PAIRS (RE)LOADED: SYSTEM DESIGN & BENCHMARKING FOR SCALABLE GEOSPATIAL APPLICATIONS

C.M. Albrecht, N. Bobroff, B. Elmegreen, M. Freitag, H.F. Hamann, I. Khabibrakhmanov, L. Klein, S. Lu, F. Marianno, J. Schmude, X. Shao, C. Siebenschuh, R. Zhang

IBM T.J. Watson Research Center, 1101 Kitchawan Road, Yorktown Heights, 10598, NY, USA
{cmalbrec, bobroff, bge, mfreitag, hendrikh, ildar, kleinl, lus, fjmarian, shaox}@us.ibm.com,
{Johannes.Schmude, Carlo.Siebenschuh, Rui.Zhang}@ibm.com

KEYWORDS: big data analytics, ML, AI, distributed geo-spatial data structures, Hadoop, HBase, Spark, GeoMesa, PAIRS Geoscope

ABSTRACT:

In this paper we benchmark a previously introduced big data platform that enables the analysis of big data from remote sensing and other geospatial-temporal data. The platform, called IBM PAIRS Geoscope, has been developed by leveraging open source big data technologies (Hadoop/HBase) that are in principle scalable in storage and compute to hundreds of PetaBytes. Currently, PAIRS hosts multiple PetaBytes of curated and geospatial-temporally indexed data. It organizes all data with key-value combinations, performing analytics close to the data to minimize data movement.

1. INTRODUCTION

Remote sensing data useful for government, industry, and research is accumulating at an ever-increasing rate with satellite imagery alone exceeding several hundred TeraBytes per day [1]. Data archives are increasing even faster, because most applications require multiple data sets for context and comparison, especially for machine-learning models where historical data sets are required for training and validation. This rapid growth leads to the notion of *data gravity,* which results from two observations: first, remote sensing data is becoming too big to move so it stays where it is, and second, the value of data is increased by proximity to other data, so it attracts more.

Geospatial-temporal data is also complex due to numerous data formats, consisting of both vector data (points, polygons) and raster data (imagery) from multiple sensors like LiDAR (Light Detection And Ranging), RaDAR (Radio Detection And Ranging), and hyperspectral cameras, using multiple platforms like satellites, drones, cell phones, or IoT (Internet of Things) devices. Efficient use requires new technology designs. In this sense, data gravity also means that future platforms must heavily leverage public or private cloud computing to reach scalability, and they need to move the analytics and computation to the data rather than the traditional way, where the data are moved to the applications. Only then will this massive and rich source of remote sensing information be fully exploited in a timely manner.

Today, most geo-coded imagery is indexed only at the metadata level, but what if one wanted to search for certain data within an image? A most basic example would be to compare images of an airport from both a satellite and a cell phone at a series of different times. Today, an analyst might download all the related data to an application, open multiple files, extract the airport locations, overlay them with the cell phone data, and then run a comparison. In the system described here, such a task can be simplified with minimum data movement and file opening.

2. IBM PAIRS GEOSCOPE

To facilitate the improved discovery of information in remote sensing data, IBM has built a cloud-based platform that continuously ingests and curates more than 10 TeraBytes per day and presents it to multiple users for example with a web-based interface for search, analysis and viewing. This platform, called *PAIRS Geoscope*, has been developed using open-source big data technologies that can be scalable to hundreds of PetaBytes [1]. PAIRS is an acronym which stands for Physical Analytics Integrated Data Repository and Services [2, 3]. Instead of organizing plain files with a relational database, it stores data of any type (vector and raster) and complexity with key-value combinations, doing analytics close to the data to minimize data motion.

PAIRS Geoscope targets geospatial-temporal applications of interest to various industries, ranging from energy and utilities (e.g., when and where to trim vegetation to avoid storm outages), to agribusiness (when and where to buy or sell commodities), insurance (when and where are the highest risk assets) and government (when and where to optimally respond to a natural disaster). In PAIRS, there are hundreds of data layers spanning a wide range of spatial resolutions and decades in time. The architecture allows flexibility for data use, and has several key attributes: (1) it is cloud-based and parallel in data and compute for wide and dynamic scaling; (2) the data organization allows efficient time sampling and change or motion detection; (3) data pre-processing puts curated data within reach of a few user commands; and (4) familiar tools and an user interface can lower the threshold for widespread use.

Figure 1 illustrates the basic architecture of PAIRS Geoscope while Figure 2 shows a screenshot of the PAIRS UI. Data from various sources enter an ingestion and curation engine (2nd tier) for cleaning, filtering, re-projecting and resampling. Raster data are organized by pixels at the appropriate spatial resolution in a quad-tree hierarchy of scales.

