6.830 2010 Lecture 15: C-Store (Sam Madden)

Why are we reading this paper?
C-store has standard interface, but very different design
   Help us understand what choices standard DBs made
   Think about different set of apps than OLTP

Paper status
Most individual techniques already existed
C-Store pulls them together
Not just a limited special-purpose DB
   transparent -- sql interface
   read/write, not just r/o
   consistency
   transactional update
Paper doesn't describe complete system
   design + partial implementation
Commercialized as Vertica

What's a data warehouse?
big historical collection of data
companies analyze to spot trends &c
what products are in? what's out? where is cherry coke popular?
   spot early, order more
mostly r/o but must be updated, maybe continuously

Example: big chain of stores, e.g. Walmart
   each store records every sale
   upload to central DB at headquarters
   keep the last few years

Typical schema (logical):
   time(tid,year,mon,day,hour)  product(pid,type,color,supplier)
      xact(tid,pid,sid,cid,price,discount,tax,coupon,&c)
      store(sid,state,mgr,size)  customer(cid,city,zip)

called a "star schema"
"fact table" in the middle
   gigantic # of rows
   might have 100s of columns
"dimension tables" typically much smaller

How big is the data?
   50K products (5 MB)
   3K stores (1 MB)
   5M customers (5 GB)
   150K times (5 MB) (10 minute granularity)
   350B xact rows (35 TB) (100 bytes/sale)
      3000 stores * 10 registers * 20 items/min * 2 years

example 1:
   total sales by store in Texas on Mondays
   join xact to time and store
   filter by day and state
   group by sid, aggregate

example 2:
   average daily sales for Nikon cameras
   join xact to product, time
   filter by supplier
   group by day, aggregate
How long would queries take on traditional DB?
probably have to look at every page of fact table
  even if only 1% of records pass filter
  means every block might have one relevant record
  so index into fact table may not be very useful
joins to dimension tables pretty cheap
  they fit in memory, fast hash lookups
how long to read the whole fact table?
  35 TB, say 100 disks, 50 MB/sec/disk => 2 hours
  ouch!
You can imagine building special setups
  e.g. maintain aggregates in real time -- pre-compute
  know all the queries in advance
  update aggregate answers as new data arrives
  table of daily sales of Nikon cameras, &c
  but then hard to run "ad-hoc" queries

C-Store

Why columns?
  Why store each column separately?
  avoid reading bytes from fact table you don't need

Why "projections" of columns?
  you usually want more than one column
  e.g. sid and price for example 1

Why is a projection sorted on one of the columns?
  to help aggregation: bring all data for a given store together
  or to help filtering by bringing all data w/ given col value together
  so you only have to read an interval of the column

What projection would help example 1?
  columns: sid, price, store.state, time.day
  note we are including columns from multiple logical tables
  note we are duplicating a lot of data e.g. store.state
  note projection must have every column you need -- can't consult "original" row
  thus you don't need a notion of tupleID
  note i'th row in each column comes from same xact row
  order?
  sid
  state, sid

Why multiple overlapping projections?
  why store the same column multiple times?

What projection would help example 2?
  columns: price, time.year, time.mon, time.day, product.supplier
  note we are not including join columns! e.g. pid
  order?
  supplier, year, mon, day
  year, mon, day, supplier

What if there isn't an appropriate projection for your query?
  You lose -> wait 2 hours
  Ask DB administrator to add a projection

Could we get the same effect in conventional DB?
  Keep heap files sorted ("clustered")?
  can only do it one way
  B+Trees for order and filtering?
have to avoid seeks into main heap file, so multi-key B+trees
copy data into many tables, one per projection
So yes, we could
But very manual
  choose right table for each query
  updating?
"materialized views" partially automates this for conventional DB
  and Eval in Section 9 shows they make row store perform 10x better
  but c-store still faster

Won't all this burn up huge quantities of disk space?

How do they compress?

Why does self-order vs foreign-order matter in Section 3.1?

How to compress for our example projections?
  sid ordered by sid?
  price ordered by sid?
  store.state ordered by sid?
  time.day ordered by sid?

Won't it be slow to update if there are lots of copies?

How does C-Store update efficiently?

How does C-Store run consistent r/o queries despite updates?

Why segment across a cluster of servers?
  Parallel speedup
    many disks, more memory, many CPUs

How do they ensure good parallel speedup on a cluster?
  What is a "horizontal partition"?
  Why will that lead to good parallel speedup?
  Sorting allows filtering and aggregating to proceed in parallel
  will talk about parallel DBs more later

Evaluation? Section 9
  what are the main claims that need to be substantiated?
    faster on data warehouse queries than a traditional row store
    uses a reasonable amount of space

Experimental setup
  standard data-warehouse benchmark "TPC-H"
  single machine
  one disk
  2 GB RAM
    this is a little odd -- original data also 2 GB
    small reduction in memory requirement could give a huge boost in this setup
    but make no difference for larger data sets

TPC-H scale 10
  standard data warehouse benchmark
  comes in different sizes ("scale")
    defines how many rows in each table
customer: 1.5 M rows, abt 15 MB
orders: 15 M rows, abt 150 MB
lineitem: 60 M rows, abt 2.4 GB

results are spectacular!
  mostly > 100x faster than row store
Q4 is 400x faster on c-store -- why?
print o_orderdate, l_shipdate
  group by o_orderdate
  filter on l_orderkey, o_orderkey, o_orderdate
must be using D2: o_orderdate, l_shipdate, l_suppkey | o_orderdate, l_suppkey
  D2 is missing o_orderkey and l_orderkey -- do we need them?
D2 already in good order to aggregate by o_orderdate
how much data is c-store scanning?
  two columns with 60 M rows
  o_orderdate probably compressed down to a bit or byte
  l_shipdate might be 4 bytes
  so 300 MB?
  read from disk in 6 seconds
  read from RAM in 0.3 seconds
  actual performance is in between: 2 seconds
  may be skipping due to o_orderdate > D? maybe some in mem, some in disk?
what is row DB probably doing? for 723 seconds
  would have to scan 2 GB LINEITEM table
  if doesn't fit in RAM, 40 seconds at 50 MB/sec from disk
  must join to ORDERS table, fits in memory, should be fast hash
  then sort (or something) by o_orderdate
  hard to guess why row DB takes 723 rather than 40+ seconds

Q2 is only 3x faster w/ c-store
  needs l_suppkey, l_shipdate
  filter by l_shipdate
  group by l_suppkey
  probably uses D1: l* | l_shipdate, l_suppkey
  D1 lets c-store only look at l_shipdate = D, needn't scan most of LINEITEM
  D1 sorted well for aggregation
  what would row DB do?
    maybe has a b+tree also keyed by l_shipdate, l_suppkey?
    does not need to scan or seek into LINEITEM

They win by keeping multiple copies, tailored to different queries
How much storage penalty for queries in Eval?
  Actually LESS storage! 2 GB vs 4.5 GB
Uncompressed data was also about 2 GB
Would be more for more queries