ATLAS: A Distributed File System for Spatiotemporal Data

Daniel Rammer, Sangmi Lee Pallickara, and Shrideep Pallickara
Department of Computer Science
Colorado State University
Fort Collins, Colorado, USA
{rammerd,sangmi,shrideep}@cs.colostate.edu

ABSTRACT

A majority of the data generated in several domains is geotagged. These data also have a chronological component associated with them. Pervasive data generation and collection efforts have led to an increase in data volumes. These data hold the potential to unlock valuable insights. To facilitate such knowledge extraction in a timely manner, the underlying file system must satisfy several objectives. In this study, we present Atlas, a distributed file system designed specifically for spatiotemporal data. Atlas includes several capabilities that are suited for performing large-scale analyses: aligning dispersion with data access patterns, load balancing storage, and facilitating interoperation with analytical engines such as Hadoop and Spark. Our empirical benchmarks profile several aspects of Atlas, and demonstrate the suitability of our methodology.

CCS CONCEPTS

• **Information systems** → **Distributed storage**; **Distributed retrieval**; **Specialized information retrieval**; *Geographic information systems*; *Retrieval effectiveness*.

KEYWORDS

spatiotemporal data, file systems, analytics, HDFS

ACM Reference Format:
Daniel Rammer, Sangmi Lee Pallickara, and Shrideep Pallickara. 2019. ATLAS: A Distributed File System for Spatiotemporal Data. In *Proceedings of the IEEE/ACM 12th International Conference on Utility and Cloud Computing (UCC '19), December 2–5, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3344341.3368802

1 INTRODUCTION

A majority of the data generated in several domains are geospatial. These domains include atmospheric sciences, environmental and ecological monitoring, smart communities, geosciences, and commerce. Sources of this data include social media, simulations, and networked observational devices and remote sensing equipment. Furthermore, increases in the number of data sources have occurred alongside the growth in the rate and precisions at which geotagged data are being generated. This has contributed to a dramatic increase in cumulative data volumes. Timely and effective knowledge extraction is predicated on efficient storage, and subsequent processing of this data.

Spatiotemporal data management efforts have primarily focused on encoding rather than the design of file systems. Encoding targets descriptiveness and access to individual features within the encoded data. The encoded data has an implicit lightweight schema associated with it. This can be used to facilitate programmatic access to observations and feature values. Another consideration during encoding is verbosity and the corresponding increase in size over binary data. Compact encoding formats, such as BUFR, focus on ensuring that that the descriptiveness of the data does not substantially add to the data volumes. Several data formats are in use to encode geospatial data. These include netCDF, HDF, BUFR, GeoTIFF, etc. However, when it comes to file systems designed specifically for spatiotemporal data, the efforts have been few and far in between.

Data storage must satisfy a set of key interrelated objectives. In particular, besides archival (for fidelity and reproducibility) the storage subsystem must facilitate discovery and subsequent analyses. Given the data volumes, storage on a single machine is infeasible. A distributed environment introduces challenges relating to load balancing to avoid hotspots and resource underutilizations. Finally, given the number and scope of analytic operations a key requirement for modern storage systems is how effectively they support completion times for analytic jobs launched over the data. In summary, storage schemes must manage dispersion of data items, coordination for servicing discovery and query evaluations, and facilitate effective support for analysis.

There are two dominant approaches to manage spatiotemporal data, both of which adversely impact completion times for analytic operations. The first approach groups all data from a geographical extent (typically by computing a geohash) and stores them on the same machine. Approaches based on distributed hash tables leverage this approach. This grouping strategy results in reduced concurrency for analytic operations. However, given the data volumes, analyses must be concurrent and sequential processing of data from a geographical extent is infeasible. The second approach, disperses data items without accounting for spatial properties. A key focus of this approach is to ensure load balancing across the set of available nodes. This results in substantial data movements during analytical operations and prolonged completion times for analytic tasks.

We posit that data accesses, and the I/O that this entails, is the key to ensuring fast, effective analytics. Our hypothesis is predicated on the characteristics of the I/O subsystem and the steep speed differential of the memory hierarchy. While storage capacities increase down the memory hierarchy (from the registers, caches, memory, and disk), there is a corresponding increase in access times and reduced throughputs during transfers. Given that the CPU is 6-7 orders of magnitude faster than disk accesses, a consequence is that analytic tasks are I/O bound. **The crux of this paper is the design of file system for spatiotemporal data that alleviates I/O overheads during processing tasks triggered by analytic tasks.**

1.1 Challenges

Designing a distributed file system that enables effective analytics over geospatial data has several challenges. These include:

- *Impedance mismatch in how the data is stored and how they must be processed.* Dispersion schemes that do not effectively account for the spatiotemporal characteristics of the data result in data movements during analytics that prolong completion times.
- *Support for concurrent processing of data from a geospatial extent.* To ensure effective completion times, analytical operations launched over a particular spatial extent must be performed concurrently with attenuated network I/O.
- *I/O amplification:* Disk blocks are the units of data transfer between the memory subsystem and the disk. This means that even if the data item that is request is a byte of data, what is read is a block. Ineffective data collocations amplify the volume of the disk I/O that need to be performed.
- *Operation in shared clusters:* Data storage, retrievals, and analytic operations occur within shared clusters. Given that analytic operations are I/O bound, ineffective dispersion schemes result in I/O amplifications. Given that these operations take place in shared clusters, these are also subject to contention and reduce throughputs that such interference introduces.
- *Interoperation with analytical engines:* Several analytical engines, such as Hadoop and Spark, are now available with effective support for orchestration of computational workloads from analytic workflows. These engines also include support for several model fitting algorithms based on statistical and machine learning. File systems that do not effectively interoperate with analytical engines represent a siloing of the data space that precludes the ability to leverage libraries that have developed as part of these engines.

1.2 Research Questions

The broader research question that guides this study is how we can design a distributed file system specifically for spatiotemporal data that supports effective analyses over voluminous data. Specific research questions that we explore include:

RQ-1: *How can we effectively account for spatiotemporal characteristics of the data?*
RQ-2: *How can we support segmentation of the data?*
RQ-3: *How can we facilitate interoperation with analytical engines?*
RQ-4: *How can we reconcile the competing pulls of dispersion and collocation to facilitate efficient concurrent analyses?*

1.3 Approach Summary

We extract spatiotemporal markers from observations. The spatial components may be expressed as either standalone <lat, long >coordinates or as bounding boxes. We leverage the geohash algorithm to organize observations as part of files. In particular, the geohash algorithm converts spatial coordinates into a configurable precision string representation of a geographical extent; the greater the precision of (or the longer) the string, the smaller the geographical bounding box represented by the geohash.

