MongoDB & Scala

Dessì Massimiliano

SpringFramework Meeting Roma
31 ottobre 2009

pro-netics group

Author

Software Architect and Engineer
ProNetics / Sourcesense

Presidente
JugSardegna Onlus

Fondatore e coordinatore
SpringFramework Italian User Group

Committer - Contributor
OpenNMS - MongoDB - MagicBox

Autore
Spring 2.5 Aspect Oriented Programming

pro-netics group

Context

Neo4j
Sesame
Graph databases
FreeBase
DirectedEdge
AllegroGraph

Hypertable
Cassandra
BigTable
HBase

MongoDB
CouchDB
Document databases
ThruDB
JackRabbit
Voldemort

Scalaris
Dynomite
TokioCabinet
Distributed key-value stores
Ringo
MemcacheDB

Hadoop
Disco
MapReduce
Skynet

pro-netics group

Document Database

Gli item sono dei Documenti

Non esistono join e transazioni su più documenti.

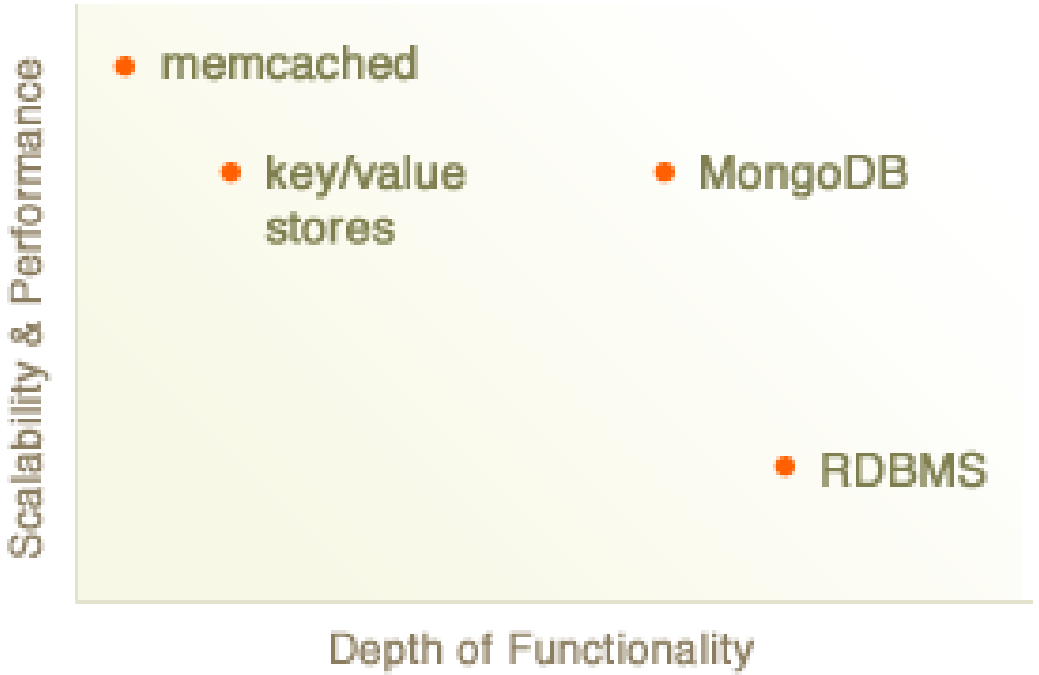
Un documento è formato da valori o liste in formato JSON o XML e viene lavorato sulla struttura di un documento.

Estrazione, indicizzazione aggregazione e filtraggio sono basati sugli attributi contenuti nel documento

Adatto alle applicazioni dove i dati sono indipendenti tra loro e non sono richieste join.

pro-netics group

Caratteristiche da 10000 piedi



pro-netics group

Mongo Production Deployments

http://www.mongodb.org/display/DOCS/Production+Deployments

SourceForge Mozilla Ubiquity Disqus

Business Insider Floxee

Silentale BoxedIce Defensio

TweetSaver ShopWiki MusicNation Detexify

Sluggy Freelance eFlyover soliMAP

Sifino Shapado @trackmeet

FetLife (NSFW) Stylesignal KLATU Networks Bloom Digital

MongoHQ Sailthru

(Oct 2009)

pro-netics group

Adatto

Siti web

High volume, low data

Alta scalabilita

Dati in formato JSON

Caching

Logging

Analisi real-time

pro-netics group

Non adatto

Sistemi altamente transazionali

Traditional Business Intelligence

Problemi che richiedono il SQL

pro-netics group

Caratteristiche

Built For Speed

Schema Free

Collection oriented storage: easy storage of object/JSON -style data

Dynamic queries and Indexes

Full index support, including on inner objects and embedded arrays

Query profiling

Replication and fail-over support

Efficient storage of binary data including large objects (e.g. photos and videos)

