A Spatiotemporal Indexing Approach for Efficient Processing of Big Array-based Climate Data with MapReduce

Zhenlong Li[a,b], Fei Hu[a], John L. Schnase[c], Daniel Q. Duffy[d], Tsengdar Lee[e], Michael K. Bowen[d], Chaowei Yang[a*]

[a] *Spatiotemporal Innovation Center, George Mason University, Fairfax, 22030*

[b] *Department of Geography, University of South Carolina, Columbia, SC, 29208*

[c] *Office of Computational and Information Sciences and Technology,NASA Goddard Space Flight Center, Greenbelt, MD, 20771*

[d] *NASA Center for Climate Simulation, Goddard Space Flight Center, Greenbelt, MD, 20771*

[e] *Earth Science Division, NASA Headquarters, Washington D.C., 20024*

[*] *Corresponding author*

zhenlong@mailbox.sc.edu, {fhu, cyang3}@gmu.edu, {john.l.schnase, daniel.q.duffy, tsengdar.j.lee, michael.k.bowen}@nasa.gov

To cite this paper:

Li, Z., Hu, F., Schnase, J. L., Duffy, D. Q., Lee, T., Bowen, M. K., & Yang, C. (2017). A spatiotemporal indexing approach for efficient processing of big array-based climate data with MapReduce. *International Journal of Geographical Information Science*, 31(1), 17-35.

A Spatiotemporal Indexing Approach for Efficient Processing of Big Array-based Climate Data with MapReduce

Climate observations and model simulations are producing vast amounts of array-based spatiotemporal data. Efficient processing of this data is essential for assessing global challenges such as climate change, natural disasters, and diseases. This is challenging not only because of the large data volume, but also because of the intrinsic high-dimensional nature of geoscience data. To tackle this challenge, we propose a spatiotemporal indexing approach to efficiently manage and process big climate data with MapReduce in a highly scalable environment. With this approach, big climate data is directly stored in a Hadoop Distributed File System in its original, native file format. A spatiotemporal index is built to bridge the logical array-based data model and the physical data layout, which enables fast data retrieval when performing spatiotemporal queries. Based on the index, a data-partitioning algorithm is applied to enable MapReduce to achieve high data locality, as well as balancing the workload. The proposed indexing approach is evaluated using the NASA Modern-Era Retrospective Analysis for Research and Applications (MERRA) climate reanalysis dataset. Experimental results show that the index can significantly accelerate querying and processing (~10x speedup compared to the baseline test using the same computing cluster), while keeping the index-to-data ratio small (0.0328%). The applicability of the indexing approach is demonstrated by a climate anomaly detection deployed on a NASA Hadoop cluster. This approach is also able to support efficient processing of general array-based spatiotemporal data in various geoscience domains without special configuration on a Hadoop cluster.

Keywords: spatiotemporal index, big climate data, array-based, Hadoop MapReduce, HDFS, NASA MERRA, climate change

1. Introduction

Big data, referring to the enormous volume, velocity, and variety of data (NIST Cloud/BigData Workshop, 2014), has become one of the most significant technology shifts in the 21st century (Mayer-Schönberger and Cukier, 2013). In climate science, large volumes of spatiotemporal data are generated to describe the complex Earth climate system. This data normally includes observational data from remote sensors (e.g. space-borne instruments), numerical simulation data from climate modelling, and model-based retrospective analysis data created by assimilating observational data into climate models (Overpeck et al. 2011). Climate data is accumulating at an exponentially increasing rate due to the fast-paced advancement of sensors and high performance computing technologies (Edwards 2010, Li et al. 2015). It is predicted that the climate simulation and observational data held by NASA alone will reach nearly 350 Petabytes by 2030 (Skytland 2012). In fact, climate science is a typical domain that represents the big data shift across all geoscience domains (Schnase et al. 2014, Edwards 2010).

Big climate data is playing a critical role in studies that enable us to better understand how the complex climate system works, and thus attempt to predict future climate changes. Processing and making sense of vast amounts of climate data enables scientists to answer key questions in climate research. However, efficient handling of this data poses critical challenges for at least two types of operations: spatiotemporal data mining and spatiotemporal query.

Spatiotemporal data mining: Mining interesting climate trends and spatiotemporal patterns from terabytes of high-dimensional data sets is important to climate studies. Example applications include detecting temperature anomalies in the global climate system, identifying geographical regions with similar climate patterns, and investigating spatiotemporal distribution of extreme weather events (Das and Parthasarathy 2009). However, climate data is high dimensional (two or three dimensions of space and one temporal dimension) and normally contains hundreds of variables describing land, oceans, and the atmosphere. Mining information from these high-dimensional big data sets is challenging. It takes approximately three hours to read one terabyte of data using a single computer with a 100 megabytes per second (mbps) hard drive read speed, and this is before analyzing complex spatiotemporal relationships. Therefore, distributed parallel computing must be employed to process big climate data in a feasible time frame. In addition, the traditional approach of storing the data in a centralized repository and later moving it to specialized computing facilities for analysis is no longer efficient (Schnase et al. 2014).

Spatiotemporal data query: Through spatiotemporal aggregation, basic statistical information such as means, maxima and minima can be derived for further analysis. Often times, scientists are only interested in part of the data, thus a spatiotemporal query is required. Such a query may include three constraints: geographic area (space), time period (time) and variables, as, for example, finding the precipitation data from 1979 to 2014 in the United States. Such an operation is challenging because climate data is normally stored in array-based high-dimensional files (e.g. NetCDF or HDF), with each file containing many variables. Since the metadata is distributed across different files, one often needs to scan all the data files to retrieve a small amount of data. Managing big climate data in such a way that supports efficient query and retrieval is essential for big climate data processing.

MapReduce, a parallel data processing framework pioneered by Google (Dean and Ghemawat 2008), has been proven to be effective when it comes to handling big data challenges. As an open source implementation of MapReduce, Hadoop (White 2009) has gained increasing popularity over the past several years. However, Hadoop is not designed to handle spatiotemporal data, which has triggered a multitude of studies to bridge the gap; this is elaborated in Section 2. Aiming to address the challenges posed by the typical operations mentioned above, we propose a novel spatiotemporal indexing approach that significantly accelerates querying and processing of big climate data. Specifically, a spatiotemporal index is proposed to bridge the logical array-based data model and the physical data layout, which enables host-aware fast data retrieval with spatiotemporal querying. A data-partitioning algorithm is introduced to enable MapReduce to achieve high data locality and a more balanced workload when processing in parallel.

The remainder of this paper is organized as follows: Section 2 reviews research on using Hadoop to process array-based spatiotemporal data; Section 3 details our indexing approach; Section 4 evaluates the proposed approach by conducting a series of experiments; Section 5 demonstrates how the indexing approach could be used in practical climate studies; finally, Section 6 summarizes the research and envisions future research.

2. Related work

To bridge the gap between array-based spatiotemporal data and Hadoop MapReduce, a variety of studies have been carried out. Zhao, et al. (2010) converted NetCDF data into ASCII-based CDL (network Common data form Description Language) files. This approach works, but it increases the size of the data set and breaks the original data integrity by disrupting its logical organization, changing format, and dissociating its embedded metadata. Duffy et al. (2012) re-organized array-based NetCDF data into Hadoop Sequence Files, a flat file consisting of binary key/value pairs. This approach keeps the data integrity since the data is still in its original format, but Hadoop Sequence Files are not optimized for random access, which significantly impairs the performance. To overcome this issue, Li et al. (2015) decomposed the array-based data and stored it in HBase, a NoSQL database built upon HDFS. However, all these methods need to transform the original data format from NetCDF. This is problematic, because converting vast amounts of data to other formats requires extra effort and time, and two copies of the data must be maintained (original and converted), which increases data management complexity.

To avoid the issues caused by data conversion, Buck et al. (2011) developed SciHadoop, which provides logical query abilities over array-based data models such as NetCDF data. SciHadoop uses techniques such as physical-to-logical translation, chunking, grouping, and sampling to bridge between the logical and physical organization of data on-the-fly. However, using sampling to identify data locality introduces overhead before the data can be processed. In addition, each query requires a sampling process, even if the resultant data is the same.

