GUIT meeting 2018

Connecting LuaTEX To MongoDB

Roberto Giacomelli
giaconet.mailbox@gmail.com

Roma, 20 ottobre 2018

Archiviazione dati: perché NoSQL?

La crescita delle applicazioni web e più in generale l'esigenza di gestire in modo efficiente grandi collezioni di dati che cambiano rapidamente struttura, ha determinato il tentativo di superamento del modello relazionale SQL.

Punti di criticità dell'architettura SQL:

- data representation $\rightarrow$ strutture dati nei programmi;
- performance $\rightarrow$ specie nel contesto della rete;
- scale out $\rightarrow$ carico su un numero variabile di server;
- disponibilità $\rightarrow$ replicazione e ridondanza dei server;
- big data $\rightarrow$ dati, dati, e ancora dati.

Reporting: perché TEX?

Esaminiamo il caso di un report complesso, cioè un documento dove:

- i dati che contiene sono condivisi in rete e devono essere esatti;
- la sua struttura cambia anche in modo consistente da caso a caso anche in funzione dei dati stessi.

Questi due punti sono ricchi di implicazioni, ma la domanda è:

> *Quando nelle aziende come negli studi professionali devono essere prodotti report complessi, il sistema TEX può essere vincente?*

Di cosa parleremo

Ipotizziamo che l'archivio dati con cui costruire i report sia un database MongoDB (`www.mongodb.com`), uno dei più noti sistemi NoSQL, e che il motore di composizione sia LuaTEX.

Alcuni punti principali:

- come ottenere i dati MongoDB in LuaTEX?
- il codice Lua di gestione è più oppure meno efficace che non quello scritto per archivi di tipo SQL?

MongoDB: breve introduzione

In MongoDB l'unità base d'informazione è il *documento*.
Un documento è un insieme di chiavi/valori. Le chiavi devono essere stringhe e i valori devono appartenere a uno dei tipi dati consentiti (numeri, stringhe, date, array e documenti, ecc).
I valori possono essere documenti, così questi oggetti possono essere strutturati con qualsiasi grado di complessità (max 16 MB).
Per esempio:

```
{
    "_id" : ObjectId("5b7af5a589923a94e7ce4cb0"),
    "name": "Columbia River",
    "length_km" : 2000,
    "regions" : [
        "British Columbia",
        "Washington",
        "Oregon"
    ]
}
```

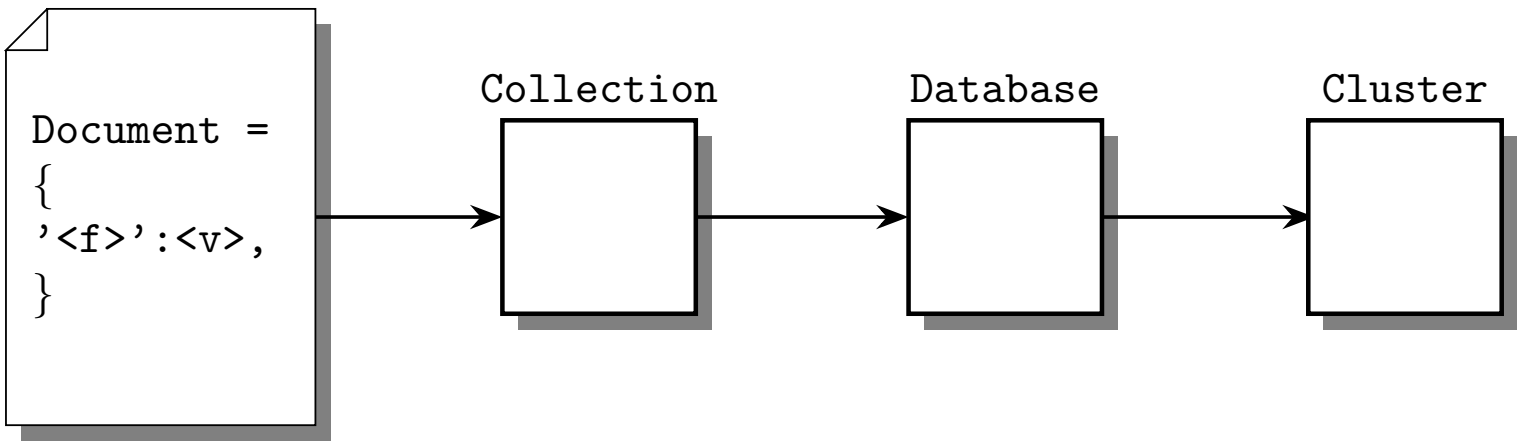
Documenti, collezioni, database, cluster in MongoDB

