels are computed against the same pair of polygons, the operations performed by different threads follow the single instruction multiple data (SIMD) fashion, a parallel computation model that GPUs are designed for. Experimental results [29] suggest that PixelBox achieves almost an orders of magnitude speedup compared to the CPU implementation, and significantly reduces the cost of computation.

1.4.3.1 Task assignment

One critical issue for GPU-based parallelization is task assignment. For example, in a recent approach that combines MapReduce and GPU-based query processing [7], tasks arrive in the form of data partitions along with spatial query operation on the data. Given a partition, the query optimizer has to decide which device should be assigned to execute the task. Such decision is not simple, and it depends on how much speedup can be obtained by assigning it to CPU or GPU. If we schedule a small task on GPU, we may not only get very little speedup, the opportunity cost of executing some other high speedup task on GPU can be high.

In such cases, a predictive modeling approach can offer a reasonable solution. Similar to the speculative execution model in Hadoop, a fraction of data (10%, for example) is used for performance profiling and model training. Then, regression or machine learning approaches are used to derive the performance model, and corresponding model parameters. Later during the runtime, the derived model is used to predict the potential speedup factor for current task, and tasks are scheduled to execute on GPU if the speedup factor is higher than certain threshold.

1.4.3.2 Effects of task granularity

Data need to be shipped to the device memory to be executed on the GPU device. Such data transfer incurs certain I/O cost. While the memory bandwidth between GPU and CPU is much higher compared to the bandwidth between memory and the disk, it should be minimized to achieve optimal performance. To achieve optimal speedup, the compute-to-transfer ratio needs to be high for GPU applications. Therefore, applications need to adjust the partition granularity to fully utilize system resources. While larger partitioning is ideal for achieving higher speedup on GPU, it causes data skew which is detrimental for MapReduce system performance. At the same time, a very small partition is not a good candidate for hardware acceleration.

1.5 Discussion and Conclusion

In this chapter, we have discussed several representative spatial Big Data applications, common challenges in this domain, and potential solutions toward those challenges. Spatial Big Data from various application domains share many similar requirements with enterprise Big Data, but has its own unique characteristics—spatial data are multidimensional, spatial queries are complex, and spatial query processing comes with high computational complexity. As the volume of data grow continuously, we need efficient Big Data systems and data management techniques to be able to cope with such challenges. MapReduce-based massively parallel query processing systems offer a scalable, yet cost-effective solution for processing large amounts of spatial data. While relying on such framework, effective data partitioning techniques can be critical for facilitating massive parallelism, and achieving satisfactory query performance. Meanwhile, as multi-core computer architecture and programming techniques become mature, hardware-accelerated query co-processing on GPUs can further improve query performance for large-scale spatial analytics tasks.

Acknowledgments

Fusheng Wang acknowledges that this material is based on work supported in part by NSF CAREER award IIS 1350885, NSF ACI 1443054, and by the National Cancer Institute under grant No. 1U24CA180924-01A1.

References

1. Satellite imagery. https://en.wikipedia.org/wiki/Satellite_imagery.

2. Foursquare Labs, Inc. https://foursquare.com/about.

3. OpenStreetMap: A map of the world, free to use. http://www.openstreetmap.org.

4. GIS tools for Hadoop, Big Data spatial analytics. http://esri.github.io/gis-tools-for-hadoop.

5. York DG, Adelman J, Anderson Jr. JE, Anderson SF, Annis J, Bahcall NA, Bakken JA et al. The sloan digital sky survey: Technical summary. *The Astronomical Journal*, 120(3):1579, 2000.

6. Anastasia Ailamaki, Verena Kantere, and Debabrata Dash. Managing scientific data. *Commun. ACM*, 53(6):68–78, 2010.

7. Ablimit Aji, Teodoro George, and Fusheng Wang. Haggis: Turbocharge a mapreduce based spatial data warehousing system with gpu engine. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, pp. 15–20, Dallas, TX, 2014.

8. Ablimit Aji, Xiling Sun, Hoang Vo, Qioaling Liu, Rubao Lee, Xiaodong Zhang, Joel Saltz et al. Demonstration of hadoop-gis: A spatial data warehousing system over mapreduce. In *SIGSPATIAL/GIS*, pp. 518–521. ACM, New York, 2013.

9. Ablimit Aji, Fusheng Wang, and Joel H. Saltz. Towards building a high performance spatial query system for large scale medical imaging data. In *SIGSPATIAL/GIS*, pp. 309–318. ACM, New York, 2012.

10. Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel Saltz. Hadoop-GIS: A high performance spatial data warehousing system over MapReduce. *Proc. VLDB Endow.*, 6(11):1009–1020, August 2013.

11. Marco Bastos, Raquel Recuero, and Gabriela Zago. Taking tweets to the streets: A spatial analysis of the vinegar protests in Brazil. *First Monday*, 19(3), 2014. http://firstmonday.org/ojs/index.php/fm/article/view/5227/3843.

12. Shekhar Borkar and Andrew A Chien. The future of microprocessors. *Commun. ACM*, 54(5):67–77, 2011.

13. Lee AD Cooper, Alexis B Carter, Alton B Farris, Fusheng Wang, Jun Kong, David A Gutman, Patrick Widener et al. Digital pathology: Data-intensive frontier in medical imaging. *Proc. IEEE*, 100(4):991–1003, 2012.

14. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

15. David DeWitt and Jim Gray. Parallel database systems: The future of high performance database systems. *Commun. ACM*, 35(6):85–98, 1992.

16. Ahmed Eldawy and Mohamed F Mokbel. A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data. *Proc. VLDB Endow.*, 6(12):1230–1233, 2013.

17. Ahmed Eldawy and Mohamed F Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *Proceedings of the IEEE International Conference on Data Engineering*. IEEE, Seol, Korea, 2015.

18. Justin J Levandoski, Mohamed Sarwat, Ahmed Eldawy, and Mohamed F Mokbel. Lars: A location-aware recommender system. In *IEEE 28th International Conference on Data Engineering*, pp. 450–461. IEEE, Arlington, VA, 2012.

