Streaming Data and Applications

- **Streaming data** refers to data that arrives in a continuous fashion
  - Contrast to **data-at-rest**
- Applications include:
  - Stock market: stream of trades
  - E-commerce site: purchases, searches
  - Sensors: sensor readings
    - Internet of things
  - Network monitoring data
  - Social media: tweets and posts can be viewed as a stream
- Queries on streams can be very useful
  - Monitoring, alerts, automated triggering of actions
Querying Streaming Data

Approaches to querying streams:

- **Windowing**: Break up stream into windows, and queries are run on windows
  - Stream query languages support window operations
  - Windows may be based on time or tuples
  - Must figure out when all tuples in a window have been seen
    - Easy if stream totally ordered by timestamp
    - **Punctuations** specify that all future tuples have timestamp greater than some value
- **Continuous Queries**: Queries written e.g. in SQL, output partial results based on stream seen so far; query results updated continuously
  - Have some applications, but can lead to flood of updates

Querying Streaming Data (Cont.)

Approaches to querying streams (Cont.):

- **Algebraic operators on streams**:
  - Each operator consumes tuples from a stream and outputs tuples
  - Operators can be written e.g., in an imperative language
  - Operator may maintain state
- **Pattern matching**:
  - Queries specify patterns, system detects occurrences of patterns and triggers actions
  - **Complex Event Processing (CEP)** systems
  - E.g., Microsoft StreamInsight, Flink CEP, Oracle Event Processing
Stream Processing Architectures

- Many stream processing systems are purely in-memory, and do not persist data
- **Lambda architecture**: split stream into two, one output goes to stream processing system and the other to a database for storage
  - Easy to implement and widely used
  - But often leads to duplication of querying effort, once on streaming system and once in database

Stream Extensions to SQL

- SQL Window functions described in Section 5.5.2
- Streaming systems often support more window types
  - **Tumbling window**
    - E.g., hourly windows, windows don’t overlap
  - **Hopping window**
    - E.g., hourly window computed every 20 minutes
  - **Sliding window**
    - Window of specified size (based on timestamp interval or number of tuples) around each incoming tuple
  - **Session window**
    - Groups tuples based on user sessions
Window Syntax in SQL

- Windowing syntax varies widely by system
- E.g., in Azure Stream Analytics SQL:
  ```sql
  select item, System.Timestamp as window end, sum(amount)
  from order
  timestamp by datetime
  group by itemid, tumblingwindow(hour, 1)
  ```
- Aggregates are applied on windows
- Result of windowing operation on a stream is a relation
- Many systems support stream-relation joins
- Stream-stream joins often require join conditions to specify bound on timestamp gap between matching tuples
  - E.g., tuples must be at most 30 minutes apart in timestamp

Algebraic Operations on Streams

- Tuples in streams need to be routed to operators
- Routing of streams using DAG and publish-subscribe representations
  - Used in Apache Storm and Apache Kafka respectively

(a) DAG representation of streaming data flow

(b) Publish-subscribe representation of streaming data flow
Publish Subscribe Systems

- **Publish-subscribe (pub-sub)** systems provide convenient abstraction for processing streams
  - Tuples in a stream are published to a topic
  - Consumers subscribe to topic
- Parallel pub-sub systems allow tuples in a topic to be partitioned across multiple machines
- **Apache Kafka** is a popular parallel pub-sub system widely used to manage streaming data
- More details in book

Query Processing in Memory

- Query compilation to machine code
  - Overheads of interpretation
    - E.g., repeatedly finding attribute location within tuple, from metadata
    - Overhead of expression evaluation
  - Compilation can avoid many such overheads and speed up query processing
    - Often via generation of Java byte code / LLVM, with just-in-time (JIT) compilation
- Column-oriented storage
  - Allows vector operations (in conjunction with compilation)
- Cache conscious algorithms
Cache Conscious Algorithms

- Goal: minimize cache misses, make best use of data fetched into the cache as part of a cache line
- For sorting:
  - Use runs that are as large as L3 cache (a few megabytes) to avoid cache misses during sorting of a run
  - Then merge runs as usual in merge-sort
- For hash-join
  - First create partitions such that build+probe partitions fit in memory
  - Then subpartition further s.t. build subpartition+index fits in L3 cache
    - Speeds up probe phase significantly by avoiding cache misses
- Lay out attributes of tuples to maximize cache usage
  - Attributes that are often accessed together should be stored adjacent to each other
- Use multiple threads for parallel query processing
  - Cache misses leads to stall of one thread, but others can proceed

Performance Tuning

- Databases may run into performance problems in practice
  - Issue with the particular workload
- Several things we can do
  - Buy more hardware
  - Indices
  - Adjust schema
  - Modify transactions
  - Relax transactional consistency
Solutions

• Buy more hardware
  – Doesn’t always help – e.g., parallel system
    • Replicated data – multiple updates slows write transactions
    • Joining data from different machines may be slower
  – Need to analyze to determine performance bottlenecks
• Indices
  – Again, helps search, but increase update cost
  – Which queries are slow? Will they be helped by index?

Schema Tuning

• Column store vs. Row store
  – “Hard-wired” into DBMS
  – But can “break apart” relations to simulate column store
    • May need to generate a key
• De-normalization
  – “Pre-join” data that is frequently joined
  – Loses consistency/integrity advantages (or requires external checks)
  – Potential replication – wastes space
• For read queries, better done through materialized views
• Some databases also support clustered store (stored as if de-normalized)
Query / Transaction Tuning

• Query optimizers good, but not perfect
  – Sometimes “give up”, particularly with sub-queries
    • Select … from ( select … from … ) where x in ( select … from … )
    • Rewrite as Select / cross-product / where clause / group by
  – Hints to the query optimizer
    • In some DBMSs, can explicitly give hints to the query optimizer
    • In others, relation statistics stored in the catalog, and you can manipulate those numbers to “trick” the query optimizer

• Programmers not perfect
  – Often break a single query into separate queries, doing operations in the program that could be done separately
  – Rewrite programs to push operations into a single query
    • Let the query optimizer do its work

• Stored Procedures
  – Some databases support stored procedures
  – Complex, frequent queries can be saved, so optimizer doesn’t run for every execution
  – Kind of like views, but more programming capabilities
Concurrency / Constraint Relaxation

- Set different levels of isolation
  - Sometimes at DB level, some DBMSs support at query level
  - Particularly useful for OLAP (Online Analytical Processing) queries when consistency not critical
- Split transaction into “mini-transactions”
  - Must handle concurrency and failure/recovery yourself
- Turn off consistency checks (key, foreign key, etc.)
  - Useful when “Bulk loading” data
  - Load the data, then check constraints and re-enable integrity constraints

Performance: Benchmarks

- You’ll frequently see benchmarks comparing DBMSs
- Suite of transactions
  - Typically report throughput (transactions per second)
  - Benchmark specifies how this is measured
- Transaction Processing Council (TPC) Benchmarks
  - OLTP benchmarks – TPC-A, TPC-B, TPC-C
    - TPC-C is current, models inventory management system
  - OLAP benchmarks – TPC-D, TPC-H, TPC-R
    - TPC-H, TPC-R – primarily aggregation, TPC-H has restrictions on use of indices and materialized views
    - TPC-W – “Web Bookstore” model, dynamically generated pages
Benchmarks: Caveats

When looking at numbers, be aware of:

• Configuration used (hardware/etc.)
• Constraint relaxations (e.g., are they enforcing serializability)
  – Transactions run concurrently or serially?
• Database sizes
  – Not fixed, TPC benchmarks can be scaled