Chapter 8 – Storage Systems

Contents

- Data storage in the age of cloud computing
- Storage and data models
- Database management systems
- Unix file system (IFS)
- Network file system (NFS)
- General parallel file system (GPFS)
- Google file system (GFS)
- Apache Hadoop
- Locks; Chubby a locking service
- Online transaction processing
- NoSQL databases
- Bigtable
- Megastore
- Storage Reliability at scale; DynamoDB
- Disk locality versus data locality
- Database provenance

Data Storage in the Age of Cloud Computing

- The volume of data generated by human activities is growing about 40% per year; 90% of the data in the world today has been gathered in the last two years.
- The network-centric data storage model is particularly useful for mobile devices with limited power reserves and local storage, now able to save and to access large audio and video files stored on computer clouds. Billions of Internet-connected mobile, as well as stationary devices, access data stored on computer clouds.
- Big Data reflects the reality that many applications use data sets so large that local computers, or even small to medium scale data centers, do not have the capacity to store and process such data.
- The management of the large collection of storage systems poses significant challenges and requires novel approaches to system design.
    - Effective data replication and storage management strategies are critical to the computations performed on the cloud.

Major Challenges

- The storage system design philosophy has shifted from performance-at-any-cost to reliability-at-the-lowest-possible-cost.
- This design philosophy has important implications on software complexity.
- Maintaining consistency among multiple copies of data records increases the data management complexity and could negatively affect the storage system performance if data is frequently updated.
- Sophisticated strategies to reduce the access time and to support multimedia access are necessary to satisfy the timing requirements of data streaming and content delivery.
- Data replication allows concurrent access to data from multiple processors and decreases the chances of data loss.

Data Storage on a Cloud

- Storage and processing on the cloud are intimately tied to one another.
  - Most cloud applications process very large amounts of data.
  - Effective data replication and storage management strategies are critical to the computations performed on the cloud.
  - Strategies to reduce the access time and to support real-time multimedia access are necessary to satisfy the requirements of content delivery.
- An ever increasing number of cloud-based services collect detailed data about their services and information about the users of these services.
  - The service providers use the clouds to analyze the data.
- Humongous amounts of data - in 2013
  - The Internet video will generate over 18 EB/month.
  - Global mobile data traffic will reach 2 EB/month.

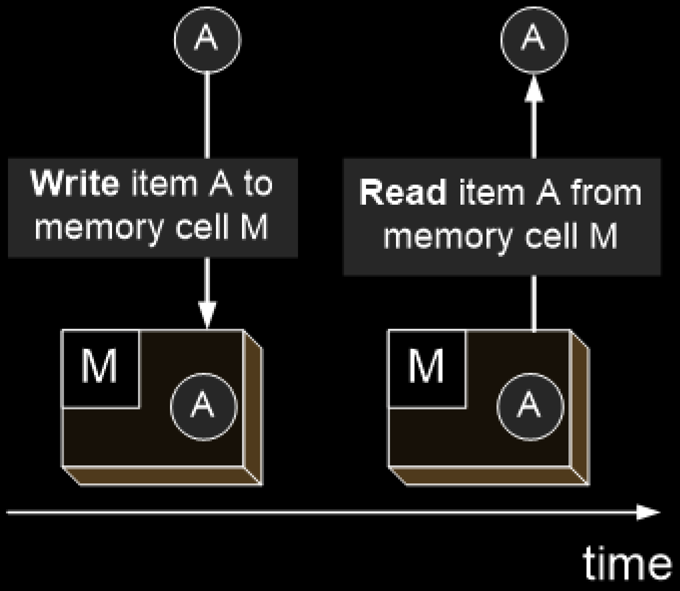
(1 EB = 10^{18} bytes, 1 PB = 10^{15} bytes, 1 TB = 10^{12} bytes, 1 GB = 10^{9} bytes)

Big Data

- Reflects the fact that many applications use data sets that cannot be stored and processed using local resources.
- Applications in genomics, structural biology, high energy physics, astronomy, meteorology, and the study of the environment carry out complex analysis of data sets often of the order of TBs (terabytes). Examples:
  - In 2010, the four main detectors at the Large Hadron Collider (LHC) produced 13 PB of data.
  - The Sloan Digital Sky Survey (SDSS) collects about 200 GB of data per night.
- Three-dimensional phenomena.
  - Increased volume of data.
  - Requires increased processing speed to process more data and produce more results.
  - Involves a diversity of data sources and data types.

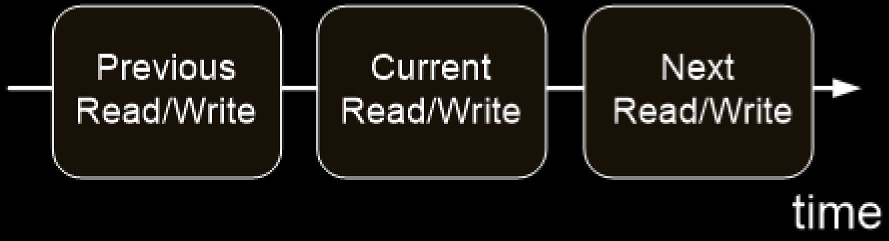
Storage and Data Models

- A storage model
  - Describes the layout of a data structure in a physical storage - a local disk, a removable media, or storage accessible via the network.
- A data model
  - Captures the most important logical aspects of a data structure in a database.
- Two abstract models of storage are used.
  - Cell storage
    Assumes that the storage consists of cells of the same size and that each object fits exactly in one cell. This model reflects the physical organization of several storage media; the primary memory of a computer is organized as an array of memory cells and a secondary storage device, e.g., a disk, is organized in sectors or blocks read and written as a unit.
  - Journal storage
    System that keeps track of the changes that will be made in a *journal* (usually a circular log in a dedicated area of the file system) before committing them to the main file system. In the event of a system crash or power failure, such file systems are quicker to bring back online and less likely to become corrupted.



Write item A to memory cell M

Read item A from memory cell M



Previous Read/Write

Current Read/Write

Next Read/Write

Read/Write coherence: the result of a **Read** of memory cell M should be the same as the most recent **Write** to that cell

Before-or-after atomicity: the result of every **Read** or **Write** is the same as if that **Read** or **Write** occurred either completely before or completely after any other **Read** or **Write**.

Read/write coherence and before-or-after atomicity are two highly desirable properties of any storage model and in particular of cell storage

Data Base Management System (DBMS)

- Database: a collection of logically-related records.
- Data Base Management System (DBMS): the software that controls the access to the database.
- Query language: a dedicated programming language used to develop database applications.
- Most cloud application do not interact directly with the file systems, but through a DBMS.
- Database models: reflect the limitations of the hardware available at the time and the requirements of the most popular applications of each period.
  - Navigational model of the 1960s.
  - Relational model of the 1970s.
  - Object-oriented model of the 1980s.
  - NoSQL model of the first decade of the 2000s.

Storage Requirements of Cloud Applications

- Most cloud applications are data-intensive and test the limitations of the existing infrastructure.
- Requirements:
  - Rapid application development and short-time to the market
  - Low latency
  - Scalability
  - High availability
  - Consistent view of the data
- These requirements cannot be satisfied simultaneously by existing database models; e.g., relational databases are easy to use for application development but do not scale well.
- The NoSQL model is useful when the structure of the data does not require a relational model and the amount of data is very large.
  - Does not support SQL as a query language.
  - May not guarantee the ACID (Atomicity, Consistency, Isolation, Durability) properties of traditional databases; it usually guarantees the *eventual consistency* for transactions limited to a single data item.

Logical and Physical Organization of a File

- File: a linear array of cells stored on a persistent storage device. Viewed by an application as a collection of logical records; the file is stored on a physical device as a set of physical records, or blocks, of size dictated by the physical media.
- File pointer: identifies a cell used as a starting point for a **read** or **write** operation.
- The logical organization of a file reflects the data model, the view of the data from the perspective of the application.
- The physical organization of a file reflects the storage model and describes the manner the file is stored on a given storage media.

