Remote Sensing of Environment 202 (2017) 18–27

Contents lists available at ScienceDirect

Remote Sensing of Environment

journal homepage: www.elsevier.com/locate/rse

Google Earth Engine: Planetary-scale geospatial analysis for everyone

Noel Gorelick [a,*], Matt Hancher [b], Mike Dixon [b], Simon Ilyushchenko [b], David Thau [b], Rebecca Moore [b]

[a] Google Switzerland, Brandschenkestrasse 110, Zurich 8002, Switzerland
[b] Google Inc., 1600 Amphitheater Parkway, Mountain View, CA, 94043, USA

ARTICLE INFO

Article history:
Received 9 July 2016
Received in revised form 5 June 2017
Accepted 27 June 2017
Available online 6 July 2017

Keywords:
Cloud computing
Big data
Analysis
Platform
Data democratization
Earth Engine

ABSTRACT

Google Earth Engine is a cloud-based platform for planetary-scale geospatial analysis that brings Google's massive computational capabilities to bear on a variety of high-impact societal issues including deforestation, drought, disaster, disease, food security, water management, climate monitoring and environmental protection. It is unique in the field as an integrated platform designed to empower not only traditional remote sensing scientists, but also a much wider audience that lacks the technical capacity needed to utilize traditional supercomputers or large-scale commodity cloud computing resources.

1. Introduction

Supercomputers and high-performance computing systems are becoming abundant (Cossu et al., 2010; Nemani et al., 2011) and large-scale cloud computing is universally available as a commodity. At the same time, petabyte-scale archives of remote sensing data have become freely available from multiple U.S. Government agencies including NASA, the U.S. Geological Survey, and NOAA (Woodcock et al., 2008; Loveland and Dwyer, 2012; Nemani et al., 2011), as well as the European Space Agency (Copernicus Data Access Policy, 2016), and a wide variety of tools have been developed to facilitate large-scale processing of geospatial data, including TerraLib (Câmara et al., 2000), Hadoop (Whitman et al., 2014), GeoSpark (Yu et al., 2015), and GeoMesa (Hughes et al., 2015).

Unfortunately, taking full advantage of these resources still requires considerable technical expertise and effort. One major hurdle is in basic information technology (IT) management: data acquisition and storage; parsing obscure file formats; managing databases, machine allocations, jobs and job queues, CPUs, GPUs, and networking; and using any of the multitudes of geospatial data processing frameworks.

This burden can put these tools out of the reach of many researchers and operational users, restricting access to the information contained within many large remote-sensing datasets to remote-sensing experts with special access to high-performance computing resources.

Google Earth Engine is a cloud-based platform that makes it easy to access high-performance computing resources for processing very large geospatial datasets, without having to suffer the IT pains currently surrounding either. Additionally, and unlike most supercomputing centers, Earth Engine is also designed to help researchers easily disseminate their results to other researchers, policy makers, NGOs, field workers, and even the general public. Once an algorithm has been developed on Earth Engine, users can produce systematic data products or deploy interactive applications backed by Earth Engine's resources, without needing to be an expert in application development, web programming or HTML.

2. Platform overview

Earth Engine consists of a multi-petabyte analysis-ready data catalog co-located with a high-performance, intrinsically parallel computation service. It is accessed and controlled through an Internet-accessible application programming interface (API) and an associated web-based interactive development environment (IDE) that enables rapid prototyping and visualization of results.

The data catalog houses a large repository of publicly available geospatial datasets, including observations from a variety of satellite and aerial imaging systems in both optical and non-optical wavelengths, environmental variables, weather and climate forecasts and hindcasts, land cover, topographic and socio-economic datasets. All of this data is preprocessed to a ready-to-use but information-preserving form that allows efficient access and removes many barriers associated with data management.

Users can access and analyze data from the public catalog as well as their own private data using a library of operators provided by the Earth Engine API. These operators are implemented in a large parallel

* Corresponding author.
E-mail address: gorelick@google.com (N. Gorelick).

http://dx.doi.org/10.1016/j.rse.2017.06.031

processing system that automatically subdivides and distributes computations, providing high-throughput analysis capabilities. Users access the API either through a thin client library or through a web-based interactive development environment built on top of that client library (Fig. 1).

Users can sign up for access at the Earth Engine homepage, https://earthengine.google.com, and access the user interface, as well as a user's guide, tutorials, examples, training videos, function reference, and educational curricula. While prior experience with GIS, remote sensing and scripting make it easier to get started, they are not strictly required, and the user's guide is oriented towards domain novices. Accounts come with a quota for uploading personal data and saving intermediate products, and any inputs or results can be downloaded for offline use.

3. The data catalog

The Earth Engine public data catalog is a multi-petabyte curated collection of widely used geospatial datasets. The bulk of the catalog is made up of Earth-observing remote sensing imagery, including the entire Landsat archive as well as complete archives of data from Sentinel-1 and Sentinel-2, but it also includes climate forecasts, land cover data and many other environmental, geophysical and socio-economic datasets (Table 1). The catalog is continuously updated at a rate of nearly 6000 scenes per day from active missions, with a typical latency of about 24 h from scene acquisition time. Users can request the addition of new datasets to the public catalog, or they can upload their own private data *via* a REST interface using either browser-based or command-line tools and share with other users or groups as desired.

Earth Engine uses a simple and highly general data model based on 2D gridded raster bands in a lightweight "image" container. Pixels in an individual band must be homogeneous in data type, resolution and projection. However, images can contain any number of bands and the bands within an image need not have uniform data types or projections. Each image can also have associated key/value metadata containing information such as the location, acquisition time, and the conditions under which the image was collected or processed.

Related images, such as all of the images produced by a single sensor, are grouped together and presented as a "collection". Collections provide fast filtering and sorting capabilities that make it easy for users to search through millions of individual images to select data that meets specific spatial, temporal or other criteria. For example, a user can easily select daytime images from the Landsat 7 sensor that cover any part of Iowa, collected on day-of-year 80 to 104, from the years 2010 to 2012, with less than 70% cloud cover.

Images ingested into Earth Engine are pre-processed to facilitate fast and efficient access. First, images are cut into tiles in the image's original projection and resolution and stored in an efficient and replicated tile database. A tile size of 256 × 256 was chosen as a practical trade-off between loading unneeded data *vs.* the overhead of issuing additional reads. In contrast to conventional "data cube" systems, this data ingestion process is information-preserving: the data are always maintained in their original projection, resolution and bit depth, avoiding the data degradation that would be inherent in resampling all data to a fixed grid that may or may not be appropriate for any particular application.

