Sander Dijkhuis
Interactive Institute
2013-06-03

About Couchm

Couchm is a server for collecting and querying sensor data.

It uses CouchDB and provides APIs for getting feed info, posting data, getting update notifications and querying data. The querying API is designed to resemble the Xively historical data API as it was before Cosm became Xively.

1. Why Couchm?

We needed a stable drop-in replacement for Cosm to use in the Instant Energy app for SP.

It is also used in other SP prototypes and might be a good solution for other projects.

2. About CouchDB

CouchDB

- has its source code available under an Apache 2.0 license,
- is a database system that stores data in JSON documents,
- supports attachments of any file format to these documents,
- is implemented in Erlang,
- uses JavaScript for queries, transformations and other operations,
- provides a MapReduce interface to create persistent JSON key - JSON value indices,
- starts building a MapReduce-based index as soon as it is requested,
- provides polling, long polling, continuous and EventSource feeds to updates,
- stores documents and indices as B-trees, allowing for quick retrieval (but ‘slow’ insertion),
- exposes all its functionality over an HTTP REST interface,
- provides basic HTTP or cookie authentication, and OAuth,
- provides per-database authorization,
- can work with SSL/TLS via HTTPS,
- stores databases as files that can be copied (while online) for backups,
- allows for easy instance-to-instance incremental replication over HTTP,
- provides eventual consistency with conflict resolution.

We chose CouchDB because

- it requires only a thin, integrated JavaScript layer to support Xively-like queries,
- it does not work with fixed schemas, so that we can extend or change the JSON structure,
- it provides high performance, data security and real-time updates out-of-the-box,
- it provides an HTTP API so that new prototypes can be implemented with just HTML + JS,
- it is easy to replicate all data over to a local CouchDB instance for experimenting,
- I know how to use it.

3. How does Couchm work?

Couchm provides a design document to a CouchDB database.

A design document is like any other JSON document, but identified with an { _id: "_design/<name>" } field. Other fields define functions and properties that influence how the database can be used.

The Couchm design document structure is:

```
{
  "_id": "_design/energy_data",
  "_rev": "125-6229369370ccb692ef006e35cd4d6eba",
  "updates": {
    "measurement": "function(doc, request) { … return [doc, 'status message']; }"
    ↳ Respond to POST requests, transforming the content to a document in the right JSON format.
  },
  "views": {
    "by_source_and_time": {
      "map": "function(doc) { … emit(couchm_key, values); … }",
      ↳ Build an index based with Couchm keys (see below). Only use docs duck-typed as measurements.
      "reduce": "function(keys, values, rereduce) { … return reduced_value; }"
      ↳ If multiple keys are combined (belonging to the same time interval), assign the last measured value.
    },
    "domains": {
      "map": "function(doc) { … emit(source, timestamp); … }",
      ↳ Build an index of all timestamps at which a measurement in 'source' was done.
      "reduce": "_stats"
      ↳ Make the built-in _stats function available to get the first and last timestamp for each key.
    }
  },
  "lists": {
    "interpolate_datastream": "function(head, request) { … while (row = getRow()) { … send(datapt); … } }",
    ↳ Loop over rows from by_source_and_time, write an array of values with evenly-spaced timestamps.
    "feeds_and_datastreams": "function(head, request) { … while (row = getRow()) { … send(feed); … } }"
    ↳ Loop over rows from a reduced by_source_and_time index, just write the feed names and metadata.
  },
  "shows": {
    "historical": "function(doc, request) { … return { code: 302, headers: { 'Location': url } }; }",
    ↳ Redirect from a Xively-like URL to a CouchDB query URL involving interpolate_datastream.
    "unix_to_couchm_ts": "function(doc, request) { … return { … body: couchm_timestamp; }; }"
    ↳ Convert a Unix timestamp to a Couchm key.
  },
  "rewrites": {
    {
      "from": "/feeds_and_datastreams",
      "to": "/_list/feeds_and_datastreams/by_source_and_time",
      "query": { "group_level": "1" }
    }
    ↳ Make the feeds_and_datastreams list available at a pretty URL.
  },
  "filters": {
    "measurements": "function(doc, request) { … return feed_matches && datastream_matches; }",
    ↳ Only lets through measurement documents, using optional feed and datastream parameters.
    "to_feed": "function(doc, request) { … return is_design_doc || feed_matches; }"
```

```
    ↳ Only lets through documents that should be replicated to a feed-specific database.
  },
  "_attachments": { "graphs.html": { … }, … }
  ↳ Browser-based apps that help test and demo the Couchm API.
}
```

It is located in the 'sp' database at http://aktivahuset.tii.se:8001/sp/_design/energy_data.