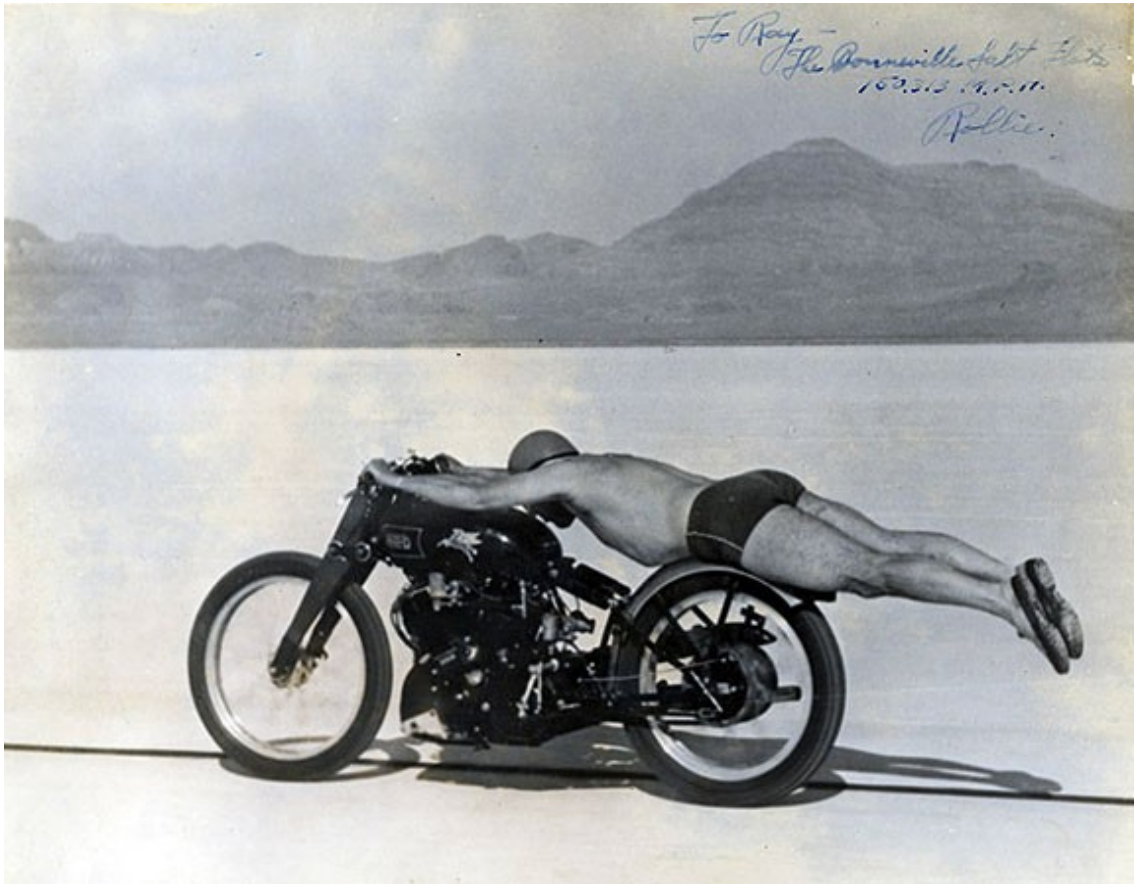
Auto-sharding for cloud-level scalability

Map/Reduce

Commercial Support, Hosting and Consulting Available

pro-netics group

Built For Speed



pro-netics group

How Fast is Mongo ?

