# Benchmark Comparison of Cloud Analytics Methods Applied to Earth Observations

**Christopher Lynnes[1], Michael M. Little[1], Thomas Huang[2], Joseph Charles Jacob[2], Chaowei Phil Yang[3], Mahabaleshwara Hegde[4], Hailiang Zhang[4]**

[1]NASA/Goddard Space Flight Center (NASA Civil Servant).

[2]NASA/Jet Propulsion Laboratory (Employed by California Institute of Technology).

[3]George Mason University.

[4]NASA/Goddard Space Flight Center (Employed by Adnet Systems Inc).

Corresponding author: Christopher Lynnes (christopher.s.lynnes@nasa.gov)

**Key Points:**

- In order to take full advantage of cloud computing for analyzing Earth Observation data, the data must typically be preprocessed, reorganized, and stored in a more tractable form.
- A variety of quite different architectures ("Analytics Optimized Data Stores") can provide 1-3 orders of magnitude improvement in analysis speed over the original data form.

## Abstract

Earth Observation data are a vital resource for studying long term changes, but the large data volumes can be challenging to analyze. Time series analysis in particular is hampered by the typical thin-time-slice file organization. We examine several potential solutions inspired in large part by the data-parallel methods that have arisen with cloud computing. These solutions include various combinations of data re-organization, spatial indexing, distributed storage and pre-computation that we term "Analytics Optimized Data Stores" (AODS). We find that even simple solutions (such as a data cube) produce more than an order of magnitude improvement; the best provide two to three orders of magnitude improvement. The most performant solutions have tradeoffs in terms of generality or storage footprint, but may nonetheless be useful components in data analytics frameworks where performance is critical.

## 1 Introduction

With the continuing growth of Earth Observation (EO) data volumes, the analysis of and knowledge extraction from these data continues to be a challenge. However, the relatively recent development of cloud computing offers some hope that analysis can be tractable. Furthermore, the availability to anyone of massively parallel cloud computing opens possibilities that formerly were available only to authorized supercomputer users. This opportunity comes at a cost, though, because massive parallel computing in the cloud is typically only possible for data-parallel computing. As a result, some problems, such as tightly coupled models, may never be tractable in cloud computing. However, a large class of problems, particularly those where spatial regions or time intervals can be computed independently, lend themselves well to cloud computing. Still, there is the need to apportion the data in such a way that many compute nodes can be brought to bear on them simultaneously and efficiently.

By way of illustration, let us examine an existing data analysis system to identify how we might transform an intractable problem for on-premises systems into a solved problem for cloud computing. The Geospatial Interactive Online Visualization and Analysis Infrastructure (Giovanni) provides an online capability to do basic statistical analysis of EO data from NASA satellites, as well as some assimilation model outputs. It provides over 20 different analysis functions, but by far the two most popular services are time-averaged maps and area-averaged time series. Giovanni has over 30,000 active users and has been cited or acknowledged in over 1700 publications. However, its popularity can sometimes be a drawback. The current server for Giovanni consists of a basic Linux 32-core multiprocessor. On occasion, the user load exceeds capacity, slowing down all users of Giovanni and sometimes requiring a cold reboot to restore service, which terminates user analysis workflows without result. The performance is limited not only by the fixed on-premises capacity, but also by the coarse parallelism used, namely on a per-variable basis.

Satellite datasets can be as long as 20 years, with some of the assimilation models covering more than 65 years. Both the satellite and model datasets are typically produced in time order, that is, by computing over the whole spatial extent for a given time period. Because of this processing mode, Earth Observation and assimilated model datasets tend to be structured with a small number of time steps per file, and often only one time step. This means that any long time series analysis involves opening and closing a large number of files. For instance, in the case of the 39-year North American Land Data Assimilation System (NLDAS) model output is stored as over 340,000 individual files. The overhead of opening and closing files is exacerbated slightly by the complex structure of the storage format, Hierarchical Data Format.

Simply doing a point extraction over the whole time series can take several hours due to this thin-time-sliced organization. As a result, Giovanni limits the number of data values for which a user can request analysis. For an area-averaged time series of the full spatial extent of NLDAS hourly data, the limit is a mere two years of the nearly 40 year time record, clearly constraining what a science user can do. Although more sophisticated parallelized code might help this somewhat, improvement is likely to be limited. The high number of file operations is also likely to tax the disk subsystems, either at the hardware level or the kernel/filesystem level.

Attempts have been made to ameliorate this, most notably the construction of datasets where each file represents the whole time series for a single grid point, known as "Data rods" (Teng et al., 2016), which essentially turn the data organization at right angles. Data rods show promise for fast processing in the time direction, but of course do not work well for synoptic analysis. Also, this approach represents a data management challenge, with respect not only to the dual copies of data to store, but also the handling of active data streams that are appending to each data rod file each day.

