data parallelism

Chris Olston
Yahoo! Research

set-oriented computation

- data management operations tend to be "set-oriented", e.g.:
  - apply f() to each member of a set
  - compute intersection of two sets
- easy to parallelize
- parallel data management is parallel computing's biggest success story

history

- relational datatabase systems (declarative set-oriented primitives) 1970's
- parallel relational database systems 1980's
- renaissance: map-reduce etc. now

architectures

- shared-memory
  - expense
  - scale
- shared-disk
- shared-nothing (clusters)
  - message overheads
  - skew

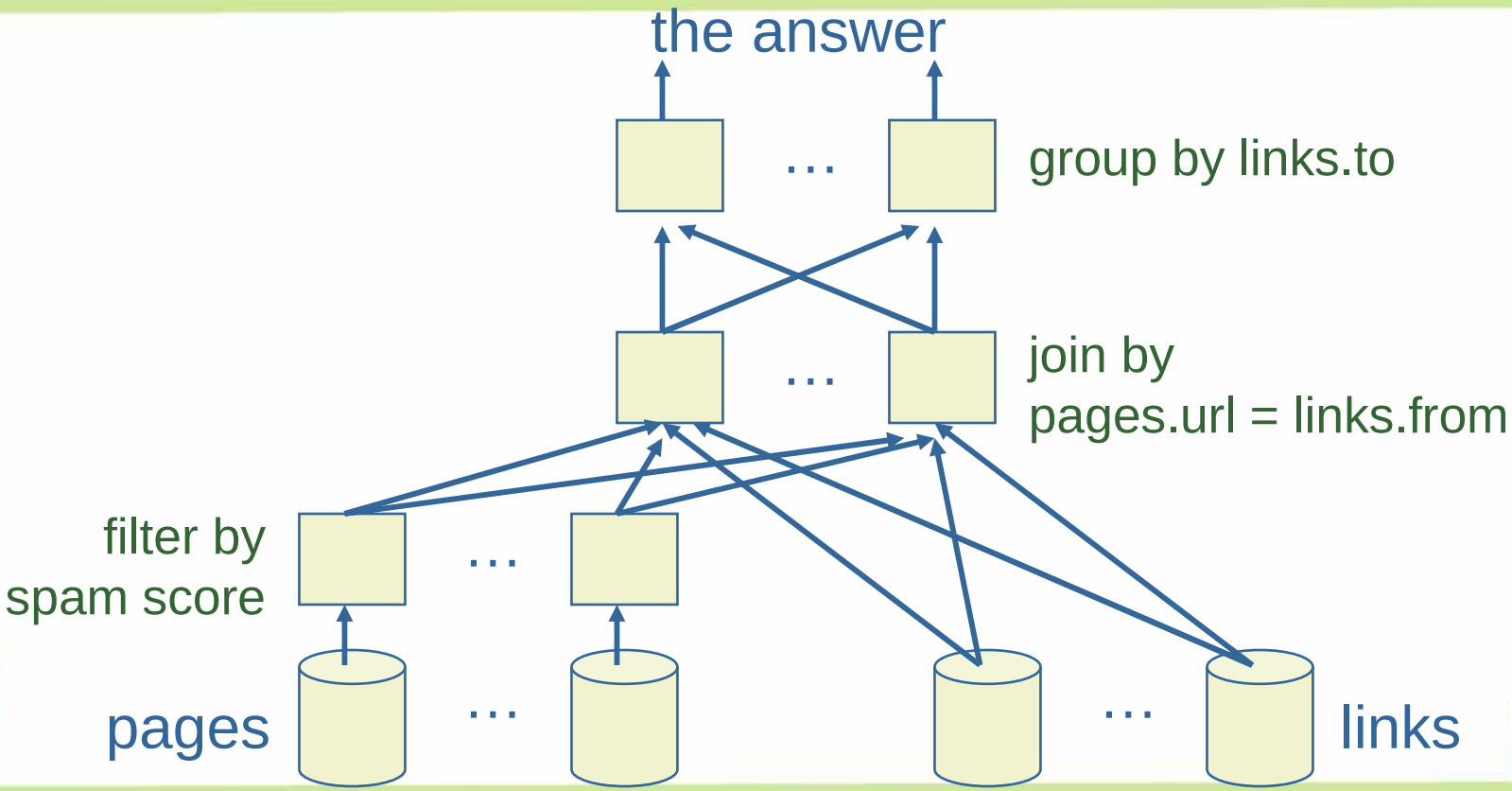
early systems

- XPRS (Berkeley, shared-memory)
- Gamma (Wisconsin, shared-nothing)
- Volcano (Colorado, shared-nothing)
- Bubba (MCC, shared-nothing)
- Terradata (shared-nothing)
- Tandem non-stop SQL (shared-nothing)

