FoundationDB Key-Value Store 3.0.x Datasheet

Read more at https://foundationdb.com

Data model
- Key-value store
- Single logical key range, lexicographically ordered

Features
- Stores data on disk, cached in memory
- Supports extremely high concurrency workloads
- Asynchronous APIs based on "futures"
- Paxos-based algorithm prevents split-brain problems
- Management command line interface (CLI)
- Multi-datacenter replication
- Online point-in-time backup
- Predictable performance even under adverse workloads
- Maintains its contract in all circumstances

API summary
- Transactions
  - create() => create new transaction
  - commit() => confirm transaction success
- Data access
  - get(key) => return value
  - set(key, value) => set value
  - clear_range(start, end) => clear range
  - get_range(start, end) => return range
  - watch(key) => notify when key changes
  - atomic ops: add, bit_or, bit_and, bit_xor => high-contention writes
  - location(key) => physical location of data
- Key selectors
  - Used to find keys without an exact match
  - key_after(key) => find the next key
  - key_before(key) => find the previous key
  - Additional offsets allowed (e.g., +2 keys)
- Tuples
  - Used to create structured keys
  - Example: ('mydata', 'users', 'id', 704)
  - Ordered by tuple element
- Directories
  - Manage keyspace via hierarchical "directories"
  - Move/rename/delete all keys in directory atomically

True ACID transactions
- Atomicity across arbitrary keys
- Strong consistency
- Serializable isolation, even with range operations
- Durability via replication and fsync to storage
- Causality (see all previously committed transactions)

Gracefully handles without downtime
- Machine failure
- Network failure
- Network partitions
- Load balancing
- Adding new machines
- Removing machines
- Migrating to a new cluster

Implementation details
- Lock-free design using optimistic concurrency
- Multi-version concurrency control (MVCC)
- Written in the Flow language for native-code performance
- Industry-leading testing via deterministic simulation

Example performance
- Linear scaling
  - Achieved 8.2 million operations/sec on a 24-machine EC2 c3.8xlarge cluster (90/10 random R/W)
- Low latencies
  - Typical when run at less than 75% load:

Start transaction	0.3 - 1ms
Read	0.1 - 1ms
Set	0 (deferred until commit)
Commit	1.5 - 2.5ms

- Throughput (per core)

	ssd engine	memory engine
Reads	55,000/sec	90,000/sec
Writes	20,000/sec	35,000/sec

- Concurrency
  - 90/10: maximum throughput is reached at about 200 ops/sec (achieved with 20 concurrent transactions per process for a workload using 10 ops/transaction)

Limits
- Key size: 10,000 bytes
- Value size: 100,000 bytes
- Transaction size: 10MB
- Transaction duration: 5 seconds (tunable)
- Cluster size: ~400 processes
- Write bandwidth: ~300MB/s
- Database size: 100TB
- Network latency requirement: <1 second
- Failure recovery time: seconds
- Upgrade time: seconds to minutes

Deployment model
- Single-threaded worker process
- One or more processes on one or more machines
- All processes work together to act as a single logical database
- Separate client and server packages

Client languages
- Java/JVM, C#/.NET, Python, C, Ruby, node.js, PHP, and Go (Common Lisp, Erlang, Julia, and Lua community developed)

Platform requirements
- Linux, Windows, OS X
- 4GB RAM per process
- SSD or other fast storage