Operazioni al secondo su un semplice MacBookPro con Driver Java

Task	Java	Task	
insert (small, no index)	12165	find (small, no index)	10940
insert (medium, no index)	19455	find (medium, no index)	10706
insert (large, no index)	1473	find (large, no index)	11737
insert (small, indexed)	98039	find_one (small, indexed)	12658
insert (medium, indexed)	121951	find_one (medium, indexed)	12690
insert (large, indexed)	1443	find_one (large, indexed)	12787
batch insert (small, no index)	30303	find range (small, indexed)	7204
batch insert (medium, no index)	26737	find range (medium, indexed)	4812
batch insert (large, no index)	106382	find range (large, indexed)	9765

L'elenco completo con i driver per i vari linguaggi:

http://www.mongodb.org/display/DOCS/Performance+Testing

pro-netics group

MongoDB Tools

- mongo – Shell javascript interattiva
- mongod - Server
- mongodump – Dump database
- mongoexport – Export collection in JSON, CSV
- mongofiles – Import GridFS DOCS
- mongoimport – Importa collection da JSON/CSV/TSV
- mongoimportjson – Importa collection da JSON
- mongorestore – Restore mongodump output
- mongos – Auto sharding
- mongosniff -Sniffer

pro-netics group

Schema Free

In un RDBMS la struttura dei dati è vincolata allo schema in cui definiamo tabelle con le colonne per meglio relazionare i dati.

In MongoDB lo “schema” viene definito dai dati quando vengono salvati dentro le collection

Raggruppiamo le entità della nostra applicazione in collections (Users, Centers...).

pro-netics group

Contenitori

RDBMS

- Dati raggruppati in tabelle
- Schema Pre Definito
-Tabelle Pre definite
- Campi indicizzabili predefiniti
- Relazioni predefinite

MongoDB

- Dati raggruppati in collections
- Collection create al momento del primo salvataggio
- Schema definito dai tipi dei campi contenuti dai singoli documenti al momento del salvataggio
- Campi indicizzabili
- Relazioni non predefinite

pro-netics group

Contenitori dati

RDBMS

- Dati salvati in righe
- Campi e tipi predefiniti
- Chiavi esterne predefinite
- Chiave primaria

MongoDB

- Dati salvati in Document
- Campi e tipi definiti al momento del salvataggio
- Sottodocumenti o reference non predefiniti
- _id field

Document JSON/BSON

Rappresentazione JSON salvataggio Binary-JSON


```
{ "_id" : "027b6e279578a64aa0684700" , "address-city" : "Ca" ,
"address-zipCode" : "09100" , "address-province" : "CA" ,
"address-region" : "Sardegna" , "address-country" : "Campidano" ,
"address-streetName" : "V.le Europe" , "address-streetNumber" : "4" ,
"telephone-contactMobile" : "3391234556" ,
"telephone-contactTelephoneHome" : "070123456" ,
"telephone-contactTelephoneWork" : "070987654" ,
"telephone-contactAcceptSms" : true ,
"userInfo-dateOfBirth" : "2009-09-08T15:30:30Z" ,
"userInfo-email" : "max@gmail.com" , "userInfo-name" : "Paperino" ,
"userInfo-surname" : "Paolino" , "userInfo-sensibleData" : "no sensible data" ,
"id" : "d37m3051128" , "description" : "descr" , "groupId" : "15" ,
"centerId" : "centerThree" , "_ns" : "centerUser"}
```


pro-netics group

Atomicità

Il motivo principale della mancanza delle transazioni è la lentezza e il costo dei lock in ambienti distribuiti.

Non essendoci le transazioni che coinvolgono più dati, qui abbiamo atomicità a livello di singolo documento con i seguenti comandi:

```
$set  $inc $push $pushAll $pull $pullAll
```

pro-netics group

Interrogazioni Avanzate

Nelle query abbiamo a disposizione anche :

`<, <=, >, >=, $ne, $in, $nin,`

`$mod, $all, $size, $exists`

Valori dentro Array

Valori dentro un oggetto embedded

Full language expressions con `$where`

`limit(), skip(), snapshot(), count(), group()`

pro-netics group

Mongo Driver

Mongo può essere utilizzato con i linguaggi più diffusi:

Java (Groovy, Scala, JRuby), PHP, Ruby, C++, Python,
Perl, C#, Erlang, Javascript server side.
Ogni driver predispone i nostri oggetti ad
essere utilizzati in Mongo
ed utilizza le “magic words” attese da mongo
per compiere le operazioni

Mongo Scala

Nel seguito vedremo l'utilizzo dei driver scala

realizzati da Alexander Azarov,

disponibili all' URL:

http://github.com/alaz/mongo-scala-driver

pro-netics group

Mongo Scala

Questi Mongo-Scala Driver wrappano le API
offerte dai driver Java ufficiali, e utilizzano il pattern del
Data Mapper (iBATIS) con la classe Shape
per mappare l'oggetto con il Document.
Con questi Driver possiamo utilizzare Mongo con Scala,
in maniera scalabile e sopratutto funzionale.
Vediamo il loro uso e successivamente
una breve introduzione a Scala

pro-netics group

Mongo Scala

La nostra classe che per ora contiene solo dei dati,

senza nessun comportamento.

```
case class CenterUserCommand(
                    @scala.reflect.BeanProperty var name: String,
                    @scala.reflect.BeanProperty var surname: String,
                     @scala.reflect.BeanProperty var email: String,
                     @scala.reflect.BeanProperty var id: String,
                     @scala.reflect.BeanProperty var description: String)
```