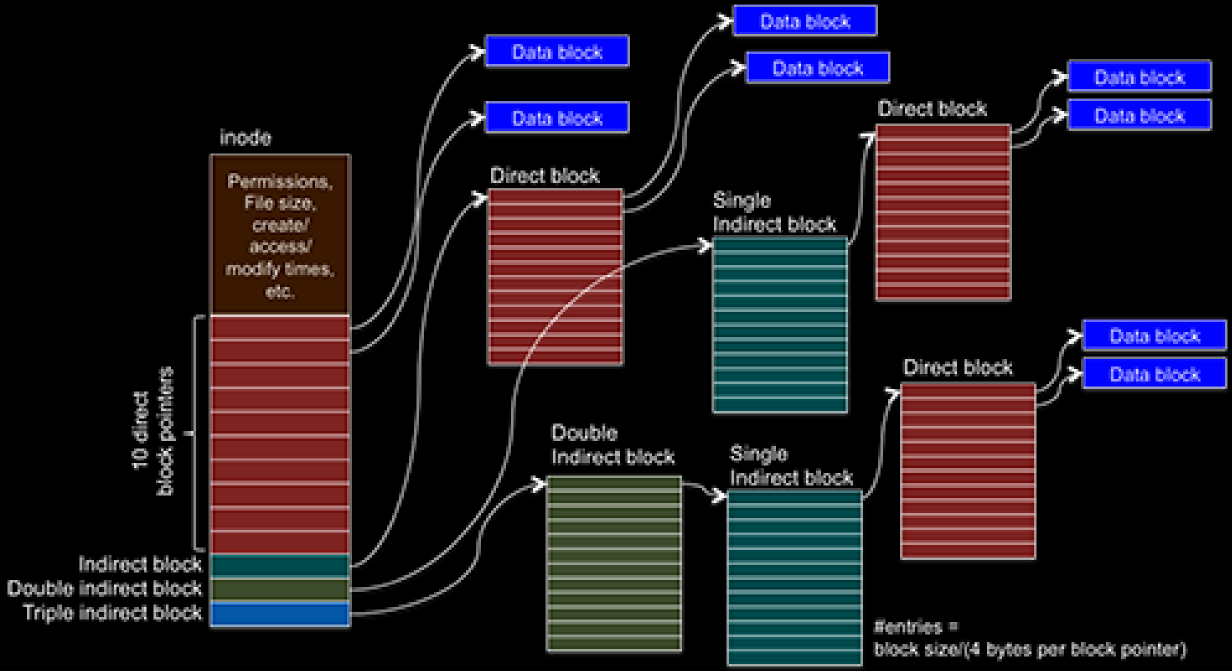
File Systems

- Collection of directories; each directory provides information about a set of files.
- Traditional – Unix File System.
- Distributed file systems
  - Network File Systems (NFS) – very popular, have been used for some time, but do not scale well and have reliability problems; an NFS server could be a single point of failure.
- Storage Area Networks (SAN) – allow cloud servers to deal with non-disruptive changes in the storage configuration. The storage in a SAN can be pooled and then allocated based on the needs of the servers. A SAN-based implementation of a file system can be expensive, as each node must have a Fiber Channel adapter to connect to the network.
- Parallel File Systems (PFS) – scalable, capable of distributing files across a large number of nodes, with a global naming space. Several I/O nodes serve data to all computational nodes; it includes also a metadata server which contains information about the data stored in the I/O nodes. The interconnection network of a PFS could be a SAN.

Unix File System (UFS)

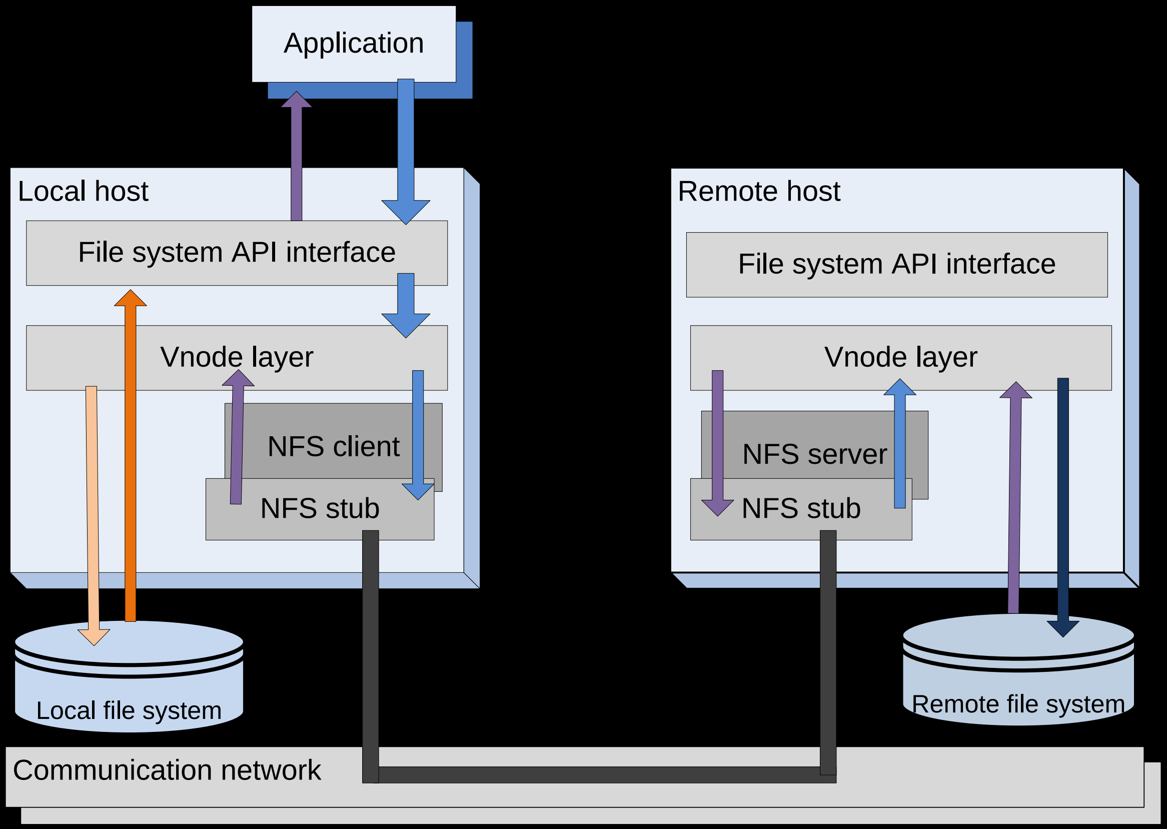
- The layered design provides flexibility.
  - The layered design allows UFS to separate the concerns for the physical file structure from the logical one.
  - The *vnode* layer allowed UFS to treat uniformly local and remote file access.
- The hierarchical design supports scalability reflected by the file naming convention. It allows grouping of files directories, supports multiple levels of directories, and collections of directories and files, the so-called file systems.
- The metadata supports a systematic design philosophy of the file system and device-independence.
  - Metadata includes: file owner, access rights, creation time, time of the last modification, file size, the structure of the file and the persistent storage device cells where data is stored.
  - The *inodes* contain information about individual files and directories. They are kept on persistent media together with the data.

Structure of UFS



Network File System (NFS)

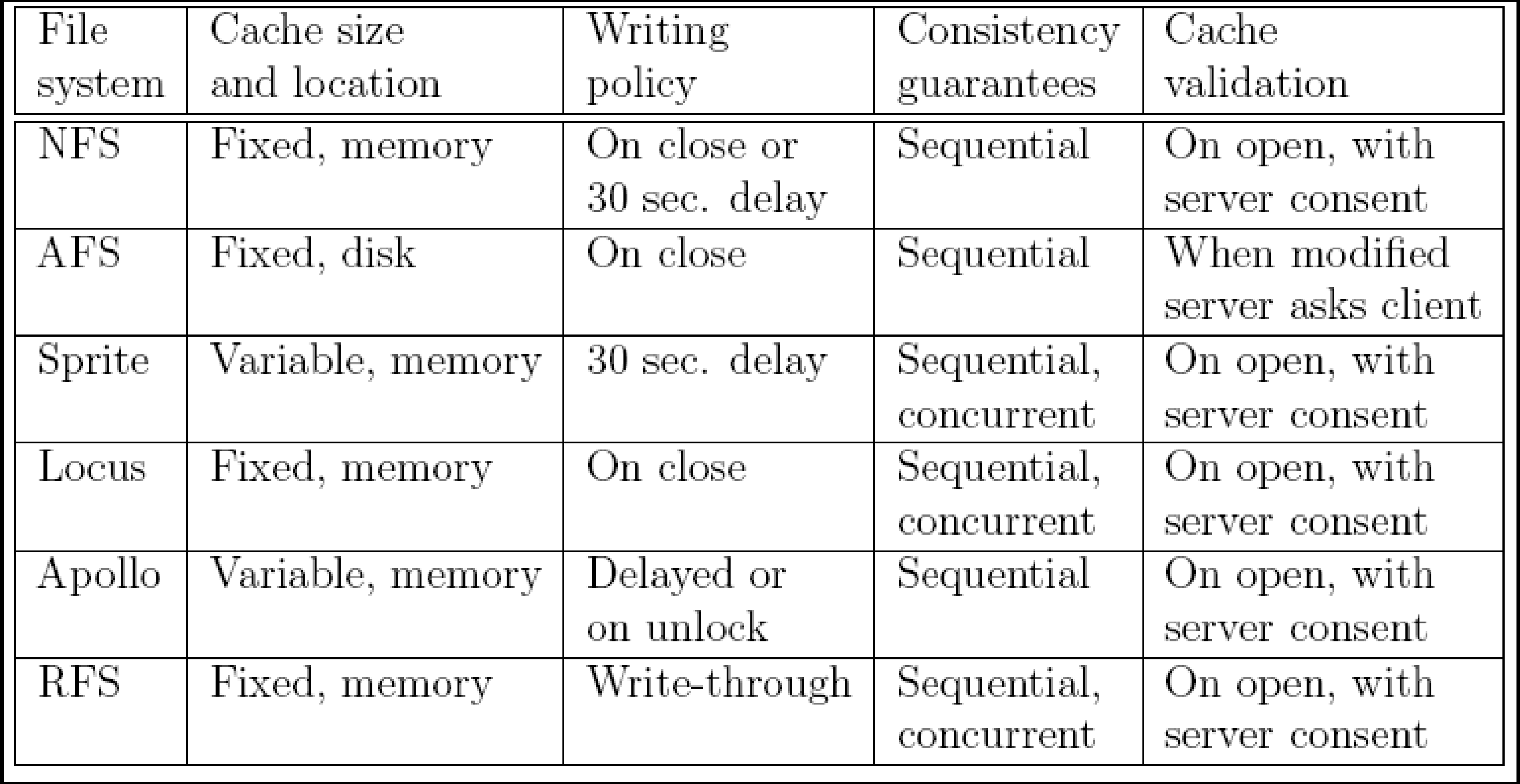
- Design objectives:
  - Provide the same semantics as a local Unix File System (UFS) to ensure compatibility with existing applications.
  - Facilitate easy integration into existing UFS.
  - Ensure that the system will be widely used; thus, support clients running on different operating systems.
  - Accept a modest performance degradation due to remote access over a network with a bandwidth of several Mbps.
- NFS is based on the client-server paradigm. The client runs on the local host while the server is at the site of the remote file system; they interact by means of Remote Procedure Calls (RPC).
- A remote file is uniquely identified by a file handle (fh) rather than a file descriptor.
  - The file handle is a 32-byte internal name - a combination of the file system identification, an inode number, and a generation number.