Additionally, in order to enable fast visualization during algorithm development, a pyramid of reduced-resolution tiles is created for each image and stored in the tile database. Each level of the pyramid is created by downsampling the previous level by a factor of two until the entire image fits into a single tile. When downsampling, continuous-valued bands are typically averaged, while discrete-valued bands, such

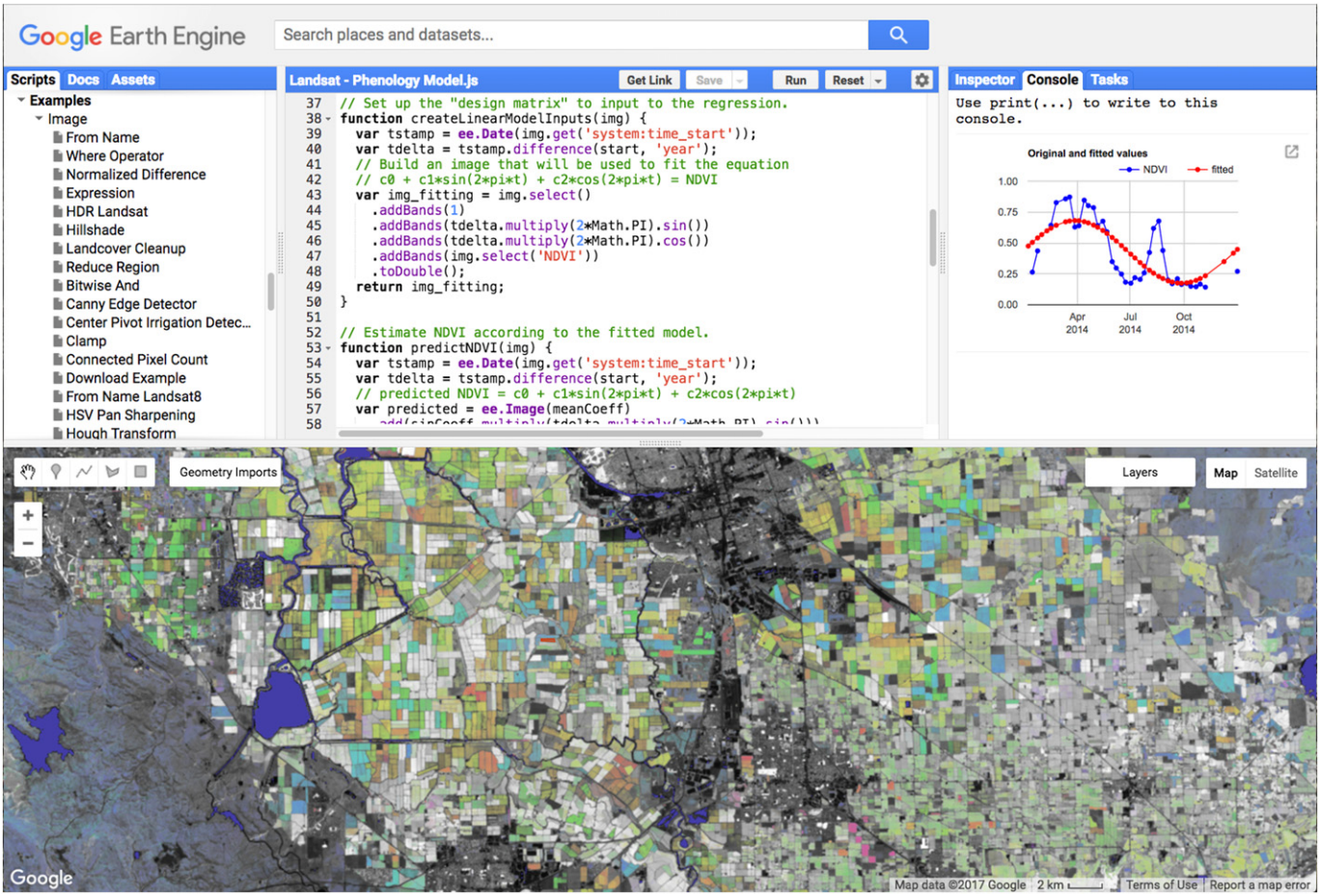

Fig. 1. The Earth Engine interactive development environment.

Table 1
Frequently used datasets in the earth engine data catalog.

Dataset	Nominal resolution	Temporal granularity	Temporal coverage	Spatial coverage
Landsat				
Landsat 8 OLI/TIRS	30 m	16 day	2013–Now	Global
Landsat 7 ETM +	30 m	16 day	2000–Now	Global
Landsat 5 TM	30 m	16 day	1984–2012	Global
Landsat 4–8 surface reflectance	30 m	16 day	1984–Now	Global
Sentinel				
Sentinel 1 A/B ground range detected	10 m	6 day	2014–Now	Global
Sentinel 2A MSI	10/20 m	10 day	2015–Now	Global
MODIS				
MOD08 atmosphere	1°	Daily	2000–Now	Global
MOD09 surface reflectance	500 m	1 day/8 day	2000–Now	Global
MOD10 snow cover	500 m	1 day	2000–Now	Global
MOD11 temperature and emissivity	1000 m	1 day/8 day	2000–Now	Global
MCD12 Land cover	500 m	Annual	2000–Now	Global
MOD13 Vegetation indices	500/250 m	16 day	2000–Now	Global
MOD14 Thermal anomalies & fire	1000 m	8 day	2000–Now	Global
MCD15 Leaf area index/FPAR	500 m	4 day	2000–Now	Global
MOD17 Gross primary productivity	500 m	8 day	2000–Now	Global
MCD43 BRDF-adjusted reflectance	1000/500 m	8 day/16 day	2000–Now	Global
MOD44 veg. cover conversion	250 m	Annual	2000–Now	Global
MCD45 thermal anomalies and fire	500 m	30 day	2000–Now	Global
ASTER				
L1 T radiance	15/30/90 m	1 day	2000–Now	Global
Global emissivity	100 m	Once	2000–2010	Global
Other imagery				
PROBA-V top of canopy reflectance	100/300 m	2 day	2013–Now	Global
EO-1 hyperion hyperspectral radiance	30 m	Targeted	2001–Now	Global
DMSP-OLS nighttime lights	1 km	Annual	1992–2013	Global
USDA NAIP aerial imagery	1 m	Sub-annual	2003–2015	CONUS
Topography				
Shuttle Radar Topography Mission	30 m	Single	2000	60°N–54°S
USGS National Elevation Dataset	10 m	Single	Multiple	United States
USGS GMTED2010	7.5″	Single	Multiple	83°N–57°S
GTOPO30	30″	Single	Multiple	Global
ETOPO1	1′	Single	Multiple	Global
Landcover				
GlobCover	300 m	Non-periodic	2009	90°N–65°S
USGS National Landcover Database	30 m	Non-periodic	1992–2011	CONUS
UMD global forest change	30 m	Annual	2000–2014	80°N–57°S
JRC global surface water	30 m	Monthly	1984–2015	78°N–60°S
GLCF tree cover	30 m	5 year	2000–2010	Global
USDA NASS cropland data layer	30 m	Annual	1997–2015	CONUS
Weather, precipitation & atmosphere				
Global precipitation measurement	6′	3 h	2014–Now	Global
TRMM 3B42 precipitation	15′	3 h	1998–2015	50°N–50°S
CHIRPS precipitation	3′	5 day	1981–Now	50°N–50°S
NLDAS-2	7.5′	1 h	1979–Now	North America
GLDAS-2	15′	3 h	1948–2010	Global
NCEP reanalysis	2.5°	6 h	1948–Now	Global
ORNL DAYMET weather	1 km	Annual	1980–Now	North America
GRIDMET	4 km	1 day	1979–Now	CONUS
NCEP global forecast system	15′	6 h	2015–Now	Global
NCEP climate forecast system	12′	6 h	1979–Now	Global
WorldClim	30″	12 images	1960–1990	Global
NEX downscaled climate projections	1 km	1 day	1950–2099	North America
Population				
WorldPop	100 m	5 year	Multiple	2010–2015
GPWv4	30″	5 year	2000–2020	85°N–60°S

as classification labels, are sampled using one of min, mode, max or fixed sampling. When a portion of data from an image is requested for computation at a reduced resolution, only the relevant tiles from the most appropriate pyramid level need to be retrieved from the tile database. This power-of-two downscaling enables having data ready at a variety of scales without introducing significant storage overhead, and aligns with the common usage patterns in web-based mapping.

