OpenEdge & CouchDB

Integrating the OpenEdge ABL with CouchDB

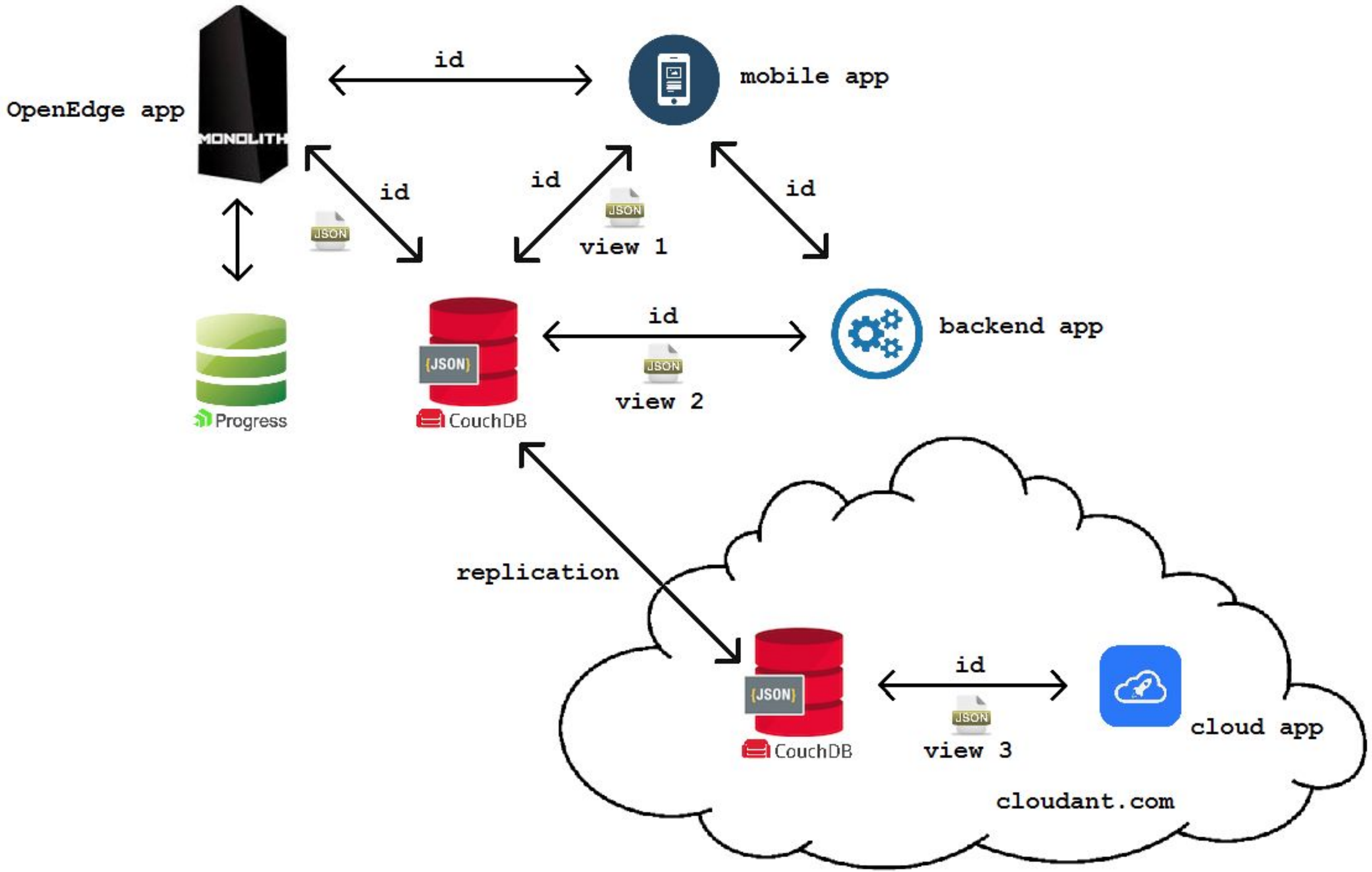
Don Beattie
Software Architect
Quicken Loans Inc.