We use couchapp to generate the Couchm design document from couchm/app.js.

4. The Couchm key-value index

Measurement documents look like:

```
{
  "_id": "01085cc031d9fa97d1d208b60800f440",
  "_rev": "1-a69d266336709f80ee50a363f2fc204f",
  ↳ Automatically inserted by CouchDB.
  "type": "measurement",
  "user": "spsensors",
  "source": "room270",
  "timestamp": 1368431644621,
  ↳ Metadata required by Couchm: document type, author, source (= feed name), Unix timestamp.
  "PowerThreshold": "5",
  "ElectricEnergy": "120.3",
  "ElectricPower": "5.3",
  "OfficeOccupied": false
  ↳ A variable amount of datastream values, as strings or booleans. (CouchDB handles decimals badly.)
}
```

By the by_source_and_time view, they indexed by Couchm keys and value arrays. This document would be indexed as:

```
key:    ["room270", 15839, 1, 2, 1, 2, 2, 2, 2, 5, 1, 5, 1368431644621]
value:  ["2013-05-13T07:54:04.621Z", "5.3", false, null, "120.3"].
```

4.1. Couchm keys

A Couchm key consists of three parts: the feed name, a set of numbers the timestamp as the amount of milliseconds since the Unix epoch. The numbers describe the largest multiple of 86400 seconds that fits in the timestamp, the largest multiple of 43200 seconds, the largest multiple of 21600 seconds, etc.

In a CouchDB index, rows are ordered and grouped by their key according to collation rules that work well with JSON arrays.A group_level argument can be specified with view queries that 'flattens down' array keys, grouping together multiple rows. For example, group_level=4 will 'flatten down' the example key to:

```
key:    ["room270", 15839, 1, 2, 1]
```

so that it belongs to the same group as some other measurements within the same time interval.

If a query uses grouping, the reduce function will be applied to groups of rows. In this Couchm view, the reduce function reduces a sequence of values to the last measured value.

For example, to get a list of values at 0:00 GMT each day, set group_level=2. Then the row corresponding to

time t_1 is identified by a a [*feed name*, *amount of 86400 second intervals before* t_1]. The row's reduced value equals the last measurement within the interval (t_0, t_1].

4.2. Couchm values

A value in the Couchm index is an array containing the time of the measurement in JSON format and the datastream values. The order and datastream identifiers are fixed in by_source_and_time map function. The feeds_and_datastreams list returns a dictionary to look up which index belongs to which datastream.

If a new datastream is introduced, the by_source_and_time map function needs to be edited and the indices recreated. An alternative design option was to use more descriptive values in the index than arrays, but this would take more disk space and bandwidth.

For this reason, clients should request datastream indices using feeds_and_datastreams rather than keeping them as constants in code.

5. Open issues

- Couchm uses the same bad interpolation algorithm as Xively. Better options exist.
- When querying on the interval [t_0, t_1] with an n second sampling rate, the datapoint returned at t_0 currently only searches for measurements in [t_0, t_0] while using ($t_0 - n$, t_0] or any larger interval would increase the chances of a non-zero value to be returned.
- CouchDB databases accumulate some extra data that we don't need. A cron job could regularly compact databases, and perhaps automatically trigger view generation so that this doesn't start when clients query views.
- Couchm could use a graphical administration interface for debugging and setting up replication.

6. Getting and reading the code

Couchm is available at github.com/interactiveinstitute/couchm.

The annotated code is nicely readable in docs/app.html. Use Docco to update the HTML file if outdated.