[1] https://www.esa.int/Applications/ Observing_the_Earth/ Working_towards_AI_and_Earth_observation

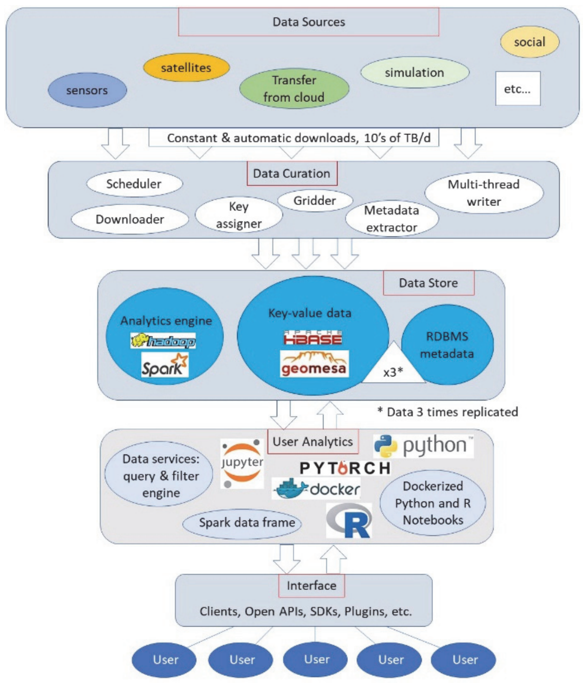


Figure 1: PAIRS Geoscope Architecture

Storage location for data values is determined by 16 Byte keys, which contain the geographical positions and times for the data in Z-order at scales ranging from 8.4 deg (940 km at the equator) to one-millionth of a deg (0.11 m), and with time resolved to the nearest second. Pre-processing like this is highly efficient when data are used multiple times. Generally, for individuals, the largest fraction of time spent with big data analytics is in the data preparation stage, and PAIRS does this once for all users.

The 3rd tier in Figure 1 is a massive distributed compute and data store based on the open source Apache Hadoop Data File System (HDFS) and Apache HBase [4]. Apache Hadoop distributes data across nodes in a computer cluster and transfers code into these nodes to process data locally; it automatically handles hardware errors, i.e. it is designed to be fault-tolerant. HBase is a key-value store that runs on Hadoop and consists of values identified by row, column, and qualifier keys. The key is a composite where the row contains the geospatial-temporal index mentioned above, the column contains the data layer, such as the red or near Infrared band for satellite data or the temperature layer for weather data, and the qualifier is a third quantity, such as the atmospheric pressure or altitude in the case of temperature. The value associated with the key is the data at that position and time. HBase intrinsically applies lossless compression by about a factor of 3, which compensates for the triple redundancy to ensure availability and fault tolerance. Another open-source technology commonly used for vector data is GeoMesa, which is able to use HBase as its backend to scalably store and process vector data or bounding box polygons for images [5, 6]. GeoMesa is an integral part of the system. In GeoMesa, distributed, scalable analytics using data from HBase are performed close to the data with Apache Spark [7]. Hadoop allows the use of the Apache Map/Reduce function in which queries or analytical tasks are done in parallel on separate nodes (mapped) and then merged into single or Master nodes (reduced). Map/Reduce gives highly scalable performance [8, 9]. Metadata are also ingested with the data values and stored in a relational database management system for easy retrieval and cross referencing of multiple sources.

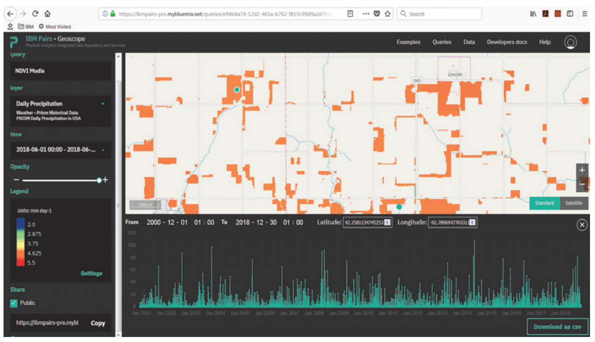


Figure 2: Example of data retrieval and geospatial-temporal browsing using today's PAIRS Geoscope web-based graphical user interface (GIU). For details and demos, visit: https://ibmpairs.mybluemix.net/