Cloud computing provides several capabilities that show promise of breaking the logjams often faced by on-premises data analysis systems. The most obvious is the ability to scale up the number of processes working on analysis workflows. However, this is of little value if the initial data retrieval is itself a bottleneck. Fortunately, cloud computing also has several ways of scaling past this bottleneck. The original solution to this was highly distributed file systems, coupled with processors near the data. The development of MapReduce (Dean and Ghemawat, 2008) and the Hadoop Distributed File System (HDFS) (Shvachko et al., 2010) made accessible the ability to scale out to thousands of processors, each working on a small portion of data has been a fundamental capability of cloud computing. This was followed shortly by highly distributed databases in the cloud. In one example, the Hive system (Thusoo et al. 2009) provides a SQL-interface to MapReduce frameworks working on data in distributed filesystems such as HDFS. Other databases in the cloud ecosystem include columnar NoSQL databases, such as MongoDB. Databases can be used in two subtly different ways. In the first mode, the database is used to distribute "chunks" of data for moderately fine-grained data parallelism, but the chunks are retrieved as-is for processing by code. Alternatively, superfine data parallelism can be achieved by storing individual data values within the database, which allows some processing to occur within the database query processing.

Thus there are a variety of ways to partition, organize and store data in the cloud to enable highly data-parallel processing. For data systems that seek to offer analytics capabilities, this gives rise to some significant challenges. The first is committing to reorganize and write the data in a structure that is optimized for analytics. The second is deciding whether to store this analytics-optimized copy, and for how long; unlike on-premises data storage hardware, data storage costs in the cloud are typically a direct function of how long they are stored. The third challenge is that the analytics software typically needs to be written (or rewritten) to work with the particular data organization or storage format. For some cases, it may be possible to insulate the code from the variations among the schemes by using versatile data structure frameworks such as Python Data Analysis Library (pandas) (McKinney, 2015) or Xarray (Hoyer and Hamman, 2017). However, the storage schemes are sufficiently diverse that the frameworks either do not cover them all yet, or if they do, lose significant performance by not optimizing to work with a particular storage structure. Thus, it is useful to understand the performance properties of various forms of AODS in order to decide which scheme to use for a given application, and because once an AODS has been implemented for a given dataset and/or analytics framework, changing to another can incur significant costs. We describe the methodology and results of investigating the performance of 7 types of AODS for 3 different geographic extents.

## 2 Experimental Setup

We designed a simple experiment to examine how much processing improvement could be obtained from a variety of AODS. We used a common analysis problem, the area-averaged time series that presents a challenge for on-premises analysis. In this calculation, cell values for a geographic projection are averaged with a weighting of cosine(latitude). The dataset selected was daily gridded aerosol optical depth from the Moderate Resolution Imaging Spectro-radiometer (MODIS) flying on NASA's Terra satellite. The data variable can be extracted from the MODIS/Terra Aerosol Cloud Water Vapor Ozone Daily L3 Global 1 degree climate model grid (Platnick, 2015). The time period selected covered from March 1, 2000 to February 29, 2016. Fig. 1 shows the global time series over this time period. In order to distinguish between the contribution to elapsed time from data retrieval vs. calculation, we tried three cases: a global average, an average over the state of Colorado, and an average for the town of Boulder, which essentially amounts to a simple point time-series extraction. We used the Giovanni system to establish a single-threaded on-premises baseline. The algorithm computation times were taken from just the analysis step in the lineage and did not include the data query, preparation or visualization steps that form a typical Giovanni request, in order to focus on the effect of the data organization and structures.

## 3 AODS Candidates

The simplest of the AODS we investigated, Architecture #1, was a pre-aggregation of the individual data files into larger files with multiple time steps per file. This is a highly simplified form of the "data cube" approach used in both Google Earth Engine (Horelick et al., 2017) and the Committee for Earth Observation Satellites (Lewis et al., 2017) to simplify time-series analysis of satellite swath data. In our case, we do not need to interpolate or average any of the data, as the level 3 MODIS AOD is already on a regular grid. Nonetheless, the data cube approach can potentially improve performance by dramatically reducing the file open and close operations. To compute the area averaged time series, we use ncwa (weighted average) and ncrcat (record concatenation) from the NetCDF Command Operators (NCO), a powerful computation package written in C++ for network Common Data Form and Hierarchical Data Format files (Zender, 2008). Fig. 2 shows an experiment conducted on an Apple Airbook with solid state drives (SSD) for different aggregation levels, where the X axis indicates how many time steps are included per file. (For the 1000-timestep case, the last file has only 789 timesteps.) The performance of the single-timestep files is on the same order of magnitude, as expected.

The performance improvement with increasing aggregation size is nearly an order of magnitude for the global case and even greater for the spatial subset cases, even without invoking any multiprocessing capabilities. To look for further improvement with multi-processing, we selected a chunk size of 1000 and executed a 6-way fork (one for each time chunk) on a single node in the cloud, using an r4_2xlarge compute node in the Amazon Web Services cloud. The results were then concatenated into a time series using ncrcat.

We also investigated two AODS based on distributed filesystems. Architecture #2 was a simple HDFS-based system using Spark for distributed computation. Architecture #3 (Fig. 3) was based on Climate Spark (Lu et al., 2017), which saves the data as netCDF files within HDFS but adds an indexing scheme of spatio-temporal information of the data, allowing for faster retrieval, particularly when dealing with spatial subsets. Since netCDF is a common standard format for EO data, this has the potential upside of using archival versions of the data (though for expedience in this case, the data were converted from Hierarchical Data Format version 4 to netCDF).

