CoS-HDFS: Co-Locating Geo-Distributed Spatial Data in Hadoop Distributed File System

Mariam Malak Fahmy, Iman Elghandour, Magdy Nagi
Computer and Systems Engineering
Alexandria University
Alexandria, Egypt
m.malak.fahmy@gmail.com, ielghand@alexu.edu.eg, Magdy.nagi@ieee.org

ABSTRACT

Given the recent advancement in the ubiquitous positioning technologies, it is now common to query terabytes of spatial data. These massive data are usually geo-distributed across multiple data centers to ensure their availability. Yet, at least one replica of the data is stored close to where the data are generated. Spatial queries are complex and computationally intensive, and therefore, distributed computation platforms, such as Hadoop are now used to improve their execution time. However, Hadoop is agnostic to the spatial data characteristics, and it randomly partitions and locates the data stored in its distributed file system which degrades the performance of the execution of spatial queries. In this paper, we propose CoS-HDFS, an extension to the Hadoop Distributed File System (HDFS) that takes into account the spatial characteristics of the data and accordingly co-locates them on the HDFS nodes that span multiple data centers. We integrate CoS-HDFS with SpatialHadoop, a MapReduce framework that natively supports spatial data, to make use of its implementation of spatial indexes, operations, and query interfaces. We experimentally demonstrate significant reduction in the network usage and total execution time in the case of spatial join queries on the TIGER dataset.

CCS Concepts

•Information systems → MapReduce-based systems; Geographic information systems;

Keywords

HDFS; Spatial Data; Co-location; Geo-distribution

1. INTRODUCTION

Many electronic gadgets that are equipped with positioning systems and satellites are now generating huge amounts of spatial data. Applications in various domains such as business, targeted advertisement, weather analysis and prediction, and traffic management execute various types of spatial queries and perform complex and intensive computations. In business, K nearest neighbor (KNN) queries and range queries are commonly used for geo-marketing and recommendation services. In social network and GIS systems, spatial join queries are commonly used. For example, maps are typically stored in different files (layers) where each contains objects (e.g. roads, points of interests, rivers, parks, etc.) that are related. Queries typically join different layers of the map to discover information such as the rivers passing through a park (rivers x parks) or the roads leading to a certain park (roads x parks). Spatial datasets that are commonly processed by these applications are as large as gigabytes and terabytes of size [11].

Traditional Spatial Database Management Systems (SDBMS) can no longer support the explosion in the sizes of the spatial datasets. Meanwhile, the MapReduce platform [7] and its open source implementation Hadoop have been successfully used in analyzing huge data in many different domains. Hadoop has the advantage of scalable processing and abstracting the parallelism of query processing. However, using Hadoop to process spatial data has many challenges, which are: (1) being unaware of spatial data properties, types, and models; and (2) query processors in the Hadoop stack do not understand spatial operations in queries. Support for spatial data and query can be integrated into the Hadoop ecosystem by providing its components with several spatial capabilities, which are: (1) adding support for spatial data types, functions and operations; (2) extending the existing query engine to allow it to parse and execute spatial queries; and (3) extending the data storage layer to support creating indexes to spatial files and using them to answer queries. These challenges have been addressed by several systems that extend Hadoop to add support for spatial data [6, 12, 17]. Changes are made to the following layers of the Hadoop stack: (1) the storage layer by custom partitioning the spatial files and adding spatial indexes; (2) the query language layer by adding support to spatial data types and operations (e.g. Pigeon [10] and Hive extension in Hadoop-GIS [6]); and (3) the operations layer by including efficient implementations of frequently used spatial operations and by allowing these spatial operations to use spatial indexes [8].

Massive data are usually geo-distributed across multiple data centers due to one or more of these reasons: (1) data are generated at different locations that are spread around the globe; (2) storing data close to users guarantees low latency

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

BDCAT'16, December 06-09, 2016, Shanghai, China

© 2016 ACM. ISBN 978-1-4503-4617-7/16/12...$15.00

DOI: http://dx.doi.org/10.1145/3006299.3006314

access of these data; and (3) availability of the data in case of failures. An example scenario is the world lightening maps maintained by University of Washington [5]. Data collected from sensors that are spread all over the world are analyzed to generate lightening maps. The data generated at each location is huge, therefore data are streamed to a geographically near data center. One solution to analyze this data is to copy data generated and stored at remote data centers to one location and analyze them together. However, this solution is not efficient and has a high network cost and high latency. A more efficient alternative is to replicate the data to other data centers (in addition to the copy that is close to where they are generated) divide the analysis query into parts that can be executed separately on each partition of the data [20]. The new interesting challenge is to keep data partitioned among multiple data centers and divide the analysis tasks to be performed locally on each one of them.

The various solutions that extend Hadoop to add spatial data support [6, 12, 17] do not take into account the case that data are geo-distributed. We propose to co-locate the data that are typically accessed together in the queries. When the query is translated into a MapReduce job(s) and divided into tasks that execute on partitions of the data, those partitions that are accessed together will be co-located on the same nodes. A key feature of the proposed solution is to push awareness of the spatial properties of the data to HDFS, and to co-locate blocks of these data based on their spatial properties.

In this paper, we present CoS-HDFS, an extension of HDFS that co-locates data blocks of files stored in it based on their spatial properties. CoS-HDFS maintains metadata about the storage location of the blocks of each file and relates it to their spatial properties. Additionally, it modifies the policy that HDFS employs to place data blocks on the cluster nodes to co-locate multiple files based on their spatial properties. CoS-HDFS introduces the following three advantages. First, it reduces the network traffic incurred during query execution, and hence reduces the execution time of these queries. This is important when the query input files are geo-distributed across nodes located in multiple data centers. In this case, the network cost is not negligible. We note that the queries that benefit the most from co-location employed by CoS-HDFS are the spatial join queries. Second, adding new files to HDFS and co-locating them with already existing files can be done autonomously without the need to perform the co-location of the files from scratch. Third, CoS-HDFS is fully integrated with SpatialHadoop [9, 12] and exploits its full-fledged support for spatial data and queries.

The main contributions of this paper are as follows:

- A new block placement policy for HDFS that co-locates spatial files based on the spatial properties of each block of the data.
- An implementation that integrates CoS-HDFS with SpatialHadoop [12] to leverage its full-fledged support of spatial data on the language, operation, and index layers.
- An experimental study using the TIGER [4] real dataset.