Apache CouchDB has started. Time to relax.

Intro

The **OpenEdge RDBMS** is a great database that most of us work with on a daily basis to store our **relational data**. However it isn’t necessarily the best place to store and manage **JSON** messages. It's also more difficult to implement as a **distributed system**. Instead we might consider a **document-oriented** database.

Case Study System Diagram



What we’ll consider...

- The CouchDB
  - CAP Theorem
  - Locking vs Multi-Version Concurrency Control (MVCC)
  - Consistency between Multiple Database Servers
  - Eventual Consistency through Incremental Replication
- The Claim Check Design Pattern
- CouchDB RESTful API
- OOABL Classes for CouchDB
- Sample Calls to CouchDB from the AVM
- Demo (if we have time and the desire)
  - _utils
  - ABL Client

What is CouchDB?

CouchDB **doesn’t store data and relationships** in tables **like** a **relational database**, instead **each database** is a **collection** of queryable **documents**.

- Open Source
- **Document-Oriented**
- NoSQL Database
- Written in fault tolerant Erlang
- Clusters and **Replication**
- **High Availability**
- Uses JSON to Store Data
- **RESTful API**
- MapReduce
- Not Couchbase

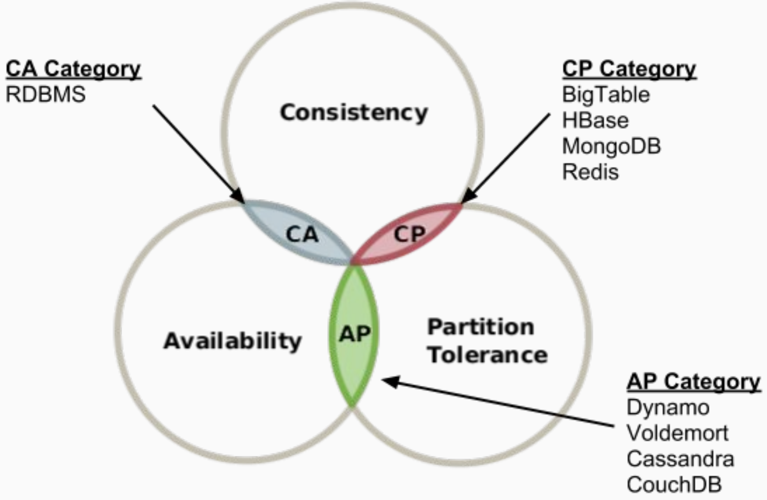
CAP Theorem

The CAP theorem states that any networked shared-data system can have at most two of three desirable properties (distributed systems):

- consistency (C) equivalent to having a single up-to-date copy of the data
- high availability (A) of that data (for updates)
- tolerance to network partitions (P)

CouchDB is **Availability** and **Partition Tolerant**.

