CHAPTER 2

2. BACKGROUND AND RELATED WORK

In this chapter, we present the necessary background of various platforms developed and proposed in the literature for processing spatial data. Next, we survey recently proposed distributed platforms that process spatial data, and we discuss the various challenges that they address.

In Section 2.1, we present the details and the types of queries used by GIS systems applications. Additionally, we describe map layers and various spatial data properties. In Section 2.2, we discuss the various spatial platforms that have been proposed in the literature such as parallel Spatial Database management systems. In Section 2.3, we introduce Hadoop platform architecture and present HDFS read/write procedures. In Section 2.4, we present the challenges that faced Hadoop-based spatial systems, and we introduce several Hadoop-based spatial approaches besides listing pros and cons of each of them. In Section 2.5, we discuss colocation data technique in HDFS and introduce Co-Hadoop [2]. In Section 2.6, we introduce this thesis' proposed work. Finally, we conclude this chapter in Section 2.7.

2.1.BACKGROUND

2.1.1. GIS Systems

A geographic information system (GIS) is a computerized data management system that allows collection, storage, querying, analysis, management, editing, and visualizing all types of geographical spatial data. GIS organizations are from all sizes and in almost every industry. It is widely used in many fields such as web mapping services.

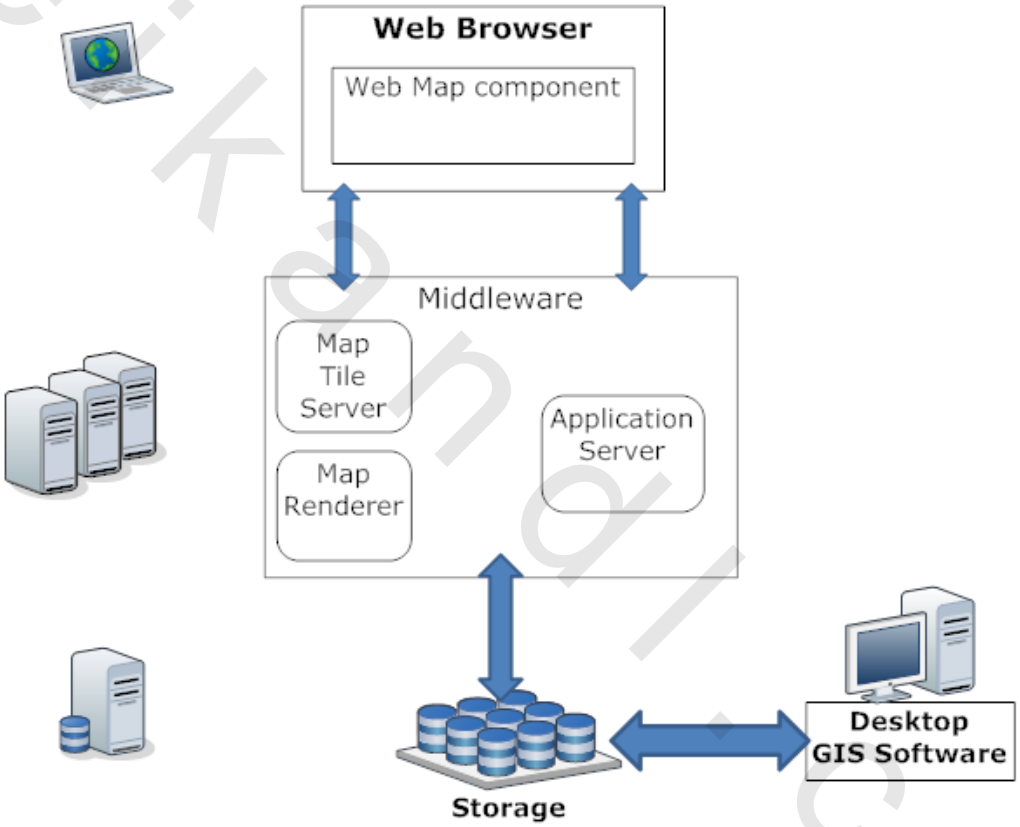

Figure 2.1: GIS system

Figure 2.1 shows the main components of a GIS system. The Desktop GIS software captures and stores data in the Storage layer that can manage and manipulate spatial data and knows about its spatial properties. The middleware layer of the GIS system consists of the following components: a Map tile server, a Map renderer, and application server. The Map renderer component requests Map data from storage layer then renders it using Map renderer component to Map tiles and save them in the Map tile

server component. The browser (web application) calls the application server component to prepare the required Map tiles with the help of other components of middleware layer. Any data processing using Hadoop for example is done on the middleware layer.

