National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

Big Data Analytics Using Apache Spark

Joseph Jacob, Frank Greguska, Thomas Huang,
Nga Quach, and Brian Wilson

NASA Jet Propulsion Laboratory / California Institute of Technology

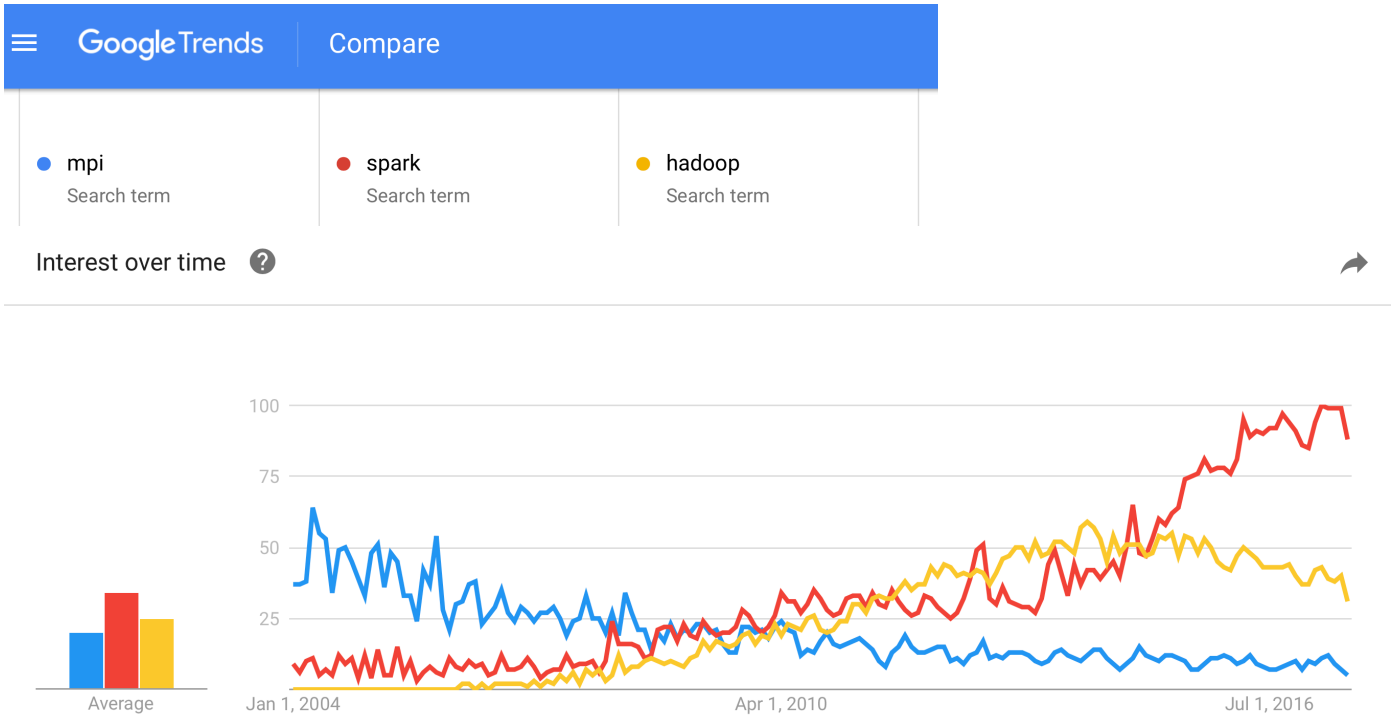
Pasadena, CA, USA

High Performance Computing: Alternative Approaches

- Dataflow / Workflow / Graph
  - Taverna
  - Pegasus
  - Spring XD
- Vector Computations
  - Vendor specific
- Message Passing
  - Parallel Virtual Machine (PVM)
  - Message Passing Interface (MPI)
- Shared Memory
  - OpenMP
  - Vendor-specific
- Hybrid OpenMPI / MPI
- Hardware-based approaches
  - VHDL, Verilog - Field Programmable Gate Array (FPGA)
  - CUDA – GPU
  - OpenCL – Heterogeneous platforms
- Master-Worker
  - Condor
  - Apache Spark
- Map-Reduce
  - Apache Hadoop
  - Apache Spark
- Grid Computing
  - Globus Toolkit
  - NASA Information Power Grid (IPG)
  - NSF TeraGrid
- Cloud Computing
  - Amazon Web Services (AWS)
  - Microsoft Azure
  - Google Cloud Platform
  - IBM Cloud

HPC Showdown: Spark vs MPI

- Jonathan Dursi's Blog [www.dursi.ca]: **"HPC is dying, and MPI is killing it"**
  - *"In [the modern era of internet-scale big data], one might expect that programmers with HPC experience – who have dealt routinely with terabytes and now petabytes of data, and have years or decades of experience with designing and optimizing distributed memory algorithms – would be in high demand. They are not."*
  - *"MPI is at the wrong level of abstraction for application writers [and] tool builders"*
  - *"MPI is more than you need for modest levels of parallelism [and] less than you need at extreme levels of parallelism."*



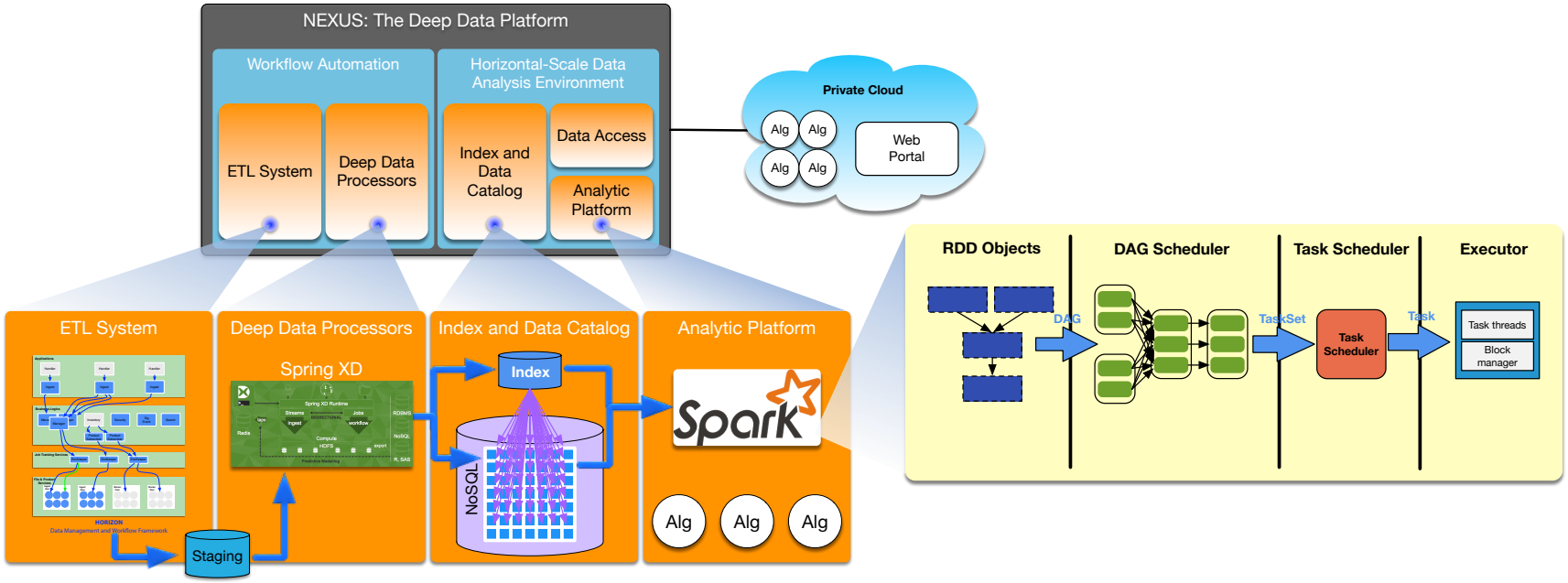
HPC Showdown: Spark vs MPI (cont.)

- J. L. Reyes-Ortiz, L. Oneto, D. Anguita, **"Big Data Analytics in the Cloud: Spark on Hadoop vs. MPI/OpenMP on Beowulf"**, 2015 INNS Conference on Big Data, Vol. 53, 2015, pp. 121-130.
  - *"the MPI/OpenMP implementation is much more powerful than the Spark on Hadoop alternative in terms of speed. [For KNN], it can be more than 10 times faster."*
  - *"MPI/OpenMP scales better than Spark."*
  - *"Nevertheless, Spark on Hadoop may be preferred because it also offers a distributed file system with failure and data replication management, allows the addition of new nodes at runtime, and provides a set of tools for data analysis and management that is easy to use, deploy and maintain."*

- MPI offers low level control and highly optimized communications
  - For applications that cannot be carved up into nice independent partitions, MPI performance can greatly exceed that of Spark, which was not designed for that use case.

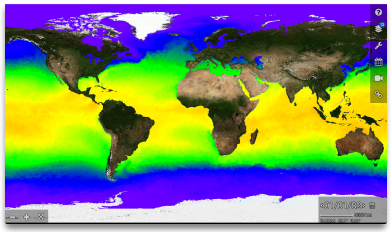
- Spark offers more than just computational speed
  - Excellent performance across clusters for algorithms that are "embarrassingly" parallel.
  - Easy to deploy and manage on a cluster computer
  - Excellent support by cloud vendors
  - Convenient API at a level of abstraction that frees application developers to focus on numerical operations rather than inter-process communications.
  - Automatic rescheduling of tasks hosted on failed or overloaded nodes.

