Imperial College London

LSDS
Large-Scale Data & Systems Group

Bigtable: A Distributed Storage System for Structured Data

Peter Pietzuch
prp@doc.ic.ac.uk

Department of Computing
Imperial College London
http://lsds.doc.ic.ac.uk

Based on slides by E. Paulson, University of Washington

Autumn 2019

Google Scale

Lots of data

- Copies of web, satellite data, user data, email and news, Subversion backing store

Many incoming requests

No commercial system big enough

- Could not afford it, even if there was one
- Might not have made appropriate design choices

Building Blocks

Scheduler (Google WorkQueue)

Google File System

Chubby Lock Service

Two other pieces helpful but not required
- Sawzall
- MapReduce

Bigtable: Build more application-friendly storage service using these parts

Google File System

Large-scale distributed file system

Master: responsible for metadata

Chunk servers: responsible for reading and writing large chunks of data

Chunks replicated on 3 machines, master responsible for ensuring replicas exist

→ USENIX OSDI'04 paper

Chubby

{lock/file/name} service

Coarse-grained locks, can store small amount of data in a lock

5 replicas, need a majority vote to be active

→ USENIX OSDI'06 paper

Data Model: A Big Map

<Row, Column, Timestamp> triple for key

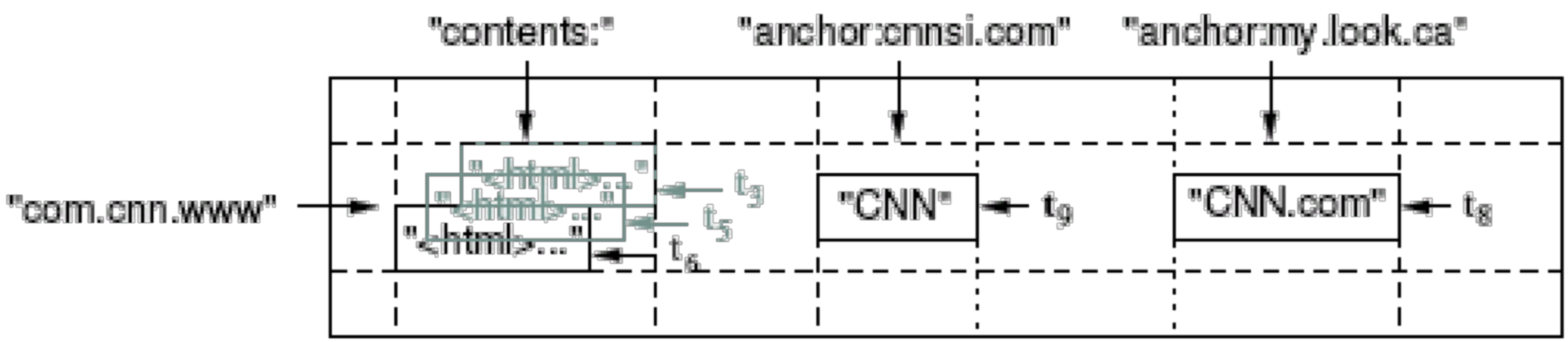
- Each value is uninterpreted array of bytes

Arbitrary “columns” on a row-by-row basis

- Column family:qualifier
- Family is heavyweight, qualifier lightweight
- Column-oriented physical store -- rows are sparse!

Lookup, insert, delete API

- Each read or write of data under a single row key is atomic



Bigtable vs. Relational DB

- No table-wide integrity constraints
- No multi-row transactions
- Uninterpreted values: No aggregation over data
- Immutable data similar to versioning DBs
  - Can specify: keep last *N* versions or last *N* days
- C++ functions, not SQL (no complex queries)
- Clients indicate what data to cache in memory
- Data stored lexicographically sorted
  - Clients control locality by naming of rows & columns