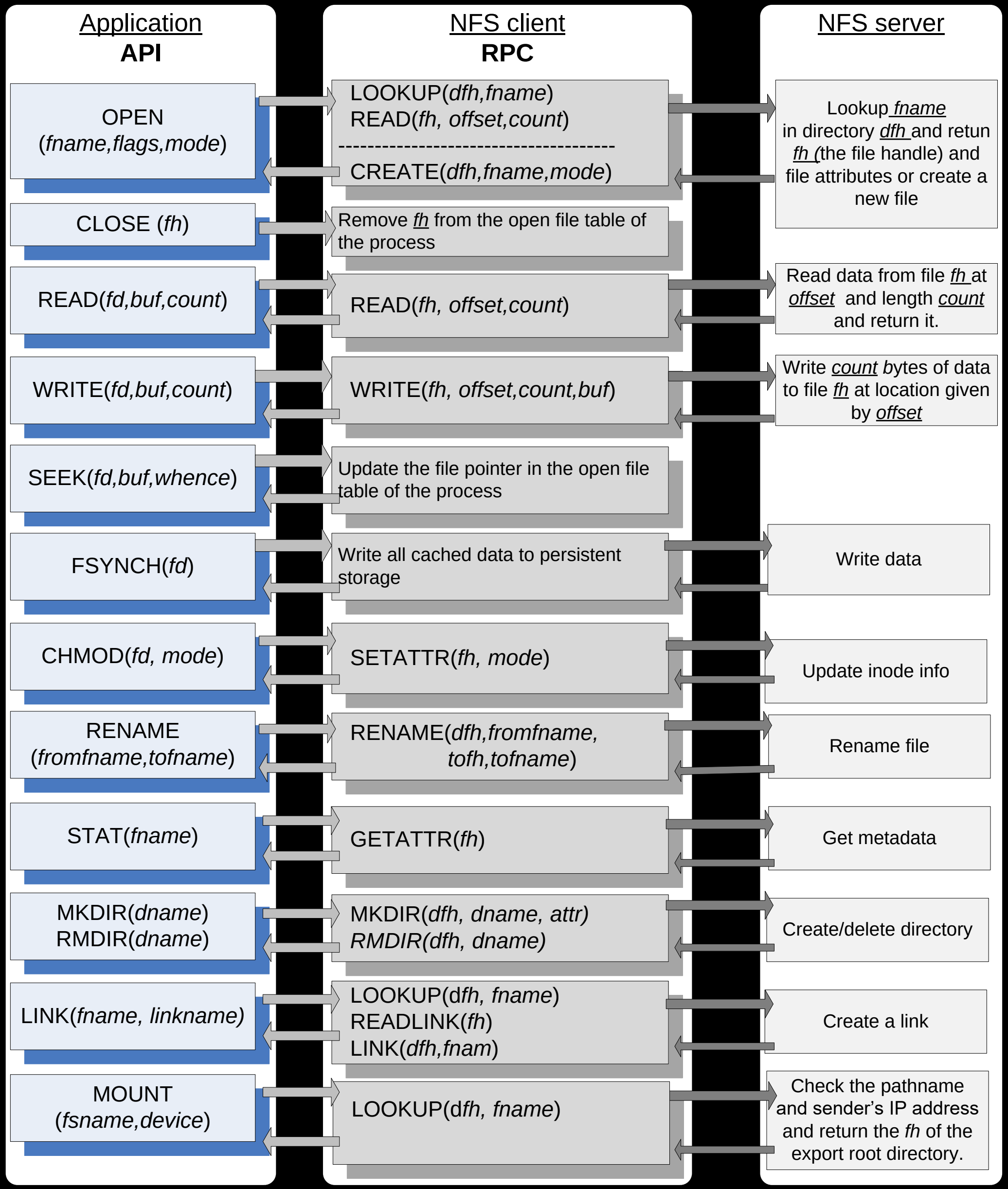
The NFS client-server interaction. The vnode layer implements file operation in a uniform manner, regardless of whether the file is local or remote.

An operation targeting a local file is directed to the local file system, while one for a remote file involves NFS; an NSF client packages the relevant information about the target and the NFS server passes it to the vnode layer on the remote host which, in turn, directs it to the remote file system.

Comparison of Distributed File Systems



File system	Cache size and location	Writing policy	Consistency guarantees	Cache validation
NFS	Fixed, memory	On close or 30 sec. delay	Sequential	On open, with server consent
AFS	Fixed, disk	On close	Sequential	When modified server asks client
Sprite	Variable, memory	30 sec. delay	Sequential, concurrent	On open, with server consent
Locus	Fixed, memory	On close	Sequential, concurrent	On open, with server consent
Apollo	Variable, memory	Delayed or on unlock	Sequential	On open, with server consent
RFS	Fixed, memory	Write-through	Sequential, concurrent	On open, with server consent



Application API	NFS client RPC	NFS server

OPEN
(fname,flags,mode)
	LOOKUP(dfh,fname)
	READ(fh, offset,count)
	--- 	
	CREATE(dfh,fname,mode)
		Lookup fname in directory dfh and retun fh (the file handle) and file attributes or create a new file

CLOSE (fh)	Remove fh from the open file table of the process	
READ(fd,buf,count)	READ(fh, offset,count)	Read data from file fh at offset and length count and return it.
WRITE(fd,buf,count)	WRITE(fh, offset,count,buf)	Write count bytes of data to file fh at location given by offset
SEEK(fd,buf,whence)	Update the file pointer in the open file table of the process	
FSYNCH(fd)	Write all cached data to persistent storage	Write data
CHMOD(fd, mode)	SETATTR(fh, mode)	Update inode info
RENAME (fromfname,tofname)	RENAME(dfh,fromfname, tofh,tofname)	Rename file
STAT(fname)	GETATTR(fh)	Get metadata

