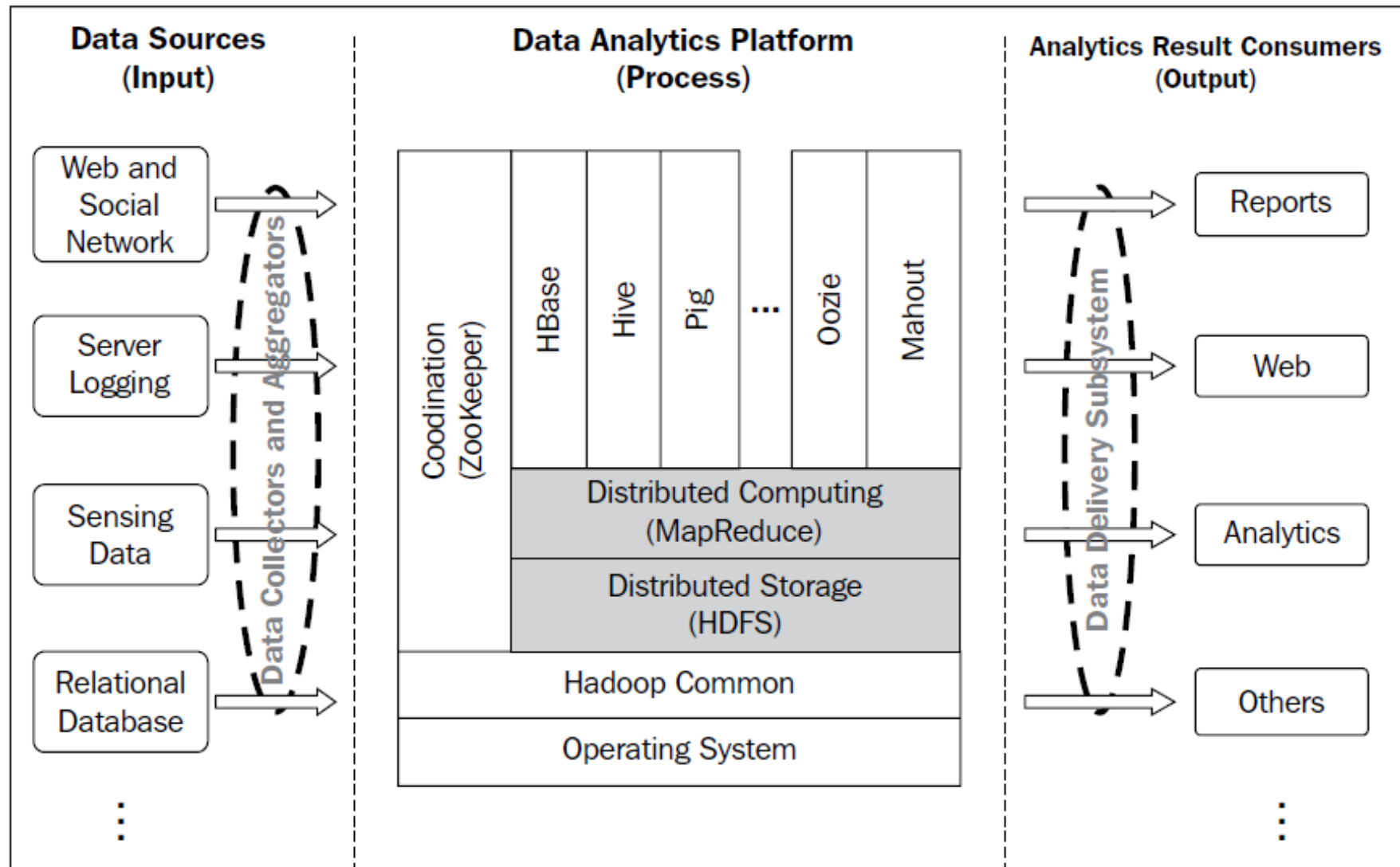


The Architecture of a Hadoop-based Big Data System



Hadoop is more than MapReduce

- Although Hadoop borrows its idea from Google's MapReduce, it is more than MapReduce.
- A typical Hadoop-based Big Data platform includes
 - ▣ the Hadoop Distributed File System (HDFS)
 - ▣ the parallel computing framework (MapReduce)
 - ▣ common utilities
 - ▣ a column-oriented data storage table (HBase)
 - ▣ high-level data management systems (Pig and Hive)
 - ▣ a Big Data analytics library (Mahout)
 - ▣ a distributed coordination system (ZooKeeper)
 - ▣ a workflow management module (Oozie)
 - ▣ data transfer modules such as Sqoop
 - ▣ data aggregation modules such as Flume
 - ▣ and data serialization modules such as Avro.

