BIG DATA

Sistemas de Datos Distribuidos

MSc. Felipe Meza

Contenidos

- Google File System (GFS)
- Bigtable
- Otros sistemas…

Google File System (GFS)

GFS: The Google File System

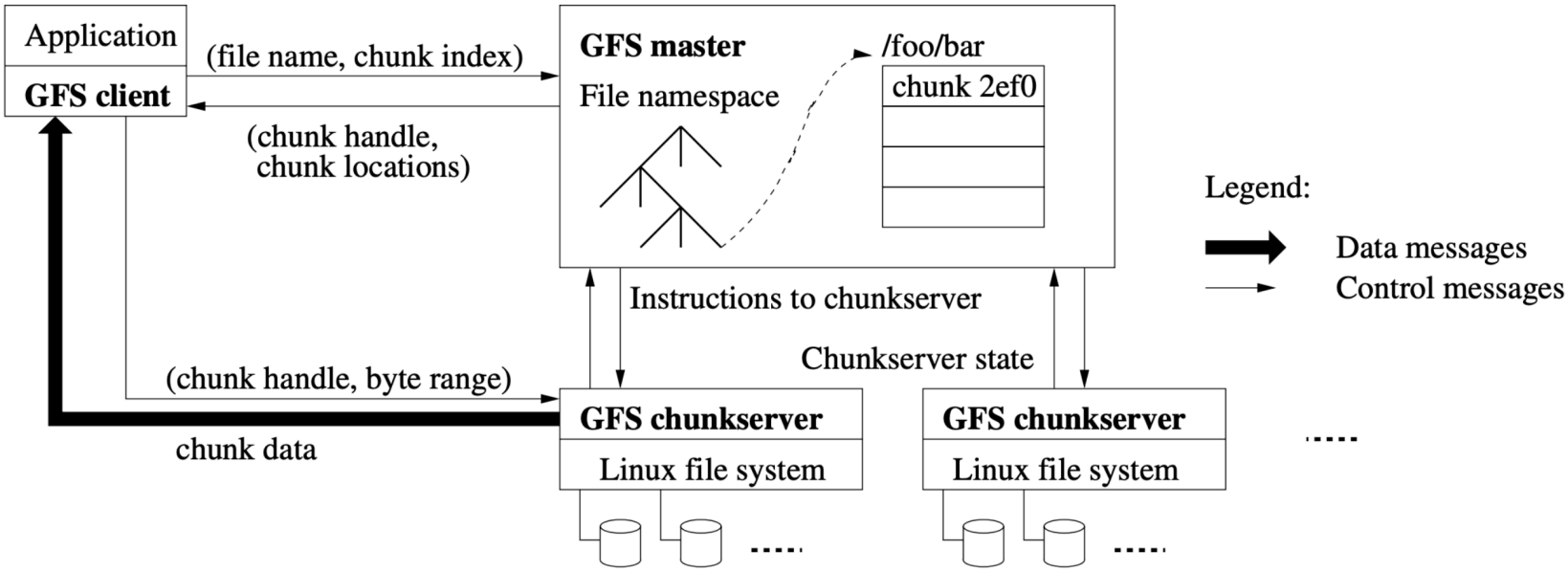
- Sistema diseñado a lo interno de ***Google*** para satisfacer las demandas internas de almacenamiento de la compañía.
- Principios de operación en ambiente de ***alta demanda***:
  - Dado el volumen, la falla de ***componentes*** es frecuente.
  - Millares de ***solicitudes***.
  - Monitoreo, detección de fallas y recuperación ***automática***.

GFS: Arquitectura

- Consiste en clusters donde cada uno posee las siguientes partes:
  - Un maestro.
  - Múltiples chunkservers.
  - Clientes.

