6.830 2009 Lecture 19: 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

data model
  figure 1 shows a table has three (four?) dimensions
    row, column family, column, time

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
"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

example use: section 8.1 Google Analytics
record and analyze user actions on web sites
are people clicking on your ads?
which ads are the most effective?
row per user session, key is <site,starttime>
column per click???
periodic batch analysis of each site's recent sessions/clicks
data must arrive at a huge rate!
worse, it arrives in the wrong order
arrives sorted by time
but we want to store and scan by site
this is a pretty classic problem
bad solution: insert each new click into a b-tree
we'll see later how they deal with this

how do they implement BigTable?

not an ordinary parallel DB
partition data over the servers and their disks
each server does reads/writes for data on its disk
this is not how BigTable works!

starting point: GFS
GFS a cluster file system
FS model: directories, files, names, open/read/write
100s of Linux chunk servers with disks
store 64MB chunks (an ordinary Linux file for each chunk)
each chunk replicated on three servers
GFS master server knows directory hierarchy
for dir, what files are in it
for file, knows chunk servers for each 64 MB
master has private recoverable DB for metadata
primary/backup to a slave
client read:
send file name and offset to master
master replies with set of servers that have that chunk
ask nearest chunk server
client write:
ask master where to store
maybe master chooses a new set of chunk servers if crossing 64 MB
one chunk server is primary
it chooses order of updates and forwards to two backups
what works well in GFS?
huge sequential reads and writes
appends
huge throughput (3 copies, striping)
fault tolerance of data (3 copies)

what works badly in GFS?
fault-tolerance of master
small files (master a bottleneck)
concurrent updates to same file from many clients (except appends)

so GFS maybe already solves some problems for BigTable
giant storage
data fault-tolerance
high sequential throughput

BigTable acts as a set of clients to GFS
BigTable servers r/w GFS across the net, no local storage
data not really tied permanently to particular BigTable servers
if one (or all) BigTable servers have permanent failures
you don't lose data -- data is in GFS
just fire up replacement BigTable servers, they read GFS
this simplifies the BigTable design

It splits each table into lots of tablets
partition by row name
each tablet is stored in a set of GFS files

given a table name and row name, how to find tablet?
1. tablet server needs to know what GFS files hold the tablet data
METADATA of Figure 4
Chubby is a mini file server that says what
GFS files hold the METADATA table
so BigTable knows where to start
2. client needs to know what tablet server serves the tablet
(not the same as question #1, can be soft state)
my guess: METADATA holds this too
client doesn't ask the master (4th para of Section 5)
but paper's only mention of tablet -> server mapping is in master mem
e.g. booting master doesn't read this info from METADATA
but by talking to all live tablet servers

so what does a METADATA entry contain?
<table ID, starting row name> ->
names of GFS files that store the tablet (sec 5.3)
what tablet server serves it (guessing, paper doesn't say)

what properties of Chubby are important?
why a master AND chubby?
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
stores root of METADATA table in a file
maintains master lock, so there's at most one master
tracks which tablet servers are alive (via locks)
key properties:
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

what is the point of the master?
  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

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

read/write processing inside a tablet server
  similar to c-store
  log for fast writes, SSTables for fast lookups
  [diagram: log in GFS, memtable, SSTables in GFS]
  SStables in GFS
    compact ordered row/family/col/time data
    compressed
    index at the end
    immutable -- why not mutable b+tree?
      fast search, compact, compression, GFS not good at rand write
  log in GFS
  compaction

recovery from tablet server crashes
  key problem:
    what if it was in the middle of some update when it crashed?
    do we need to wait for it to reboot and recover from its log?
  chubby notices server is dead (stops refreshing its lock)
    and/or master notices it is dead?
  even if tablet server is live but partitioned,
    it won't be able to refresh its lock if Chubby thinks it is dead
    so table server will know to stop serving
  if master sees tablet server no longer has its lock:
    picks another tablet server (preferably lightly loaded one)
    tells is "load that tablet from GFS"
  new tablet server reads the crashed server's log from GFS!

recovery from BigTable master crashes
  Chubby takes away its lock
  some other machine(s) decide to be master
    only one gets the Chubby lock
  recreate old master's state:
    read set of tablets from METADATA
    ask Chubby for list of live tablet servers
    ask tablet servers what they serve

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

i like this paper's evaluation section
  shows good and bad aspects
  explains reasons for results
  connects performance back to design

6.830 / 6.814 Database Systems

Fall 2010