**YARN has total three major components**

* ResourceManager
* NodeManager
* ApplicationMaster

## 1) ResourceManager

* This daemon process resides on the Master Node (not necessarily on NameNode of Hadoop)
* Responsible for,
  + Managing resources scheduling for different compute applications in an optimum way
  + Coordinating with two process on master node, Scheduler and ApplicationManager

#### Scheduler

* This daemon process resides on the Master Node (runs along with ResourceManager daemon )
* Responsible for,
  + Scheduling the job execution as per submission request received by ResourceManager
  + Allocating resources to applications submitted to the cluster
  + Coordinating with ApplicationManager daemon and keeping track of resources of running applications

#### ApplicationManager

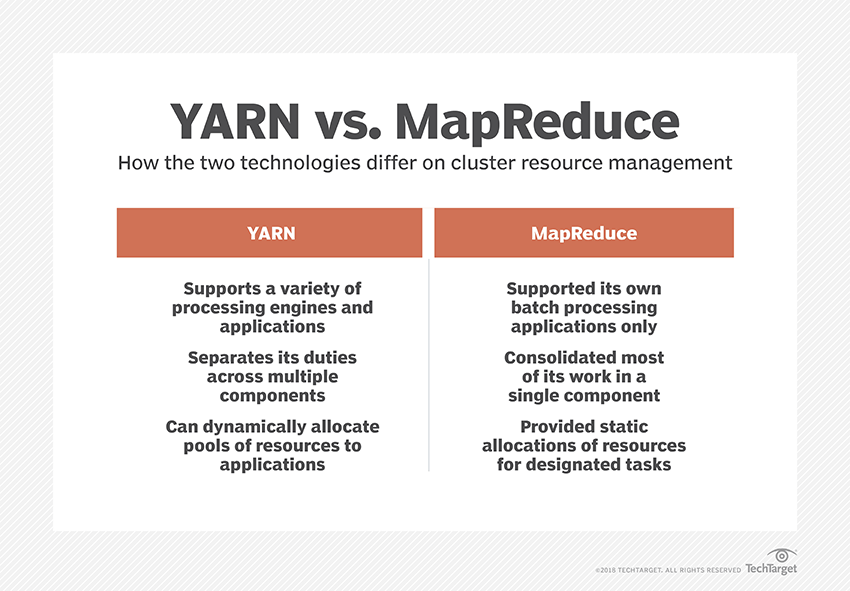
* This daemon process resides on the Master Node (runs along with ResourceManager daemon )
* Responsible for,
  + Helping Scheduler daemon to keeps track of running application by coordination
  + Accepting job submissions from client
  + Negotiating first container for executing application specific task with suitable ApplicationMaster on slave node

## 2) NodeManager

* This daemon process resides on the slave nodes (runs along with DataNode daemon)
* Responsible for,
  + Managing and executing containers
  + Monitoring resource usage (i.e. usage of memory, cpu, network etc..) and reporting it back to ResourceManager daemon
  + Periodically sending heart-bits to ResourceManager for its health status update

## 3) ApplicationMaster

* This daemon process runs on the slave node (along with the NodeManager daemon)
* It is per application specific library works with NodeManager to execute the task
* The instance of this daemon is per application, which means in case of multiple jobs submitted on cluster, it may have more than one instances of ApplicationMaster on slave nodes
* Responsible for,
  + Negotiating suitable resource containers on slave node from ResourceManager
  + Working with one or multiple NodeManagers to monitor task execution on slave nodes



**NEEDS OF YARN**

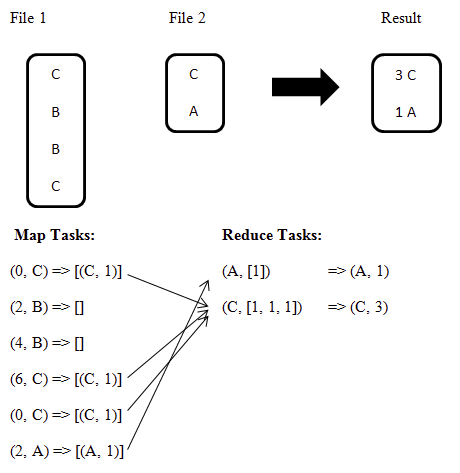
* YARN has also opened up new uses for [Apache HBase](https://searchdatamanagement.techtarget.com/definition/Apache-HBase), a companion database to HDFS, and for Apache Hive
* YARN offers scalability, resource utilization, high availability and performance improvements over MapReduce.
* Using Apache Hadoop YARN to separate HDFS from MapReduce made the Hadoop environment more suitable for real-time processing uses and other applications that can't wait for batch jobs to finish.
* [YARN](https://www.techopedia.com/definition/30154/hadoop-yarn) is a core component of Hadoop 2.0. It basically manages the resources in a [clustered](https://www.techopedia.com/definition/17/clustering-databases) environment.
* The YARN interacts with the compute resources (on behalf of the applications) and assigns resources to each application based on different filtering criteria.