2.1.2. Maps Data properties

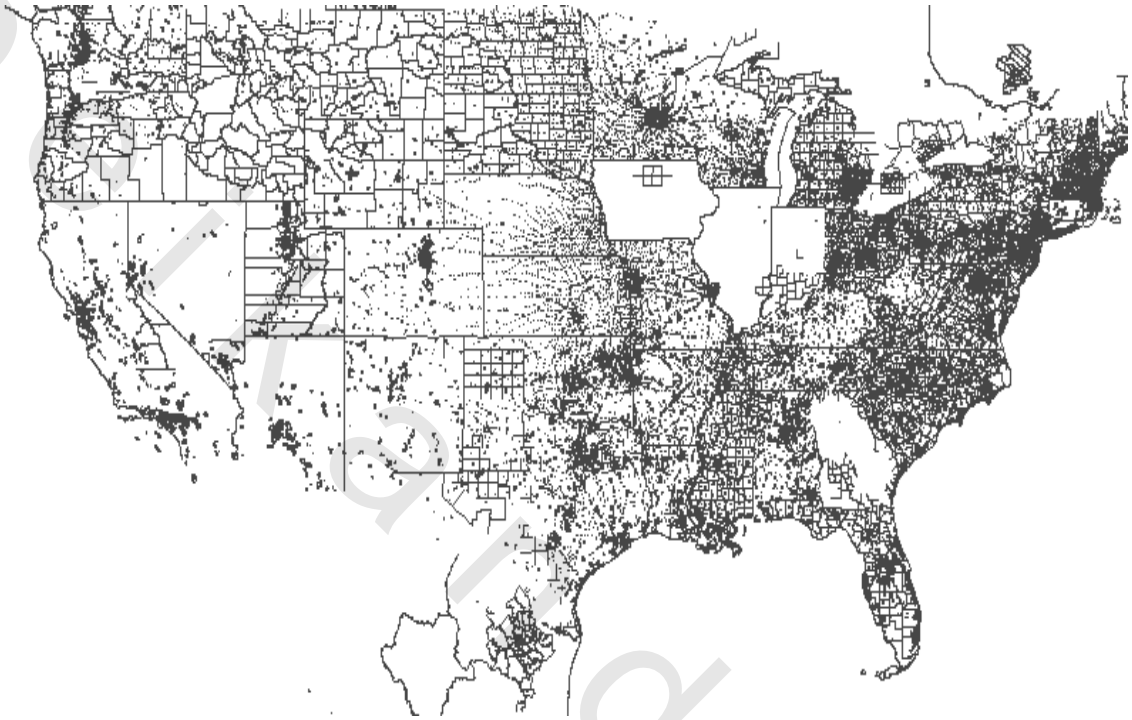

Figure 2.2: OSM cities layer for USA

Maps are broken down into several layers, such as background, parks, cities, roads, and roads names. For example, the road layer draws some lines on the map that are corresponding to the real world roads.

A map layer consists of spatial data objects (Features). Figure 2.3 shows that each feature is defined by several spatial and non-spatial attributes. Spatial attributes are represented as vectors of points, lines or polygons. For example, in Figure 2.2, the cities layer consists of a large number of polygons, where each polygon is defined by several points that represent a city.

The other types of attributes used to describe layers in map files are non-spatial attributes. They are represented using basic data structures such as strings and integers. Non-

spatial attributes are used to keep information about each object in the map, and are used as labels on the map or for generating statistics reports. For example, for each polygon in the map that represents a city, the ID of the city, its name, its address and any other additional information about the city are stored with the city object. These attributes are used either as labels on the map or in another statistics reports. Moreover, non-spatial attributes can be used as a time-stamp stored for each object to keep track of the position of this object at different times, for instance, to be used in location-based applications.

Maps represent the assembly of objects in GIS and location-based applications while microscope slides in the counterpart of maps in pathology applications. Microscope slides contain millions of objects that represent cells and nuclei identified by their boundaries.

Several map providers such as Google maps, OSM (Open Street Map), and Tigers corroboratively prepare maps and allow users to download or access them using several APIs. Figure 2.2 shows the cities layer of USA downloaded from OSM. Each map provider has its private map coordinate system, and all features points of its map layers must be defined by this coordinate system.

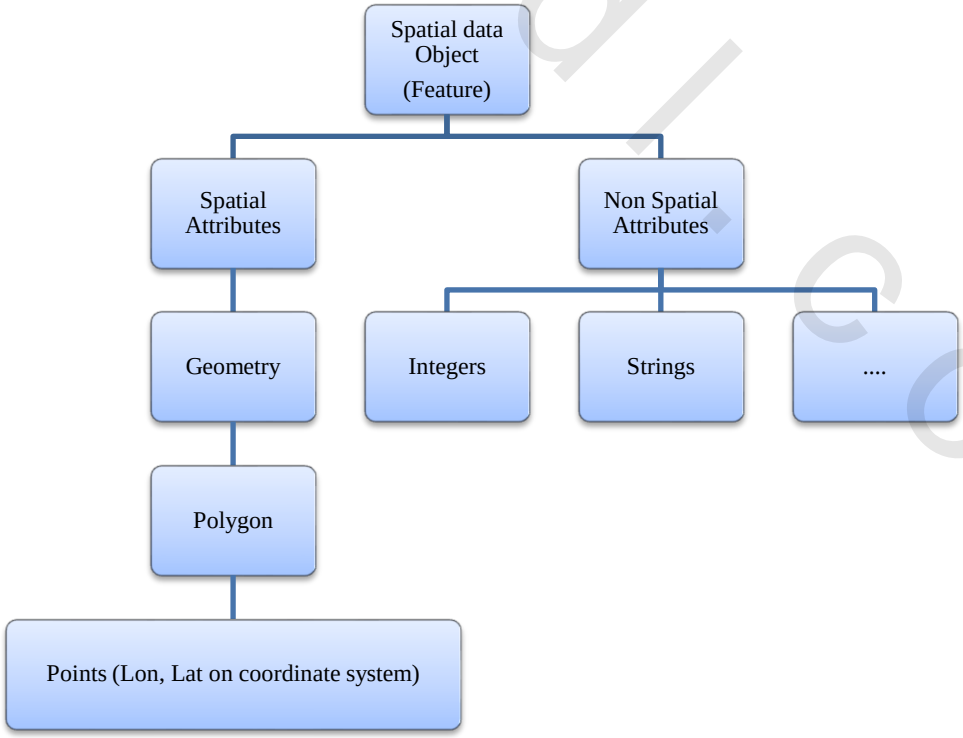

Figure 2.3: Spatial data model

2.2. SPATIAL DATA SUPPORT IN DATABASE SYSTEMS

As discussed in the previous section, spatial data objects consist of many details. In addition, spatial platforms handle various types of queries such as, KNN, Range, and Spatial Join queries. These queries are complex and computationally intensive and they access huge streams of complex spatial data. The main challenges that need to be addressed by platforms that process spatial data are: (1) scalability and (2) query execution in near real-time. We discuss next how spatial data and query support have been integrated into various database systems.

2.2.1. Spatial Database Management Systems (SDBMS)

Traditional spatial database management systems (SDBMSs) are created by adding spatial capabilities to ORDBMS to manage and query spatial data. These spatial capabilities are summarized as follows: (1) build query engine that can parse and execute spatial queries, (2) declare spatial data types such as points and polygons, and (3) spatial indexes such as R-Tree and R+Tree to enhance spatial data access.

To achieve more scalability, parallel SDBMSs have been introduced to reduce the I/O bottleneck by partitioning spatial data on multiple parallel disks such as [4]. However, parallel SDBMS have other various limitations: (1) they do not have effective mechanisms to partition and balance data across partitions, and (2) they have another bottleneck issue with high data loading and insertion [5]. A research team from Emory University has introduced PAIS system [6], [7] to scale out spatial queries through parallel SDBMS. Moreover, PAIS requires software licenses, dedicated hardware, and a lot of maintenance and sophisticated tuning efforts.

