- Place chunks on servers with below-average disk space utilization because our goal is to equalize disk utilization
- place chunks on servers with low number of recent creations, prevents heavy write traffic in near future
- spread chunks across racks

GARBAGE COLLECTION

- GFS does not immediately reclaim physical storage after a file is deleted
- "Lazy" garbage collection mechanism
  - master logs are changed to a "hidden file name"
  - master removes hidden files during regular file system scan (3 day window to undelete)

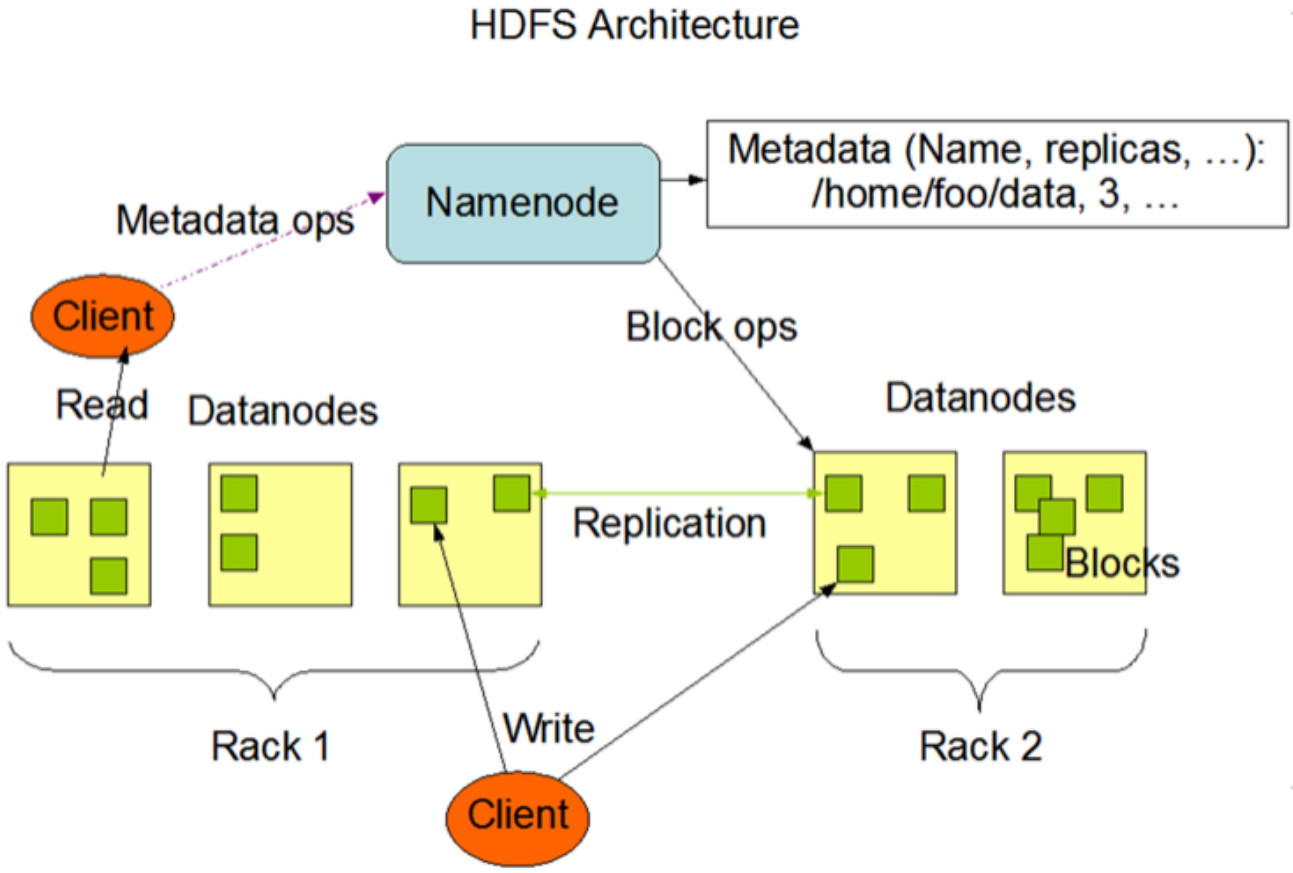
HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

- Open source clone of GFS
  - similar assumptions, design, architecture
    Differences
- no support for random writes, append only
- platform independence (implemented in java)

Terminology

- HDFS -> GFS
- namenode -> master
- datanode -> chunkserver
- block -> chunk

- edit log -> operation log



Bigtable: a distributed storage system for structured data
(HBase is an open-sourced Bigtable)

Issues with HDFS

- inefficient for random accesses (optimized for large-file sequential accesses)
- inefficient fro writers (optimized for read)
- inefficient for supporting structured/semi-structured data, in particular big tables (AKA sparse tables)

Usually when tables have lot of columns there are several empty entries.