We leverage geohashes to organize observations in files. The file that an observation belongs to is deterministically calculated using the geohash algorithm. All data within a particular spatial extent represented by the geohash is organized within a file. Each file is distributed over the cluster of available machines. This allows the file, either in its entirety or portions thereof, to be processed concurrently as part of analytics jobs. We place no limits on the size of the files. Because the files are dispersed over the collection of machine, and there no size limits that are imposed, we are able to cope with diversity in the available observations.

Each file in Atlas comprises chunks. Each chunk is fully resident on a single machine, but chunks that comprise the file are distributed over the available machine. Chunks have a fixed size, and as new observations become available a new chunk is added to the file. Newly ingested observations are added to chunks. The system maintains metadata about the number and order of chunks comprising a file.

Atlas satisfies two key objectives during the placement of chunks within the cluster: redundancy and load balancing. We accomplish redundancy via replication. Once the replication level for a file has been specified, each chunk within the file is replicated on different machines (to avoid correlated failures) until the desired replication level has been reached. The chunks comprising a file are dispersed over the cluster such that loads are evenly distributed. In particular, our dispersion scheme ensures that multiple, successive chunks are stored on different machines and also ensures that storage imbalances do not occur within the cluster. Each machine takes on storage loads that are commensurate with their capabilities, and disk utilization skews (based on percentage utilizations) do not occur within the cluster.

We leverage segmentation within chunks to group proximate observations; this segmentation allows us to facilitate block transfers of observations. Such block transfers reduce both the number and intensity of I/O operations within the system. Each chunk segments data into smaller, contiguous spatial extents within the larger geographical scope encapsulated by the file. Successive chunks include observations that are temporally proximate. This scheme allows chunks within a file to preserve spatiotemporal proximity and counteract inefficient spatiotemporal data access patterns.

The organization and dispersion of observations within an Atlas file aligns well with how data are processed during analytic jobs. In particular, tasks have data locality and transfers are performed in blocks based on spatiotemporal characteristics. This counteracts I/O amplifications and the ensuing prolonged completion times. The Atlas metadata management scheme allows fast identification of relevant portions of the data space; queries may specify spatiotemporal scopes of interest and the files and chunks representing this interest are identified in real-time.

To support interoperation with analytical engines, we make Atlas HDFS (Hadoop Distributed File System) compliant. Several analytical engines, such as Hadoop, Spark, and TensorFlow, use HDFS as the source of their input data. HDFS is a very mature specification, and several robust implementations of HDFS-compliant storage systems now exist. Making Atlas HDFS compliant allows us to seamlessly interoperate with analytical engines during spatiotemporal data analyses. Furthermore, because Atlas is tailored towards minimizing I/O amplifications during processing, the spatiotemporal analyses expressed using these analytical engines complete faster.

1.4 Paper Contributions

The Atlas file system supports high-throughput data storage, retrieval and analytic operations for geospatial data using analytical engines. In particular, our contributions include:

- A file system that accounts for the spatiotemporal characteristics of the data. Atlas performs metadata management and organizes the spatiotemporal data so that discovery and seek operations within a file are fast and amenable to concurrent analysis using analytical engines.
- To our knowledge, Atlas is the first spatiotemporal file system that is HDFS-compliant.
- Our data organization scheme and HDFS-compliance facilitate concurrent analytics of data. Our methodology also allows us to launch processing tasks on several spatial extents concurrently. In particular, Atlas allows users to effectively isolate/identify files, and portions thereof, that need to be accessed.

2 METHODOLOGY

Figure 1 depicts the Atlas systems architecture. To ensure systems performance at scale, our design separates control and data planes by partitioning machines within cluster into namenodes and datanodes. Namenodes support discovery and manage namespace information relating to file placements, replication, and spatiotemporal indices. Datanodes are responsible for data storage and retrieval. This functional separation simplifies fault tolerance and scalability considerations within Atlas.

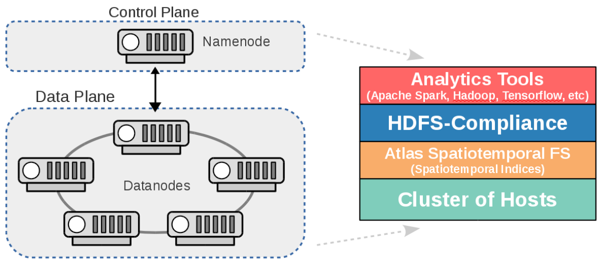


Figure 1: The Atlas architecture highlighting the separation of control and data planes and the hierarchy of provided protocols and application interfaces within Atlas.

Our methodology to design an efficient distributed file system for spatiotemporal data encompasses the following:

- *Counteracting inefficient spatiotemporal data access patterns:* Retrieval of data within a specific spatiotemporal scope from a set of files often requires non-sequential disk reads that adversely impact analytics performance. These inefficiencies are a product of data reporting techniques, where data seldom aligns with spatiotemporal attributes resulting in intertwined spatiotemporal scopes within files. Atlas distributes and dynamically rearranges data to align with spatiotemporal scopes, providing targeted, sequential disk reads during retrieval. **[RQ-1, RQ-2]**
- *Reconciling the competing pulls of dispersion and locality:* Distributed analytics tasks tend to perform optimally with a specific data distribution policy. However, the optimal policy varies across analytic tasks and different data distribution policies often contradict each other – no scheme fits all. Data dispersion and locality are the two extremes, distributing data evenly over the cluster and confining data to a small subset of hosts respectively. By chunking the dataspace and replicating data based on spatiotemporal scopes Atlas provides a unique data distribution policy, where analytics may leverage both dispersion and locality concurrently. **[RQ-1, RQ-4]**
- *Support for effective reduction operations:* Spatiotemporal analytics often require identification and retrieval of a specific data scope. Traditional solutions require iterating over each observation, and filtering out those that do not satisfy the query predicates. Atlas maintains a spatiotemporal index over the underlying data, facilitating efficient identification and retrieval of spatiotemporal subspaces. **[RQ-1]**
- *Integrating with analytics tools:* Distributed analytics tools are a key component in many existing workflows. Several analytical engines interoperate with HDFS using it as the source and destination of analytic jobs. Therefore, we have

