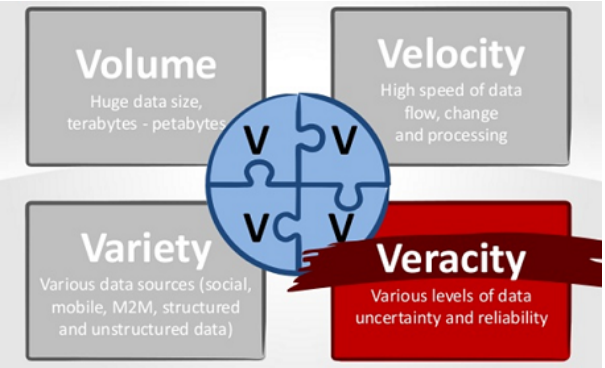
## Data Science in the Real World

- Web Search – How do search engines like Google or Bing rank search results?
- Shopping – How does Amazon forecast how many items it needs to store in its warehouses?
- Astronomy – How can we process terabytes/day of telescope data?
- Physics – How do you write software to search for new physics particles?
- Medicine – How can researchers use genomics to make personalized medical recommendations?
- Sports – How can we visualize and understand massive amounts of game sensor data?
- Social media – How does Facebook recognize people in images?

All of these applications use Data Science!

## 4 V's of Big Data




**Volume-based value**: The more comprehensive your integrated view of the customer and the more historical data you have on them, the more insight you can extract from it. In turn, you are making better decisions when it comes to acquiring, retaining, growing and managing those customer relationships.

**Velocity-based value**: The more rapidly you can process information into your data and analytics platform, the more flexibility you get to find answers to your questions via queries, reports, dashboards, etc. A rapid data ingestion and rapid analysis capability provides you with the timely and correct decision achieve your customer relationship management objectives.

**Variety-based value**: The more varied customer data you have – from the Customer relationship management (CRM) system, social media, call-center logs, etc. – the more multifaceted view you develop about your customers, thus enabling you to develop customer journey maps and personalization to engage more with customers.

**Veracity-based value**: Amassing a lot of data does not mean the data becomes clean and accurate. Data on customers must remain consolidated, cleansed, consistent, and current to make the right decisions.

## Key Ideas (Principles)

- Scale "out", not "up"
- Assume failures are common and find remedy
- Move processing to the data
- Process data sequentially, avoid random access
- Seamless scalability

### 1. Scale out, not up

- For data-intensive workloads, a large number of commodity low-end servers (i.e., the scaling out approach) is preferred over a small number of high-end servers (i.e., the scaling up approach).
- The latter approach of purchasing symmetric multi-processing (SMP) machines with a large number of processor sockets (dozens, even hundreds) and a large amount of shared memory (hundreds or even thousands of gigabytes) is not cost effective, since the costs of such machines do not scale linearly (i.e., a machine with twice as many processors is often significantly more than twice as expensive).

### 2. Assume failures are common

At warehouse scale, failures are not only inevitable, but commonplace.

**Problem:**
Assume that a 10000 server cluster is built from reliable machines with a mean-time between failures (MTBF) of 1000 days.

(a) What is the failure rate?

(b) If MTBF of a machine is 10000 days, what is the failure rate?

Let us suppose that a cluster is built from reliable machines with a mean-time between failures (MTBF) of 1000 days (about three years). Even with these reliable servers, a 10,000-server cluster would still experience roughly 10 failures a day.

For the sake of argument, let us suppose that a MTBF of 10,000 days (about thirty years) were achievable at realistic costs (which is unlikely). Even then, a 10,000-server cluster would still experience one failure daily.

### 3. Move processing to the data

In traditional high-performance computing (HPC) applications (e.g., for climate or nuclear simulations), it is commonplace for a supercomputer to have processing nodes and storage nodes linked together by a high-capacity interconnect.

Many Big Data workloads are not very processor-demanding, which means that the separation of compute and storage creates a bottleneck in the network.