The NEXUS Architecture

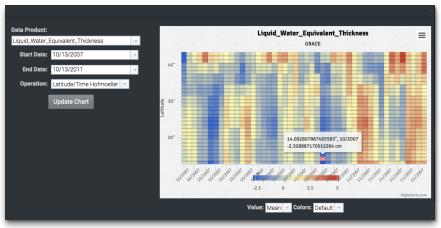


- Extract/Transform/Load (ETL) System – Ingest and stage data
- Deep Data Processors – metadata, statistics, and tiles
- Index and Data Catalog – horizontal-scale geospatial search (Solr) and tile retrieval (Cassandra)
- Analytic Platform – Spark- based domain-specific analytics
- Data Access – tile and collection-based data access
- Cloud Platform – portal and custom VMs

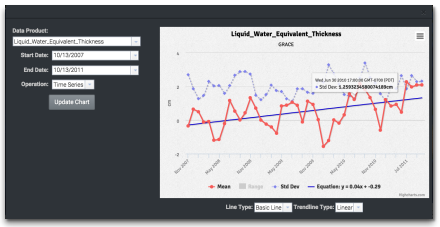
From Files to Tiles



Display Variables on Map



Latitude-Time Hovmoller



Plot Aggregate Statistics

Chunk Chunk Chunk
Chunk Chunk Chunk ...
Chunk Chunk Chunk

Cassandra DB Cluster &
Spark In-Memory
Parallel Compute!

Fast & Scalable

Solr DB Cluster

Meta Data | Meta Data | Meta Data | Meta Data ...

Metadata (JSON): Dataset and granule metadata, Spatial Bounding Box & Summary Statistics

Custom Analytics

Subset Variables & Chunk Spatially

Each file contains many high-resolution geolocated arrays

SMAP | MODIS | GRHSST | JASON

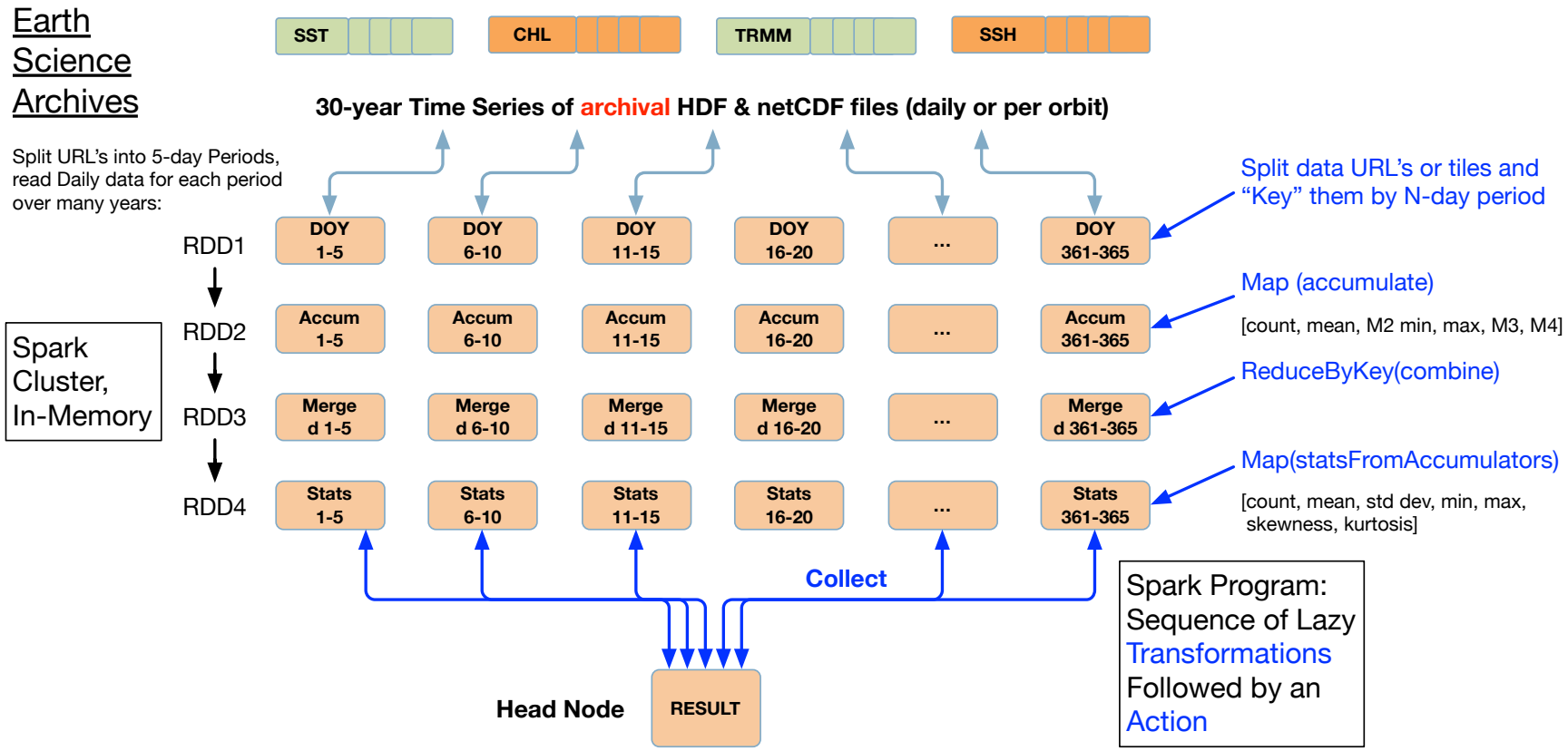
Slow File I/O

30-Year Time Series of archival HDF & netCDF files (daily or per orbit)

Map-Reduce Style Computation

Resilient Distributed Dataset (RDD)

- The RDD is Spark's abstraction of a dataset as a collection of elements that are partitioned across a cluster computer.
- Parallel computations can be applied to an RDD to produce a new RDD.
- Fault tolerant



NEXUS Spark Analytics Algorithms

Included with NEXUS:

- Area-Averaged Time Series
  - Compute statistics (e.g., mean, minimum, maximum, standard deviation) for each time step within a user-specified spatiotemporal bounding box.
  - Optionally apply seasonal or low-pass filters.
  - Return result in ascending time order in JSON format.
- Time-Averaged Map
  - Compute a geospatial map that averages gridded measurements over time at each grid coordinate within a user-defined spatiotemporal bounding box.
- Correlation Map
  - Computes the correlation coefficient at each grid coordinate within a user-specified spatiotemporal bounding box for two identically gridded datasets.
  - Automatically aligns the time stamps for the two datasets being compared.
- Climatological Map
  - Similar to Time-Averaged Map, but only includes measurements in the time average that are within a user specified month.

Application Specific Extension: Anomaly Detection (OceanXtremes)

- Climatology
  - For each day-of-year (1-366) or month (1-12), computes a ”typical value” for each coordinate grid location.
  - The ”typical value” may be the result of either (1) a standard pixel mean with optional smoothing over time (e.g. 5-day average), (2) Gaussian interpolation [Armstrong and Vazquez-Cuervo, 2001], or Empirical Orthogonal Function (EOF).
- Daily Difference Average
  - Subtract a dataset from its climatology, then, for each time stamp, average the differences within a user-specified spatiotemporal bounding box.
  - Product can be used to search for anomalies compared to the historical norm.

Application-Specific Extension: Distributed Oceanographic Match-up Service (DOMS)

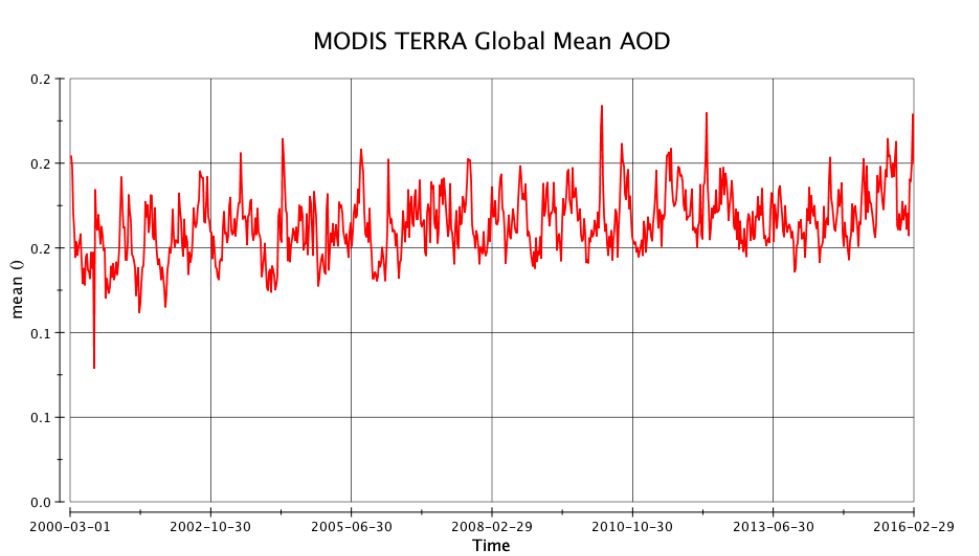
- In Situ Match
  - Discover in situ measurements that correspond with a gridded satellite measurement.

The Spark Approach I

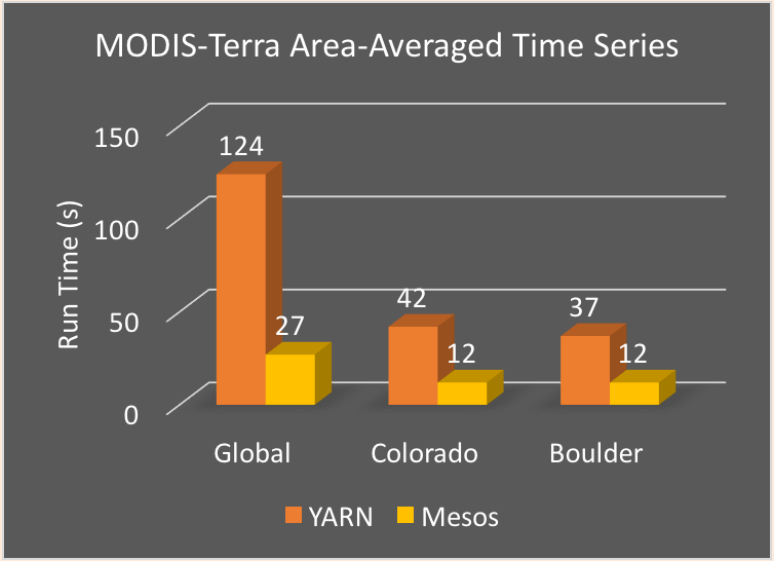
- Multiple Shells/API:
  - Java
  - Scala
  - **Python (PySpark)**
  - R (SparkR)
- Multiple deployment modes:
  - Standalone
  - Hadoop YARN
  - **Apache Mesos**

The Spark Approach I

- Multiple Shells/API:
  - Java
  - Scala
  - **Python (PySpark)**
  - R (SparkR)
- Multiple deployment modes:
  - Standalone
  - Hadoop YARN
  - **Apache Mesos**



- NEXUS run on 8-node cluster computer at JPL running Solr, Cassandra, Spark 2.0, with the YARN or Mesos scheduler, as indicated in the plot.
- Area-Averaged Time Series over the indicated spatial subset (Global, State, City) run with 16-way parallelism.
- Variable plotted: MODIS-Terra Aerosol Optical Depth (AOD) 550 nm dark target
- 5,789 daily data granules covering the globe at 1 deg resolution with date range: 3/1/2000 – 2/29/2016 (3 GB input data volume).
- In our experiments, using Mesos consistently yields a speedup of 2 to 4 times over YARN.