The rest of this paper is organized as follows. We present related work in Section 2. Section 3 gives a high-level overview of CoS-HDFS and related systems. We describe the details of CoS-HDFS in Section 4. Finally, we describe our experimental study to evaluate CoS-HDFS in Section 5 and conclude in Section 6.

2. RELATED WORK

Adding support to spatial data and queries in commercial and research database systems have been long studied, and Spatial Database Management Systems (SDBMS) have been extensively used by users for years. These systems have many limitations considering the recent evolution of the sizes of spatial data files. To achieve more scalability, parallel SDBMSs have been introduced to reduce the I/O bottleneck by partitioning spatial data on multiple parallel disks [18]. 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 [19]. The PAIS spatial database system addressed these challenges as presented in [21, 22]. PAIS leverages parallel SDBMS to scale out spatial data and query executed on them. However, it requires dedicated hardware and sophisticated tuning efforts.

MapReduce [7] introduces new features as compared to parallel database systems, mainly, scalability and fault tolerance. Hadoop [1] is an open source implementation of MapReduce. The work proposed in [24], introduces a new layer on top of Hadoop without making any changes to it. Other Hadoop-based spatial systems such as SpatialHadoop [12], Hadoop-GIS [6], and MD-HBase [17] have proposed solutions to address the challenges mentioned above, by extending Hadoop to support spatial data and queries. These approaches can be categorized as follows: (1) adding new layers to the Hadoop stack to support spatial indexes for data stored in HDFS and processed by Hadoop. The main goal of this approach is to improve accessing spatial data; and (2) extending the language layer of the Hadoop stack, which allows compiling queries written in high level languages to Hadoop job(s). The extended language supports spatial data types and spatial primitive functions and operations. These proposed solutions run on top of Hadoop and HDFS, without any changes of them. They use the MapReduce functionality to index files and to implement different spatial queries. CoS-HDFS adds awareness to spatial data in the storage layer, which allow us to co-locate blocks of the files based on their spatial properties.

Co-partitioning tables and co-locating these partitions have been extensively studied in the context of parallel relational database systems [16]. A data placement advisor is integrated into the database system to advise about placement strategies of the table partitions and materialized views. The main objective is to minimize the execution time of a given query workload. Co-partitioning data files and co-locating these partitions have been proposed in the context of HDFS in CoHadoop [13]. CoHadoop partitions multiple data files that are queried together using the same partitioning function and places related partitions on the same HDFS cluster nodes. CoHadoop shows that co-partitioning and co-locating files stored on HDFS can improve the execution performance of join and sessionization queries. It relies on hints from the users about which files to co-locate and what partitioning function is suitable for the files. More-

over, it uses a locator table to keep track of the files that need to be co-located and changes the HDFS data blocks placement policy to enforce storing blocks that need to be co-located on the same HDFS nodes. CoS-HDFS employs a similar approach for using a locator table to keep track of the file blocks that needs to be co-located based on their spatial characteristics. Morover, CoS-HDFS autonomously decides on the file data blocks that need to be co-located.

Geo-distribution of big data is very common to achieve availability and low latency of local access of the data [15, 20, 23]. Most of the work studies the placement of the data replicas and the consistency between these replicas [23]. The work presented in [20] studies how analytic queries can be executed on data that are distributed across multiple data centers. In this paper, we focus on spatial data and queries. Co-locating partitions of multiple datasets based on their spatial characteristics reduces the network traffic incurred while executing the queries, and therefore their execution time.

3. OVERVIEW

3.1 Data Placement in HDFS

HDFS [3] follows the Google File System (GFS) [14] in partitioning files stored in it into blocks and storing multiple replicas of each block. HDFS employs a block placement policy that has the following objectives: (1) maximize reliability and availability; (2) optimize network bandwidth utilization; and (3) balance the load on the HDFS nodes. These objectives are enforced through the policy HDFS uses to place the data blocks and their replicas on its nodes. The factors that HDFS considers to serve their objectives are as follows [14]: (1) store replicas of the same block on nodes that are spread on various racks to guarantee availability if a rack goes down; (2) store at least two replicas on nodes of the same rack for better network bandwidth utilization; and (3) store new blocks on HDFS nodes with below-average disk space utilization to guarantee load balancing among nodes on the long run. To put it together, the HDFS placement policy for a block and its replicas is as follows:

- Choose a node in the local rack with below-average disk space utilization to put the first replica on.
- Choose two nodes from a different remote rack to put the other two replicas on.
- If the number of replicas is greater than three, then spread the rest of the replicas on nodes of different racks.

HDFS divides files into blocks of equal size. It does not take into consideration the contents of these blocks. Furthermore, the placement of these blocks and their replicas is agnostic to the contents of these blocks. However, the contents of a spatial file are spatial objects such as points and polygons. Adding awareness of the spatial properties of the objects contained in these blocks and placing blocks that are usually accessed together on the same HDFS nodes can reduce the execution time of operations performed on these data.

3.2 Indexes in SpatialHadoop

A spatial file contains shapes such as points, rectangles, and polygons. SpatialHadoop [12] partitions the data according to their spatial properties. These partitions are of the same size as the block size in HDFS. The new partitioned data is stored in HDFS, a local index is created for each partition, and a global index is created to index all these partitions (blocks). The procedure for creating an index for a dataset is as follows. First, SpatialHadoop creates a MapReduce job that scans the entire file and partitions it into blocks of the same size as the HDFS blocks. The objects that are spatially near by are assigned to the same block. Second, for each block, the objects assigned to that blocks are used to create a spatial index (e.g. grid or R-tree). This means that these objects are now organized as a spatial index that is local to this block. Finally, SpatialHadoop creates a global index (e.g. grid or R-tree) that locates each block given its rectangular boundaries. This is a one level index that indexes each block using the spatial rectangular boundary of the objects stored in it. This global index is stored in the memory of the master node.

SpatialHadoop stores the data partitions of the index on HDFS using its default placement policy described in Section 3.1. This means that a MapReduce task reading multiple data blocks from different datasets, which is the case of spatial join [25], will probably read data from a remote node of the cluster. The latency caused by remote reads cannot be ignored if the cluster is geo-distributed and the nodes are spread across multiple data centers. Next, we describe how we co-locate blocks that are processed together by MapReduce tasks based on their spatial data characteristics.