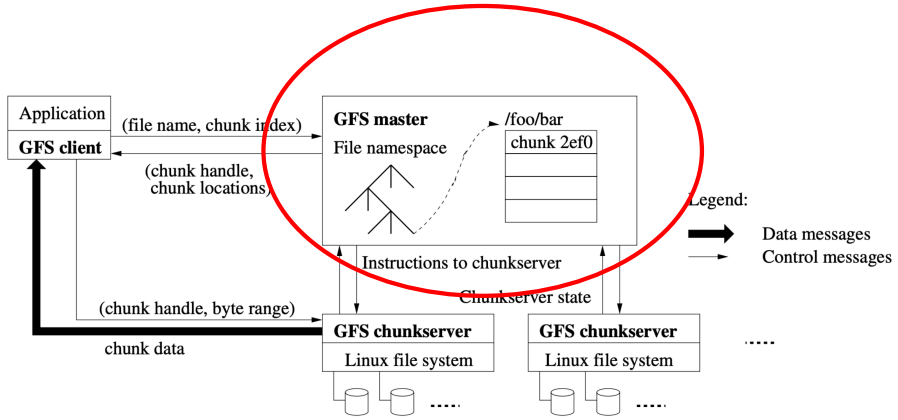
- Principio de Operación:
  - Los archivos se dividen en “pedazos” (chunks).
  - El maestro asigna un ID a cada chunk.
  - Cada chunk es replicado en multiples Chunkservers (3 default).
  - Los clientes acceden al chunkservers mediante un ***protocolo***.
  - Los clientes **NO** acceden al Maestro (cuellos de botella), sólo para control.

GFS: Arquitectura



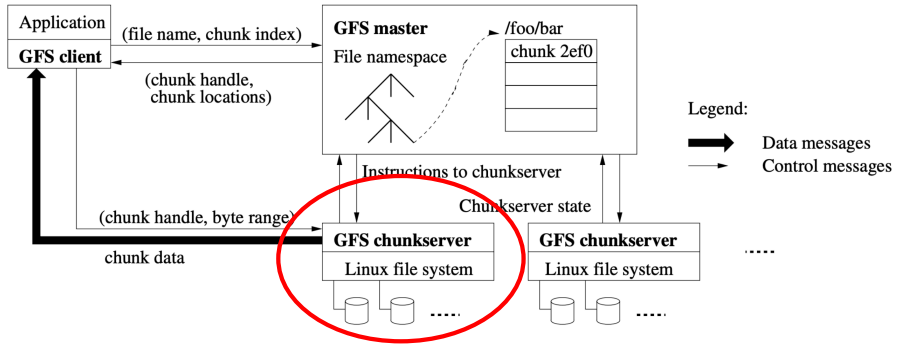
GFS: Arquitectura - Maestro

- Administra el *namespace*.
- Información de control y acceso.
- Mapea archivos y chunks, los localiza.
- Recolecta basura (chunks huérfanos).
- Migra chunks entre servidores.