The Spark Approach II

- Spark applications follow the master-worker paradigm:
  - Driver program creates the RDD and defines “map” and ”reduce” operations to be applied to the data.
  - Executors (Workers) collectively store the RDD in memory and in parallel perform the specified transformations on the data.

- Spark applications follow the map-reduce paradigm:
  - Spark API provides multiple “map” functions:
    - `map()` : operates on dataset elements
    - `mapPartitions()` : operates on entire data partitions
  - Spark API provides a variety of ways to collect the result of a computation back to the driver node, or perform reduction (summarization of the results)
    - collect() : merge all elements of the RDD into a single list
      - Use with care: the combined dataset must be small enough to fit in the driver node’s memory!
    - reduce()
    - reduceByKey()
    - foldByKey()
    - combineByKey()

PySpark Program Structure I

- This is a simple program structure for cases where:
  - All the data fits in memory on the head node.
  - The computation simply maps each element in the data array to an output
  - The results are simply collected back to the head node. There is no further reduction operation.

```
def map(x):
        # transform x and return a value
        return f(x)

def main():
        data = <an array of input data>
        ...
        # Configure Spark
        sp_conf = SparkConf()
        sp_conf.setAppName("MyApp")
        sp_conf.set("spark.executor.memory", "4g")
        ...
        sp_conf.set( < more settings >)

        # Create Spark Context and initial Spark RDD
        sc = SparkContext(conf=sp_conf)
        rdd = sc.parallelize(data, num_partitions)

        # Perform map computation and collect the results back to
        # this head node.
        results = rdd.map(map).collect()
        <print, save, or plot results>
```

PySpark Program Structure II

- This program structure is for cases where a map and reduce operation is required.
  - All the data fits in memory on the head node.
  - The computation maps each element in the data array to an output key-value pair
  - The results are added by key and collected back to the head node.

```
from operator import add

def map(x):
        # transform x and return a key-value pair (example: time, value)
        return create_key(x), create_value(x)

def main():
        data = <an array of input data>
        ...
        # Configure Spark
        sp_conf = SparkConf()
        sp_conf.setAppName("MyApp")
        sp_conf.set("spark.executor.memory", "4g")
        ...
        sp_conf.set( < more settings >)

        # Create Spark Context and initial Spark RDD
        sc = SparkContext(conf=sp_conf)
        rdd = sc.parallelize(data, num_partitions)

        # Perform map computation and collect the results back to
        # this head node.
        map_result = rdd.map(map)
        reduce_result = map_result.foldByKey(0, add).collect()

        <print, save, or plot results>
```

Practical Considerations I

- For big data, the map function might have to do the I/O.
  - The canonical Spark tutorial example application, word count, is not a good example of how a science algorithm would use Spark!
  - Insufficient memory on a single node.
  - Carefully consider size of the data being collected.
  - Single pass algorithms needed
  - Database query size must be considered
    - Very small queries are inefficient
    - Very large queries might time out, or result in excessive memory consumption.
- The map function must be static
  - Trying to use a class method as your map function will result in an exception.
  - Python decorator: @staticmethod
  - Other class variables or methods ("self") not available inside map function.
  - Can get a program working with python's built-in map() function first; small additional leap to make it work with Spark.
- The map function takes just one input argument
  - Use a tuple or list to pass in more information if needed. Unpack the tuple inside the map function.
  - If many necessary arguments don't change from one map call to the next, consider using the partial function in the Python functools module to create a new version of the function with only the arguments that change.

```
from functools import partial
def f(a, b, c):
   return a+b*c
f_part = partial(f,1,2)
x = 3
# The following returns the same
# result as f(1,2,x)
y = f_part(x)
```

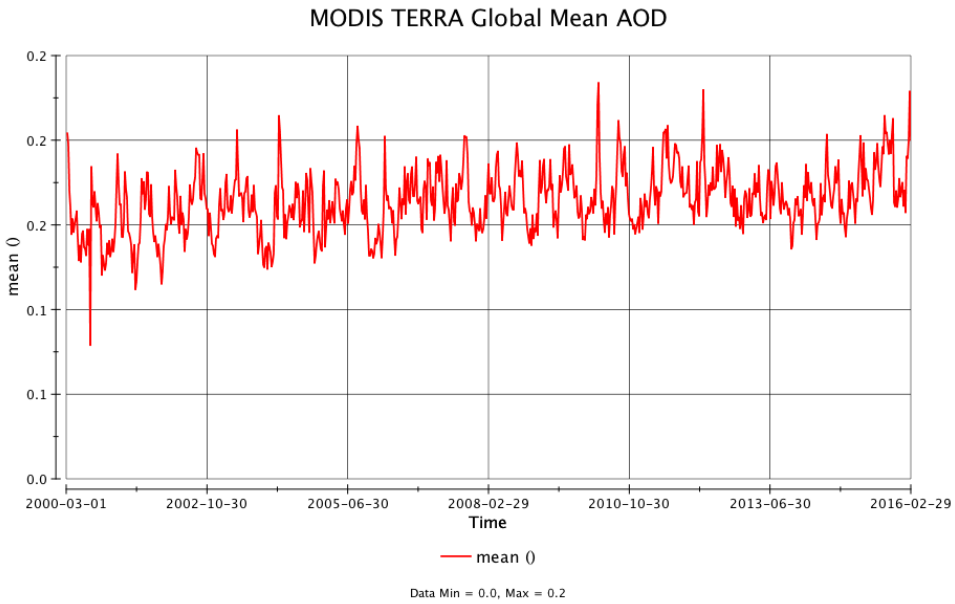
Practical Considerations II

- Control number of Spark executors and data partitions
  - Executors are the worker processes that are instantiated when the Spark cluster is initialized and last for the life of the Spark application.
  - A data partition represents a chunk of work that is scheduled for processing on an executor.
- Spark performance depends on configuration.
  - Number of executors, E
  - Cores per executor
  - Memory per executor
  - Number of data partitions, P
  - Recommended that 2 <= P/E <= 4
  - > 200 configuration parameters in Spark 2.2.0 documentation
  - Nontrivial to squeeze best performance out of Spark for complex applications.
- The Scheduler used can impact performance
  - Spark uses YARN by default
  - Mesos is available as a separate package
  - In our benchmarks, Mesos consistently yielded 2-4 times faster run times compared to YARN.
- The data partitioning scheme used can impact performance
  - Calculations on global data or very large subsets have best performance with a few large tiles.
  - Many small tiles are preferred for calculations on smaller subsets.