example

- data:
  - pages(url, change_freq, spam_score, …)
  - links(from, to)
- question:
  - how many inlinks from non-spam pages does each page have?

parallel evaluation



parallelism opportunities

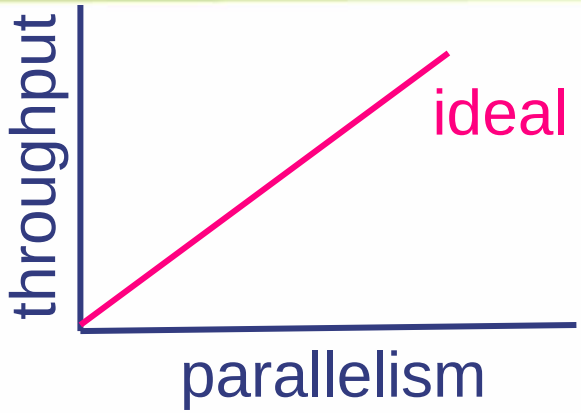
- inter-job
- intra-job
  - inter-operator
    - pipeline $f(g(X))$
    - tree $f(g(X), h(Y))$
  - intra-operator
    - partition $f(X) = f(X_1) \cup f(X_2) \cup \ldots \cup f(X_n)$

parallelism obstacles

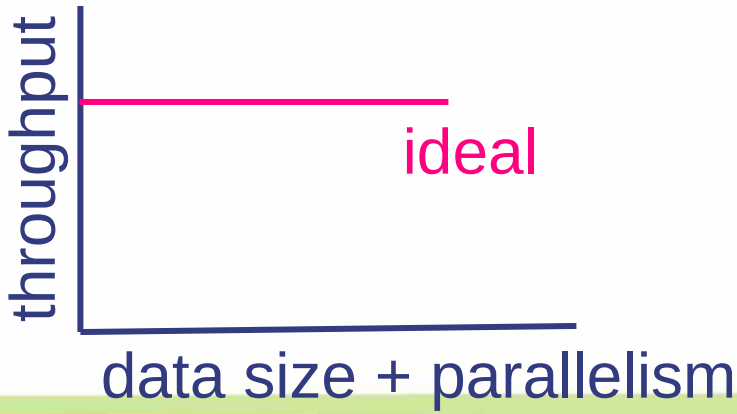
- data dependencies
  - e.g. set intersection w/asymmetric hashing: must hash input1 before reading input2
- resource contention
  - e.g. many nodes transmit to node X simultaneously
- startup & teardown costs