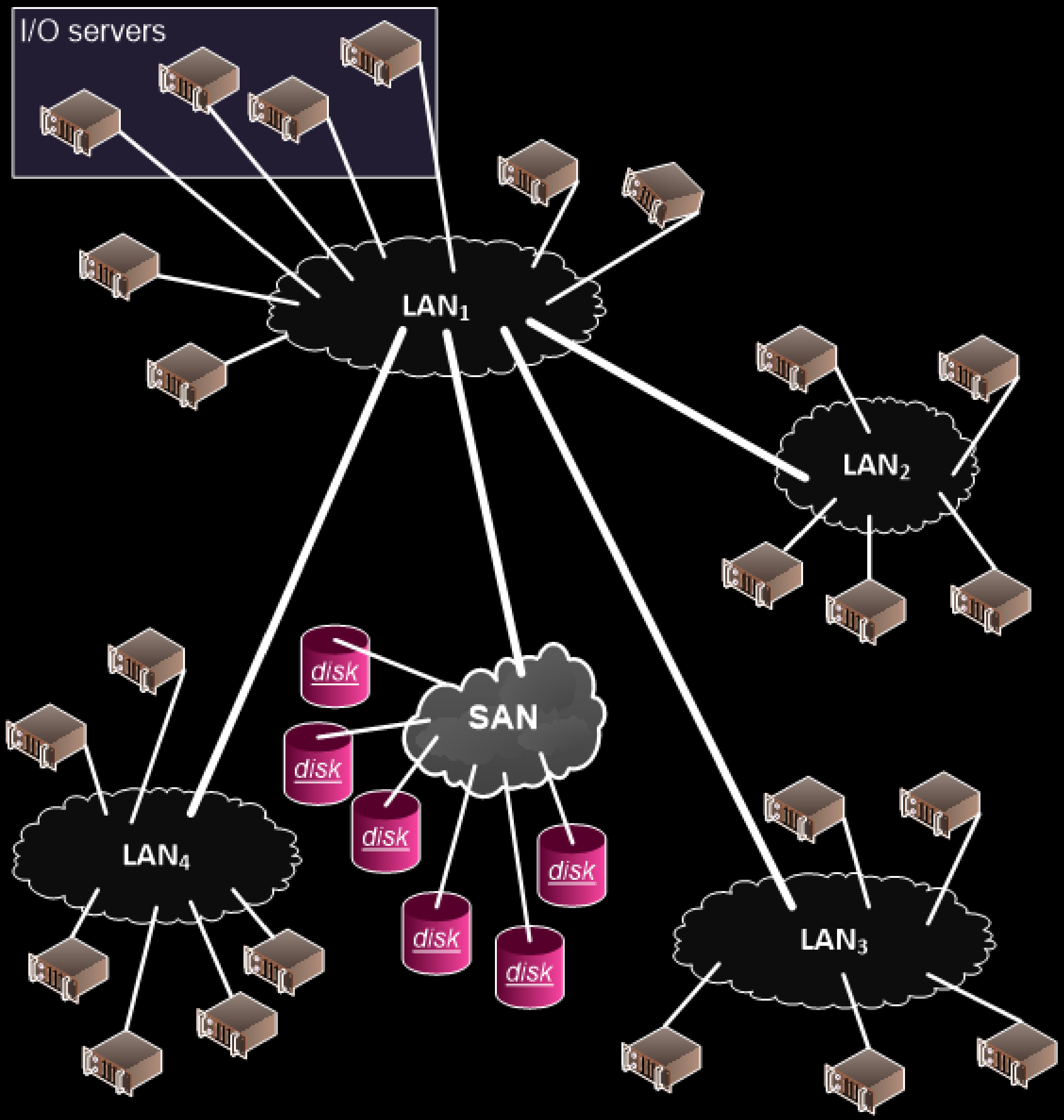
MKDIR(dname)
RMDIR(dname)
	MKDIR(dfh, dname, attr)
	RMDIR(dfh, dname)
		Create/delete directory

LINK(fname, linkname)	LOOKUP(dfh, fname)
	READLINK(fh)
	LINK(dfh,fnam)
		Create a link

MOUNT (fsname,device)	LOOKUP(dfh, fname)	Check the pathname and sender's IP address and return the fh of the export root directory.

General Parallel File System (GPFS)

- Parallel I/O implies concurrent execution of multiple input/output operations.
- Support for parallel I/O is essential for the performance of many applications.
- Concurrency control is a critical issue for parallel file systems. Several semantics for handling the shared access are possible.
  - For example, when the clients share the file pointer successive reads issued by multiple clients advance the file pointer; another semantics is to allow each client to have its own file pointer.
- GPFS
  - Developed at IBM in the early 2000s as a successor of the Tiger Shark multimedia file system.
  - Designed for optimal performance of large clusters; it can support a file system of up to 4 PB consisting of up to 4,096 disks of 1 TB each.
  - Maximum file size is (2^{63} -1) bytes.
  - A file consists of blocks of equal size, ranging from 16 KB to 1 MB, stripped across several disks.


I/O servers
LAN$_1$
LAN$_2$
LAN$_3$
LAN$_4$
SAN
disk
disk
disk
disk
disk
disk
disk

GPFS reliability

- To recover from system failures, GPFS records all metadata updates in a write-ahead log file.
- Write-ahead: updates are written to persistent storage only after the log records have been written.
- The log files are maintained by each I/O node for each file system it mounts; any I/O node can initiate recovery on behalf of a failed node.
- Data striping allows concurrent access and improves performance, but can have unpleasant side-effects. When a single disk fails, a large number of files are affected.
- The system uses RAID devices with the stripes equal to the block size and dual-attached RAID controllers.
- To further improve the fault tolerance of the system, GPFS data files as well as metadata are replicated on two different physical disks.

GPFS distributed locking

- In GPFS, consistency and synchronization are ensured by a distributed locking mechanism. A central lock manager grants lock tokens to local lock managers running in each I/O node. Lock tokens are also used by the cache management system.
- Lock granularity has important implications on the performance. GPFS uses a variety of techniques for different types of data.
  - Byte-range tokens – used for read and write operations to data files as follows: the first node attempting to write to a file acquires a token covering the entire file; this node is allowed to carry out all reads and writes to the file without any need for permission until a second node attempts to write to the same file; then, the range of the token given to the first node is restricted.
  - Data-shipping – an alternative to byte-range locking, allows fine-grain data sharing. In this mode the file blocks are controlled by the I/O nodes in a round-robin manner. A node forwards a read or write operation to the node controlling the target block, the only one allowed to access the file.

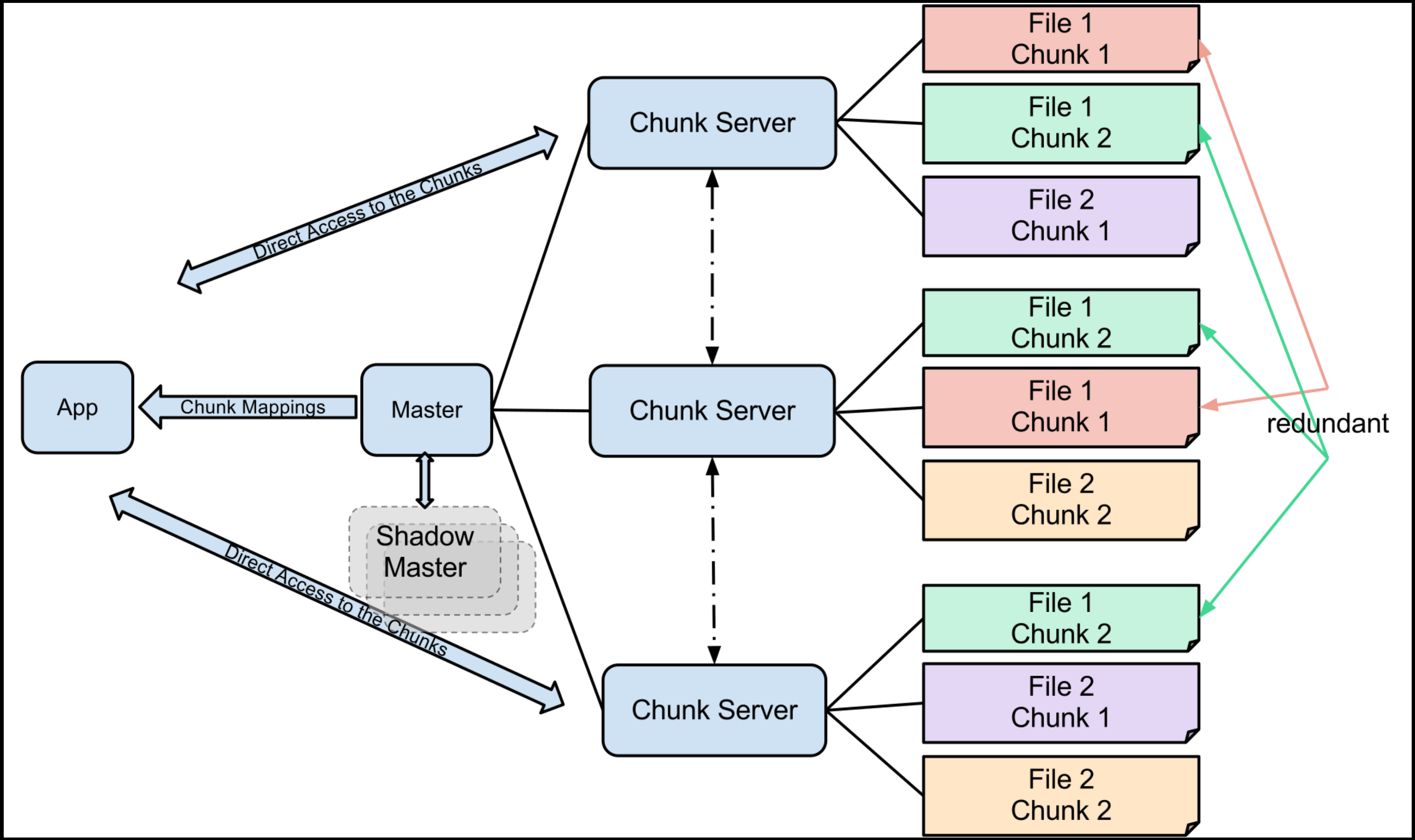
Google File System (GFS)

- GFS – developed in the late 1990s; uses thousands of storage systems built from *inexpensive commodity components* to provide petabytes of storage to a large user community with diverse needs.
- Design considerations.
  - Scalability and reliability are critical features of the system; they must be considered from the beginning, rather than at some stage of the design.
  - The vast majority of files range in size from a few GB to hundreds of TB.
  - The most common operation is to append to an existing file; random write operations to a file are extremely infrequent.
  - Sequential read operations are the norm.
  - The users process the data in bulk and are less concerned with the response time.
  - The consistency model should be relaxed to simplify the system implementation but without placing an additional burden on the application developers.

