Lecture 20 -- 11/19/2014

BigTable

big picture
  parallel db (one data center)
  mix of OLTP and batch analysis
  lots of data, high r/w rates, 1000s of cheap boxes thus many failures

what does paper say Google uses BigTable for?
  analyzing big web crawls
  analyzing click records to optimize ads
  some on-line uses: orkut, personalized search
  google earth

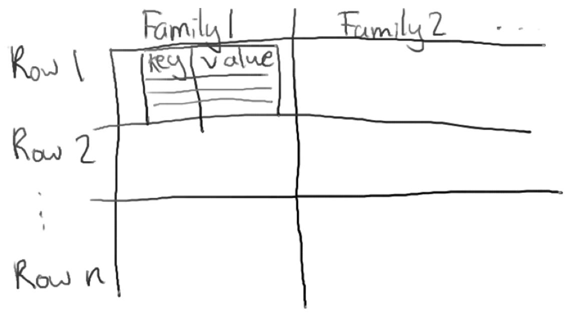
BigTable

What is the basic idea?

4D table

Row, column family, column name, time -> value

Diagram:



query model
  single-row fetch by row/column key
  single-row atomic update and read-modify-write
  scans in key order
  no joins
  no aggregates (but they have MapReduce for this)

example use: Figure 1, web crawl for various analyses

- one row per URL (== page)
- one column for each link *to* a page! that's a lot of columns.
- how to store row/col/time in a file?
  - the model may be 3d, but underlying storage has only one dimension

guess flattened layout for figure 1?

```
com.cnet.www
com.cnn.www
  anchor:cnnsi.com
    t9: CNN
  anchor:my.look.ca
    t8: CNN.com
  ...
  content
    t6: ...
    t5: ...
    t3: ...
com.cnx.www
```

- very different kind of "column" than an SQL db
  - different rows have different columns!

- like a mini-b-tree table in every row
  - a hierarchical data model
  - or like one big btree
  - keys are rowname+family+colname+time

- so it's cheap to scan all the links to a certain page
  - but *not* cheap to scan all content inserted at t5
  - i.e. BigTable is not a three-dimensional DB

how would we store Figure 1 in a relational DB?

```
anchor(site, from, time, text)
content(site, time, html)
```

does it make any difference?

- usual argument against hierarchy is repeated data -- doesn't really apply here
- we'd want to cluster tables for good scan performance
- relational model lets us ask for all anchors by from column
  - while BigTable really only lets you scan by site
- but implementing scan-by-from requires indices, would be slowish

(of course, can use clustering, replication, mat. views, etc. in an RDBMS)

"Locality group" mechanism puts some column families in separate file
so you could scan all pages' anchors w/o having to scan+ignore content
much like c-store

How is query model different from a relational database:

- Simplified query processing language -- e.g., no joins, or aggregates (SawzAll)

- Relaxed transaction semantics
    - Only atomic writes to a single row

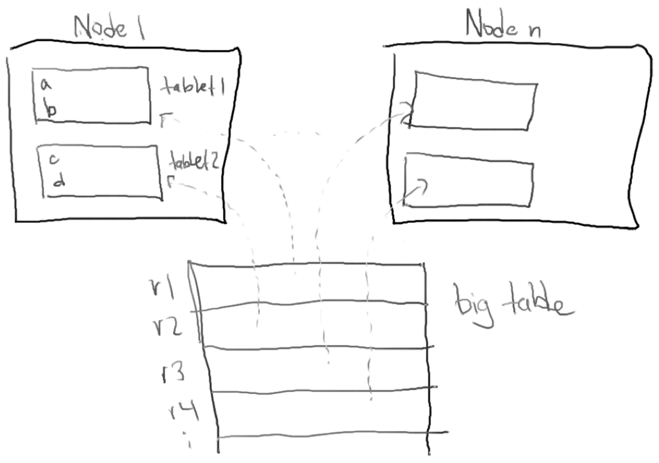
<u>Architecture:</u>

Table fragmented into a number of horizontal partitions -- groups of rows -- just like in a DBMS. "Tablets" Each tablet is just a file in the GFS file system. <u>What's GFS?</u>