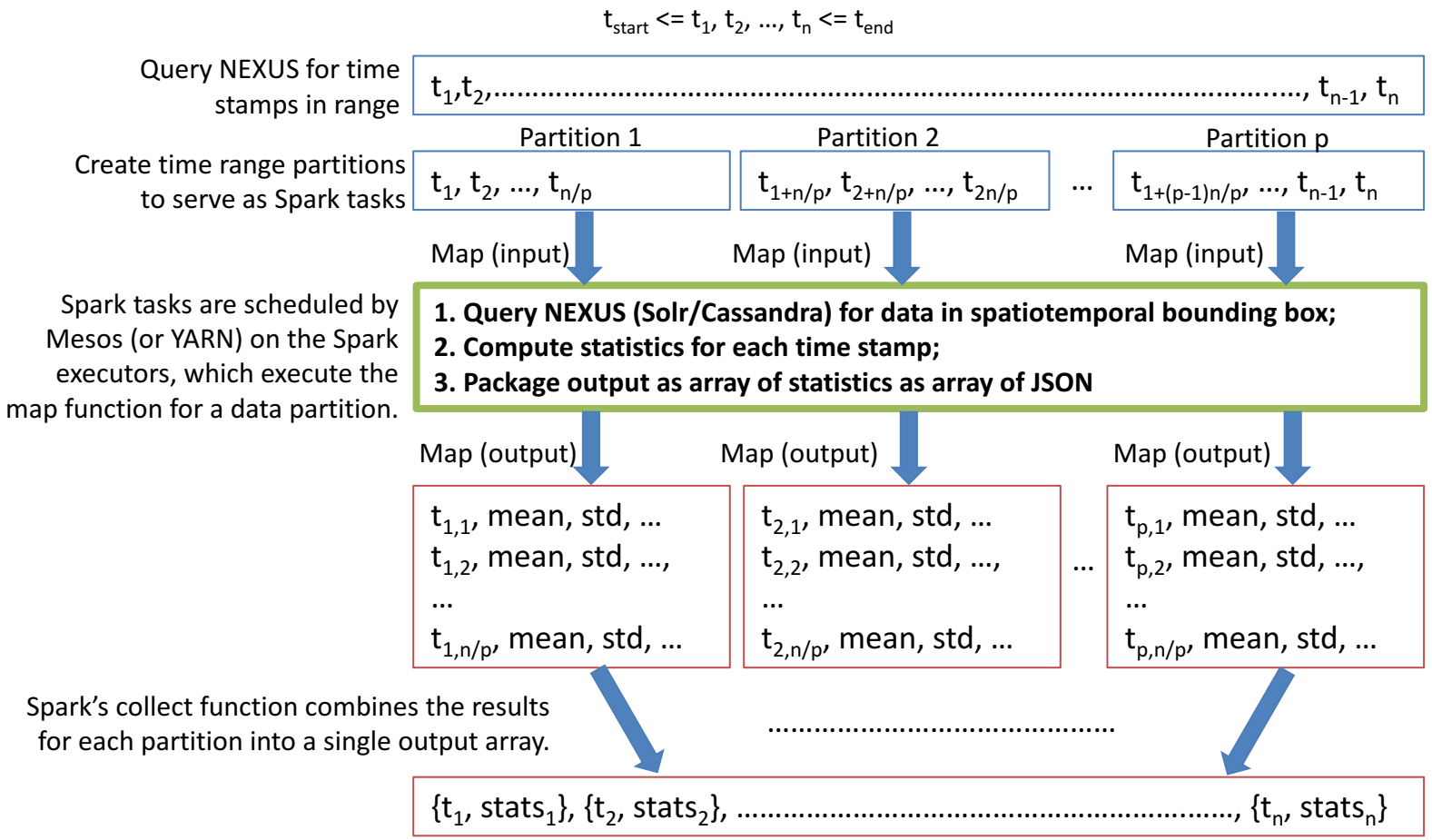
Time Series Analysis

- Compute statistics (e.g., mean, minimum, maximum, standard deviation) for each time step within a user-defined spatio-temporal bounding box.
  - Optionally apply seasonal or low-pass filters.
  - Return result in ascending time order in JSON format.


```
"data": [
  [
    {
      "std": 1.4744961225186004,
      "cnt": 2107,
      "minSeasonalLowPass": -1.3469589497685839,
      "minSeasonal": -1.3745376124526523,
      "maxLowPass": 15.435052751030272,
      "min": 9.32000732421875,
      "max": 15.44000244140625,
      "meanSeasonal": -1.9343310558434688,
      "ds": 0,
      "meanSeasonalLowPass": -1.9285414148993034,
      "maxSeasonalLowPass": -2.1962260569545178,
      "time": 1427846400,
      "maxSeasonal": -2.1875730572324805,
      "meanLowPass": 12.279365215634886,
      "minLowPass": 9.3507517896462442,
      "mean": 12.273533821105957
    }
  ], ...
```

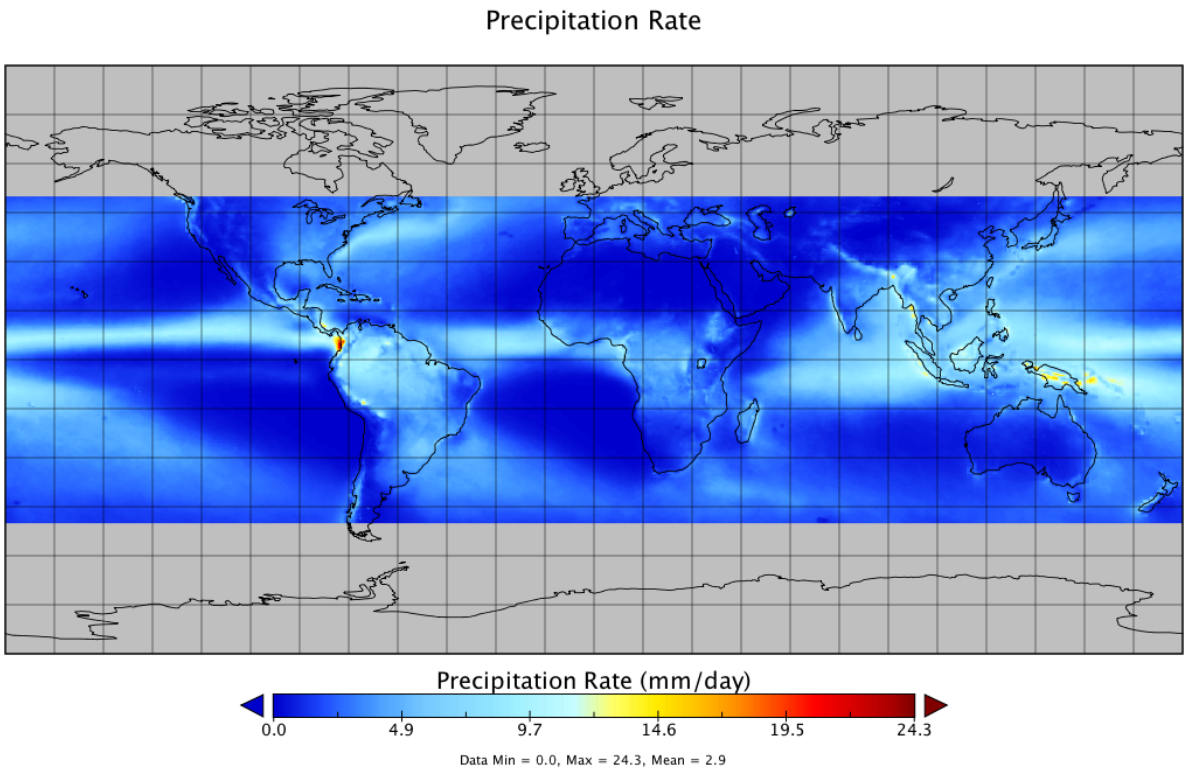



Time Series Spark Implementation

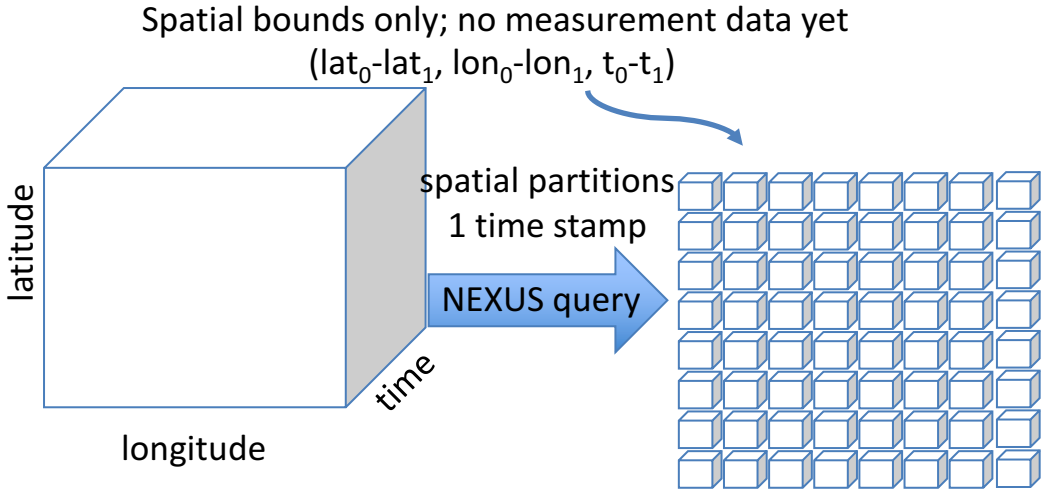


Time-Averaged Map

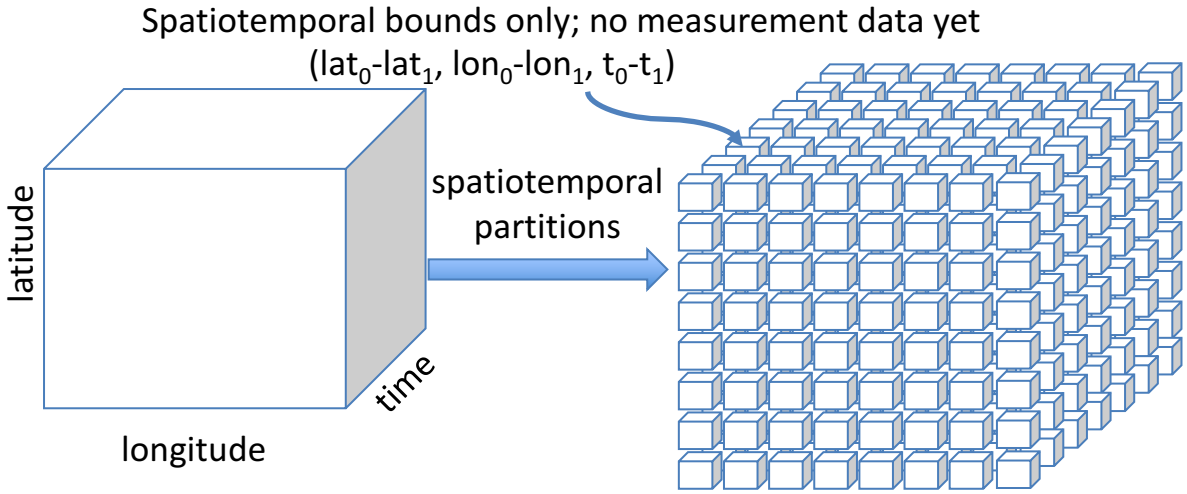
- Compute a geospatial map that averages gridded measurements over time at each grid coordinate within a user-defined spatiotemporal bounding box.
- Fill values are excluded from the calculation
- Result provided as in JSON format or NetCDF.