Currently, there are many SDBMS used by GIS providers such as: (1) PostgreSQL which uses spatial extension PostGIS [8], and (2) Oracle spatial [9].

2.2.2. Cloud Based Frameworks

With the large scale of spatial computations performed by GIS providers, Cloud based framework is suitable to overcome the significant variation of resources needed by computations execution. However, cloud platform lacks parallel processing that is

needed by large-scale vector data computations, so several approaches are introduced such as Crayons [10] aiming to build end-to-end parallel/distributed processing for spatial data over cloud. These systems build parallel environment that parallelizes various workflow components including file I/O, partitioning, task creation, and load balancing. However, these parallel environments have many limitations: (1) it is difficult to code parallel algorithms that work efficiently, (2) Spatial data is frequently updated, and then the parallel program must be revised, and (3) if one process is down all other processes will be down too.

MapReduce [11] is a suitable choice to overcome the limitations of parallel platforms; it is scalable, and it focuses on fault tolerance. Hadoop [12] is an open source implementation of MapReduce. Support for spatial data and query can smoothly be integrated into the Hadoop ecosystem. Besides, Hadoop has the advantage of scalable processing and abstracting the parallelism of query processing saving the developers the overhead of distributing the query execution themselves on nodes of a cluster. However, using Hadoop to process spatial data has many challenges, which are: (1) being unaware of spatial data properties, types and models, (2) query processors in the Hadoop stack do not understand spatial operations in queries, and (3) Hbase Key-value store used by Hadoop cannot efficiently handle multi-dimensional nature of spatial data. If we solve these challenges, we will gain better performance compared to parallel platforms. In the next sections, we present MapReduce and its open source implementation, Hadoop. Moreover, we discuss these different challenges and a several research work that extends Hadoop to add support for spatial data and query.

Table 2.1 describes comparison between Parallel SDBMS and Hadoop

	Parallel SDBMS	Hadoop
Fault Tolerance		✔
Better for I/O bottleneck	✔	
Better for computations		✔
Handle multi-dimensional data	✔	

Table 2.1: Comparison between Parallel SDBMS & Hadoop

2.3. HADOOP

The previous section discusses the advantages of using distributed Hadoop platform, which is scalable and fault tolerant. This section describes MapReduce and its open source implementation, Hadoop. It describes Hadoop architecture and read/write procedure of HDFS. However, using Hadoop to process spatial data has many challenges so next section presents several spatial challenges and briefly describe the proposed solutions by several research work to address them and to extend Hadoop by the suitable support for spatial data and query.

2.3.1. Hadoop Architecture

Hadoop [12] is an open source implementation of MapReduce [11], which is a programming model for parallel data processing.

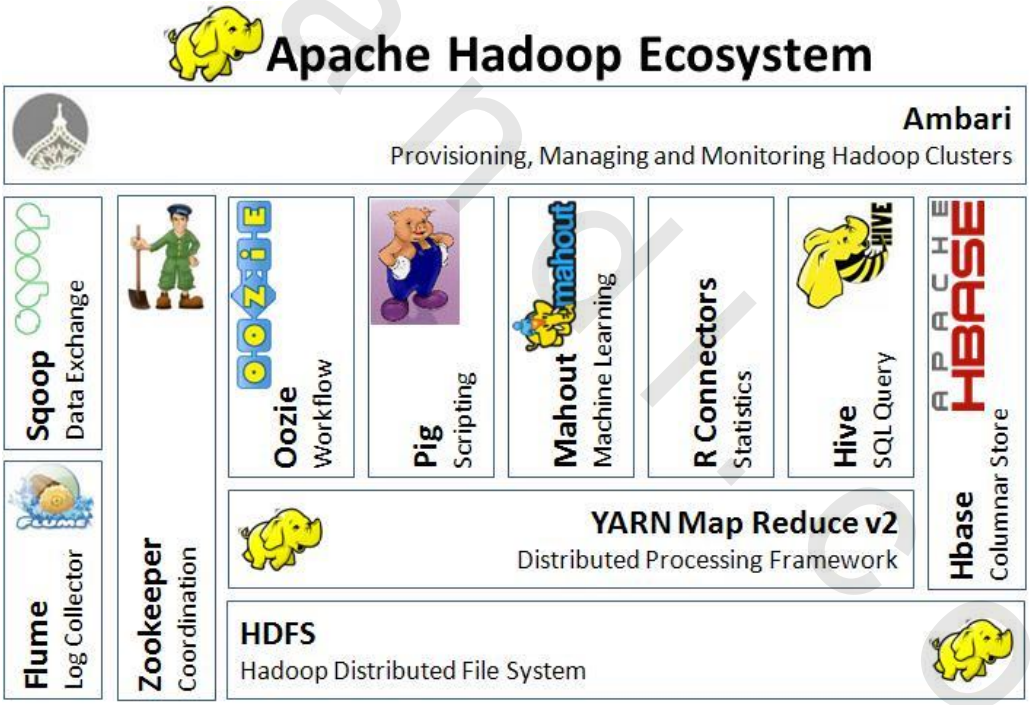

Figure 2.4: Hadoop Ecosystem [13]

Figure 2.4 shows the main components and tools that are currently available by Apache Hadoop Ecosystem. In this section, we focus generally on MapReduce module and especially on HDFS module.

Hadoop cluster consists of a master node and a multiple workers/slaves nodes. Each physical node in Hadoop cluster contains DataNode and TaskTracker. Only Master node has a NameNode and a JobTracker beside DataNode and TaskTracker. Figure 2.5 cited by [14], shows the physical and logical structure of Hadoop cluster nodes.

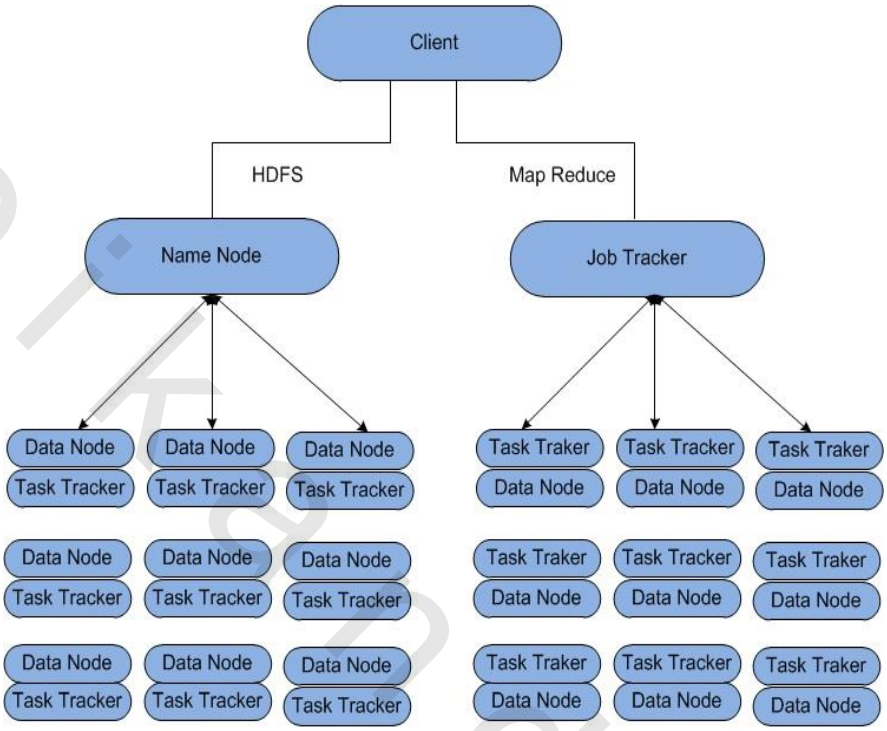

Figure 2.5: Hadoop physical architecture

Hadoop runs through a JobTracker and a number of TaskTrackers. The JobTracker runs on the master node of the cluster and TaskTrackers run at worker/slave nodes. The JobTracker distributes tasks over TaskTrackers, and it manages resources availability. Besides, it tracks tasks life cycle over TaskTrackers and maintains fault-tolerance for the entire Hadoop framework. Users submit new jobs to the JobTracker which breaks them to map and reduce tasks and assigns these tasks to TaskTrackers. The main job of the developer is to specify those map and reduce functions.

The data processed by Hadoop is stored and managed by Hadoop Distributed File System (HDFS) [12], which is based on the Google File System (GFS) [15]. The data is managed and information about it is kept about it through daemon processes: NameNode runs at the master node and a number of DataNodes run at worker/slave

nodes. The NameNode controls files distributions over namespace to keep balance, and it keeps metadata about other DataNodes and all files and directories. Examples of these metadata: files blocks and their locations over DataNodes.

DataNodes are the workers of the HDFS: (1) they store and retrieve blocks requested by NameNode, (2) they maintain a periodic heartbeat with NameNode and piggyback with the list of blocks that they are storing. Clients can access the data stored and maintained by HDFS by communicating their requests to the NameNode.

2.3.2. HDFS

HDFS partitions each file to multiple blocks (64 MB each by default) and replicates every block (three replicas by default) among the DataNodes to achieve reliability. Metadata of these blocks – as blocks replicas location- is stored in the NameNode.

Next, we describe the read/write procedures in HDFS as described and presented by "Hadoop: The Definitive Guide" book [16]. These procedures are extended to store spatial data in Section 2.4.

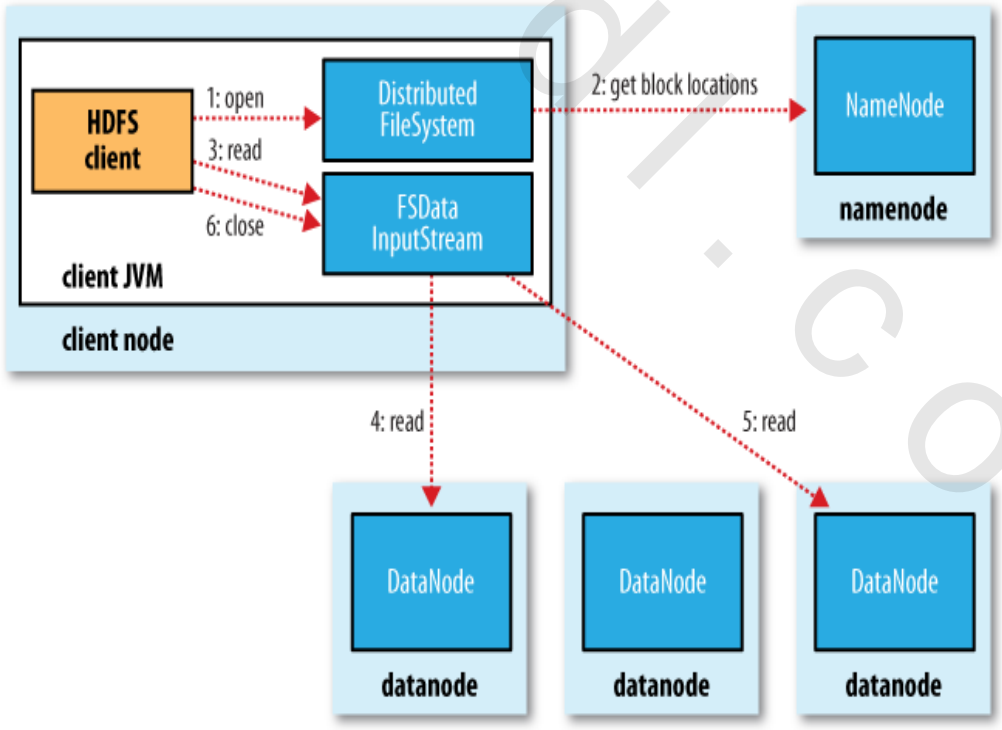

Figure 2.6: A client reading data from HDFS

Read File from HDFS

Figure 2.6 presented by [16], shows the steps of reading a file from HDFS. The steps are as follows: (1) HDFS client calls "open" method of DistributedFileSystem object. (2) "open" method calls the NameNode to determine the metadata this file's blocks. (3) The client calls "read" method of FSDataInputStream using the returned metadata. (4) "read" method chooses one location of file's replicas to stream data from it to the client. (5) FSDataInputStream repeats this step for all blocks. (6) Finally, it closes this input stream after it finishes.

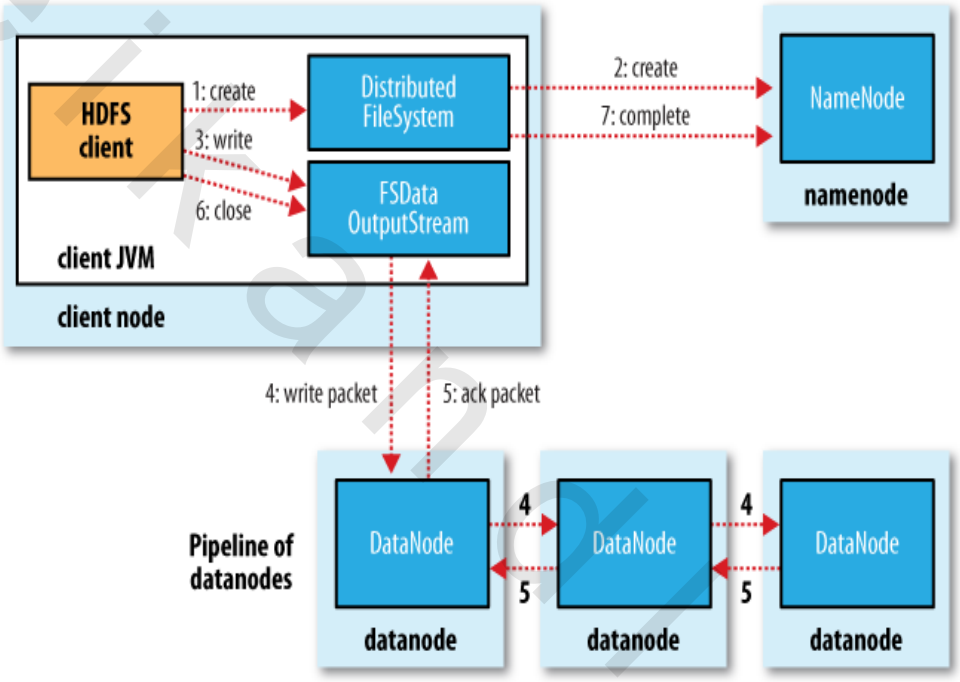

Figure 2.7: A client writing data to HDFS

Write File to HDFS

Figure 2.7 shows the steps of writing a file to HDFS. The steps are as follows: (1) client calls "create" method of DistributedFileSystem. (2) "create" method calls the NameNode to create new record for the new file. (3) The client uses FSDataOutputStream to split data into packets and to write them to data queue. This data queue asks the NameNode to allocate each block by choosing a list of suitable DataNodes to

its replicas using *default block placement policy* of HDFS. (4) This list of DataNodes forms a pipeline and the OutputStream streams packets to this pipeline's DataNodes. (5) After all nodes receive the packets, they send acknowledge to the OutputStream. (6) When the client finishes, it closes the stream. (7) Finally, the DistributedFileSystem notifies the NameNode that the write procedure is completed.

The *default HDFS block placement policy* used by write procedure, tries to balance load and read/write bandwidth without being aware of spatial data characteristics and without concerning about the relation between spatial partitions. Next chapter introduces new *spatial block placement policy* to enhance writing operation of spatial data.

2.4. SPATIAL DATA SUPPORT BY HADOOP

Section 2.2.2 mentioned the advantages of using distributed Hadoop platform, which is scalable and fault tolerant. However, using Hadoop to process spatial data has many challenges. In this section, we discuss these challenges in details. Besides, we present several research work that extends Hadoop to add support for spatial data and query.

2.4.1. Challenges of Extending Hadoop to Support Spatial Data and Queries

To move from parallel SDBM platforms to Hadoop framework we have to provide its components with several spatial capabilities: those are (1) declaring spatial data types, functions and operations, (2) building a query engine that can parse and execute spatial queries, and (3) updating HDFS storage layer with spatial properties to obtain better performance. Figure 2.4 shows the different layers of Hadoop ecosystem, each layer must be provided by the suitable spatial update. Storage layer needs several updates to handle the different features which were provided by SDBMS, such as: data indexing and partitioning. Besides, it must enhance the I/O bottleneck.

High Level Language

As Figure 2.4 shows that a high-level language is needed on top of MapReduce to provide it with high level, easy to use programming abstractions and a declarative query interface. Data flows systems compile queries written in high level languages: Hive QL

[17] and Pig Latin [18], respectively in workflows of MapReduce jobs that can be executed on Hadoop. Hence, they provide Hadoop with declarative query interface.

Hive and Pig can be used to declare spatial data types, spatial primitive functions and spatial operations. These spatial data types are useful for non-technical users to access and analyze their spatial data sets.

Operations Layer

Spatial systems use many types of queries; we can categorize them into non-spatial queries, spatial queries, integrated spatial/non spatial queries and multi-dimensional queries. These query types are defined as follows.

Non-spatial queries are usually aggregation queries, which are used to get mean values or select features using non-spatial attributes.

Spatial queries use spatial attributes in their operations. They select, filter or join spatial data using spatial attributes, which is described in Section 2.1.2. For example, the following range query:
"SELECT objects WHERE objects.X > X1 and objects.X < X2 and objects. Y > Y1 and objects.Y < Y2 "
We can categorize them to complex and non-complex spatial queries. Non-complex spatial queries are point based queries, containment queries (get features in specific region) and spatial joins. Complex spatial queries are large-scale spatial join queries between many spatial data sets, such as spatial cross matching or overlay and nearest neighbor queries.

Spatial/non-spatial queries combine between spatial and non-spatial attributes. For example, a query that selects all addresses of objects located in a specific region. The address is a non-spatial attribute (string) and the region is a spatial attribute.

Queries for multidimensional data are queries that access multiple data dimensions such as time and location. Examples of these queries are traffic and weather prediction and location based services (LBS) queries. Any location has different weather measurements for different times, and therefore a weather prediction query has to deal with both data dimensions. Additionally, examples of LBS are LBS range query, a query that de-

termines all users that are in a specific region in a specific time, and LBS KNN query, a query that determines the K nearest users from a specific geometric point given a specific time.

To execute these operations, we need a query engine that parses and executes the different queries but we cannot use a spatial database engine for such purpose because it is not feasible due to its tight integration with RDBMS engine and complexity on setup and optimization.

These operations can be implemented using the MapReduce functionality to be adapted to the partitions-based behavior of HDFS. These MapReduce functions must preserve queries meaning and they must be adapted to storage layer updates that will be discussed in the following paragraphs.