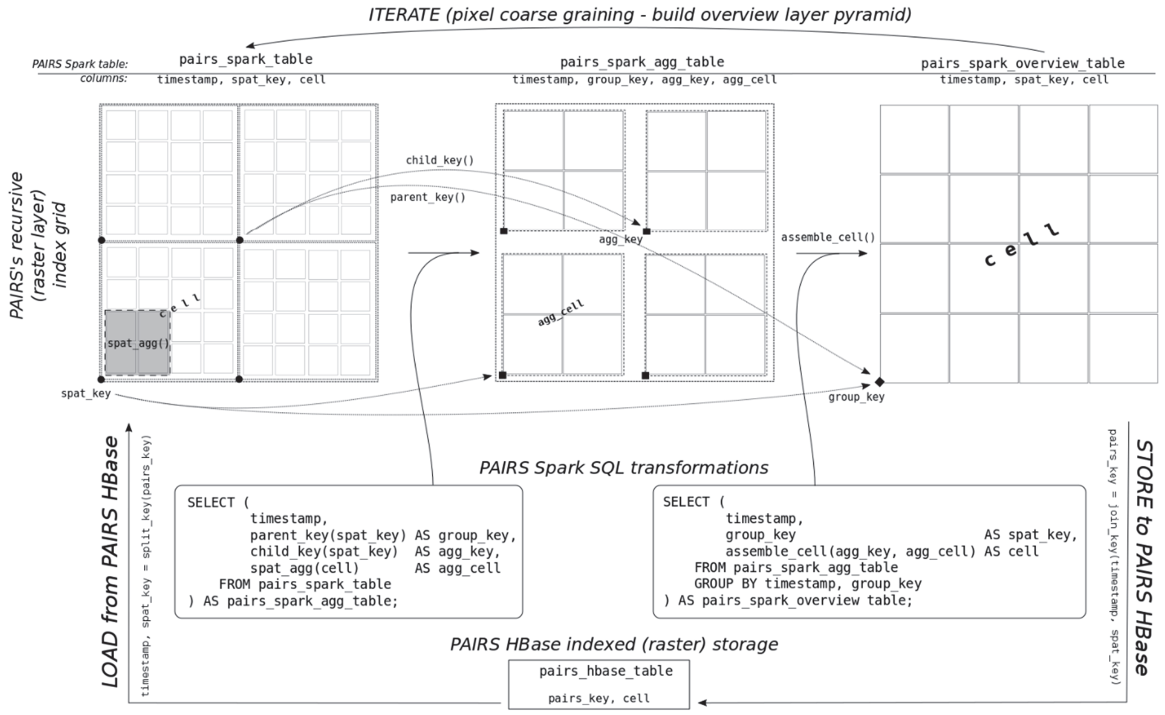
The 4th tier in Figure 1 is an analytics and data platform, which enables users via an interface (Tier 5) to interact with the system. Because of the common geospatial-temporal organization for all data, analytics involving syntheses and comparisons of data from different sources with different resolutions, source formats, and types (vector, raster, LiDAR point cloud, etc.) are efficient, scalable, and involve minimal data movement which is usually the bottleneck for big data analytics. Ready access to PetaBytes of data can give machine learning and artificial neural network techniques greater predictive accuracy. Inter-comparisons among diverse data layers may allow for discovery of subtle correlations. Rapid change detection becomes more feasible because the keys used for storage have the time dimension in their lowest bits, which means that all events happening at the same spatial location are stored on the same or nearby servers for rapid use. With PAIRS, producing a time sequence at a single position or searching for time-variability in a spatial region are simple operations (see below).

```
aoi=136           #Area of Interest = Iowa
query_json = {
    "layers" : [{
                "alias": "LandSat_B10",
                "type" : "raster", "id" : "49677",
                "output" : False,
                "temporal" : {"intervals" : [{"snapshot" : "2018-05-18T00:00:00.000Z"}]}},
                {
                "alias": "LandSat_DataQuality",
                "type" : "raster", "id" : "49679",
                "output" : False,
                "temporal" : {"intervals" : [{"snapshot" : "2018-05-18T00:00:00.000Z"}]}},
                {
                "alias": "Amb Temp",
                "type" : "raster", "id" : "49257",
                "output" : False,
                "temporal" : {"intervals" : [{"snapshot" : "2018-05-18T00:00:00.000Z"}]}},
                {
                "alias": "RadiativeHeatLoss",
                "expression":"$LandSat_DataQuality == 2720 ? line \
                5.67e-9*(math:pow($LandSat_B10,4)-0.787*math:pow($Amb_Temp,4)) : -9999"
                }
                ],
    "spatial" : {"type" : "poly", "aoi" : str(aoi)},
    "temporal" : {"intervals" : [{"snapshot" : "2018-05-18T00:00:00.000Z"}]}
}
query = paw.PAIRSQuery(query_json, PAIRS_SERVER, PAIRS_CREDENTIALS)
query.submit()
```