19. Ming-Ling Lo and Chinya V Ravishankar. Spatial hash-joins. In *ACM SIGMOD Record*, pp. 247–258. ACM, Montreal, Canada, 1996.

20. Jiamin Lu and Ralf H Guting. Parallel secondo: Practical and efficient mobility data processing in the cloud. In *IEEE International Conference on Big Data*, pp. 107–125. IEEE, Silicon Valley, CA, 2013.

21. Jiamin Lu and Ralf Hartmut Guting. Parallel secondo: A practical system for large-scale processing of moving objects. In *IEEE 30th International Conference on Data Engineering*, pp. 1190–1193. IEEE, Chicago, IL, 2014.

22. Bogdan Simion, Suprio Ray, and Angela D Brown. Surveying the landscape: An in-depth analysis of spatial database workloads. In *SIGSPATIAL*, pp. 376–385. ACM, New York, 2012.

23. Benjamin Sowell, Marcos V Salles, Tuan Cao, Alan Demers, and Johannes Gehrke. An experimental analysis of iterated spatial joins in main memory. *Proc. VLDB Endow.*, 6(14):1882–1893, 2013.

24. Jack Steenstra, Alexander Gantman, Kirk Taylor, and Liren Chen. Location based service (lbs) system and method for targeted advertising, March 23, 2006. US Patent App. 10/931,309.

25. Michael Stonebraker. The case for shared nothing. *IEEE Database Eng. Bull.*, 9(1):4–9, 1986.

26. Ashish Thusoo, Joydeep S Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu et al. Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, 2(2):1626–1629, August 2009.

27. Fusheng Wang, Jun Kong, Lee Cooper, Tony Pan, Tahsin Kurc, Wenjin Chen, Ashish Sharma et al. A data model and database for high-resolution pathology analytical image informatics. *J. Pathol. Inform.*, 2(1):32, 2011.

28. Fusheng Wang, Jun Kong, Jingjing Gao, Lee Cooper, Tahsin M Kurc, Zhengwen Zhou, David Adler et al. A high-performance spatial database based approach for pathology imaging algorithm evaluation. *J. Pathol. Inform.*, 4:5, 2013.

29. Kaibo Wang, Yin Huai, Rubao Lee, Fusheng Wang, Xiaodong Zhang, and Joel H Saltz. Accelerating pathology image data cross-comparison on CPU-GPU hybrid systems. *Proc. VLDB Endow.*, 5(11):1543–1554, 2012.

30. Randall T Whitman, Michael B Park, Sarah M Ambrose, and Erik G Hoel. Spatial indexing and analytics on hadoop. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 73–82. ACM, New York, 2014.

31. Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, p. 10. USENIX Association, Berkeley, CA, 2010.

32. Xiaofang Zhou, David J Abel, and David Truffet. Data partitioning for parallel spatial join processing. *GeoInformatica*, 2(2):175–204, 1998.

Chapter 2

Storage and Database Management for Big Data*

Vijay Gadepally

Jeremy Kepner

Albert Reuther

CONTENTS

*This work is sponsored by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, recommendations and conclusions are those of the authors and are not necessarily endorsed by the United States Government.

2.1 Introduction

The ability to collect and analyze large amounts of data is a growing problem within the scientific community. The growing gap between data and users calls for innovative tools that address the challenges faced by big data volume, velocity, and variety. While there has been great progress in the world of database technologies in the past few years, there are still many fundamental considerations that must be made by scientists. For example, which of the seemingly infinite technologies are the best to use for my problem? Answers to such questions require a careful understanding of the technology field in addition to the types of problems that are being solved. This chapter aims to address many of the pressing questions faced by individuals interested in using storage or database technologies to solve their big data problems.

Storage and database management is a vast field with many decades of results from very talented scientists and researchers. There are numerous books, courses, and articles dedicated to the study. This chapter attempts to highlight some of these developments as they relate to the equally vast field of big data. However, it would be unfair to say that this chapter provides a comprehensive analysis of the field—such a study would require many volumes. It is our hope that this chapter can be used as a launching pad for researchers interested in the study. Where possible, we highlight important studies that can be pursued for further reading.

In Section 2.2, we discuss the big data challenge as it relates to storage and database engines. The chapter goes on to discuss database utility compared to large parallel storage arrays. Then, the chapter discusses the history of database management systems with special emphasis on current and upcoming database technology trends. In order to provide readers with a deeper understanding of these technologies, the chapter will provides a deep dive into two canonical open source database technologies: Apache Accumulo [1], which is based on the popular Google BigTable design, and a NewSQL array database called SciDB [59]. Finally, we will provide insight into technology selection and walk readers through a case study which highlights the use of various database technologies to solve a medical big data problem.

2.2 Big Data Challenge

Working with big data is prone to a variety of challenges. Very often, these challenges are referred to as the three Vs of big data: Volume, Velocity and Variety [45]. Most recently, there has been a new emergent challenge (perhaps a fourth V): Veracity. These combined challenges constitute a large reason why big data is so difficult to work with.

Big data volume stresses the storage, memory, and computational capacity of a computing system and often requires access to a computing cloud. The National Institute of Science and Technology (NIST) defines cloud computing to be “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources ... that can be rapidly provisioned and released with minimal management effort or service provider interaction” [47]. Within this definition, there are different cloud models that satisfy different problem characteristics and choosing the right cloud model is problem specific. Currently, there are four multibillion dollar ecosystems that dominate the cloud-computing landscape: enterprise clouds, big data clouds, Structured Query Language (SQL) database clouds, and supercomputing clouds. Each cloud ecosystem has its own hardware, software, conferences, and business markets. The broad nature of business big data challenges makes it unlikely that one cloud ecosystem can meet its needs, and solutions are likely to require the tools and techniques from more than one cloud ecosystem. For this reason, at the Massachusetts Institute of Technology (MIT) Lincoln Laboratory, we developed the MIT SuperCloud architecture [51] that enables the prototyping of four common computing ecosystems on a shared hardware platform as depicted in Figure 2.1. The velocity of big data stresses the rate at which data can be absorbed and meaningful answers produced. Very often, the velocity challenge is mitigated through high-performance databases, file systems, and/or processing. Big data variety may present the largest challenge and greatest opportunities. The promise of big data is the ability to correlate diverse and heterogeneous data to form new insights. A new fourth V [26], veracity, challenges our ability to perform computation on data while preserving privacy.

