15-721 DB Sys. Design & Impl.

GAMMA

Christos Faloutsos
www.cs.cmu.edu/~christos

Roadmap

Citation

DeWitt, et al. "*The Gamma Database Machine Project*." IEEE TKDE 2(1): 44-63 (March 1990)

Detailed outline

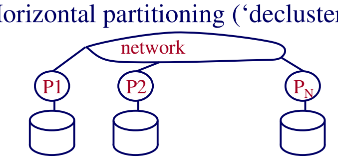
➡ Introduction / History
Hardware
Software architecture
Query processing
Performance evaluation

History

- DIRECT 1977 - 84
  - early database machine project
  - showed parallelism useful for db apps
- Flaws curtailed scalability
  - shared memory
  - central control of execution
- Gamma 1984 - 92

Key Ideas

- Shared-nothing
- Hash-based parallel algorithms
- Horizontal partitioning (‘declustering’)



Gamma Hardware (v2.0)

- iPSC/2 Intel hypercube
- 32 x386 processors
- 8MB of memory
- 330MB Maxtor drive / node (45KB cache)
- Routing modules
  - 2.8 Mb/s
  - full duplex, serial, reliable

Gamma v2.0

- OS: NOSE
- multiple, lightweight processes with shared memory
- Entire DB in one NX/2 process

Detailed outline

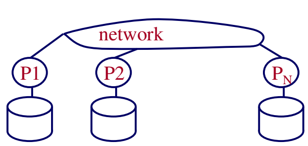
Introduction / History
Hardware
➡ Software architecture
Query processing
Performance evaluation

Storage Organization

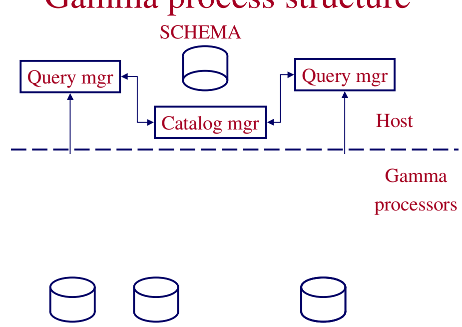
- Horizontal partitioning (user selectable)
  - round-robin; hashed; range partitioned
  - all relations, on all units
  - method of partitioning: recorded in catalog
- Clustered index (not necessarily on partitioning attribute)
- (in retrospect: Partitioning relations should have been based on 'heat')

Gamma Process Structure

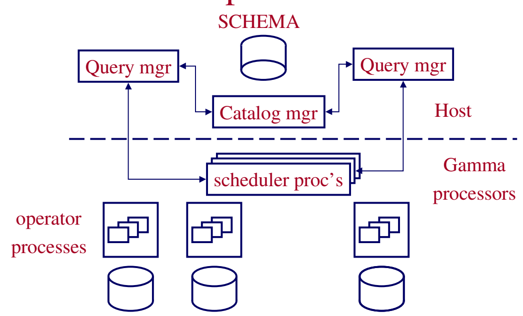
- there is a 'host' machine;
- and the Gamma processors



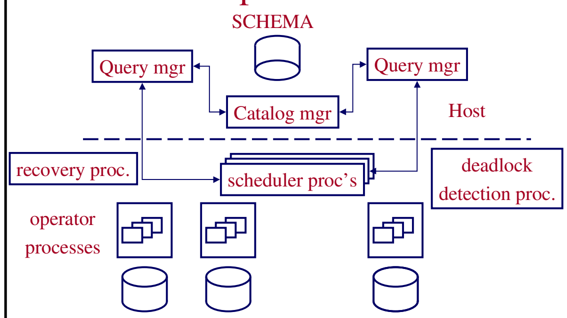
Gamma process structure



Gamma process structure



Gamma process structure



Gamma Process Structure

- Catalog manager
- Query manager
  - one associated with each user (on host)
- Scheduler processes
  - coordinates multi-site queries (spread out - why?)
- Operator processes
  - each executes a single relational operator