HDFS is the default filesystem of Hadoop

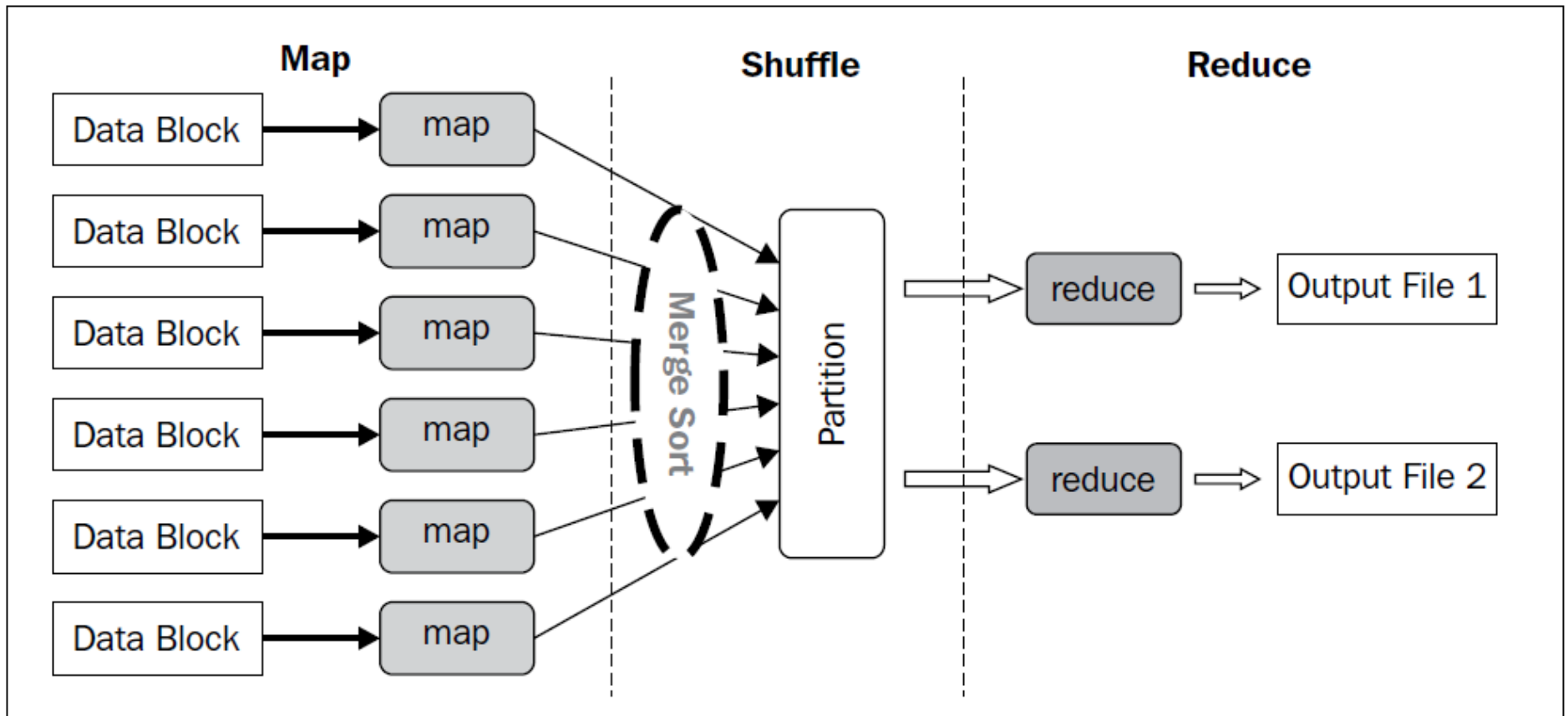


- It was designed as a distributed filesystem that provides high-throughput access to application data.
- Data on HDFS is stored as data blocks.
- The data blocks are replicated on several computing nodes and their checksums are computed.
- In case of a checksum error or system failure, erroneous or lost data blocks can be recovered from backup blocks located on other nodes.