3.3 Overview of CoS-HDFS

SpatialHadoop partitions the data into blocks and creates a global index to locate these blocks given their rectangular boundaries. These blocks are stored on HDFS. Therefore, when a map task is processing two of these blocks, even though they have objects that are spatially nearby, they might be stored on two different nodes. Even worse, they can be stored on nodes on different data centers. CoS-HDFS addresses this challenge. For each block of the file, we identify its minimum bounding rectangle (MBR), which is a 2D rectangle that encloses all the spatial shapes in this block. CoS-HDFS changes the HDFS default placement policy to use the MBR of the file blocks to co-locate the blocks with overlapping or neighboring MBRs. These are the blocks that are expected to be accessed together in queries such as spatial join.

To illustrate the opportunities gained from the co-location of spatial files, we use the example depicted in Figure 1. Consider two spatial files `A` and `B`. To store them on a distributed file system such as HDFS, they are partitioned into blocks. The figure shows an example of how the files `A` and `B` can be partitioned[1]. The figure also shows the objects in each partition (block) and the MBR of that block represented as the coordinates on X and Y axes. Note that HDFS partitions a file into blocks of equal sizes, which means that the data stored in these blocks are equal but the perimeter of the MBR may vary. HDFS creates replicas of each block and stores them on its nodes. Figure 2 shows the blocks of

[1]Note that we are using a neat partitioning of the files for the purpose of the clarity of the example.

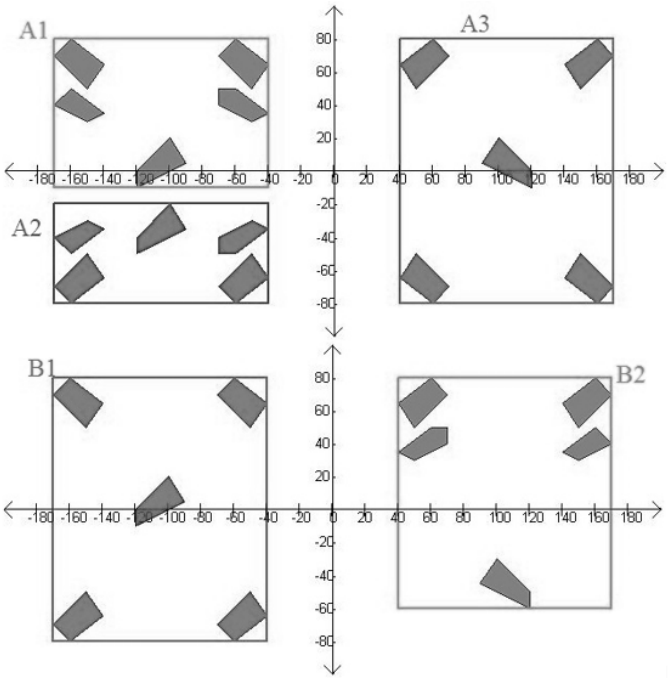

Figure 1: An example of blocks of two spatial files A and B.

the files `A` and `B` and their replicas as distributed across an HDFS cluster of eight nodes. These nodes are distributed across four data centers that are located in USA, Europe, Asia, and Australia. As discussed in Section 3.1, the main objective of HDFS is to balance the data stored on all its nodes. Next, we describe a spatial join query that takes as input the two files in our example.

The spatial join query merges two records, one from each input spatial file. The join condition requires that the spatial features of the merged records are matched, which is reflected as an overlap in their MBR. A spatial join query will check all the shapes (records) in file `A` with those in file `B`. Spatial indexes created for the files are typically used to filter the records so only those with overlapping MBRs are considered by the join operation.

An implementation of MapReduce spatial join (SJMR) [25] creates `VxW` map tasks to join the shapes in the two files, where `V` are the number of block in the first file and `W` are the number of blocks in the second file. In the example, `3x2` map tasks are created to join the shapes in the blocks of the two files: (`A1xB1`), (`A1xB2`), (`A2xB1`), (`A2xB2`), (`A3xB1`), and (`A3xB2`). However, given the spatial properties of the contents of the blocks of `A` and `B`, only these map tasks (`A1xB1`), (`A2xB1`), and (`A3xB2`) will produce outputs.

SpatialHadoop [12] creates an index (Grid, R-tree, or R+ tree) of the blocks of the files that are distributed on the HDFS nodes. Therefore, the queries executed using SpatialHadoop create only the required map tasks. In the example, only map tasks that join (`A1xB1`), (`A2xB1`), and (`A3xB2`) are executed as shown in Figure 2. However, the pairs of blocks that are joined together in map tasks are not guaranteed to be located on the same nodes. The Hadoop scheduler follows the design of MapReduce [7] and does a best effort attempt to schedule map tasks on the same nodes where their input blocks are located. Figure 2 shows how the map tasks can be started on nodes where one of the input blocks is stored, yet the other one is read remotely from a node that is located in a different data center. In the example, these three map tasks are performed: (1) the map task processing the blocks (`A1xB1`) reads both of them local to the USA region; (2) the map task processing the blocks (`A2xB1`) is executed in Europe region, which means that one block is read locally and the other is read from a remote data center (Australia); and (3) the map task processing the blocks (`A3xB2`) is executed in Asia region, which means that one block is read locally and the other is read from a remote data center (Europe); The data blocks read from remote data centers are expected to incur a high network traffic cost, which increases the execution time of the query.

CoS-HDFS modifies the placement policy of HDFS (Section 3.1) to enforce the placement of the blocks that are joined together on the same nodes. In the example, the replicas of the blocks `A1`, `A2`, and `B1` are better co-located together. Similarly, the replicas of the blocks `A3` and `B2`. Figure 3 shows the state when co-location of blocks is employed. In that case, all of the three mappers read their blocks locally.

This proposed co-location of the blocks of the spatial files `A` and `B` gives the scheduler of Hadoop a higher opportunity to schedule map tasks that read their input blocks locally. This is expected to reduce network overhead and therefore to reduce the total execution time of the spatial join queries. Co-location of files, is of additional importance when the HDFS cluster of nodes is very large and spans multiple racks as the network bandwidth connecting remote racks is usually slower than that connecting nodes in the same rack. Next, we describe how CoS-HDFS co-locates spatial files.

4. CO-LOCATING SPATIAL DATA FILES IN COS-HDFS

CoS-HDFS integrates the awareness of spatial data into HDFS. This is achieved through autonomously building a locator table that incorporates spatial data information about the blocks of the files stored in CoS-HDFS. Each locator table entry represents data blocks that are placed on the same nodes of HDFS. The contents of data blocks assigned to the same locator table entry need to be spatially related. To guarantee that, we divide the locator table into multiple entries, and we assign a unique MBR for each entry. Moreover, for each block in the spatial files that we want to store on HDFS, we calculate its MBR and assign it to the spatial locator entry with an overlapping MBR. All the replicas of data blocks assigned to the same entry in the locator table are co-located, which means that they are assigned to the same set of nodes in the HDFS cluster.