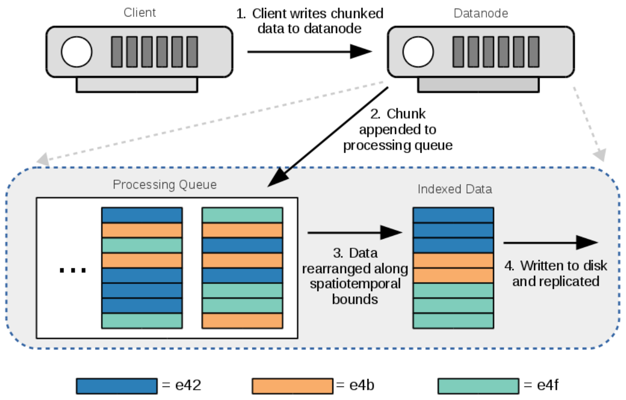

Client
1. Client writes chunked data to datanode
Datanode
2. Chunk appended to processing queue
Processing Queue
...
Indexed Data
3. Data rearranged along spatiotemporal bounds
4. Written to disk and replicated
= e42
= e4b
= e4f

Figure 2: Block indexing within Atlas datanodes. Block observations are rearranged to align with spatiotemporal characteristics improving search and retrieval performance.

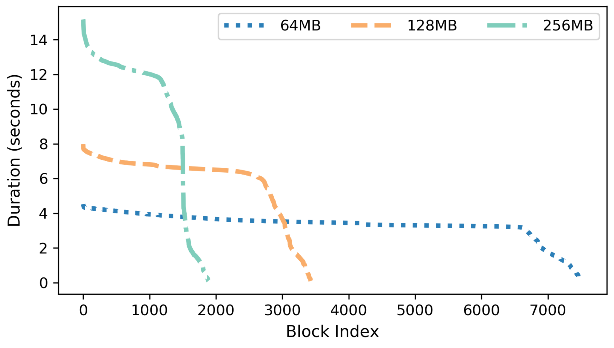


Figure 3: Block indexing durations (sorted longest to shortest) over 1TB of data using block sizes of 64MB, 128MB, and 256MB. Regardless of the block size, cumulative completion times do not incur significant variance.

incorporated HDFS-compliance within Atlas; since it uses the same communication protocols as canonical HDFS it can be used anywhere that HDFS is used. This enables seamless integration with applications supporting HDFS. **[RQ-1, RQ-3]**

2.1 Counteracting Inefficient Spatiotemporal Data Access Patterns **[RQ-1, RQ-2]**

Sequential file reads offer the best performance, in terms of throughput and transfer times, during data retrieval. However, this process lacks support for spatiotemporal filtering, where identification and retrieval of a specific spatiotemporal scope requires iteration over the entire dataset and filtering of observations individually. This introduces a number of inefficiencies including computational overhead and unnecessary disk and network I/O.

The main challenge in identification and retrieval of specific spatiotemporal scopes is that data reporting seldom aligns with spatiotemporal attributes. Rather, files tend to contain a diverse collection of spatiotemporal scopes. This contributes to two inefficiencies. First, relying on filtering at the observation granularity incurs unnecessary computational overhead, disk I/O, and network I/O. Second, non-sequential disk reads entail significant disk head movements, negatively impacting read performance.

A critical component within Atlas is the geohash algorithm, which converts 2-dimensional coordinates to a string representation. The original implementation is base-32, but we have designed a base-16 variant, producing hexadecimal strings. The algorithm produces a bit string, in iterations of 4 bits, where each iteration produces another output character. The basis is splitting the maximum and minimum bounds, alternating between X and Y coordinates, and appending a 0 or 1 to the bit sequence if the indexed value is in the upper or lower half, respectively. For example, the original Y minimum and maximum values are -90 and 90 in the Cartesian coordinate system. We begin by splitting the range in half, the lower half being -90 to 0 and the upper 0 to 90. If we are indexing a coordinate with a Y value of 45 we append a 1 to the bit string. The next iteration of Y computation is performed with minimum and maximum bounds of 0 and 90 respectively.

Atlas relies on *chunking* the dataset to aid in efficient data access. This is performed by partitioning input data files into multiple blocks which may then be distributed and replicated over the cluster. As blocks are written to the system, observations are dynamically rearranged to align with spatiotemporal boundaries. We refer to this segmentation as producing *striped sets*. The process begins by computing the geohashes for each observation, an operation that supports points, lines, and polygons. Next, observations are rearranged so that spatial scopes are contiguous, and temporally adjacent and increasing, within the block. This procedure is performed in-memory to reduce computational and I/O overhead.

Rearranging observations within blocks to align with spatiotemporal boundaries simplifies identification and retrieval of dataspace subsets. The shuffling allows for fast identification of a spatiotemporal dataspace, where positional indices (i.e., data offsets and length) for each block correspond to a spatiotemporal range. Additionally, data retrievals can be performed with sequential disk reads, mitigating the performance degradation caused by excessive disk head movements. The process of chunking and indexing the dataspace is outlined in Figure 2.

Micro-Benchmark Profiling Indexing: In Figure 3 we profile the block indexing performance when writing 1TB of data into an Atlas cluster of 50 hosts. We performed three separate experiments with three different block sizes, namely 64MB, 128MB, and 256MB. Block index durations were sorted from longest to shortest and plotted along the x-axis. We see that index duration is consistently efficient, on the order of a few seconds. Moreover, data insertions in Atlas are asynchronous, where completing the indexing over one block does not delay insertion of the next. Additionally,

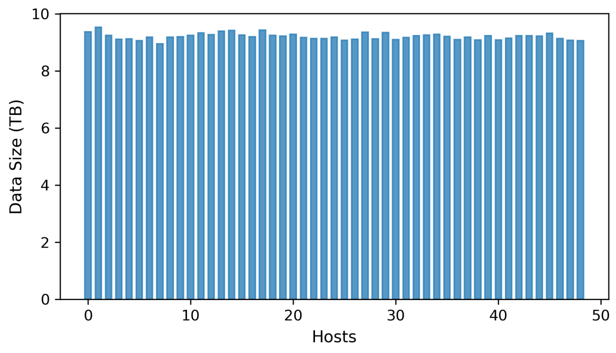


Figure 4: Data managed by each of 49 datanodes within the Atlas cluster after inserting 150TB of data. This demonstrates Atlas' ability to load-balance data while ensuring dispersion and collocality.