As a simple example of the scale of data and how it has changed in the recent past, consider the social media analysis developed by [24]. In 2011, Facebook had approximately 700,000 pieces of content per minute; Twitter had approximately 100,000 tweets per minute; and YouTube had approximately 48 hours of video per minute. By 2015, just 4 years later, Facebook had 2.5 million pieces of content per minute; Twitter had approximately 277,000 tweets per minute; and YouTube had approximately 72 hours of new video per minute. This increase in data generated can be roughly approximated to be 350 MB/min for Facebook, 50 MB/min for Twitter, and 24–48 GB/min for YouTube! In terms of the sheer volume of data, IDC estimates that from the year 2005 to the year 2020, there will an increase in the amount of data generated from 130 EB to 40,000 EB [30].

One of the greatest big data challenges is in determining the ideal storage engine for a large dataset. Databases and file systems provide access to vast amounts of data but differ at a fundamental level. File system storage engines are designed to provide access to a potentially large subset of the full dataset. Database engines are designed to index and provide access to a smaller, but well defined, subset of data. Before looking at particular storage and database

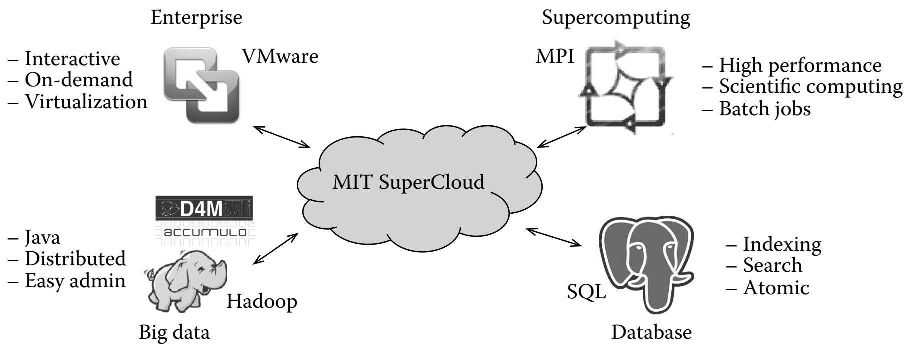


Figure 2.1: The MIT SuperCloud infrastructure allows multiple cloud environments to be launched on the same hardware and software platform in order to address big data volume.

engines, it is important to take a look at where these systems fall within the larger big data system.

2.3 Systems Engineering for Big Data

Systems engineering studies the development of complex systems. Given the many challenges of big data as described in Section 2.2, systems engineering has a great deal of applicability to developing a big data system. One convenient way to visualize a big data system is as a pipeline. In fact, most big data systems consist of different steps which are connected to each other to form a pipeline (sometimes, they may not be explicitly separated though that is the function they are performing). Figure 2.2 shows a notional pipeline for big data processing.

First, raw data is often collected from sensors or other such sources. These raw files often come in a variety of formats such as comma-separated values (CSVs), JavaScript Object Notation (JSON) [21], or other proprietary sensor formats. Most often, this raw data is collected by the system and placed into files that replicate the formatting of the original sensor. Retrieval of raw data may be done by different interfaces such as cURL (http://curl.haxx.se/) or other messaging paradigms such as publish/subscribe. The aforementioned formats and retrieval interfaces are by no means exhaustive but highlight some of the popular tools being used.

Once the raw data is on the target system, the next step in the pipeline is to parse these files into a more readable format or to remove components that are not required for the end-analytic. Often, this step involves removing remnants of the original data collection step such as unique identifiers that are no longer needed for further processing. The parsed files are often kept on a serial or parallel file system and can be used directly for analytics by scanning files. For example, a simple word count analytic can be done by using the Linux grep command on the parsed files, or more complex analytics can be performed by using a parallel processing framework such as Hadoop MapReduce or the Message Passing Interface (MPI). As an example of an analytic which works best directly with the file system, dimensional analysis [27] performs aggregate statistics on the full dataset and is much more efficient working directly from a high-performance parallel file system.

For other analytics (especially those that wish to access only a small portion of the entire dataset), it is convenient to ingest this data into a suitable database. An example of such an analytic is given in [28], which performs an analysis on the popularity of particular entities in a database. This example takes only a small, random piece of the dataset (the counts of words is much smaller than the full dataset) and is well suited for database usage. Once data is in the database or on the file system, a user can write queries or scans depending on their use case to produce results that can then be used for complex analytics such as topic modeling.

Each step of the pipeline involves a variety of choices and decisions. These choices may depend on hardware, software, or other factors. Many of these choices will also make a

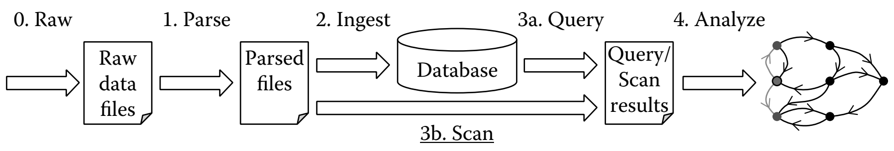


Figure 2.2: A standard big data pipeline consists of five steps to go from raw data to useful analytics.

difference to the later parts of the pipeline and it is important to make informed decisions. Some of the choices that one may have at each step include the following:

- *Step 0*: Size of individual raw data files, output format
- *Step 1*: Parsed data contents, data representation, parser design
- *Step 2*: Size of database, number of parallel processors, pre-processing
- *Step 3*: Scan or query for data, use of parallel processing
- *Step 4*: Visualization tools, algorithms

For the remainder of this chapter, we will focus on some of the decisions in steps two and three of the pipeline. By the end of the chapter, we hope that readers will have an understanding of different storage and database engines, the right time to use technology, and how these pieces can come together.