We also investigated four different database-based AODS, two of which represent hybrids of a sort. Architecture #4 uses Hive (Capriolo et al., 2012) on top of HDFS with schema-on-read and Spark as a processing engine. Architecture #5 is the NEXUS architecture (Fig. 4 and Fig. 5), which uses a Cassandra database into which space-time chunks of data were stored for quick retrieval (Huang, et al. 2018). In the above database-oriented architectures, the database is primarily used for storage and retrieval with computation happening in Spark.

In Architecture #6, data were stored in a Parquet (Le Dem, 2013) format in Web Object Storage. The data were then accessed via the Athena service in the cloud, which provides a schema-on-read SQL-like interface to the data. Finally, in Architecture #7 data were stored as individual values in MongoDB. In this architecture, simple processing can actually be embedded in query aggregation functions.

Clearly, the need to run the AODS in a parallel processing architecture brings with it a set of architectural complications that may be difficult to tease out. It is particularly difficult to decouple the computation algorithms from the AODS aspects of the architecture because the ways that data are partitioned and accessed are so different. However, we are less interested in picking a winner than in investigating the range of variation and possible causes of variation. The hardware platforms are shown in Table 1. Note that even though the on-premises Giovanni runs on a multi-processor server, we list the number of nodes and cores as one (1) because it runs in single-threaded mode. Ideally, the tests would also be run on a common set of hardware. However, the different software architectures may have different hardware requirements to run in a performant configuration. The Athena-based architecture represents a "pure" AODS, in that the computational hardware is completely abstracted and indeed impossible for users of the service to determine. Instead customers are billed by the amount of data scanned in response to a request.

An important aspect to several of the AODS is the pre-processing performed on the data during the Extract-Transform-Load process, which may include some precomputation. The NEXUS system precomputes some area statistics, for example. For the Athena+Parquet architecture, we tried two different methods. In the first, a spatial index (SI) was constructed to simplify the query. In the second, a running sum (RS) was computed as a function of latitude and longitude and then stored in the Parquet AODS. This dramatically simplifies the run-time computation of the area-averaged time series to a simple difference operation between the running sums at the end time and beginning time. This not only produces a faster response (usually) but also reduces the amount of data scanned from 350 MB in the SI case to under 4 MB in the RS case. However, it does carry significant drawbacks in terms of data management as the running sums must be recalculated and updated in the AODS if the source data are replaced through reprocessing, and its optimization capabilities are limited to a small set of computation types.

## 4 Experimental Results

The elapsed times for the three geographic extents are shown for each architecture in Fig. 6. Two of the architectures were tested in two different modes: the Data Cube was tested with and without forked multi-processing. For the Athena+Parquet on-premises Giovanni, the three geographic extents take similar amounts of time, indicating that the overriding factor is the file operations. However, when the files are pre-aggregated into data cubes, a dramatic improvement is achieved for all three geographic extents with little increase in architectural complexity. Furthermore, adding rudimentary fork-based multiprocessing on a single node achieves further time savings, resulting in more than an order of magnitude for the global case and two orders of magnitude for the spatial subset cases.

Interestingly, the two distributed-filesystem parallel architectures have similar results for the Global and Colorado geographic extents, suggesting that file operations are still a limiting factor in these cases. On the other hand, the various database architectures show a fair degree of separation among the geographic extents, in keeping with the lack of explicit file open/close operations in these architectures.

On the whole, the filesystem architectures tend to perform roughly equal to the database architectures for the global case. The database architectures tend to be more efficient for the Colorado and Boulder case with the maximum improvement being the MongoDB case, which outperforms all other architectures for the two small-area regions. This suggests that for retrieval of modest amounts of data from much larger overall datasets, storing the data values individually in the database is the most performant. However, an important caveat to this is that the storage footprint of the dataset in MongoDB was as much as 50X the storage footprint of the other architectures.

The most important result is that all of the architectures improve significantly over the on-premises Giovanni case with improvement ranging from one to more than three orders of magnitude, but most of them falling between one and two orders of magnitude. Thus, in many cases implementation decisions among them are likely to be equally driven by cost, data management and architectural considerations. For instance, the fastest architecture, MongoDB, is also by far the least parsimonious in storage footprint and thus cost. It may thus be particularly appropriate for cases where speed is critical and data system managers are willing to pay a premium for it, such as in highly interactive data exploration or analysis applications. Alternatively, data managers might wish to maintain two copies of data with different running sums in the Athena+Parquet case: more expensive than a single copy, but less expensive than the 50X premium. At the other end, the data cube architecture is attractive for its simplicity and generality. The software needed to create access the data cubes is extremely simple and access can be through a variety of languages and methods. Most of the Spark-based architectures fall in between these extremes. It should be noted that the computations used in this experiment were quite simple; it is possible that more complicated algorithms would benefit more from the massive parallelism available from Spark and similar frameworks.