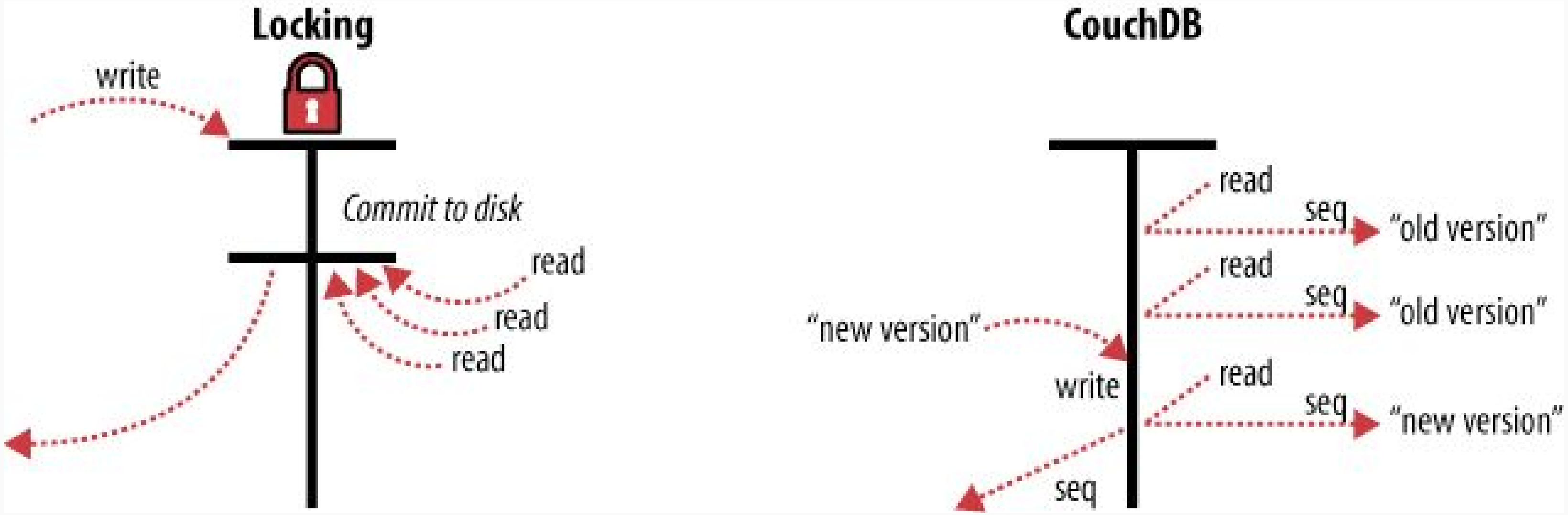
Pick Two...



Traditional Record Locking versus MVCC

In a **relational database,** to modify a table the RDMBS must ensure that nobody else is trying to update or read that row. A common way to handle that is with a **record lock**.

Instead of locks, **CouchDB** uses **Multi-Version Concurrency Control** (MVCC) to manage concurrent access to the database.

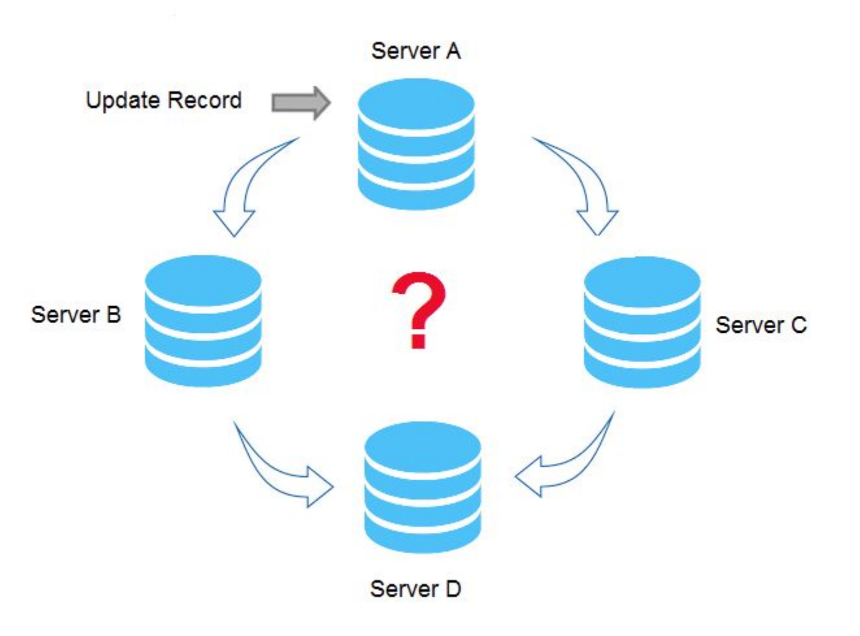


Consistency between Multiple Database Servers

Maintaining consistency within a single database node is relatively easy for most databases. The real problems start to surface when you try to maintain **consistency between multiple database servers**. If a update is done against Server A, how do we make sure additional servers are consistent. **With relational databases it is a very complex problem.**