Il compilatore genererà i set i get dei parametri passati,

il costruttore con quei parametri, oltre al metodo toString,

hashCode e equals.

pro-netics group

Mongo Scala

Ora vediamo cosa dobbiamo definire dentro il nostro oggetto perchè i driver scala-mongo possano creare il nostro oggetto da un oggetto com.mongodb.DBObject

```
....
object CenterUserCommand extends AbstractShape[CenterUserCommand] {

   override def factory(dbo: DBObject): Option[CenterUserCommand] =
    Some(new CenterUserCommand(dbo.get("name").toString,
                               dbo.get("surname").toString,
                               dbo.get("email").toString,
                               dbo.get("id").toString,
                               dbo.get("description").toString))
..
}
```

pro-netics group

Mongo Scala

Ora vediamo il resto della classe dove sono definiti i campi che possono essere aggiornati, nel nostro caso tutti quelli che compongono l'oggetto

```
....
object id extends Scalar[String]("id", _.id) with Updatable[String] {
    override def update(x: CenterUserCommand, id: String) {x.id = id}
}

object name extends Scalar[String]("name", _.name)with Updatable[String]
{
    override def update(x: CenterUserCommand, name: String) {
          x.name = name
   }
}

..
}
```

pro-netics group

Mongo Scala

Ora vediamo come attraverso i driver ci colleghiamo con Mongo.

Crediamoci una Classe Datastore

```
import com.mongodb.{DB, DBAddress, Mongo}

class Datastore(val url: String, val port: Int, val dbName: String) {

  private val address = new DBAddress(url, port, dbName)
  private val mongo = new Mongo(address)
  private val db = mongo.getDB(dbName)

  def getConnection(): DB = db

  def getMongo(): Mongo = mongo
}
```

pro-netics group

Mongo Scala

Ora passiamo il Datastore al costruttore della nostra classe repository che parla con una determinata collection.

```
class CenterUserRepository(val datastore:Datastore) {

// leggiamo la collection di questo repository
var collection= datastore.getConnection.getCollection("usersCollection")

// convertiamo attraverso i driver in una scala collection
val users = collection of CenterUserCommand

...
```

pro-netics group

Mongo Scala

Ora vediamo i metodi

```
// insert or update
def saveUser(user:CenterUserCommand) = users +=  user


def deleteUser(user:CenterUserCommand) = users -= user


def retieveUserByEmail(email:String) = (
   CenterUserCommand
      where {CenterUserCommand.email eq_? email}
      take 1 in users
   ).firstOption


def numberUsers() = users.size


def retrieveUsersPaginated(usersOnPage:Int, page:Int) = {
for {val user <- CenterUserCommand
      drop page*usersOnPage take usersOnPage in users
    }  yield user
}
```

pro-netics group

Mongo Scala

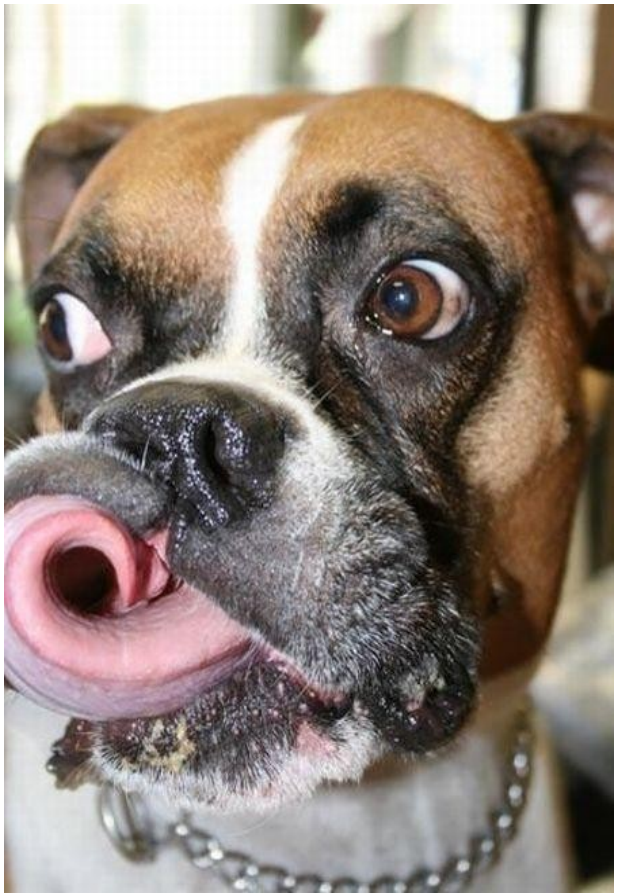
Ora modifichiamo la classe repository per gestire la chiamata ai metodi in modo “transazionale” con l' Execute Around Method pattern

```
class CenterUserRepository private(val datastore:Datastore) {
  val connection = datastore.getConnection
  connection.requestStart

   private def cleanUp() { connection.requestDone }
  //others methods
  …

object CenterUserRepository {
 def use(datastore:Datastore, codeBlock: CenterUserRepository => Unit) {
    val repo = new CenterUserRepository(datastore)
    try {
      codeBlock(repo)
    }
    finally {
      repo.cleanUp()
    }
  }
}
```