2.4 Disks and File Systems

One of the most common ways to store a large quantity of data is through the use of traditional storage media such as hard drives. There are many storage options that must be carefully considered that depend upon various parameters such as total data volume and desired read and write rates. In the pipeline of Figure 2.2, the storage engine plays an important part of steps two and three.

In order to deal with many challenges such as preserving data through failures, the past decades have seen the development of many technologies such as RAID (redundant array of independent disks) [17], NFS (network file system), HDFS (Hadoop Distributed File System) [11], and Lustre [67]. These technologies aim to abstract the physical hardware away from application developers in order to provide an interface for an operating system to keep track of a large number of files while allowing support for data failure, high-speed seeks, and fast writes. In this section, we will focus on two leading technologies, Lustre and HDFS.

2.4.1 Serial memory and storage

The most prevalent form of data storage is provided by an individual's laptop or desktop system. Within these systems, there are different levels of memory and storage that trade off speed with cost calculated as bytes per dollar. The fastest memory provided by a system (apart from the relatively low capacity system cache) is the main memory or random access memory (RAM). This volatile memory provides relatively high speed (10s of GB/s in 2015) and is often used to store data up to hundreds of gigabytes in 2015. When the data size is larger than the main memory, other forms of storage are used. Within serial storage technologies, some of the most common are traditional spinning magnetic disc hard drives and solid-state drives (solid-state drives may be designed to use volatile RAM or nonvolatile flash technology). The capacity of these technologies can be in the 10s of TB each and can support transfer rates anywhere from approximately 100 MB/s to GB/s in 2015.

2.4.2 Parallel storage: Lustre

Lustre is designed to meet the highest bandwidth file requirements on the largest systems in the world [12] and is used for a variety of scientific workloads [49]. The open source Lustre parallel file system presents itself as a standard POSIX, general-purpose file system and is mounted by client computers running the Lustre client software. Files stored in Lustre contain two components—metadata and object data. Metadata consists of the fields associated with each file such as i-node, filename, file permissions, and timestamps. Object data consists of the binary data stored in the file. File metadata is stored in the Lustre metadata server (MDS). Object data is stored in object storage servers (OSSes) shown in Figure 2.3. When a client requests data from a file, it first contacts the MDS, which returns pointers to the appropriate objects in the OSSes. This movement of information is transparent to the user and handled fully by the Lustre client. To an application, Lustre operations appear as standard file system operations and require no modification of application code.

A typical Lustre installation might have many OSSes. In turn, each OSS can have a large number of drives that are often formatted in a RAID configuration (often RAID6) to allow for the failure of any two drives in an OSS. The many drives in an OSS allows data to be read in parallel at high bandwidth. File objects are striped across multiple OSSes to further increase parallel performance. The above redundancy is designed to give Lustre high availability while avoiding a single point of failure. Data loss can only occur if three drives fail in the same OSS prior to any one of the failures being corrected. For Lustre, the typical storage penalty to provide this redundancy is approximately 35%. Thus, a system with 6 PB of raw storage will provide 4 PB of data capacity to its users.

Lustre is designed to deliver high read and write performance to many simultaneous large files. Lustre systems offer very high bandwidth access to data. For a typical Lustre configuration, this bandwidth may be approximately 12 GB/s in 2015 [2]. This is achieved by the clients having a direct connection to the OSSes via a well-designed high-speed network. This connection is brokered by the MDS. The peak bandwidth of Lustre is determined by the aggregate network bandwidth to the client systems, the bisection bandwidth of the network switch, the aggregate network connection to the OSSes, and the aggregate bandwidth of all

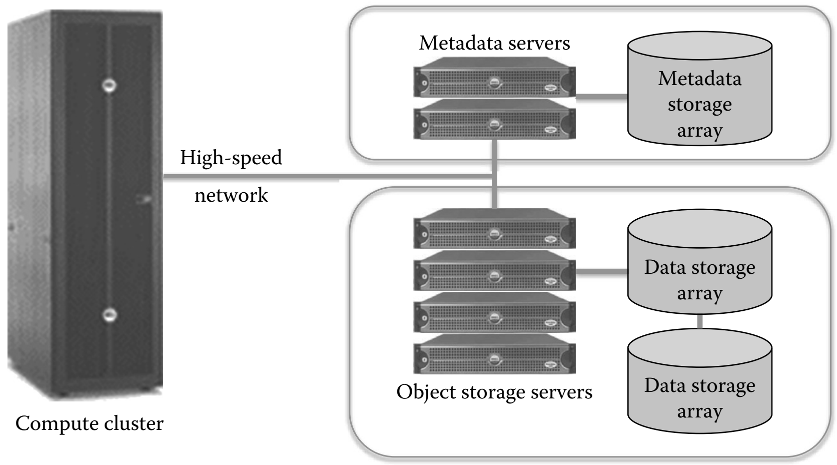


Figure 2.3: A Lustre installation consists of metadata servers and object storage servers. These are connected to a compute cluster via a high-speed interconnect such as at 10 GB Ethernet or Infiniband.

the disks [42]. Like most file systems, Lustre is designed for sequential read access and not random lookups of data (unlike a database). To find a particular data value in Lustre requires, on average, scanning through half the file system. For a typical system with approximately 12 GB/s of maximum bandwidth and 4 PB of user storage, this may require approximately 4 days.

2.4.3 Parallel storage: HDFS

Hadoop is a fault-tolerant, distributed file system and distributed computation system. An important component of the Hadoop ecosystem is the supporting file system called the HDFS that enables MapReduce [22] style jobs. HDFS is modeled after the Google File System (GFS) [33] and is a scalable distributed file system for large, distributed, and data-intensive applications. GFS and HDFS provide fault tolerance while running on inexpensive off-the-shelf hardware, and deliver high aggregate performance to a large number of clients. The Hadoop distributed computation system uses the MapReduce parallel programming model for distributing computation onto the data nodes.

