Non-relational data stores

Overview, coding and assessment: MongoDB, Tokyo Tyrant & CouchDB

github.com/igal/ruby_datastores

Igal Koshevoy, Pragmaticraft
Business-Technology Consultant
igal@pragmaticraft.com
@igalko on Twitter & Identi.ca

Terminology

- **Relational database:** Rigidly-defined relations of table columns to data rows, e.g., PostgreSQL, MySQL, etc.
- **Key-value store:** Hash-like struct, often mapping a string key to a string value, e.g., Berkely DB, TokyoTyrant hash, etc.
- **Document-oriented DB:** Hash-like struct, typically mapping key to arbitrary columns, e.g., CouchDB, MongoDB, Tokyo Tyrant's table, etc.

Why non-relational?

- Easier replication and scaling
- Incremental schema migrations
- Better performance, sometimes

Why NOT non-relational?

- Most are bad for transactions and durability
- Most are newfangled, young and brash with teething pains
- Require different coding patterns and skills to use effectively

Non-relational coding patterns

- Denormalization to reduce finds/queries due to lack of relational join:
    - Calculations
    - Foreign keys
    - Foreign values
- Track schema versions per document
- Use as-needed incremental migrations
- Store associations externally or internally
- Shard data

MongoDB

http://mongodb.org/

A 10gen project under GNU AGPL v3.0

Document-oriented

Pros

- Great site, docs, API & community
- General purpose
- Quick performance
- Scalable: master-slaves, shards
- Resilient: replica pair
- Many data types
- Multiple indexes
- Sophisticated queries
- Many atomic operations
- Map-Reduce (on shards, soon)

Cons

- No transactions across collections
- No ACID or MVCC
- Not the fastest

MongoDB (cont.)

```
require 'mongo' # Using mongo 0.16 driver from GemCutter

# Connect.
db = Mongo::Connection.new("localhost", 27017).db("mydb")

# Get a collection.
collection = db.collection("mycollection")

# Add an index.
collection.create_index("number")

# Insert an item.
collection << { :number => 1, :message => "Hello" }

# Retrieve an item.
p collection.find_one(:number => 1)

# Query items.
p collection.find(:message => /ello/).to_a
```

Tokyo Cabinet + Tyrant

http://tokyocabinet.sourceforge.net/

A mixi.jp project under GNU LGPL v2.1

Key-value, document-oriented & other engines

Pros

- Specialized engines
- Very fast
- Scalable: master-slaves
- Resilient: dual master
- Multiple indexes
- Can do transactions
- memcache-compatible API

Cons

- Fewer features
- Strings only
- Simplistic queries

Tokyo Cabinet + Tyrant (cont.)

```
require 'rufus/tokyo/tyrant'

# Connect.
db = Rufus::Tokyo::TyrantTable.new('localhost', 1978)

# Insert an item.
db["foo"] = { "number" => "1", "message" => "Hello" }

# Retrieve an item.
p db["foo"]

# Query items.
p db.query do |q|
  q.add_condition("message", :includes, "ello")
  q.limit(5)
end
```

CouchDB

http://couchdb.apache.org/

An Apache project under Apache License 2.0

Document-oriented

Pros

- Very scalable: multi-master
- MVCC
- ACID
- Versioned documents
- REST
- Sophisticated queries
- Map-Reduce

Cons

- Very, very, very slow
- Must create views
- Harder to use than others
- Site and docs: FAIL

CouchDB (cont.)