Indexing methods are widely used to accelerate querying of structured data. Much work has been done to embed spatial trees such as Quad-tree (Finkel and Bentley, 1974) and R-Tree (Guttman 1984) into Hadoop to support large-scale spatial data querying. SpatiaHadoop (Eldawy and Mokbel 2013) adapts traditional spatial index structures to form a two-level spatial index of global and local indexing for vector data. Based on SpatialHadoop, SHAHED (Eldawy et al. 2015) builds Quad-trees to index satellite data. However, these index trees require loading all the values into themselves, so they may be much bigger than the original data. How to build, store and search the index becomes a new problem. Once generated, scientific data is usually read-only, and bitmap indexes can be used to reduce the index size. Fastbit (Wu et al., 2009) takes advantage of bitmap compression, encoding, and binning to build a multidimensional bit index for scientific data. This approach, however, does not work for data sets that contain variables with very high cardinalities. SciHive (Geng et al. 2013, Geng et al. 2014) calculates the value range for each HDFS block by historical queries to build a distributed adaptive index. But it has a special requirement for the block size and the value range of the raw data. When the block size is not big enough to cover a file, it will result in a large amount of remote reading, which is very slow. GeoBase develops a Z-region index to interface array-based data and key-value stores based on HBase (Malik 2013). The Z-region index enables efficient range queries and aggregation queries. When partitioning the data set, however, some data rearrangement is required, as well as processing prior to reading the input splits.

The above studies provided valuable guidelines for leveraging the Hadoop MapReduce framework to handle big spatiotemporal data challenges in geoscience domains. However, these approaches introduce new overhead problems, such as disk overhead caused by data transformation and index construction, and CPU overhead caused by logical-to-physical transformation. In addition, most approaches are not flexible due to their special requirements for each particular dataset or the specific configurations of the computing environment. The indexing approach proposed in this paper aims to efficiently query and process big array-based spatiotemporal data natively without any data transformation or special configuration of the Hadoop cluster.

3. Methodologies

3.1. Spatiotemporal Query Model for MapReduce

Array-based data models are widely used to represent spatiotemporal scientific data. Typically, an array-based data file is a multi-dimensional array consisting of two or three spatial dimensions (latitude, longitude and/or altitude) and one temporal dimension (time) (Figure 1). In climate data, each layer in the array is a two-dimensional spatial grid storing the values for a specific climate variable. These layers are further grouped by altitude, and all variables for a given time period are physically stored together to form the temporal dimension. This hierarchical data structure, coupled with metadata and associated access libraries (e.g. NetCDF for JAVA), makes it possible to retrieve the value for any variable in the collection given a specified time and location.

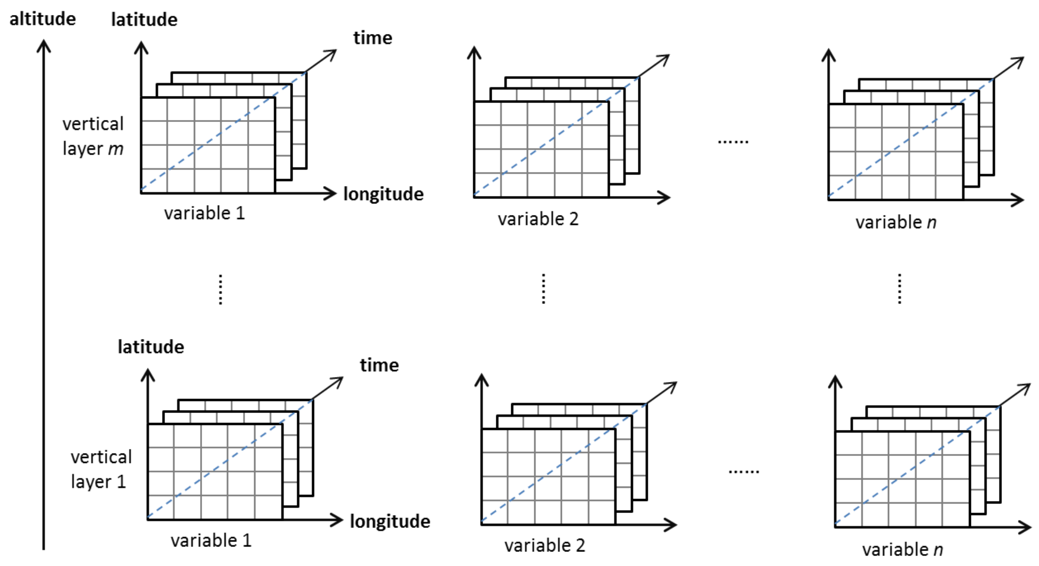

Figure 1. Illustration of array-based data model (n variables with 3D space and 1D time).

Despite the elegance of this approach, the logical structure of array-based data makes it difficult to use MapReduce. In a MapReduce environment, files are loaded into HDFS as fix-sized blocks (e.g. 64 Mb or 128 Mb) distributed across the nodes of a storage cluster. Each block is generally replicated with a factor of three (Figure 2). HDFS's block-oriented storage model is essential for MapReduce parallelization. However, these blocks are created using the byte streams of source data files. With array-based data, important information about its logical organization, such as variables, space, and time, is ignored. In addition, MapReduce operations are based on key-value pairs, which do not easily map into the hierarchical, multi-dimensional logical data model that is intrinsic to geospatial data. These mismatches make it challenging to process array-based spatiotemporal data with MapReduce.

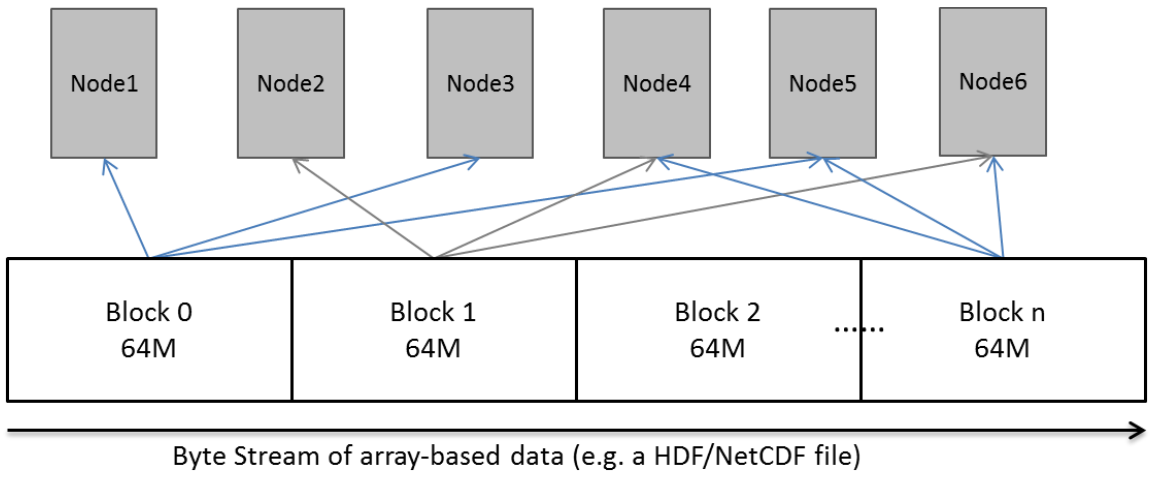

Figure 2. Array-based data is stored across nodes as blocks without any logical data model information; in this example, every block is triplicated across nodes.