metrics

- speed-up



- scale-up



talk outline

- introduction
- query processing
- data placement
- recent systems

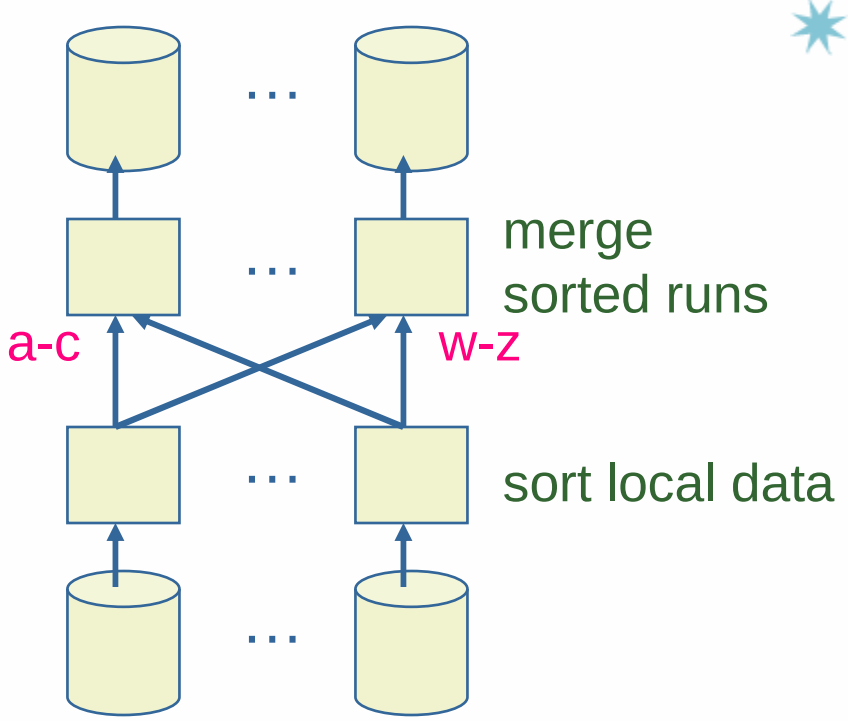
query evaluation

- key primitives:
  - lookup
  - sort
  - group
  - join

lookup by key

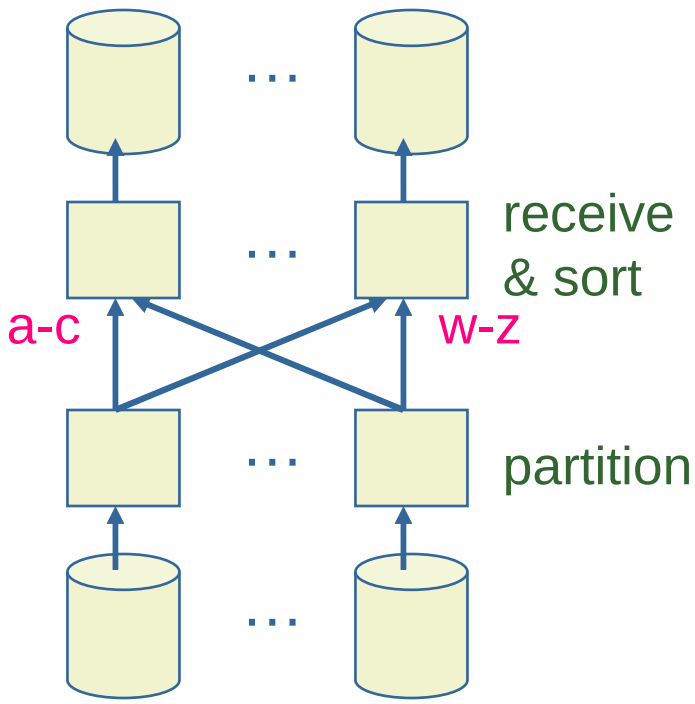
- data partitioned on function of key?
  - great!
- otherwise
  - d'oh

sort



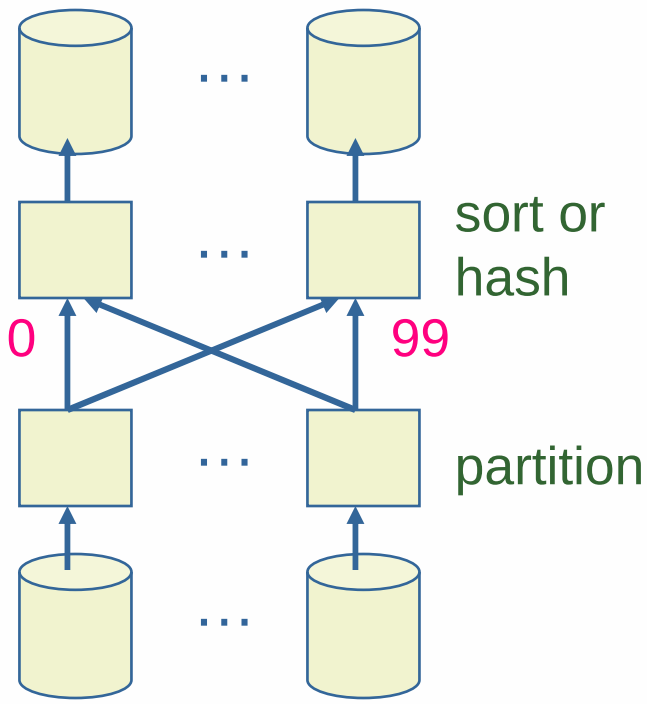
- problems with this approach?