4. System architecture

Earth Engine is built on top of a collection of enabling technologies that are available within the Google data center environment, including the Borg cluster management system (Verma et al., 2015); the Bigtable (Chang et al., 2008) and Spanner (Corbett et al., 2013) distributed databases; Colossus, the successor to the Google File System (Ghemawat et al., 2003; Fikes, 2010); and the FlumeJava framework for parallel pipeline execution (Chambers et al., 2010). Earth Engine also interoperates with Google Fusion Tables (Gonzalez et al., 2010), a web-based database that supports tables of geometric data (points, lines, and polygons) with attributes.

A simplified system architecture is shown in Fig. 2. The Earth Engine Code Editor and third-party applications use client libraries to send interactive or batch queries to the system through a REST API. On-the-fly requests are handled by Front End servers that forward complex sub-queries to Compute Masters, which manage computation distribution among a pool of Compute Servers. The batch system operates in a

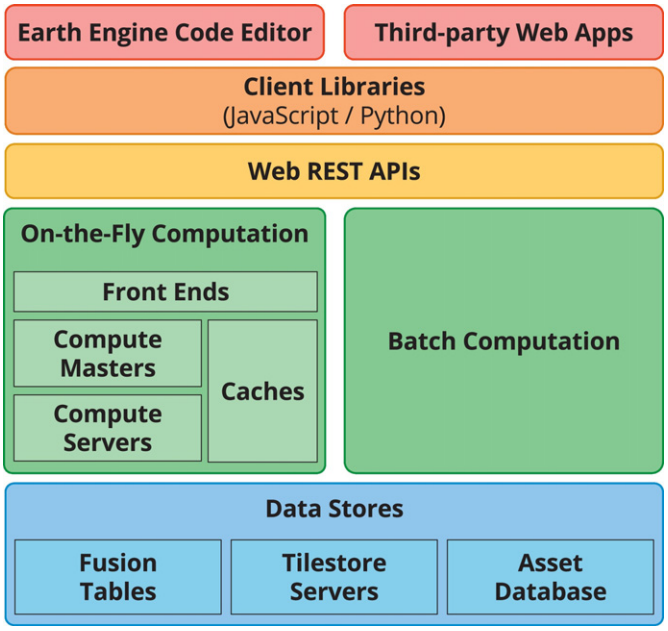

Fig. 2. A simplified system architecture diagram.

similar manner, but uses FlumeJava to manage distribution. Backing both computation systems are a collection of data services, including an Asset Database that contains the per-image metadata and provides efficient filtering capabilities. The Borg cluster management software manages each component of the system and each service is load-balanced over multiple workers. Failure of any individual worker just results in the caller reissuing the query.

Queries to Earth Engine are based on functional composition and evaluation. Users construct queries by chaining together operations drawn from the Earth Engine library of more than 800 functions, which range in complexity from simple mathematical functions to powerful geostatistical, machine learning, and image processing operations. The library makes it easy to express operations between images using a form of image algebra, and supports higher-order functions: map() and iterate() allow applying arbitrary functions to collections of images, while reduce() is used to compute statistical results in a variety of ways including regional, sliding-window, temporal, spectral and columnar contexts. Table 2 summarizes the types of operations available in the client library.

The bulk of the library's image-based functions are per-pixel algebraic operations that operate on a per-band or band-to-band basis, covering integer and floating point math, logical comparisons, bit manipulation, type casting, conditional replacement and multidimensional array operations for processing on array-valued pixels. Also included are common pixel manipulation functions such as table lookup, piecewise-linear interpolation, polynomial evaluation and the ubiquitous normalized difference. The library leverages several pre-existing machine-learning toolkits to provide easy access to more than 20 types of supervised classification, regression and unsupervised clustering, as well as operations on confusion matrices for accuracy assessment. For machine-vision tasks, common kernel-based windowing operations such as convolution, morphological operations, distance and texture analysis are available as well as simple neighbor-based operations such as gradient, slope, aspect and connectedness. Other capabilities include image and band metadata operations, projection and resampling manipulations, masking and clipping, image-to-image displacement and coregistration and a variety of specialized tools common to remote sensing applications including constrained spectral unmixing, region growing and cost mapping operations.

These library functions can be composed to build up a description of the computation the user wishes to perform. This computational description ultimately takes the form of a directed acyclic graph (DAG) in which each node represents the execution of an individual function or data accessor and contains key/value pairs of named function arguments. This is, in essence, a pure functional programming environment, and Earth Engine takes advantage of standard techniques commonly used by functional languages, such as referential transparency and lazy evaluation, for significant optimization and efficiency gains.

Users write Earth Engine programs using client libraries (currently available for the Python and JavaScript languages) that allow the user to describe processing graphs using a familiar procedural programming paradigm. The client libraries provide proxy objects for Images, Collections, and other data types such as numbers, strings, geometries, and lists. User scripts manipulate these proxy objects, which record the chain of operations and assemble them into a DAG that expresses the complete computation. This DAG is then sent to the Earth Engine service for evaluation.

DAGs are evaluated through a sequence of graph transformations. Subgraphs are greedily simplified by immediate evaluation where possible, to avoid redundant computation and anywhere a parallel implementation isn't available. For example, a subgraph representing $3 + 7$ will be immediately simplified to the value 10. Other nodes in the graph are expanded, for example when a node that refers to a collection of images is evaluated it is expanded to a sequence of images to be consumed in batches by subsequent processing operations. Nodes that represent complex processing operations may employ any of several strategies for distributed processing described in the next section.

Earth Engine is designed to support fast, interactive exploration and analysis of spatial data, allowing the user to pan and zoom through results to examine a subset of the image at a time. To facilitate this, Earth Engine uses a lazy computation model that allows it to compute only the portions of output that are necessary to fulfill the current request.