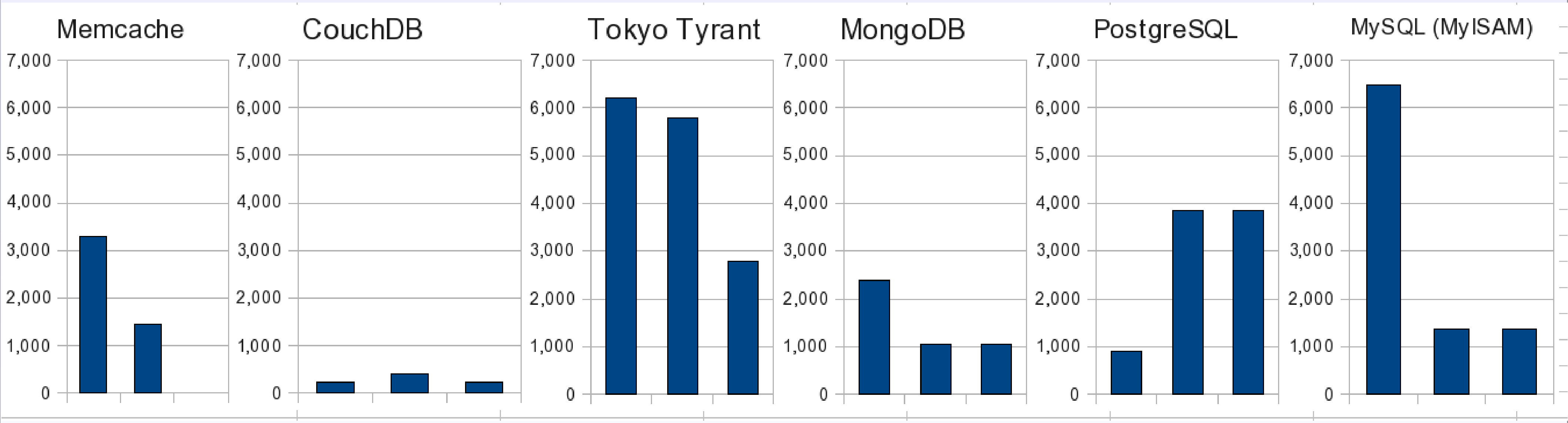
```
require 'couchrest'

# Connect.
db = CouchRest.database!("http://127.0.0.1:5984/couchrest-test")

# Insert an item.
db.save_doc("_id" => "foo", "number" => 1, "message" => "Hello")

# Retrieve an item.
p db.get("foo")
```

CouchDB (cont.)