GFS – design decisions

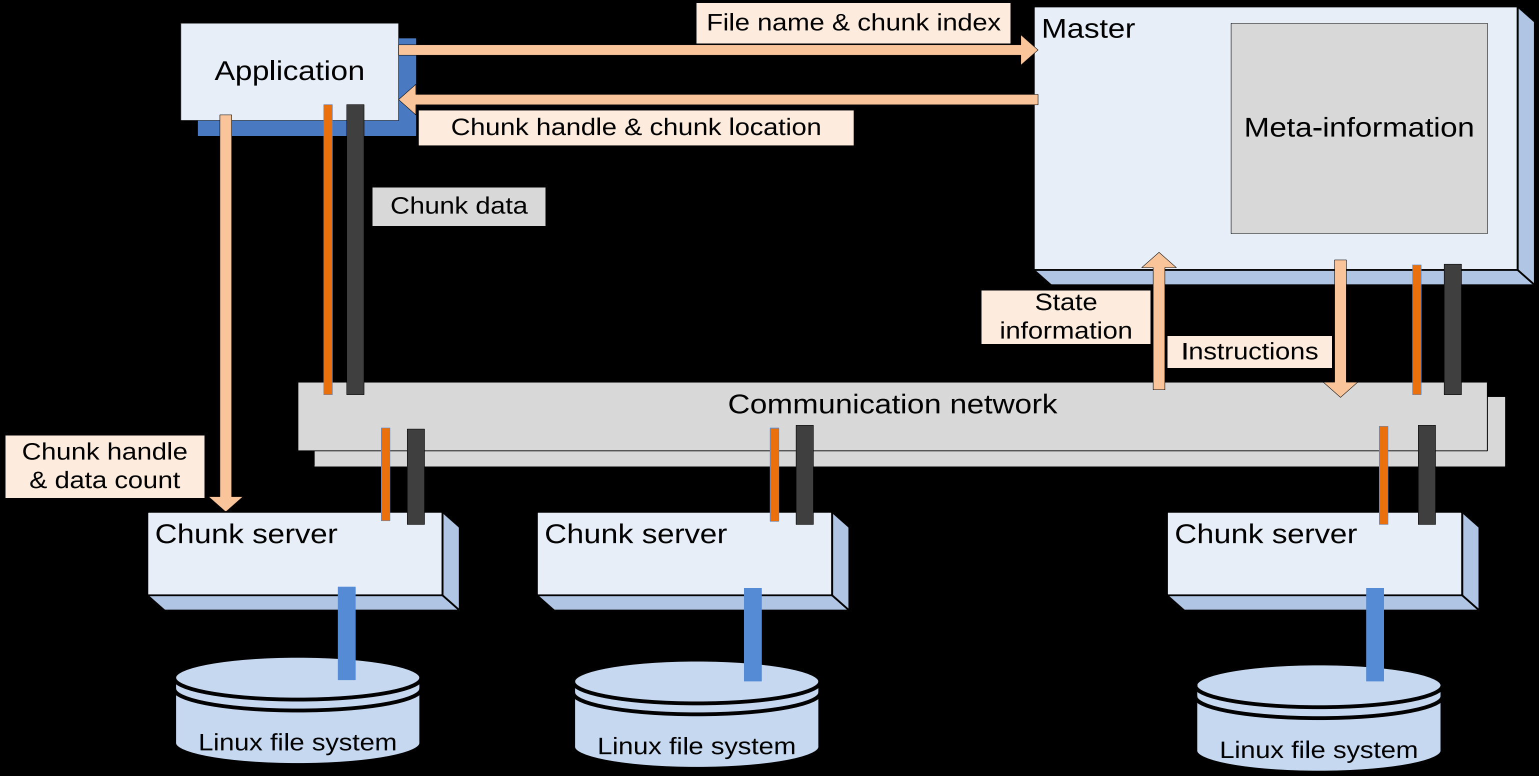
- Segment a file in large chunks.
- Implement an atomic file append operation allowing multiple applications operating concurrently to append to the same file.
- Build the cluster around a high-bandwidth rather than low-latency interconnection network. Separate the flow of control from the data flow. Pipeline data transfer over TCP connections. Exploit network topology by sending data to the closest node in the network.
- Eliminate caching at the client site. Caching increases the overhead for maintaining consistency among cashed copies.
- Ensure consistency by channeling critical file operations through a master, a component of the cluster which controls the entire system.
- Minimize the involvement of the master in file access operations to avoid hot-spot contention and to ensure scalability.
- Support efficient checkpointing and fast recovery mechanisms.
- Support an efficient garbage collection mechanism.

Google File System (GFS)



GFS chunks

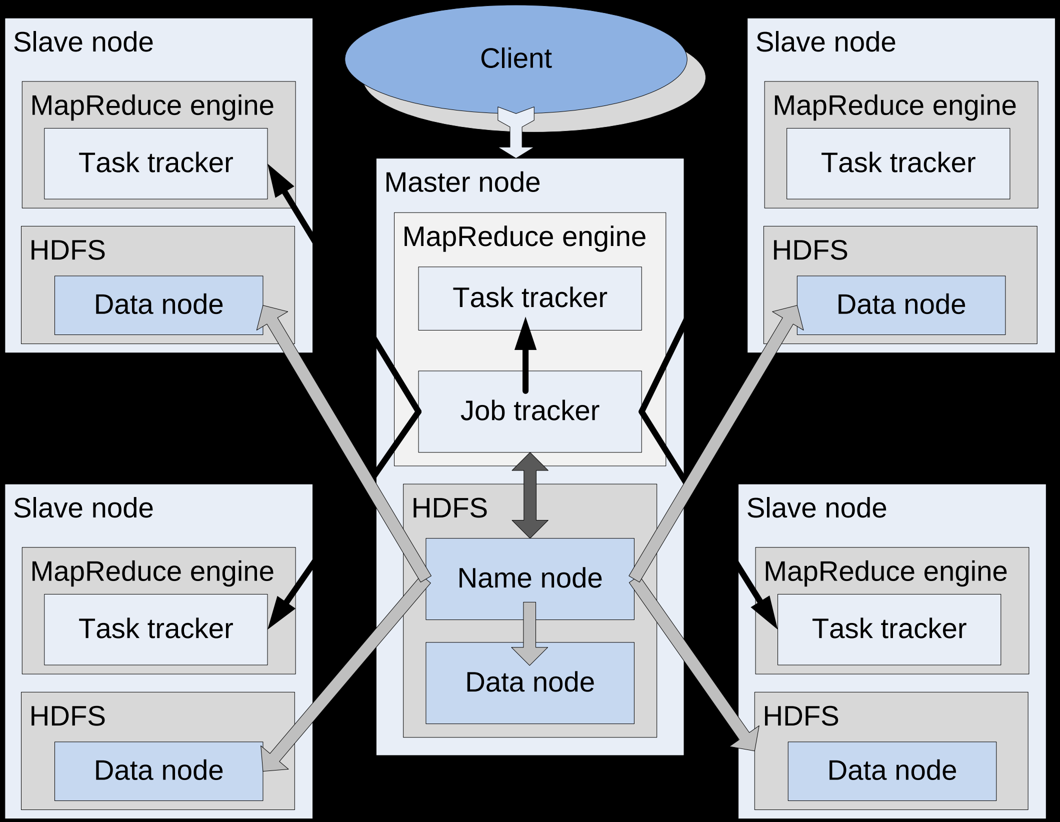
- GFS files are collections of fixed-size segments called chunks.
- The chunk size is 64 MB; this choice is motivated by the desire to optimize the performance for large files and to reduce the amount of metadata maintained by the system.
- A large chunk size increases the likelihood that multiple operations will be directed to the same chunk thus, it reduces the number of requests to locate the chunk and, at the same time, it allows the application to maintain a persistent network connection with the server where the chunk is located.
- A chunk consists of 64 KB blocks and each block has a 32 bit checksum.
- Chunks are stored on Linux files systems and are replicated on multiple sites; a user may change the number of the replicas, from the standard value of three, to any desired value.
- At the time of file creation each chunk is assigned a unique *chunk handle*.