MapReduce

- ❑ MapReduce provides a programming model that transforms complex computations into computations over a set of key-value pairs.
- ❑ It coordinates the processing of tasks on a cluster of nodes by scheduling jobs, monitoring activity, and re-executing failed tasks.
- ❑ In a typical MapReduce job, multiple map tasks on slave nodes are executed in parallel, generating results buffered on local machines.
- ❑ Once some or all of the map tasks have finished, the shuffle process begins, which aggregates the map task outputs by sorting and combining key-value pairs based on keys.
- ❑ Then, the shuffled data partitions are copied to reducer machine(s), most commonly, over the network.
- ❑ Then, reduce tasks will run on the shuffled data and generate final (or intermediate, if multiple consecutive MapReduce jobs are pipelined) results.
- ❑ When a job finishes, final results will reside in multiple files, depending on the number of reducers used in the job.

The anatomy of the job flow can be seen in the following chart



HDFS has two types of nodes



- HDFS has two types of nodes:
 - ▣ NameNode
 - ▣ DataNode
- A NameNode keeps track of the filesystem metadata (the locations of data blocks).
- For efficiency reasons, the metadata is kept in the main memory of a master machine.
- A DataNode holds physical data blocks and communicates with clients for data reading and writing.
- In addition, it periodically reports a list of its hosting blocks to the NameNode in the cluster for verification and validation purposes.

The MapReduce framework has two types of nodes

- The MapReduce framework has two types of nodes:
 - ▣ master node
 - ▣ slave node
- JobTracker is the daemon on a master node, and TaskTracker is the daemon on a slave node.
- The master node is the manager node of MapReduce jobs.
- It splits a job into smaller tasks, which will be assigned by the JobTracker to TaskTrackers on slave nodes to run.
- When a slave node receives a task, its TaskTracker will fork a Java process to run it.
- Meanwhile, the TaskTracker is also responsible for tracking and reporting the progress of individual tasks.

Hadoop Distributed Filesystem

- HDFS was built to support high throughput, streaming reads and writes of extremely large files.
- Traditional large storage area networks (SANs) and network attached storage (NAS) offer centralized, low-latency access to either a block device or a filesystem on the order of terabytes in size.
- These systems are fantastic as the backing store for relational databases, content delivery systems, and similar types of data storage needs because they can support full-featured POSIX semantics, scale to meet the size requirements of these systems, and offer low-latency access to data.
- Imagine hundreds or thousands of machines all waking up at the same time and pulling hundreds of terabytes of data from a centralized storage system at once.
- This is where traditional storage doesn't necessarily scale.

HDFS Goals

- By creating a system composed of independent machines, each with its own I/O subsystem, disks, RAM, network interfaces, and CPUs, and relaxing (and sometimes removing) some of the POSIX requirements, it is possible to build a system optimized, in both performance and cost, for the specific type of workload we're interested in.
- There are a number of specific goals for HDFS:
 - ▣ Store millions of large files, each greater than tens of gigabytes, and filesystem sizes reaching tens of petabytes.
 - ▣ Use a scale-out model based on inexpensive commodity servers with internal JBOD ("Just a bunch of disks") rather than RAID to achieve large-scale storage. Accomplish availability & high throughput thru application-level replication of data.
 - ▣ Optimize for large, streaming reads and writes rather than low-latency access to many small files. Batch performance is more important than interactive response times
 - ▣ Gracefully deal with component failures of machines and disks.
 - ▣ Support the functionality and scale requirements of MapReduce processing. See

HDFS, in many ways, follows traditional filesystem design

- Files are stored as opaque blocks and metadata exists that keeps track of the filename to block mapping, directory tree structure, permissions, and so forth.
- This is similar to common Linux filesystems such as ext3. So what makes HDFS different?
- Traditional filesystems are implemented as kernel modules and together with tools, can be mounted and made available to end users.
- HDFS is what's called a userspace filesystem.
- This is a fancy way of saying that the filesystem code runs outside the kernel as OS processes and by extension, is not registered with or exposed via the Linux VFS layer.
- While this is much simpler, more flexible, and arguably safer to implement, it means that you don't mount HDFS as you would ext3, for instance, and that it requires applications to be explicitly built for it.