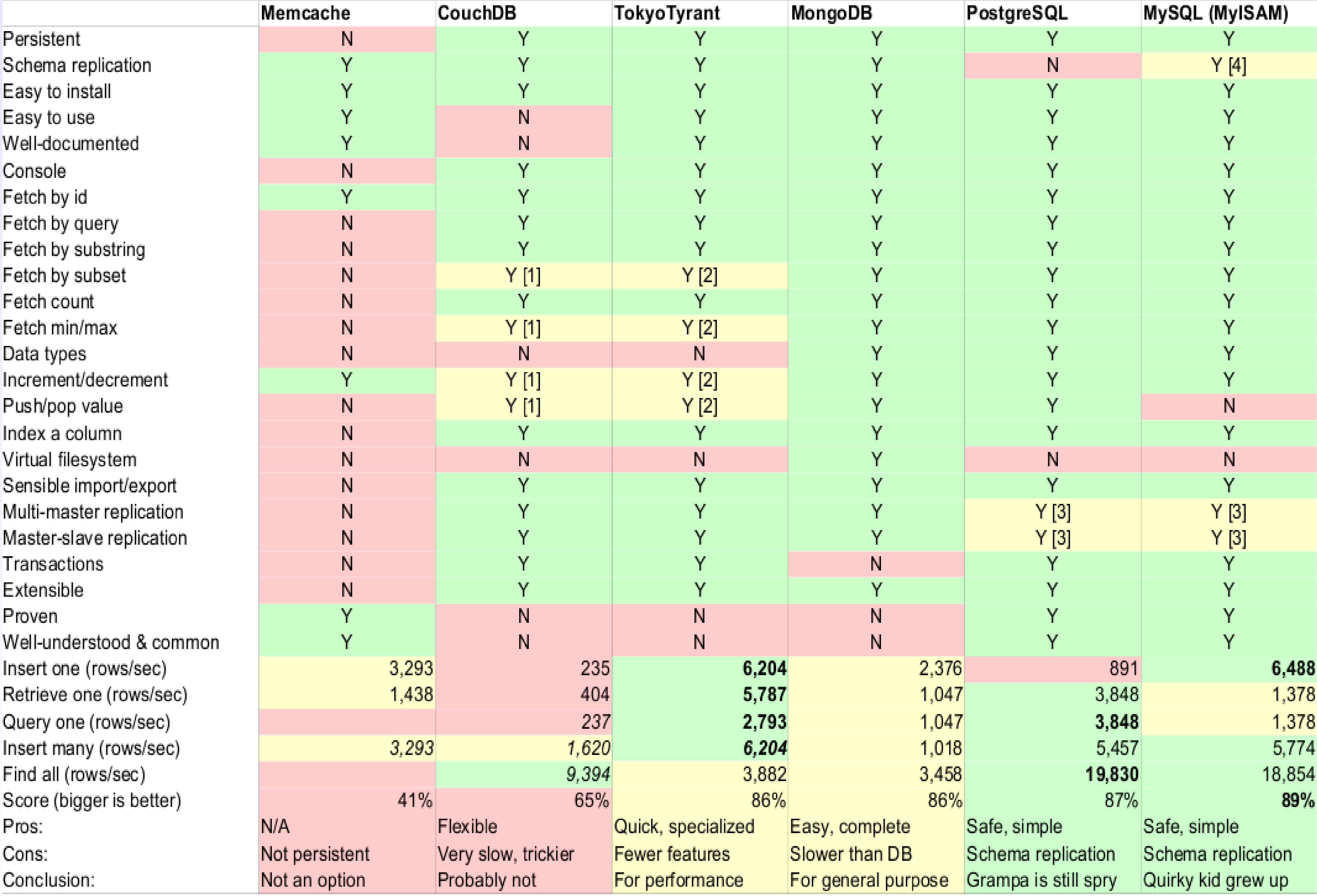
```
# ...continued from last slide

# Add an view.
db.delete_doc db.get("_design/queries") rescue nil
db.save_doc({
  "_id" => "_design/queries",
  :views => {
    :by_number => {
      :map => "function(doc) {
        if (doc.number) {
          emit(doc.number, doc);
        }
      }"
    }
  }
})

# Query items.
p db.view("queries/by_number", :key => 1)["rows"].map{|row| row["value"]}
```

Naive benchmarks

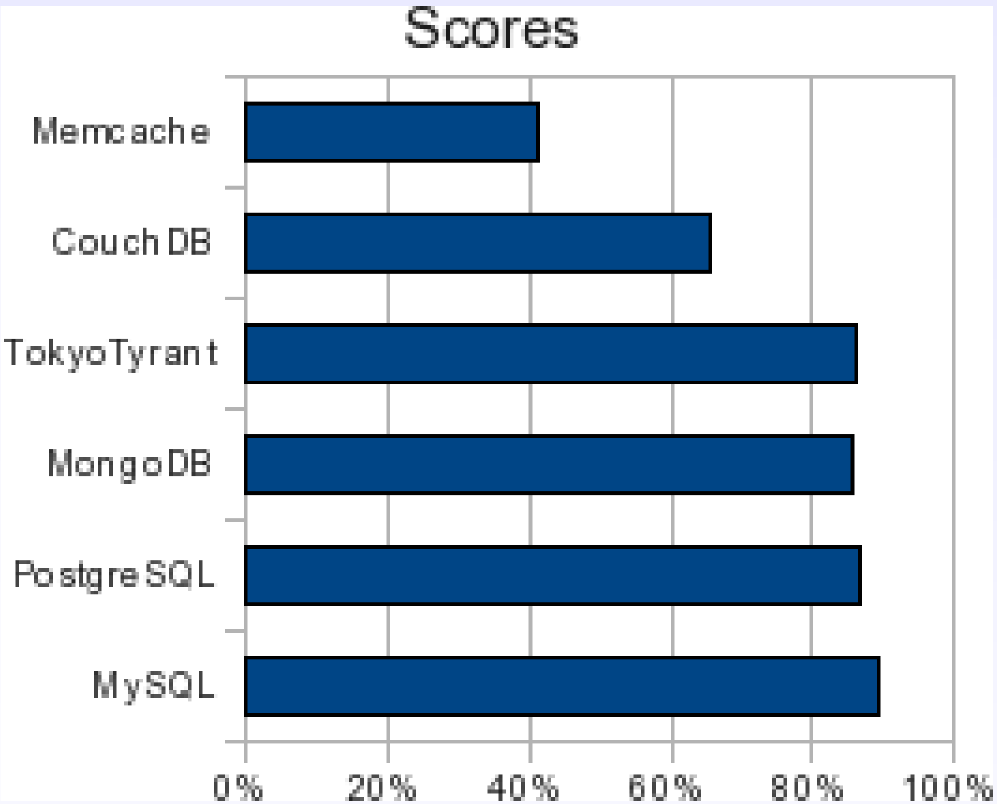
Columns in graphs, left to right:
1. Insert one
2. Retrieve one
3. Query one