## 5 Conclusions

Earth Observation and related model output data lend themselves to a variety of analyses that rely on long time series: long-term trends, phenology, climatologies, and diurnal variations to name but a few. It is thus ironic that their mode of production so often leads to a data organization that is at odds with time series analysis. Data archives are often reluctant to modify the data given to them for safekeeping. However, cloud computing provides a flexible set of storage and compute options that provide data system architects with the possibility of decoupling the archive form of the data from the form in which they are analyzed. This brings with it a number of challenges:  verifying that the analysis-optimized data represent the same content as the archive copy; managing live data collections as data arrive; and maintaining the provenance chain.  Also, most of the AODS architectures rely on the data residing on relatively expensive spinning or solid-state disk, rather than the Web Object Store that the archive copies are likely to inhabit. Thus, data managers will need to be selective in deciding which data variables should be presented in analysis-optimized form. Given the cloud paradigm of costing data storage as a function of residence time, some data variables may even be temporarily staged to AODS to be rotated out after the user community has had a chance to exploit them in favor of the next data variables. This staging and de-staging of particular variables could even be triggered automatically, on the occurrence of some event (e.g., a volcanic eruption) or alternatively, tied to a particular funding cycle on a given topic. It is even conceivable that the particular AODS form might be chosen according to the expected community or usage. For instance, if combining data with, say, a data cube from Australia, then a data cube container might be easier for data interoperability than Spark working over HDFS.  This becomes ever more likely as scientists combine data from a variety of providers across different organizations, agencies or nations, which may have a different preferred form of AODS.

## References

Capriolo, E., Wampler, D., & Rutherglen, J. (2012). Programming Hive: Data warehouse and query language for Hadoop. " O'Reilly Media, Inc.".

Dean, J., & Ghemawa, S. (2008), MapReduce: simplified data processing on large clusters. *Commun. ACM* 51(1) , 107-113. doi:10.1145/1327452.1327492

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017), Google Earth Engine: Planetary-scale geospatial analysis for everyone, *Remote Sensing of Environment*, 202, 18-27, doi:10.1016/j.rse.2017.06.031.

Hoyer, S. and Hamman, J. (2017). xarray: ND labeled Arrays and Datasets in Python. *Journal of Open Research Software*, 5.

Hu, F., Yang, C. P., Duffy, D., Schnase, J. L., & Li, Z. (2016), ClimateSpark: an in-memory distributed computing framework for big climate data analytics." *AGU Fall Meeting Abstracts*.

Huang, T., E. M. Armstrong, F. R. Greguska, J. C. Jacob, N. T. Quach, L. J. McGibbney, V. M. Tsontos, B. D. Wilson, S. R. Smith, M. A. Bourassa, J. L. Elya, S. J. Worley, T. Cram, Z. Ji, C. Yang, Y. Jiang, and Y. Li (2018). High Performance Open Source Platform for Ocean Sciences., *2018 Ocean Sciences Meeting*, Portland, OR.

Le Dem, J., (2013), Parquet: Columnar storage for the people, *Strata/Hadoop World,* New York City, https://www.slideshare.net/julienledem/parquet-stratany-hadoopworld2013, accessed 26 March 2018.

Lewis, A., Oliver, S., Lymburner, L., Evans, B., Wyborn, L., Mueller, N., Raevksi, G., Hooke, J., Woodcock, R., Sixsmith, J., Wu, W., Tan, P., Li, F., Killough, B., Minchin, S., Roberts, D., Ayers, D., Bala, B., Dwyer, J., Dekker, A., Dhu, T., Hicks, A., Ip, A., Purss, M., Richards, C., Sagar, S., Trenham, C., Wang, P., & Wang, L.-W. (2017), The Australian Geoscience data cube — foundations and lessons learned, *Remote Sensing of Environment*, 202, 276-292, doi:10.1016/j.rse.2017.03.015.

Liu, Z., & Acker, J. (2017), Giovanni: The bridge between data and science, *Eos, 98,*doi:10.1029/2017EO079299.

McKinney, W. and Team, P.D., 2018. Pandas—Powerful Python Data Analysis Toolkit. https://pandas.pydata.org/pandas-docs/stable/pandas.pdf.

Platnick, S. (2015), MODIS/Terra Aerosol Cloud Water Vapor Ozone Daily L3 Global 1Deg CMG, NASA Level-1 and Atmosphere Archive & Distribution System (LAADS) Distributed Active Archive Center (DAAC), Goddard Space Flight Center, Greenbelt, MD, doi:10.5067/MODIS/MOD08_D3.006.

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010), The Hadoop Distributed File System, *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 1-10, doi: 10.1109/MSST.2010.5496972.

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., & Murthy, R. (2009), Hive: a warehousing solution over a map-reduce framework. *Proc. VLDB Endow.* 2, 1626-1629. doi:10.14778/1687553.1687609.

Teng, W., Rui, H., Strub, R., & Vollmer, B. (2016), Optimal reorganization of NASA Earth science data for enhanced accessibility and usability for the hydrology community. *Journal of the American Water Resources Association (JAWRA)* 52(4), 825–835, doi:10.1111/1752-1688.12405.