- The architecture of a GFS cluster; the master maintains state information about all system components; it controls a number of chunk servers. A chunk server runs under Linux; it uses metadata provided by the master to communicate directly with the application. The data and the control paths are shown separately, data paths with thick lines and the control paths with thin lines. Arrows show the flow of control between the application, the master and the chunk servers.

Apache Hadoop

- Apache Hadoop an open source, Java-based software, supports distributed applications handling extremely large volumes of data.
- Hadoop is used by many organization from industry, government, and research; major IT companies e.g., Apple, IBM, HP, Microsoft, Yahoo, and Amazon, media companies e.g., New York Times and Fox, social networks including, Twitter, Facebook, and LinkedIn, and government agencies such as Federal Reserve.
- A Hadoop system has two components, a MapReduce engine and a database. The database could be the Hadoop File System (HDFS), Amazon's S3, or CloudStore, an implementation of GFS.
- HDFS is a distributed file system written in Java; it is portable, but it cannot be directly mounted on an existing operating system. HDFS is not fully POSIX compliant, but it is highly performant.



A Hadoop cluster using HDFS; the cluster includes a master and four slave nodes. Each node runs a MapReduce engine and a database engine. The job tracker of the master's engine communicates with task trackers on all the nodes and with the name node of HDFS. The name node of the HDFS shares information about the data placement with the job tracker to minimize communication between the nodes where data is located and the ones where it is needed.

Chubby - a locking service

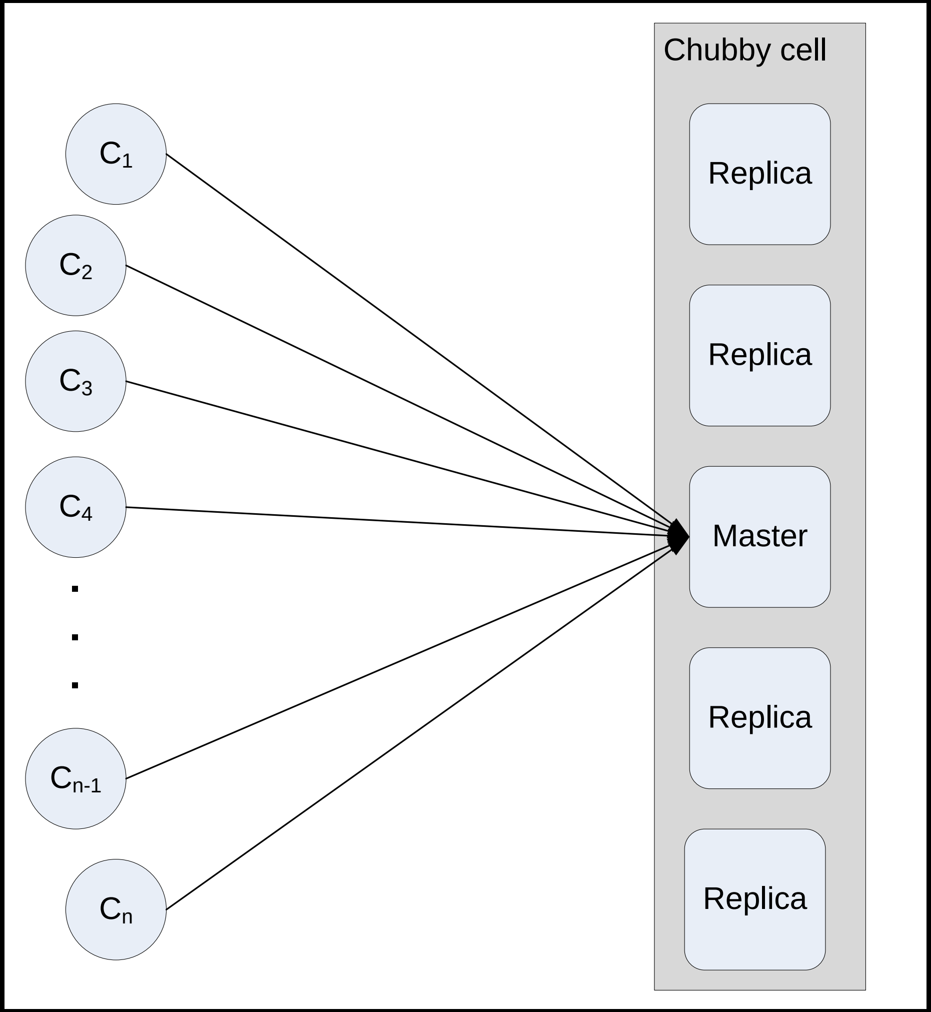
- Locks support the implementation of reliable storage for loosely-coupled distributed systems; they enable controlled access to shared storage and ensure atomicity of read and write operations.
- Distributed consensus problems, such as the election of a master from a group of data servers; e.g., the GFS master maintains state information about all systems components.
- Two approaches possible:
  - delegate to the clients the implementation of the consensus algorithm and provide a library of functions needed for this task.
  - create a locking service which implements a version of the asynchronous Paxos algorithm and provide a library to be linked with an application client.
- Chubby is based on the Paxos algorithm which guarantees safety without any timing assumptions, a necessary condition in a large-scale system when communication delays are unpredictable; the algorithm must use clocks to ensure liveliness and to overcome the impossibility of reaching consensus with a single faulty process.

The Paxos algorithm

- Used to reach consensus on sets of values, e.g., the sequence of entries in a replicated log.
- The phases of the algorithm.
  - Elect a replica to be the *master/coordinator*. When a master fails, several replicas may decide to assume the role of a master; to ensure that the result of the election is unique each replica generates a sequence number larger than any sequence number it has seen, in the range (1,**r**) where **r** is the number of replicas, and broadcasts it in a *propose* message. The replicas which have not seen a higher sequence number broadcast a *promise* reply and declare that they will reject proposals from other candidate masters; if the number of respondents represents a majority of replicas, the one who sent the propose message is elected as the master.
  - The master broadcasts to all replicas an *accept* message including the value it has selected and waits for replies, either *acknowledge* or *reject*.
  - Consensus is reached when the majority of the replicas send the *acknowledge* message; then the master broadcasts the *commit* message.

Locks

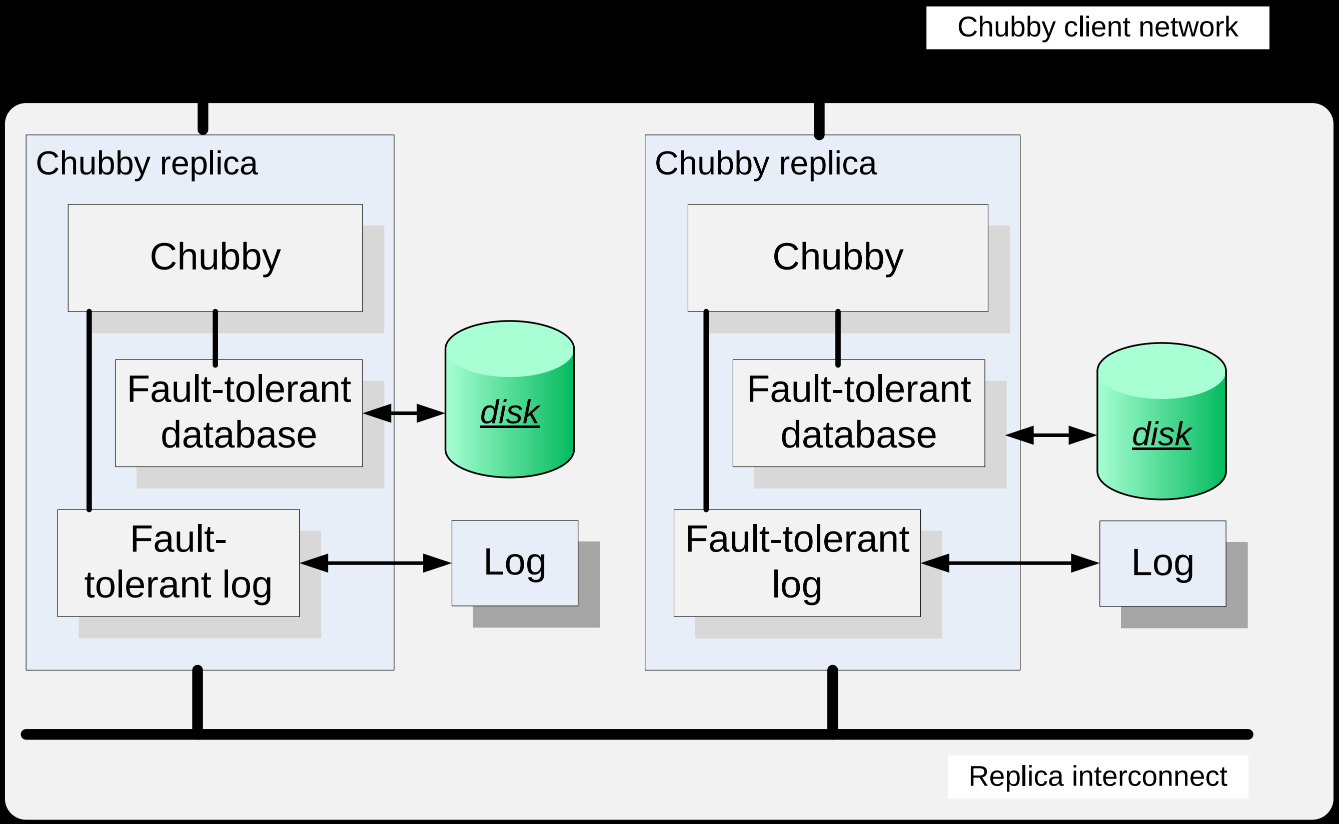
- Advisory locks – based on the assumption that all processes play by the rules; do not have any effect on processes that circumvent the locking mechanisms and access the shared objects directly.
- Mandatory locks – block access to the locked objects to all processes that do not hold the locks, regardless if they use locking primitives or not.
- Fine-grained locks – locks that can be held for only a very short time. Allow more application threads to access shared data in any time interval, but generate a larger workload for the lock server. When the lock server fails for a period of time, a larger number of applications are affected.
- Coarse-grained locks – locks held for a longer time.