The foundational assumptions of HDFS are that its hardware and applications have the following properties [11]: high rates of hardware failures, special purpose applications, large datasets, write-once-read-many data, and read-dominated applications. HDFS is designed for an important, but highly specialized class of applications for a specific class of hardware. In HDFS, applications primarily employ a co-design model whereby the HDFS is accessed via specific calls associated with the Hadoop API.

A file stored in HDFS is broken into two pieces: metadata and data blocks as shown in Figure 2.4. Similar to the Lustre file system, metadata consists of fields such as the filename, creation date, and the number of replicas of a particular piece of data. Data blocks consist of the binary data stored in the file. File metadata is stored in an HDFS name node. Block data is stored on data nodes. HDFS is designed to store very large files that will be broken up into multiple data blocks. In addition, HDFS is designed to support fault-tolerance in massive distributed data centers. Each block has a specified number of replicas that are distributed across different data nodes. The most common HDFS replication policy is to store three copies of each data block in a location-aware manner so that one replica is on a node in the local rack, the second replica on a node in a different rack, and the third replica on another node in the same different rack [3]. With such a policy, the data will be protected from node and rack failure.

The storage penalty for a triple replication policy is 66%. Thus, a system with 6 PB of raw storage will provide 2 PB of data capacity to its users with triple replication. Data loss can only occur if three drives fail prior to any one of the failures being corrected. Hadoop is written in Java and is installed in a special Hadoop user account that runs various Hadoop daemon processes to provide services to connecting clients. Hadoop applications contain special

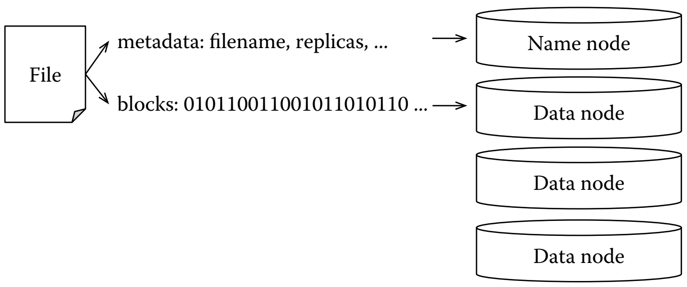

File
metadata: filename, replicas, ...
blocks: 010110011001011010110 ...
Name node
Data node
Data node
Data node

Figure 2.4: Hadoop splits a file into metadata and replicates it in data blocks.

application program interface (API) calls to access the HDFS services. A typical Hadoop application using the MapReduce programming model will distribute an application over the file system so that each application is exclusively reading blocks that are local to the node on which it is running. A well-written Hadoop application can achieve very high performance if the blocks of the files are well distributed across the data nodes. Hadoop applications use the same hardware for storage and computation. The bandwidth achieved out of HDFS is highly dependent upon the computation to communication ratio of the Hadoop application. For a well-designed Hadoop application, this aggregate bandwidth may be as high as 100 GB/s for a typical HDFS setup. Like most other file systems, HDFS is designed for sequential data access and no random access of data.

2.5 Database Management Systems

Relational or SQL databases [20,62] have been the de facto interface to databases since the 1980s and are the bedrock of electronic transactions around the world. For example, most financial transactions in the world make use of technologies such as Oracle or dBase. With the great rise in quantity of unstructured data and analytics based on the statistical properties of datasets, NoSQL (Not Only SQL) database stores such as the Google BigTable [19] have been developed. These databases are capable of processing the large heterogeneous data collected from the Internet and other sensor platforms. One style of NoSQL databases that have become used for applications that require support for high velocity data ingest and relatively simple cell-level queries are key-value stores.

As a result, the majority of the volume of data on the Internet is now analyzed using key-value stores such as Amazon Dynamo [23], Cassandra [44], and HBase [32]. Key-value stores and other NoSQL databases compromise on data consistency in order to provide higher performance. In response to this challenge, the relational database community has developed a new class of relational databases (often referred to as NewSQL) such as SciDB [16], H-Store [37], and VoltDB [64] to provide the features of relational databases while also scaling to very large datasets. Very often, these newSQL databases make use of a different datamodel [16] or advances in hardware architectures. For example, MemSQL [56] is a distributed in-memory database that provides high-performance, atomicity, consistency, isolation, and durability (ACID)-compliant relational database management. Another example, BlueDBM [36], provides high-performance data access through flash storage and field programmable gate arrays (FPGA).

In this section, we provide an overview of database management systems, the different generations of databases, and a deep dive into two newer technologies: a key-value store—Apache Accumulo and an array database—SciDB.

2.5.1 Database management systems and features

A database is a collection of data and all of the supporting data structures. The software interface between users and a database is known as the database management system. Database management systems provide the most visible view into a dataset. There are many popular database management systems such as MySQL [4], PostgreSQL [63], and Oracle [5]. Most commonly, users interact with database management systems for a variety of reasons, which are listed as follows:

1. To define data, schema, and ontologies
2. To update/modify data in the database

3. To retrieve or query data
4. To perform database administration or modify parameters such as security settings
5. More recently, to perform analytics on the data within the database

Databases are used to support data collection, indexing, and retrieval through transactions. A database transaction refers to the collection of steps involved in performing a single task [31]. For example, a single financial transaction such as *credit $100 towards the account of John Doe* may involve a series of steps such as locating the account information for John Doe, determining the current account value, adding $100 to the account, and ensuring that this new value is seen by any other transaction in the future. Different databases provide different guarantees on what happens during a transaction.

Relational databases provide ACID guarantees. Atomicity provides the guarantee that database transactions either occur fully or completely fail. This property is useful to ensure that parts of a transaction do not occur successfully if other parts fail, which may lead to an unknown state. The second guarantee, consistency, is important to ensure that all parts of the database see the same data. This guarantee is important to ensure that when different clients perform transactions and query the database, they see the same results. For example, in a financial transaction, a bank account may be debited before further transactions can occur. Without consistency, parts of the database may see different amounts of money (not a great database property!). Isolation in a database refers to a mechanism of concurrency control in a database. In many databases, there may be numerous transactions occurring at the same time. Isolation ensures that these transactions are isolated from other concurrent transactions. Finally, database durability is the property that when a transaction has completed, it is persisted even if the database has a system failure. Nonrelational databases such as NoSQL databases often provide a relaxed version of ACID guarantees referred to as BASE guarantees in order to support a distributed architecture or performance. This stands for Basically Available, Soft State, Eventual Consistency guarantees [50]. As opposed to the ACID guarantees of relational databases, nonrelational databases do not provide strict guarantees on the consistency of each transaction but instead provide a looser guarantee that *eventually* one will have consistency in the database. For many applications, this may be an acceptable guarantee.

For these reasons, financial transactions employ relational databases that have the strong ACID guarantees on transactions. More recent trends that make use of the vast quantity of data retrieval from the Internet can be done via nonrelational databases such as Google BigTable [19], which are responsible for fast access to information. For instance, calculating statistics on large datasets are not as susceptible to small eventual changes to the data.

While many aspects of learning how to use a database can be taught through books or guides such as this, there is an artistic aspect to their usage as well. More practice and experience with databases will help overcome common issues, improved performance tuning, and help with improved database management system stability. Prior to using a database, it is important to understand the choices available, properties of the data, and key requirements.

2.5.2 History of open source databases and parallel processing

Databases and parallel processing have developed together over the past few decades. Parallel processing is the ability to take a given program and split it across multiple processors in order to reduce computation time or resource availability for the application. Very often, advances in parallel processing are directly used for the computational piece of databases such as sorting and indexing datasets.

Open source databases have been around since the mid-1990s. Some of the first relational databases were based on the design of the Ingres database [62] originally developed at UC Berkeley. During the same time period, there were many parallel processing or high-performance computing paradigms [38,53] that were being developed by industry and academia. The first few (popular) open source databases that were created were PostgreSQL and MySQL. The earliest forms of parallel cluster processing take their root in the early 1990s with the wide proliferation of *nix-based operating systems and parallel processing schedulers such as Grid Engine. For about 10 years, until the mid-2000s, these technologies continued to mature, and developers saw the need for greater adoption of distributed computing and databases.

Based on a series of papers from Google in the mid-2000s, the MapReduce computing paradigm was created which gained wide acceptance through the open source Apache Hadoop soon after. These technologies, combined with the seminal Google BigTable [19] paper helped spark the NoSQL movement in databases. Not long after this, numerous technologies such as GraphLab [46], Neo4j [68], and Giraph [9] were developed to apply parallel processing to large unstructured graphs such as those being collected and stored in NewSQL databases. Since the year 2010, there has been renewed interest in developing technologies that offer high performance along with some of the ACID guarantees of relational databases (which will be discussed in Section 2.5.3). This requirement has driven the development of a new generation of relational databases often called NewSQL. In the parallel processing world, users are looking for better ways to deal with streaming data or machine learning and graph algorithms than the Hadoop framework offered and are developing new technologies such as Apache Storm [65] and Spark [69]. Of course, the worlds of parallel processing and databases will continue to evolve, and it will be interesting to see what lies ahead! A brief informational graphic of the history of parallel cluster processing and databases is provided in Figure 2.5.

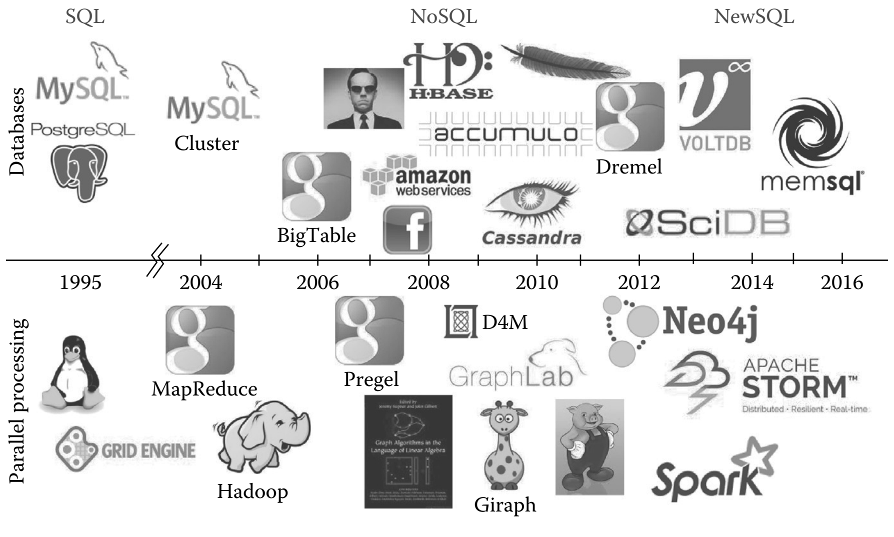


Figure 2.5: An incomplete history of open source databases and parallel cluster computing technologies.

2.5.3 CAP theorem

The CAP theorem is a seminal theorem [13] used to specify what guarantees can be provided by a distributed database. The CAP theorem states that no distributed database can simultaneously provide strong guarantees on the consistency, availability, and partition tolerance of a database. This is often stated as the two-out-of-three rule, though in reality it is more of a loose guarantee rather than losing the guarantee completely. In practice partition tolerance is an important aspect of NoSQL distributed databases; the two-out-of-three rule of the CAP theorem implies that most such databases fall into a consistency-partition tolerance or availability-partition tolerance style. Traditional relational databases are examples of choosing consistency and availability as the two-out-of-three CAP theorem guarantees.