Zender, C. S. (2008), Analysis of self-describing gridded geoscience data with netCDF Operators (NCO), *Environmental Modelling & Software* 23(10-11), 1338-1342.

| | **Architecture** | **Nodes** | **Cores/Node** | **Total Cores** | **RAM/Node** | **Storage** |
|---|---|---|---|---|---|---|
| 0 | Giovanni | 1 | 1 | 1 | 32 | Disk |
| 1 | Data Cube | 1 | 8 | 8 | 61 | SSD |
| 2 | Spark+HDFS | 19 (1) | 8 | 8 | 61 | Disk |
| 3 | ClimateSpark | 19 (1) | 12 | 228 (12) | 24 | Disk |
| 4 | Spark+Hive | 19 (1) | 12 | 228 (12) | 24 | Disk |
| 5 | Spark+Cassandra (NEXUS) | 1 | 64 | 64 | 122 | SSD |
| 6 | Athena+Parquet | not available | | | | WOS |
| 7 | MongoDB | 19 (4) | 12 | 228 (12) | 24 | Disk |

**Table 1**. Hardware systems supporting the AODS architectures. Numbers in parentheses indicate the number of nodes (out of the total) that act as head nodes. “SSD” indicates Solid State Drives and “WOS” indicates Web Object Storage.