Traditional DBs must store empty entries, which is a waste of space.

Sparse table is a semi-structured data model where tuples do not exactly follow a fixed schema

Formally, a Bigtable is a sparse, distributed, persistent, multidimensional sorted map

The map is indexed by a row and column key, and a timestamp.
Each value in the map is an uninterpreted array of bytes.

UserID	Name	Age	City	JobTitle	Company	Salary	Address
1	test1			SDE		S1	A1
2		20		Sales	C2		
3			Chicago		C3	S3	A3
4	test4		New York	Consultant		S4	
5	test5	25					A5

- We can use key-value store to represent this table:
  - (Row ID + Column Name ➔ Value)
  - Key is a compound key
  - This can reduce space (only store non-empty cells)

(1,name, test1)
(1,JobTitle, SDE)
(1,Salary, S1)

key value

)UE

For new updates we do not update immediately (lazy update)
must append a new tuple of the same key but with a different timestamp for every cell.

- every cell is a collection of pairs with (value, timestamp)
- represents the value at that time
- {(SDE,0),(Senior SDE, 1)}

Support timestamp in the key-value model:

- (Row ID + Column Name + Timestamp ➔ Value)
- This can support updates efficiently

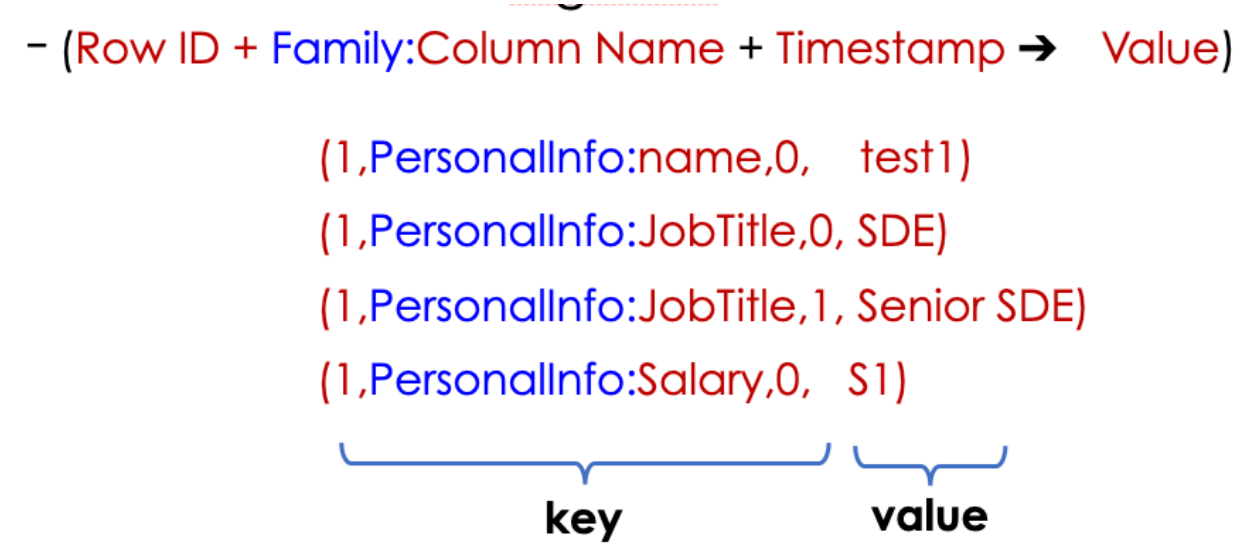
(1,name,0, test1)
(1,JobTitle,0, SDE)
(1,JobTitle,1, Senior SDE)
(1,Salary,0, S1)

key value

- Bigtable organizes the columns into *column families*
  - Easier to manage because there are many columns
  - every column is referenced by family

	PersonalInfo			JobInfo			
UserID	**Name**	**Age**	**City**	**JobTitle**	**Company**	**Salary**	**Address**
1	test1			SDE		S1	A1
2		20		Sales	C2		
3			Chicago		C3	S3	A3
4	test4		New York	Consultant		S4	
5	test5	25					A5