GFS: Arquitectura - chunkserver

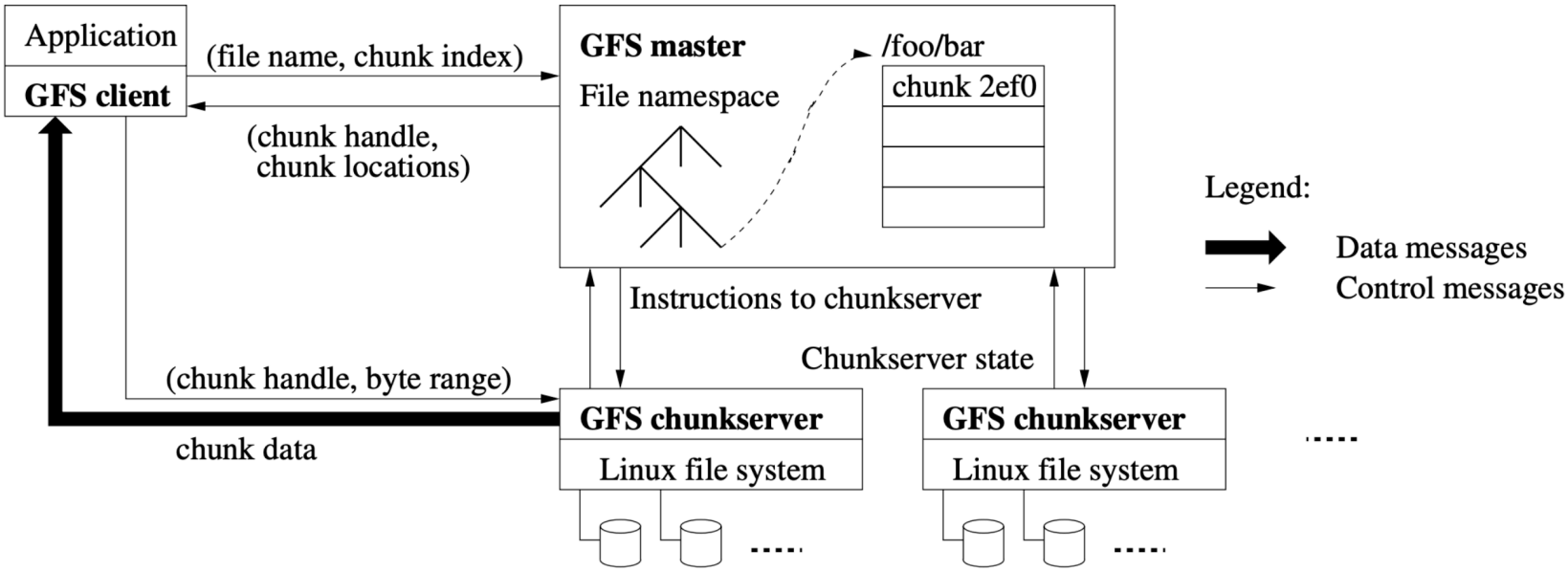
- Los clientes lo acceden directamente.
- Máquina de bajo costo.
- Corre un programa cliente.



GFS: Protocolo

- Los clientes solicitan una serie de bytes (nombre de archivo), como se haría con un sistema de archivos normal.
- Se traduce la información al ID del chunk, dentro del archivo que se requiere.
- Se envía al maestro la petición del archivo e ID de pedazo.
- Con esto el maestro indica cuál servidor posee la información y se puede solicitar directamente.

GFS: Arquitectura



GFS: Tamaño del *chunk*

- Por convención se ha definido en 64MB, con las siguientes:
  - VENTAJAS:
    - Menor número de interacciones con el *Maestro*.
    - Mejor rendimiento en lecturas secuenciales.
    - No requiere de conexión continua, reduce el tráfico.
    - Tabla de IDs de menor tamaño.
  - DESVENTAJAS:
    - Al inicio radicó, en que muchos archivos pequeños eran accedidos por muchos clientes en un solo momento.
    - Se solucionó creando réplicas de los archivos pequeños.