Figure 3: An example of a user-defined function submitted within the query

To summarize, PAIRS has the following technical merits:

(i) Most importantly, it is a big geospatial-temporal platform that indexes raster data on a pixel level. The basic technical challenges here lie in the design of the key and an appropriate


ITERATE (pixel coarse graining - build overview layer pyramid)
PAIRS Spark table:
columns:
pairs_spark_table
timestamp, spat_key, cell
pairs_spark_agg_table
timestamp, group_key, agg_key, agg_cell
pairs_spark_overview_table
timestamp, spat_key, cell
PAIRS's recursive (raster layer) index grid
child_key()
parent_key()
agg_key
assemble_cell()
cell
agg_cell
spat_agg()
spat_key
group_key
PAIRS Spark SQL transformations
LOAD from PAIRS HBase
timestamp, spat_key = split_key(pairs_key)
STORE to PAIRS HBase
pairs_key = join_key(timestamp, spat_key)
PAIRS HBase indexed (raster) storage
pairs_hbase_table
pairs_key, cell

```
SELECT (
        timestamp,
        parent_key(spat_key) AS group_key,
        child_key(spat_key)  AS agg_key,
        spat_agg(cell)       AS agg_cell
    FROM pairs_spark_table
) AS pairs_spark_agg_table;
```

```
SELECT (
        timestamp,
        group_key                         AS spat_key,
        assemble_cell(agg_key, agg_cell) AS cell
    FROM pairs_spark_agg_table
    GROUP BY timestamp, group_key
) AS pairs_spark_overview table;
```

Figure 4: Cartoon to illustrate how Apache Spark DataFrames can be used in IBM PAIRS in order to coarse-grain raster data.

value format to host the remote sensing data. Geospatial-temporal relational databases such as e.g. Postgres with PostGIS extension becomes inefficient at large data sizes (>10 Terabytes), or are based on files (GeoTIFF, ESRI Shapefile, etc.) in object store or file systems, whose content is very difficult and often slow to search. An example is the retrieval of a time-series of Earth surface reflectance from a satellite. Doing this today using satellite image repositories is quite involved on other platforms, and may require searching for available tiles, downloading, re-sampling, re-projecting, and opening each tile to extract pixels within an area of interest (aoi). To the best of our knowledge, alternative approaches which go beyond file-based indexing have been proposed and researched in academia, but still not demonstrated at scale on top of a total of PetaBytes of indexed data. PAIRS does this type of search quickly and with minimal data movement because only the aoi needs to be addressed using the spatial component of the key, and because time is the lowest order bit string in the key so that all times for a given position are located near each other in storage.

(ii) PAIRS deploys nested “resolution” levels to accommodate different spatial and temporal resolutions, thereby linking the different layers of geospatial-temporal information in a hierarchical tree. For the spatial domain, in the current system, the keys are arranged in Z-order, where the length of the key determines the resolution on a logarithmic scale, rapidly spanning the range from kilometers to centimeters with each additional bit. Extending this hierarchical organization to the temporal domain, e.g., adding additional information to the key or optimizing the order of the information in the key etc., are current research goals.

(iii) PAIRS aligns all data to a global grid and organizes it in linked layers at the time of data ingestion, versus on-demand. This enables scalable performance but also fast response to queries including multiple layers. For such computationally intensive data ingestion, the design of the key and its corresponding value, thereafter referenced as *supercell,* needs to be carefully picked. In fact, PAIRS enables analytics during the query, i.e., in addition to filtering and aggregation over space and time, user-defined functions allow queries to perform arbitrary math between the requested data layers. User defined functions could be as complex as a machine-learnt decision tree, which can be submitted within the query. A most basic example for such a user defined function is shown in Figure 3, where the radiative heat loss for the state of Iowa (aoi=136) is calculated using the Landsat 8 Band 10 (10.6-11.2 μm). Initially three data sets are being requested from PAIRS: Landsat Band 8 (id=49677), Landsat Data Quality (id=49679), and the ambient temperature from an IBM weather reanalysis data set id=49257). Two features are highlighted: First, we note that the output for each data set is “False”, which means they will not be downloaded. Instead of downloading we can calculate within the query the radiative heat loss for all high-quality pixels (=2720) using the Stefan-Boltzmann Law based on the difference of the surface and ambient temperature. Second, we note that spatial resolutions between the Landsat Band 10 (100 m) and the ambient temperatures, which come from a weather reanalysis model (4 km) are quite different but nevertheless this difference is being taken care of by the uniform spatial grids.

(iv) Other subtler features of PAIRS include Spark and GPU nodes on the PAIRS core cluster to host query results and enable users to interact with these results using a dockerized Python environment for customized analytics; this also minimizes data movement.