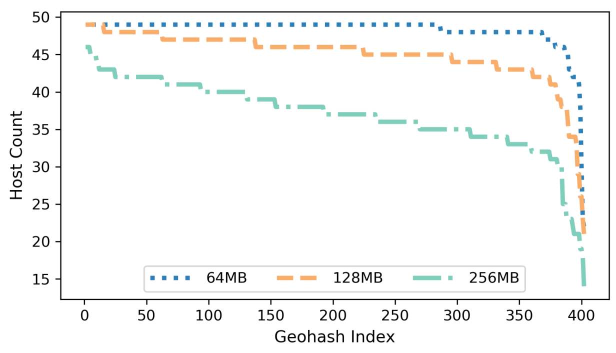


Figure 5: Depiction of the number of hosts that each unique geohash is present on after 1TB of data is inserted in a 50 node Atlas cluster. Highlights the effectiveness of Atlas' data dispersion policies.

total data indexing time does not significantly differ across the three experiments.

2.2 Reconciling the Competing Pulls of Dispersion and Locality [RQ-1, RQ-4]

Many analytics tasks benefit from specific data distribution policies – a one size fits all solution does not exist. Data dispersion and locality are the two extremes. Dispersion focuses on distributing data evenly across each machine, but often triggers excessive data movements within the shared cluster. Alternatively, locality supports processing an entire dataspace on a single host, which is useful when the algorithm requires an omniscient view (ex. data joins).

Supporting a data distribution technique that performs well in myriad analytic flavors is difficult. This problem is compounded by support for spatiotemporal data, where analytics are often performed on dataspace subsets, meaning control of data distribution must be performed at a much finer granularity. Atlas addresses two key challenges in providing efficient execution over diverse analytics. (1) The spatiotemporal scope of blocks are not known until they are indexed. This means that data distribution must be performed in multiple stages, comprising pre-indexing and post-indexing. (2) Balancing the trade-off between distributing data to provide efficient analytics and load-balancing data storage. Skewed data distributions result in host hotspots, mitigating many of the advantages of distributed storage and processing.

The Atlas distribution algorithm aims to support both data dispersion and locality simultaneously. Clients begin by writing data, in partitioned blocks, to the cluster. Initial block placement is determined by the namenode, which analyzes storage patterns to determine hosts with lower disk usage. Once written, the block indexing process is performed as described in Section 2.1, resulting in computation of the specific spatiotemporal scopes within the block.

Data is asynchronously replicated to multiple hosts (the replication level is configurable with a default of 3). The replication policy for data within a spatiotemporal subspace favors hosts that have stored similar data. This process is assisted by the Atlas namenode, which contains a spatiotemporal index over the data (as discussed further in Section 2.3). Thus, the initial data write is processed on a pseudo-random host, and replicas are written to hosts containing data from a similar spatiotemporal extant.

Atlas' data distribution policy results in data for each spatiotemporal scope that is thinly distributed over the cluster hosts, where replicas are concentrated on a small subset of hosts. This provides the ability to access spatiotemporal subsets with data dispersion and/or locality principles. This facilitates scheduling analytics tasks with host data locality, leveraging data distributions that are aligned with the particular task.

Micro-Benchmark Profiling Load Balancing of Storage: In Figure 4 we profile data distribution across Atlas hosts. For this experiment we load 150TB of data into the cluster using a 128MB block size. Each bar on the x-axis corresponds to a datanode and the y-axis displays the cumulative data storage size at that particular host. Overall, the experiment highlights Atlas' ability to load-balance data storage while preserving the trade-off between data dispersion and locality for individual spatiotemporal scopes, comparing and contrasting different regions and/or durations.

Micro-Benchmark Profiling Geohash Dispersion: Figure 5 highlights the effectiveness of Atlas' thin geohash distributed over cluster hosts. For this experiment we loaded 1TB of data into a cluster of 50 machines (1 namenode and 49 datanodes). We identified the collection of unique indexed geohashes, sorted the number of hosts each geohash was distributed over, and plotted the results across the x-axis. We performed this experiment for block sizes of 64MB, 128MB, and 256MB. We see that smaller block sizes provide a wider distribution, where each unique geohash is distributed over more hosts. However, even geohashes in the 88th percentile for 256MB blocks are dispersed across 35 hosts, enabling efficient parallel processing for the particular geohash.

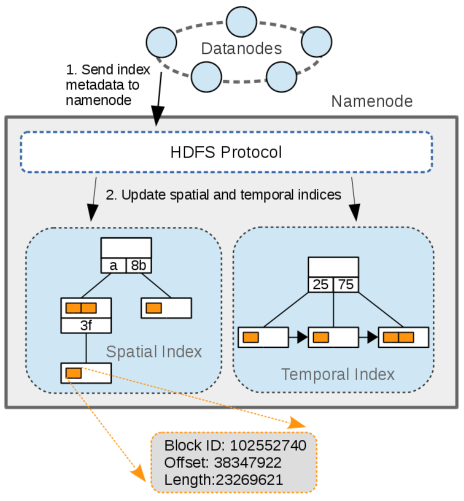


Figure 6: Atlas' namenode architecture and maintenance of spatiotemporal indices. The Atlas spatial index uses a radix tree over geohashes and the temporal index is maintained as a B+-Tree with block timestamps.

2.3 Support for Effective Reduction Operations [RQ-1]

Data reduction operations are important for spatiotemporal analytics to identify and retrieve subsets of the dataspace. This is a primary operation for spatiotemporal analytics tasks, which often evaluate over a particular spatial and temporal range.

To provide a platform for efficient spatiotemporal analytics Atlas maintains an index over data blocks, enabling fast identification of spatiotemporal data subsets. Our methodology includes support for processing index updates efficiently, and enables for robust, efficient query operations. This is important because inefficiencies in index construction may halt subsequent analytics. We have also incorporated a robust collection of operations that increase the utility of indices while ensuring efficient query evaluation that is paramount to data retrieval throughput and latency.

Atlas provides efficient indexing by storing a collection of data structures at the namenode to facilitate both spatial and temporal queries. Index metadata is computed at the datanodes, as reported in Section 2.1, and relayed to the namenode. An overview of this process is outlined in Figure 6.