Time-Averaged Map Spark Implementation

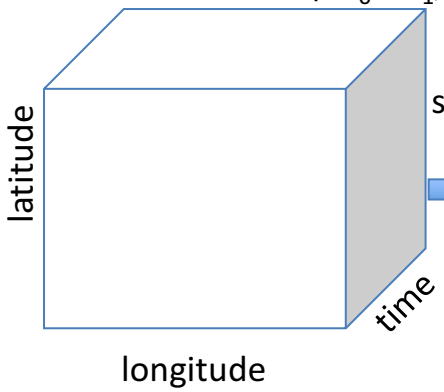


Time-Averaged Map Spark Implementation

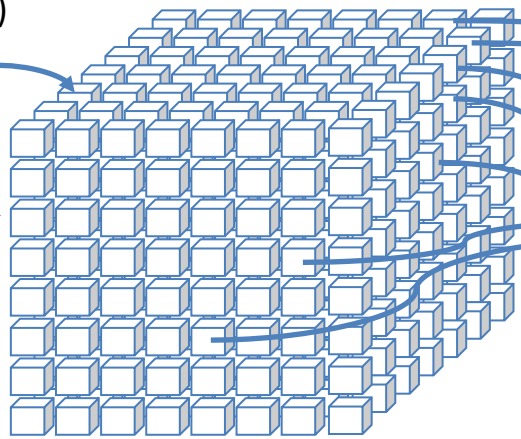


Time-Averaged Map Spark Implementation

Spatiotemporal bounds only; no measurement data yet
(lat_0-lat_1, lon_0-lon_1, t_0-t_1)



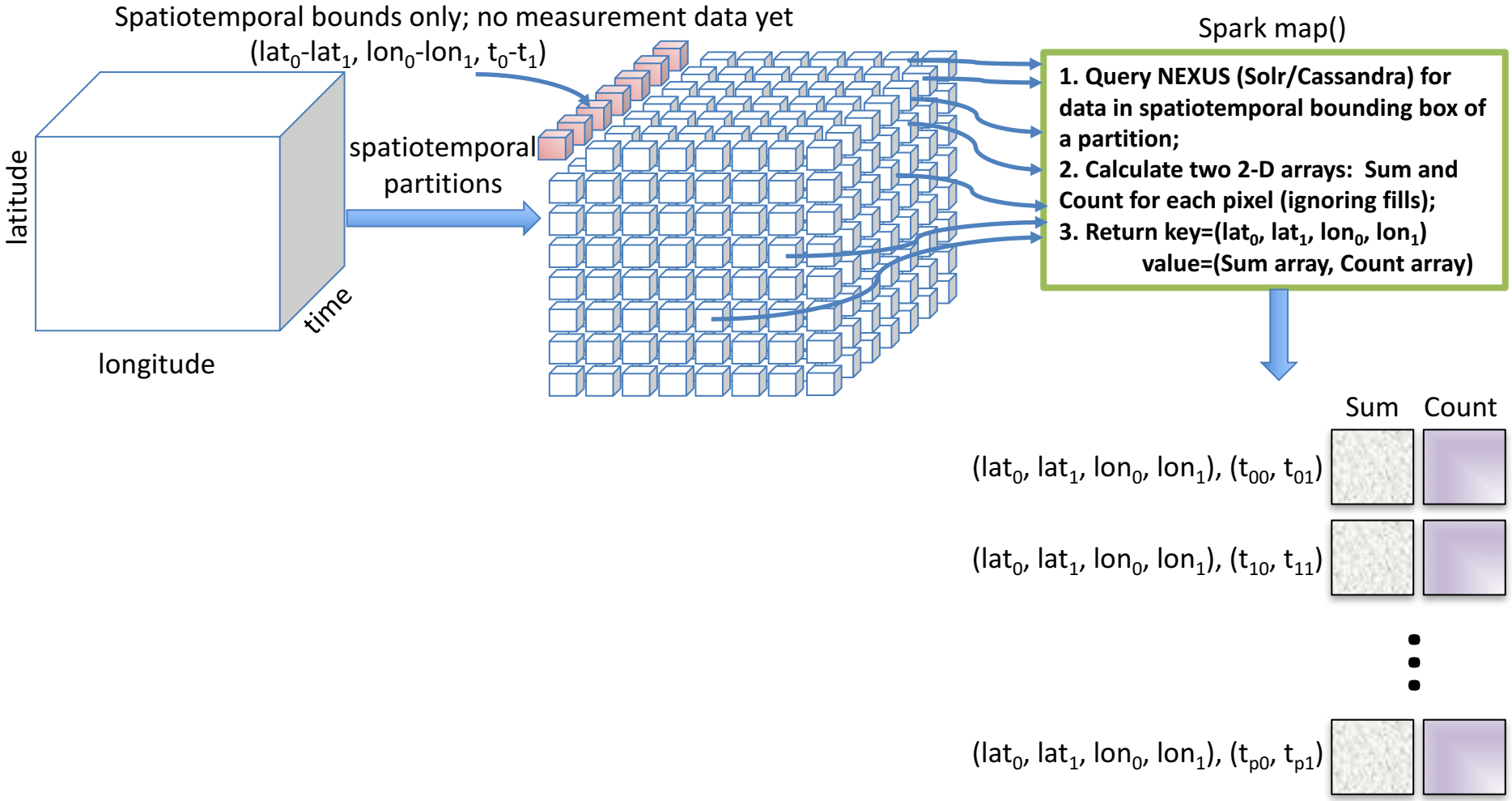
spatiotemporal partitions



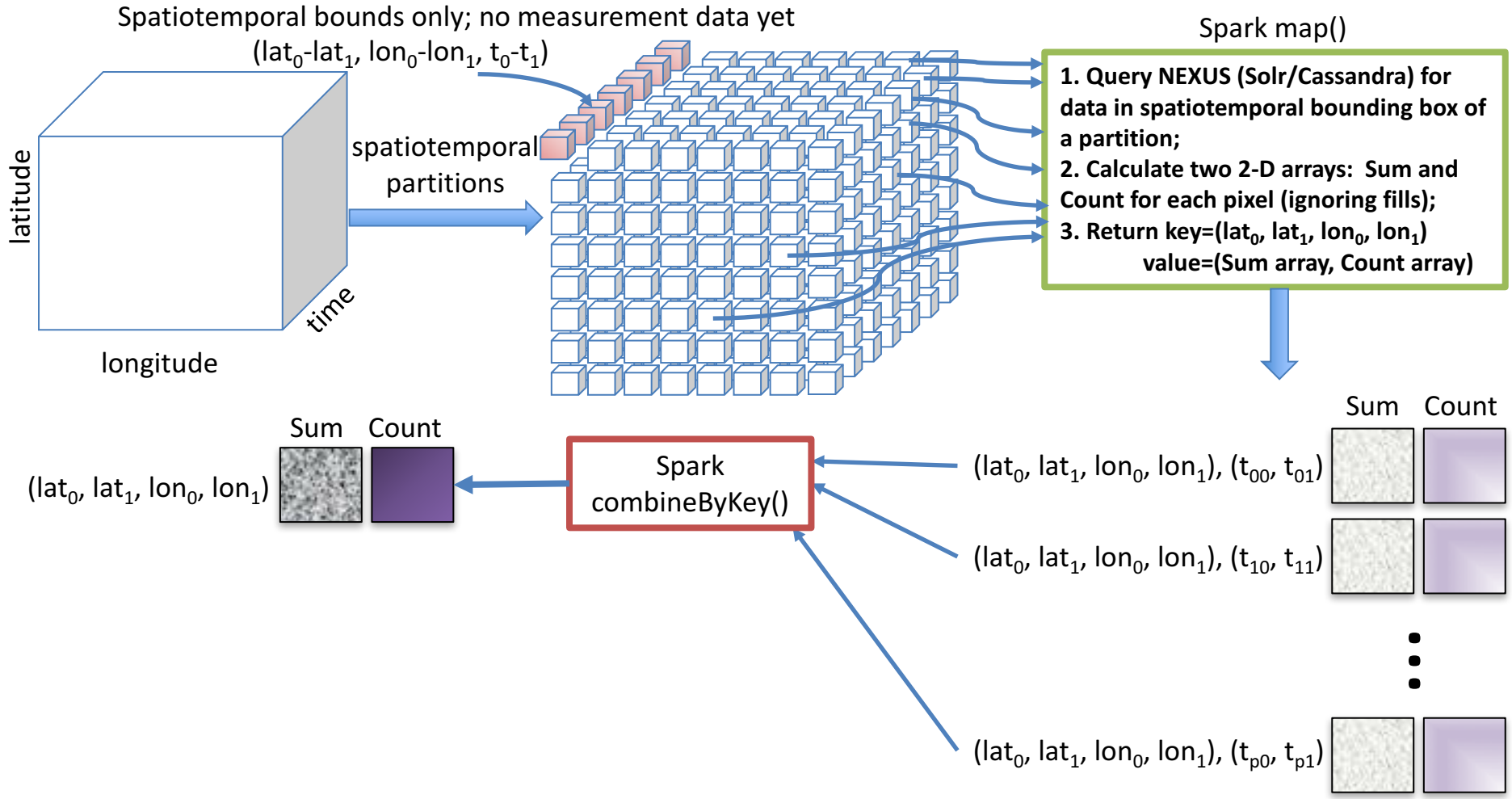
Spark map()

1. Query NEXUS (Solr/Cassandra) for data in spatiotemporal bounding box of a partition;
2. Calculate two 2-D arrays: Sum and Count for each pixel (ignoring fills);
3. Return key=(lat_0, lat_1, lon_0, lon_1)
value=(Sum array, Count array)

