Embedded Databases

Arne Claassen

Why use an embedded DB

- In-process
- Speed
- No network/pipe latency
- Not shared
- Avoid Impedance mismatch
- Bring the work to the data
  - Map Reduce
  - Native manipulation (LINQ to objects, etc.)

Types of embedded databases

- Key-Value Storage
- Relational (SQLite)
- OODB (db4Objects)
- Graph (Sones GraphDB)
- Other (lucene)

Key Value Storage

- Dictionary
- Single Key, Single Value
- Key and Value can be any type
  - Serialization permitting
- CRUD
- Enumerable
  - Key array
  - Cursor

Traditional Representations

- Hash
  - Faster, O(1)
  - Undefined order
  - Best for random access
- Tree (B-tree, B+tree, etc.)
  - Slower, O(log N)
  - Ordered by key comparison logic
  - Fast range queries on key

KVS compared to Relational

- Table
  - Key = PK
  - Value = Row blob
- Index
  - Use additional dictionaries for indicies
  - Key = index
  - Value = collection of Table Keys

Good Scenarios for KVS

- Authoritative owner of Data
- Stream access of large Values
- Data processing
- Cache

DBM – the original KVS

- Original Unix database (disk bound hash)
- Accessible from virtually all unix scripting languages
- Supports Get, Set, Delete, Iteration
- Successors
  - QDB
  - JDBM
  - Berkeley DB
  - Tokyo/Kyoto Cabinet

.NET embedded KVS option

- ESENT
- Berkeley DB
- Madcow Memory Mapped Data Structures
- Firkin
- Write a wrapper for Kyoto Cabinet

Firkin

- Inspired by Riak's BitCask
- In-memory index
- Append-only file structure
- Concurrency determined by segmentation size
- Stream interface

Serialization Options

- KVS store binary data
- BinarySerializer - worst of class solution
- Faster alternatives
  - Protocol Buffers
  - BSON
  - MetSys.Little

Modeling Strategies

- Serialize object graphs
- Denormalize
- Concentrate on 90% use case

And then there's Lucene

- Indexing engine
- Records are Documents
- Can store records of field/value pairs
- Better optimized for indexing field/value pairs