In order to bridge between the array-based data and MapReduce, we introduce a spatiotemporal query model (*STQuery*):

$$ArrayData \rightarrow STQuery(Var, Cov, Alt, Time) \rightarrow Grids \rightarrow MapReduce$$

In the query,

- ***Var*** denotes a subset of the variables: $Var \in \{v_1, v_2, \dots, v_n\}$
- ***Cov*** denotes a 2D spatial grid (coverage) defined by $Lat \in \{-90°, 90°\}$ and $Lon \in \{-180°, 180°\}$
- ***Alt*** denotes a subset of the vertical layers (altitude): $Alt \in \{a_1, a_2, \dots, a_n\}$
- ***Time*** denotes a subset of the timestamps: $Time \in \{t_1, t_2, \dots, t_n\}$

STQuery filters the *ArrayData* using constraints on the logical data view. The result of a query is a subset of the original data consisting of many 2D grids, with each grid being represented as a $grid(v,t,a)$, where $(v,t,a) \in (Var, Time, Alt)$. Thus, a query containing *n* variables, *m* timestamps, and *k* vertical layers results in a total of $n \times m \times k$ number of grids. These grids are stored as key-value pairs recognizable by MapReduce. Grids generated by each query have the same or overlapping geographic extent as specified by the *Coverage* query parameter. When the *Coverage* parameter covers an entire geographic region, the grid is equal to the layer in the original array-based data model. Based on the query model, the problem of processing spatiotemporal array-based data with MapReduce is transformed to the problem of processing the $n \times m \times k$ spatial grids (key-value pairs).

3.2. Spatiotemporal Index

The spatiotemporal query model is an essential concept for building the spatiotemporal index and improving the efficiency of MapReduce, because it uses *Grids* to map a file's logical, multi-dimensional array data model to the key-value pairs used by MapReduce. For example, suppose that *STQuery* references 600 grids on a six-node cluster. Each node can potentially process 100 grids in parallel during the *map* stage of a MapReduce operation. However, this parallelism cannot typically be realized, because MapReduce's physical view of the data (byte streams, blocks, and nodes) lacks locality information about the data's logical organization. The problem is worsened by the fact that an array data file can easily reach several gigabytes in size, while a grid might only be a small portion of the file. This problem can be solved, however, if we know where grids are stored. The spatiotemporal index makes this possible by linking physical location information to logical, spatiotemporal information.

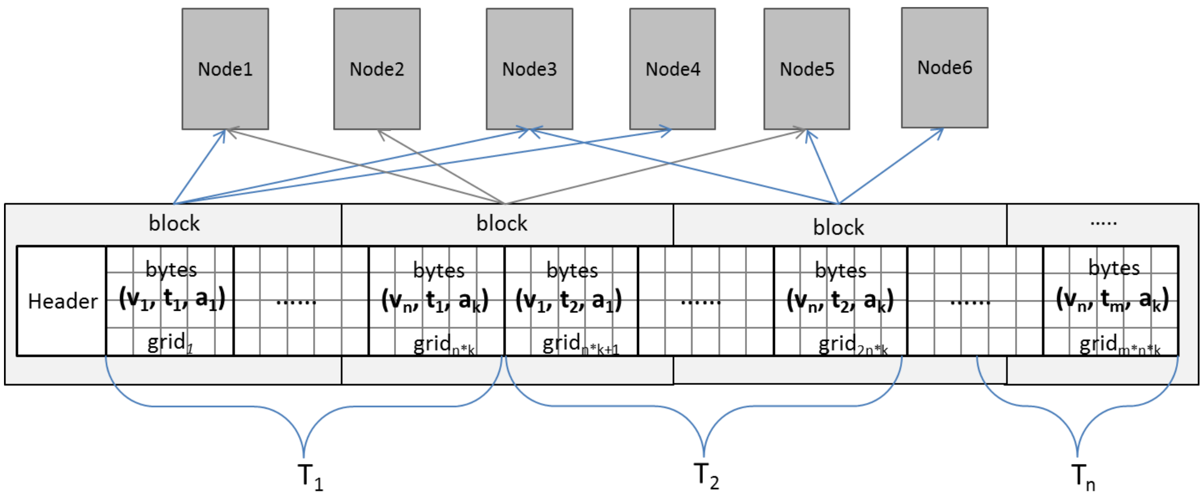

Figure 3. Relationship between the physical location information (node, file, and byte) and the logical spatiotemporal information (time, space, and variable) (assumed replication factor of 3).

Figure 3 depicts the relationship between a file's logical space, time, and variable view of the data, and Hadoop's physical byte stream, block, and node view of the data. The array-based file is stored in HDFS as a byte stream starting with a file header, which is followed by a sequence of grids that are temporally ordered. Each block is duplicated across HDFS nodes based on a replication factor. The block size is configured by the user (128 megabytes by default in Hadoop Yarn1). Grid sizes are determined by the spatial resolution and spatial coverage of the original file. For global climate data, the grid size is normally not bigger than several megabytes. For instance, the NASA Modern-Era Retrospective Analysis for Research and Applications (MERRA) product used in our work has a spatial resolution of 2/3 °longitude by 1/2 °latitude; the resultant grid size is ~ 0.66 megabytes per climate variable. Therefore, each block generally contains many grids, with each grid being replicated across the nodes.

The proposed index structure is illustrated in Figure 4. The index contains five components: *gridId*, *startByte*, *endByte*, *nodeList,* and *fileId* for three levels of the index: byte level, file level, and node level. To build the index, the values of the five components are extracted from the array-based data stored in HDFS using an appropriate access library (e.g. NetCDF for Java).

gridId is the bridge between the logical data view and physical data layout. It consists of three parameters in the logical view: variable, time, and altitude. *startByte* and

1 http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

endByte are the byte-level indices that record the exact byte location of the grid in a file. *fileId* is the file-level index that records which file a grid belongs to and how data in that file is compressed. The byte- and file-level indices enable each grid to be read directly from a file's byte stream using the file system's native I/O method. This improves efficiency by eliminating the need to consult metadata to retrieve a piece of data from a large file.

nodeList records the node location where grids are physically stored. The number of nodes in each list is equal to the replication factor. Some grids are split into two blocks as illustrated in Figure 3. For these grids, a node in the *nodeList* may only store part of the grid. However, since the block size is generally much larger than the grid size, most grids remain intact within blocks and nodes across the HDFS. As shown in Section 3.3, an effective grid assignment strategy allows most grids to be read locally, which maximizes data locality, an important factor affecting performance in MapReduce.

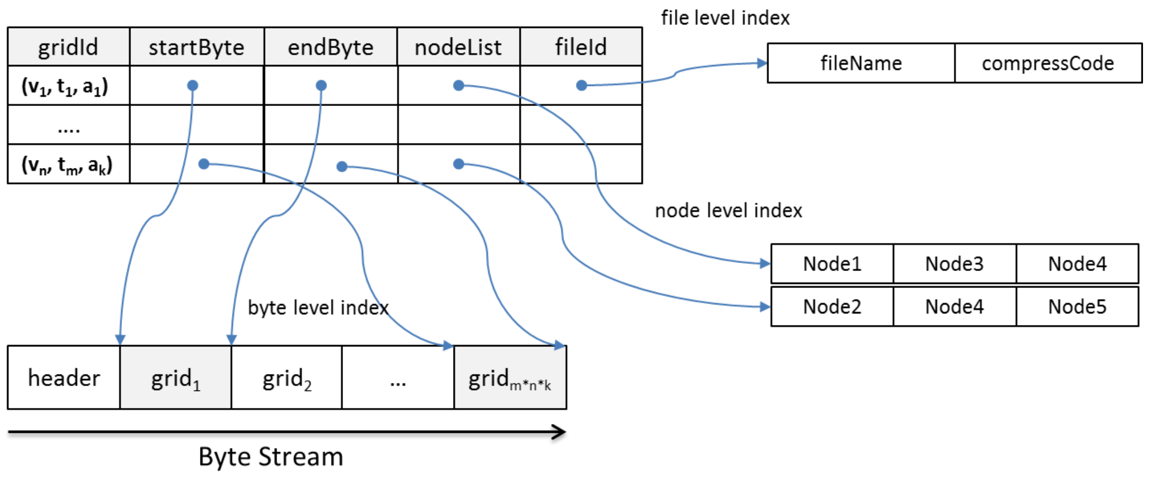

Figure 4. Structure of the spatiotemporal index.

Note that the index structure discussed here is a logical organization showing the essential components of the index and how they are linked. An actual implementation of the spatiotemporal index will depend on specific data, application, and deployment requirements. Generally, storing the index in a relational database is recommended, because the index's logical structure maps well to relational tables, and relational databases provide mature and efficient querying capabilities.