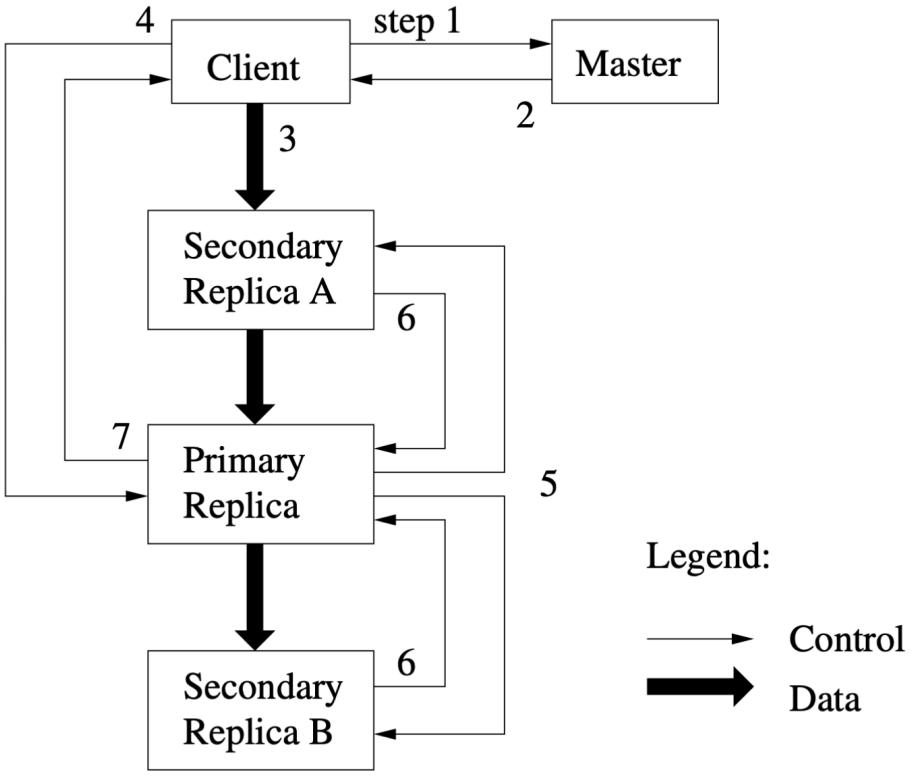
GFS: Metadatos y Almacenamiento

- Los metadatos se almacenan en memoria para efectos de rendimiento y recuperación rápida de fallas.
- Almacenamiento en *Maestro*, consiste en:
  - Mapeo de archivos y *chunks* (se obtiene dinámicamente de los *chunkservers*).
  - El espacio de nombre (e.g. cómo se llaman los archivos y sus *chunks*)
  - Localización de réplicas.

GFS: Proceso de escritura

- Dado que cada chunk está replicado, debe contarse con un protocolo estructurado para la escritura de los datos.
- El maestro maneja el concepto de *arrendamiento* que se asigna a los diferentes *chunks* secuencial y determinísticamente.
- El *chunk* que en este momento está asignado, se considera el primario.

GFS: Proceso de escritura



GFS: Proceso de escritura

1. Un cliente le pregunta al maestro que *chunk* tiene el arrendamiento.
2. El maestro responde y además envía la localización de todas las réplicas.
3. El cliente envía la información a todas las réplicas.
4. El cliente pide al primario aplicar la mutación.
5. El primario reenvía todas las modificaciones a las réplicas, usando la misma secuencia definida por el primario.
6. Los secundarios confirman al primario las escrituras.
7. El primario responde al cliente.

GFS: Proceso de escritura

- Se considera exitoso **solamente** cuando todos fueron escritos. Si al menos uno falló, el estado se considera inconsistente. El código en el cliente está diseñado para reintentar.

GFS: Ejemplo de clusters GFS

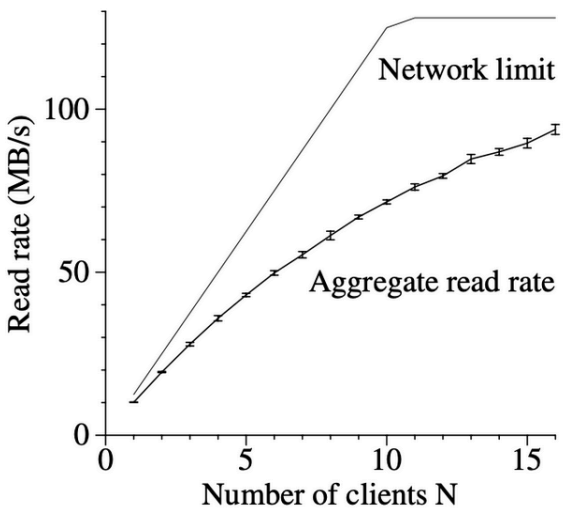
Cluster	A	B
Chunkservers	342	227
Available disk space	72 TB	180 TB
Used disk space	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at chunkservers	13 GB	21 GB
Metadata at master	48 MB	60 MB

A: R&D
B: Producción

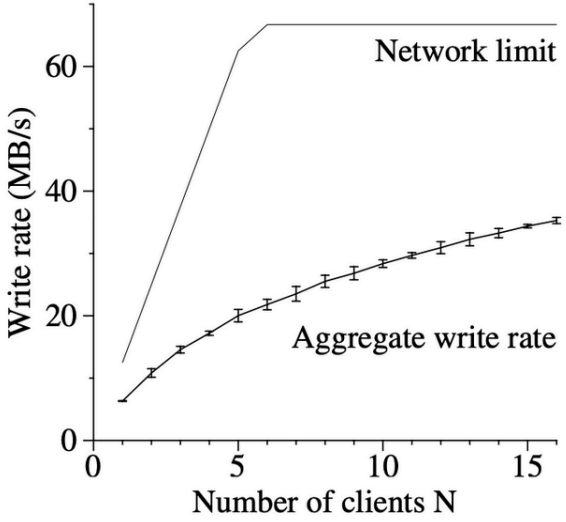
GFS: Ejemplo de clusters GFS

- Observaciones:
  - Cientos de servidores cada uno soporta varios TB.
  - Archivos tienen 3 réplicas.
  - Metadatos en master *únicamente* ~50MB