- A Chubb cell consisting of 5 replicas, one of them elected as a master; **n** clients use RPCs to communicate with the master.

Chubby operation

- Clients use RPCs to request services from the master.
  - When it receives a write request, the master propagates the request to all replicas and waits for a reply from a majority of replicas before responding.
  - When it receives a read request, the master responds without consulting the replicas.
- The client interface of the system is similar to, yet simpler than, the one supported by the Unix file system; in addition, it includes notification for events related to file or system status.
- A client can subscribe to events such as: file contents modification, change or addition of a child node, master failure, lock acquired, conflicting lock requests, invalid file handle.
- Each file or directory can act as a lock. To write to a file the client must be the only one holding the file handle, while multiple clients may hold the file handle to read from the file.



Chubby replica architecture; the Chubby component implements the communication protocol with the clients. The system includes a component to transfer files to a fault-tolerant database and a fault-tolerant log component to write log entries. The fault-tolerant log uses the Paxos protocol to achieve consensus. Each replica has its own local file system; replicas communicate with one another using a dedicated interconnect and communicate with clients through a client network.

Transaction processing

- Online Transaction Processing (OLTP) → widely used by many cloud applications.
- Major requirements:
  - Short response time.
  - Scalability.
    - Vertical scaling → data and workload are distributed to systems that share resources, e.g., cores/processors, disks, and possibly RAM
    - Horizontal scaling → the systems do not share either primary or secondary storage.
- The search for alternate models to store the data on a cloud is motivated by the needs of OLTP applications:
  - decrease the latency by caching frequently used data in memory.
  - allow multiple transactions to occur at the same time and decrease the response time by distributing the data on a large number of servers.

Sources of OLTP overhead

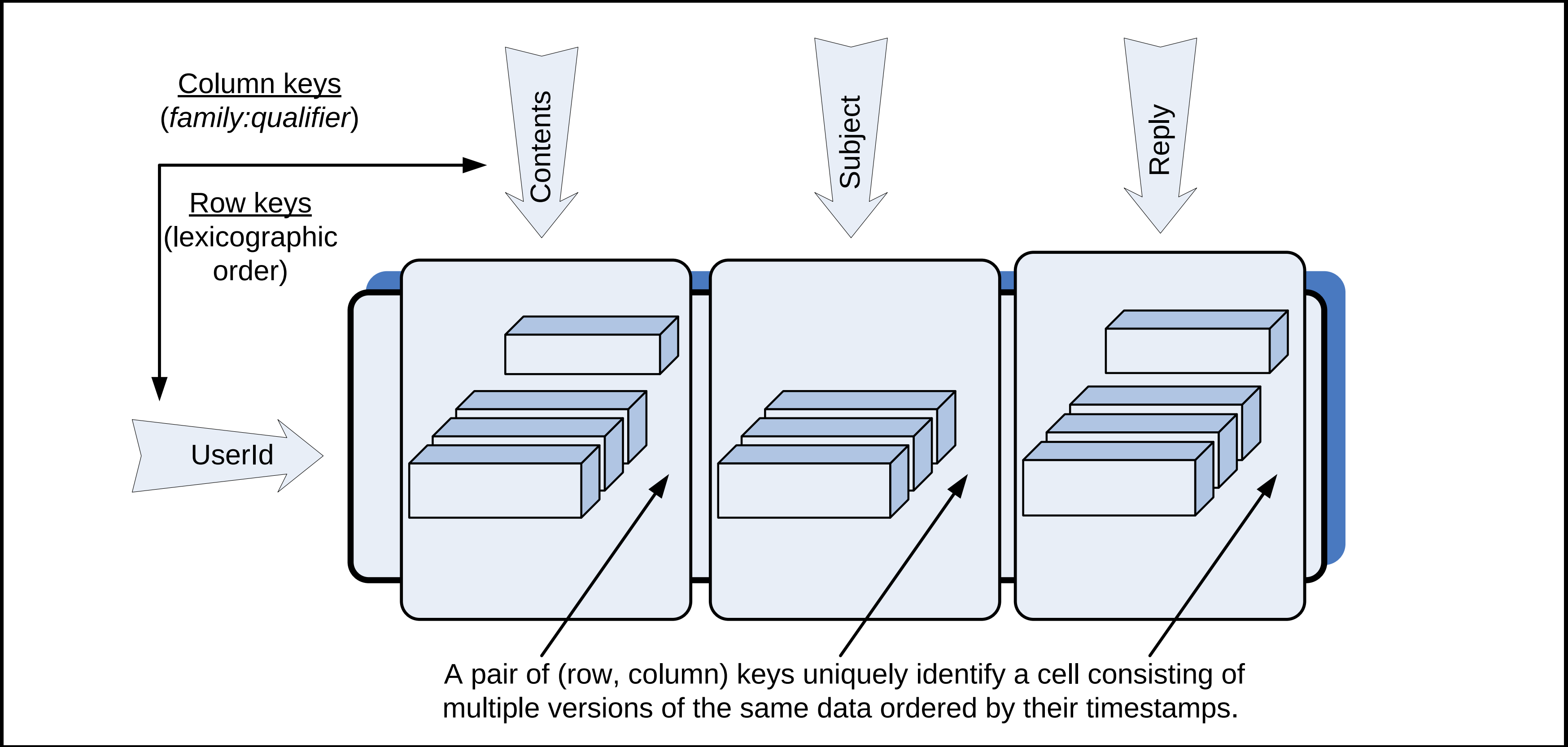
- Four sources with equal contribution:
  - Logging – expensive because traditional databases require transaction durability thus, every ***write*** to the database can only be completed after the log has been updated.
  - Locking – to guarantee atomicity, transactions lock every record and this requires access to a lock table.
  - Latching – many operations require multi-threading and the access to shared data structures, such as lock tables, demands short-term latches for coordination. A latch is a counter that triggers an event when it reaches zero; for example a master thread initiates a counter with the number of worker threads and waits to be notified when all of them have finished.
  - Buffer management.
- The breakdown of the instruction count for these operations in existing DBMS is: 34.6% for buffer management, 14.2% for latching, 16.2 % for locking, 11.9% for logging, and 16.2 % for manual optimization.