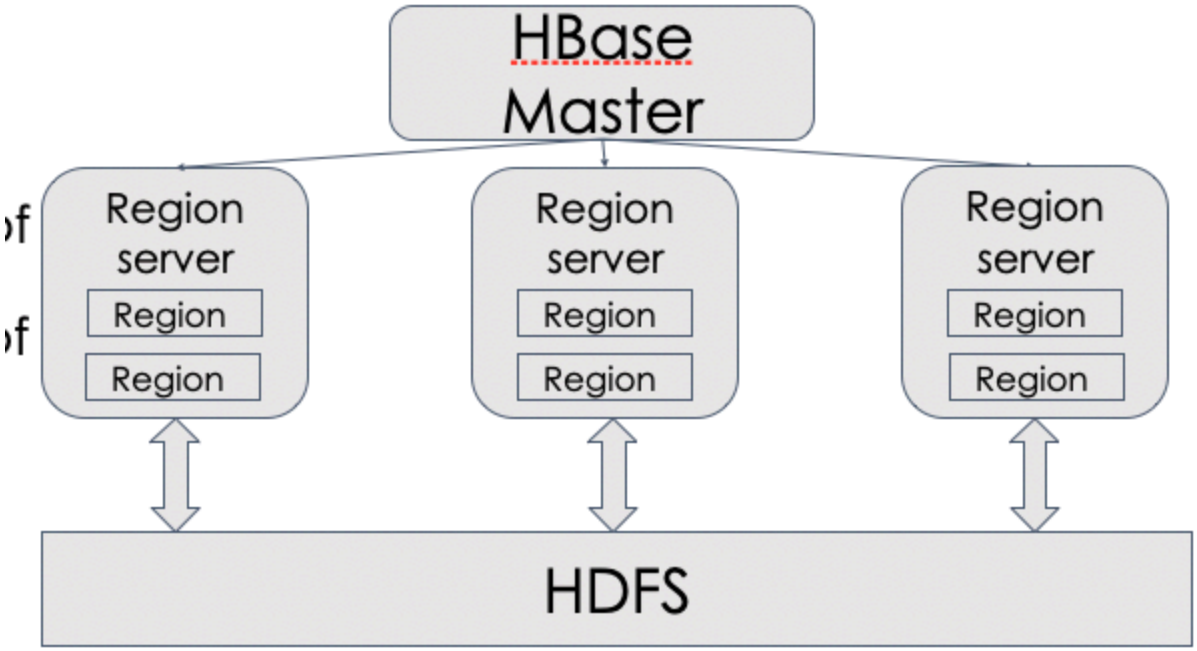
Naturally we need to include column families in the key value and thus:



HBASE STORAGE

- on top of HDFS
- HBase tables are divided horizontally by row key range into regions
- regions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of column families
- region server run on HDFS datanode which is present in hadoop cluster

- storage inside each region is based on log-structured merge tree



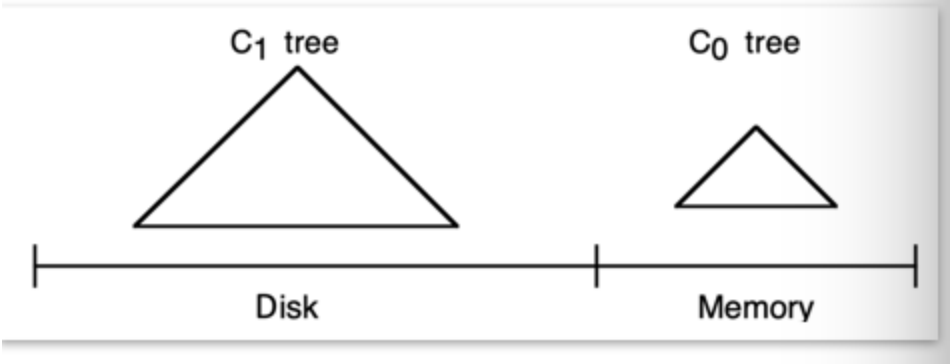
LOG STRUCTURED MERGE TREE (LSM)

- used widely in modern DBs systems
- more than a data structure or storage engine, it is a design principle (okay)
- B-tree insertion incurs many random writes -> LSM converts random writes to sequential writes
- B tree insertion incurs high costs due to in place update -> LSM uses out of place update
  Any static or hard to update structure (vector index) can use LSM

Basic idea of LSM

- if you have existing structure like index table, and new updates come in
- do not update existing structure directly
- instead store new updates in separate structure
- merge the two structures later on
- this is know as out of place update
- it is an immutable index (while B tree is mutable index)

- it has two parts, main memory component (mutable), disk component (immutable)



- Initially C0 and C1 are empty
- when data comes go to C0
- when C0's size exceed a threshold, flush to disk becoming C1
- new data comes again go to C0
- when C0's size exceed a threshold merge with C1
- no random writes to the disk tree C1
- only sequential accesses to C1 with a big chunk
- background compaction
- improve performance for write intensive workloads
- LSM can contain multiple levels
- advantage of LSM: no random writes, no in place update