Maintaining Consistency in a RDBMS

- Multi-primary
- Primary/replica
- Partitioning
- Sharding
- Write-through caches
- Other complex techniques

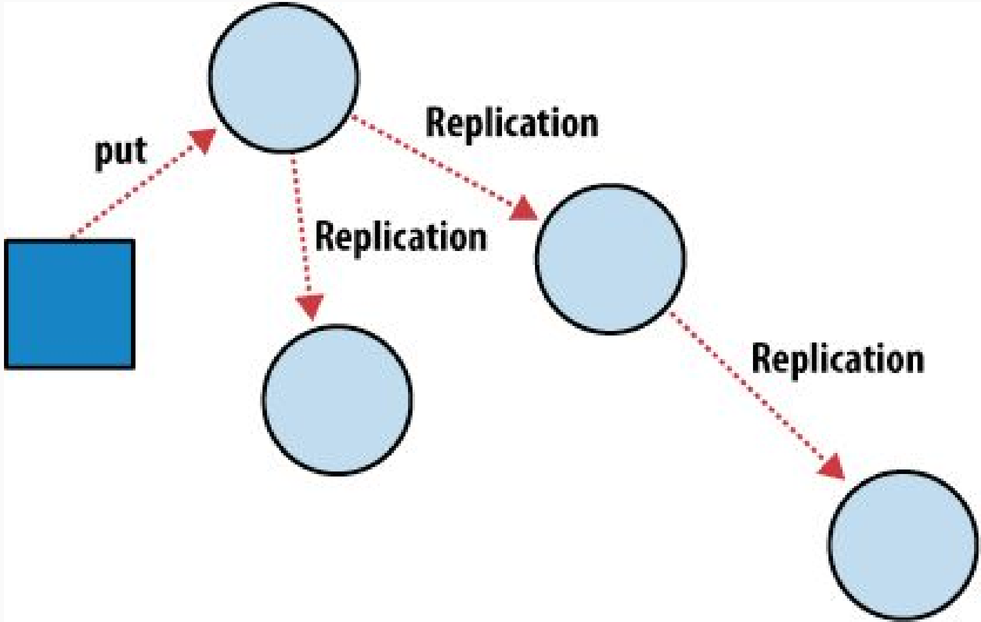


Eventual Consistency through Incremental Replication

When a**vailability is a priority over consistency**, updates can be performed against one node of the database without waiting for other nodes to come into agreement. If the **database knows how to take care of reconciling these operations** between nodes, we achieve **Eventual Consistency** in exchange for **high availability.**

A CouchDB achieves **Eventual Consistency** by using **Incremental Replication**.

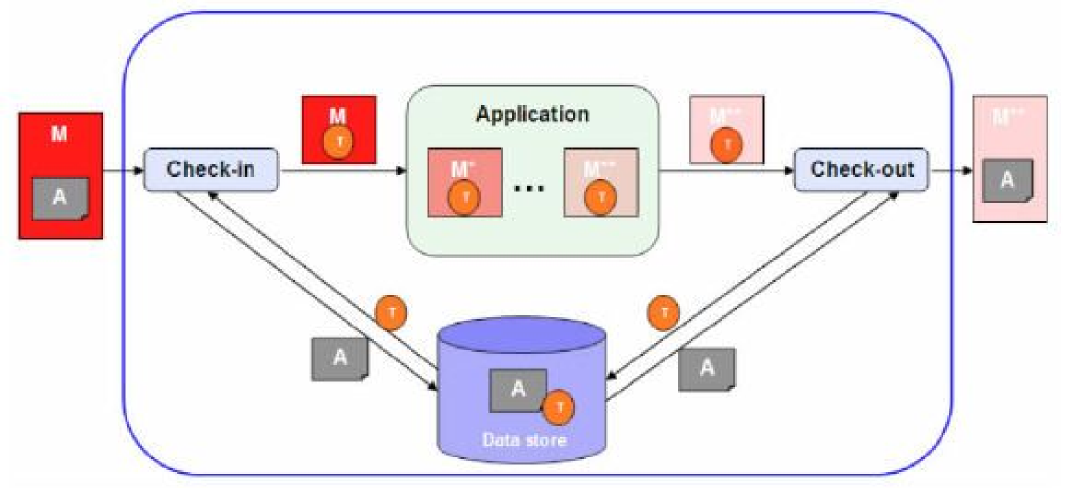
Incremental Replication is a process where **document changes** are **periodically copied between servers**. If there is a conflict, the newest wins, but the older conflict is also retained if needed later by some process.