sort, improved



- a key issue: avoiding skew
  - sample to estimate data distribution
  - choose ranges to get uniformity

group

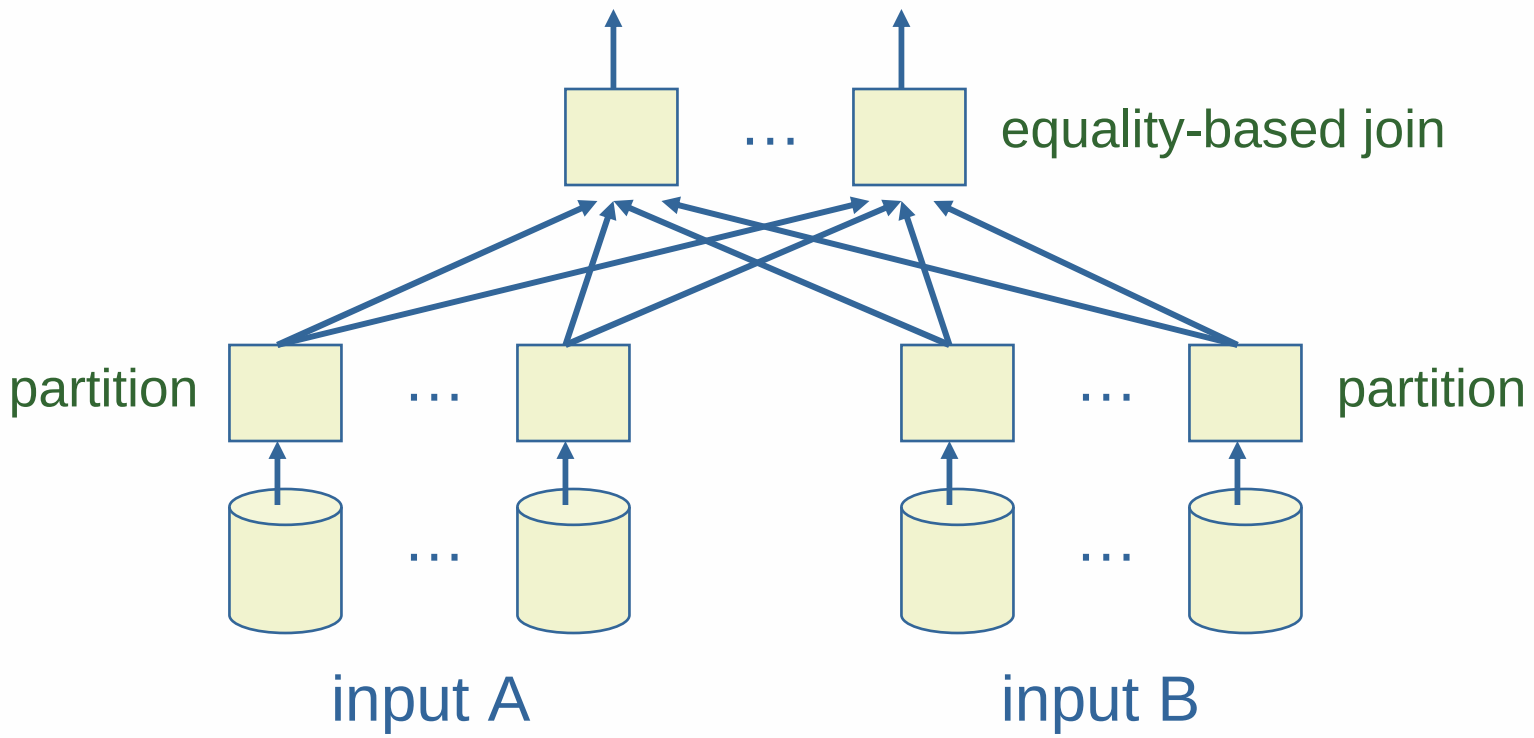


- again, skew is an issue
- approaches:
  - **avoid** (choose partition function carefully)
  - **react** (migrate groups to balance load)