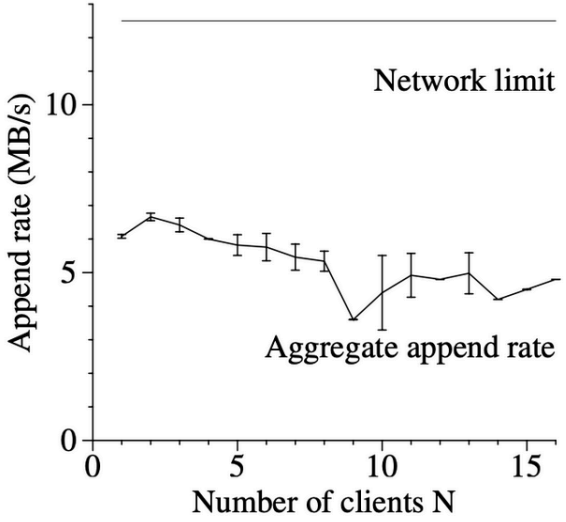
GFS: Ejemplo de clusters GFS



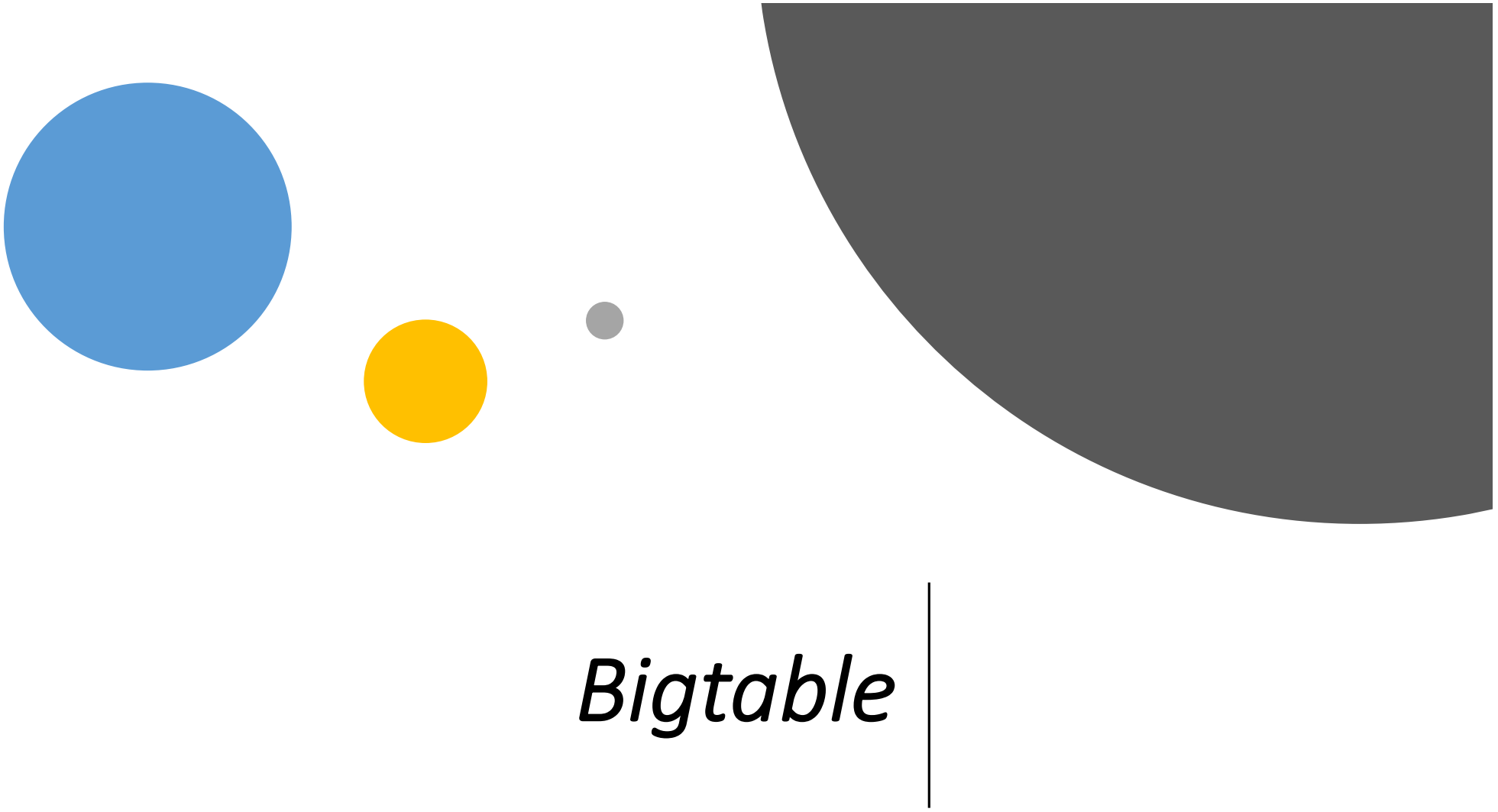
(a) Reads



(b) Writes



(c) Record appends


Bigtable

Bigtable

Se puede pensar como un híbrido entre una base de datos y un sistema de archivos.

Objetivo primario: almacenar datos estructurados, que pueda escalar a petabytes de información con acceso rápido y confiable, a lo largo de miles de máquinas.

Bigtable

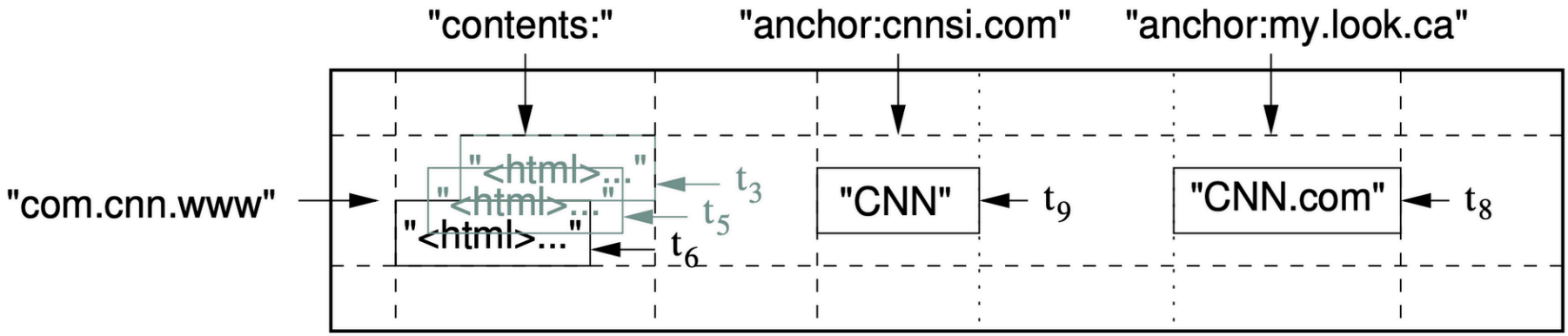
Son celdas, de tipo string, que pueden ser referenciadas por un nombre de fila y un nombre de columna, de tipo string también.

Es un mapa multi-dimensional persistente y distribuido dado por el par llave/valor (fila:string, columna:string, tipo:int64): string.

Bigtable: caso de uso (origen)

- Con el diseño *Bigtable* en páginas web, se buscaba:

- Indexar por URL de las páginas, lo que se convertiría en el identificador de la fila.
- Información sobre la página, almacenada en diferentes columnas, dependiendo del tipo de dato y uso.
- Contenido de la página con versiones, reflejando el punto en el tiempo cuando se obtuvieron.