Il documento MongoDB corrisponde alla tupla in una tabella SQL, tuttavia *non* ci sono vincoli di schema.

L'unico obbligo è l'unicità del campo `_id` per i documenti all'interno di una stessa collezione, nient'altro.

I documenti sono raccolti in *collezioni* (raccolte equivalenti alle tabella SQL).

La struttura ad albero è qui raffigurata:



NoSQL → No Schema

Già dalle poche informazioni fornite, conseguono due importanti caratteristiche di MongoDB:

- l'assenza dello schema è una delle chiavi principali per risolvere le criticità degli archivi SQL, in particolare ciò consente di suddividere i dati in un numero variabile di server (lo *sharding*);
- l'unità di base dell'informazione, il documento, è una struttura flessibile adatta a rappresentare in modo strutturato dati di qualsiasi entità.

Tipologie di connessione dati in LuaTEX

Il flusso dei dati dal database verso il motore di composizione è:

- diretto;
- indiretto.

La connessione *diretta* consiste nell'esecuzione delle query da parte del motore di composizione, che perciò può essere solamente LuaTEX o LuaJITTEX, che a quel punto disporrà dei dati per comporre il report.

La connessione *indiretta* consiste nell'estrarre i dati, tradurli in uno specifico formato e nel memorizzarli in un file che più tardi potrà essere letto dal motore di composizione per produrre il report.

Connessione diretta a MongoDB

Occupiamoci solamente della connessione di tipo diretto che essenzialmente può essere implementata con:

- un connettore in puro Lua scritto secondo le specifiche di comunicazione binaria del server MongoDB, connesso con il protocollo di rete TCP;
- un binding di basso livello al driver MongoDB C `http://mongoc.org/`;
- un nodo intermedio (proxy) tra LuaTEX e il database MongoDB connesso via TCP usando la libreria `luasocket`, interna al compositore perché staticamente collegata all'eseguibile.

Il binding del progetto `lua-mongo`

Tra i progetti open source per il binding ho scelto `lua-mongo`, basato sul linguaggio C e conforme alle specifiche API di Lua in modo da offrire agli utenti un'interfaccia in stile Lua verso il driver Mongo C.

`https://github.com/neoxic/lua-mongo`

Dopo aver concluso con successo la compilazione dei sorgenti, avremo a disposizione il file binario della libreria a caricamento dinamico, che fornisce le funzionalità per connettere LuaTEX a MongoDB ed eseguire le query.

Connessione con lua-mongo

Ipotizzando che un server MongoDB sia attivo all'indirizzo `localhost`, per eseguire la connessione in LuaTEX, devono essere installati sia i binari del driver Mongo C, sia il file binario del binding nell'albero locale della TeX Live.
Una funzione di connessione può allora essere questa:

```
function Connect()
    local path = kpse.expand_var [[$TEXMFLOCAL]]
    if os.type == [[windows]] then
        path = path..[[/scripts/mongo/mongo.dll]]
    else
        path = path..[[/scripts/mongo/mongo.so]]
    end
    local _mongo = package.loadlib(path,"luaopen_mongo")
    local mongo = assert(_mongo())
    local client = assert(mongo.Client("mongodb://127.0.0.1"))
    return client
end
```