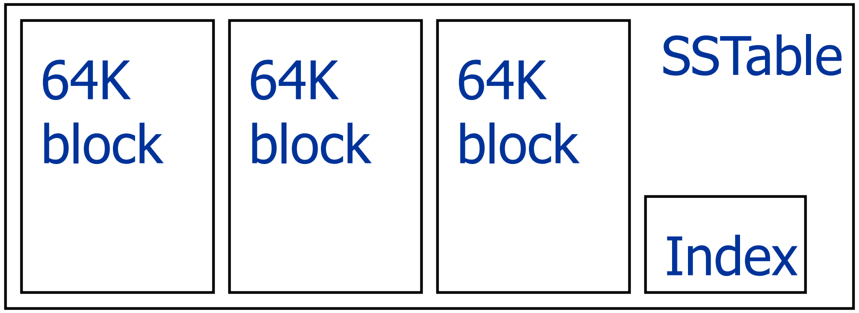
SSTable

Immutable, sorted file of key-value pairs

Chunks of data + index

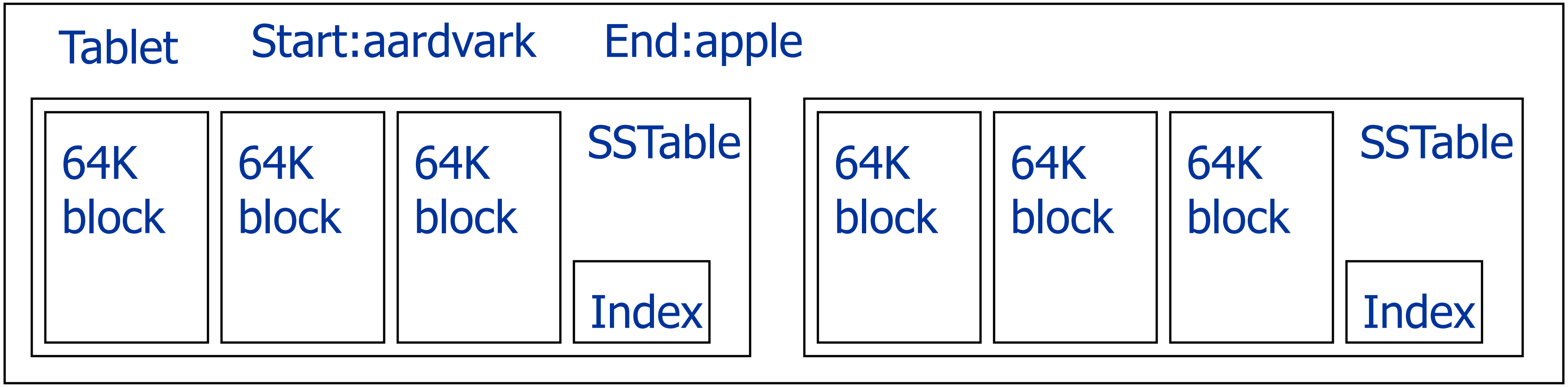
- Index is of block ranges, not values
- Index loaded into memory when SSTable is opened
- Lookup is single disk seek

Alternatively, client can load SSTable into memory



Tablet

- Contains some range of rows of table
- Unit of distribution & load balancing
- Built out of multiple SSTables