As an illustrative example, a user might wish to compute the difference between two seasonal composites, to highlight the changes due to phenology or snow-cover. A simplistic example of this could be expressed using the Earth Engine client library as the subtraction of two composite images (Listing 1). This code creates two filtered collections, one of all Landsat 8 images for November, December, and January, and a second of all Landsat 8 images from June, July, and August. A temporal median value is computed for each band in each collection (to minimize the effects of cloud and cloud shadows), and the resulting composites are subtracted to compute the change in values. This computational description is represented by the DAG in Fig. 3.

```
collection = ee.ImageCollection("LANDSAT8")
winter = collection.filter(ee.Filter.calendarRange(11, 1, "month"))
summer = collection.filter(ee.Filter.calendarRange(6, 8, "month"))
diff = summer.median().subtract(winter.median())
```

Listing 1. Computing the difference of median composites from two seasons.

A traditional (non-lazy) computing environment might start computing the pixels for one or both of the composites as soon as the expression is processed, which typically requires the input datasets to be preprocessed to a common map projection, resolution, and region of interest in advance.

Instead, Earth Engine takes a different approach: it postpones computing output pixels until it knows more about the context in which they are needed. For example, if the result is being displayed on an interactive map then the map's zoom level and view bounds can dynamically determine the projection and resolution of the output, and can constrain the pixel computation to just the pixels that are viewable. Alternatively, if the result is being used as an input to another computation, then that computation can request an appropriate projection, resolution, and bounds for the pixels needed. This information is used to automatically resample and reproject input data on the fly, making it possible to rapidly visualize results or to use that expression in a

Table 2
Earth Engine function summary.

Function category	Examples	Mode of operation
Numerical operations		
Primitive operations	add, subtract, multiply, divide, *etc.*	Per pixel/per feature
Trigonometric operations	cos, sin, tan, acos, asin, atan, *etc.*	
Standard functions	abs, pow, sqrt, exp., log, erf, *etc.*	
Logical operations	eq, neq, gt, gte, lt, lte, and, or	
Bit/bitwise operations	and, or, xor, not, bit shift, *etc.*	
Numeric casting	int, float, double, uint8, *etc.*	
Array/matrix operations		
Elementwise operations	(numeric operations as above)	Per pixel/per feature
Array manipulation	Get, length, cat, slice, sort, *etc.*	
Array construction	Identity, diagonal, *etc.*	
Matrix operations	Product, determinant, transpose, inverse, pseudoinverse, decomposition, *etc.*	
Reduce and accumulate	Reduce, accum	
Machine learning		
Supervised classification and regression	Bayes, CART, Random Forest, SVM, Perceptron, Mahalanobis, *etc.*	Per pixel/per feature
Unsupervised Classification	K-Means, LVQ, Cobweb, *etc.*	
Other per-pixel image operations		
Spectral operations	Unmixing, HSV transform, *etc.*	Per pixel
Data masking	Unmask, update mask, *etc.*	
Visualization	Min/max, color palette, gamma, SLD, *etc.*	
Location	Pixel area, pixel coordinates, *etc.*	
Kernel operations		
Convolution	Convolve, blur, *etc.*	Per image tile
Morphology	Min, max, mean, distance, *etc.*	
Texture	Entropy, GLCM, *etc.*	
Simple shape kernels	Circle, rectangle, diamond, cross, *etc.*	
Standard kernels	Gaussian, Laplacian, Roberts, Sobel, *etc.*	
Other kernels	Euclidean, Manhattan and Chebyshev distance, arbitrary kernels and combinations	
Other Image Operations		
Band manipulation	Add, select, rename, *etc.*	Per image
Metadata properties	Get, set, *etc.*	
Derivative	Pixel-space derivative, spatial gradient	
Edge detection	Canny, Hough transform	
Terrain operations	Slope, aspect, hillshade, fill minima, *etc.*	
Connected components	Components, component size	
Image clipping	Clip	
Resampling	Bilinear, bicubic, *etc.*	
Warping	Translate, changeProj	
Image registration	Register, displacement, displace	
Other tile-based operations	Cumulative cost, medial axis, reduce resolution with arbitrary reducers, *etc.*	
Image aggregations	Sample region(s), reduce region(s) with arbitrary reducers	
Reducers		
Simple	Count, distinct, first, *etc.*	Context-dependent
Mathematical	sum, product, min, max, *etc.*	
Logical	Logical and/or, bitwise and/or	
Statistical	Mean, median, mode, percentile, standard deviation, covariance, histogram, *etc.*	
Correlation	Kendall, Spearman, Pearson, Sen's slope	
Regression	Linear regression, robust linear regression	
Geometry Operations		
Types	Point, LineString, Polygon, *etc.*	Per-feature
Measurements	Length, area, perimeter, distance, *etc.*	
Constructive operations	Intersection, union, difference, *etc.*	
Predicates	Intersects, contains, withinDistance, *etc.*	
Other operations	Buffer, centroid, transform, simplify, *etc.*	
Table/collection operations		
Basic manipulation	Sort, merge, size, first, limit, distinct, flatten, remap, *etc.*	Streaming
Property filtering	eq, neq, gt, lt, date range, and, or, not, *etc.*	
Spatial filtering	Intersects, contains, withinDistance, *etc.*	
Parallel processing	Map, reduce, iterate	
Joins	Simple, inner, grouping, *etc.*	
Vector/raster operations		
Rasterization	Paint/draw, distance	Per tile
Spatial interpolation	Kriging, IDW interpolation	
Vectorization	reduceToVectors	Scatter/gather
Other data types		
Number, string, list, dictionary, date, daterange, projection, *etc.*		Context-dependent

more complex calculation without requiring the user to pre-specify which pixels will be needed from it. Reprojection and resampling to the requested output projection is by default performed using nearest-neighbor resampling of the input(s), to preserve spectral integrity, selecting pixels from the next-highest-resolution pyramid level of each input. However, when the user has preferences for how this reprojection is managed, they have the option of precisely controlling the projection grid and can choose from bilinear and bicubic sampling modes.

This approach encourages an interactive and iterative mode of data exploration and algorithm development. Once a user has developed an algorithm that they would like to apply at scale, they may submit a

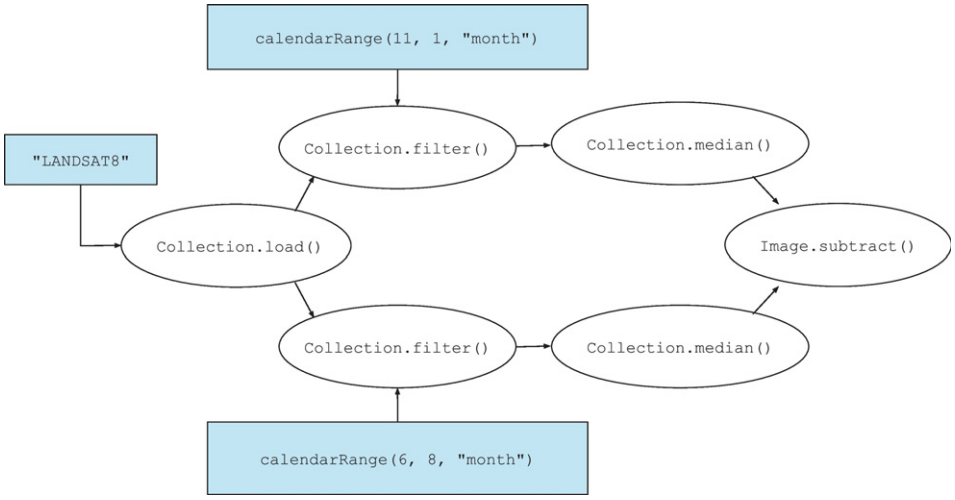

Fig. 3. The DAG produced for Listing 1.

batch-processing request to Earth Engine to compute the complete result and materialize it either as an image in Earth Engine or as one or more image, table, or video files for download.