Bigtable: metadatos

FILAS:

- Tienen un tamaño máximo de 64KB, uso común es entre 10-100 bytes.
- Las filas son almacenadas agrupadas, basado en orden lexicográfico

COLUMNAS

- Las columnas se agrupan en familias.
- En ellas se da el control de acceso, contabilidad de memoria y compresión de datos.
- La cantidad de columnas de cada familia pueda ser muy amplia.
- El nombre de la familia es lo primero que aparece en el nombre de la columna. El nombre de una columna siempre será *familia:calificador*

Bigtable: metadatos

- VERSIONES:
  - Las versiones son manejadas no por consecutivos sino por tiempo.
  - Versiones más recientes son las leídas primero.
  - Se puede activar recolección de basura automática para las últimas N versiones o con respecto a una fecha (e.g. cantidad de días atrás)

Bigtable: almacenamiento

- Reside sobre GFS y sobre un formato de archivo *SSTable*.
- Cada *SSTable* es un mapa inmutable ordenado de pares llave/valor.
- Cada tabla posee bloques (64Kb) con sus índices que permiten ser ubicados en regiones de la *SSTable*.

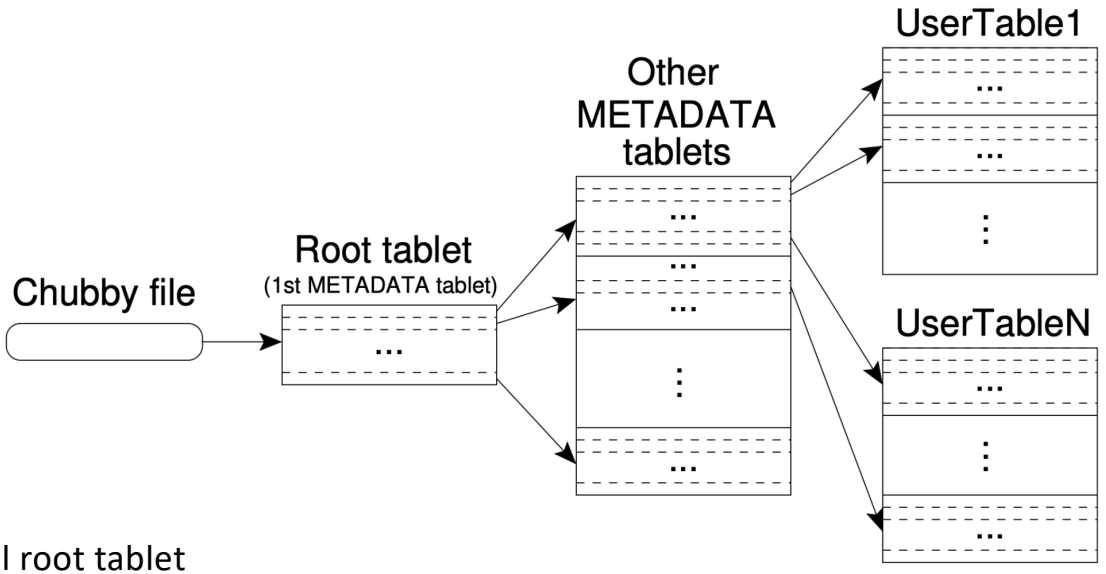
Bigtable: servidores

- Bigtable tiene servidores coordinadores, llamados *maestros*, y los servidores con los datos propiamente, llamados *tablet servers*.
- Cada servidor maneja entre 10 y 1000 *tablets* con información y tiene la responsabilidad de particionar cuando el tamaño lo amerita (cada 100-200MB por defecto).
- El maestro se encarga de asignar *tablets* a cada *tablet server*, además de balancear la carga, recolección de basura, manejar la estructura de familias de columnas y administrar la creación de *tablet servers* adicionales.

Bigtable: servidores

- La comunicación de los clientes con el maestro no es muy frecuente ya que, al igual que en GFS, los clientes se comunican directo con los servidores de tablets.
- Cada cluster de Bigtable puede almacenar múltiples *Bigtables*, cada una compuesta de múltiples tablets.

Bigtable: Jerarquía de localización de tablas

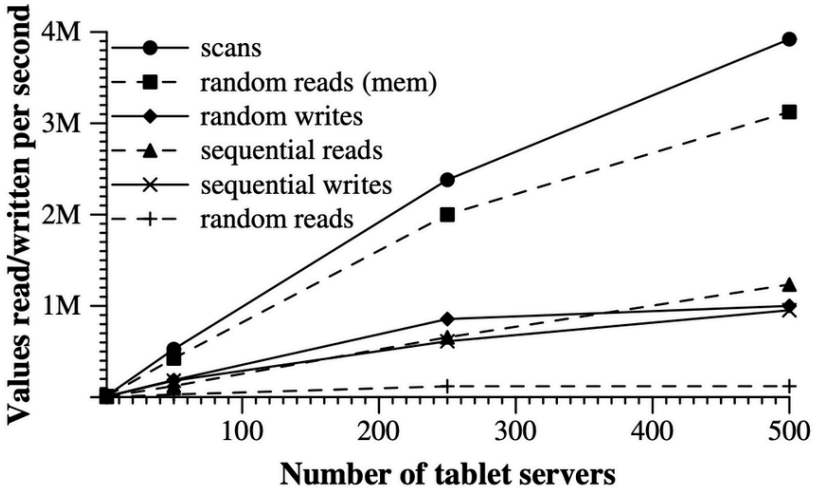


Chubby: localización del root tablet
Root table: localización del todas las tablets (nunca se parte)
OTHER: localización de UserTable

Bigtable: rendimiento

- **Escenario**: se diseñaron diferentes benchmarks de prueba, se utilizaron N tablet servers, con 1 GB de memoria, que escribían a una celda GFS con 1786 máquinas. Se tenían N clientes generando carga de lectura sobre Bigtable. Se define R como el número de filas en cada prueba, seleccionada para que cada una escribiera alrededor de 1GB de datos por servidor de tablets.

Experiment	# of Tablet Servers			
	1	50	250	500
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843



Bigtable: rendimiento

- **Detalles de cada benchmark**
- Sequential write: Nombre de filas entre 0 y R−1 particionado en $10N$ rangos del mismo tamaño. Cada rango fue asignado por un calendarizador a los clientes conforme iban terminando cada uno. En cada fila se escribía un string aleatorio.
- Random write: Similar al anterior pero la fila no era el número secuencial sino que se le aplicaba el módulo a una función de hash, para tratar de simular una distribución uniforme.
- Sequential read: Igual que senquential write pero en lugar de escribir leía los contenidos.

Bigtable: rendimiento

- **Detalles de cada benchmark**
- Random read: Igual que random write, pero de lectura.
- Scan: Utilizaba las funcionalidades para obtener múltiples rangos al mismo tiempo, basado en nombre de la fila. Esto permitía obtener más datos con una sola llamada.
- Random reads (mem): Similar a random read pero se usó una funcionalidad de la familia para marcarla "en memoria" que debería maximizar el uso de la memoria de los tablets servers, versus tener que obtener los datos de disco (en esta época era *spinning disks* en su mayoría, no de estado sólido).

Bigtable: casos de uso

Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
Crawl	800	11%	1000	16	8	0%	No
Crawl	50	33%	200	2	2	0%	No
Google Analytics	20	29%	10	1	1	0%	Yes
Google Analytics	200	14%	80	1	1	0%	Yes
Google Base	2	31%	10	29	3	15%	Yes
Google Earth	0.5	64%	8	7	2	33%	Yes
Google Earth	70	–	9	8	3	0%	No
Orkut	9	–	0.9	8	5	1%	Yes
Personalized Search	4	47%	6	93	11	5%	Yes

Ghemawat, S; Gobioff, H; Leung, S. The Google File System.
https://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf

Chang, F; Dean, J; Ghemawat, S; Hsieh, W; Wallach, D; Burrows, M; Chandra, T; Fikes, A; Gruber, R.
Bigtable: A Distributed Storage System for Structured Data.
https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf

TAREA # 2

1. HDFS
2. PrestoDB
3. Cassandra
4. HBase
5. Accumulo
6. DynamoDB
7. ?
8. ?