Esempio 1: il database river

Una collezione d'esempio riguardante i fiumi del Nord America. I documenti MongoDB hanno tre campi, il nome, la lunghezza e la lista delle regioni attraversate dal fiume:

```
{
    "name":"Missouri River",
    "length_km":3768,
    "regions":["Montana", "North Dakota", "South Dakota", ...ecc]
}, {
    "name":"Mississippi River",
    "length_km":3544,
    "regions":["Minnesota", "Wisconsin", "Iowa", "Illinois", ...ecc]
}, {
    "name":"Yukon River",
    "length_km":3185,
    "regions":["British Columbia", "Yukon Territory", "Alaska"]
},
...
```

Corrispondenza documento/tabella Lua

Essendo il documento MongoDB un oggetto chiave/valore, esso trova la sua naturale rappresentazione in Lua nel tipo tabella.

Così il fiume Missouri sarà rappresentato in Lua con la tabella:

```
{
    name = ”Missouri River”,
    length_km = 3768,
    regions = {”Montana”, ”North Dakota”, ”South Dakota”, ...ecc}
}
```

La corrispondenza non è esatta: in Lua non esiste il tipo array e al suo posto si usa ancora la tabella che di fatto è un tipo misto. Essa può essere sia un dizionario chiave/valore, sia un array, o anche entrambe le cose.
Dunque le regioni attraversate dal fiume sono memorizzate in una tabella a indici interi e non in un array come nel documento MongoDB.

Una funzione generale di query

Consideriamo questa funzione:

```
function FindIn(client, dbname, collection, query)
    local coll = client:getCollection(dbname, collection)
    local data = {}
    query = query or {}
    for doc in coll:find(query):iterator() do
       data[#data + 1] = doc
    end
    return data
end
```

Essa restituisce il risultato di una query eseguita sulla collezione `collection` del database `dbname`, presente sul server MongoDB rappresentato dall'istanza `client`.
La variabile `doc` sarà la tabella Lua che corrisponde a ciascun documento MongoDB estratto dalla query.

Corrispondenza query/tabella Lua

In MongoDB la query stessa è espressa da un documento. In particolare per trovare i fiumi di lunghezza superiore a 2000 km, si può usare la tabella Lua:

```
query = { length_km = { ["$gte"] = 2000 }}
```