From a user's perspective, PAIRS offers four core services. On a very basic level, PAIRS provides *data services* for the many PetaBytes of curated information. On the next level, *search and query services* can be used to extract data and run analytics on it, such as filtering different layers of geospatial-information or aggregation. For users who want to develop custom analytics, PAIRS provides *analytics platform services* such as a dockerized Python environment to work with the queried data.

PAIRS Geoscope performs queries using a RESTful (Representational State Transfer) API employing the HTTP protocol that is agnostic to programming language. It allows one to define virtual layers that are computed and combined on-the-fly during query time. A comprehensive tutorial may be found here: https://pairs.res.ibm.com/tutorial/. For Python users, an open-source module wrapping the core PAIRS API is available here: https://github.com/ibm/ibmpairs. It loads query results into Python data structures at the user's end for further consumption and local processing. PAIRS is available commercially with a Fremium version at https://ibmpairs.mybluemix.net/.

Finally, users can upload their own data using the *data curation and ingestion services*. This allows users to run scalable searches and do analytics on their data along with the other multi-PetaBytes of PAIRS data. Data layers in PAIRS Geoscope can be shared, private or public, according to pre-established permissions.

fixed on HBase table creation. Values can be tensors, for example, or hyperspectral cubes, or Twitter commentary. Vector data such as discrete points and polygons, and point clouds, such as LiDAR scans, are stored in PAIRS Geoscope too, using a format that can be queried by Spark SQL. It is the query engine's responsibility to correctly interpret the incoming data using the associated Metadata or other information for data fusion. The format is fixed at ingestion by the PAIRS upload and curation module.

Figure 4 schematically depicts the integration of Spark DataFrames in combination with Spark SQL and HBase: A PAIRS raster query's result is placed into an HBase table that can be loaded as a Spark DataFrame employing Spark's HBase connector. The resulting table `pairs_spark_table` hosts supercells `cell` (4x4 pixels in the cartoon) with spatial `spat_key` and temporal `timestamp` indices derived from the PAIRS key `pairs_key`. Running the user-defined function `spat_agg()` on the supercells reduces them to 16x16 (2x2 in the cartoon). In order to assemble these back to 32x32 pixels by the user-defined function `assemble_cell()`, the spatial part of the PAIRS key needs to be truncated with `parent_key()` to `group_key`, on which a *GROUP BY*-SQL operation is applied. The resulting, spatially aggregated data can be readily stored back into HBase. A looping scheme allows to iteratively generate a pyramid of overview layers critical for accelerated analytics to be discussed next.

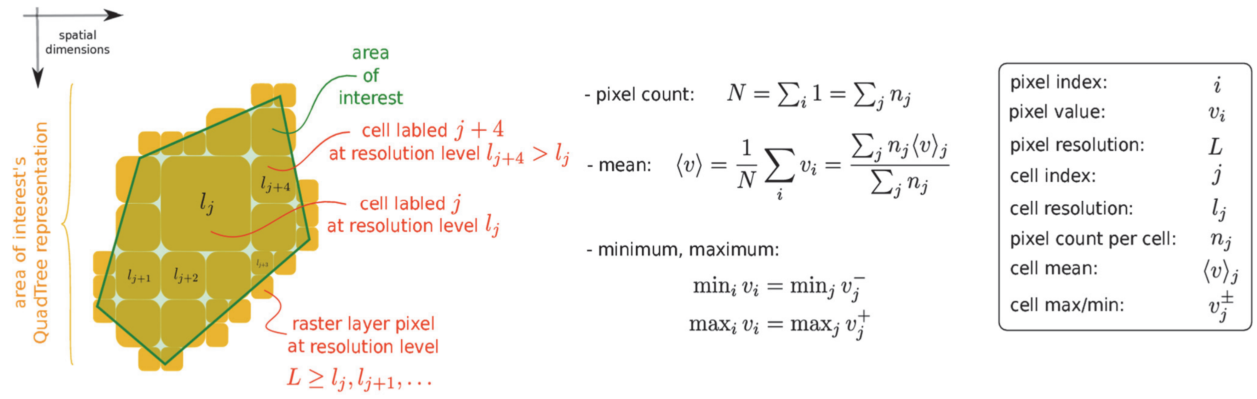

spatial dimensions
area of interest's QuadTree representation
area of interest
cell labled $j+4$ at resolution level $l_{j+4} > l_j$
cell labled j at resolution level l_j
raster layer pixel at resolution level $L \geq l_j, l_{j+1}, \dots$
l_j
l_{j+4}
l_{j+1}
l_{j+2}
l_{j+3}
- pixel count: $N = \sum_i 1 = \sum_j n_j$
- mean: $\langle v \rangle = \frac{1}{N} \sum_i v_i = \frac{\sum_j n_j \langle v \rangle_j}{\sum_j n_j}$
- minimum, maximum:
$\min_i v_i = \min_j v_j^-$
$\max_i v_i = \max_j v_j^+$
pixel index: i
pixel value: v_i
pixel resolution: L
cell index: j
cell resolution: l_j
pixel count per cell: n_j
cell mean: $\langle v \rangle_j$
cell max/min: $v_j^{\pm}$

Figure 5: Illustration of efficient analytics exploiting the generation of overview layers in PAIRS.