As an alternative to moving data around, it is more efficient to move the processing around.

In such a setup, we can take advantage of data locality by running code on the processor directly attached to the block of data we need.

### 4. Process data sequentially and avoid random access

- Data-intensive processing means that the relevant datasets are too large to fit in memory and must be held on disk.
- Seek times for random disk access are fundamentally limited by the mechanical nature of the devices: read heads can only move so fast and platters can only spin so rapidly. As a result, it is desirable to avoid random data access, and instead organize computations so that data is processed sequentially.

### 5. Hide system-level details from the application developer

Writing code is difficult because the programmer must simultaneously keep track of many details in short term memory.

**Short-term memory** (or "primary" or "active memory") is the capacity for holding a small amount of information in mind in an active, readily available state for a short period of time. The duration of short-term memory (when rehearsal or active maintenance is prevented) is believed to be in the order of seconds. A commonly cited capacity is 7 ± 2 elements. In contrast, long-term memory can hold an indefinite amount of information.

### 6. Seamless Scalability

- For data-intensive processing, it goes without saying that scalable algorithms are highly desirable.
- Adding load to the system should result in a graceful decline in performance of individual jobs
    - Not failure of the system
- Increasing resources should support a proportional increase in load capacity

## File System on a Single Disk

File system is a structured data representation and a set of metadata that describe the stored data.

- Hard drives are divided into sectors of about 512 bytes each. Sectors in turn are grouped into clusters. Clusters have a defined size of 512 bytes to 64 KBs, so they usually contain multiple sectors.
- A cluster is called as a filesystem block which represents a continuous block of space on the disk and it is the minimum amount of data that can be read or written.
- Depending on the file system block (typically 4KBs) a single file can be stored in one or across hundreds or thousands of clusters.
- This is generally transparent to the filesystem user who is simply reading or writing a file of whatever length.

### Why is a block in HDFS so large compared to disk block size?

Large block size in HDFS minimizes the cost of seeks (seek time - the time taken for a disk drive to locate the area on the disk where the data to be read is stored).

If the block is large enough, the time it takes to transfer the data from the disk can be significantly longer than the time to seek to the start of the block.

Thus, transferring a large file made of multiple blocks operates at the disk transfer rate.

E.g. if the seek time is around 10 ms and the transfer rate is 100 MB/s, then to make the seek time 1% of the transfer time, we need to make the block size around 100 MB.

The default is actually 128 MB, although many HDFS installations use larger block sizes. This number will continue to be revised upward as transfer speeds grow with new generations of disk drives.

This argument shouldn’t be taken too far, however. MapReduce tasks operate on one block at a time. So if data is not distributed across many data nodes, we are not taking advantage of the parallelism possible to its fullest extent. So your jobs will run slower than they could otherwise!

## Advantages of Blocks

Block abstraction of HDFS has the following advantages:

- A file can be larger than any single disk in the network.
  - In fact, it would be possible, if unusual, to store a single file on an HDFS cluster whose blocks filled all the disks in the cluster.
- Making the unit of abstraction a block rather than a file simplifies the storage subsystem
  - It is easy to calculate how many blocks can be stored on a given disk and it eliminates metadata concerns (file metadata such as permissions information does not need to be stored with the blocks so another system can handle metadata separately).
- DataNode has no knowledge about what it is storing or processing.
- Blocks fit well with replication for providing fault tolerance and availability.

## Rack in Hadoop

A **rack** is a collection of 30 or 40 nodes that are physically stored close together and are all connected to the same network switch. Network bandwidth between any two nodes in **rack** is greater than bandwidth between two nodes on different racks. A **Hadoop** Cluster is a collection of racks.

## NameNode

The NameNode in Hadoop is the node where Hadoop stores all the location information of the files in HDFS. In other words, it holds the metadata for HDFS. Whenever a file is placed in the cluster a corresponding entry of its location is maintained by the NameNode.