The Atlas spatial index is a radix tree over block geohashes. Each node in the tree contains a list of block ID, offset, and length tuples representing data belonging to the corresponding nodes geohash. Queries over the spatial index return a superset of requested data to ensure complete coverage. For example, a data query for the geohash "8bce" will return data that is indexed as "8b", because the later may contain data bounded by the former. The use of this structure ensures simple identification of dataspace subsets satisfying the geohash predicate.

Table 1: Spatial and temporal filtering operands alongside units that may be embedded within Atlas' HDFS-compliant request URLs.

Filter Type *(Unit)*	Operand
Spatial *(Geohash)*	Equality *(=)* Non-Equality *(!=)*
Temporal *(Timestamp)*	Less Than *(<)* Less Than or Equal *(<=)* Greater Than *(>)* Greater Than or Equal *(>=)*

A temporal index is maintained using a B+-Tree with start and end timestamp ranges for each block. B+-Trees are self-balancing, space-efficient data structures that are optimized for efficient range queries. This allows Atlas to quickly identify blocks that contain data for a specific temporal range.

2.4 Integration with Analytics Tools [RQ-1, RQ-3]

There is a rich ecosystem of analytical engines that support efficient, distributed analytics. Interfacing with these is paramount for integrating Atlas into **existing** workflows. The HDFS file system interface is a mature, well-defined specification that is leveraged by several analytical engines such as Hadoop, Spark, TensorFlow, Flink, etc. for hosting datasets (inputs) or writing results (outputs).

As part of this study, we have incorporated HDFS compliance into Atlas. Atlas incorporates communication protocols, message exchanges, and discovery operations that are required (and supported) by canonical HDFS. This allows seamless interfacing with analytics tools. To provide this spatiotemporal support within the bounds of HDFS protocol we have implemented three novel extensions; (1) Extension of HDFS' *storage policy* framework, (2) embedding spatiotemporal queries with HDFS URLs, and (3) extending the block ID to encode spatial filters.

Canonical HDFS enables each file and directory to be tagged with a *storage policy*. This tag provides system hints as to where data should be available within the memory hierarchy, for example: disk, RAMdisk, RAM, etc. Atlas extends this tag to include the data format for that particular directory. Specifically, it explains where to find that spatial and temporal attributes within each observation. This allows Atlas to support a variety of file formats including CSV, WKT/WKB, NetCDF, and HDF5.

Atlas file URLs may contain an embedded query, similar to the HTTP protocol, which can include spatial (geohash) and / or temporal (numeric) filtering criteria. Queries may be applied to directories, returning blocks from children that satisfy the criteria, or files. An example of a URL with an embedded query is "hdfs://noaa-imputed/data0/lattice-126.csv+geohash=8bc&timestamp>1419897600". In this example we are requesting the file "/noaa-imputed/data0/lattice-126.csv" and filtering geohash equal to "8bc" and timestamp greater than "1419897600". Spatial and temporal filtering

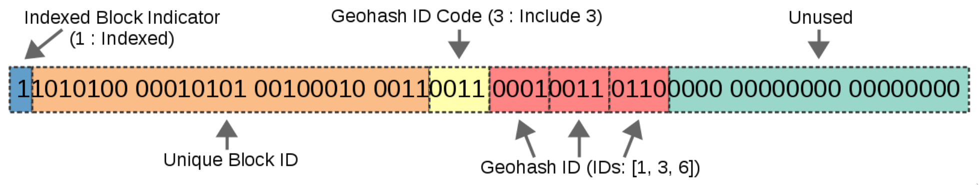


Figure 7: A sample HDFS block ID broken down according to the Atlas spatial encoding scheme. A request for the block ID reads data from the specified block for the 1st, 3rd, and 6th geohashes.

operands that are supported by Atlas are presented in Table 1. Evaluation of queries is performed at the namenode in the "getBlockLocations" RPC call, transparently returning all block IDs (and metadata) that satisfy the query.

We also extended the HDFS block ID to encode spatial scopes. This is necessary to provide efficient data retrieval, allowing sequential disk reads to process any spatial subset of a block. For example, a query which requests non-consecutive spatial partitions of a stripped set is able to be performed in a single request with our solution. Without it, a request for each spatial partition is necessary, incurring excessive overheads due to disk head movements. HDFS block ID's are 64-bit unique values. In Table 2 we have outlined our partitioning of these bits to encode spatial scopes. Our encoding scheme relies on the fact that Atlas can include 16 unique geohashes within each block (hence using a base 16 variant of the geohash algorithm). Figure 7 provides an example geohash and the decoded spatial scope.

Introduction of these novel extensions into the HDFS protocol enables Atlas to provide efficient, robust spatiotemporal query support. Additionally, they allow seamless integration of Atlas into existing workflows.

3 EMPIRICAL BENCHMARKS AND EVALUATION

Here, we report on systems benchmarks that we have designed to profile several aspects of our methodology to assess:

- Efficient identification and retrieval of spatiotemporal subspaces. The Atlas architecture is designed to provide efficient filtering over the underlying datasets, a common requirement for many spatiotemporal analytic tasks. [**RQ-1, RQ-3, RQ-4**]
- Reductions in disk and network I/O during spatiotemporal analytics. Distributed analytics frameworks are often deployed in shared environments that are subject to contention and reduced throughputs. Effective utilization of system resources facilitates overall performance improvements, where disk and network I/O are subject to contention. [**RQ-1, RQ-3, RQ-4**]

3.1 Experimental Setup

Atlas systems benchmarks were performed on a cluster of 50 HP-DL60-G9-E5-2620v4 machines running Fedora 29. These are equipped with an Intel Xeon E5-2620 (8 cores, 16 hyperthreads, 2.10GHz) and 64GB RAM. The cluster is connected with a 1Gb/s ethernet switch. Storage layer applications (i.e., Atlas and HDFS v2.8.5) use 1 machine for the namenode and the other 49 for datanodes. Similarly, our Apache Spark v2.4.3 deployment uses 1 machine for the Master and the other 49 as Workers.

Table 2: The bit sequence index layout of Atlas' HDFS block ID extensions to support embedding of spatial queries.