3. KEY-VALUE DESIGN & ANALYTICS

For raster data, the values associated with the three keys in HBase are packed in supercell blocks of 32x32 that come from nearby pixels in the image. This blocking makes the storage overhead required for the keys much smaller than the storage required for the data, saving storage overall. Blocking also significantly improves read/write speeds by a factor of about 50 compared to single-pixel keys. Blocking means that the keys assigned at ingest contain the position coordinates of the lower left corners of the supercells, rather than the coordinates of all individual pixels. Reading and writing processes all 1024 pixels at once, while the value assigned to each supercell key is the array of 1024 variables. In general, the value of the HBase (key1, key2, key3, value)-tuple might host any kind of geospatial-temporally tagged data structure. The keys and values can be any sequence of bytes, although key2 is limited to printable characters and needs to be

4. OVERVIEW LAYERS TO ACCELERATE ANALYTICS

Queries involving multiple layers can be sped up exponentially by constructing a pyramid with increasingly coarse-grained pixels commensurate with the PAIRS grid, built recursively and stored at the time of data curation by combining 2x2 pixels (cf. previous section) in each pyramid layer into one new pixel in the next-higher *overview* layer. Following a geometric series, the required storage is only 30% larger than the original volume. In mathematical terms: Given the size S of a raster layer, the generation of M overview layers would grow the required storage to:

$$S \rightarrow S\,(1+q+q^2+\ldots+q^M) = S\,(1-q^{M+1})/(1-q)$$

with $q=1/b$ where b=2x2=4. Therefore, the total amount of data needed is bounded from above ($M \rightarrow \infty$) by

$$S/(1-q) = S\,b/(b-1)$$

i.e. each pyramid of overview layers adds a fraction of $1/(b-1)=1/3$ to the HBase storage. Of course, skipping overview layers is an option to further minimize the extra data storage required. Systematically skipping k layers from overview to overview leads to a fraction $1/(b^k-1)$ which practically becomes zero for sufficiently large k.

The advantages for speed up are large, as this allows rapid cross-filtering of layers at different resolutions. Figure 5 shows a schematic of this hierarchically gridded pyramid (yellow boxes) for a geospatial area of interest (green polygon). For example, if one is interested in areas of layer A where the values in another layer B are larger than a given value X, e.g., "areas in a population density map where the average temperature exceeds a certain value", then a series of overview layers consisting of maximum values in B can exclude big patches where B<X. Thus, the high-resolution data do not need to be retrieved in full, but only the relevant parts of the overview layer and the associated parts of the high-resolution layer that satisfy the condition. Moreover, storing raster data statistics helps to compute certain aggregate values such as the mean of a raster layer's pixels in a given aoi (green polygon) as depicted in Figure 5: Recording the number of valid pixels n_j in an overview layer at PAIRS resolution level l_j prevents PAIRS from counting them down on the high-resolution pixel level L of the PAIRS raster layer. It allows one to more efficiently determine the total number N of pixels in the aoi. To identify the overview layers to be queried, the aoi is decomposed into a corresponding QuadTree (yellow boxes). In combination with the mean values $\langle v\rangle_j$ of the pixels in the overview layers, it is possible to efficiently reconstruct the pixel mean value $\langle v\rangle$ over the aoi without costly access to all pixel values v_i from the raster layer under consideration.

In summary, for accelerated analytics, a PAIRS raster layer can be enriched by adding several overview layer statistics with PAIRS resolution level difference k. The additional storage required is limited by

$$a\, m/(4^k-1)$$

from above relative to the raster data ingested. The multiplicative factor a (greater than or equal to 1) captures the fact that e.g. a raster layer with one byte per pixel might still require, for reasons of numerical precision, overview layers that store e.g. 4 or even 8 bytes single/double precision floating point numbers (cf. pixel mean), or integers (cf. pixel count).