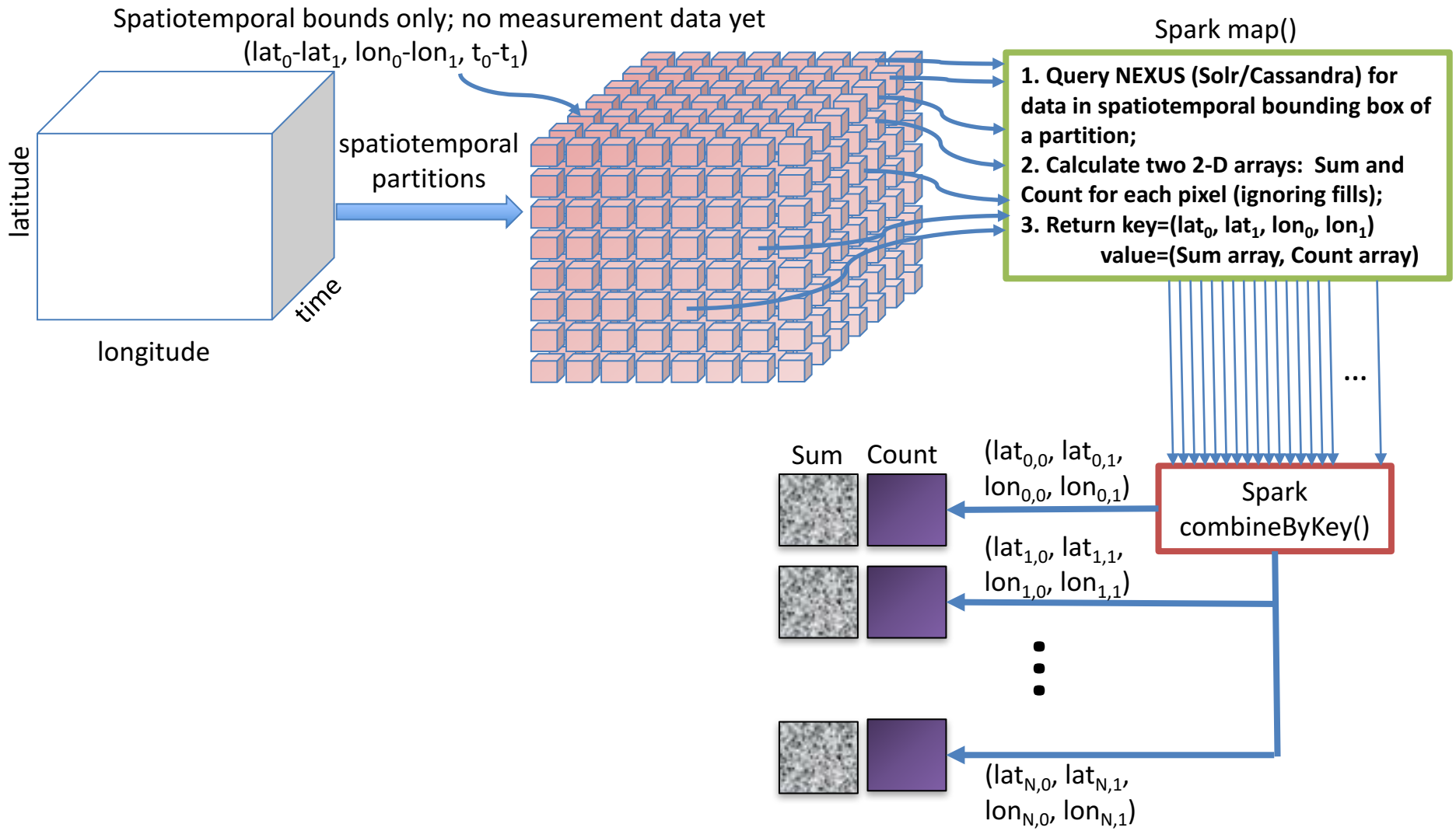
Time-Averaged Map Spark Implementation



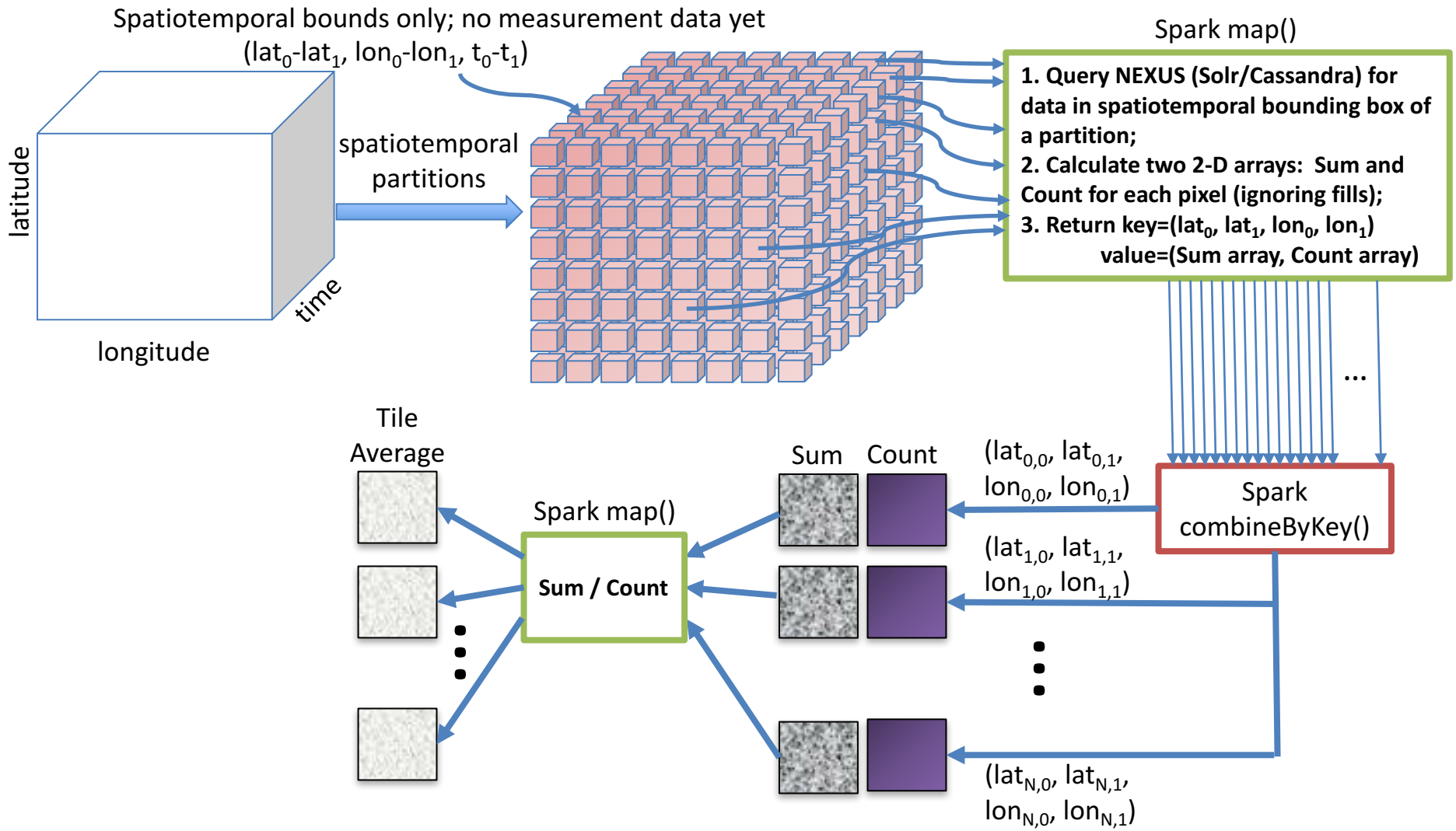
Time-Averaged Map Spark Implementation



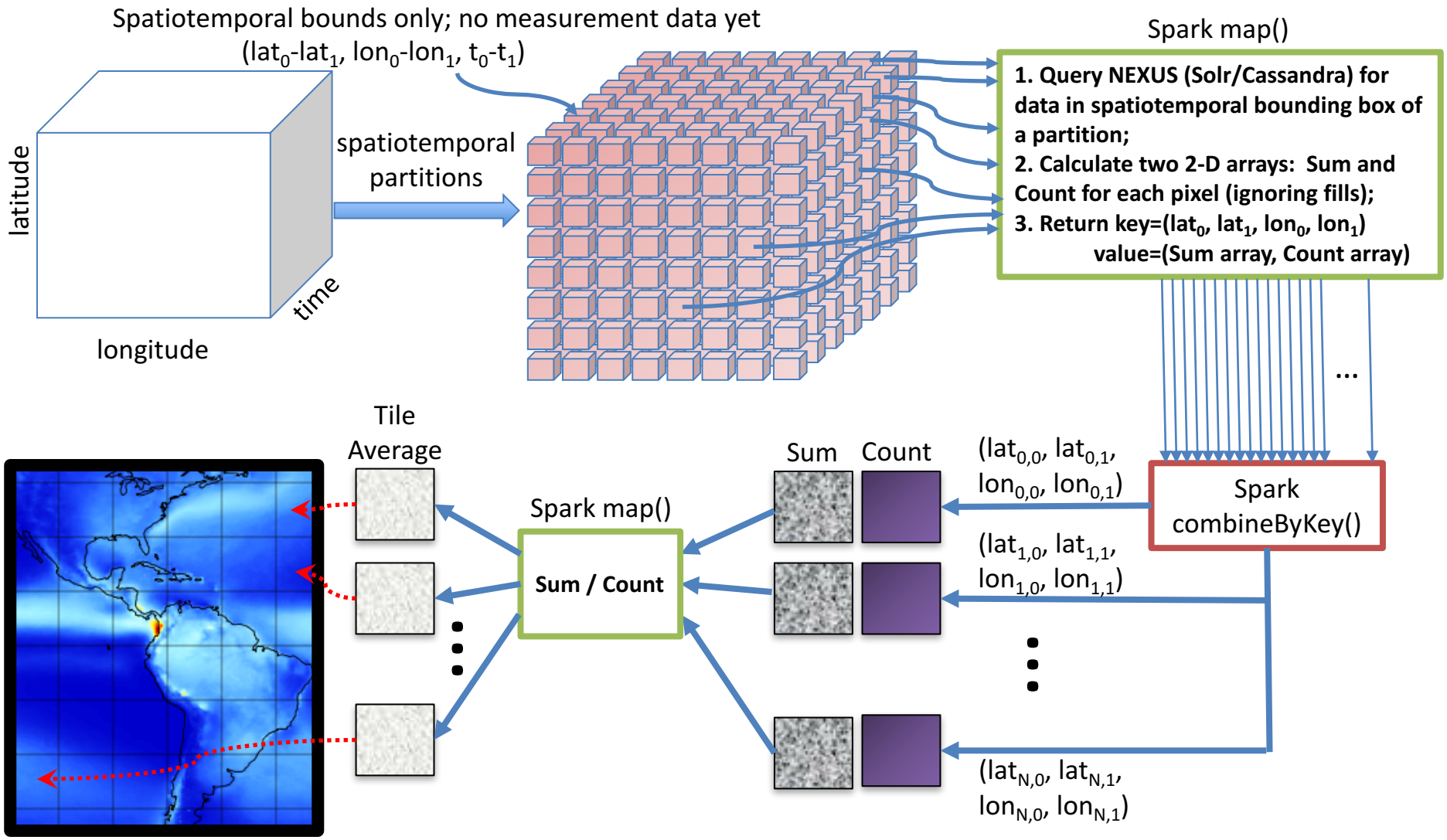
Time-Averaged Map Spark Implementation



Time-Averaged Map Spark Implementation



Time-Averaged Map Spark Implementation



NEXUS Spark Analytics Algorithms

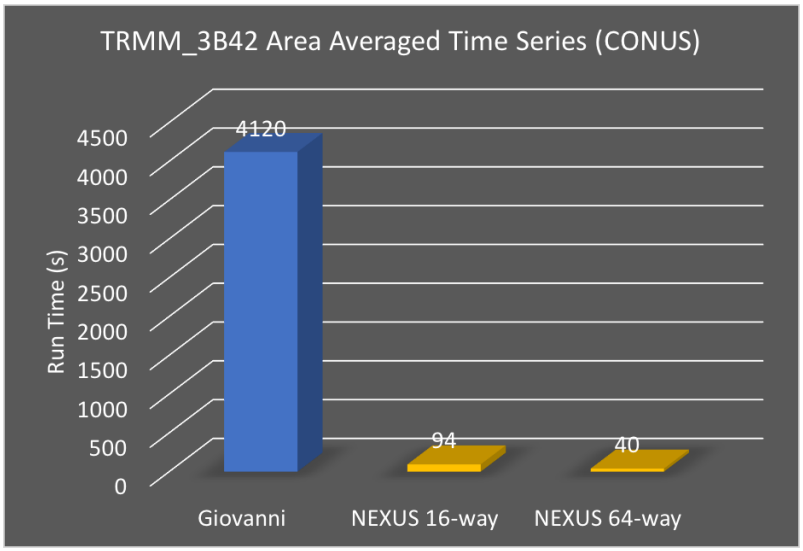
Compute statistics over time to produce a single value for each coordinate grid location.

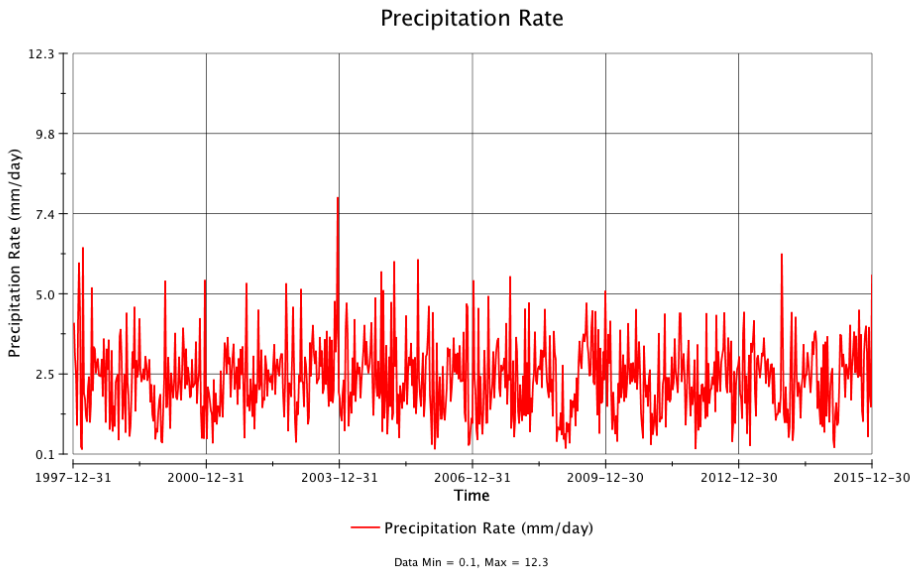
- Time-Averaged Map
  - Compute a geospatial map that averages gridded measurements over time at each grid coordinate within a user-defined spatiotemporal bounding box.
- Correlation Map
  - Computes the correlation coefficient at each grid coordinate within a user-specified spatiotemporal bounding box for two identically gridded datasets.
  - Automatically aligns the time stamps for the two datasets being compared.
- Climatological Map
  - Similar to Time-Averaged Map, but only includes measurements in the time average that are within a user specified month.
- Climatology
  - For each day-of-year (1-366) or month (1-12), computes a ”typical value” for each coordinate grid location.
  - The ”typical value” may be the result of either (1) a standard pixel mean with optional smoothing over time (e.g. 5-day average), (2) Gaussian interpolation [Armstrong and Vazquez-Cuervo, 2001], or Empirical Orthogonal Function (EOF).

