Overview: Chapter 6

- Sensor Network Databases
  - Sensor networks are conceptually a distributed DB
    - Store collected data
    - Indexes the data
    - Serves queries from users on the data
  - DB abstraction
    - Separates logical data view(naming,access,operations) from implementation details
    - Loss of efficiency, but increased ease of use

Challenges

- Transferring data to central server not feasible
  - Need in-network storage
- Node memories/storage are limited
  - Older data needs to be discarded
- Classic DB performance metrics not suitable
- Query languages need additional operators
  - Support for continuous long-running queries
  - Need to correlate current readings with past statistics

Querying the Physical Environment

- Need a high-level query language
  - Separate query from underlying implementation details
  - Allow non-expert users easy access to data
    - Query execution plan dependent on system state and implementation
  - Continuous and historical queries need special operators

Query Interface: Cougar Sensor DB

- Each type of sensor is an abstract data type (ADT)
  - Similar to Object-Relational DB
  - Can perform functions on sensor data
- Distributed query processing
  - Query transmitted to sensors
  - Only those that satisfy query respond
  - Eliminate need to transmit all sensor data to central server

Query Interface: Probabilistic Queries

- Uncertainty in data caused by noise etc.
- Requests for exact sensor values not appropriate
  - Need query language that can handle uncertainty
  - Gaussian ADT (GADT) models uncertainty as continuous pdf
  - Query: retrieve all values from sensors with value X with probability at least Y

Database Organization

- Centralized data storage
  - Sensors forward all data to centralized server
  - Queries do not incur additional network overhead
  - Nodes near server act as routing hotspots and become depleted of energy
- In-network data storage
  - Storage points for data in the network act as rendezvous points between queries and data
  - Load is balanced across network
  - Query time depends on indexing scheme
  - In-network storage allows aggregation of data ahead of query processing

Query Propagation & Aggregation

- Server-based approach
  - Data is sent to server and aggregated at server
- In-network aggregation
  - Data is aggregated at sensors
  - Query propagated to sensors
  - Need efficient routing protocol
  - How to aggregate the data
    - Can some data be computed ahead of time and then aggregated?

TinyDB

- SQL-style query interface
  - SQL operators: count, min, max, sum, average
  - Extension operators: median, histogram
- In-network aggregate query-processing
  - Sensitive to resource constraints and lossy communication channels

Query Processing Scheduling

- Processing based on sampling period
  - Sampling period divided into time intervals
  - Aggregation results reported at end of each sampling period
  - Choice of sampling period is important
    - Routing tree depends on number of intervals in sampling period
    - Nodes schedule processing, communication etc. based on routing tree

Optimizations

- Aggregation can be pipelined to increase throughput
- Nodes can snoop packets of others to make early decisions
- Report only changed data
- Adaptive aggregation to support changing network conditions

Data-Centric Storage

- Flexible storage needed when queries can originate from *within* network
  - Server-based approaches flood queries to all nodes
  - Data-Centric Storage (DCS) make use of rendezvous points to aggregate queries and data

Geographical Hash Table (GHT)

- Mapping from node attribute to storage location by hash function
  - Distributes data evenly across network
  - Nodes know their location
  - Objects have keys
  - Nodes responsible for storing a range of keys
- Geographical routing algorithm
  - Any node can locate storage node for any attribute
- Nodes *put* and *get* data to/from storage
  - Hashtable interface
- Data replicated at nodes near the storage point for a key

Data Indices & Range Queries

- Sensor network DBs need to support range queries
  - Query specifies value range for each attribute
- GHTs and base TinyDB model not adequate as is
  - Need to index data to support complex queries
- Metrics for measuring index
  - Speed gains for query processing
  - Size of index
  - Costs of building and maintaining index

1-D Indices

- 1-Dimensional indices
  - Most prevalent type of DB index
  - Indexes data parameterized by single value
  - Implemented with data structures: B-trees, hash tables, etc.
- Not directly mapable to distributed implementation
- 1D Index for range queries
  - Pre-compute and store answer to certain range queries
  - Compute answer to arbitrary range query from pre-computed answers
  - Adapting to distributed storage model requires partial aggregation of results

Multi-Dimensional Indices

- Can’t simply create multiple 1D indices
  - Queries will retrieve many more records than necessary
- Indexing scheme must take into account need to range queries and distributed storage
- Quickly eliminate irrelevant records
  - Top-down hierarchical approach
  - K-D trees, quad trees, R trees etc.
- Assumes orthogonal ranges
  - Non-orthogonal range searching requires more complex schemes

Multi-Resolution Summarization

- Aggregate summaries of data can be used to "drill down" to more specific queries
- Saves on storage space and network communication
- Need appropriate data structures (e.g., quad trees, etc.)
  - Indexing based on data structure
  - Match queries based on ranges from child branches at each node
  - Non-matching branches pruned off

Partitioning the Summaries

- Balances workload across nodes
- DIFS
  - Nodes distributed in 2D space
  - Use multi-rooted quad-trees to partition spatial domain
    - Wider spatial domain of node = narrower value range indexed
    - Couples spatial domain decomposition to value indexing
  - Uses a GHT to locate index nodes

Fractional Cascading

- Intuition: queries exhibit temporal and spatial locality and are not for arbitrary data
- Idea: store information about other nodes at each node, but nodes only know "fraction" of info from far nodes
  - Little duplicate info
  - Spatial locality: nearby neighbors know a lot about local information
  - Can still satisfy queries of distant nodes

Locality-Preserving Hashing

- Map high-dimensional attribute space to a plane
    - Close values in high-D space are close in the plane
- Spatial domain divided into zones
- Locality-preserving geographical hash
    - Maps attribute space to spatial domain
    - Values that are similar are mapped to nearby nodes

Data Aging

- Continuous data acquisition and limited storage = data storage problem
- Older data is aggregated and summarized
  - Eventually older data is discarded
- Aging functions are difficult to construct
  - Aging policy is application dependent

Indexing Motion Data

- Continuously changing sensor data
- Fixed index structure not appropriate
  - Heavy index update penalty for continuous data
- Need to support temporal queries, predictive queries, etc.
- Apriori motion knowledge
  - Time is treated as another attribute
  - Standard indexing methods may apply
- No apriori knowledge
  - More likely in physical world
  - Dynamic index. Updates only on critical events