5. Data distribution models

The functions in the Earth Engine library utilize several built-in parallelization and data distribution models to achieve high performance. Each of these models is optimized for a different data access pattern.

5.1. Image tiling

Many raster processing operations used in remote sensing are local: the computation of any particular output pixel depends only on input pixels within some fixed distance. Examples include per-pixel operations such as band math or spectral unmixing, as well as neighborhood operations such as convolution or texture analysis. These operations can be easily processed in parallel by subdividing an area into tiles and computing each independently. Processing each output tile usually requires retrieving only one or a small number of tiles for each input. This fact, combined with pyramided inputs and judicious caching, allows for fast computation of results at any requested scale or projection. As previously mentioned, inputs are reprojected on the fly as needed to match the requested output projection. However, if the user determines that using downsampled or reprojected inputs is undesired, they are free to explicitly specify computation in the input's projection and scale.

Most tile-based operations are implemented in Earth Engine using one of two strategies, depending on their computational cost. Expensive operations, and operations that benefit significantly from computing an entire tile at once, write results into a tile-sized output buffer. Tiles are typically 256 × 256 pixels, to match the tiling-size of the input preprocessing.

Inexpensive per-pixel operations are implemented using a pixel-at-a-time interface in which the image processing operations in a graph directly invoke one another. This structure is designed to take advantage of the fact that these operations execute in a Java Virtual Machine (JVM) environment with a Just-In-Time (JIT) compiler that extracts and compiles sequences of function calls that occur repeatedly. The result is that in many cases, arbitrary chains of primitive image operations such as band math can execute almost as efficiently as hand-built compiled code. Experiments detailing these efficiency gains are discussed in Section 6.

5.2. Spatial aggregations

Just as some classes of computation are inherently local, others are inherently non-local, such as the computation of regional or global statistics, raster-to-vector conversion, or sampling an image to train a classifier. These operations, or portions of them, can often still be performed in parallel, but computing the final result requires aggregating together many sub-results. For example, computing the mean value of an entire image can be performed by subdividing the image, computing sums and counts in parallel over each portion, and then summing these partial sums and counts to compute the desired result.

In Earth Engine these types of computations are executed as distributed processes using a scatter-gather model. The spatial region over which an aggregation is to be performed is divided into subregions that are assigned to workers in a distributed worker pool, to be evaluated in batches. Each worker fetches or computes the input pixels that it needs and then runs the desired accumulation operation to compute its partial results. These results are sent back to the master for this computation, which combines them and transforms the result into the final form. For example, when computing a mean value each worker will compute sums and counts, the master collects and sums these intermediates, and the final result will be the total sum divided by the total count.

This model is very similar to a traditional Map/Reduce with a fixed pool of mappers and a single reducer, however the user need not be aware of this implementation and need only specify the map projection, resolution and spatial region in which to perform the operation, which in turn determines the grid in which the input pixels will be computed and the number of subregions. Typically each subregion is a multiple of the default input tile size (usually 1024 × 1024 pixels) to minimize the RPC overhead during these computations. However, because of the large range of computational complexity of the intermediate products over which users might be attempting to aggregate, controls were introduced to the system to allow users to adjust this multiple, should their computation require it, *e.g.* due to per-worker memory limitations.

5.3. Streaming collections

Another common operation in the processing of large remote-sensing datasets is time-series analysis. The same statistical aggregation operations that can be applied spatially can also be applied temporally over the images in a collection to compute per-pixel statistics of an entire image stack through time. These operations are performed using a combination of tiling and aggregation. Each output tile is computed in

parallel using lazy image evaluation, in the manner described above. Within each tile, an aggregation operation is performed at each pixel. Tiles of pixel data from the images in the input collection are requested in batches and "streamed" one at a time through the per-pixel aggregators. Once all inputs that intersect the output tile have been processed, the final transformation is applied at each pixel to generate the output result.

This distribution model can be fast and efficient for aggregations that have a small intermediate state (*e.g.* computing the minimum value), but it can be prohibitively memory-intensive for those that don't (*e.g.*: Pearson's correlation, which requires storing the complete data series at each pixel prior to computing the final result). Streaming through even very large collections can still be fast as long as the size of a tile is significantly smaller than a full image. For example, the entire stack of Landsat 5, 7 and 8 collections, collectively containing more than 5 million images, is less than 2000 tiles deep at any point, and only about 500 deep on average.

5.4. Caching and common sub-expression elimination

Many processing operations in Earth Engine can be expensive and data-intensive, so it can be valuable to avoid redundant computation. For example, a single user viewing results on a map will trigger multiple independent requests for output tiles all of which frequently depend on one or more common sub-expressions, such as a large spatial aggregation or the training of a supervised classifier. To avoid re-computing values that have already been previously requested, expensive intermediate results are stored in a distributed cache using a hash of the subgraph as a cache key.

While it is possible that multiple users could share an item in the cache, it is uncommon that two separate users independently make identical queries. However, it is very common for a single user to repeat the same queries during incremental algorithm development and thus to benefit from this mechanism. The cache is also used as a form of shared memory during distributed execution of a single query, storing the intermediate results corresponding to subgraphs of the query.

When subsequent requests for the same computation arrive, the earlier computation may already have completed or it may still be in progress. Previously computed results are simply retrieved and returned by checking the cache prior to starting expensive operations. To handle the case in which the earlier computation is still in progress, all computations are sent to distributed workers *via* a small number of computation master servers. These servers track the computations that are executing in the cluster at any given moment. When a new query arrives that depends on some computation already in progress, that query will simply join the original query in waiting for the computation to complete. Should a compute master fail, handles to in-progress computations could be lost, possibly allowing redundant computations to start, but only if a query is re-requested before the existing ones finish.

6. Efficiency, performance, and scaling

Earth Engine takes advantage of the Java Just-In-Time (JIT) compiler to optimize the execution of chains of per-pixel operations that are common in image processing. To evaluate the efficiency gains provided by the JIT compiler, a series of experiments were conducted to compare the performance of three execution models: executing a computation graph in Java using the JIT compiler; executing a graph using a similar general implementation in C++; and finally, specialized native C++ code in which the same calls are made directly instead of through a graph, thereby avoiding function virtualization. Five test cases, each testing a different type of image computation graph, were explored:

- SingleNode: A trivial graph with a single node consisting of an image data buffer. This test simply computes the sum of all the values in a buffer.
- NormalizedDifference: A graph that computes the normalized difference of two input buffers. This small-graph scenario contains five nodes in total: two input nodes, one sum, one product, and one quotient.
- DeepProduct: A graph that consists of 64 binary product nodes in a chain, computing the product of 65 input nodes.
- DeepCosineSum: A graph with the same structure as DeepProduct, but where each node computes the more expensive binary operation cos(a + b).
- SumOfProducts: A graph that computes the sum over all pairwise products of 40 inputs. This graph has 40 input nodes, 780 product notes, and a tree of 779 sum nodes. Here the total number of nodes is much larger than the number of input nodes, allowing us to evaluate the performance of complex graphs of primitive operations on fixed amounts of input data, a common real-world scenario.