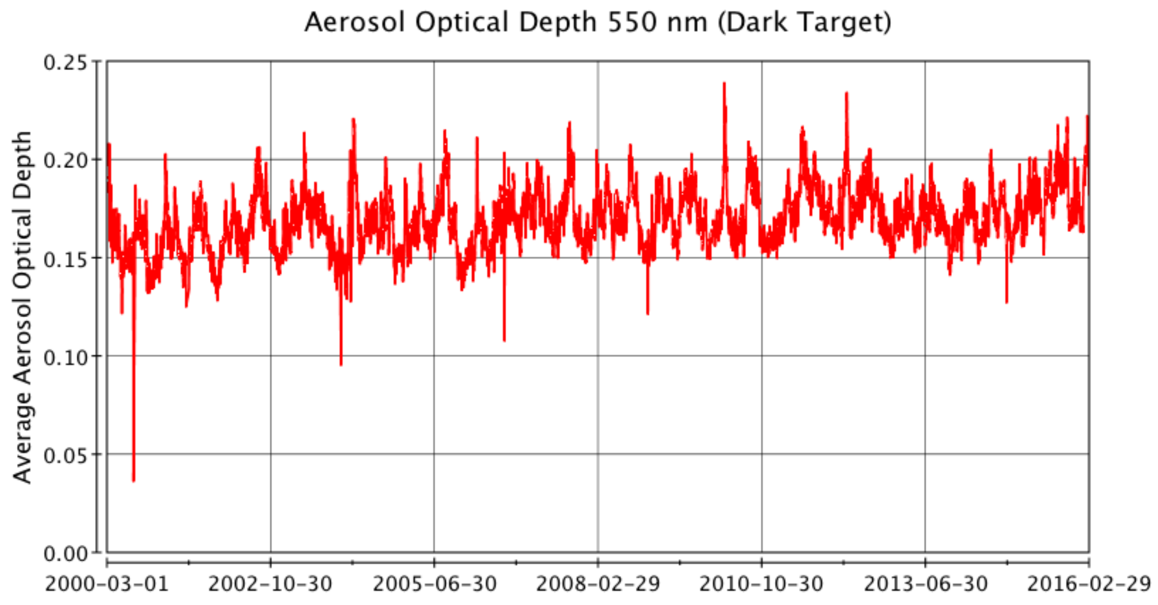


**Figure 1.** Area-averaged time series of MODIS/Terra Aerosol Optical Depth from 2000-03-01 to 2016-02-29.

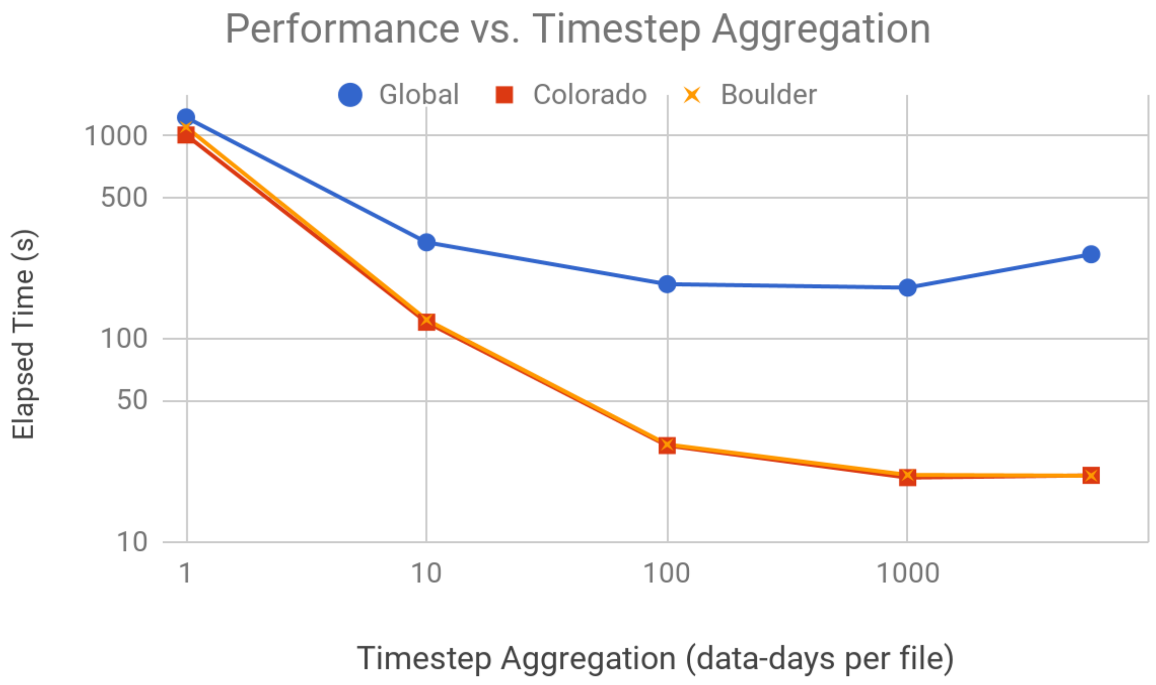


**Figure 2.** Elapsed time for computations of area-averaged time series of MODIS/Terra AOD for global (circle), Colorado (square) and Boulder ('X'). (The latter two times are virtually identical for all levels of aggregation.)

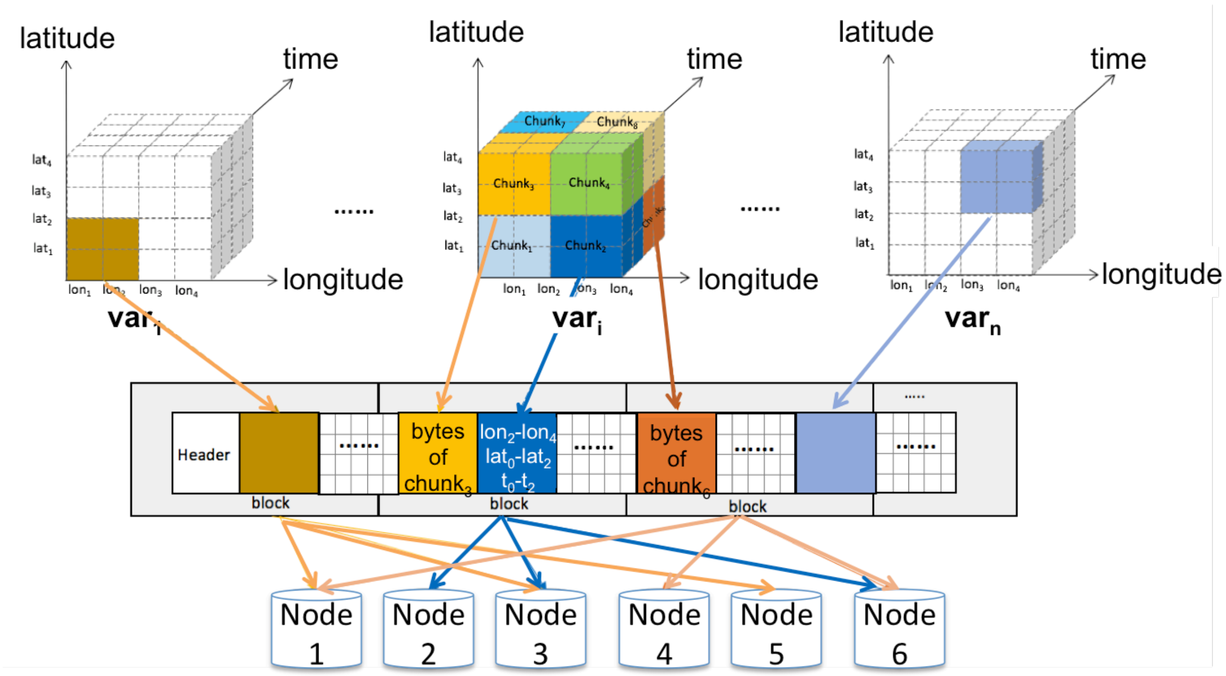


**Figure 3.** Depiction of ClimateSpark architecture. An external spatio-temporal index of data chunks across the HDFS helps optimize retrieval for analysis.