Bit Indices	Description	Key	Value
1	Index Flag	0 1	Non-indexed Indexed
2 - 27	Unique ID	-	-
28 - 31	Geohash ID code	0 - 8 9 - 15	Include 0 - 8 Exclude 1 - 7
32 - 64	4 bit Geohash ID's	-	-

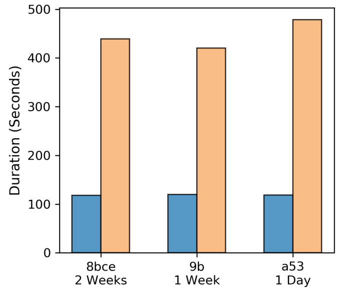
We have performed benchmarks over two real-world datasets to highlight performance and applicability in diverse scenarios. The NOAA (National Oceanic and Atmospheric Administration) dataset contains data reported by 1.3 million globally dispersed vantage points every 6 hours. It contains point spatiotemporal data, meaning each observation is tagged with a latitude, longitude, and timestamp. A few examples of the 56 uniformly reported features are surface temperature, wind speed, humidity, pressure, and precipitation. We have chosen to analyze 2014 data, reporting at just under 1TB. The EPA (US Environmental Protection Agency) dataset contains point data reported by 275 million US-based vantage points every 4 hours. This dataset reports at just under 200GB and contains features encompassing airborne particulates, meteorological information, and toxics (such as lead and ozone precursors).

3.2 Identification and Retrieval of Spatiotemporal Subspaces [**RQ-1, RQ-3, RQ-4**]

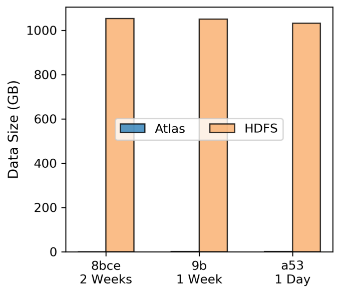
In this experiment, we performed spatiotemporal filtering operations over the EPA dataset using Apache Spark. We provide analyses over a variety of spatiotemporal scopes. Spatial filtering is performed using geohashes and the corresponding latitude and longitude boundaries, temporal filtering evaluates over the bounding timestamps, and spatiotemporal operations leverage a combination of the aforementioned criteria. Within Spark, we read the base dataset into a DataFrame, filter based on the necessary criteria, and perform a count operation over the results. Counting the resulting observations

Table 3: Performance improvements comparing Atlas with canonical HDFS for spatiotemporal subspace identification and retrieval. Filtering duration is improved by up to 7.9x, 2.8x, and 5.2x for spatial, temporal, and spatiotemporal criteria respectively. Disk and network I/O are consistently reduced by 3 orders of magnitude.

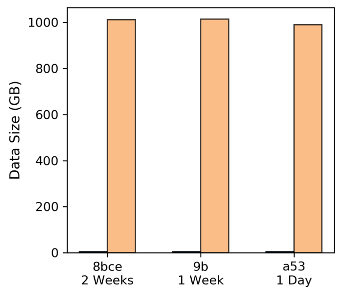
Filter Type *(Unit)*	Filter	Duration		Disk I/O		Network I/O	
		Atlas	HDFS	Atlas	HDFS	Atlas	HDFS
Spatial *(Geohash)*	8e80	9.97s	1:17.3s	126.4MB	193.9GB	320MB	188.9GB
	a76b	10.9s	1:15.3s	130.7MB	194.7GB	316.4MB	188.2GB
	b2296	10.9s	1:26.5s	130.7MB	195.3GB	317.0MB	188.5GB
Temporal *(Duration)*	2 Weeks	28.6s	1:20.7s	15.8GB	195.0GB	17.4GB	186.4GB
	1 Week	28.5s	1:11.9s	15.7GB	195.0GB	17.2GB	188.6GB
	1 Day	29.6s	1:17.4s	17.8GB	194.9GB	19.2GB	188.7GB
Spatiotemporal *(Geohash - Duration)*	8bce - 2 Weeks	15.9ss	1:23.6s	344MB	195.2GB	582.4MB	187.6GB
	9b - 1 Week	18.9s	1:14.2s	568.9MB	193.8GB	819.6MB	187.0GB
	a53 - 1 Day	19.2s	1:21.3s	1.2GB	195.8GB	1.5GB	188.0GB



(a) Duration of spatiotemporal histogram analytics. Atlas reduces by up to 4x compared to canonical HDFS.



(b) Disk I/O is consistently reduced by 3 orders of magnitude.



(c) Network I/O is measurable in single GBs instead of TBs after a 3 order of magnitude reduction.

Figure 8: Duration, disk I/O, and network I/O incurred during histogram evaluation over a single dataset feature comparing Atlas with HDFS.

is necessary because Spark's lazy evaluation of DataFrames, where operations are only performed when necessary.

In Table 3 we present the results of our filtering experiments; reporting durations, disk I/O, and network I/O. We see that Atlas consistently outperforms canonical HDFS. Filtering duration is reduced by up to 7.9x, 2.8x, and 5.2x for spatial, temporal, and spatiotemporal filtering criteria respectively. Both disk I/O and network I/O are **reduced by up to 3 orders of magnitude**. The variance in duration between spatial, temporal, and spatiotemporal filtering is caused by the amount of data that needs to be read from disk. We observed that Atlas temporal filtering provides an order of magnitude better disk I/O performance (for similar filtered data sizes). This is because Atlas' spatial indices provide finer querying granularity over blocks.

These improvements can be attributed to (1) spatiotemporal indices within Atlas providing targeted data access and (2) support for sequential disk reads that circumvent the overheads from disk head movements. Furthermore, HDFS-compliance allows Atlas to seamlessly interface with the Apache Spark framework, facilitating adoption into existing workflows.

3.3 Spatiotemporal Analytics [RQ-1, RQ-3, RQ-4]

To highlight the effectiveness of spatiotemporal analytics using Atlas we have chosen to construct a histogram over a single feature within a spatiotemporal subspace. The histogram algorithm is a common analytics operation to estimate the distribution of feature values. We experimented using the NOAA dataset, specifically targeting the surface temperature feature and evaluating under a variety of spatiotemporal scopes, presenting geographically diverse regions with varying durations.