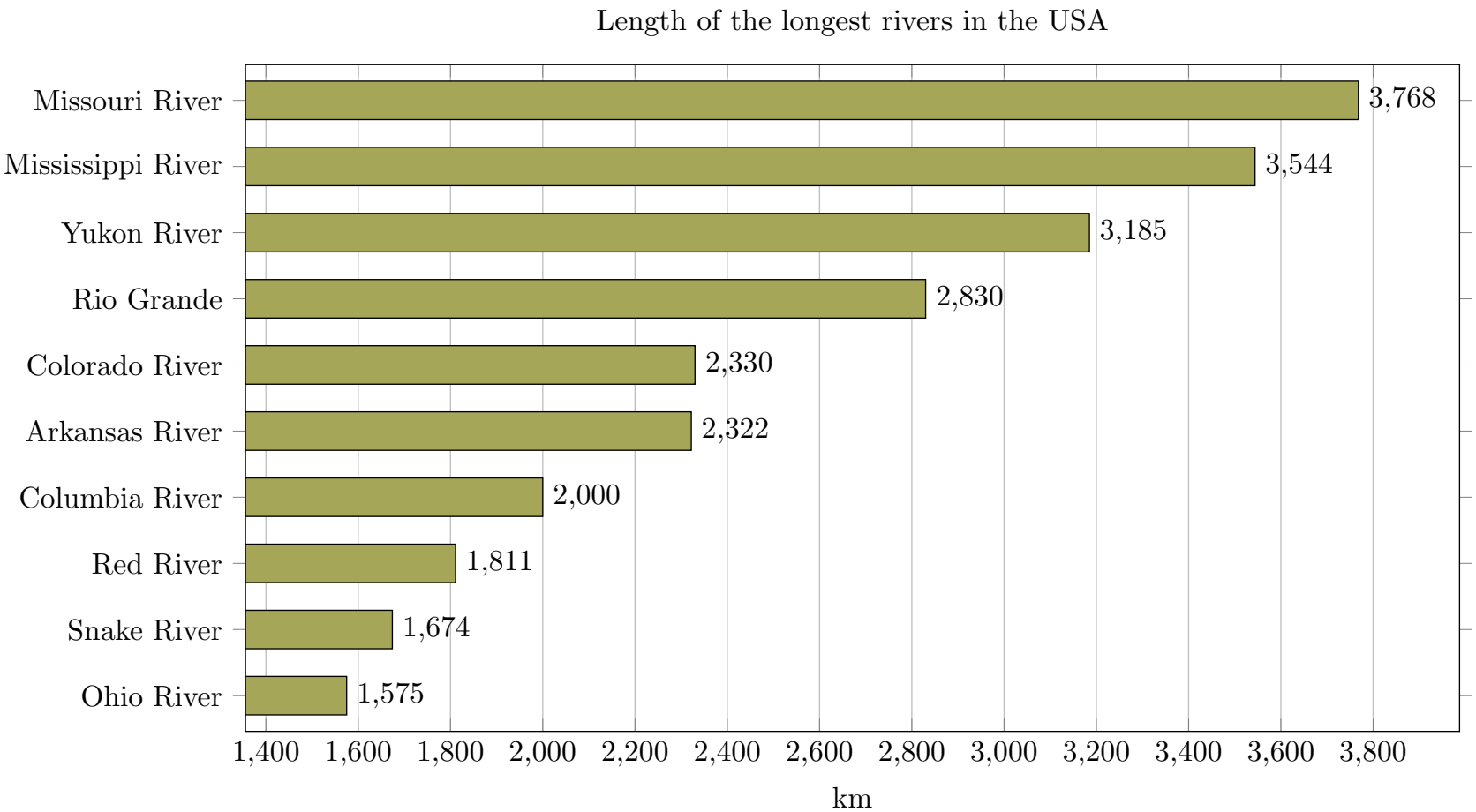
e quindi si chiama la funzione precedente così:

```
local res = FindIn(client, "river", "generalinfo",
    { length_km = { ["$gte"] = 2000 }}
)
```

La variabile `res` si riferirà alla tabella/array che contiene le tabelle relative ai fiumi *filtrati* dalla query.

Un istogramma sui dati dei fiumi

Un report, potrebbe contenere un grafico a istogrammi sulla lunghezza dei fiumi (costruito con il pacchetto pgfplots):



Una vista tabellare sui dati dei fiumi

oppure potrebbe contenere una vista tabellare su tre colonne, nome fiume, lunghezza, lista regioni attraversate:

River name	Length (km)	Regions
Missouri River	3768	Montana, North Dakota, South Dakota, Nebraska, Iowa, Kansas, Missouri
Mississippi River	3544	Minnesota, Wisconsin, Iowa, Illinois, Missouri, Kentucky, Tennessee, Arkansas, Mississippi, Louisiana
Yukon River	3185	British Columbia, Yukon Territory, Alaska
Rio Grande	2830	Colorado, New Mexico, Texas, Chihuahua, Coahuila, Nuevo León, Tamaulipas
Colorado River	2330	Colorado, Utah, Arizona, Nevada, California, Sonora, Baja California
Arkansas River	2322	Colorado, Kansas, Oklahoma, Arkansas
Columbia River	2000	British Columbia, Washington, Oregon
Red River	1811	Oklahoma, Texas, Arkansas, Louisiana
Snake River	1674	Wyoming, Idaho, Oregon, Washington
Ohio River	1575	Pennsylvania, Ohio, West Virginia, Indiana, Illinois, Kentucky