NoSQL databases

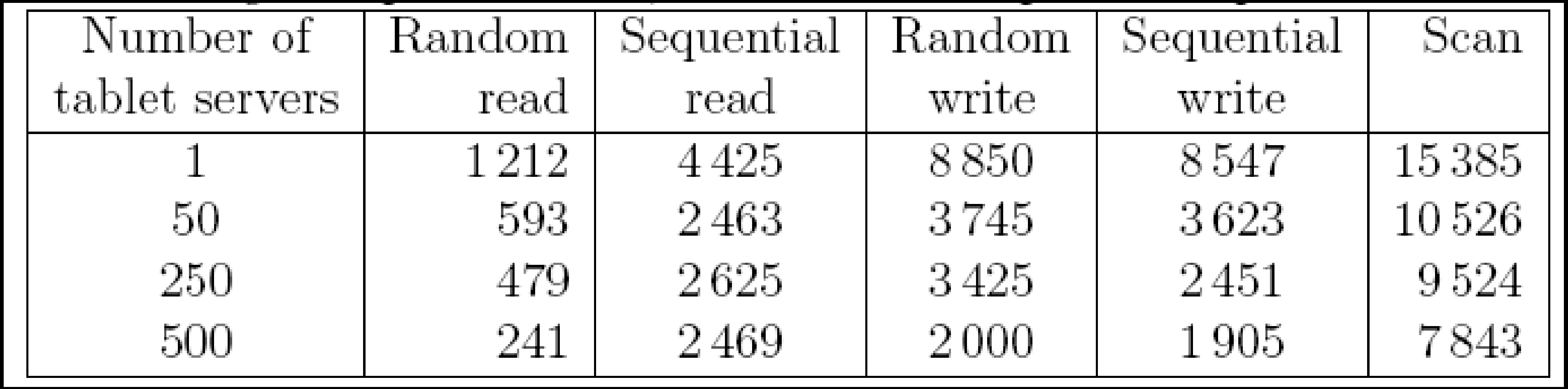
- The name NoSQL is misleading. Stonebreaker notes that "blinding performance depends on removing overhead. Such overhead has nothing to do with SQL, it revolves around traditional implementations of ACID transactions, multi-threading, and disk management."
- The soft-state approach allows data to be inconsistent and transfers the task of implementing only the subset of the ACID properties required by a specific application to the application developer.
- NoSQL systems ensure that data will be eventually consistent at some future point in time, instead of enforcing consistency at the time when a transaction is *committed*.
- Attributes:
  - Scale well.
  - Do not exhibit a single point of failure.
  - Have built-in support for consensus-based decisions.
  - Support partitioning and replication as basic primitives.

Bigtable

- Distributed storage system developed by Google to
  - store massive amounts of data.
  - scale up to thousands of storage servers.
- The system uses
  - Google File System → to store user data and system information.
  - Chubby distributed lock service → to guarantee atomic **read** and **write** operations; the directories and the files in the namespace of Chubby are used as locks.
- Simple and flexible data model a multidimensional array of cells.
  - A row key → an arbitrary string of up to 64 KB and a row range is partitioned into tablets serving as units for load balancing. The timestamps used to index different versions of the data in a cell are 64-bit integers; their interpretation can be defined by the application, while the default is the time of an event in microseconds.
  - A column key → consists of a string, a set of printable characters, and an arbitrary string as qualifier.


Column keys
(*family:qualifier*)
Row keys
(lexicographic order)
Contents
Subject
Reply
UserId
A pair of (row, column) keys uniquely identify a cell consisting of multiple versions of the same data ordered by their timestamps.

The organization of an Email application as a sparse, distributed, multidimensional map. The slice of Bigtable shown consists of a row with the key *UserId* and three *family* columns; the *Contents* key identifies the cell holding the contents of Emails received, the one with key *Subject* identifies the subject of Emails, and the one with the key *Reply* identifies the cell holding the replies; the version of records in each cell are ordered according to timestamps. Row keys are ordered lexicographically; a column key is obtained by concatenating *family* and the *qualifier* fields



Number of tablet servers	Random read	Sequential read	Random write	Sequential write	Scan
1	1 212	4 425	8 850	8 547	15 385
50	593	2 463	3 745	3 623	10 526
250	479	2 625	3 425	2 451	9 524
500	241	2 469	2 000	1 905	7 843

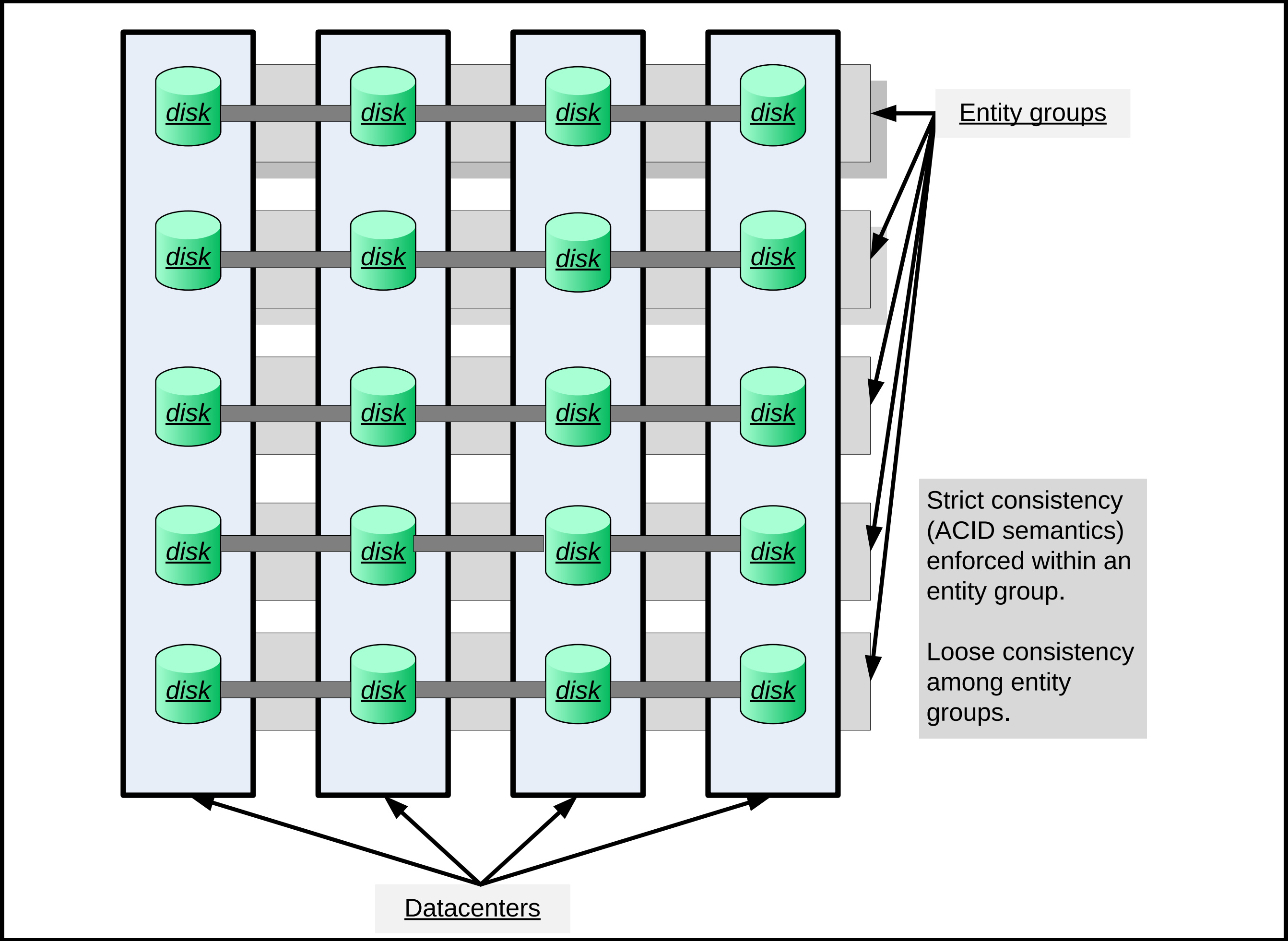
Bigtable performance – the number of operations

Megastore

- Scalable storage for online services. Widely used internally at Google, it handles some 23 billion transactions daily, 3 billion **write** and 20 billion **read** transactions.
- The system, distributed over several data centers, has a very large capacity, 1 PB in 2011, and it is highly available.
- Each partition is replicated in data centers in different geographic areas. The system supports full ACID semantics within each partition and provides limited consistency guarantees across partitions.
- The Paxos consensus algorithm is used to replicate primary user data, metadata, and system configuration information across data centers and for locking. The version of the Paxos algorithm does not require a single master, instead any node can initiate read and write operations to a write-ahead log replicated to a group of symmetric peers.
- The system makes extensive use of Bigtable.

Megastore’s data model

- Reflects a middle ground between traditional and NoSQL databases.
- The data model is declared in a schema consisting of a set of *tables*, composed of *entries*.
- An entry ➔ a collection of named and *typed properties*; the unique primary key of an entity in a table is created as a composition of entry properties. An entity group consists of the primary entity and all entities that reference it.
- A table can be a root or a child table.



Strict consistency (ACID semantics) enforced within an entity group.

Loose consistency among entity groups.

Megastore organization. The data is partitioned into entity groups; full ACID semantics within each partition and limited consistency guarantees across partitions are supported. A partition is replicated across data centers in different geographic areas.