5. PERFORMANCE BENCHMARK OF PAIRS

PAIRS Geoscope can be built with commodity hardware as speed is achieved by scale-out and fault tolerance through redundancy. Moreover, since PAIRS's big data software stack orchestrates Hadoop, HBase, Spark, and GeoMesa (which itself is based on Java and Scala) portability from one hardware configuration to another is straightforward. PAIRS Geoscope has also been successfully ported to IBM Power Systems.

Currently, PAIRS is running with Hadoop/HBase/Spark on about 80 nodes and upload functions plus curation processing on about 40 nodes. Each data center rack contains up to 24 data nodes with 10 GB/s network interconnect switches, while the racks are connected by an aggregator switch. No additional resources are needed for the analytics because it is done on the data nodes themselves - some directly on HBase tables and others loaded from HBase into Spark DataFrames.

The most basic query in PAIRS Geoscope is that of a time series for a single geospatial location. It retrieves all the data with the same position key, resulting in an efficient HBase scan due to the storage of temporal information in the HBase key's least significant bits. The current PAIRS Geoscope platform returns data for 14,000 timestamps at a given location for two physical quantities (i.e. 28,000 data points in total) in about one second. Carrying out the same procedure in a file stored system, one would need to open the files separately for each timestamp of the scene containing the location and extract the single pixel[2]. Furthermore, range queries based on polygon outlines for a single timestamp, i.e. maps, are easily set up and rapidly processed, too. For example, generating a raster image for the mean temperature in Iowa at a spatial resolution of 250m meters over around 4 months takes less than a minute to process.

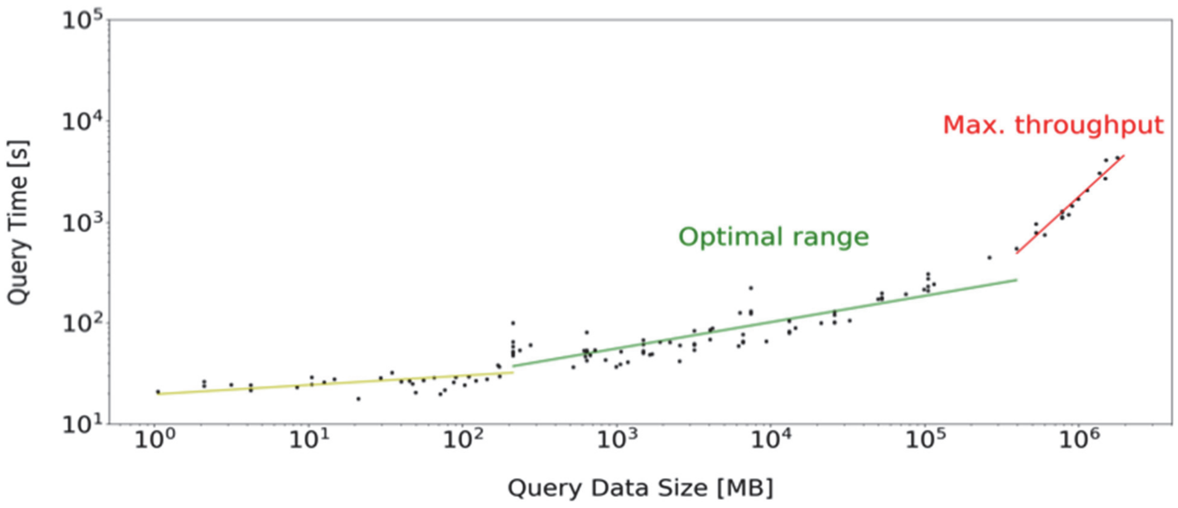


Figure 6: Query time versus query data size on PAIRS Geoscope, showing the increase from latency at low data size to a plateau of relatively constant query time for a wide range of data sizes in the scalable regime. Query time increases for larger data sizes when each storage node requires more than full utilization. The scalable range can be increased by adding more nodes.

[2] The test was performed for the full history of PRISM temperature and precipitation data.

This organizational feature of PAIRS Geoscope, with rapid processing of data along the time axis, is ideal for change or motion detection in large spatial fields. Field size does not affect processing time because the spatial dimensions are scaled out in parallel, which is a basic feature of Z-ordering in a quad-cube. Discovery Notification would be an example of such scaled analytics.

Latency and scaling are key for performance. Figure 6 provides insights of how to think about the performance of the PAIRS platform or other similar platforms for that matter. The graph of a piecewise regression analysis is plotted and shows the estimated data retrieval time against the processed data size. Three data size regimes are apparent. Firstly, if few data values are retrieved, the performance is limited by *latency* which is determined by the overhead of the parallelization (i.e. the Map/Reduce job). Secondly, as the processing time increases with query size, latency becomes negligible. The interval of optimal query sizes is referred to as the *optimal range*. In it, the data retrieval time is only weakly dependent on the actual data size as the number of parallel map jobs increases accordingly to the requested data size. Finally, as data size grows even larger, parallelization capabilities are depleted and cause the data retrieval time to be highly sensitive with respect to data size once again.