3.3. Grid Partition Strategy

HDFS partitions large files into many logical splits, and then assigns these splits to physical data blocks on physical nodes. How these splits are partitioned and assigned directly impacts data locality, which has a dramatic affect on the performance of MapReduce. This section develops a partition strategy that uses the spatiotemporal index to optimize processing performance by 1) keeping high data locality for each map task, 2) balancing the workload across cluster nodes, and 3) generating a proper number of map tasks to minimize the overhead.

3.3.1. Grid assignment

As discussed in Section 3.1, an array-based spatiotemporal query generates $n \times m \times k$ grids where *n*, *m,* and *k* represent the number of variables, timestamps, and vertical layers respectively. The spatiotemporal index helps maximize locality by tracking grid location at the node level. The grid assignment strategy described here further improves locality by grouping and assigning grids to the nodes where they are physically stored (Algorithm 1).

First, the number of grids that will be assigned to each node is calculated. When the total number of grids (N_{total}) can be evenly divided by the number of data nodes (S_{node}), each node is assigned the same number of grids as $G = N_{total}/S_{node}$. When N_{total} cannot be evenly divided by $S_{node},$ an equal number of nodes cannot be assigned across the nodes. In this case, we first assign $G = [N_{total}/S_{node}]$ grids to each node, each of the remaining grids ($N_{total} - S_{node} * G$) are randomly assigned to different nodes.

After the number of grids to be assigned to each node is determined, we assign grids from the grid pool. A grid is suitable to be assigned to a node if the node appears in a grid's node list in the spatiotemporal index. Because the data is replicated, a grid may be suitable for several nodes. Each node chooses their suitable grids from the grid pool. Any leftover grids are randomly assigned to a node. When all grids have been assigned, some nodes may have more or less than the calculated number of grids, which will result in an unbalanced workload. This is resolved by transferring some grids from the nodes with too many grids to the nodes with fewer grids. This assignment strategy ensures that each node is assigned nearly the same number of grids, and most grids (over 99%) fully reside on their suitable nodes. This will significantly increase the performance because transferring grids across nodes will add significant communication overhead.

3.3.2. Grid combination

The improvements enabled by the spatiotemporal index and grid partitioning strategy are further enhanced by a grid combing strategy that optimizes the use of a MapReduce cluster's available resources. Slots (or *containers* in Hadoop V2) refer to the number of parallel map tasks that can run on a node. This is a configurable feature of the Hadoop environment. Because a grid normally refers to a small amount of data, making each grid an input split to a map task would result in many tasks and much overhead. To address this problem, we introduce a combining strategy that organizes an appropriate number of grids to an input split according the cluster's available map slots.

```
Step 1: for each node ($node_i$) in $nodeList$
                determine the number of grid($G_i$) for $node_i$
                assign $G_i$ suitable grids from the grid pool for $node_i$
        end for
Step 2: for each grid ($grid_j$) left in the grid pool
                assign $grid_j$ to a suitable $node$
        end for
Step 3: for each node ($node_j$) in $nodeList$
                if ($num_j$ of grids in $node_j$)> $G_i$ then
                        add $node_j$ to $fullNodeList$
                end if
                if the $num_j$ of grids in $node_j$ < $G_i$ then
                        add $node_j$ to $unfullNodeList$
                end if
         end for
         for each node ($node_j$) in $unfullNodeList$
                 choose ($G_j$ - $num_j$) suitable grids from $fullNodeList$
                 assign the grids to $node_j$
         end for
```

Algorithm1. Grid assignment strategy (*nodeList* denotes the list of nodes in the cluster; *fullNodeList* denotes the nodes that have been assigned the number of grids determined in Step1; *unfullNodeList* denotes the nodes that have been assigned less than the number of grids determined in Step 1).

Assuming there are n_{slot} map slots available for $node_i$, and the number of grids assigned to $node_i$ (denoted as G_i) can be exactly divided by n_{slot}, the number of grids (n_j) for $slot_j$ can be calculated as $n_j = G_i/n_{slot}$. If G_i cannot be evenly divided by n_{slot}, slots cannot be assigned an equal number of grids. Using the same strategy described in Section 3.3.1, we first assign $n'_j = [G_i/n_{slot}]$ grids to $slot_j$, then randomly assign each of the left $count_{left}$ grids to a slot, where $count_{left} = G_i - \sum_{j=1}^{n_{slot}} n'_j$. By default, the leftover grids will be assigned to $slot_1 \sim slot_{count_{left}}$ as in Formula 1

$$n_{\mathrm{j}} = \begin{cases} 1 + n'_j, & 0 < j \leq count_{left} \\ \quad n'_j \ , & count_{left} < j \leq n_{slot} \end{cases} \qquad (1)$$

Figure 5 shows the overall process for index-based parallelization with MapReduce. As we describe in the next section, with indexing, grid assignment, and grid combination, we have achieved a high level of data locality (99% in our experiments) and significantly improved workload balance.

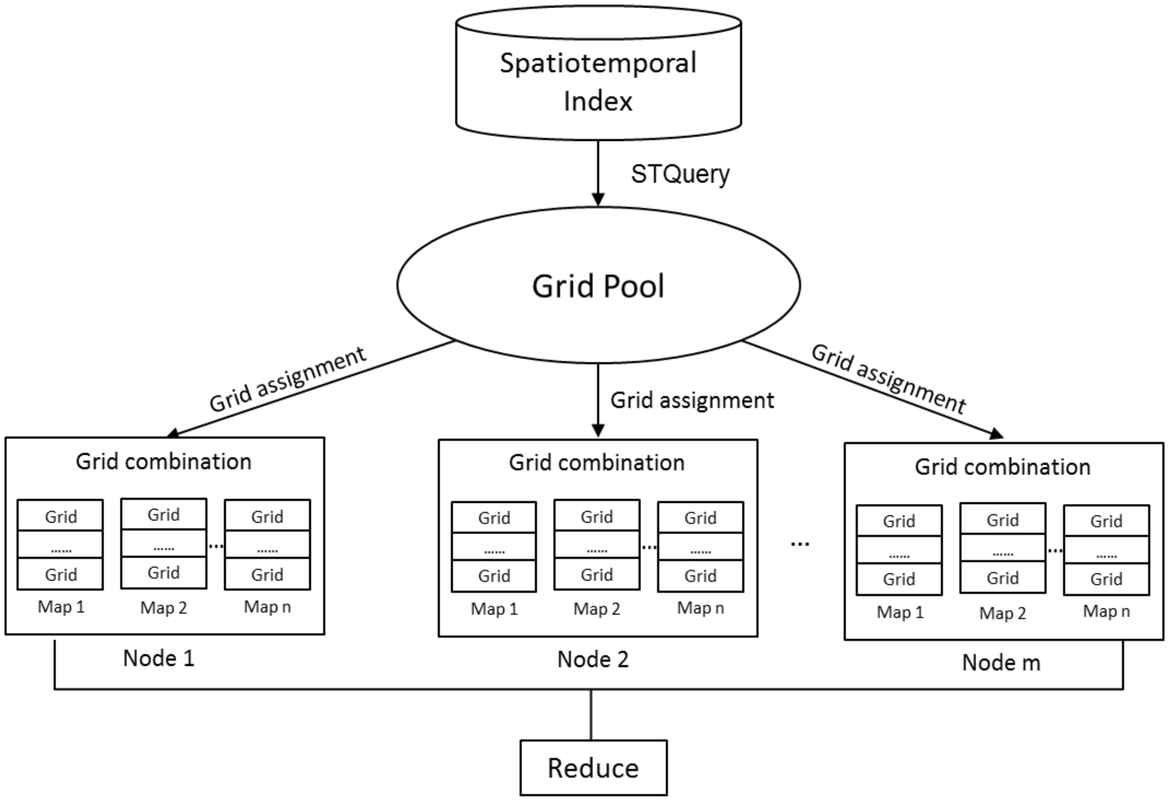

Figure 5. Index-based parallelization with MapReduce.