HDFS is a distributed filesystem

- Distributed filesystems are used to overcome the limits of what an individual disk or machine is capable of supporting.
- Each machine in a cluster stores a subset of the data that makes up the complete filesystem with the idea being that, as we need to store more block data, we simply add more machines, each with multiple disks.
- Filesystem metadata is stored on a centralized server, acting as a directory of block data and providing a global picture of the filesystem's state.
- Another major difference between HDFS and other filesystems is its block size.
- It is common that general purpose filesystems use a 4 KB or 8 KB block size for data.
- Hadoop, on the other hand, uses the significantly larger block size of 64 MB by default.

Increasing the Block Size

- In fact, cluster adminis usually raise this to 128 MB, 256 MB, or even as high as 1 GB.
- Increasing the block size means data will be written in larger contiguous chunks on disk, which in turn means data can be written and read in larger sequential operations.
- This minimizes drive seek operations—one of the slowest operations a mechanical disk can perform—and results in better performance when doing large streaming I/O operations.


HDFS Replication

- Rather than rely on specialized storage subsystem data protection, HDFS replicates each block to multiple machines in the cluster.
- By default, each block in a file is replicated three times.
- Because files in HDFS are write once, once a replica is written, it is not possible for it to change.
- This avoids the need for complex reasoning about the consistency between replicas and as a result, applications can read any of the available replicas when accessing a file
- Having multiple replicas means multiple machine failures are easily tolerated, but there are also more opportunities to read data from a machine closest to an application on the network.
- HDFS actively tracks and manages the number of available replicas of a block as well.
- Should the number of copies of a block drop below the configured replication factor, the filesystem automatically makes a new copy from one of the remaining replicas.

HDFS Blocks

- Applications, of course, don't want to worry about blocks, metadata, disks, sectors, and other low-level details.
- Instead, developers want to perform I/O operations using higher level abstractions such as files and streams.
- HDFS presents the filesystem to developers as a high-level, POSIX-like API with familiar operations and concepts.

Daemons



Daemon	# per cluster	Purpose
Namenode	1	Stores filesystem metadata, stores file to block map, and provides a global picture of the filesystem
Secondary namenode	1	Performs internal namenode transaction log checkpointing
Datanode	Many	Stores block data (file contents)

Blocks

- Blocks are nothing more than chunks of a file, binary blobs of data.
- In HDFS, the daemon responsible for storing and retrieving block data is called the datanode (DN).
- The datanode has direct local access to one or more disks—commonly called data disks—in a server on which it's permitted to store block data.
- In production systems, these disks are usually reserved exclusively for Hadoop.
- Storage can be added to a cluster by adding more datanodes with additional disk capacity, or even adding disks to existing datanodes.

HDFS Doesn't Require RAID Storage

- One of the most striking aspects of HDFS is that it is designed in such a way that it doesn't require RAID storage for its block data.
- This keeps with the commodity hardware design goal and reduces cost as clusters grow in size.
- Rather than rely on a RAID controller for data safety, block data is simply written to multiple machines.
- This fulfills the safety concern at the cost of raw storage consumed; however, there's a performance aspect to this as well.
- Having multiple copies of each block on separate machines means that not only are we protected against data loss if a machine disappears, but during processing, any copy of this data can be used.

NameNode

- While datanodes are responsible for storing block data, the namenode (NN) is the daemon that stores the filesystem metadata and maintains a complete picture of the filesystem.
- Clients connect to the namenode to perform filesystem operations; although, as we'll see later, block data is streamed to and from datanodes directly, so bandwidth is not limited by a single node.
- Datanodes regularly report their status to the namenode in a heartbeat.
- This means that, at any given time, the namenode has a complete view of all datanodes in the cluster, their current health, and what blocks they have available

Block Report

- When a datanode initially starts up, as well as every hour thereafter, it sends what's called a block report to the namenode.
- The block report is simply a list of all blocks the datanode currently has on its disks and allows the namenode to keep track of any changes.
- This is also necessary because, while the file to block mapping on the namenode is stored on disk, the locations of the blocks are not written to disk.
- This may seem counterintuitive at first, but it means a change in IP address or hostname of any of the datanodes does not impact the underlying storage of the filesystem metadata.