Indexes and Storage layers

Hbase [19] is an open source key-value store, which is used by Hadoop systems. Key-value structure means that every data record in HDFS contains number of columns and it is labeled by sortable key. Data records are sortable in Hbase based on this key.

Hbase can scale large updates of big data volumes with high throughput while being fault-tolerant and highly available. However, it does not support multi-attribute data access of spatial data nature; spatial data can be accessed using spatial attribute which contains a vector of two-dimensional points: X and Y values. Therefore, we need to build mutli-dimensional spatial index to arrange HDFS data based on mutli-dimensional spatial attribute. The absence of spatial indexing mechanism goes to full scan of the entire data files and the performance benefits obtained by Hadoop MapReduce is wasted. Spatial index layer enhances queries to scan the needed subset of data instead of full data. This will enhance performance and response time of spatial operations.

Spatial indexes such as R-Tree [20] and R+Tree [21] can do this job but they must be implemented using MapReduce functionality to be adapted to the partitioned-based behavior of HDFS. Additionally, spatial data partitioning is not easy because spatial data is skew. Besides, the nature of spatial objects leads to boundary object problem after

data partitioning which may cause incorrect query results. Additionally, spatial data records are both spatially and temporally, and the time dimension is unbounded.

Next sections discuss the several solutions of these challenges introduced by Hadoop-based spatial systems.

2.4.2. Spatial Index Use Case

This section illustrates the use case which describes the spatial indexing operation implemented by many Hadoop-based spatial approaches such as: [23], [24], SpatialHadoop [1], and Hadoop-GIS [26]. They use several spatial indexes such as R-Tree index.

Spatial records of spatial files – prepared by map providers- are two-dimensional data which changes with X and Y axes and they are not arranged by any specific attribute. To save a non-indexed file, HDFS partitions the file first into group of blocks. Each block size is equal to the block size of Hadoop environment (By default 64 MB), and contains spatial objects which are located in random locations on the map. If the user requests records of specific area, Hadoop scans all file blocks to return records of this specific area, and MapReduce parallelism advantage has been wasted by the full scan of the data. So as we have mentioned in the last section, two-dimensional spatial index layer is needed on top of HDFS.

The creation process of spatial index such as R-Tree, consists of two main phases: partitioning and indexing. R-Tree arranges records of the spatial file according to the spatial attribute and partitions them into multiple tiles. The size of each tile must be the same HDFS block size, so the number of obtained tiles is equal to the number of blocks obtained by the normal partitioning operation by HDFS for the same file. These tiles are the processing unit for querying tasks, and each one has different minimum bounding box (MBR) which is the outer rectangle that contains all the polygons in this tile. The area of the tile MBR is depending on the density of data; if the data is skew in a specific area, the tiles are partitioned recursively until the size of each output tile is equal to the size of normal HDFS block, and the area of each tile MBR will be smaller.

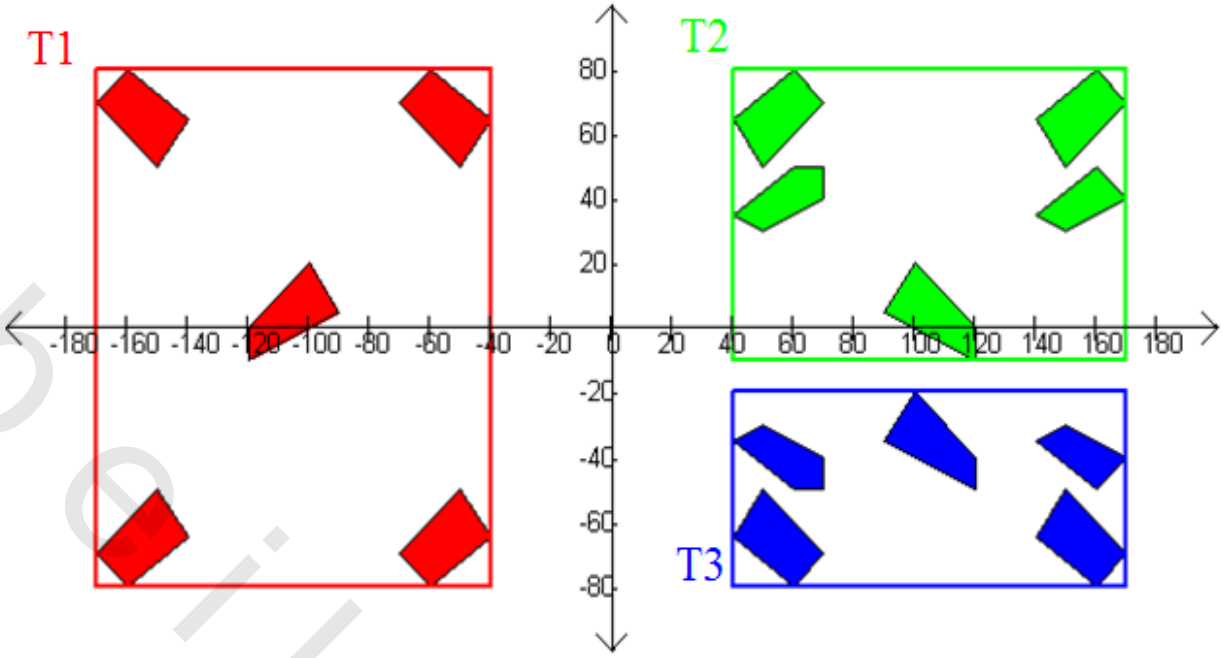

Figure 2.8: Spatial file Partitioning

For example, Figure 2.8 locates polygons of a spatial file on the map. If the size of this file is 190 MB, normal HDFS partitions it to three blocks (if the block size is 64 MB). The spatial indexing algorithm does the following steps before calling HDFS to save the file: (1) arranges polygons according to their places in the map, (2) partitions them into tiles, and each tile size must be equal to HDFS block size, and finally (3) sends each tile to HDFS to write it as a separate block. The indexing operation in our example creates three tiles or blocks, each one has different MBR. From Figure 2.8, the MBRs of the three tiles are:

T1: MBR [-170, -80, -40, 80], T2: MBR [40,-10, 170, 80], T3: MBR [40,-80, 170, -20].

If the client requests objects of the negative area of X-axis, the index filters tiles by their MBRs and returns T1 only to be scanned. We can notice that index layer here saves 60% of the time needed by data scan.

The experiments have shown that the overhead of building indexes is very small if it is compared by the time saved in MapReduce computations of intensive spatial queries and this index layer allows efficient real time processing for range and KNN queries. Besides, using modern hardware, building indexes does not require huge overhead.

2.4.3. Hadoop-based spatial systems

There are primary models that have used Hadoop and cloud computing framework without extending Hadoop such as the work proposed in [22]. This research has proposed the idea to use Hadoop and cloud computing framework with spatial data. It compares Hadoop framework with traditional parallel cloud computing platforms, and it discusses how Hadoop prevents the programming from caring about the distribution of the processes between nodes.

In this section, we discuss Hadoop-based spatial systems such as [23], [24], MD-HBase [25], SpatialHadoop [1], Hadoop-GIS [26]. These systems have proposed solutions to address the challenges mentioned in the last sections, by extending Hadoop to support spatial data and queries.

In [23], a new approach is proposed that implements two languages: (1) Well Known Text language (WKT) which presents spatial data to the user in understandable spatial data objects, (2) Well Known Binary (WKB) language which is used to manipulate and store spatial data on HDFS. In addition, it has implemented some spatial operations such as spatial join, intersection and overlapping. Finally, it has built spatial indexes using Quad-Tree and R-tree algorithms.

In [24], a new approach is proposed that uses linearization techniques to transform multi-dimensional spatial data to key-value data that can be used by Hadoop and to simplify the sorting during the indexing operation. It uses two strategies to partition multi-dimensional data: (1) Zorder values [27], which transforms multi-dimensional coordinates data into single-dimensional records to simplify the sorting in indexing operation. (2) Iterative algorithm based on X-means clustering [28], it clusters data of the space into equivalent clusters. Then, for every partition, it builds R-Tree index, and inverted index [29] using non-spatial attribute, which allows this system to efficiently execute both spatial and non-spatial queries. Figure 2.9 shows the architecture of [24]. Spatial data partitions are $\{d_1, \ldots, d_R\}$, every partition has R-Tree index *r* and inverted index *i*. Partitions indexes are gathered to local site which has spatial query processor that handles upcoming spatial queries.

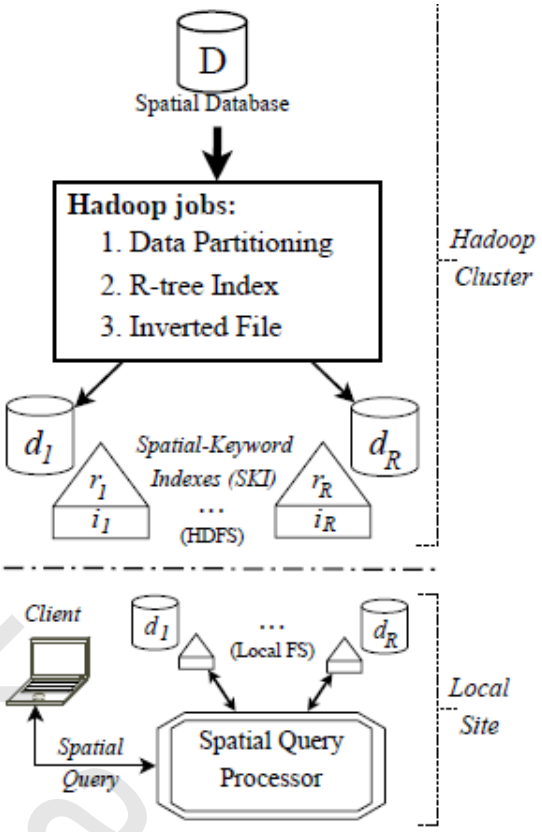

Figure 2.9: Architecture of [24] approach

Hadoop-GIS [26] approach has built spatial query language by integrating major operators and functions from "ISO SQL/MM Spatial" with Hive. It supports fundamental spatial queries such as point, containment and join queries. Besides, it has implemented some complex queries such as spatial cross-matching, spatial join and KNN queries. It adds to operations layer some components to provide spatial query processing such as "Spatial Query Translator", "Spatial Query Optimizer" and "Spatial Query Engine". Spatial Query Translator parses and translates different queries into abstract syntax tree. Spatial Query Optimizer takes this syntax tree and applies some rule-based optimizations to it. Spatial Query Engine supports following infrastructure operations: spatial relationship comparisons such as intersects, centroid, contains, and spatial access methods such as R/R+ trees.

For index layer, Hadoop-GIS has combined between two approaches Global spatial indexes and On-demand local indexes using R+tree. Pre-generated indexes might not be suitable to support dynamic spatial query types.

Finally, it has solved some partitioning problem such as boundary objects problems. The nature of spatial objects leads to boundary object problem after data partitioning which may cause incorrect query results. They have tried to solve this issue by using the "multiple assignment-based" approach, which replicates objects and assigns them to each intersected tile. Queries are followed by a post-processing step to redeem results from redundant objects.

MD-HBase [25] approach has also implemented operation layer contains some spatial operations such as Range query and KNN query.

It has used linearization techniques (Z-ordering) like [24] to transform multi-dimensional location information into a one-dimensional space. It implemented the index layer using multi-dimensional index structure on top of the Hbase Key-value store using K-d trees or Quad trees. Unlike previous approaches, it uses multi-dimensional index because it supports location and time dimensions. This index layer is also partitioned - for better scalability and load balancing - using B+ tree into two levels (ROOT & META). MD-Hbase caches the index layer and connection sharing between clients on the same host to reduce the number of accesses to the index.

SpatialHadoop [1] is a spatial model framework based on Hadoop. Figure 2.10 shows the different modules that SpatialHadoop has implemented on top of Hadoop to provide it with spatial capabilities and to solve mentioned challenges.

First module is a spatial language layer, which extends Pig Latin high-level language. SpatialHadoop adds some new spatial constructs to Pig while preserving the original functionality. In addition, it provides a built-in support for spatial data types (Point, Rectangle, and Polygon), spatial primitive functions (Distance, Overlaps, MBR …) and spatial operations (Range query, KNN and spatial join). These spatial operations are implemented using MapReduce functionality. Developers can implement new spatial operations.