The Claim Check Design Pattern

The idea behind Claim Check is simple:

- **Put away** or detach the **data** that your **application doesn’t need** by storing the data into some persistent data store.
- Let your application **run efficiently** with the **minimal data** that it requires.



- **When** finally there is **a need**, **retrieve** the **data** from the **persistent data store** before continuing on with processing.

CouchDB RESTful API

How do we integrate CouchDB with an OpenEdge application?

It's all about the RESTful API... Here is a small subset:

Create the *invoice* database:
PUT http://server/**invoice**

Retrive all databases:
GET http://server/all_dbs

Create an index on invoice:
PUT http://server/invoice/_index

```
{
  "index": {
      "fields": ["InvoiceNumber"]
   },
   "name" : "InvoiceNumber-index"
}
```

Create a document in the invoice database
PUT http://server/invoice/f1dc1b12-05d9-488e-2614

```
{
   "Invoice": [
   {
     "ID": "f1dc1b12-05d9-488e-2614",
     "InvoiceNumber": "ABCD1234", ...
```

Find a document in the invoice database
POST http://server/invoice/_find

```
{
   "selector": {
      "_id": "f1dc1b12-05d9-488e-2614-08114466b4f3"
   }
}
```

OOABL Classes for CouchDB

CouchDB.cls - The lowest level functionality (primitives) for communicating with any CouchDB database.

```
class abl.docstore.CouchDB:
  define private variable oHTTPClient  as abl.http.IHTTPClient no-undo.
  define private variable oJsonParsing as abl.json.JsonParsing no-undo.

  method public OpenEdge.Core.Collections.IStringCollection _all_dbs():
```

InvoiceDB.cls - Inherits CouchDBPrimitives to create high-level functionality for the **invoice** docstore.

```
&GLOBAL-DEFINE DatabaseName invoice
class abl.docstore.InvoiceDB
  inherits abl.docstore.CouchDB
  implements abl.docstore.IDocStore:

  { abl/docstore/dataset/dsInvoice.i }
```

ABL Code: Creating Document in CouchDB

```
// sampleCreateDocument.p
{ abl/docstore/dataset/dsInvoice.i }

define variable lcJson     as longchar   no-undo.
define variable cID        as character  no-undo.
define variable oInvoiceDB as abl.docstore.InvoiceDB no-undo.

oInvoiceDB = new abl.docstore.InvoiceDB().
dataset dsInvoice:write-json("longchar":u, lcJson, true, ?, ?, true).

cID = oInvoiceDB:CreateDocument(lcJson).

return.
finally:
  delete object oInvoiceDB no-error.
end finally.
```

ABL Code: Finding Document in CouchDB

```
// sampleFindDocument.p
{ abl/docstore/dataset/dsInvoice.i }

define variable oInvoiceDB as abl.docstore.InvoiceDB no-undo.

oInvoiceDB = new abl.docstore.InvoiceDB().

oInvoiceDB:Find('"_id": "f1dc1b12-05d9-488e-2614-08114466b4f3"':u,
                output dataset dsInvoice by-reference).

return.
finally:
  delete object oInvoiceDB no-error.
end finally.
```

Questions?

Resources

- http://couchdb.apache.org - CouchDB Home
- https://cloudant.com - CouchDB in the Cloud
- https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed - CAP Theorem
- http://www.enterpriseintegrationpatterns.com/patterns/messaging/StoreInLibrary.html - Claim Check

My Contact Information:
Don Beattie
donaldbeattie@quickenloans.com