Each of these tests was performed on a single 256 × 256-pixel tile using a single-threaded execution environment on an Intel Sandy Bridge processor at 2.6 GHz, a configuration that is representative of commercial cloud data center environments, with all non-essential system services disabled to minimize profiling noise. The results, summarized in Table 3, show that in 4 out of 5 of these common test cases, Java with the JIT compiler outperforms similar dynamic graph-based computation in C++ by as much as 50%, and in one case it even outperforms a direct C++ implementation.

6.1. System throughput performance

In the Google data center, CPUs are abundant. In this environment raw efficiency, while still important, is not as important as the ability to efficiently distribute complex computations across many machines and much of Earth Engine's performance is due to its ability to marshal and manage a large number of CPUs on a user's behalf. There is a hard ultimate upper limit on the efficiencies that can be achieved through code or query optimization, but there are fewer limitations on the additional computing resources that can be brought to bear.

Experiments were conducted to demonstrate Earth Engine's ability to scale horizontally (Fig. 4). In this test, two large collections of Landsat images were reprojected to a common projection, temporally aggregated on a per-pixel basis and spatially aggregated down to a single number while varying the number of CPUs per run. The two collections consisted of all available Landsat 8 Level-1T images acquired from 2014 to 01-01 to 2016–12-31, covering CONUS (26,961 scenes, 1.21 trillion pixels) and Africa (77,528 scenes, 3.14 trillion pixels). Tests were run using shared production resources over multiple days and times to capture the natural variability due to load on the fleet. The results show a nearly linear scaling in throughput with the number of machines.

7. Applications

Earth Engine is in use across a wide variety of disciplines, covering topics such as global forest change (Hansen et al., 2013), global surface water change (Pekel et al., 2016), crop yield estimation (Lobell et al., 2015), rice paddy mapping (Dong et al., 2016), urban mapping (Zhang et al., 2015; Patel et al., 2015), flood mapping (Coltin et al., 2016), fire

Table 3
Results from Java JIT *vs.* C++ efficiency tests.

Test case	C++ function	C++ graph	Java graph
SingleNode	0.057 ms	0.17 ms	0.056 ms
NormalizedDifference	0.40 ms	0.95 ms	0.41 ms
DeepProduct	18 ms	55 ms	43 ms
DeepCosineSum	160 ms	200 ms	240 ms
SumOfProducts	110 ms	790 ms	360 ms

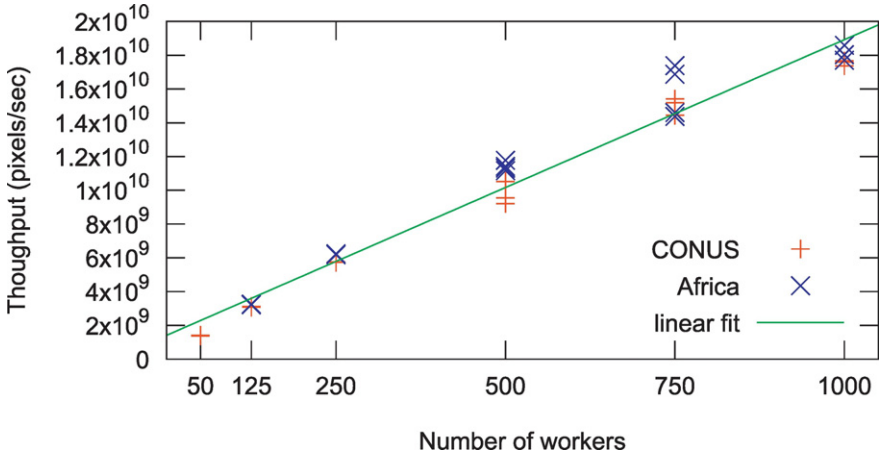

Fig. 4. Horizontal scaling tests results.

recovery (Soulard et al., 2016) and malaria risk mapping (Sturrock et al., 2014). It has also been integrated into a number of third-party applications, for example analyzing species habitat ranges (Map of Life, 2016), monitoring climate (Climate Engine, 2016), and assessing land use change (Collect Earth, 2016). The details of a few of these applications will illustrate how Earth Engine's capabilities are being leveraged.

Hansen et al. (2013) characterized forest extent, loss, and gain from 2000 to 2012 using decision trees generated from an extensive set of training data and a deep stack of metrics computed from large collections of Landsat scenes. Filtering operations supported by the data catalog reduced the 1.3 M Landsat scenes available at the time to 654,178 growing-season scenes from the study period. These images were then screened for clouds, cloud shadows, and water, and converted from raw Landsat digital numbers to normalized top of atmosphere reflectance. All necessary data access, format conversion, cartographic reprojection, and resampling were handled automatically by the system. Operations in the API were used to compute input metrics, such as per-band percentile values and linear regressions of reflectance values *versus* image date. These metrics, along with training data, were used to generate decision trees, which were applied to the metrics globally to produce the final output data. Those results were used for publication and made available as part of the Earth Engine catalog for further analysis by others.

Many other users, both scientific and operational, have since successfully built on the Hansen results to produce derivative results using Earth Engine. Global Forest Watch (2014) incorporated it into an interactive analysis application using Earth Engine to perform on-the-fly calculations of summary statistics. Joshi et al. (2016) used it to track changes in tiger habitat by extracting forest loss within protected areas for each year, finding that the areas most suitable for doubling the wild tiger population were also the best protected.

In another example, Lobell et al. (2015) related the output of hundreds of crop model simulations to vegetation indices, such as the green chlorophyll vegetation index (GCVI), that are measurable with satellite data. They then related simulated yields to measured vegetative indices and weather for a set of dates early in the growing season and late in the growing season. This resulted in a table of regression coefficients for each pairwise combination of early/late dates. They used Earth Engine to select, on a per-pixel basis, the best Landsat scenes for the early and late periods, by first calculating reflectance of the Landsat scenes using LEDAPS (Masek et al., 2006), automatically removing cloudy scenes using Earth Engine's SimpleCloudScore function, computing GCVI values, and finally selecting the scenes with the highest GCVI. Once the best pair of Landsat scenes was determined for a given pixel, weather data stored in Earth Engine and the GCVI were used to compute the predicted yield. This method was applied to roughly 6.75 million hectares of maize and soy fields in the Midwestern United States to compute annual yields from 2008 to 2012. The total computation completed in approximately 2 min per 10,000 km^2 per year.