Compute spatial statistics to produce a single value (or set of statistics) for each time stamp.

- Area-Averaged Time Series
  - Compute statistics (e.g., mean, minimum, maximum, standard deviation) for each time step within a user-specified spatiotemporal bounding box.
  - Optionally apply seasonal or low-pass filters.
  - Return result in ascending time order in JSON format.
- Daily Difference Average
  - Subtract a dataset from its climatology, then, for each time stamp, average the differences within a user-specified spatiotemporal bounding box.
  - Product can be used to search for anomalies compared to the historical norm.

Time Series: Comparison of Giovanni and NEXUS Performance

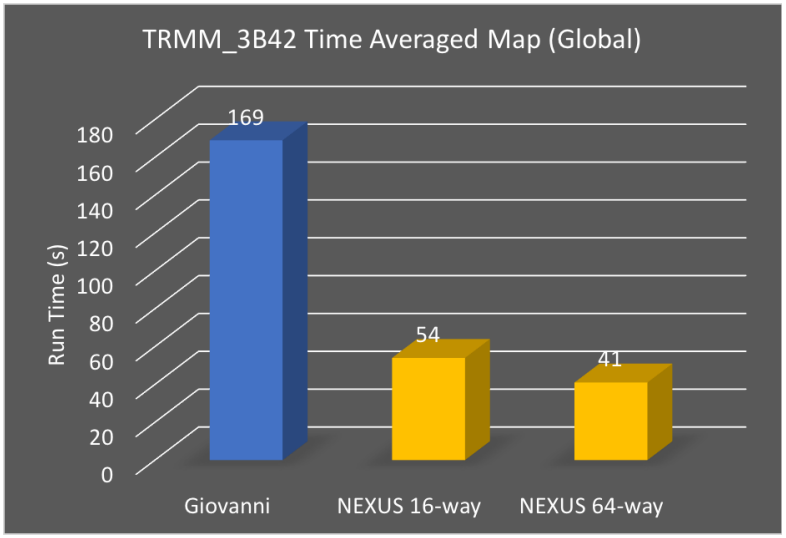


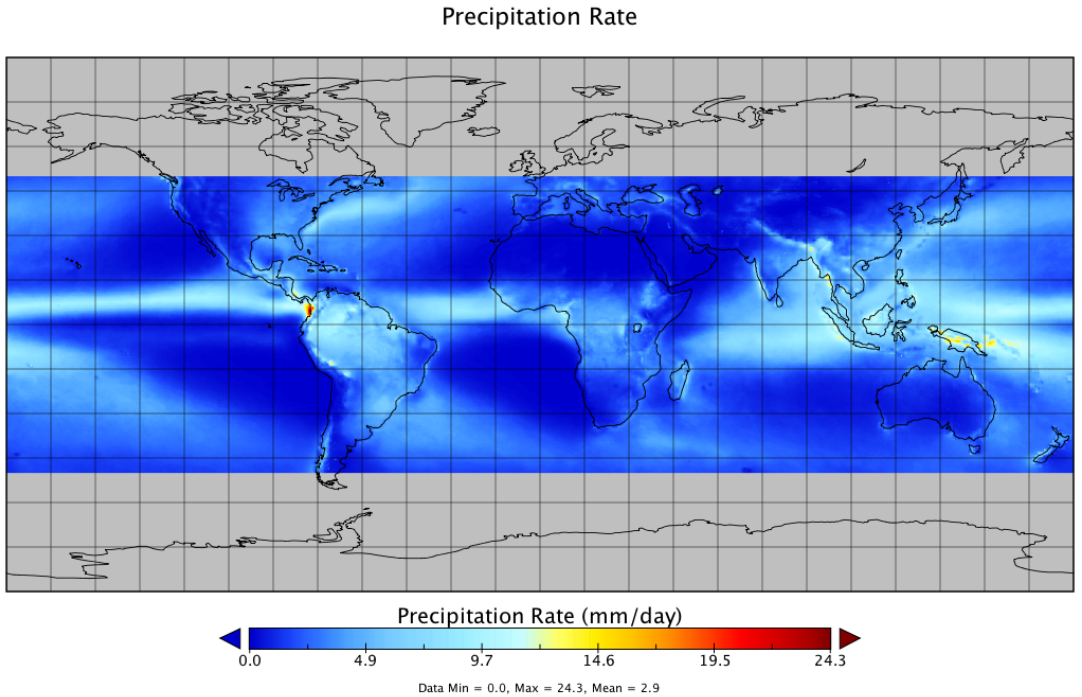


- NEXUS run on 8-node cluster computer at JPL running Solr, Cassandra, Spark 2.0, Mesos
- Area-Averaged Time Series over the continental United States
- Variable plotted: TRMM daily precipitation rate (TRMM_3B42_daily_precipitation_V7)
- 6,574 daily data granules covering the globe at 0.25 deg resolution with latitude +/- 50 deg and date range: 1/1/1998 – 12/31/2015 (26 GB input data volume).
- Giovanni implementation uses highly optimized compiled code based on NetCDF Operator (NCO) toolkit, but is single threaded. NEXUS is implemented in Python and parallelized with Apache Spark.
- Giovanni execution time is compared with NEXUS for 16-way and 64-way parallelism.