Una vista tabellare: codice

```
\directlua{
local libmongo = require ”libmongo”
local db = libmongo:Connect(”river”)
river = db:FindIn(”generalinfo”)
}
...
\begin{tabular}{lcp{120mm}}
\toprule
River name & Length (km) & Regions\\
\midrule
\directlua{local stop = string.char(92); stop = stop..stop
for _, r in ipairs(river) do
    tex.print(r.name); tex.print(”&”)
    tex.print(r.length_km); tex.print(”&”)
    tex.print(table.concat(r.regions, ”, ”)); tex.print(stop)
end
}
\bottomrule
\end{tabular}
```

Premi Nobel: modellare i documenti

Questo secondo esempio, offre l'occasione per inoltrarci nell'analisi e nella progettazione della struttura dei documenti di un database MongoDB.

Il database è composto da due collezioni:

- `laureates`,
- `prizes`,

La collezione dei premiati `laureates` include documenti semplici dove alle chiavi sono associati tipi non strutturati, mentre quella dei premi `prizes` è strutturata includendo altri documenti relativi. In fase di costruzione si possono sperimentare facilmente diverse strutture, non essendoci vincoli di schema.

Premi Nobel: modellare i documenti

Scopriamo la struttura del database `nobelprize` interrogando per prima la collezione `laureate`:

```
% !TeX program = LuaTeX
...
\directlua{
local libmongo = require ”libmongo”
local db = libmongo:Connect(”nobelprize”)
local fermi = db:FindOne(”laureate”, {
    firstname = ”Enrico”,
    surname = ”Fermi”
})
}
...
```

Premi Nobel: i laureati

Il connettore `lua-mongo` traduce il documento MongoDB di risposta alla query in una tabella Lua:

```
{
    _id = 46,
    firstname = ”Enrico”,
    surname = ”Fermi”,
    born = mongo.DateTime(”1901-09-29”),
    died = mongo.DateTime(”1954-11-28”),
    bornCountry = ”Italy”,
    bornCountryCode = ”IT”,
    bornCity = ”Rome”,
    diedCountry = ”USA”,
    diedCountryCode = ”US”,
    diedCity = ”Chicago, IL”,
    gender = ”male”
},
```

Premi Nobel: i premi Nobel

Mentre un premio Nobel è rappresentato da questo documento:

```
{   "year": 1938, "category": "physics",
    "laureates": [
      { "id_laureate": 46,
        "motivation": "for his demonstrations of the existence of new
         radioactive elements produced by neutron irradiation, and for
         his related discovery of nuclear reactions brought about by
         slow neutrons",
        "share": 1,
        "affiliation": [ {
            "name": "Rome University",
            "city": "Rome",
            "country": "Italy"
          }
        ]
      }
    ]
}
```

Premi Nobel: modello alternativo a tre collezioni

Modello alternativo a tre collezioni: non è ridondante ma richiede più query (e un modello con un solo tipo di collezione?):

```
{ // alternative prize model
    ”year”: 1938,
    ”category”: ”physics”,
    ”laureates”: [{
            ”id_affiliation”: 100, // reference
            ”motivation”: ”for his demonstrations of ...”,
            ”share”: 1,}]
}
{ // alternative affiliation model
    ”_id”: 100,
    ”id_laureate”: 46, // reference
    ”affiliation”: [{
            ”name”: ”Rome University”,
            ”city”: ”Rome”,
            ”country”: ”Italy”}]
}
```

Conclusioni

Conseguenze della tecnologia MongoDB:

- le query rendono i dati sotto forma di usuali tabelle Lua. Ciò rende il codice da scrivere più compatto e la forma dei dati più intuitiva rispetto a client SQL;
- è più semplice rispetto a SQL progettare e modificare la struttura del database;
- compilare e configurare le librerie di binding per il proprio sistema operativo può *non* essere semplice.

Risorse e ringraziamenti

Principali risorse sulle tecnologie trattate:

- **LuaTEX manual**: `> texdoc luatex`
- **MongoDB**: `https://docs.mongodb.com/`

Grazie per l'attenzione

e grazie mille Roma!

Domande?