4. Evaluation

We conducted several experiments to evaluate the effect of this approach on run time, data locality, and load balance in MapReduce processing.

4.1. Experimental Design

The experiments were conducted on a Hadoop cluster (version 2.6.0) consisting of seven computer nodes (one master node and six slave nodes) connected with 1 Gigabit Ethernet (Gbps). Each node was configured with eight CPU cores (2.35GHz), 16 GB RAM, and CentOS 6.5.

Modern Era Retrospective-Analysis for Research and Applications data was used in all experiments. MERRA is a reanalysis of the last 35 years of global climate observation data from NASA (Rienecker et al. 2011). MERRA was created with the newest version of the Goddard Earth Observing System Data Assimilation System Version 5 (GEOS-5) (Duffy et al. 2012), and is playing an important role in studying weather and climate variability. MAT1NXINT (Bloom et al. 2005) is one product of MERRA. It contains nearly 111 2-dimensional hourly variables with a spatial resolution of 2/3 °longitude by 1/2 °latitude. This data is archived in the HDF-EOS format, based on HDF4 (Berrick et al. 2008). One month (January 2015) of the MAT1NXINT product (45.29 GB) was used as experimental data.

Three scenarios were evaluated. Each scenario computed the daily mean for a specified climate variable in a specified spatiotemporal range. The first scenario was performed without using the spatiotemporal index or grid partition strategy and served as a baseline. The second scenario used the spatiotemporal index without the grid partition strategy. The third scenario used the index with the grid partition strategy.

We embedded the HDFS file system interface into the NetCDF-java library to support reading MERRA's native HDF files into the HDFS without any preprocessing. The HDFS block size was configured as 128MB with a replication factor of three. It took two minutes to build the index for 45 GB of data. The index was stored in a MySQL database. The size of the index was 15.11 MB, resulting in an index-to-data ratio of 0.0328%.

4.2. Results and Discussion

4.2.1. Supporting spatiotemporal data mining

Computing basic statistics such as average for an entire dataset is considered as canonical operations in climate analytics (Schnase et al. 2014), and is an essential step in mining climate data. To test our approach in the three scenarios, we computed the daily global mean for all 111 variables in our dataset for January 2015. We evaluated three metrics: run time (time spent on each operation), data locality (ratio of local data read to all data read), and load balance (number of grids assigned to each node). To reduce variability and measurement error, we conducted the operation ten times and took average values of the three metrics. The average run time and data locality results are shown in Figure 6.

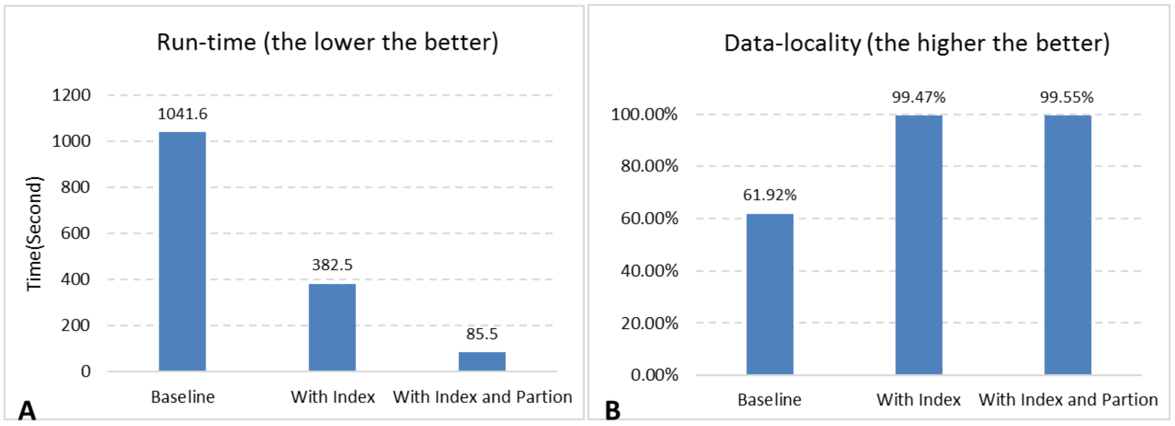

Figure 6. Daily global mean computation for all 111 variables of January 2015.

Figure 6A illustrates that using the index can reduce the run time from 1041 seconds to 382 seconds, achieving a 2.7x speed up. Applying the grid partition strategy reduced the run time even further to 85 seconds, achieving a 12x speed up over the baseline. Figure 6B shows that the index significantly improved data locality from 61.9% to 99.5%. This indicates that almost all data required for the parallel MapReduce operations was read from local nodes throughout the cluster. The reason data locality did not reach 100% is that a few grids are split and stored on different nodes, as explained in Section 3.2. In the baseline test, the locality was 61.92%, which is higher than the expected 50%. This was caused by the relatively small cluster size (six nodes) and the high replication factor (three).

The grid partition strategy further reduced run time compared to using the index alone. This speed-up resulted from balancing the workload among cluster nodes and generating the appropriate number of splits in order to reduce overhead. Figure 7 compares the grid assignments on each node for the latter two scenarios. When using only the index, the grids are distributed unequally across the nodes. The largest difference among them was 2325 grids (~1.32 GB). When applying the partition strategy, each node had nearly the same number of grids. The biggest difference among these nodes was only five grids (~2.92 MB). Therefore, with the grid partition strategy, the workload distribution in the cluster was much more balanced. In addition, the grid combination algorithm reduced the number of splits from 376 to 36, which alleviates the overhead caused by too many small map tasks.

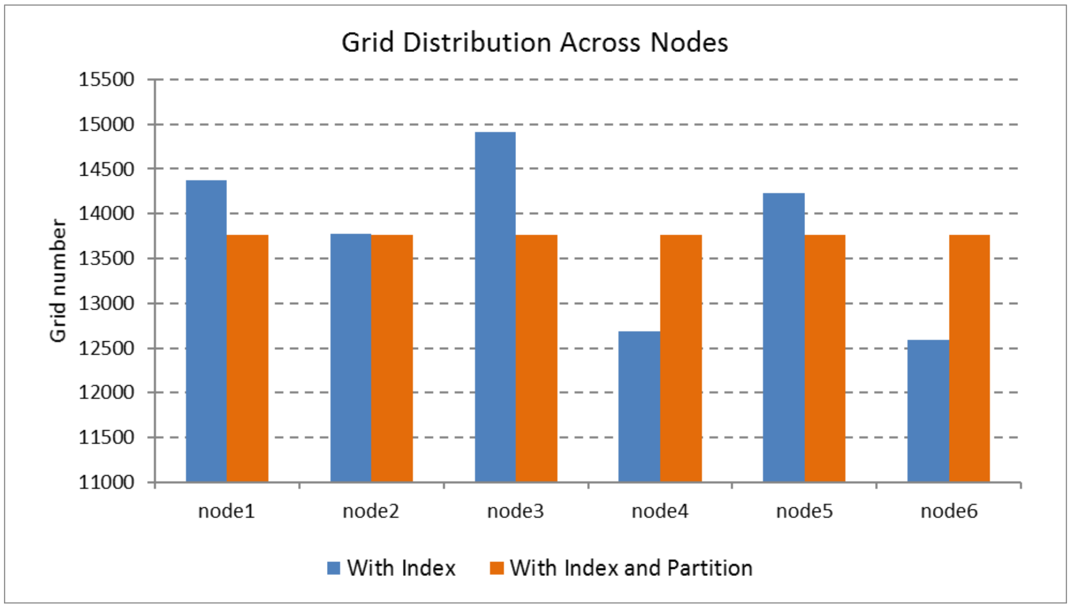

Figure 7. Grid distributions across the cluster nodes.

These experiments demonstrate that the index improved performance by maximizing data locality. The grid partition strategy further increased performance by balancing the workload and reducing overhead.

4.2.2. Supporting spatiotemporal data query

Indexing is designed to support highly efficient querying by reading only the required data rather than scanning an entire data set as well as maintaining high data locality. To test this, we evaluated the effect of varying *Var* and *Time* in the query model $STQuery(Var, Cov, Alt, Time)$. The first set of tests queried and computed the daily global mean for January 2015 for different numbers of variables. The second set of tests queried and computed the daily global mean for all variables for a varying numbers of days in January 2015.