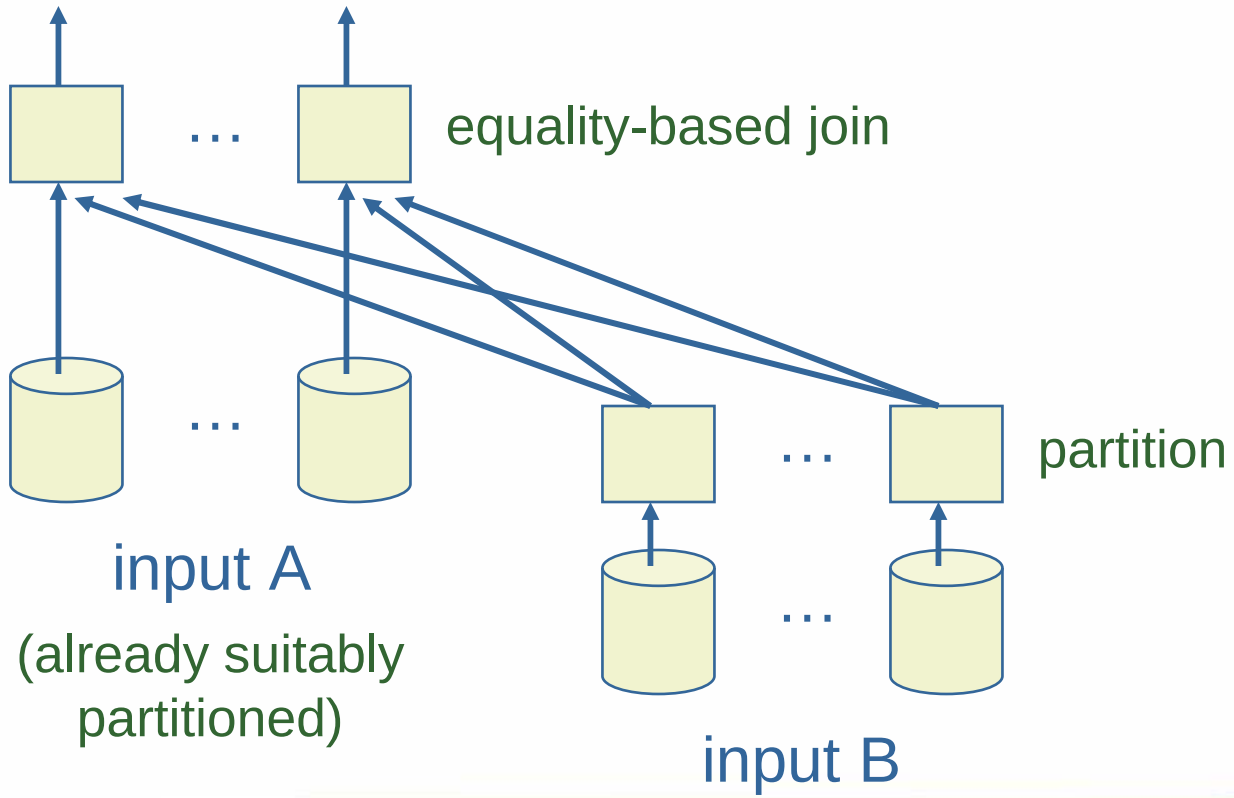
join

- alternatives:
  - symmetric repartitioning
  - asymmetric repartitioning
  - fragment and replicate
  - generalized f-and-r