# Applications of Map Reduce

## Distributed GREP

Distributed grep is used to search for a given pattern in a large number of files. For example, a web administrator can use distributed grep to search web server logs in order to find the top requested pages that match a given pattern [[1]](http://mapreduce-specifics.wikispaces.asu.edu/Applications+and+Limitations+of+MapReduce#cite_note-1) .  
  
Map function would take input as (input file, line) and generate one of the following outputs

* An empty list [], if there is no match found
* A key value pair [(line, 1)] if a match is found



The reduce function would take input as (line, [1, 1, 1, ....]) and generate output as (line, n) where 'n' is the number of 1's in the list.

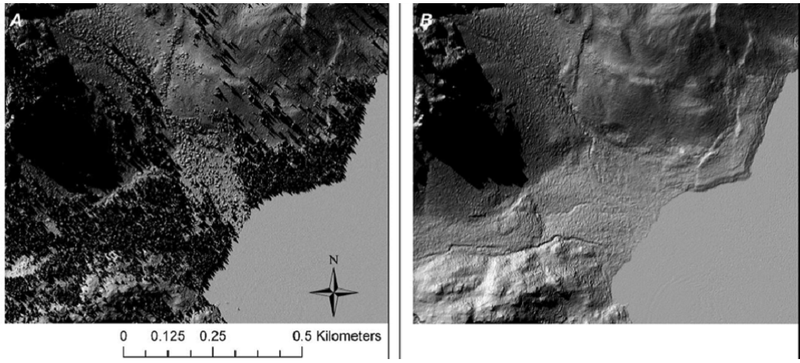
## Geospatial Query Processing

With the technological advancements in location-based services, there is a huge surge in the amount of geospatial data. Geospatial queries[[2]](http://mapreduce-specifics.wikispaces.asu.edu/Applications+and+Limitations+of+MapReduce#cite_note-2) (nearest neighbor [[3]](http://mapreduce-specifics.wikispaces.asu.edu/Applications+and+Limitations+of+MapReduce#cite_note-3) queries and reverse nearest neighbor queries) consume lot of computational resources and it is observed that their processing in intrinsically parallelizable.[[4]](http://mapreduce-specifics.wikispaces.asu.edu/Applications+and+Limitations+of+MapReduce#cite_note-4)  
  
Google Maps uses MapReduce to solve problems like

* Given an intersection, find all roads connecting to it
* Rendering of the tiles in the map
* Finding the nearest feature to a given address or current location

## Map Reduce for Gridding LIDAR data

Digital Elevation Models are digital or 3D representation of the landscape, where each (X, Y) position is represented by a single elevation value. DEMs are also referred by the name Digital Terrain Model (DTM) or Digital Surface Model (DSM). A DEM can be represented as a raster (a grid of squares) or as a triangular irregular network (TIN), and can be generated from remotely sensed (using satellites) or directly surveyed land elevation information. DEMs are used for a range of scientific and engineering applications, including hydrologic modeling, terrain analysis, and infrastructure design. One of the fundamental processing tasks in the OpenTopography system is generation of DEMs from very dense (multiple measurements per square meter) LIDAR[[5]](http://mapreduce-specifics.wikispaces.asu.edu/Applications+and+Limitations+of+MapReduce" \l "cite_note-5) (Light detection and Ranging) topography data.



[[6]](http://mapreduce-specifics.wikispaces.asu.edu/Applications+and+Limitations+of+MapReduce#cite_note-6)  
  
  
The local gridding algorithm utilizes the elevation information from LIDAR measurements contained within a circular search area to compute the elevation of each grid cell. This implementation is data parallel and hence an ideal use case for MapReduce.

|  |
| --- |
|  |
| MapReduce implementation of local gridding algorithm in Hadoop |