SpatialHadoop indexes spatial files using spatial indexes such as grid, R-tree or R+-tree. The index layer has two levels: local and global. Each local index is responsible for one tile to organize its records. The global index concatenates MBRs of local indexes into one big file, which is stored in the main memory of the master node to organize

partitions across all nodes. It is small and can be pre-generated or shared across cluster nodes through Hadoop distributed cache mechanism.

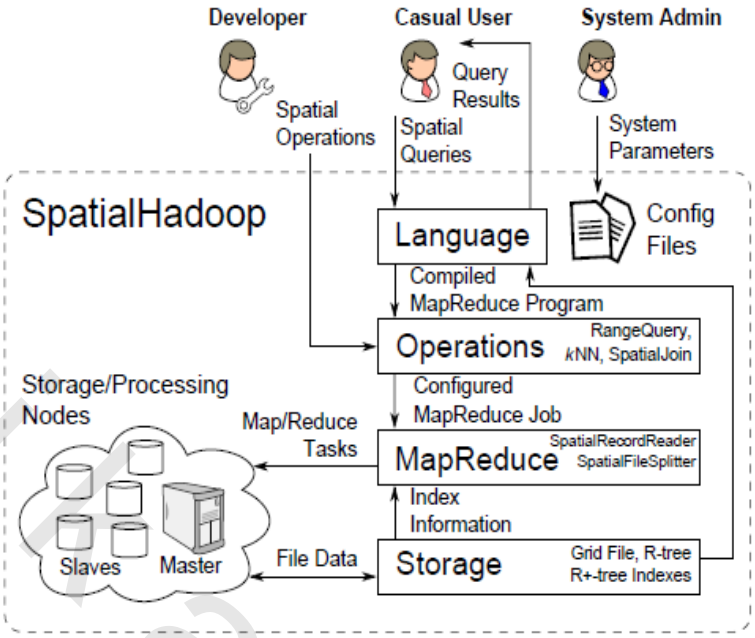

Figure 2.10 : SpatialHadoop [1] Architecture

To connect operation layer with index layer, SpatialHadoop [1] has introduced two new components to the MapReduce layer “SpatialFileSplitter” and “SpatialRecordReader". “SpatialFileSplitter” uses global index to filter file blocks that do not contribute to the query answer – outside query range for example. “SpatialRecordReader” uses local index to filter records of the scanned tile instead of iterating over all of them one-by-one. These two components are used by spatial operations to enhance the performance of queries.

We can summarize last paragraphs using Table 2.2, and introducing the main advantages of each system that distinguish it from others. [23] has implemented WBT to simplify data storing. [24] has used several linearization techniques to partition spatial data, and it has built inverted indexes using non-spatial attributes. Hadoop-GIS [26] has combined between two approaches: Global spatial indexes and On-demand local indexes, and has solved partitioning boundary objects problem. MD-Hbase [25] has built multi-dimensional index on top of Hbase, and it cache its indexes. SpatialHadoop fea-

tures are close to Hadoop-GIS, besides, it in a well-implemented open source platform that helps developers to implement their own operations to the operation layer.

Approach	High Level language & Operations Layer	Co-location	Index Layer	Boundary intersection	Index Optimization
[23]	✔		-Quad/R trees		
[24]			- R-tree -Inverted files for non spatial attributes		
SpatialHadoop [1]	✔		-Grid/R/R+ trees -Local/Global index		
Hadoop-GIS [26]	✔		-R+ tree -Local/Global index	✔	✔
MD-HBase [25]			-K-d/Quad trees		✔
Co-Hadoop [2]		✔			
Co-SpatialHadoop	✔	✔	-Grid/R/R+ trees -Local/Global index -Inverted files for non spatial attributes		✔

Table 2.2: Spatial approaches challenges

2.5. COLOCATING DATA IN HADOOP DISTRIBUTED FILE SYSTEM

To write new file to HDFS, it partitions files to fixed size blocks and it saves them over cluster's nodes using *default block placement policy*. To save block replicas, this policy chooses DataNodes that guarantee data load balance over Hadoop cluster without consider data characteristics.

Co-Hadoop [2] is a lightweight extension of HDFS to allow applications to control where file blocks are stored without affect Hadoop properties such as fault tolerance and load balance. This can be achieved through partitioning the various data sets accessed by a query and colocationg them together on HDFS. The main objective of Co-Hadoop to reorganize the data on HDFS so queries can be written to be executed in map only jobs. It has introduced new block placement policy that receives hints from applications that some files are related and may be processed together, therefore they need to be colocated together to improve the execution of queries that join these files.

Colocating files has the following advantages: (1) it eliminates data shuffling in MapReduce tasks, and (2) it decreases network overhead because the data processed by any map task will be located on the node executing this map task. It can be used to improve the efficiency of many exhaustive operations, such as indexing, grouping, aggregation, joins, and sessionization.

2.6. PROPOSED EXTENSION

In Section 2.4, we describe several Hadoop-based systems that support spatial data and queries. However, the solutions proposed by all these systems focus on providing a query engine that supports spatial queries, its operations, and spatial indexes. The proposed solutions by all these research works are on top of Hadoop and HDFS, without any change of them. They used MapReduce functionality to index files and to implement different spatial queries. These approaches partition and index spatial files to enhance the performance, but HDFS randomly distributes the replicas of these indexed blocks between nodes using *default block placement policy*. It guarantees data availabil-

ity, load balance and fault-tolerance between all nodes without considering spatial data characteristics.

This thesis introduces Co-SpatialHadoop, an extension to SpatialHadoop [1]. It extends HDFS to colocate related blocks of spatial files that are accessed together by queries. It uses SpatialHadoop libraries to index spatial files and it introduces *new spatial block placement policy* to physically store the indexed blocks together. The approach of colocating related blocks is inspired by the approaches introduced by Co-Hadoop [2]. In addition, it extends SpatialHadoop by creating inverted indexes on non-spatial attributes in the data to enhance the execution performance of non-spatial queries. The idea of building inverted indexes [29] on non-spatial attributes is inspired by the approach described in [24].

2.7. CONCLUSION

We have presented in this chapter the required background to understand this thesis. We have briefly described how spatial support has been added to distributed database management systems and parallel database management systems. Moreover, we have listed the challenges that need to be addressed to allow Hadoop to store spatial data and process spatial queries. We have discussed various approaches that have been proposed in the literature that address these challenges. We have identified the challenges that yet to be addressed by these systems. We address some of them in Co-SpatialHadoop as we describe in the next chapter.