Informatica Biomedica

lezione21

Alberto Paoluzzi Mauro Ceccanti
www.dia.uniroma3.it/ paoluzzi/web/did/biomed/

Informatica e Automazione, "Roma Tre" — Medicina Clinica, "La Sapienza"

May 17, 2010

Informatica Biomedica: Lezione 21

Apache CouchDB is a document-oriented database that can be queried and indexed in a MapReduce fashion using JavaScript

- CouchDB also offers incremental replication with bi-directional conflict detection and resolution

CouchDB provides a RESTful JSON API than can be accessed from any environment that allows HTTP requests

- There are myriad third-party client libraries that make this even easier from your programming language of choice
- CouchDB's built in Web administration console speaks directly to the database using HTTP requests issued from your browser

CouchDB is written in Erlang, a robust functional programming language ideal for building concurrent distributed systems

- Erlang allows for a flexible design that is easily scalable and readily extensible

Design

CouchDB is most similar to other document stores like MongoDB and Lotus Notes

- It is not a relational database management system
- Instead of storing data in rows and columns, the database manages a collection of JSON documents
- The documents in a collection need not share a schema, but retain query abilities via views
- Views are defined with aggregate functions and filters are computed in parallel, much like MapReduce

Document Storage

CouchDB stores documents in their entirety

- You can think of a document as one or more field/value pairs expressed as JSON
- Field values can be simple things like strings, numbers, or dates
- But you can also use ordered lists (arrays) and associative maps (associative array, hash, whatever your language may call them)
- Every document in a CouchDB database has a unique id and there is no required document schema

Map/Reduce Views and Indexes

To provide some structure to the data stored in CouchDB, you can develop views that are similar to their relational database counterparts

- In CouchDB, each view is constructed by a JavaScript function (server-side JavaScript by using CommonJS and SpiderMonkey) that acts as the Map half of a MapReduce operation
- The function takes a document and transforms it into a single value which it returns
- The logic in your JavaScript functions can be arbitrarily complex
- Since computing a view over a large database can be an expensive operation, CouchDB can index views and keep those indexes updated as documents are added, removed, or updated
- This provides a very powerful indexing mechanism that you get unprecedented control over compared to most databases

Distributed Architecture with Replication

CouchDB was designed with bi-direction replication (or synchronization) and off-line operation in mind

- That means multiple replicas can have their own copies of the same data, modify it, and then sync those changes at a later time
- The biggest gotcha typically associated with this level of flexibility is conflicts

-

RESTful API

CouchDB treats all stored items (there is more than documents) as a resource

- All items have a unique URI that gets exposed via HTTP
- REST uses the HTTP methods PUT, GET, POST and DELETE for the four basic CRUD (Create, Read, Update, Delete) operations on all resources
- HTTP is wildly understood, interoperable, scalable and proven technology
- A lot of tools, software and hardware, are available to do all sorts of things with HTTP like caching, proxying and load balancing

The MapReduce Algorithm

- MapReduce is a patented software framework introduced by Google to support distributed computing on large data sets on clusters of computers.
- The framework is inspired by map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as their original forms.
- MapReduce libraries have been written in C++, C, Erlang, Java, Python, Ruby, F, R and other programming languages.

Overview

Processing huge datasets on clusters

MapReduce is a framework for processing huge datasets on certain kinds of distributable problems using a large number of computers (nodes), collectively referred to as a cluster.

Computational processing can occur on data stored:

1. either in a filesystem (unstructured)
2. or within a database (structured).

"Map" step

The master node takes the input, chops it up into smaller sub-problems, and distributes those to worker nodes

- A worker node may do this again in turn, leading to a multi-level tree structure
- The worker node processes that smaller problem, and passes the answer back to its master node

"Reduce" step

The master node then takes the answers to all the sub-problems

- it combines them in a way to get the output
- returning the answer to the problem it was originally trying to solve

The advantage of MapReduce

The advantage of MapReduce is that it allows for distributed processing of the map and reduction operations

- Provided each mapping operation is independent of the other, all maps can be performed in parallel - though in practice it is limited by the data source and/or the number of CPUs near that data
- Similarly, a set of *reducers* can perform the reduction phase - all that is required is that all outputs of the map operation which share the same key are presented to the same reducer, at the same time

Efficient process?

While this process can often appear inefficient compared to algorithms that are more sequential, MapReduce can be applied to significantly larger datasets than that which "commodity" servers can handle

- A large server farm can use MapReduce to sort a petabyte of data in only a few hours
- The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation
- If one mapper or reducer fails, the work can be rescheduled — assuming the input data is still available

The *frozen part* of the MapReduce framework is a large distributed sort.

The *hot spots*, which are application dependent, are:

1. an input reader
2. a Map function
3. a partition function
4. a compare function
5. a Reduce function
6. an output writer

Input reader

The input reader divides the input into 16MB to 128MB splits and the framework assigns one split to each Map function

- The input reader reads data from stable storage (typically a distributed file system) and generates key/value pairs
- A common example will read a directory full of text files and return each line as a record

Map function

Each Map function takes a series of key/value pairs, processes each, and generates zero or more output key/value pairs

- The input and output types of the map can be (and often are) different from each other
- If the application is doing a word count, the map function would break the line into words and output the word as the key and "1" as the value

Partition function

The output of all of the maps is allocated to a particular reducer by the application's partition function

- The partition function is given the key and the number of reducers and returns the index of the desired reduce
- A typical default is to hash the key and modulo the number of reducers

Comparison function

The input for each reduce is pulled from the machine where the map ran and sorted using the application's comparison function.

Reduce function

The framework calls the application's reduce function once for each unique key in the sorted order

- The reduce can iterate through the values that are associated with that key and output 0 or more values
- In the word count example, the reduce function takes the input values, sums them and generates a single output of the word and the final sum

Output writer

The Output Writer writes the output of the reduce to stable storage, usually a distributed file system.

CouchDB uses a MapReduce framework for defining views over distributed documents and is implemented in Erlang.

The sequential subset of Erlang

- The sequential subset of Erlang is a general-purpose concurrent programming language and runtime system.
- is a functional language, with strict evaluation, single assignment, and dynamic typing. For concurrency it follows the Actor model. It was designed by Ericsson to support distributed, fault-tolerant, soft-real-time, non-stop applications. The first version was developed by Joe Armstrong in 1986.[1] It supports hot swapping thus code can be changed without stopping a system.[2]
- was originally a proprietary language within Ericsson, but was released as open source in 1998.

Erlang concurrent programming model

- While threads are considered a complicated and error-prone topic in most languages
- Erlang provides language-level features for creating and managing processes with the aim of simplifying concurrent programming
- Though all concurrency is explicit in Erlang, processes communicate using message passing instead of shared variables, which removes the need for locks

Philosophy

The philosophy used to develop Erlang fits equally well with the development of Erlang based systems. Quoting Mike Williams, one of the three inventors of Erlang:

- Find the right methods: Design by Prototyping
- It is not good enough to have ideas, you must also be able to implement them and know they work.
- Make mistakes on a small scale, not in a production project.

Erlang Tutorial

Erlang Tutorial