Figure 8 shows the run time for the first set of tests comparing the baseline and the indexing approach with different numbers of variables. When increasing the number of variables in the query, the run time for the baseline increased dramatically from 55 seconds to 1042 seconds (a 19-fold increase), while the run time for the indexing approach increased only slightly from 35 to 85 (a 2.4-fold increase). To explain this result, the data locality for each test was measured (Figure 9). In the baseline, the average data locality for all tests was only 56.11%, indicating that nearly half of generated grids were read from non-local nodes. When increasing the number of variables, more grids were generated; as a result more grids are read remotely. This degraded the performance, as network congestion became the bottleneck. With the indexing approach, the data locality for each test was over 99%, indicating that nearly all generated grids were read locally regardless of how many variables were queried.

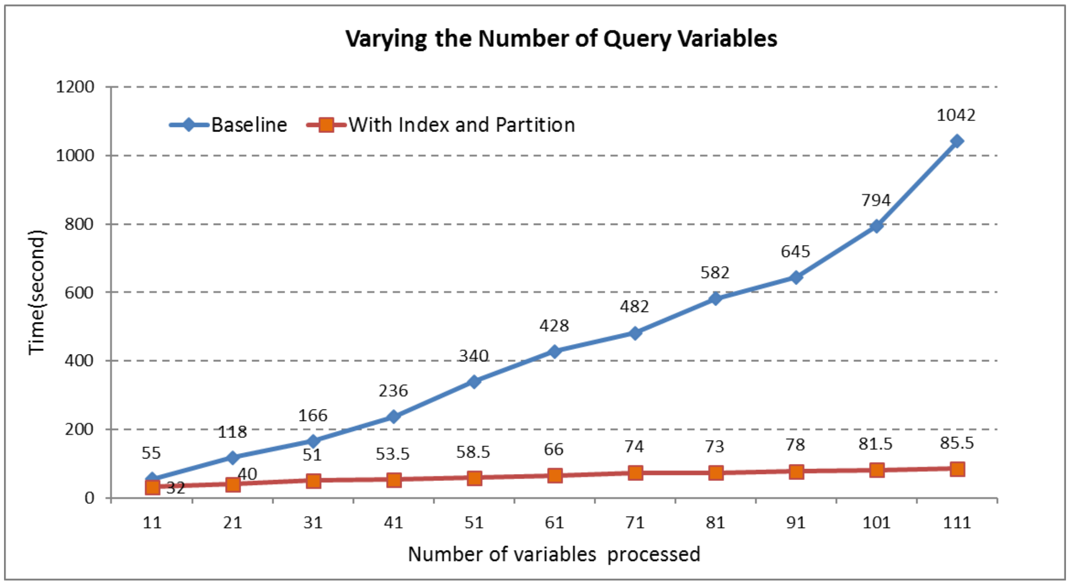

Figure 8. Run time for querying/computing the daily global mean for January 2015 for different numbers of variables.

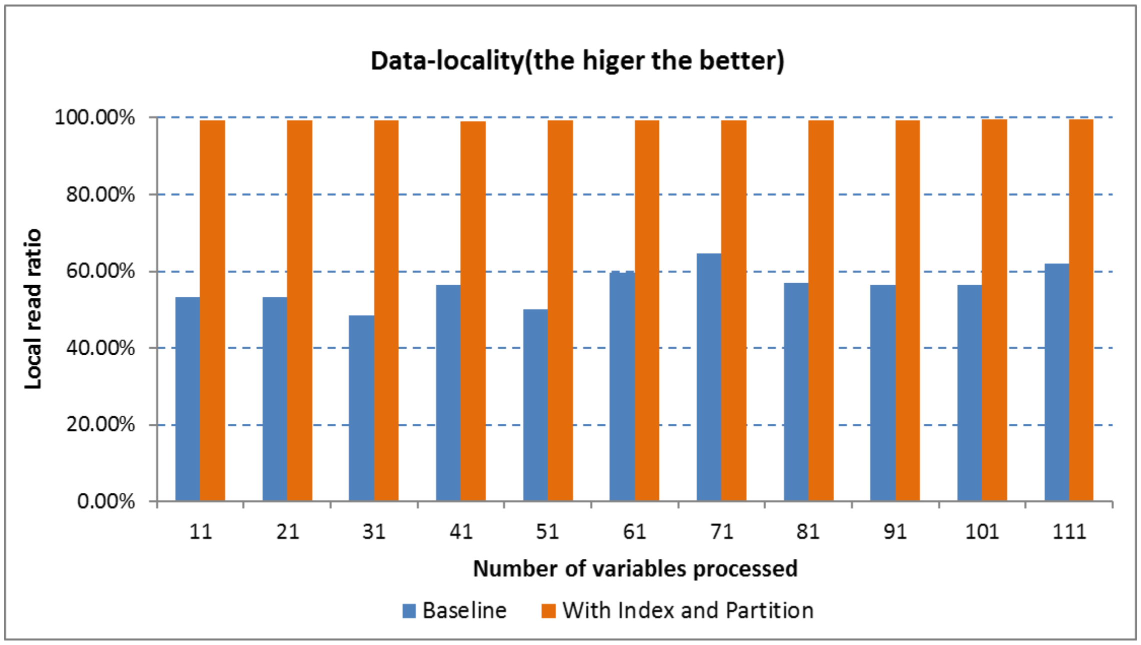

Figure 9. Data locality for querying/computing the daily global mean for January 2015 for different numbers of variables.

For the second set of tests, with varying numbers of days, the run time for each test showed similar patterns as the first set of tests (Figure 10). When the query time period increased, the baseline run time increased sharply with the number of days due to low data locality (a large amount of data was read remotely), while the indexing approach maintained a steady, low run time. When querying a small date range where the network bottleneck does not exist, the run time for the indexing approach is still far less than that of the baseline. This is because the indexing approach provides finer parallelization and load balancing mechanisms across the cluster nodes. For example, when querying the data from 2015-01-01 to 2015-01-03, the baseline launched only three map tasks, while the indexing approach launched 36 tasks on six nodes. The combination of the index and grid partition strategy can effectively decompose a data- and compute-intensive job into a reasonable number of small jobs to process in parallel.

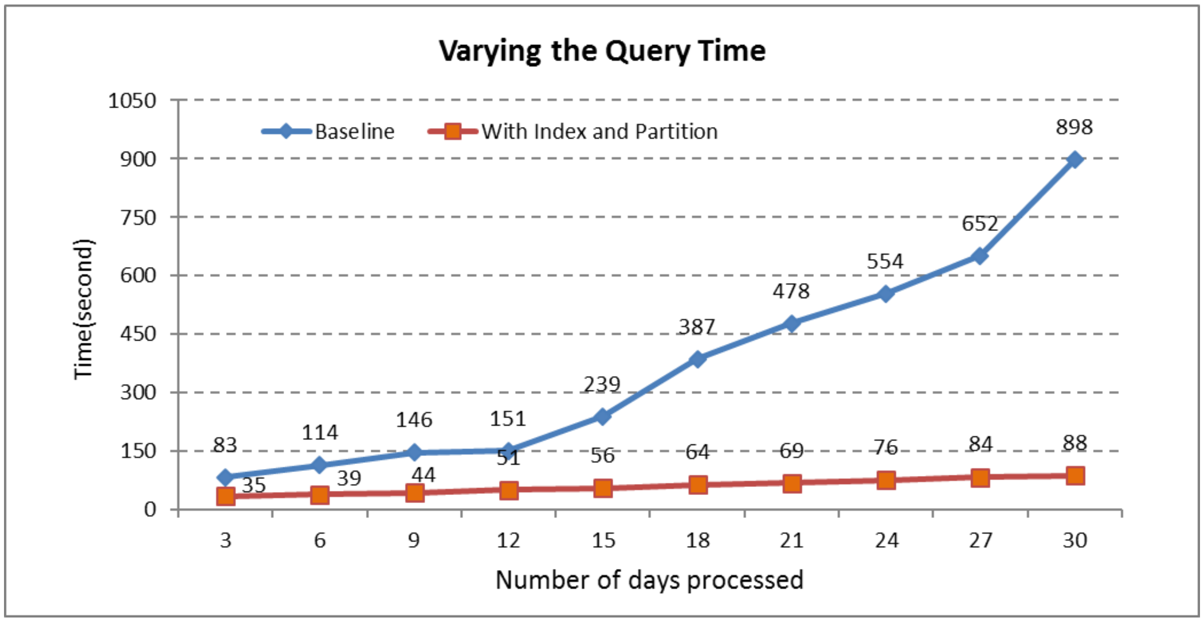

Figure 10. Run time for querying/computing the daily global mean for all variables for different numbers of days in January 2015.