Query Processing

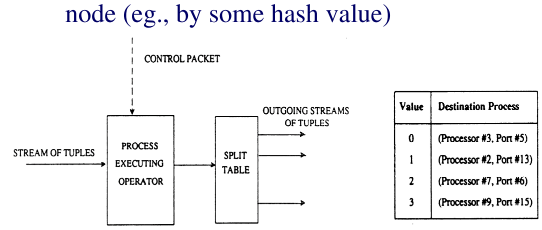
- ad-hoc and embedded query interfaces
- standard parsing, optimization, code gen.
- left deep trees only
- hash joins only
- at most two join operators active simultaneously

Operator and process structure

- data-flow: each operator process
  - reads input stream
  - output tuples in one or more output streams:
- split tables (partitioning, joining)

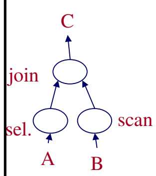
Split Table

Directs operator output to the appropriate node (eg., by some hash value)



Eg.: Parallel Hash Join

P3,P4: diskless
P1,P2: w/disk



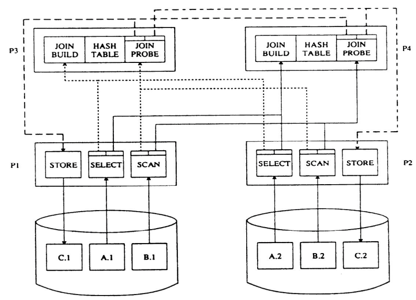


Fig. 6.

Detailed outline

Introduction / History
Hardware
Software architecture
➡ Query processing
Performance evaluation

Selections

- How?

Selections

- Start selection operator on each node
- Exclusion of nodes for hash and range partitioning
- Throughput considerations
  - one page read-ahead

Joins

- Partition into buckets, join buckets
- Implemented: sort-merge, Grace, Simple, Hybrid
- Parallel Hybrid

More operations

- Aggregation - How?

CMU SCS

More operations

- Aggregation
  - Compute partial results for each partition
  - Hash on "group-by"attribute
- Updates
  - Standard techniques
  - How to update partitioning attribute?

CMU SCS

Concurrency Control

- How?

CMU SCS

Concurrency Control

- 2PL
- Granularity:?

CMU SCS

Concurrency Control

- 2PL
- Granularity: file and page
- Modes: S, X, IS, IX, SIX
- Local lock manager and deadlock detector
- (how?)

CMU SCS

Concurrency Control

- 2PL
- Granularity: file and page
- Modes: S, X, IS, IX, SIX
- Local lock manager and deadlock detector
- wait-for graph
- Centralized multi-site lock detector

CMU SCS

Concurrency Control

- Centralized multi-site lock detector
  - Q: How often to check for deadlocks?

Concurrency Control

- Centralized multi-site lock detector
  - Q: How often to check for deadlocks?
  - A: Period halves/doubles on deadlock / no-deadlock

Recovery

- How?

Recovery

- Standard WAL protocol
  - local log manager generates log records
- one or more log processors
  - collect these log records and write them to the disk

Node Failure

- Goal: Availability in spite of a processor or disk failure
- Goal#2: in that case, spread load as uniformly as possible
- How to handle a (single) node failure?

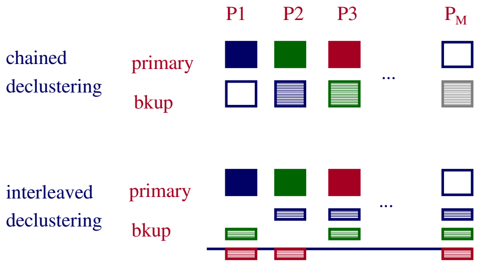
Node Failure

- Chained declustering (Gamma)
  - backup copy on ‘next’ node
- Mirrored disk (Tandem)
- Interleaved declustering (Teradata)
  - backup copy: spread over rest of nodes

Fault Tolerance

- specifically:
- Chained Declustering
  - Primary: i mod M; Backup: (i+1) mod M
- Interleaved declustering
  - Divide fragments into N-1 parts
  - Store parts in all disks but the one containing primary

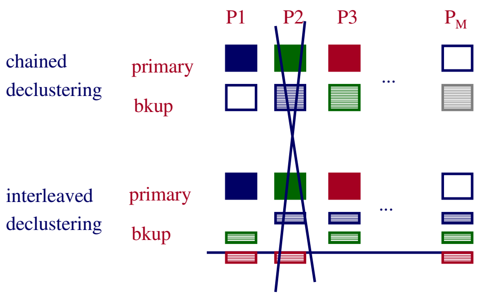
Fault Tolerance



Fault Tolerance

- single-node failure:
  - Load redirection results in 1/n increase (why?)

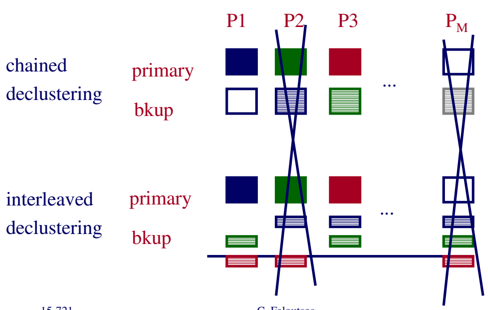
Fault Tolerance



Fault Tolerance

- single-node failure:
  - Load redirection results in 1/n increase (why?)
- Chained vs Interleaved declustering
  - how about a 2-node failure?

Fault Tolerance



Detailed outline

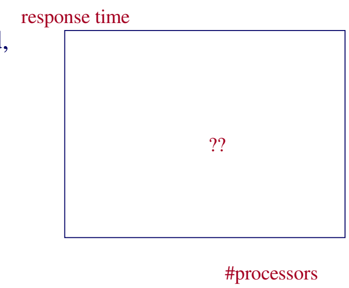
Introduction / History
Hardware
Software architecture
Query processing
➡ Performance evaluation

Experimental setup

- Wisconsin benchmark (100K, 1M, 10M tuples);
- tables: hash partitioned
- selections (1%, 10%) x (non-indexed, clustered index)
- joins
- wallclock time; speedup; scale-up

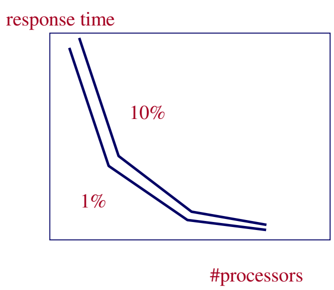
Selections

- non-indexed, 1%, 10%



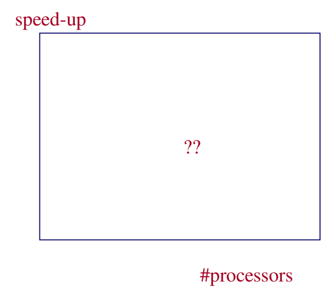
Selections

- non-indexed, 1%, 10%



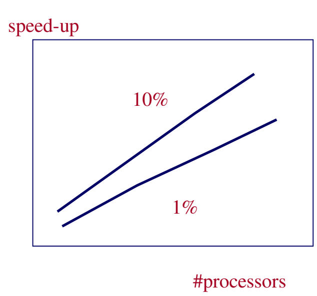
Selections

- non-indexed, 1%, 10%



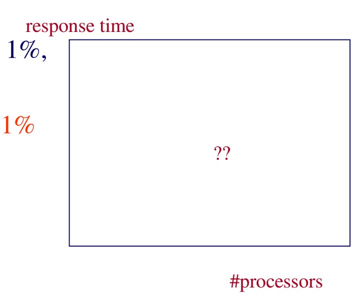
Selections

- non-indexed, 1%, 10%
- 10%: superlinear! (why?)



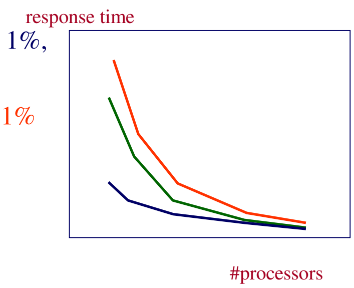
Selections

- clustered ind., 1%,
- clustered 10%
- non-clustered 1%



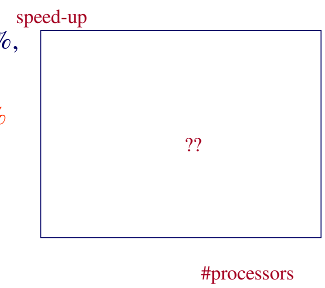
Selections

- clustered ind., 1%,
- clustered 10%
- non-clustered 1%



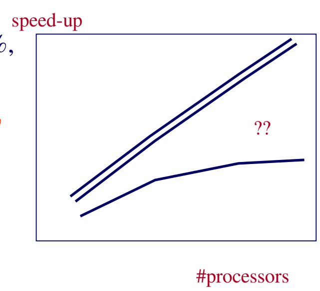
Selections

- clustered ind., 1%,
- clustered 10%
- non-clustered 1%



Selections

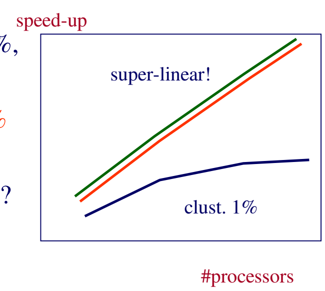
- clustered ind., 1%,
- clustered 10%
- non-clustered 1%



Selections

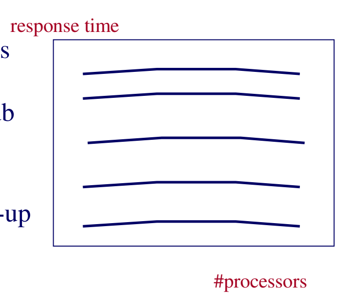
- clustered ind., 1%,
- clustered 10%
- non-clustered 1%

- why super-linear?
- why sub-linear?



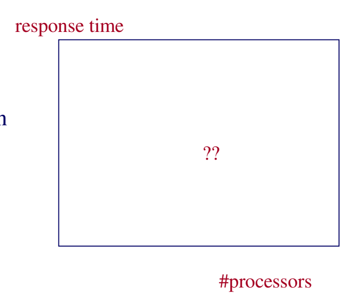
Selections - scaleup

- response time vs processors, increasing the db size
- All queries: ~constant scale-up



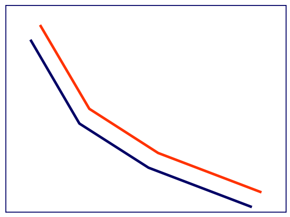
Joins

- joinAB
  - part. = join attr
  - part. attr != join attr



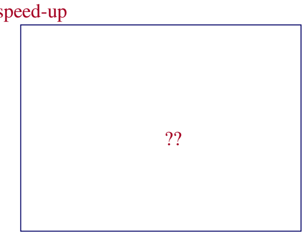
Joins

- joinAB
  - part. = join attr
  - part. attr != join attr



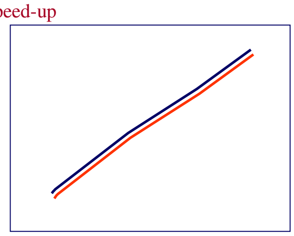
Joins

- joinAB
  - part. = join attr
  - part. attr != join attr



Joins

- joinAB
  - part. = join attr
  - part. attr != join attr
- (scale-up is good, too)



Conclusions

- Shared-nothing architecture
- horizontal partitioning
- parallel hashing
- data-flow scheduling
- good speed-up and scale-up