Empirically speaking, the first segment's regression function characterizes the *latency* while the second characterizes scalability. Remarkably, the first segment's slope is rather flat indicating that retrieval time for small queries is almost independent of data retrieval size. On the other hand, the slope of the regression function within the optimal range characterizes the *scalability* and is given by 0.26. Although this value is significantly different from zero with a p-value of $2.2 \cdot 10^{-16}$, an adjusted R-squared of 72% indicates that data size is not the sole contributor to this data retrieval time. Therefore, the number of involved data layers or timestamps could play a crucial role too. Moreover, scalability remains a relative term. For example, even for large query sizes in the range of one TeraByte, retrieval rates at around 540 MB/s were consistently observed proving that PAIRS is rapid across the board.

However, not only size matters but complexity too. PAIRS Geoscope can respond to complex queries such as "Calculate the average precipitation in the 2017 growing season on all planted corn fields in the contiguous United States where the Normalized Difference Vegetation Index (NDVI) in the first 15 days of June was larger than 0.5." Such a query involves 216 data layers, 22 TB of processed data (temporarily matched to the highest resolution data set) and three different data sets: MODIS (Satellite), Precipitation (weather service), and USDA Cropscope (government). It recently took 693 seconds, which is 31.7 GB/s of data analysis or 2.7 PB/day equivalent. On the current platform, five such queries can be run in parallel without performance degradation, amounting to 13.7 PB/day. The output of this query is also shown in Figure 2.

Although PAIRS Geoscope uses HBase which does not allow SQL for queries, the capability to use SQL is present as a PAIRS query can be directly exported into a Spark DataFrame which is similar to a distributed relational database table. The Spark SQL functionality then allows the use of well-known SQL expressions and user-defined functions on the DataFrame. There can also be multiple result tables with, e.g., different resolutions which can be joined using SQL commands and the geospatial-temporal key indexing.

The Spark DataFrame is distributed throughout the memory of the PAIRS computer cluster. Using PySpark, the DataFrame may be accessed via a (local) Python Jupyter notebook. The link between the user's local machine and PAIRS is established through a RESTful API employing an Apache Livy server (https://livy.apache.org/). No downloading of data to the local machine is required as the notebook instructions operate on the data where they reside in the database.

6. CONCLUSION

PAIRS Geoscope is a curation, storage and analytics platform for discovery and collaboration that can incorporate a wide variety of geospatial-temporal data. Given its software design, PAIRS is able to deliver the following unique features: (1) scalability to accommodate multiple sources of real-world data efficiently sorted by position and time; (2) ability to compare, combine, filter, sort and display multiple data sets simultaneously for correlation discovery and change detection; (3) PetaByte processing close to the data for advanced analytics, and (4) familiar user interfaces involving common tools and software in a combination of private, collaborative and public work spaces.

7. REFERENCES

[1] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, *et al.*, "Dynamo: amazon's highly available key-value store," in *ACM SIGOPS operating systems review*, 2007, pp. 205-220.

[2] L. J. Klein, F. J. Marianno, C. M. Albrecht, M. Freitag, S. Lu, N. Hinds, *et al.*, "PAIRS: A scalable geo-spatial data analytics platform," in *Big Data (Big Data), 2015 IEEE International Conference on*, 2015, pp. 1290-1298.

[3] S. Lu, X. Shao, M. Freitag, L. J. Klein, J. Renwick, F. J. Marianno, *et al.*, "IBM PAIRS curated big data service for accelerated geospatial data analytics and discovery," in *Big Data (Big Data), 2016 IEEE International Conference on*, 2016, pp. 2672-2675.

[4] L. George, *HBase: the definitive guide*: " O'Reilly Media, Inc.", 2011.

[5] J. N. Hughes, A. Annex, C. N. Eichelberger, A. Fox, A. Hulbert, and M. Ronquest, "Geomesa: a distributed architecture for spatio-temporal fusion," in *Geospatial Informatics, Fusion, and Motion Video Analytics V*, 2015, p. 94730F.

[6] A. Hulbert, T. Kunicki, J. N. Hughes, A. D. Fox, and C. N. Eichelberger, "An experimental study of big spatial data systems," in *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 2664-2671.

[7] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," *HotCloud,* vol. 10, p. 95, 2010.

[8] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM,* vol. 51, pp. 107-113, 2008.

[9] J. Ekanayake, S. Pallickara, and G. Fox, "Mapreduce for data intensive scientific analyses," in *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, 2008, pp. 277-284.