In Figure 8 we present our results comparing Atlas with canonical HDFS. Specifically Figures 8a, 8b, and 8c provide the analytics duration, disk I/O, and network I/O incurred during evaluation respectively. We see that **Atlas facilitates a 4x reduction in duration when compared with canonical HDFS**. Additionally, disk I/O and network I/O are continually reduced by 3 orders of magnitude, making results measurable in single GBs instead of TBs.

These reductions are directly attributable to Atlas' spatiotemporal optimizations in identifying spatiotemporal subspaces and providing efficient data access patterns. Additionally, the analytics task benefits from Atlas' data distribution policies, facilitating parallel processing enabled by the effective dispersion of data throughout the cluster.

4 RELATED WORK

A variety of spatial indexing data structures have been proposed. Quadtrees [8], R-tree's [12], and many R-tree variants [4, 14, 22] build trees where each node is responsible for a spatial subset of its parent. Leaf nodes contain buckets of data and when the bucket's reach capacity the node is split, adding a layer to the tree. Vornoi diagrams [3] partition a plane into multiple regions (Voronoi cells) based on distances to predefined points (seeds). Voronoi diagrams provide optimizations for many spatial operations. All of these algorithms focus on indexing spatial data at a fine granularity. Atlas' approach of constructing a radix tree over observational geohashes produces a more concise index, while providing the finest granularity over queries allowed within the system.

Efforts to distribute spatial indices include the P2P R-tree [19], distributed quad-tree [25], and spatial P2P [15]. These algorithms provide distributed spatial indexing, but lack optimizations in data distributed policies and data access patterns. Additionally, interfacing with analytics tools is increasingly difficult; often requiring significant implementation. Atlas facilitates efficient spatiotemporal access by provide dispersed and collocated data distribution and optimizes data access patterns by rearranging data along spatiotemporal bounds.

Scientific communities have leveraged P2P Grids [9, 10] and problem-specific storage solutions [24]; our approach is broadly applicable to any spatiotemporal datasets that need to be processed concurrently.

The Hadoop Distributed File System [23] was designed to combat the inefficiencies of the Google File System [11]. It partitions files into many blocks and distributes and replicates them among a set of datanodes. This facilitates efficient distributed analytics and provides fault-tolerance. HBase [27] and Hive [26] provide a relational interface and data warehousing solution over Hadoop respectively. These simplify interfacing and integration of Hadoop. While these solutions are generally performant, they lack optimization for specific workloads, particularly spatiotemporal analytics. Atlas leverages algorithms designed within these frameworks for efficient distributed analytics and extends them for spatiotemporal support, specifically targeting efficient data distribution and storage for spatiotemporal subspace identification and retrieval.

A number of projects have been introduced targeting spatial functionality within the MapReduce framework. Individual spatial functions have been proposed addressing range queries [18], K-nearest neighbors [31], and all-nearest neighbors [28]. Join operations, focused on spatial parameters and K-nearest neighbors, were evaluated in [32], [29], and [17]. Full spatial functionality suites have been proposed using Voronoi-based spatial MapReduce operations [2], distributed spatial indices [5], and at the Hadoop language interface layer in Pigeon [6], an extension of Hadoop's PigLatin [21] language. These efforts provide efficient implementations of spatial functionality using MapReduce but lack optimizations in the data storage layer, alternatively targeting the analysis phase. Atlas provides spatiotemporal performance improvements by targeting inefficiencies in the storage layer, providing a new platform for optimized spatiotemporal functionality.

Many extensions to the Hadoop framework aim to integrate native spatial support. Parallel-secondo [16] is a parallel spatial DBMS using Hadoop for spatial support in the analytics layer. VegaGiStore [33], HadoopGIS [1], and SpatialHadoop [7] are extensions to the Hadoop source-code to support spatial indexing and analytics within the framework. Extensions over H-Base include MD-Hbase [20], H-Grid [13], and HBase-Spatial [30]. These systems lack support to tailor data placement based on spatiotemporal similarities. Rather, they maintain two-tiered spatial indices over the data. Atlas facilitates efficient analytics by optimizing the data distribution policies and data access patterns, producing sequential disk reads.

5 CONCLUSIONS AND FUTURE WORK

This study describes our methodology for enabling native spatiotemporal support within a distributed file system. Atlas targets every layer of processing to optimize for identification and retrieval of spatiotemporal subspaces. Datasets are chunked and rearranged to align with spatiotemporal data access patterns, enabling sequential reads on operations over a specific spatiotemporal range. We manage the competing pulls of dispersion and locality during data distribution to facilitate effective support over a variety of analytics tasks. Our integration of HDFS' communication protocols allows seamless integration of Atlas with several analytics engines, while allowing for adoption into existing workflows.

- **[RQ-1]:** We account specifically for spatiotemporal data characteristics within every layer of our solution. Atlas partitions and rearranges data to enable spatiotemporal aligned disk accesses. Balancing data distribution using dispersion and locality over the cluster facilitates effective spatiotemporal analytics for myriad scenarios, and the combination of spatiotemporal indices and HDFS extensions allows for efficient identification and retrieval of spatiotemporal subspaces. Our benchmarks in Section 3 demonstrate the effectiveness of our solution.
- **[RQ-2]:** Effectively supporting data segmentation involves partitioning the dataspace into many smaller chunks. In Atlas, this is performed during data insertion. Data segmentation is a core principle for integrating spatiotemporal support into Atlas.
- **[RQ-3]:** Interoperating with analytical engines is achieved by natively supporting canonical-HDFS' communication protocols. Data insertion and retrieval within Atlas may be performed using the HDFS protocol. We have implemented two specific extensions, namely URL-embedded queries and block ID spatial encoding. Ultimately, this enables

seamless integration with several analytical engines, while promoting adoption into existing workflows.

- **[RQ-4]:** Atlas uses a spatiotemporally aligned data distribution algorithm to manage the competing pulls of dispersion and collocation. Atlas places data chunk replicas throughout the cluster enabling a variety of spatiotemporal access. As a result, data distribution for a specific spatiotemporal subspace provides both a thin layer, where data is present on many hosts, and a concentrated group, where data is united over a few hosts. This approach minimizes network I/O (in the form of data movements) while facilitating efficient processing of the dataspace for diverse analytics.

As part of future work, we will explore optimizations of specialized spatiotemporal algorithms within Atlas, including k-nearest neighbor queries and a variety of spatiotemporal dataset joins. Another topic for investigation is support for data imputations, where we estimate portions of a spatiotemporal scope by leveraging spatiotemporally proximate relationships.