DataNode Failures

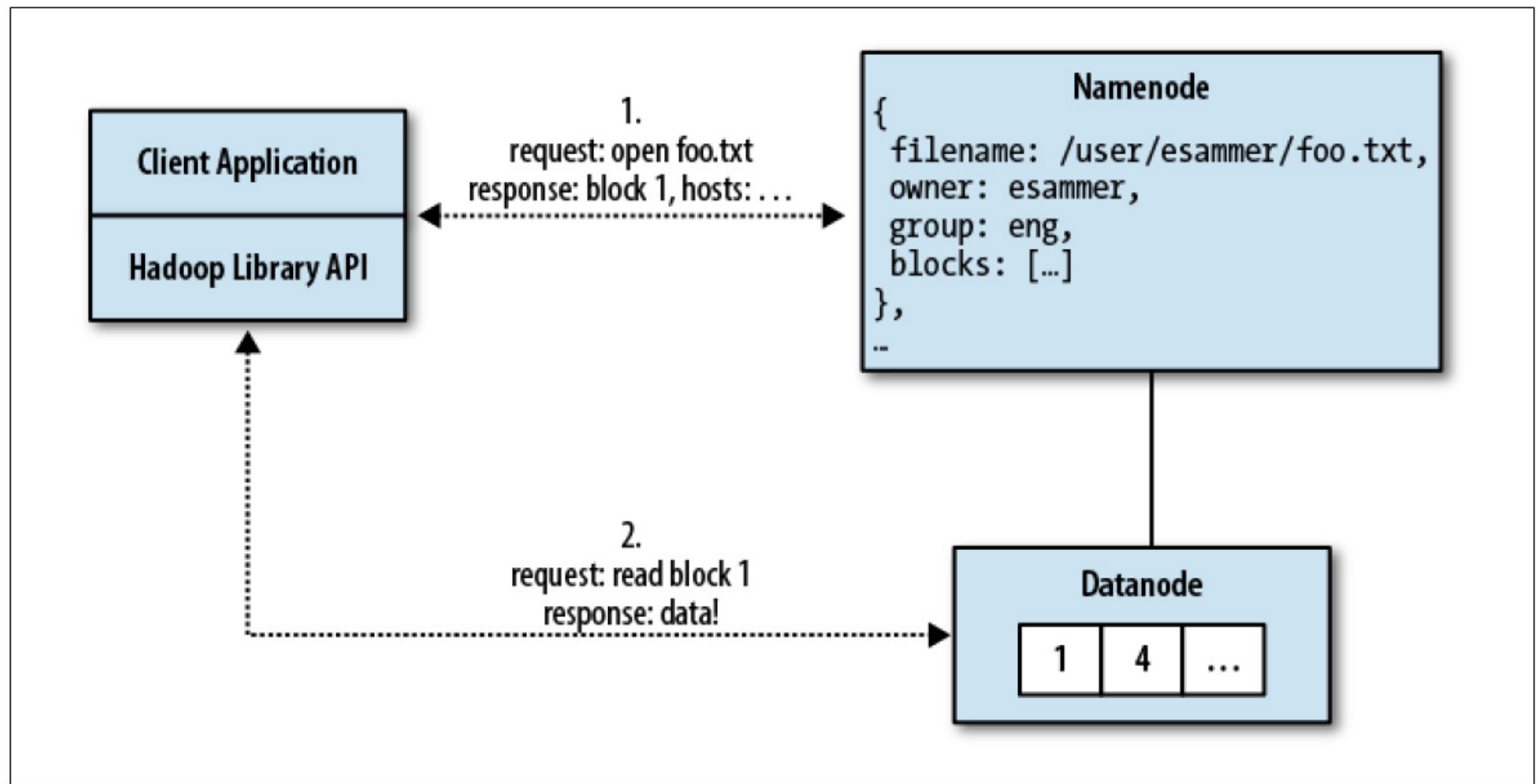


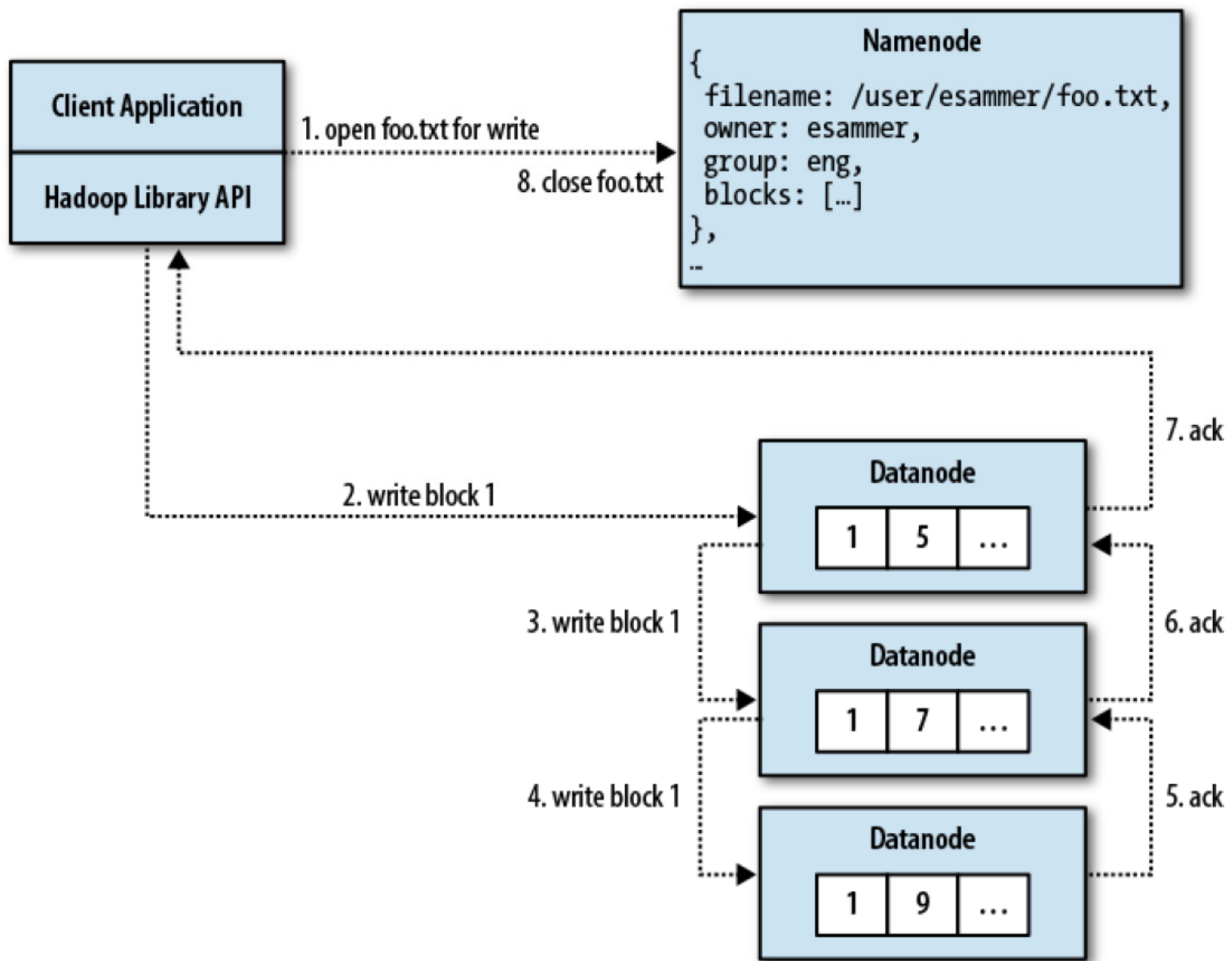
- Another nice side effect of this is that, should a datanode experience failure of a motherboard, administrators can simply remove its hard drives, place them into a new chassis, and start up the new machine.
- As far as the namenode is concerned, the blocks have simply moved to a new datanode.
- The downside is that, when initially starting a cluster (or restarting it, for that matter), the namenode must wait to receive block reports from all datanodes to know all blocks are present.

Secondary Namenode

- Finally, the third HDFS process is called the secondary namenode and performs some internal housekeeping for the namenode.
- Despite its name, the secondary namenode is not a backup for the namenode and performs a completely different function.

With the block IDs and datanode hostnames, the client can now contact the most appropriate datanode directly and read the block data it needs. This process repeats until all blocks in the file have been read or the client closes the file stream.





Hadoop has the following drawbacks as a Big Data platform:

- As an open source software, Hadoop is difficult to configure and manage, mainly due to the instability of the software and lack of properly maintained documentation and technical support
- Hadoop is not an optimal choice for real-time, responsive Big Data applications
- Hadoop is not a good fit for large graph datasets
- Hadoop is not a good choice for data that is not categorized as Big Data;
 - ▣ for example, data that has the following properties:
 - small datasets
 - datasets with processing that requires transaction and synchronization.