8. Challenges and future work

One of the benefits of using Earth Engine is that the user is almost completely shielded from the details of working in a parallel processing environment. The system handles and hides nearly every aspect of how a computation is managed, including resource allocation, parallelism, data distribution, and retries. These decisions are purely administrative, and none of them can affect the result of a query, only the speed at which it is produced. The price of liberation from these details is that the user is unable to influence them: the system is entirely responsible for deciding how to run a computation. This results in some interesting challenges in both the design and use of the system.

8.1. Scaling challenges

The Earth Engine system as a whole can manage extremely large computations, but the underlying infrastructure is ultimately clusters of low-end servers (Barroso et al., 2013). In this environment, the option of configuring arbitrarily large machines is not available, and there is a hard limit on the amount of data that can be brought into any individual server. This means that users can only express large computations by using the parallel processing primitives provided in the Earth Engine library and some non-parallelizable operations simply cannot be performed effectively in this environment. Additionally, the requirement to express computations using the Earth Engine library means that existing algorithms and workflows have to be converted to use the Earth Engine API to utilize the platform at all.

Earth Engine's API by design makes it easy to express extremely large computations. For example, it would take only a few lines of Earth Engine code to request a global aggregation of the 800 billion pixel Hansen forest cover map: while this computation is straightforward, simply retrieving all the input pixels from storage involves a substantial quantity of resources for a significant length of time. By chaining operations on large collections of data over a wide range of spatial scales, it is easy to express queries that vary in cost by many orders of magnitude and to describe computations that are impractical even in an advanced parallel computing environment.

Since Earth Engine is a shared computing resource, limits and other defenses are necessary to ensure that users do not monopolize the system. For interactive sessions, Earth Engine imposes limits on the maximum duration of requests (currently 270 s), the total number of simultaneous requests per user (40), and the number of simultaneous executions of certain expensive operations such as spatial aggregations (25). For illustrative purposes, the interactive computational time-limit

is sufficient to complete the following workflow within a single timeout: Retrieve all Landsat 8 images covering the states of California and Nevada for 1 year (1177 scenes), use them to compute a maximum-NDVI composite, and from that an average peak-NDVI for each of the 17 IGBP land cover classes over the region (735,000 km^2). Note that the bulk of the time for this example is spent in transferring the raw pixels for the full-resolution spatial aggregation; simply creating and displaying the maximum-NDVI composite completes in a few seconds.

None of the interactive limits apply when queries are invoked in a batch context, and jobs that are orders of magnitude larger can run there, but there is still a limit to what each individual machine can accommodate when a request involves tile-based computations that cannot be streamed or further distributed using the current data models. This memory limit does not translate directly into a specific spatial or temporal limit, but a common rule of thumb for the maximum size of these sorts of requests is a stack depth of about 2000 bytes per pixel. The current RPC and caching system imposes an additional limitation that applies in both interactive and batch cases: individual objects to be cached cannot exceed 100 MB in size. This limit is most often encountered when the output of an aggregation operation is large, such as extracting data to train a machine-learning algorithm where it may limit the total number of points in a training set.

Batch jobs are each run independently making it much harder for them to negatively impact each other, but to prevent monopolization, jobs are still managed using a shared queuing system, and under heavy load, jobs may wait in the queue until resources become available.

8.2. Computational model mismatch

While parallelizable operations are very common in remote sensing, there are of course many other classes of operations that are not parallelizable or are not accommodated by the parallel computation constructions available in Earth Engine. The platform is well suited to per-pixel and finite neighborhood operations such as band-math, morphological operations, spectral unmixing, template matching and texture analysis, as well as long chains (hundreds to thousands) of these sorts of operations. It is also highly optimized for statistical operations that can be applied to streaming data, such as computing statistics on a time-series stack of images, and can easily handle very deep stacks this way (ie: millions of images; trillions of pixels). It performs poorly for operations in which a local value can be influenced by arbitrarily distant inputs, such as watershed analysis or classical clustering algorithms; operations that require a large amount of data to be in hand at the same time, such as training many classical machine learning models; and operations that involve long-running iterative processes, such as finite element analysis or agent-based models. Additionally, data intensive models that require large volumes of data not already available in Earth Engine could require substantial additional effort to ingest.

These computational techniques can still be applied in Earth Engine, but often with sharp scaling limits. Extending Earth Engine to support new computational models is an active area of research and development. Users with problems that do not match Earth Engine's computational model can run computations elsewhere in the Google Cloud Platform to capitalize on running computations close to the underlying data and still taking advantage of Earth Engine for its data catalog, preprocessing, post-processing, and visualization.

8.3. Client/server programming model

Earth Engine users are often unfamiliar with the client-server programming model. The Earth Engine client libraries attempt to provide a more familiar procedural programming environment, but this can lead to confusion when the user forgets that their local programming environment (*e.g.* a Python script) is not performing any of the computation itself. The entire chain of operations is recorded by client-side proxy objects and sent to the server for execution, but this means that it is not possible to mix Earth Engine library calls with standard local processing idioms. This includes some basic language features like conditionals and loops that depend on computed values, as well as standard numerical packages. Users can still use these external tools, but they cannot apply them directly to Earth Engine proxy objects, sometimes leading to confusion. Fortunately, these programming errors are usually easy to resolve once identified.

It is worth noting that this style of programming model is becoming increasingly common for large-scale cloud-based computing; it is also used in TensorFlow (Abadi et al., 2016) when constructing and executing graphs.

8.4. Advancing the state of the art

The overarching goal of Earth Engine is to make progress on society's biggest challenges by making it not just possible, but easy, to monitor, track and manage the Earth's environment and resources. Doing so requires providing access not just to vast quantities of data and computational power, but also increasingly sophisticated analysis techniques, and making them easy to use.

To this end, experiments are ongoing in the integration of deep learning techniques (Abadi et al., 2016) and facilitating easy access to other scalable infrastructures such as Google Compute Engine (Gonzalez and Krishnan, 2015) and BigQuery (Tigani and Naidu, 2014).

Acknowledgements

The authors would like to thank the many scientists and practitioners whose requests, advice and critiques inspired the initial creation of Earth Engine, guided its ongoing development and drove positive scientific and societal outcomes; special thanks to Dr. Carlos Souza, Dr. Gilberto Câmara, Prof. Matthew Hansen, Dr. Alan Belward, Prof. Martin Herold, and Prof. Curtis Woodcock. Thanks to Tyler Erickson for providing the script and data visualization shown in Fig. 1. Thanks to the rest of the Earth Engine team: Christiaan Adams, Jeffrey Beis, Peter Birch, Christopher Brown, David Carmichael, Andrew Chang, Nicholas Clinton, Joel Conkling, Michael DeWitt, Eric Engle, Tyler Erickson, Hector Gonzalez, Chris Herwig, Max Heinritz, Rachel Inman, Renee Johnston, Allison Lieber, Igor Nazarenko, Eric Nguyen, Eduardo Poyart, Kevin Reid, Amanda Robinson, Randy Sargent, Nadav Savio, Kurt Schwehr, Lauren Scott, Max Shawabkeh, and Frank Warmerdam. Finally, we would like to thank the providers of the hundreds of public datasets in Earth Engine; in particular, NASA, USGS, NOAA, and EC/ESA, whose enlightened open data policies and practices are responsible for the bulk of the data in our catalog. Special thanks to Tom Loveland of USGS whose team worked with us starting in 2009 for more than three years to bring the entire multi-decadal Landsat archive off tape and online for the first time.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., et al., 2016. Tensorflow: Large-scale Machine Learning on Heterogeneous Distributed Systems. arXiv preprint arXiv:1603.04467.