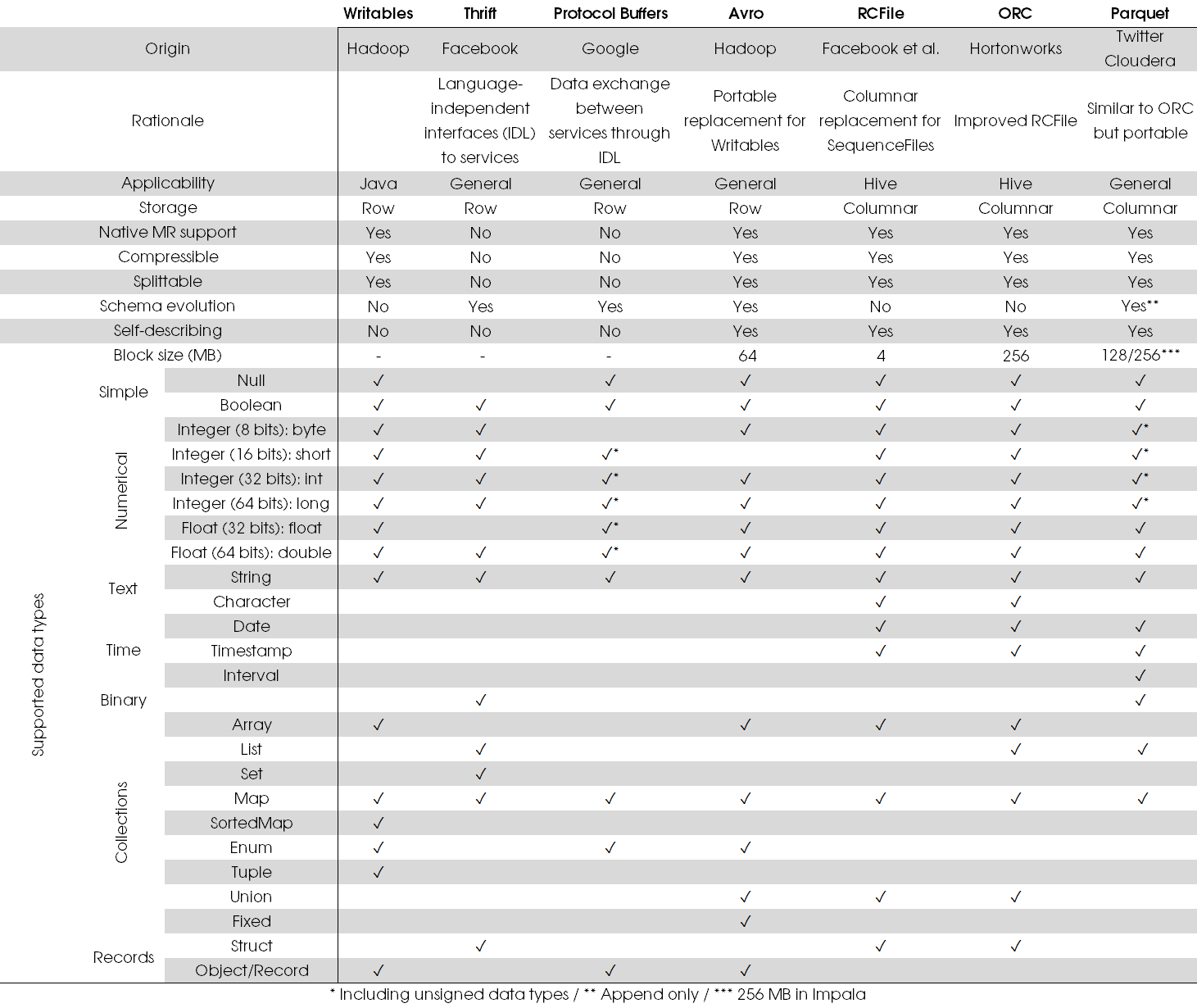
In the Map phase, input points are assigned to corresponding grid cells (local bins), and in the Reduce phase the corresponding elevations for each grid cell are computed from the local bins. The reduced outputs are merged, sorted and the DEM is generated in the ArcASCII grid format (also called as ESRI ASCII grid format [[7]](http://mapreduce-specifics.wikispaces.asu.edu/Applications+and+Limitations+of+MapReduce#cite_note-7) ).

## Other Applications

Examples[[8]](http://mapreduce-specifics.wikispaces.asu.edu/Applications+and+Limitations+of+MapReduce" \l "cite_note-8) of other applications where MapReduce is used.

1. Count of URL Access Frequency - The map function processes logs of web page requests and outputs <URL, 1>. The reduce function adds together all values for the same URL and emits a <URL, total count> pair.
2. Reverse Web-Link Graph - The map function outputs <target, source> pairs for each link to a target URL found in a page named "source". The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: <target, list(source)>.
3. Term-Vector per Host - A term vector summarizes the most important words that occur in a document or a set of documents as a list of <word, frequency> pairs. The map function emits a <hostname, term vector> pair for each input document (where the hostname is extracted from the URL of the document). The reduce function is passed all per-document term vectors for a given host. It adds these term vectors together, throwing away infrequent terms, and then emits a final <hostname, term vector> pair.
4. Inverted Index - The map function parses each document, and emits a sequence of <word, document ID> pairs. The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a <word, list(document ID)> pair. The set of all output pairs forms a simple inverted index. It is easy to augment this computation to keep track of word positions.