In the above experiment we tested mean calculation, a canonical data processing operation in climate study. Calculating the mean value of a variable for a specified time period and geographic region clearly illustrates two functionalities of the spatiotemporal indexing approach: 1) *query*: one important function of the index is to quickly locate the required data (grids) from the huge data repository based on the spatiotemporal query criteria; 2) *process*: once the required grids are located, these grids will be simultaneously read into the cluster and distributedly processed (computing mean) in parallel.

5. Application: Anomaly Detection

The purpose of anomaly detection is to find abnormal patterns, or outliers, in a dataset. These patterns do not "fit" the dataset because they do not follow the expected behavior of the data (Chandola et al. 2009). Detecting anomalies in variables over time, such as fluctuations in temperature, is an important consideration in climate studies. To demonstrate how the proposed indexing approach could be used in practical climate studies, we deployed a preliminary application to analyze temperature anomalies on a NASA Goddard Space Flight Center (GSFC) Hadoop cluster. This cluster consists of 36 compute nodes (one master node, one high availability node, and 34 slave nodes) connected with 56 Gigabit per second (Gbps) Infiniband (IB). Each node is configured with 16 CPU cores (2.60 GHz) and 16 GB of RAM running CentOS 6.5.

The geospatial region used in the study was derived from the 100 GB MERRA MAIMNPANA[2] data (HDF4). The time period covers 1979 to 2013 (35 years) at a daily

2 http://disc.sci.gsfc.nasa.gov/mdisc/data-holdings/merra/instM_3d_ana_Np.shtml

(24 hour) resolution. This dataset contains 18 four-dimensional variables (time, latitude, longitude, and altitude) and four three-dimensional variables (time, latitude, and longitude), each with a spatial resolution of 2/3 °longitude by 1/2 °latitude. The spatiotemporal query for detecting the temperature anomalies is constructed as following:

$$STQuery(temperature, \{48.99, -125.52, 30.77, -62.15\}, layer1, \{1979 - 2013\}).$$

The spatial region of interest for this study is shown in Figure 11. Since there are 42 vertical layers, the first layer, nearest to the Earth's surface, was selected. The monthly temperature mean was then calculated for all logical points in the dataset (12 months per year * 35 years = 420 mean value calculations). Next, we performed statistical analysis by overlaying a Gaussian (normal) distribution in order to detect monthly temperature anomalies over the 35 year time period.

For each month (from January to December), the z-score for month *i* and year *j* is calculated using Formula 2.

$$z_{ij} = \frac{x_{ij} - u_i}{s_i}, i \in [1,12], j \in [1,35] \qquad (2)$$

Where,

x_{ij} is the mean temperature for month *i* and year *j*

u_i is the mean temperature for month *i* across 35 years

s_i is the standard deviation for month *i* across 35 years

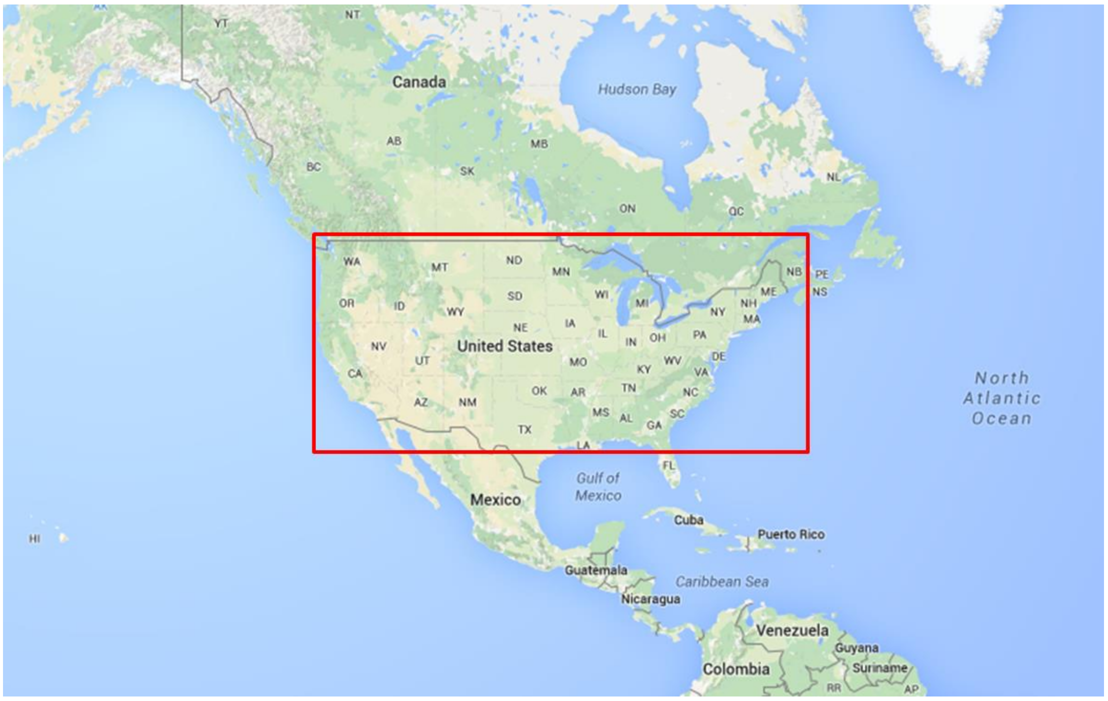

Figure 11. Selected geographic region for the temperature anomaly detection.

This application took only 32 seconds to compute all the mean values and subsequently detect the temperature anomalies for each month by analysing the 100 GBs of data. Table 1 shows the temperature anomalies for June for the 35 years.

Table 1. June temperature anomalies for the selected geospatial area from 1979 to 2013 (confidence level $\alpha = 0.01$, two-tail test: $z_{critical} = \pm 2.576$).

Year	**z-score**	**Anomaly (unit: Celsius)**
1984	-3.283	-0.670
1988	2.930	0.598
1990	2.974	0.607

1992	-2.813	-0.575
1994	3.289	0.671
1996	-4.434	-0.905
2005	-3.035	-0.620
2007	-6.011	-1.227
2013	3.518	0.718

Traditional approaches for climate anomaly detection usually first compute and cache various mean values (e.g. monthly mean, yearly mean) for a pre-defined spatial region (Li et al. 2013), such as the globe or the northern/southern hemispheres, then conduct analyses based on these aggregated values. This is not flexible, since it does not support a user-defined, arbitrary spatial region. The proposed indexing approach overcomes this by enabling us to 1) conduct various data analysis techniques (e.g. anomaly detection) directly on the original, raw data set while delivering fast response and 2) subset and focus on the region of interest in the data set by performing flexible spatiotemporal queries.

6. Conclusion

A spatiotemporal indexing approach is proposed to efficiently retrieve and process big array-based climate data in parallel using MapReduce. The spatiotemporal index bridges the gap between array-based data models and block-oriented HDFS storage models by linking the logical spatiotemporal information (space, time, and variables) to the physical location information (node, file, and byte). Based on the index, a grid partition algorithm was developed to optimize MapReduce processing performance by maximizing data locality and balancing the workload across cluster nodes.

The efficiency of the proposed approach was demonstrated by conducting spatiotemporal querying and processing on the NASA MERRA climate reanalysis dataset. Results show that the proposed approach can effectively work with the array-based data natively, and efficiently query and process big climate data without unnecessary disk reads. With the indexing approach, data- and compute-intensive operations can be intelligently decomposed into smaller tasks to be processed in parallel with high data locality (over 99%), which also results in a well-balanced workload across the nodes. Finally, a climate anomaly detection application based on the indexing approach was developed and deployed on the NASA GSFC Hadoop cluster. This application demonstrates that the indexing approach can be easily applied to efficiently solve real scientific problems.

6.1 Limitations and future research

There are several limitations of the proposed index, and future research is desired to further improve its performance as well as make it handle more generic scenarios.

First, *grid* is the atomic parallelization unit in the proposed spatiotemporal indexing approach. Generally, this approach works well with most array-based climate data (such as HDF and NetCDF) that has a relatively low spatial resolution (e.g. 1/2, 1/4 or even 1/8 degree grids). However, for high spatial resolution data sets (e.g. in the level of meters), the size of each global grid may reach several gigabytes or larger, this could impair parallelization performance. To overcome this limitation, we plan to improve the granularity of the index by further decomposing the large 2D grid into smaller tiles, and using *tile* as the atomic parallelization unit.

Second, for the grid partitioning strategy, each node is assumed to have the same available computing resources. However, each node in the cluster may have different hardware, or it may have been occupied by other jobs, so the available computing resources may be different for each node. Consequently, the proposed data partition algorithm could be improved to consider a possibly unbalanced computing resource distribution on the cluster when assigning grids.

Third, we will investigate more complicated spatiotemporal analytic use cases (such as Taylor diagram and climate ensemble analysis) based on the proposed index. In this case, some other issues need be taken into consideration. For example, based on the logical-physical relationship captured by the index, how to optimize the data *shuffle* between the *map* stage and *reduce* stage considering spatiotemporal principles (Yang et al., 2011)? Finally, the spatiotemporal index also works with Spark[3], and we will compare the performance of MapReduce and Spark on complicated use cases in the next step.

[3] http://spark.apache.org

References

Abouzeid, Azza, et al., 2009. "HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads." Proceedings of the VLDB Endowment 2.1: 922-933.

Berrick, S. W., Shen, S., & Ostrenga, D., 2008. Modern Era Retrospective Restrospective-Analysis for Research and Applications (MERRA) Data and Services at the GES DISC.

Bloom, S., A. da Silva, D. Dee, M. Bosilovich, et al., 2005. Documentation and Validation of the Goddard Earth Observing System (GEOS) Data Assimilation System - Version 4. Technical Report Series on Global Modeling and Data Assimilation 104606, v26.

Brown, Paul G., 2010. "Overview of SciDB: large scale array storage, processing and analysis." Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM.

Buck, J. B., Watkins, N., LeFevre, J., Ioannidou, K., Maltzahn, C., Polyzotis, N., & Brandt, S., 2011. Scihadoop: Array-based query processing in hadoop. In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (p. 66). ACM.

Chandola, V., Banerjee, A., & Kumar, V., 2009. Anomaly Detection: A Survey. ACM Computing Surveys (pp. 1-72). ACM

Das, M., & Parthasarathy, S., 2009. Anomaly detection and spatio-temporal analysis of global climate system. In Proceedings of the Third International Workshop on Knowledge Discovery from Sensor Data (pp. 142-150). ACM.

Dean, J., & Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

Duffy, D. Q., Schnase, J. L., Thompson, J. H., Freeman, S. M., & Clune, T. L., 2012. Preliminary Evaluation of MapReduce for high-performance climate data analysis.

Edwards, P. N., 2010. A vast machine: Computer models, climate data, and the politics of global warming (518 pp.). Cambridge, Mass: MIT Press.

Eldawy, A., & Mokbel, M. F., 2013. A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data. Proceedings of the VLDB Endowment, 6(12), 1230-1233.

Eldawy, Ahmed, and Mohamed F. Mokbel, 2015. "SpatialHadoop: A MapReduce Framework for Spatial Data." ICDE.

Eldawy, Ahmed, et al., 2015. "SHAHED: A MapReduce-based System for Querying and Visualizing Spatio-temporal Satellite Data." ICDE.

Geng, Yifeng, et al., 2013. "SciHive: Array-based query processing with HiveQL."Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on. IEEE.

Geng, Yifeng, Xiaomeng Huang, and Guangwen Yang, 2014. "Adaptive Indexing for Distributed Array Processing." Big Data (BigData Congress), 2014 IEEE International Congress on. IEEE.

Guttman, 1984. “R-trees: A Dynamic Index Structure for Spatial Searching,” in Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, pp. 47–57.

H. Fuchs, Z. M. Kedem, and B. F. Naylor, 1980. “On Visible Surface Generation by a Priori Tree Structures,” in Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, pp. 124–133.

J. L. Bentley, 1975. “Multidimensional binary search trees used for associative searching,” Commun. ACM, vol. 18, no. 9, pp. 509–517.

Li Z., Yang C., Sun M., Li J., Xu C., Huang Q., & Liu K., 2013. A High Performance Web-Based System for Analyzing and Visualizing Spatiotemporal Data for Climate Studies. In W2GIS, Lecture Notes in Computer Science, Volume 7820 (pp. 190-198). Springer Berlin Heidelberg.

Li Z., Yang C., Yu M., Liu K., Sun M. Enabling Big Geoscience Data Analytics with a Cloud-based, MapReduce-enabled and Service-oriented Workflow Framework, 2015, PloS one, 10(3), e0116781.

Lu, Ming-Yee, and Willy Zwaenepoel, 2010. "HadoopToSQL: a mapReduce query optimizer." Proceedings of the 5th European conference on Computer systems. ACM, 2010.

Malik, T., 2013, GeoBase: Indexing NetCDF Files for Large-Scale Data Analysis. In Big Data Management, Technologies, and Applications. Hu, W. C. (Ed.). IGI Global.

Mayer-Schönberger, V., & Cukier, K., 2013. Big data: A revolution that will transform how we live, work, and think. Houghton Mifflin Harcourt.

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 Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, pp. 322–331.

Overpeck, J. T., Meehl, G. A., Bony, S., & Easterling, D. R., 2011. Climate data challenges in the 21 st century. Science(Washington), 331(6018), 700-702.

R. A. Finkel and J. L. Bentley, 1974. Quad trees a data structure for retrieval on composite keys, Acta Informatica, vol. 4, no. 1, pp. 1–9, Mar. 1974

Rienecker, M. M., Suarez, M. J., Gelaro, R., Todling, R., Bacmeister, J., Liu, E., ... & Woollen, J., 2011. MERRA: NASA's modern-era retrospective analysis for research and applications. Journal of Climate, 24(14), 3624-3648.

Schnase JL, Duffy DQ, Tamkin GS, Nadeau D, Thompson JH, et al., 2014. MERRA analytic services: Meeting the big data challenges of climate science through cloud-enabled climate analytics-as-a-service. Computers, Environment and Urban Systems doi:10.1016/j.compenvurbsys.2013.12.003

Skytland, N., 2012. Big Data: What is NASA doing with Big Data today? *Open.Gov open access article*. http://open.nasa.gov/blog/2012/10/04/what-is-nasa-doing-with-big-data-today/. Last accessed on 03/15/2015.

T. Sellis, N. Roussopoulos, and C. Faloutsos, 1987. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects," Computer Science Department.

Wang, Daniel L., Charles S. Zender, and Stephen F. Jenks., 2008, Clustered workflow execution of retargeted data analysis scripts. Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on. IEEE.

White, T., 2009. Hadoop: the definitive guide: the definitive guide. " O'Reilly Media, Inc.".

Wu, Kesheng, et al., 2009. FastBit: interactively searching massive data. Journal of Physics: Conference Series. Vol. 180. No. 1. IOP Publishing.

Yang C., Wu H., Huang Q., Li Z., and Li J., 2011. Using spatial principles to optimize distributed computing for enabling the physical science discoveries, Proceedings of National Academy of Sciences, 108(14): 5498-5503