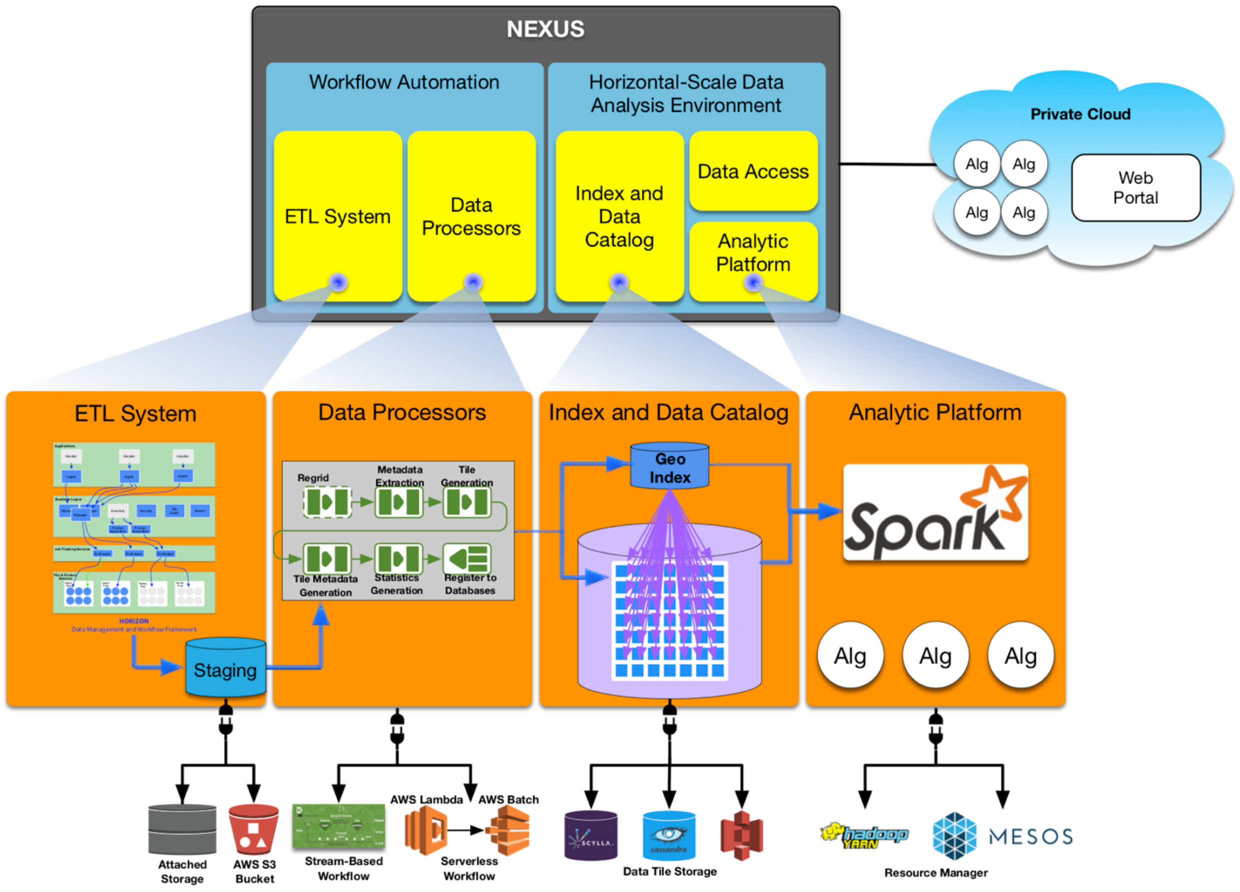
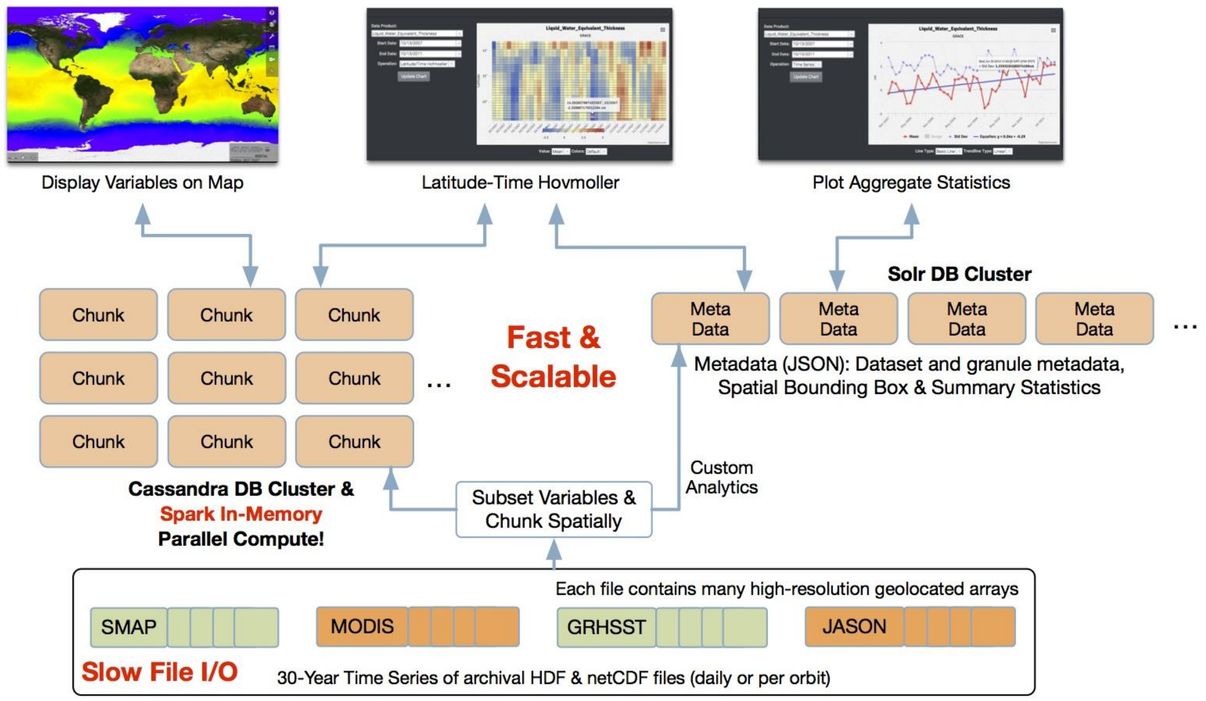


**Figure 4.** High-level architecture of NEXUS. Data ingest is handled by an Extract-Transform-Load system, which hands off to a Data Processor to partition the input data file into data tiles with pre-compute some statistics. On-demand analysis then uses Spark to make computations, fetching data tiles from a Cassandra database.