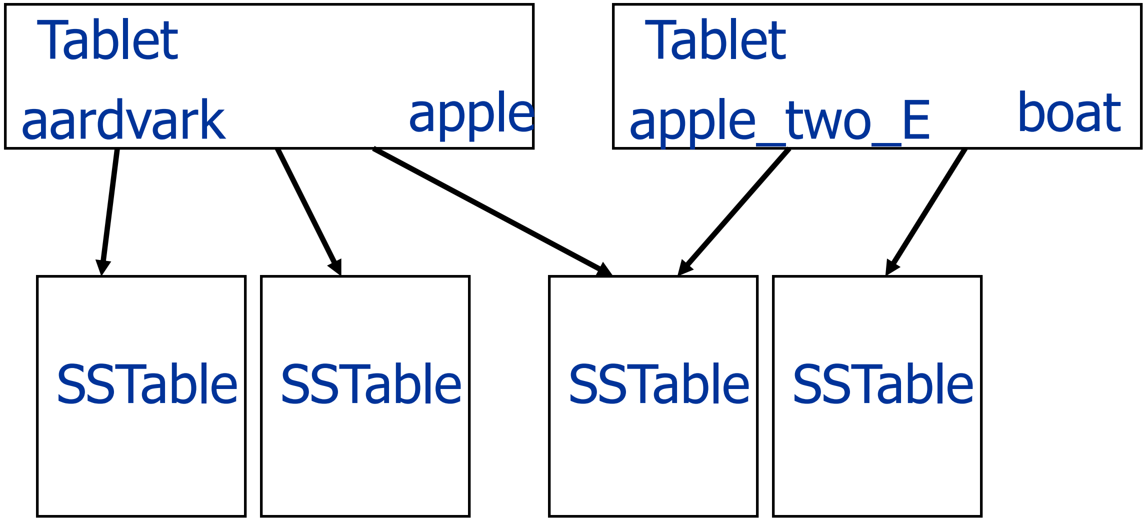


Table

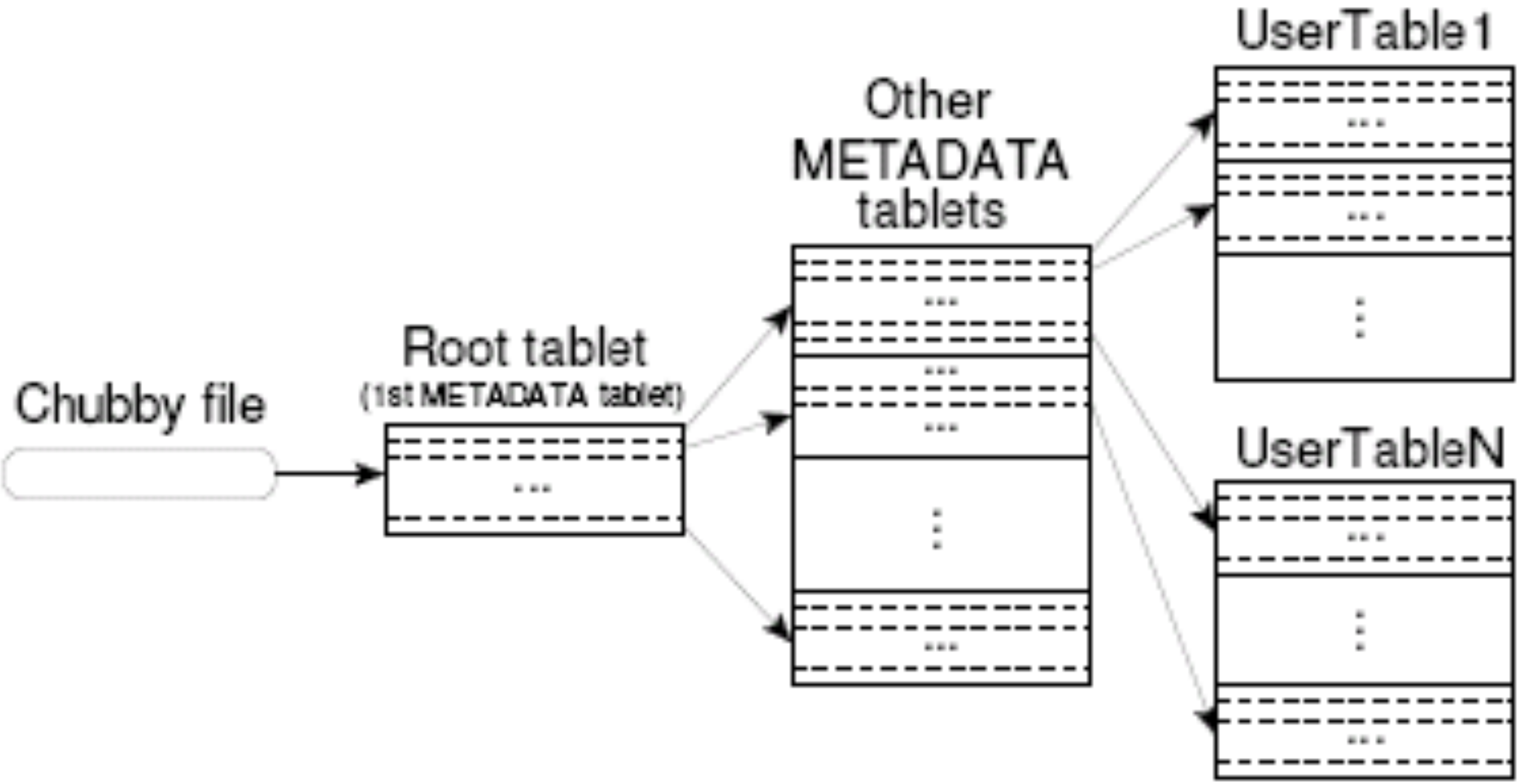
Multiple tablets make up table

SSTables can be shared

Tablets do not overlap, SSTables can overlap



Finding a Tablet



Client library caches tablet locations
Metadata table includes log of all events pertaining to each tablet

Servers

Tablet servers manage tablets, multiple tablets per server

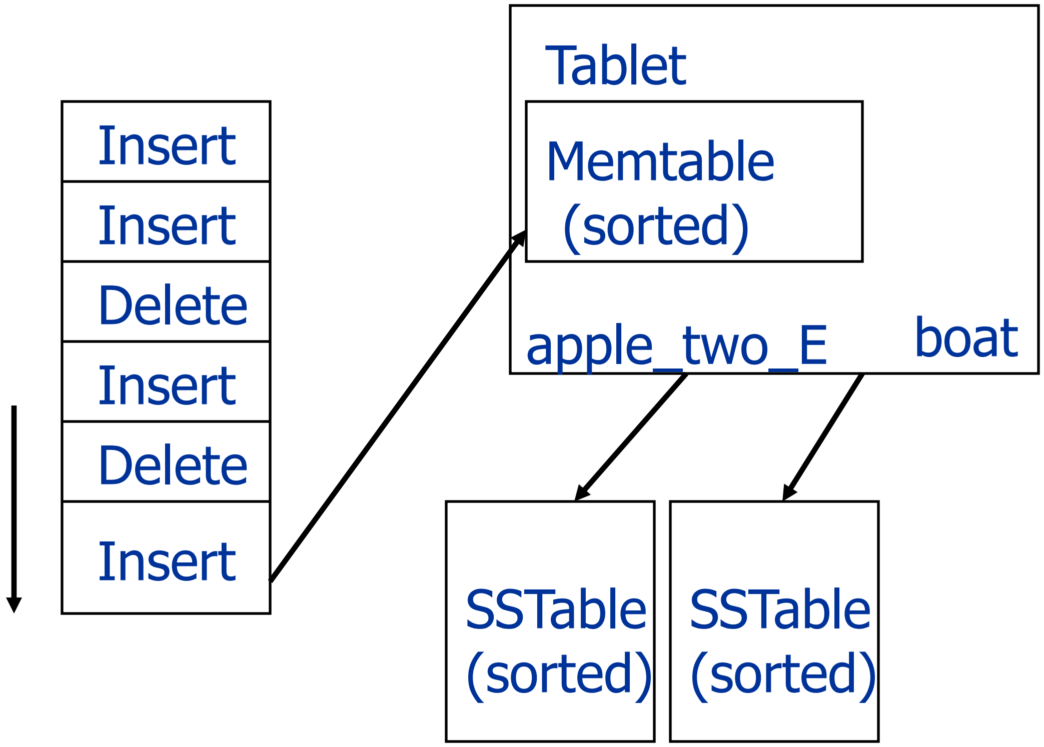
- Each tablet is 100-200 MBs
- Each tablet lives at only one server
- Tablet server splits tablets that get too big

Master responsible for load balancing and fault tolerance

- Use Chubby to monitor health of tablet servers, restart failed servers
- GFS replicates data
- Prefer to start tablet server on same machine that the data is already at

Editing/Reading a Table

- Mutations committed to commit log (in GFS)
  - Then applied to in-memory version (memtable)
  - For concurrency, each memtable row is copy-on-write
- Reads applied to merged view of SSTables & memtable
  - Reads & writes continue during tablet split or merge