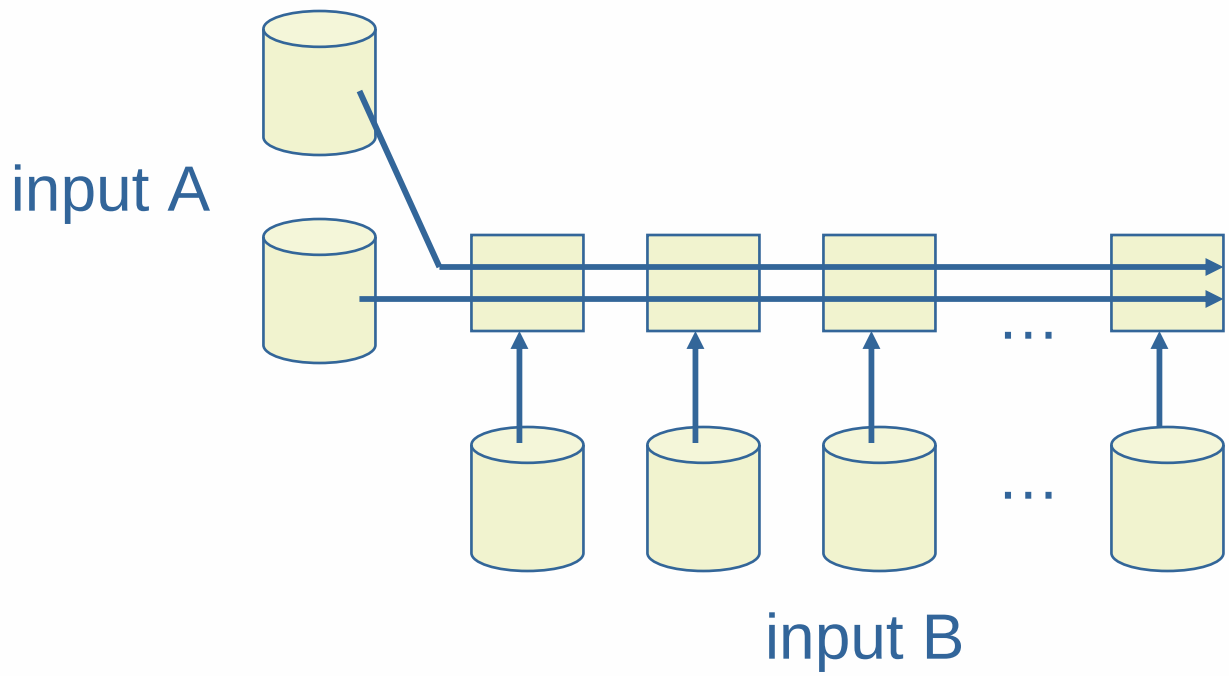
join: symmetric repartitioning



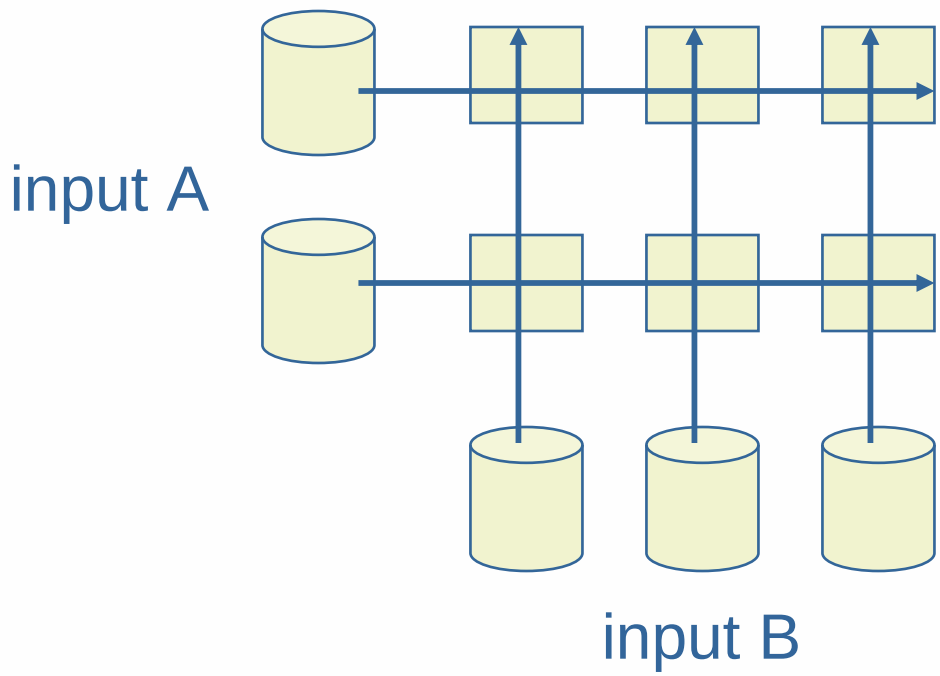
join: asymmetric repartitioning



join: fragment and replicate



join: generalized f-and-r



join: other techniques …

- semi-join
- bloom-join

query optimization

- degrees of freedom
- objective functions
- observations
- approaches

degrees of freedom

- conventional query planning stuff:
  - access methods, join order, join algorithms, selection/projection placement, …
- parallel join strategy (repartition, f-and-r)
- partition choices (“coloring”)
- degree of parallelism
- scheduling of operators onto nodes
- pipeline vs. materialize between nodes