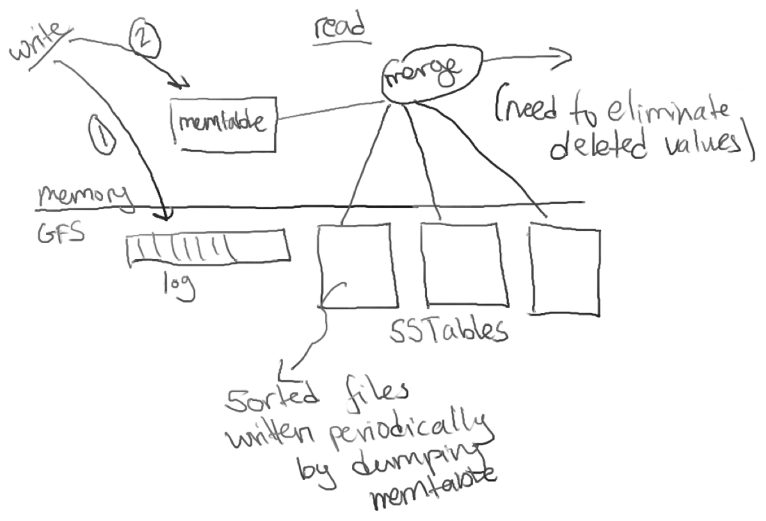
Logically the key space is fully partitioned across all nodes -- each key has a unique owner in Big Table.

Tablets are range partitioned. <u>What are the tradeoffs?</u>

Each physical node is responsible for some number of tablets.



As in a DBMS, each tablet is actually represented as a collection of in memory and on disk structures.


write
2
1
read
merge
(need to eliminate deleted values)
memtable
memory
GFS
log
SSTables
Sorted files written periodically by dumping memtable

Query processing:

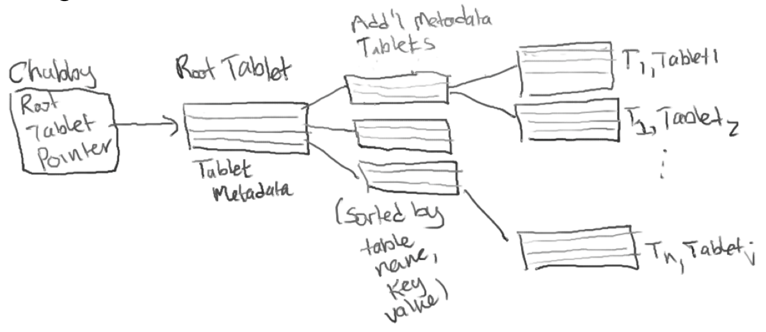
1) Finding the tablets that may be needed for a particular query

"Lock service" -- Chubby -- stores the location of a metadata block..

Lock service is basically a replicated state machine where all nodes agree on the value of a set of variables. Organized as directories of key-value pairs where each pair can be locked. Uses a consensus protocol to ensure majority of nodes agree on each write/read.

Lookup involves following a hierarchy of pointers through metadata to find the tablets that contain ranges needed for a query.

Diagram:



2) Scanning a tablet to answer a query

Need to merge data structures together from memtable and SS tables

or

3) Updating a tablet

Use WAL to update log, then insert in memtable

WAL is on GFS (so is replicated, etc.)

<u>**Maintenance**</u>

As inserts happen, memtable will grow. Periodically write out as an SSTable.

More SSTables increase performance overhead. Periodically merge.