Compactions

Minor compaction: convert full memtable into an SSTable, and start new memtable

- Reduce memory usage
- Reduce log traffic on restart

Merging compaction

- Reduce number of SSTables
- Good place to apply policy “keep only N versions”

Major compaction

- Merging compaction that results in only one SSTable
- No deletion records, only live data

Locality Groups

Group column families together into SSTable

- Avoid mingling data, ie page contents and page metadata
- Can keep some locality groups in memory

Can compress locality groups (10:1 typical)

- Popular compression scheme: (1) long common strings across a large window, (2) standard compression across small 16KB windows

Bloom Filters on locality groups

- Avoids searching SSTable

Microbenchmarks: Throughput

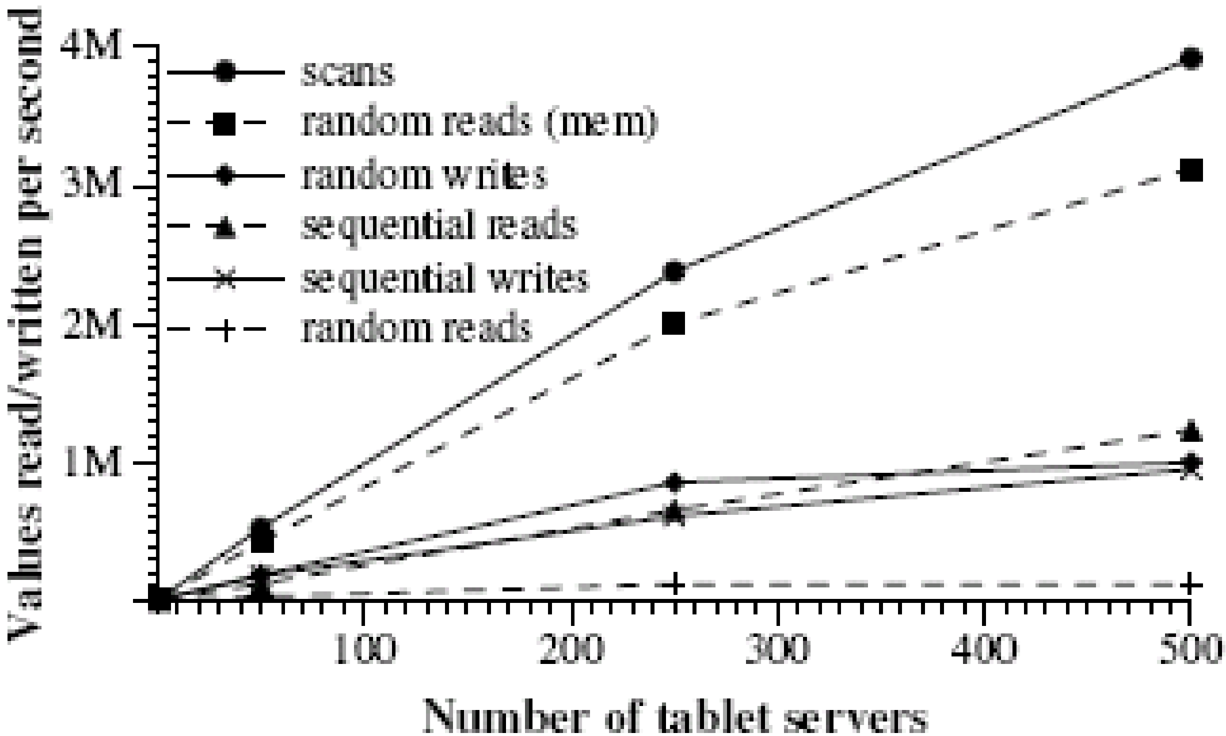
1786 machines, with two dual-core Operton 2 GHz chips, large physical mem, two 400 GB IDE hard drives each, Gb Ethernet LAN

Tablet server: 1 GB mem, # clients = # servers

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

1000-byte values per server per second

Aggregate rate



Application at Google

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

Lessons Learned

Only implement some of the requirements, since the last is probably not needed

Many types of failure possible

Big systems need proper systems-level monitoring

Value simple designs