objective functions

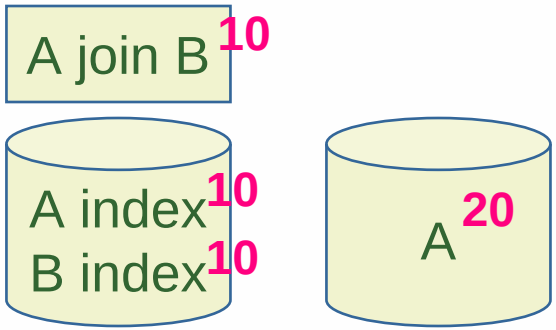
- want:
  - low response time for jobs
  - low overhead (high system throughput)
- these are at odds
  - e.g., pipelining two operators may decrease response time, but incurs more overhead

proposed objective functions

- Hong:
  - linear combination of response time and overhead
- Ganguly:
  - minimize response time, with limit on extra overhead
  - minimize response time, as long as cost-benefit ratio is low

observations

- response time metric violates “principle of optimality”
  - every subplan of an optimal plan is optimal
  - dynamic programming relies on this property
  - hence, so does System-R (w/“interesting orders” patch)
- example:



approaches

- two-phase [Hong]:
  1. find optimal sequential plan
  2. find optimal parallelization of above ("coloring")
     - optimal for shared-memory w/intra-operator parallelism only
- one-phase (still open research):
  - model sources & deterrents of parallelism in cost formulae
  - can't use DP, but can still prune search space using partial orders (i.e., some subplans dominate others) [Ganguly]

talk outline

- introduction
- query processing
- data placement
- recent systems

data placement

- degrees of freedom:
  - declustering degree
  - which set of nodes
  - map records to nodes

declustering degree

- spread table across how many nodes?
  - function of table size
  - determine empirically, for a given system

which set of nodes

- three strategies [Mehta]:
  - random *(worst)*
  - round-robin
  - heat-based *(best, given accurate workload model)*
- (none take into account locality for joins)

map records to nodes

- avoid hot-spots
  - hash partitioning works fairly well
  - range partitioning with careful ranges better [DeWitt]
- add redundancy
  - chained declustering [DeWitt]:

talk outline

- introduction
- query processing
- data placement
- recent systems

academia

- C-store [MIT]
  - separate transactional & read-only systems
  - compressed, column-oriented storage
  - k-way redundancy; copies sorted on different keys
- River & Flux [Berkeley]
  - run-time adaptation to avoid skew
  - high availability for long-running queries via redundant computation

Google (batch computation)

- Map-Reduce:
  - grouped aggregation with UDFs
  - fault tolerance: redo failed operations
  - skew mitigation: fine-grained partitioning & redundant execution of “stragglers”
- Sawzall language:
  - SELECT-FROM-GROUPBY style queries
  - schemas (“protocol buffers”)
  - convert errors into undefined values
  - primitives for operating on nested sets

Google (random access)

- Bigtable:
  - single logical table, physically distributed
    - horizontal partitioning
    - sorted base + deltas, with periodic coalescing
  - API: read/write cells, with versioning
  - one level of nesting: a top-level cell may contain a set
    - e.g. set of incoming anchortext strings

Yahoo!

- Pig (batch computation):
  - relational-algebra-style query language
  - map-reduce-style evaluation
- PNUTS (random access):
  - primary & secondary indexes
  - transactional semantics

IBM, Microsoft

- Impliance [IBM; still on drawing board]
  - 3 kinds of nodes: data, processing, xact mgmt
  - supposed to handle loosely structured data
- Dryad [Microsoft]
  - computation expressed as logical dataflow graph with explicit parallelism
  - query compiler superimposes graph onto cluster nodes

summary

- big data = a good app for parallel computing
- the game:
  - partition & repartition data
  - avoid hotspots, skew
  - be prepared for failures
- still an open area!
  - optimizing complex queries, caching intermediate results, horizontal vs. vertical storage, …