National Aeronautics and Space Administration
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

Time Averaged Map: Comparison of Giovanni and NEXUS Performance

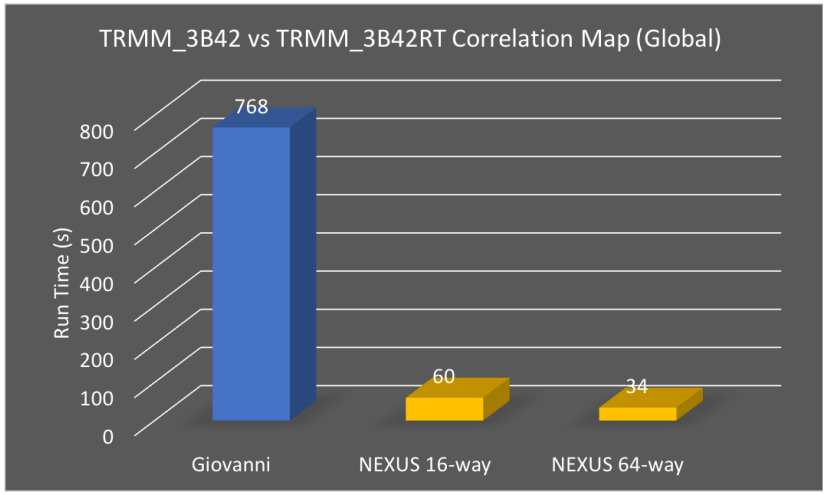


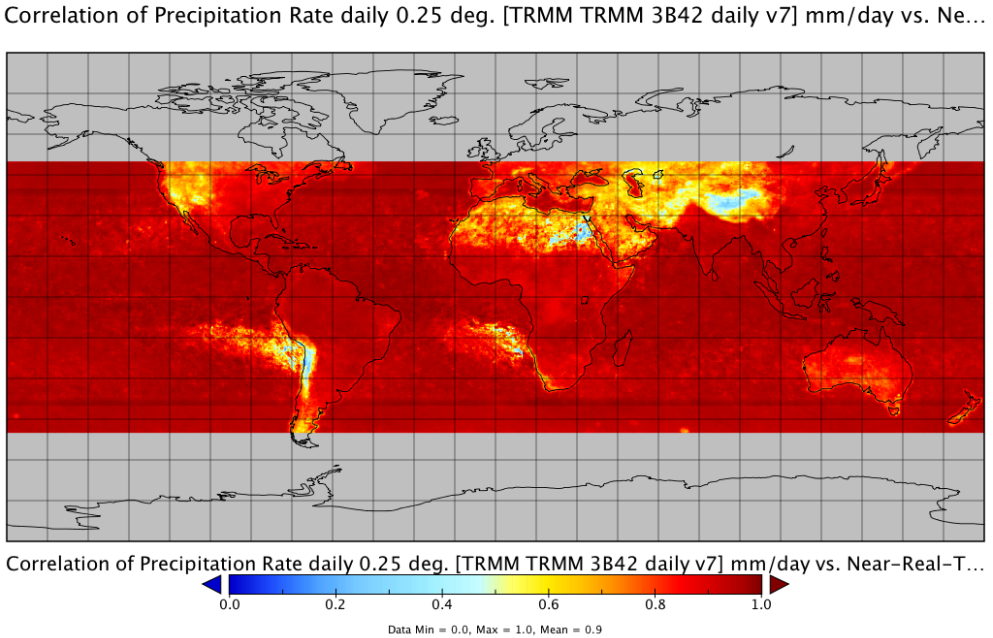


- NEXUS run on 8-node cluster computer at JPL running Solr, Cassandra, Spark 2.0, Mesos
- Global Time-Averaged Map
- Variable plotted: TRMM daily precipitation rate (TRMM_3B42_daily_precipitation_V7)
- 6,574 daily data granules covering the globe with latitude +/- 50 deg and date range: 1/1/1998 – 12/31/2015 (26 GB input data volume).
- Giovanni implementation uses highly optimized compiled code based on NetCDF Operator (NCO) toolkit, but is single threaded. NEXUS is implemented in Python and parallelized with Apache Spark.
- Giovanni execution time is compared with NEXUS for 16-way and 64-way parallelism.

National Aeronautics and Space Administration
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

Correlation Map: Comparison of Giovanni and NEXUS Performance

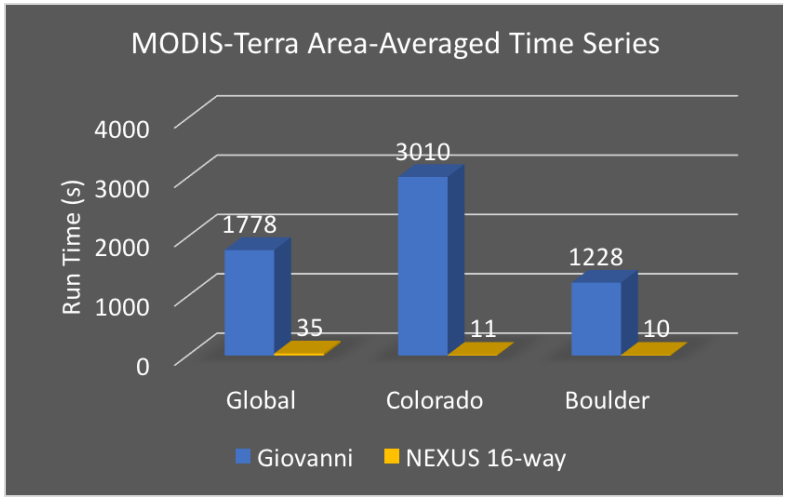


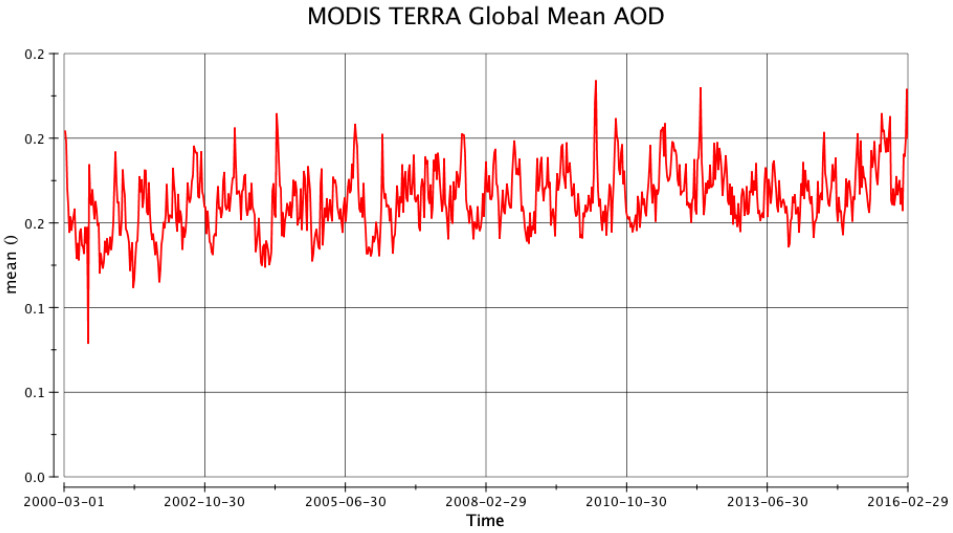


- NEXUS run on 8-node cluster computer at JPL running Solr, Cassandra, Spark 2.0, Mesos
- Global Correlation Map
- Variable plotted: TRMM daily precipitation rate (TRMM_3B42_daily_precipitation_V7) vs TRMM real-time daily precipitation (TRMM_3B42RT_daily_precipitation_V7)
- 5,113 daily data granule pairs covering the globe with latitude +/- 50 deg and date range: 1/1/2001 – 12/31/2014 (40 GB input data volume).
- Giovanni implementation uses highly optimized compiled code based on NetCDF Operator (NCO) toolkit, but is single threaded. NEXUS is implemented in Python and parallelized with Apache Spark.
- Giovanni execution time is compared with NEXUS for 16-way and 64-way parallelism.

National Aeronautics and Space Administration
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

Time Series: Comparison of Global vs. Subset Performance of Giovanni and NEXUS





- NEXUS run on 6 Amazon Web Services (AWS) Cloud instances of type “i2.4xlarge” running Solr, Cassandra, Spark 2.0, Mesos
- Area-Averaged Time Series over the indicated spatial subset (Global, State, City)
- Variable plotted: MODIS-Terra Aerosol Optical Depth (AOD) 550 nm dark target
- 5,789 daily data granules covering the globe at 1 deg resolution with date range: 3/1/2000 – 2/29/2016 (3 GB input data volume).
- Giovanni implementation uses highly optimized compiled code based on NetCDF Operator (NCO) toolkit, but is single threaded. NEXUS is implemented in Python and parallelized with Apache Spark.
- Giovanni execution time is compared with NEXUS for 16-way parallelism.
- NEXUS has superior performance for subsetting operations, as indicated by the ~300x speedup for the Colorado subset.

Summary

- Spark is a good choice for many algorithms, but consider inter-process communication.

- Benchmark NEXUS speedup factors over Giovanni
  - ~300x speedup for 16-year area-averaged time series of Moderate Resolution Imaging Spectroradiometer (MODIS-Terra) Aerosol Optical Depth (AOD) at 1 degree resolution for Colorado with NEXUS running on 6 “i2.4xlarge” Amazon Web Services Cloud instances with Spark configured for 16-way parallelism.
  - ▪ ~100x speedup for area-averaged time series of daily precipitation rate for the Tropical Rainfall Measuring Mission (TRMM with 0.25 degree spatial resolution) for the Continental United States over 18 years (1998 - 2015) with 64-way parallelism on an 8-node cluster computer at JPL.
  - ▪ ~4x speedup for 18-year (1998 - 2015) TRMM daily precipitation global time averaged map (64-way parallel).
  - ▪ ~22x speedup for 14-year (2001 - 2014) global map of correlation between TRMM daily and real time precipitation rate (64-way parallel).
  - For small datasets, compiled, optimized, single- threaded executables like the NetCDF Operators (NCO) toolkit used in Giovanni work well.
  - For large data analytics, NEXUS significantly outperforms due to its ability to horizontally scale in a cloud computing environment.

- Data tiling recommendations
  - Tiling data into chunks yields significant performance benefits over monolithic global granule files, particularly for regional subsets.
  - For calculations on small subsets, use a small tile size.
  - For global or near-global calculations use larger tiles to optimize data read performance.

- Choose your scheduler carefully
  - In our benchmarks, Mesos consistently yielded 2-4 times faster run times compared to YARN.
  - YARN included with Spark distribution; Mesos is a separate package.

- JPL and NASA support open source development
  - NEXUS is an open-source big data analytics framework, available at: *https://github.com/dataplumber/nexus*