Barroso, L.A., Clidaras, J., Hölzle, U., 2013. The datacenter as a computer: an introduction to the design of warehouse-scale machines. Synth. Lect. Comput. Archit. 8 (3), 1–154.

Câmara, G., Souza, R., Pedrosa, B., Vinhas, L., Monteiro, A.M.V., Paiva, et al., 2000. TerraLib: technology in support of GIS innovation. Proc. II Brazilian Symposium on GeoInformatics. GeoInfo2000. 2, pp. 1–8.

Chambers, C., Raniwala, A., Perry, F., Adams, S., Henry, R.R., Bradshaw, R., Weizenbaum, N., 2010. FlumeJava: easy, efficient data-parallel pipelines. ACM SIGPLAN Not. 45 (6), 363–375.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., et al., 2008. Bigtable: a distributed storage system for structured data. ACM Trans. Comput. Syst. 26 (2), 4.

Climate Engine, 2016. Desert Research Institute, University of Idaho. http://climateengine.org (accessed July 2016).

Collect Earth, 2016. United Nations Food and Agriculture Organization. http://www.openforis.org/tools/collect-earth.html (accessed July 2016).
Coltin, B., McMichael, S., Smith, T., Fong, T., 2016. Automatic boosted flood mapping from satellite data. Int. J. Remote Sens. 37 (5), 993–1015.
Copernicus Data Access Policy, 2016. http://www.copernicus.eu/main/data-access (accessed June 30, 2016).
Corbett, J.C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J.J., Ghemawat, S., et al., 2013. Spanner: Google's globally distributed database. ACM Trans. Comput. Syst. 31 (3), 8 (TOCS).
Cossu, R., Petitdidier, M., Linford, J., Badoux, V., Fusco, L., Gotab, B., Hluchy, L., et al., 2010. A roadmap for a dedicated earth science grid platform. Earth Sci. Inf. 3 (3).
Dong, J., Xiao, X., Menarguez, M.A., Zhang, G., Qin, Y., Thau, D., Biradar, C., Moore, B., 2016. Mapping paddy rice planting area in northeastern Asia with Landsat 8 images, phenology-based algorithm and Google Earth Engine. Remote Sens. Environ 185, 142–154.
Fikes, A., 2010. Storage Architecture and Challenges. http://goo.gl/pF6kmz (accessed June 30, 2016).
Ghemawat, S., Gobioff, H., Leung, S., 2003. The Google file system. Proc. SOSP 29–43.
Global Forest Watch, 2014. World Resources Institute, Washington, DC. http://www.globalforestwatch.org (accessed June 30, 2016).
Gonzalez, J.U., Krishnan, S.P.T., 2015. Building Your Next Big Thing with Google Cloud Platform: A Guide for Developers and Enterprise Architects. Apress.
Gonzalez, H., Halevy, A.Y., Jensen, C.S., Langen, A., Madhavan, J., Shapley, R., et al., 2010. Google fusion tables: web-centered data management and collaboration. ACM SIGMOD 1061–1066.
Hansen, M.C., Potapov, P.V., Moore, R., Hancher, M., Turubanova, S.A., Tyukavina, A., et al., 2013. High-resolution global maps of 21st-century forest cover change. Science 342, 850–853.
Hughes, J.N., Annex, A., Eichelberger, C.N., Fox, A., Hulbert, A., Ronquest, M., 2015. GeoMesa: a distributed architecture for spatio-temporal fusion. SPIE Defense + Security (pp. 94730F–94730F). Int. Soc. Optics Photonics.
Joshi, A.R., Dinerstein, E., Wikramanayake, E., et al., 2016. Tracking changes and preventing loss in critical tiger habitat. Sci. Adv. 2 (4), e1501675.
Lobell, D., Thau, D., Seifert, C., Engle, E., Little, B., 2015. A scalable satellite-based crop yield mapper. Remote Sens. Environ. 164, 324–333.
Loveland, T.R., Dwyer, J.L., 2012. Landsat: Building a strong future. Remote Sens. Environ. 122, 22–29.
Map of Life, 2016. http://www.mol.org (accessed June 30, 2016).
Masek, J.G., Vermote, E.F., Saleous, N.E., Wolfe, R., Hall, F.G., Huemmrich, K.F., et al., 2006. A Landsat surface reflectance dataset for North America, 1990–2000. Geosci. Remote Sensing Lett. IEEE 3, 68–72.
Nemani, R., Votava, P., Michaelis, A., Melton, F., Milesi, C., 2011. Collaborative supercomputing for global change science. EOS Trans. Am. Geophys. Union 92 (13), 109–110.
Patel, N., Angiuli, E., Gamba, P., Gaughan, A., Lisini, G., Stevens, F., Tatem, A., Trianni, A., 2015. Multitemporal settlement and population mapping from Landsat using google earth engine. Int. J. Appl. Earth Obs. Geoinf. 35, 199–208.
Pekel, J.F., Cottam, A., Gorelick, N., Belward, A.S., 2016. High-resolution mapping of global surface water and its long-term changes. Nature.
Soulard, C.E., Albano, C.M., Villarreal, M.L., Walker, J.J., 2016. Continuous 1985–2012 Landsat monitoring to assess fire effects on meadows in Yosemite National Park, California. Remote Sens. 8 (5), 371.
Sturrock, H.J., Cohen, J.M., Keil, P., Tatem, A.J., Le Menach, A., Ntshalintshali, N.E., Hsiang, M.S., Gosling, R.D., 2014. Fine-scale malaria risk mapping from routine aggregated case data. Malar. J. 13 (1), 1.
Tigani, J., Naidu, S., 2014. Google BigQuery Analytics. John Wiley & Sons.
Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., Wilkes, J., 2015. Large-scale cluster management at Google with Borg. Proc. EuroSys 10, 18. ACM.
Whitman, R.T., Park, M.B., Ambrose, S.M., Hoel, E.G., 2014. Spatial indexing and analytics on hadoop. Proc. 22 ACM SIGSPATIAL. pp. 73–82.
Woodcock, C.E., Allen, A.A., Anderson, M., Belward, A.S., Bindschadler, R., Cohen, W.B., et al., 2008. Free access to Landsat imagery. Science 320, 1011.
Yu, J., Wu, J., Sarwat, M., 2015. Geospark: a cluster computing framework for processing large-scale spatial data. Proc. 23 SIGSPATIAL International Conference on Advances in Geographic Information Systems (p. 70). ACM.
Zhang, Q., Li, B., Thau, D., Moore, R., 2015. Building a better urban picture: combining day and night remote sensing imagery. Remote Sens. 7 (9), 11887–11913.