## Serialization Formats in Hadoop

[](https://databaseline.bitbucket.io/images/2015-12-07-hadoop-serialization-formats.png)

**Data Serialization Comparison: JSON, YAML, BSON, MessagePack**

**serialization is:**

The process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment.

Let’s say you want to collect certain data about a group of people — name, last name, nickname, date of birth, instruments they play. You could easily set a spreadsheet, define some columns, and make every row an entry. You could go just a little further, define that the date of birth column must be a number, and that the instruments columns could be a list of options. It’d look like this:

| name | last name | dob | nickname | instruments |
| --- | --- | --- | --- | --- |
| William | Bailey | 1962 | Axl Rose | vocals, piano |
| Saul | Hudson | 1965 | Slash | guitar |

In short: serializing data is finding some sort of universal format that can be easily shared across different applications.

**The Formats**

**1)JSON**

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It’s easy for humans to read and write; it’s easy for machines to parse and generate.

[JSON](http://www.json.org/) is the most widespread format for data serialization

**it has the following features:**

* (Mostly) [human readable](https://en.wikipedia.org/wiki/Human-readable_medium) code: even if the code has been obscured or [minified](https://en.wikipedia.org/wiki/Minification_(programming)), you can always indent it with tools such as [JSONLint](http://pro.jsonlint.com/) and make it readable again.
* Very simple and straightforward specification: a summary of the whole spec fits on a single page ([as displayed on the JSON site](http://www.json.org/)).
* Widespread support: not only does every programming language or IDE come with JSON support, but also many web services APIs offer JSON as a means of data interchange.
* As a subset of JavaScript, it supports the JavaScript data types: string,,number,,object,,array etc

This is how our previous spreadsheet would look, after being serialized in JSON:

[Report Advertisement](https://www.sitepoint.com/data-serialization-comparison-json-yaml-bson-messagepack/#report-ad)

[

{

"name": "William",

"last name": "Bailey",

"dob": 1962,

"nickname": "Axl Rose",

"instruments": [

"vocals",

"piano"

]

},

{

"name": "Saul",

"last name": "Hudson",

"dob": 1965,

"nickname": "Slash",

"instruments": [

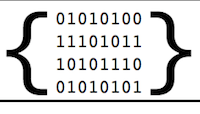
"guitar"

]

}

]

**2)BSON**

[](http://bsonspec.org/)

* BSON, short for Bin­ary JSON, is a bin­ary-en­coded seri­al­iz­a­tion of JSON-like doc­u­ments. … It also con­tains ex­ten­sions that al­low rep­res­ent­a­tion of data types that are not part of the JSON spec.
* JSON is a plain text format, and while binary data can be encoded in text, this has certain limitations and can make JSON files very big. [BSON](http://bsonspec.org/) comes in to deal with these problems.

**It has the following features:**

* convenient storage of binary information: better suitable for exchanging images and attachments designed for fast in-memory manipulation simple specification: like JSON, BSON has [a very short and simple spec](http://bsonspec.org/spec.html).

**3)MessagePack**

It’s like JSON. But fast and small.

[MessagePack](http://msgpack.org/) (also msgpack) is another binary format for serialization. Not as well known as BSON, but it’s worth having a look at.

**Among its features:**

* designed for efficient transmission over the wire
* better JSON-compatibility than BSON
* smaller than BSON: is has a smaller [overhead](https://en.wikipedia.org/wiki/Overhead_(computing)) than BSON, and can serialize smaller objects most of the time
* type checking: it supports [static-typing](https://en.wikipedia.org/wiki/Type_system#Static_type_checking)
* streaming API: support for streaming deserializers, which is useful for network communication.

**4)YAML**

* YAML is a human friendly data serialization standard for all programming languages.

**Back to plaintext formats,** [**YAML**](http://yaml.org/) **is an alternative to JSON**:

* (truly) human readable code: YAML is so readable that even [its front-page content](http://yaml.org/) is displayed in YAML to make this point
* compact code: whitespace indentation is used to denote structure, no need for quotes nor brackets
* syntax for relational data: to allow internal references with anchors ( &) and aliases (\*)
* especially suited for viewing/editing of data structures: such as configuration files, dumping during debugging, and document headers

a rich set of [language independent types](http://yaml.org/type/):

collections:

unordered set of key (!!map)

ordered sequence of key (!!omap)

ordered sequence of key (!!pairs)

unordered set of non-equal values (!!set)

sequence of arbitrary values (!!seq)

scalar types:

null values (~, null)

decimals (1234), hexadecimal (0x4D2) and octal (02333) integers

fixed (1\_230.15) and exponential (12.3015e+02) floats

This is how our little spreadsheet looks when serialized in YAML:

- name: William

last name: Bailey

dob: 1962

nickname: Axl Rose

instruments:

- vocals

- piano

- name: Saul

last name: Hudson

dob: 1965

nickname: Slash

instruments: