FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Datenbanksystemimplementierung

Prof. Dr. Viktor Leis

Professur für Datenbanken und Informationssysteme

Parallel Query Processing

Parallelism

Why parallelism

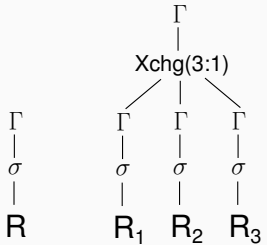
- multiple users at the same time
- modern server CPUs have dozens of CPU cores
- better utilize high-performance IO devices

Forms of Parallelism

- inter-query parallelism: execute multiple queries concurrently
  - map each query to one process/thread
  - concurrency control mechanism isolates the queries
  - except for synchronization this form of parallelism is “for free”
- intra-query parallelism: parallelize a single query
  - horizontal (bushy) parallelism: execute independent sub plans in parallel (not very useful)
  - vertical parallelism: parallelize operators themselves

Vertical Parallelism: Exchange Operator

- optimizer statically determines at query compile-time how many threads should run
- instantiates one query operator plan for each thread
- connects these with “exchange” operators, which encapsulate parallelism, start threads, and buffer data
- relational operator can remain (largely) unchanged
- often (also) used in a distributed setting



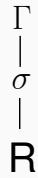
Exchange Operator Variants

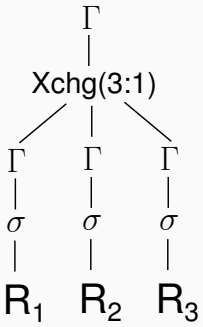
- Xchg(N:M) N input pipelines, M output pipelines

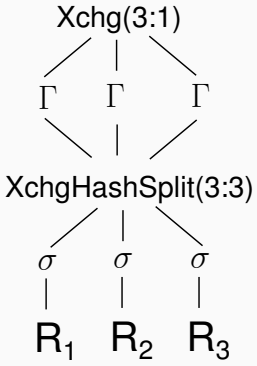
Many useful variants

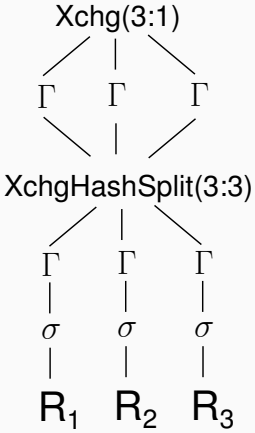
- XchgUnion(N:1) specialization of Xchg
- XchgDynamicSplit(1:M) specialization of Xchg
- XchgHashSplit(N:M) split by hash values
- XchgBroadcast(N:M) send full input to all consumers
- XchgRangeSplit(N:M) partition by data ranges

Aggregation with Exchange Operators (3-way parallelism)

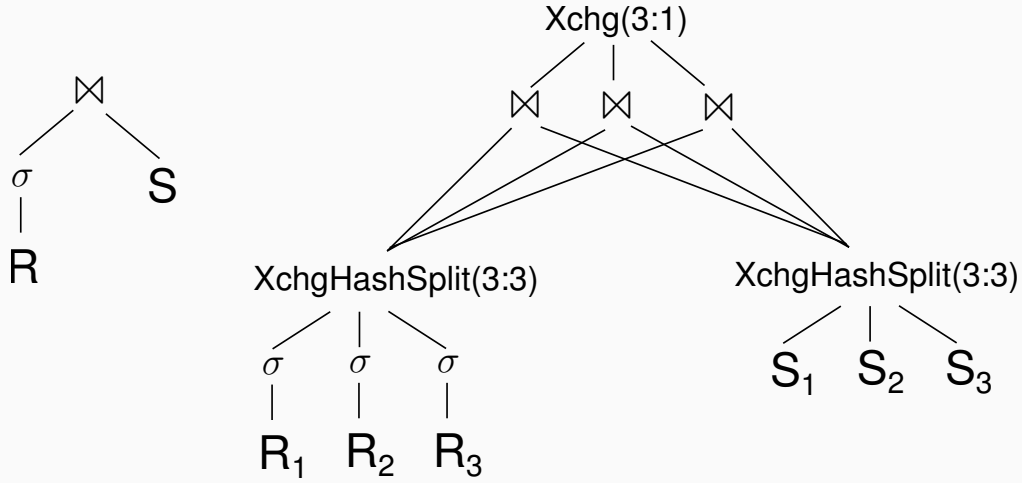








Join with Exchange Operators (3-way parallelism)



Disadvantages of Exchange Operators

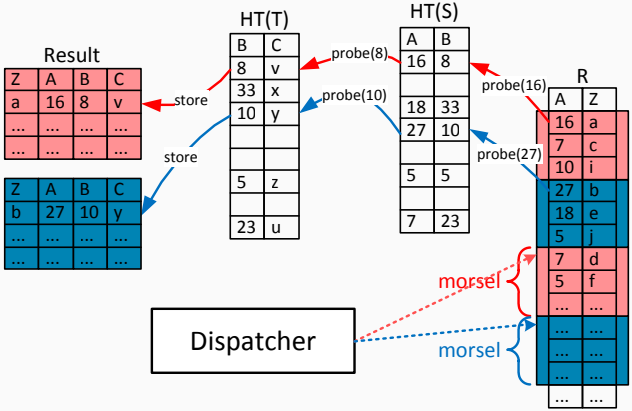
- static work partitioning can cause load imbalances (large problem with many threads)
- degree of parallelism cannot easily be changed mid-query (workload changes)
- overhead:
  - usually implemented using more threads than CPU cores (context switching)
  - hash re-partitioning often does not pay off
  - exchange operators create additional copies of the tuples

Parallel Query Engine

- alternative to Exchange Operators: parallelize operators themselves
- requires synchronization of shared data structures (e.g., hash tables)
- allows for more flexibility in designing parallel algorithms for relational operators

Morsel-Driven Query Execution

- break input into constant-sized work units (“morsels”)
- dispatcher assigns morsels to worker threads
- # worker threads = # hardware threads



Dynamic Scheduling

- the total runtime of a query is the runtime of the slowest thread/core/machine
- when dozens of cores are used, often a single straggler is much slower than the others (e.g., due to other processes in the system or non-uniform data distributions)
- solution: do not partition input data at the beginning, but use dynamic work stealing:
  - synchronized queue of small jobs
  - threads grab work from queue
  - the `parallel_for` construct can provide a high-level interface

Parallel In-Memory Hash Join

1. build phase:
   1.1 each thread scans part of the input and materializes the tuple
   1.2 create table of pointers of appropriate size (tuple count sum of all threads)
   1.3 scan materialized input and add pointers from array to materialized tuples using atomic instructions
2. probe phase: can probe the hash table in parallel without any synchronization (as long no marker is needed)

Aggregation/Group By

- parallel aggregation is one of the most difficult relational operators
- main challenge: behaves very differently depending on whether there are few or many distinct keys
- two-phase approach:
  1. small (cache-resident) per-thread pre-aggregation with spilling to partitions
  2. exchange partitions and aggregate each partition independently

Aggregation/Group By (2)