The high level overview of building and using the spatial locator table to co-locate data blocks based on the spatial characteristics of their contents is as follows. We divide the spatial locator table into multiple entries each has a specific MBR. SpatialHadoop creates an index and partitions it into multiple blocks, and then stores these blocks the distributed file system. We calculate the MBR of each data block based on the object shapes contained in it. Next, we find an entry in the locator table whose MBR contains or overlaps with the MBR of the data block. Next, we assign this block to that entry of the locator table. All the replicas of data

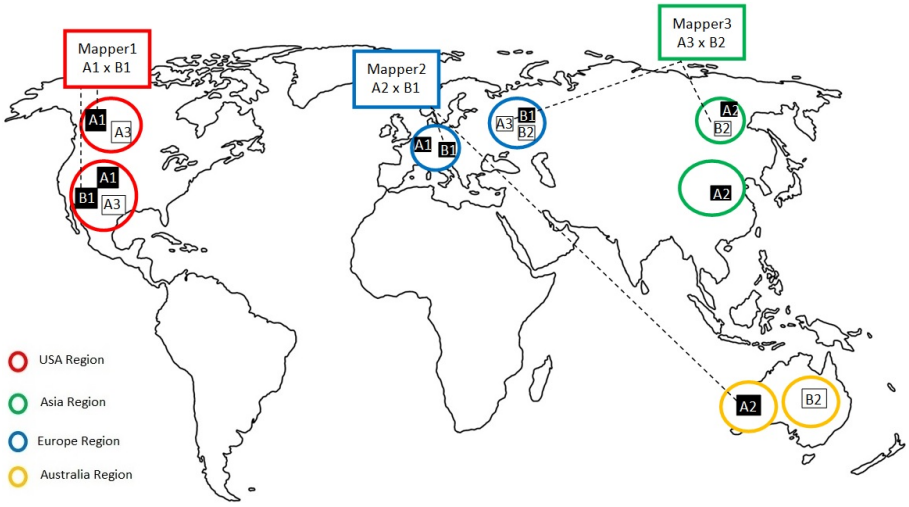

Figure 2: Storing the blocks of the files A and B in HDFS. The HDFS cluster contains eight nodes that are located in four different data centers. Two of the three mapper tasks in the example read one of the data blocks form a remote data center.

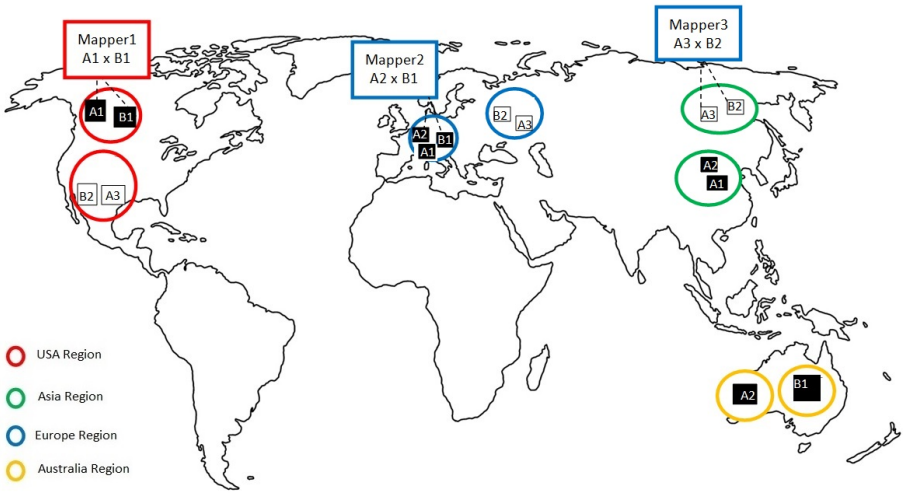

Figure 3: Co-locating the data blocks of files `A` and `B` in CoS-HDFS. The HDFS cluster contains eight nodes that are located in four different data centers. The three mapper tasks read their input locally.

blocks assigned to the same entry in the locator table are co-located, which means that they are assigned to the same set of nodes in the CoS-HDFS cluster.

In the rest of this section, we first introduce challenges that are related to co-locating spatial data. We then discuss how we build on the idea of the locator table used in [13] to autonomously build a spatial locator table that we can use to co-locate blocks of data files based on the spatial properties of their contents. Finally, we discuss how we exploit the spatial locator table to co-locate data files.

4.1 Challenges of Building a Spatial Locator Table

We employ a spatial locator table to help us co-locate data blocks based on the spatial characteristics of their contents. CoS-HDFS keeps the features guaranteed by HDFS, namely load balancing among the cluster nodes and fault tolerance. The following two challenges are introduced, and we propose solutions to address them in CoS-HDFS.

4.1.1 *Balancing Data on CoS-HDFS Nodes*

One of the objectives of HDFS is to balance the data stored on all the nodes in its cluster. This is reflected in the data placement policy briefly described in Section 3.1. CoS-HDFS need to retain this feature. However, enforcing a new placement policy based on how the data blocks are distributed in the spatial locator item can disturb the load balancing of data in the CoS-HDFS cluster of nodes.

4.1.2 *Overlapping between Blocks and Multiple Locator Table Entries*

We assign each block in the data to one entry in the spatial locator table. This decision is made by calculating the MBR of all the shapes contained of each data block and matching the resulting MBR with the MBRs of each entry in the spatial locator table. However, the MBR calculated for a data block can overlap with the MBRs of multiple entries in the spatial locator table. It is important to decide which of these entries to choose to assign a given data block. This

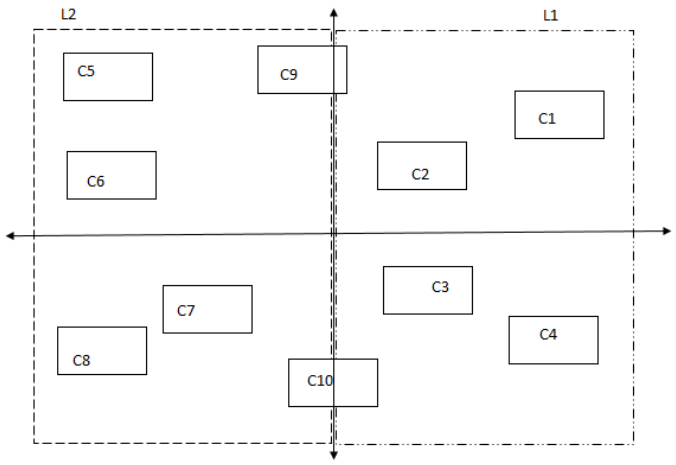

Figure 4: Example: Overlapping between MBRs of data blocks and MBRs of spatial locator table entries.

choice affects the opportunities given to Hadoop to schedule map tasks that perform only local reads.