pro-netics group

Scala

Ora una mini introduzione su Scala



pro-netics group

Funzionale vs Imperativo

I linguaggi funzionali dicono

cosa deve fare il programma

Java e i linguaggi Object Oriented

sono imperativi,

dicono ***come*** deve farlo

Scala

Scala può essere utilizzato sia come linguaggio di scripting, sia come linguaggio ad alta concorrenza

Possiede le caratteristiche di un linguaggio dinamico

Con la velocità di un linguaggio *statico* e *tipizzato*, grazie alla inferenza si può scrivere codice molto più coinciso

pro-netics group

Function value

Possiamo passare una funzione come parametro

```
def store(arr: Array[Int], start: Int, funzione: (Int, Int) => Int) :
Int = {
    var precedente = start
    arr.foreach(element => precedente = funzione(precedente, element) )
    precedente
}
```

Passando funzioni anonime diverse possiamo avere:

```
scala> val array = Array(2, 3, 5, 1, 6, 4, 13)

scala> val somma = store(array, 0, (precedente, elem) => precedente + elem)
somma: Int = 34

scala> val max = store(array, Integer.MIN_VALUE, (precedente, elem) =>
Math.max(precedente, elem))
max: Int = 13
```

Function value

Possiamo anche ordinare meglio riusando le funzioni

```
def somma(precedente: Int, elem : Int) = { precedente + elem }

def max(precedente: Int, elem : Int) = { Math.max(precedente, elem)}


scala> val sum = store(array, 0, somma)
sum: Int = 34

scala> val massimo = store(array, Integer.MIN_VALUE, max)
massimo: Int = 13
```

pro-netics group

Scala Concurrency

Scala oltre ad essere coinciso e avere le caratteristice di linguaggio funzionale ha anche altre interessanti caratteristiche. Una delle più importanti è l'utilizzo semplificato della concorrenza.

pro-netics group

Actors

Gli actors sono unità di esecuzione

threadless e stackless

che processano messaggi (eventi)

in maniera seriale.

Un actor riceve i messaggi nella sua mailbox,

li processa uno alla volta

in maniera asincrona

prendendoli dalla mailbox.

pro-netics group

Actors

Un actor non esponendo nessuno stato

e venendo modificato o interrogato

attraverso i messaggi,

che sono processati in maniera seriale

non ha bisogno di lock sul suo stato interno

ed è perciò thread safe

pro-netics group

Scala & MongoDB

Ora è evidente il vantaggio di avere un linguaggio che
nasce con la scalabilità come punto di forza unito ad
un sistema di persistenza ad alte prestazioni come MongoDB.
Fino ad ora era necessario utilizzare Java
con il suo modello di gestione della concorrenza
valido ma non semplice da gestire
abbinato a sistemi di persistenza SQL
che sono sempre stati il
collo di bottiglia rispetto alle applicazioni.

pro-netics group

Domande ?

pro-netics group

Grazie per l'attenzione !

Massimiliano Dessì
desmax74 at yahoo.it
massimiliano.dessi at pronetics.it

http://twitter.com/desmax74

http://jroller.com/desmax

http://www.linkedin.com/in/desmax74

http://www.slideshare.net/desmax74

http://wiki.java.net/bin/view/People/MassimilianoDessi

http://www.jugsardegna.org/vqwiki/jsp/Wiki?MassimilianoDessi

pro-netics group