ACKNOWLEDGMENTS

This research was supported by grants the US National Science Foundation [OAC-1931363, ACI-1553685], the US Department of Homeland Security [D15PC00279], the Advanced Research Projects Agency- Energy(ARPA-E), and a Cochran Family Professorship.

REFERENCES

[1] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. 2013. Hadoop gis: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment* 6, 11 (2013), 1009–1020.
[2] A. Akdogan, U. Demiryurek, F. Banaei-Kashani, and C. Shahabi. 2010. Voronoi-based geospatial query processing with mapreduce. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. IEEE, 9–16.
[3] F. Aurenhammer. 1991. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)* 23, 3 (1991), 345–405.
[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. 1990. The R*-tree: an efficient and robust access method for points and rectangles. In *Acm Sigmod Record*, Vol. 19. Acm, 322–331.
[5] A. Eldawy, Y. Li, M. F. Mokbel, and R. Janardan. 2013. CG_Hadoop: computational geometry in MapReduce. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 294–303.
[6] A. Eldawy and M. F. Mokbel. 2014. Pigeon: A spatial mapreduce language. In *2014 IEEE 30th International Conference on Data Engineering*. IEEE, 1242–1245.
[7] A. Eldawy and M. F. Mokbel. 2015. Spatialhadoop: A mapreduce framework for spatial data. In *2015 IEEE 31st international conference on Data Engineering*. IEEE, 1352–1363.
[8] R. A. Finkel and J. L. Bentley. 1974. Quad trees a data structure for retrieval on composite keys. *Acta informatica* 4, 1 (1974), 1–9.
[9] G. Fox, S. Lim, S. Pallickara, and M. Pierce. 2005. Message-based cellular peer-to-peer grids: foundations for secure federation and autonomic services. *Future Generation Computer Systems* 21, 3 (2005), 401–415.
[10] G. Fox, S. Pallickara, and X. Rao. 2005. Towards enabling peer-to-peer Grids. *Concurrency and Computation: Practice and Experience* 17, 7-8 (2005), 1109–1131.
[11] S. Ghemawat, H. Gobioff, and S.-T. Leung. 2003. The Google file system. (2003).
[12] A. Guttman. 1984. *R-trees: A dynamic index structure for spatial searching*. Vol. 14. ACM.
[13] D. Han and E. Stroulia. 2013. Hgrid: A data model for large geospatial data sets in hbase. In *2013 IEEE Sixth International Conference on Cloud Computing*. IEEE, 910–917.
[14] I. Kamel and C. Faloutsos. 1993. *Hilbert R-tree: An improved R-tree using fractals*. Technical Report.
[15] V. Kantere, S. Skiadopoulos, and T. Sellis. 2008. Storing and indexing spatial data in p2p systems. *IEEE Transactions on Knowledge and Data Engineering* 21, 2 (2008), 287–300.
[16] J. Lu and R. H. Güting. 2012. Parallel secondo: boosting database engines with hadoop. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*. IEEE, 738–743.
[17] W. Lu, Y. Shen, S. Chen, and B. C. Ooi. 2012. Efficient processing of k nearest neighbor joins using mapreduce. *Proceedings of the VLDB Endowment* 5, 10 (2012), 1016–1027.
[18] Q. Ma, B. Yang, W. Qian, and A. Zhou. 2009. Query processing of massive trajectory data based on mapreduce. In *Proceedings of the first international workshop on Cloud data management*. ACM, 9–16.
[19] A. Mondal, Y. Lifu, and M. Kitsuregawa. 2004. P2pr-tree: An r-tree-based spatial index for peer-to-peer environments. In *International Conference on Extending Database Technology*. Springer, 516–525.
[20] S. Nishimura, S. Das, D. Agrawal, and A. El Abbadi. 2011. Md-hbase: A scalable multi-dimensional data infrastructure for location aware services. In *2011 IEEE 12th International Conference on Mobile Data Management*, Vol. 1. IEEE, 7–16.
[21] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. 2008. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 1099–1110.
[22] T. Sellis, N. Roussopoulos, and C. Faloutsos. 1987. *The R+-Tree: A Dynamic Index for Multi-Dimensional Objects*. Technical Report.
[23] K. Shvachko, H. Kuang, S. Radia, R. Chansler, et al. 2010. The hadoop distributed file system.. In *MSST*, Vol. 10. 1–10.
[24] Y. L. Simmhan, S. L. Pallickara, N. N. Vijayakumar, and B. Plale. 2007. Data management in dynamic environment-driven computational science. In *Grid-based problem solving environments*. Springer, 317–333.
[25] E. Tanin, A. Harwood, and H. Samet. 2007. Using a distributed quadtree index in peer-to-peer networks. *The VLDB Journal—The International Journal on Very Large Data Bases* 16, 2 (2007), 165–178.
[26] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. 2009. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment* 2, 2 (2009), 1626–1629.
[27] M. N. Vora. 2011. Hadoop-HBase for large-scale data. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, Vol. 1. IEEE, 601–605.
[28] K. Wang, J. Han, B. Tu, J. Dai, W. Zhou, and X. Song. 2010. Accelerating spatial data processing with mapreduce. In *2010 IEEE 16th International Conference on Parallel and Distributed Systems*. IEEE, 229–236.
[29] C. Zhang, F. Li, and J. Jestes. 2012. Efficient parallel kNN joins for large data in MapReduce. In *Proceedings of the 15th international conference on extending database technology*. ACM, 38–49.
[30] N. Zhang, G. Zheng, H. Chen, J. Chen, and X. Chen. 2014. Hbasespatial: A scalable spatial data storage based on hbase. In *2014 IEEE 13th international conference on trust, security and privacy in computing and communications*. IEEE, 644–651.
[31] S. Zhang, J. Han, Z. Liu, K. Wang, and S. Feng. 2009. Spatial queries evaluation with mapreduce. In *2009 Eighth International Conference on Grid and Cooperative Computing*. IEEE, 287–292.
[32] S. Zhang, J. Han, Z. Liu, K. Wang, and Z. Xu. 2009. Sjmr: Parallelizing spatial join with mapreduce on clusters. In *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 1–8.
[33] Y. Zhong, J. Han, T. Zhang, Z. Li, J. Fang, and G. Chen. 2012. Towards parallel spatial query processing for big spatial data. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE, 2085–2094.