Figure 4 illustrates an example of blocks overlapping with the MBR of multiple locator items. The two dotted rectangles L1 and L2 represent the MBRs of two entries in the locator table. The solid rectangles C1 through C10 represent blocks of a spatial file C. The MBRs of the blocks C1, C2, C3, and C4 are only overlapping with MBR of L1 and therefore we can simply assign to the locator table entry L1. Similarly, the MBRs of the blocks C5, C6, C7, and C8 are only overlapping with the MBR of L2 and therefore we assign them to the locator table entry L2. However, the MBRs of the blocks C9 and C10 are overlapping with both locator table entries L1 and L2. In such case, we need to choose between these two locator table entry to assign each block to only one of them.

4.2 Spatial Locator Table

The design of the spatial locator table needs to guarantee the following: (1) the number of data blocks assigned to each entry in the locator table are roughly close to each other; (2) The mapping between the data blocks assigned to each spatial locator table entry and the nodes in the CoS-HDFS cluster fairly distributes these blocks on the cluster nodes; and (3) the division of the table into multiple entries reflects the spatial characteristics of the input files. To achieve these design goals that mainly address the challenge described in Section 4.1.1, we use a spatial index (Grid, R-tree, or R+ tree) to guide us to how to divide the spatial locator table into entries. The spatial index gives us a hint about the correct MBR for each spatial locator table entry so the number of data blocks assigned to this entry are roughly equal. To avoid duplicating the work, we leverage the spatial index created by SpatialHadoop. This means that if a Grid index is created by SpatialHadoop for the data, we use that index as a hint to decide on the MBRs of the locator items in the spatial locator table.

However, the global index created by SpatialHadoop is a one level index, where each leaf node corresponds to a block in the data. Dividing the locator table to the same number of leaf nodes in this index will not be useful because each block will be assigned a different entry in the table. Our objective is that each entry of the table covers a wider bounding rectangle to be able to include multiple blocks. Therefore, we decide on the number of entries in the locator table as described later in this section. We send this number to the index creation module in SpatialHadoop to construct a global index with the number of leaf nodes equal to the number of locator table entries. We only use this index to identify the MBR of each locator item entry.

Locator MBR	Blocks
L1: MBR[-170, -80, -40, 80]	A1, A2, B1
L2: MBR[-40, -80, 170, 80]	A3, B2

Table 1: Spatial locator table for the example shown in Figure 3.

Table 1 shows the locator table that corresponds to the example data block distribution in Figure 3. `A1`, `A2`, and `B1` are assigned to the same locator table entry, and therefore they are co-located on the same CoS-HDFS nodes. Similarly, A3 and B2 are assigned to the same locator table entry that is different from the other one, and therefore, they are co-located on a different set of nodes. This placement of the blocks on CoS-HDFS allows at a higher probability that the scheduler of Hadoop schedules map tasks that locally reads all its input (Figure 3).

As mentioned above, we use a spatial index to guide us on how to divide the spatial locator table. We can use any of these spatial indexes: Grid, R-tree, or R+ tree. The key is that we use the same type of index created by SpatialHadoop for given datasets. In our experimental evaluation, the load balancing of blocks on locator items was best achieved when we used R+-tree, this is mainly because there are no overlapping between the MBRs of the index nodes, which make the balancing of nodes easier. This is also consistent with the results presented in [12].

Another issue that we address is the number of entries in the spatial locator table K. The choice of K can affect the performance of the CoS-HDFS because it specifies the blocks that are co-located. Small K means that a very large number of blocks of the file are expected to be co-located. However, each node in the cluster has a capacity. When there is no space on the nodes that already has replicas of the blocks assigned to one entry in the locator table, new nodes are chosen. This will lead to blocks assigned to the same table entry stored on a large percentage of the cluster nodes. For example, if K equals 1, it have the same effect of no co-location. Large K means that blocks assigned to different locator table entries might be stored on the same set of nodes. If K becomes much larger than the number of nodes in the cluster, the co-location will hardly be enforced since multiple locator table entries will be using the same set of nodes. The best choice for K will be close to the number of nodes in the cluster.

4.3 Assigning Data Blocks to Spatial Locator Table

After building a spatial locator table and dividing it into multiple entries where each has a specific MBR, we assign each block of the file to one of the entries in the spatial loca-

tor table. The assignment of each block of the file to one of the entries of the spatial locator table indicates where it will be placed in the CoS-HDFS cluster. The new spatial placement policy that replaces the one described in Section 3.1 is as follows. For each block in the input file, we follow these steps:

1. We calculate the MBR for the input block.
2. We check the entries in the locator table that overlaps with the MBR of the input block. We filter these entries to find those that overlap with more than 80%[2].
3. If multiple entries of the locator table are selected, we choose the one with the smallest number of blocks assigned to it. This serves our goal of assigning almost equal number of blocks to each entry in the locator table.
4. We choose the nodes to place the replicas of this block on them.
   (a) If it is the first block to be assigned to this locator table entry, then we rely on the HDFS default placement policy to select nodes to store replicas of the block on them.
   (b) Else, we query the HDFS `NameNode` for the nodes that have replicas of the data blocks that are assigned to the same locator table entry. We store replicas of the block on the least loaded ones.

Our proposed block placement policy for CoS-HDFS addresses the following cases:

- If there is no entry in the spatial locator table that overlaps with the MBR of the input data block with more than 80% (the threshold value), we prepare a list of top m spatial locator entries that overlap with data block. We then continue the placement policy from Step 3.
- After retrieving the nodes where the replicas of blocks assigned to an entry in the locator table, we select the least loaded nodes to store replicas of the new block on them. If the number of nodes with free space is less than the replication factor of CoS-HDFS, we employ one of the following two approaches: (1) we choose the second best locator item and assign the block to it, and hence continue from Step 4; or (2) we use the HDFS default policy to select a new set of nodes to store the replicas of the block.
- We use the default HDFS placement policy to choose the nodes of the cluster on which we place the first 5% of data blocks assigned to any locator item. This strategy allows us to choose enough different cluster nodes to store the replicas of the blocks assigned to the same locator table entry. An alternative approach is to employ HDFS to select 3 nodes[3] to store the replicas of the first block, and use these same nodes to store any subsequent blocks assigned to the same entry in the locator table. Once these nodes become full, we have HDFS select a new set of nodes. The latter approach will create disjoint sets of co-located blocks that are assigned to the same entry of the same locator table. Note that we can roughly estimate the number of blocks assigned to each entry in the locator table by dividing the total number of blocks in an input file by the number of entries in the locator table.