Unlike the definition of consistency in ACID (which refers to a single node view of data in a database), consistency in the CAP theorem refers to the property that all nodes in a distributed database see the same data and provide a client with the same results regardless of which server is responding to a client request. Very often, a strong consistency guarantee is enforced by placing locks on a table, column, row or cell until all parts of the transaction are performed. While this property is very useful to ensure that all queries subsequent to the completion of the transaction see the same value, locking can hinder performance and availability guarantees. For example, in the case of a partition, enforcing consistency implies that certain nodes will not be able to respond to requests until all nodes have a consistent view of the data; thus compromising the availability of these nodes. A NoSQL database that prioritizes consistency over availability is Google BigTable [19] (though it may still have relaxed consistency between data replicas spread across database instances).

Database availability is a property in a distributed database which implies that every transaction must receive a response about whether the transaction has succeeded or failed. Databases that provide strong availability guarantees typically prioritize nodes responding to requests over maintaining a consistent system-wide view of the data. This availability, however, often comes at the cost of consistency. For example, in the event of a database partition, in order to maintain availability, certain parts of a distributed database may provide different results until a synchronization occurs. An example of a NoSQL database that provides a strong availability guarantee is Amazon Dynamo [23], which provides a consistency model often called *eventual consistency* [66].

Consider the example transaction given in Figure 2.6. In this example, the transaction is to update the count of the word *Apple* in *Doc*1 to be 5 from an application that computes a word count in documents. In a relational database, this transaction can occur within a single transaction that locks the row (*Doc*1) and performs the update before relinquishing the lock. In a highly available distributed database that supports eventual consistency, this update may be performed in parallel and eventually combined to show the correct count of the word Apple in *Doc*1 to be 5. For a short period of time, until this consistency is achieved, different nodes may provide a different response when queried about the count of the word Apple in *Doc*1.

The final aspect of the CAP theorem is database partition tolerance. Database partition tolerance is a database property that allows a database to function even after system failures. This property is often a fundamental requirement for large-scale distributed databases. In the event of failure of a piece of a distributed database, a well-designed database will handle the failure and move pieces of data to working components. This property is usually guaranteed by most of NoSQL databases. Traditional relational databases do not rely on distributed networks which are prone to disruption, thus avoiding the need for partition tolerance.

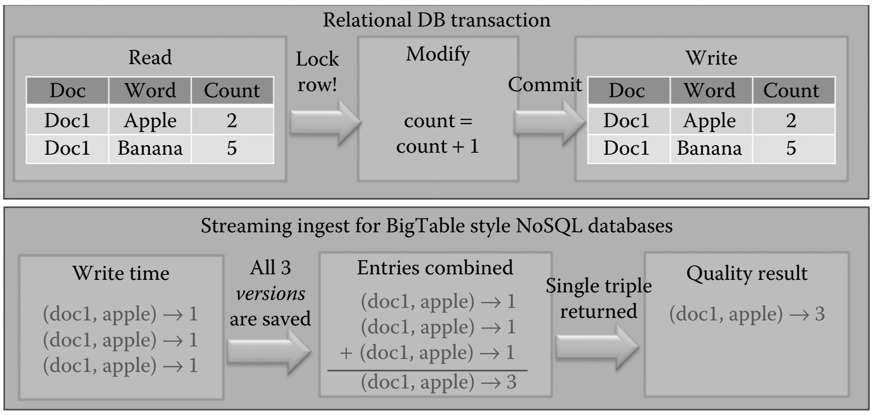


Figure 2.6: Relational update transaction compared with nonrelational database update transaction.

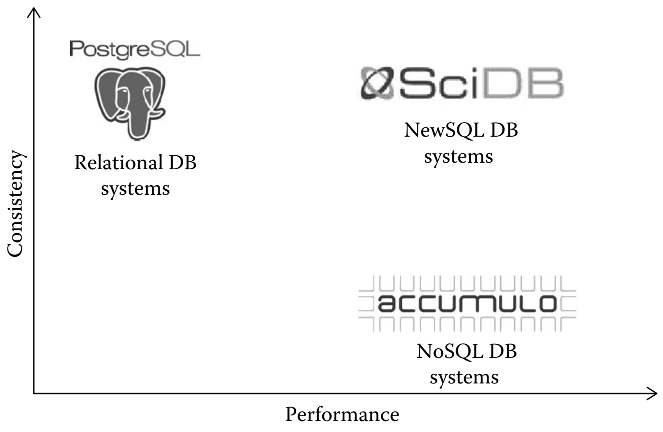


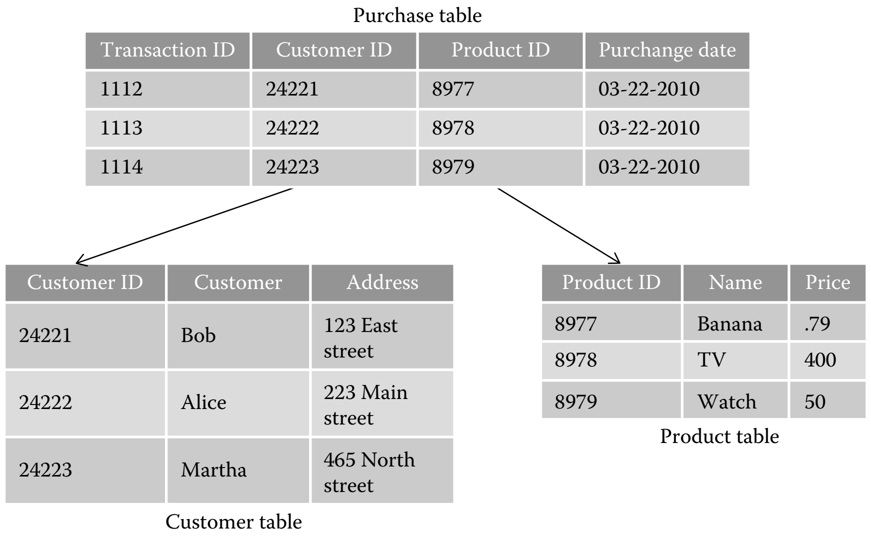
Figure 2.7: Notional guide to implication of the CAP theorem for database design. Traditional relational databases provide high consistency, NoSQL databases provide high performance at the cost of consistency, and NewSQL databases attempt to bridge the gap.