Above benchmarks are naive: They use serial operations in tight loops with small datasets from single host to localhost, rather than concurrent mixture across many clients & servers with much data.



	Memcache	CouchDB	TokyoTyrant	MongoDB	PostgreSQL	MySQL (MyISAM)
Persistent	N	Y	Y	Y	Y	Y
Schema replication	Y	Y	Y	Y	N	Y [4]
Easy to install	Y	Y	Y	Y	Y	Y
Easy to use	Y	N	Y	Y	Y	Y
Well-documented	Y	N	Y	Y	Y	Y
Console	N	Y	Y	Y	Y	Y
Fetch by id	Y	Y	Y	Y	Y	Y
Fetch by query	N	Y	Y	Y	Y	Y
Fetch by substring	N	Y	Y	Y	Y	Y
Fetch by subset	N	Y [1]	Y [2]	Y	Y	Y
Fetch count	N	Y	Y	Y	Y	Y
Fetch min/max	N	Y [1]	Y [2]	Y	Y	Y
Data types	N	N	N	Y	Y	Y
Increment/decrement	Y	Y [1]	Y [2]	Y	Y	Y
Push/pop value	N	Y [1]	Y [2]	Y	Y	N
Index a column	N	Y	Y	Y	Y	Y
Virtual filesystem	N	N	N	Y	N	N
Sensible import/export	N	Y	Y	Y	Y	Y
Multi-master replication	N	Y	Y	Y	Y [3]	Y [3]
Master-slave replication	N	Y	Y	Y	Y [3]	Y [3]
Transactions	N	Y	Y	N	Y	Y
Extensible	N	Y	Y	Y	Y	Y
Proven	Y	N	N	N	Y	Y
Well-understood & common	Y	N	N	N	Y	Y
Insert one (rows/sec)	3,293	235	**6,204**	2,376	891	**6,488**
Retrieve one (rows/sec)	1,438	404	**5,787**	1,047	3,848	1,378
Query one (rows/sec)		*237*	**2,793**	1,047	**3,848**	1,378
Insert many (rows/sec)	*3,293*	*1,620*	***6,204***	1,018	5,457	5,774
Find all (rows/sec)		*9,394*	3,882	3,458	**19,830**	18,854
Score (bigger is better)	41%	65%	86%	86%	87%	**89%**
Pros:	N/A	Flexible	Quick, specialized	Easy, complete	Safe, simple	Safe, simple
Cons:	Not persistent	Very slow, trickier	Fewer features	Slower than DB	Schema replication	Schema replication
Conclusion:	Not an option	Probably not	For performance	For general purpose	Grampa is still spry	Quirky kid grew up


Scores
Memcache
CouchDB
TokyoTyrant
MongoDB
PostgreSQL
MySQL
0%
20%
40%
60%
80%
100%

Conclusions

- MongoDB and Tokyo Tyrant are useful now. CouchDB has promise, but is too slow currently.
- Non-relational databases have shown their worth at larger sites when used cleverly.
- Non-relational databases will continue to improve performance, stability & features.
- Relational databases are still a great choice: fast, powerful and proven. With caching, denormalization, rework (e.g. Drizzle) & better replication, they will continue to be competitive.