[2]We found 80% a good threshold in our experiments

[3]We are assuming that the default replica number employed by HDFS is 3

Locator MBR	Blocks
L1: MBR[positive x-coordinate]	C1, C2, C3, C4, C10
L2: MBR[negative x-coordinate]	C5, C6, C7, C8, C9

Table 2: Spatial locator table for the example shown in Figure 4.

In the example depicted in Figure 4, applying the proposed block placement policy for the blocks C1 through C8 is straightforward. The MBR of the block C9 overlaps with the MBR of the locator table entry L2 with more than 80% and therefore, we assign it to L2 according to Step 2. C10 fails the condition in Step 2 and therefore, we apply the special condition in which we choose the locator table entries with the most overlapping MBRs (L1 and L2) and then apply Step 3 to choose the locator table entry with the fewer number of blocks assigned to it. In the example, C1-C4 are assigned to L1, and C5-C9 are assigned to L2, therefore we assign C10 to the locator table entry L1. Table 2 shows the final locator table.

In Sections 4.2 and 4.3, we made sure to design our solution to address the challenges discussed in Section 4.1. The method we employed in dividing the spatial locator table into a specific number of entries and assign an MBR to each entry (Section 4.2) guarantees that the number of data blocks assigned to each entry in the table is almost the same. Moreover, for each entry, we make sure that we fill the least loaded nodes first. The latter guarantees that all the CoS-HDFS nodes are almost loaded with equal sizes of data.

5. EXPERIMENTS

We evaluated our system by running experiments on a cluster of 20 Amazon EC2[4] virtual machine instances (nodes) of type m4.large. Each node has 2 cores (2.4GHz), 8 GB of memory, and 500 GB SSD. The cluster used for executing the Hadoop jobs (spatial queries) is the same as the cluster that HDFS (and CoS-HDFS) is deployed. Each node is running Ubuntu 14.04 LTS, and it has Java 1.7.0 and Hadoop 0.20. HDFS and CoS-HDFS are configured to create 3 replicas of each data block. For all the experiments, we constructed the HDFS, CoS-HDFS, and Hadoop clusters to span two AWS regions: Oregon (US west) and Frankfurt (EU). We choose to have 10 nodes per region.

In our experiments, we use the TIGER datasets of spatial features of the US from the US Census Bureau [4] of sizes up to 70 GB. The small size TIGER datasets are available at the SpatialHadoop [12] web page. We prepared the larger TIGER datasets by extracting the downloaded files from [4] and importing them into POSTGIS DB to convert them to CSV files ready to use in our experiments. We use spatial

[4]Available at: http://aws.amazon.com/ec2/

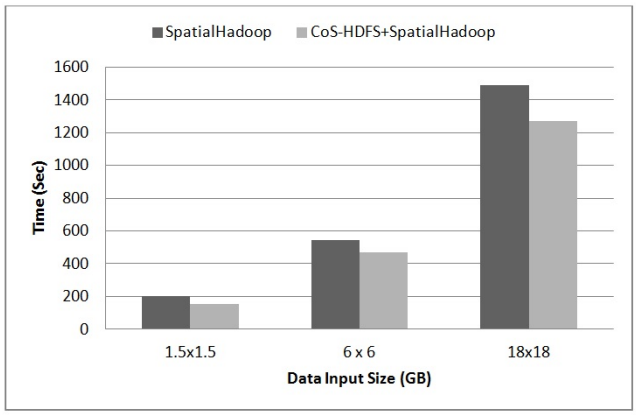

Figure 5: The effectiveness of CoS-HDFS for spatial join queries.

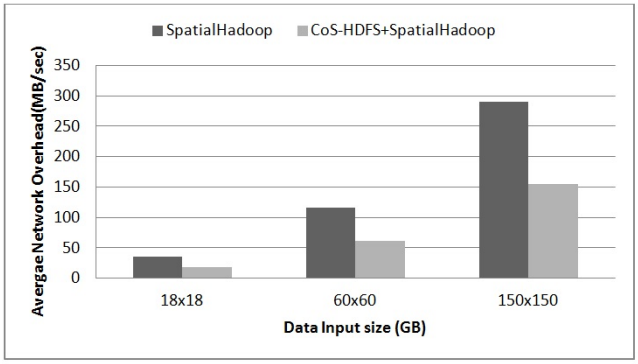

Figure 6: Network traffic incurred by CoS-HDFS vs HDFS for different input file sizes.

join queries to evaluate the performance of CoS-HDFS and our proposed data co-location approach. The MapReduce implementation of spatial join (SJMR) is described in [25] and is available as part of the SpatialHadoop implementation [12].

We compare the execution time of the spatial join queries (namely their SJMR implmentation) and the network traffic (in Bytes/sec) incurred as a result of their execution in a Hadoop cluster. We use Ganglia [2] to monitor the Hadoop cluster and measure the network usage during query execution. Moreover, we use additional Hadoop counters for fine grain measurements of the number of mappers tasks that (1) read all their input data from local nodes; (2) read all their input data from nodes located in the same rack; or (3) read some of their input from nodes in remote racks.

First we study the effectiveness of CoS-HDFS in Sections 5.1 and 5.2 and how it is sensitive to parameters in Section 5.3. We discuss the overhead introduced by CoS-HDFS as compared to HDFS in Section 5.4. Finally, we examine the load balancing of the nodes in the cluster in Section 5.5.