In recent years, there has been some controversy [14,34,57] surrounding the use of the CAP theorem as a fundamental rule in the design of modern databases. Most often, the CAP theorem is used to imply that one can have an all or nothing of two of the three aspects. However, it has been shown in [15] that careful partition and availability optimization may be able to achieve a database that provides a version of all three guarantees. While the CAP theorem can be used for high-level understanding of tradeoffs and design of current technologies, it is certainly possible to design databases that provide versions of guarantees on all three properties through different data models or hardware. In Figure 2.7, we provide a notional guide to the CAP theorem and database classes and also show an example technology for each of these database classes.

2.5.4 Relational databases

Relational databases such as MySQL, PostgreSQL, and Oracle form the bedrock of database technologies today. They are by far the most widely used and accessed databases. We interact with these databases daily: everywhere financial transactions, medical records, and purchases are made. From the CAP theorem, relational databases provide strong consistency and availability; however, they do not support partition tolerance. In order to avoid issues with partition tolerance in distributed databases, relational databases are often vertically scalable. Vertical scalability refers to systems that scale by improving existing software or hardware. For example, vertically scaling a relational database involved improving the resources of a single node (more memory, faster processor, faster disk drive, etc.). Thus, relational databases often run on high-end, expensive nodes and are often limited by the resources of a single node. This is in contrast to nonrelational database that are designed to support horizontal scalability. Scaling a database horizontally involves adding more nodes to the system. Most often, these nodes can be inexpensive commercial off-the-shelf systems (COTS) that are easy to add as resource requirements change.

Relational databases provide ACID guarantees and are used extensively in practice. Relational databases are called *relational* because of the underlying data model. A relational database is a collection of tables that are connected to each other via relations expressed as keys. The specification of tables and relations in a database is referred to as the schema. Schema design requires thorough knowledge of the dataset. Consider a very simple example of a relational database that maintains a record of purchases made by customers as depicted in Figure 2.8. The main purchase table can be used to track purchases. This table is related to a customer table via the customer ID key. The purchase table is also connected to a product table via the product ID key. Using a combination of these tables, one can query the database for information such as *who purchased a banana on March 22, 2010?*. Many databases may contain tens to hundreds of tables and require careful thought during the design.



Purchase table

Transaction ID	Customer ID	Product ID	Purchange date
1112	24221	8977	03-22-2010
1113	24222	8978	03-22-2010
1114	24223	8979	03-22-2010

Customer ID	Customer	Address
24221	Bob	123 East street
24222	Alice	223 Main street
24223	Martha	465 North street

Customer table

Product ID	Name	Price
8977	Banana	.79
8978	TV	400
8979	Watch	50

Product table

Figure 2.8: A simple relational database that contains information about purchases made. The database consists of three tables: a purchase table, a customer table, and a product table.

2.5.5 NoSQL databases

Since the mid-2000s and the Google BigTable paper, there has been a rise in popularity of NoSQL databases. NoSQL databases support many of the large-scale computing activities with which we interact regularly such as web searches, document indexing, large-scale machine learning, and graph algorithms. NoSQL databases support horizontal scaling: you can increase the performance through the addition of nodes. This allows for scaling through the addition of inexpensive COTS as opposed to expensive hardware upgrades required for vertical scaling. NoSQL databases often need to relax some of the consistency or availability guarantees of relational databases in order to take advantage of strong partition tolerance guarantees. In order to keep up with rising data volumes, organizations such as Google looked for ways to incorporate inexpensive off-the-shelf systems for scaling their hardware. However, incorporating such systems requires the use of networks which can be unreliable. Thus, partition tolerance to network disruptions became an important design criteria. In keeping with the CAP theorem, either consistency or availability must be relaxed to provide partition tolerance in a distributed database.

At a transaction level, NoSQL databases provide BASE guarantees. These guarantees may not be suitable for many applications where strong consistency or availability is required. However, for a variety of *big data* applications, BASE guarantees are sufficient for the purpose. For example, recall the example transaction described in Figure 2.6. If the end analytic (step 5 in the big data pipeline) is an approximate algorithm to look for trends in word count, the exact count of the word Apple in *Doc*1 may not be as important as the fact that the word Apple exists in the document. In this case, BASE guarantees may be sufficient for the application. Of course, before choosing a technology to use for an application, it is important to be aware of all design constraints and the impact of technology choice on the final analytic requirements.

NoSQL database use a variety of data models and typically do not have a pre-defined schema. This allows developers the flexibility to specify a schema that leverages the database capabilities while supporting the desired analytics. Further, the lack of a well-defined schema allows dynamic schemas that can be modified as data properties change. Certain graph databases [8] use data structures based on graphs. In such databases, an implicit schema is often generated based on the graph representation of the data in the database. Key-value databases [35] take a given dataset and organize them as a list of keys and values. Document stores such as those described in [6] may use a schema based on a JSON or XML representation of a dataset. Figure 2.9 shows an example of a JSON-based schema applied to the dataset shown in Figure 2.8.

2.5.6 New relational databases

The most recent trend in database design is often referred to as NewSQL databases. Given the controversy surrounding the CAP theorem, such databases attempt to provide a version of all three distributed database properties. These databases were created to approach the performance of NoSQL databases while providing the ACID transaction guarantees of traditional relational databases [58]. In order to provide this combination, NewSQL databases often employ different hardware or data models than traditional database management systems. NewSQL databases may also make use of careful optimizations on partitioning and availability in order to provide a version of all three aspects of the CAP theorem. NewSQL databases may be considered as an alternative to both SQL and NoSQL style databases [43]. Most NewSQL databases provide support for the SQL.

NewSQL databases, while showing great promise, are a relatively new technology area. In the market now are databases designed for sensor processing [25], high-speed online transaction processing (OLTP) [56], and streaming data and analytics [18].