550: Data Parallelism

Mapreduce: data parallelism

Idea is to make it possible for mortals to write parallel programs by having them write programs in a sequential language that exposes parallelism. High level language – allows for compiler optimization (and makes scheduler activations and similar approaches even more important).

A similar model: bulk synchronous programming (BSP). Each CPU progresses through a set of barriers: computes on local state + global state of previous barrier. Behavior is completely deterministic, so easy to debug.

A slightly more complex model: Dryad (Microsoft's version of MapReduce). Instead of just two operators and two stages, can have a sequence of stages and arbitrary operators (BSP). Each stage is split across a cluster, and uses the output of the previous stage, but in an arbitrary fashion.

Data parallelism didn't start with google and microsoft.

Story starts back with SQL. Issues in the mapreduce paper: simple algorithm, very complex infrastructure to deal with (i) failures (ii) parallelism (iii) data movement (to/from disk, but ultimately, between machines), all present in SQL (ideas date from 1973, roughly the same time as UNIX and the Internet).

Structured data (tables), with a few data operators:

1) Select (where) (which rows of the table are relevant)
2) project (group by) (rows with common values, to smaller set of rows)
3) match (having) (rows with a particular value)
4) join (cross-product of two tables – can simulate with a mapreduce stage)
5) sort (accumulate records with the same key)

Data can be anything that can be stored in tables – airline reservations, or say, everything ever purchased at walmart. One step is just to make sure books balance, but another is decision support: what items are bought at the same time

Key idea: write data manipulation in terms of these operators, and then compiler converts program into data movement operations

For example, if you have a table that's too big to fit in memory, it's the database's job, and not the programmer's to stage it into memory to do the operations. And to handle failures, etc.

Of course, key question is: can you make it efficient enough? Seems painfully slow/inefficient. After all, you are creating a new table each time. But compiler can, say, optimize by reordering the data flow to put decimating operations first.

Aside: why didn't google use SQL? After all, databases were supposed to be the experts at data manipulation, and SQL looks like a superset of mapreduce

Tried! And it was really really slow (and really really expensive!). Of the major web sites, only Amazon built their system around COTS databases (and that was a problem for them). Seems odd – how could databases have missed the biggest explosion in data in decades?

Christensen: upstart technologies appear unimportant at first. But if they improve more rapidly, then they are a long term threat.

For databases, longstanding problem was dealing with arbitrary/non-table data. E.g., files, or images, or …

A bunch of attempts by the research community to address this, no real commercial traction. So google needed to reinvent the wheel (or perhaps, invent a new wheel).

Characteristics of this kind of disruption: very profitable biz for incumbent; very ignored (low margin) market for upstart; more rapid rate of improvement over time for upstart

Draw picture of time vs. capability, but we can label each exponential curve.
Relational dbms vs. mapreduce

Relabel curves:
Hierarchical databases vs. relational
Relational vs. mapreduce

But others:
Lotus vs. excel
Excel vs. google docs

Multics vs. unix

Phone network vs. internet
Internet vs. p2p? wifi?

Back to databases:

What did we give up as part of mapreduce? (DeWitt flame, on blog)

"mapreduce provides only a sliver of functionality found in modern DBMS's"

A. Mapreduce is major step backwards in terms of programming paradigm
0) schemas are good (structured data)
1) separation of schema from application is good (structure of data is implicit)
2) high level access language is good (hide details)

B. Sub-optimal implementation
no indexing (google has a separate system, called BigTable, which does support indexed data, and that integrates with mapreduce)
parallel databases date from 80's
skew (what happens when data is unbalanced)
lacks careful scheduling of I/O (e.g., what if two nodes pull data from same place at same time during reduce phase)

C. Not novel

D. Missing features

Bulk loader (convert input data into settled format)
Indexing
Updates/Transactions
Integrity constraints
Referential integrity (keep garbage out)
Views (akin to abstract data type – change impl without changing application code)

E. Missing tools

Report writers/visualization
Data mining
Replication

Dryad: try to pull together the best of both worlds