5.1 Effectiveness of the Spatial Co-Location Algorithm

In this section, we demonstrate the effectiveness of CoS-HDFS. The co-location introduced by CoS-HDFS is important for spatial join queries, and therefore we measure the execution time of performing spatial join when we use CoS-HDFS to store the data as compared to when we use HDFS. In both cases, we use SpatialHadoop to index the data and execute the spatial join query. Note that when we use CoS-HDFS, the input files (namely the indexes of these files used by SpatialHadoop) to the spatial join query are co-located. We measure the execution time of the spatial join query in both cases for different input data sizes of the TIGER dataset: 1.5 GB x 1.5 GB, 6 GB x 6 GB, and 18 GB x 18 GB[5]. Figure 5 shows the results of the experiment. The co-location of input data files enhances the execution time of the spatial join queries (up to 15% for the join query that has an input files of sizes 18 GB). The larger the data sizes of the input files, the more performance enhancement that can be achieved. The improvement in the execution time of the spatial join query is due to the saving in the network traffic, which is high because HDFS and CoS-HDFS are deployed on nodes that span two AWS regions.

The join query implemented using the SJMR algorithm and that uses the indexed files created by SpatialHadoop divides its work on multiple map tasks executed on the nodes of the cluster. Each map task reads two blocks each from one of the indexed files that the query joins. Table 3 shows how using CoS-HDFS to store the indexes created by SpatialHadoop increases the number of map tasks that reads all of its input blocks locally. The table highlights the number of mappers that perform all local reads (both input blocks are read locally), part local reads (one of the input blocks is read locally and the other is read from a remote node), and all remote read (both blocks are read from remote nodes. We only show the results when we perform spatial join on data of size 18 GB x 18 GB. The table shows that when we co-locate the input files of the spatial join query by storing them on CoS-HDFS, 70% of the map tasks read their both two input blocks from local node. This percentage is dropped to less than 1% when no co-location is employed. In the latter case, 95% of the map tasks were assigned to nodes that have one of their input blocks, which is the default scheduling policy of Hadoop. Thus we conclude that CoS-HDFS reduces the network traffic by co-locating files stored in it and hence giving the scheduler of Hadoop a higher opportunity to schedule map tasks on nodes in the cluster that have all of their input blocks.

5.2 Effect of the Spatial Co-Location on Network Traffic

To validate our conclusion in Section 5.1, we measure the network traffic incurred because of using CoS-HDFS as compared to HDFS. We run this experiment on a cluster of 20 Amazon EC2 nodes that that are all located in the same region (US West). We report the network usage as reported by Ganglia [2]. We measure the network traffic (measured in MB/sec) incurred by executing spatial join queries for TIGER input data files of sizes: 18 GB x 18 GB, 60 GB x 60 GB, and 150 GB x 150 GB. Figure 6 shows that co-locating input files of the spatial join queries reduces the network traffic by a percentage up to 47% in the case of joining files of 150 GB x 150 GB. This percentage is expected to significantly increase when the nodes of the cluster are divided among two AWS regions.

[5]We mean by an input of size c GB x c GB that the join query has two input files, each has a size of c GB

	All local reads	Part local reads	All remote reads
HDFS	0.09%	95%	4.91%
CoS-HDFS	70.7%	20%	9.3%

Table 3: Comparing the number of mappers performing local reads and remote reads when the data are stored on CoS-HDFS vs HDFS.

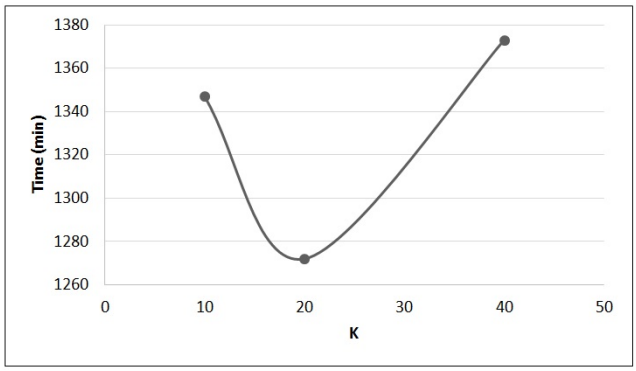

Figure 7: The effect of changing the size of the spatial locator table on the performance of the spatial join queries.

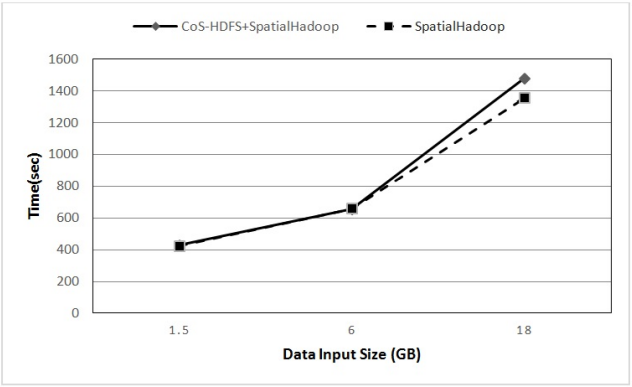

Figure 8: The overhead introduced by CoS-HDFS as compared to HDFS.

5.3 Sensitivity to the Size of the Locator Table

We discuss in Section 4.2 that the number of partitions (entries) in the spatial locator table affect the efficiency of co-location. Therefore, we assume that better co-location leads to lower execution time of spatial join. In this experiment, we change the number of entries in the spatial locator table (K) and measure the execution time of the spatial join query when we use the TIGER dataset of 18 GB x 18 GB as input. We chose k to take the following values 5, 10, 20, 40, and 100. Figure 7 supports our claim that the co-location is sensitive to the number of partitions in the locator table and shows that the best choice of the number of partitions is to have it equal to the number of nodes in the cluster. As a result, we use K as 20 in all the other experiments presented in this section.

5.4 Overhead Introduced by Co-Locating Spatial Data Files

The new block placement policy used by CoS-HDFS introduces overhead during the loading of the data. However, our claim is that it is insignificant overhead that can be tolerated given the benefits discussed in this paper and shown in Sections 5.1 and 5.3. Since the current implementation of CoS-HDFS is tightly coupled with SpatialHadoop to leverage its contributions in the Hadoop stack, we measure the overhead of storing the index created by SpatialHadoop in CoS-HDFS vs storing the same index in HDFS. We measure the time required for indexing and storing the indexes in the distributed file system for files of sizes: 1.5 GB, 6 GB, and 18 GB. Figure 8 shows that the overhead introduced by co-location increases when the size of the data file becomes larger. However, we note that the overhead of co-location is still negligible (a maximum of 1 minute).