Lots of updates/deletes lead to lots of unused records in SSTables that have to be removed at merge time. Periodically reclaim by performing a "major compaction"

<u>what properties of Chubby are important?</u>
  <u>why a master AND chubby?</u>
  most systems integrate them; separation means chubby can be reused
  chubby is a generic fault-tolerant file and lock server

  chubby does three things for BigTable
1. stores root of METADATA table in a file
2. maintains master lock, so there's at most one master
3. tracks which tablet servers are alive (via locks)

  <u>key properties:</u>
    Chubby replicates METADATA and locks
    Chubby keeps going even if one (two?) Chubby servers down
    Chubby won't disagree with itself
      example: network partition
      you update Chubby replica in one partition
      Chubby replica in other partition will *not* show stale data

<u>what is the point of the master?</u>
  after all, the METADATA is all in Chubby and GFS
  answer: there had better be only one entity assigning tablets to servers
    only the master writes METADATA
  chubby locking ensures there's at most one master
    even during network partitions

<u>why isn't Chubby a bottleneck?</u>
  clients cache METADATA
  METADATA doesn't change often
  tablet server will tell client if it is talking to wrong server

<u>Fault tolerance:</u>

<u>What if a tablet server fails?</u>

Master node detects the failure (when it fails to renew its locks with Chubby).

It assigns the tablets that server was responsible for to another tablet server.

(This works because tablets are stored & replicated on GFS.)

Have to perform a log-based recovery when tablets are re-assigned, because of memory resident part of tablet in failed tablet server.

Other clients may cache metadata pointing to failed node as owner of a tablet, but that's OK because they will re-load metadata when they can't find owner/owner says it doesn't have tablet.

<u>What if the master fails?</u>

Don't discuss this too much, but a new master server can use Chubby to figure out who the tablet servers are. Can only be one master, so if nodes don't hear from master for awhile they can attempt to promote themselves.

Can then check with each tablet server to determine what tablets it owns.

Can then scan the entire metadata table to figure out what tablets exist.

Finally, can give unassigned tablets to some of the tablet servers.

<u>What if Chubby fails?</u>

Shouldn't happen much -- it's a 5 node cluster, 3 of which would have to simultaneously fail for the state to be unavailable.

If it fails the system is unavailable.

<u>What if GFS fails?</u>

Same story as MapReduce -- all data in GFS is replicated in several places and rebalanced when a node fails, so the hope is that data should never be lost.

GFS is a powerful abstraction that makes it much easier to tolerate node failures and maintain availability.

Basically manages the spreading of data across a cluster of nodes to ensure it is replicated.

Evaluation

setup

- 1700 GFS servers, N tablet servers, N clients
- all using same set of machines
- two-level LAN with gig-e
- each row has 1000 bytes

single-tablet-server random read

- first row, first column of Figure 6
- single client reads random rows
- how can one server do 1212 random reads/second?
  - you can't seek 1212 times per second!
  - answer: only 1 GB of data, split up over maybe 16 GFS servers
  - so all the data is in the GFS Linux kernel file cache
- so why only 1212, if in memory?
  - that's only 1 megabyte/second!
  - each row read reads 64KB from GFS
  - 78 MB / second, about all gig-e or TCP can do

single-tablet-server random write

- single client reads random rows
- traditionally a hard workload
- how could it write 8850 per second?
  - each write must go to disk (the log, on GFS) for durability
  - log is probably in one GFS chunk (one triple of servers)
  - you cannot seek or rotate 8850 times per second!
  - presumably batching many log file writes, group commit
    - does that means BigTable says "yes" to client before data is durable?

what about scaling

- read across a row in Figure 6
- the per-server numbers go down
- so performance goes up w/ # tablet servers, but not linearly
- why not linear?
  - paper says load imbalance:
    - some BigTable servers have other stuff running on them
    - master doesn't hand out tablets 100% balanced
  - also network bottleneck, at least for random read
    - remember 64K xfer over LAN per 1000-byte row read
    - root of net only has about 100 gbit/second total

enough to keep only about 100 tablet servers busy

<u>So why didn't they use a distributed database?</u>

Again, somewhat unclear. Probably didn't feel that distributed databases could be made to work on their existing cluster based infrastructure very easily -- e..g, they had GFS and Chubby already, and they had already headed down the path of rolling everything themselves.

This paper makes you realize how complicated building a DDBMS is -- and it doesn't even provide for multirow transactions or complex query processing!

Multirow transactions require 2PC; complicated in the face of failures. Distributed query optimization requires investigating different strategies -- e.g., shipping table fragments for joins, etc.