Display Variables on Map
Latitude-Time Hovmoller
Plot Aggregate Statistics
Solr DB Cluster
Chunk
Fast & Scalable
Meta Data
…
Metadata (JSON): Dataset and granule metadata, Spatial Bounding Box & Summary Statistics
Custom Analytics
Subset Variables & Chunk Spatially
Cassandra DB Cluster & Spark In-Memory Parallel Compute!
Each file contains many high-resolution geolocated arrays
SMAP
MODIS
GRHSST
JASON
Slow File I/O
30-Year Time Series of archival HDF & netCDF files (daily or per orbit)

**Figure 5.** NEXUS' two-database architecture to enable in-memory map-reduce processing of geospatial array data.

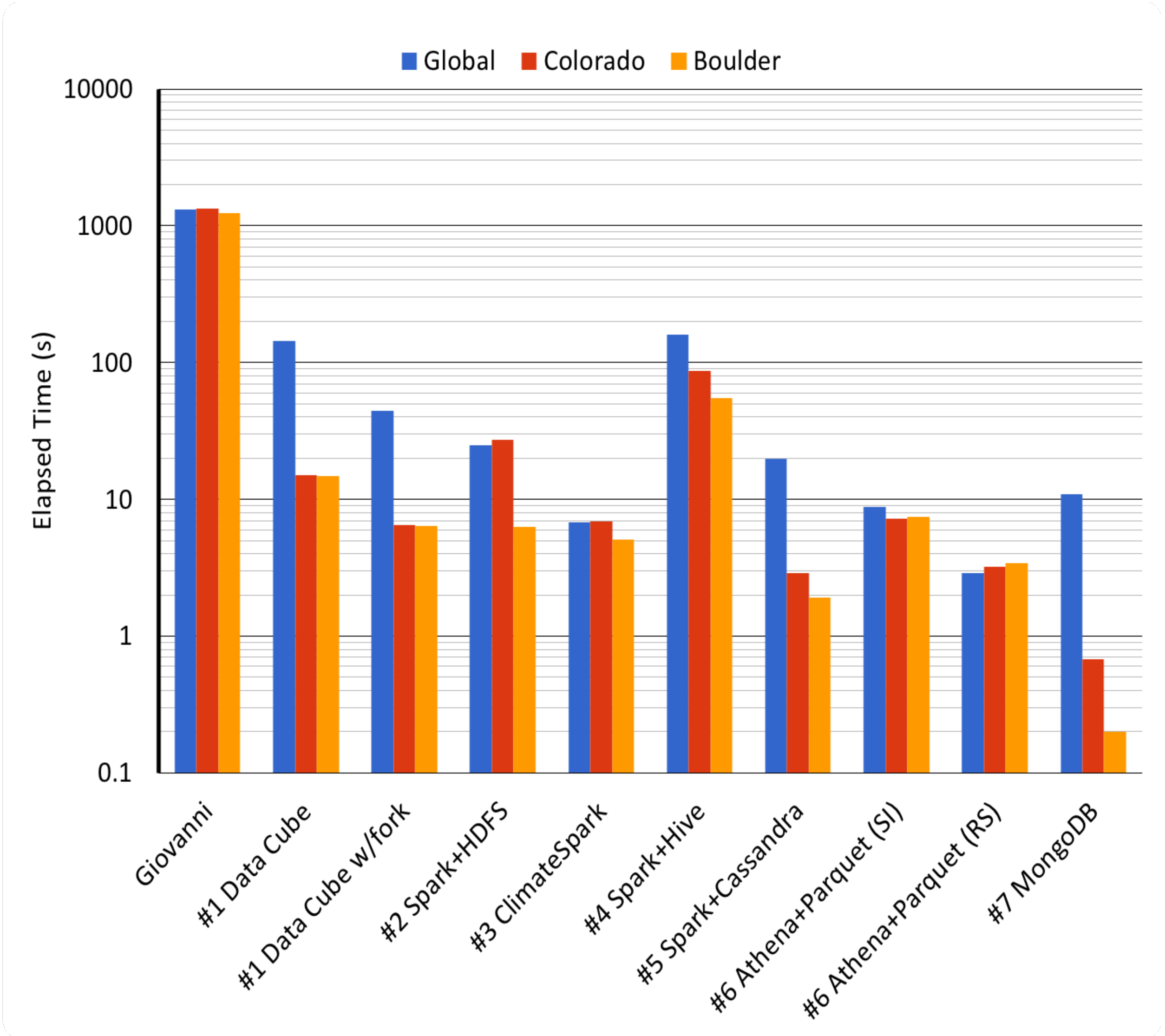


**Figure 6.** Elapsed time results for computation of area-averaged time series for global, Colorado, and Boulder cases. The Data Cube architecture (#1) is run in two modes, without and with simple forked multi-processing. The Athena+Parquet architecture (#6) is populated with two different forms of the data, the first accompanied by a Spatial index, and the second using a Running Sum of the data instead of the data themselves.