5.5 Load Balancing in CoS-HDFS

We discuss in Section 4.1 the importance of load balancing

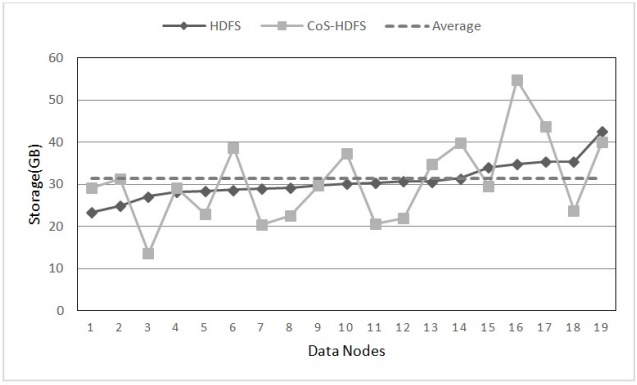

Figure 9: Load balancing in CoS-HDFS as compared to HDFS.

the data stored on nodes of the cluster. This is one of the objectives of the default block placement policy of HDFS. In this experiment, we show that the new block placement policy that we introduce in this paper and that co-locates files has not damaged the balancing of data on the cluster nodes. In this experiment, we compare the size of the data assigned to each node of the cluster when we use the default HDFS block placement policy and the modified CoS-HDFS block placement policy. For the purpose of this experiment, we load a file of size 210 GB on a cluster of 20 nodes (one node is dedicated as a master). The replication factor is 3, and therefore, if perfect load balancing is implemented, each node stores 33 GB. Figure 9 shows that CoS-HDFS introduces higher standard deviation in the amount of loaded data on different nodes. However, we can conclude that this effect is negligible and that CoS-HDFS effectively load-balances the data on all nodes of the cluster.

6. CONCLUSION

It is now common to query gigabytes and terabytes of spatial datasets. Therefore, it is becoming important to store these large datasets on distributed file systems such as HDFS and use distributed computation platforms such as Hadoop to query them. Adding awareness of spatial data characteristics to Hadoop and HDFS can significantly improve the query execution performance. This becomes with significant importance when the data is geo-distributed across multiple data centers.

In this paper, we present CoS-HDFS, an extension to HDFS that is aware of the spatial data files stored in it and that co-locates blocks of files that have contents with overlapping spatial properties on the same nodes. Co-locating spatial file blocks reduces the network traffic exerted while executing spatial queries in the form of MapReduce jobs on these files, and hence reduces the execution time of these queries. This is notably important when the HDFS nodes span multiple data centers, and the network between these data centers is not very fast. We integrated CoS-HDFS with SpatialHadoop to exploit its added spatial awareness to the language, operations, MapReduce, and indexing layers. Our experiments show that spatial join queries executed on data stored in CoS-HDFS incur a reduction of more than 14% of the network traffic as compared to network traffic incurred by the same queries when executed on data stored in HDFS.

7. REFERENCES

[1] Apache Hadoop. Available at: http://hadoop.apache.org/.

[2] Ganglia Monitoring System. Available at: http://ganglia.sourceforge.net/.

[3] HDFS Architecture Guide. Available at: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[4] Tiger Data Set. Available at: http://www.census.gov/geo/www/tiger/.

[5] World Wide Lightning Location Network. Available at: http://wwlln.net/.

[6] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. Hadoop GIS: A high performance spatial data warehousing system over Mapreduce. *Proc. VLDB Endow. (PVLDB)*, 6(11):1009–1020, 2013.

[7] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. OSDI*, 2004.

[8] A. Eldawy, Y. Li, M. F. Mokbel, and R. Janardan. CG_Hadoop: Computational geometry in MapReduce. In *Proc. ACM Int. Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 294–303, 2013.

[9] A. Eldawy and M. F. Mokbel. A demonstration of SpatialHadoop: An efficient Mapreduce framework for spatial data. *Proc. VLDB Endow. (PVLDB)*, 6(12):1230–1233, 2013.

[10] A. Eldawy and M. F. Mokbel. Pigeon: A spatial MapReduce language. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, pages 1242–1245, 2014.

[11] A. Eldawy and M. F. Mokbel. The era of big spatial data. In *Workshops Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, pages 42–49, 2015.

[12] A. Eldawy and M. F. Mokbel. SpatialHadoop: A MapReduce Framework for Spatial Data. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, pages 1352–1363, 2015.

[13] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson. CoHadoop: Flexible data placement and its exploitation in Hadoop. *Proc. VLDB Endow. (PVLDB)*, 4(9):575–585, 2011.

[14] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. pages 29–43, 2003.

[15] M. M. Khan, A. Kneece, I. Sivagnanam, and J. J. Shoultz. A better way to handle GIS data. Available at: http://www.esri.com/news/arcuser/0507/dhec.html.

[16] W. Li, D. C. Zilio, V. S. Batra, M. Subramanian, C. Zuzarte, and I. Narang. Load balancing for multi-tiered database systems through autonomic placement of materialized views. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, pages 102–113, 2006.

[17] S. Nishimura, S. Das, D. Agrawal, and A. El Abbadi. $\mathcal{MD}$-HBase: Design and implementation of an elastic data infrastructure for cloud-scale location services. *Distributed and Parallel Databases (DAPD)*, 31(2):289–319, 2013.

[18] J. Patel et al. Building a scaleable Geo-spatial DBMS: Technology, implementation, and evaluation. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 336–347, 1997.

[19] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 165–178, 2009.

[20] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese. Wanalytics: Geo-distributed analytics for a data intensive world. In *Proc. Conf. on Innnovative Data Systems Research (CIDR)*, pages 1087–1092, 2015.

[21] F. Wang, J. Kong, L. Cooper, T. Pan, T. Kurc, W. Chen, A. Sharma, C. Niedermayr, T. W. Oh, D. Brat, et al. A data model and database for high-resolution pathology analytical image informatics. *Journal of pathology informatics*, 2(1):32, 2011.

[22] F. Wang, J. Kong, J. Gao, L. A. Cooper, T. Kurc, Z. Zhou, D. Adler, C. Vergara-Niedermayr, B. Katigbak, D. J. Brat, et al. A high-performance spatial database based approach for pathology imaging algorithm evaluation. *Journal of pathology informatics*, 4, 2013.

[23] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha. SPANStore: Cost-effective geo-replicated storage spanning multiple cloud services. pages 292–308, 2013.

[24] R. A. Xu and D. B. Zhu. An approach for processing underground spatial-temporal data by cloud computing. *Journal of Automation and Control Engineering*, 1(2), 2013.

[25] S. Zhang, J. Han, Z. Liu, K. Wang, and Z. Xu. SJMR: Parallelizing spatial join with MapReduce on clusters. In *IEEE Int. Conf. on Cluster Computing (CLUSTER)*, pages 1–8, 2009.