Università degli studi di Messina
Dipartimento di Matematica e Informatica

Progetto Database 2

Realizzare un database basato su MongoDB di un Hospital Information System (HIS)

Studenti: Pafumi Francesco, Gangemi Salvatore

Matricole: 446322, 443193

A.A 2015/2016

Professore: Antonio Celesti

INDICE

Introduzione…….…….…….…….…….…….…….…3

Scenario di riferimento…….…….…….…….……...…….4

Ambiente di sviluppo...…….…….…….…….…….……..6

Architettura applicazione…….……...….…….…….……...…….7

Implementazione …….…….…….…….…….…….…….8

Esperimenti…….……..……....…….……...….…….12

Conclusione …….…….…….……..…….…….15

1. Introduzione

Si richiede la realizzazione di un progetto che confronti le performance di diversi DBMS NoSql su uno schema preesistente e comune. Le performance andranno ad essere analizzate effettuando un set di stesse query su tutti i DBMS. Il dataset è stato concordato tra i vari gruppi e generato casualmente. Sono stati generati diversi dataset rispettivamente da 10, 100, 1000, 10000 e 100000 record. Il lavoro è stato suddiviso in gruppi per ogni database che consistono in:

- Cassandra
- MongoDB
- Neo4J
- Hbase

Si richiede infine di rappresentare i risultati per mezzo di istogrammi andando in seguito ad analizzare un intervallo di confidenza del 95%.

Il DBMS in questione è MongoDB.

2. Scenario di riferimento

MongoDB è un DBMS non relazionale orientato ai documenti.
MongoDB si allontana dalla tradizionale struttura basata su tabelle dei database relazionali in favore dei documenti stile JSON con schema dinamico, il che rende l'integrazione dei dati di alcuni tipi di applicazioni più facile e veloce.

Caratteristiche principali:

- Query ad hoc: MongoDB supporta ricerche per campi, intervalli e regular expression. Le query possono sia restituire campi specifici del documento che includere funzioni definite dall'utente in javaScript
- Indicizzazione: qualunque campo in MongoDB può essere indicizzato. Sono disponibili anche indici secondari, unici, sparsi, geospaziali e indici full text.
- Alta affidabilità: MongoDB fornisce alta disponibilità e aumento del carico gestito attraverso un replicaSet che consiste in una o più copie dei dati.
  Ogni replica può avere il ruolo di copia primaria o secondaria. La replica primaria effettua tutte le scritture e le letture, mentre le repliche secondarie mantengono copie di quella primaria attraverso un meccanismo di replicazione. Quando la replica primaria fallisce, il replicaSet inizia un processo di elezione per la copia primaria.
- Sharding e bilanciamento dei dati: MongoDB scala orizzontalmente usando la frammentazione (sharding), però l'utente deve scegliere una chiave di frammentazione, che determina come i dati di una collection saranno distribuiti tra i vari nodi.
  MongoDB include inoltre un meccanismo di bilanciamento dei dati spostando gli intervalli dei dati da un frammento all'altro in modo da bilanciare la distribuzione dei dati.

MongoDB rappresenta i documenti JSON in codice binario chiamato BSON, che estende il modello JSON stesso ed inolter prevede dati addizionali ed è più efficiente per encodarlo e decodificarlo.

3. Ambiente di sviluppo

L'applicazione è stata implementata utilizzando il linguaggio di programmazione Python e MongoDB come DBMS.
L'Ide utilizzato per la stesura del codice è PyCharm.
Il tutto è stato implementato e testato su piattaforma linux (Debian).
Sono state utilizzate le seguenti librerie:

- matplotlib: libreria grafica di python;
- pymongo: libreria/driver per interagire con MongoDB attraverso il codice python;
- json: libreria per la gestione dei file json.

4. Architettura applicazione

L'architettura dell'applicazione è stata strutturata cercando di massimizzare la modularità e la riusabilità.

L’intero progetto è stato suddiviso in 3 files Python:

- query.py: contiene tutte le query per eseguire i test;
- utility.py: contiene varie funzioni necessarie per eseguire i test, tra cui caricaDatiDaJson che è una funzione che legge un file json dal file system;
- main.py: è la parte principale del software; in essa vi è tutto il codice che esegue i vari test compresa la creazione dei vari grafici.

5. Implementazione

Per quanto riguarda l'implementazione di seguito sono riportati gli elementi che compongono la progettazione e il modo in cui sono stati collegati fra di loro.

- Connessione al DBMS:
  Per l'utilizzo di MongoDB tramite Python è stata utilizzata la libreria pymongo, che è in grado di sincronizzare e permettere un utilizzo più efficiente e veloce di MongoDB e della sua architettura.
  Tramite questo codice è possibile connettersi al server specificando indirizzo e porta.

```
client = MongoClient("localhost", 27017)
db = client.test
```

- Importazione del dataset:
  Per andare ad effettuare i test sui vari tipi di dataset composti da diversi elementi è stato utilizzato il seguente metodo che, dato il path, permette il facile import del file JSON.

```
def caricaDatiDaJson(jsonFile):
    with open(jsonFile) as data_file:
        data = json.load(data_file)
        return data
```

- Query per l'inserimento dei dati (queryInserisciDati):
  Tramite il metodo queryInserisciDati è stato possibile andare a popolare il database. Passando infatti il file JSON dei pazienti è possibile effettuare un parse dei dati inserendoli negli appositi campi.

```
def queryInserisciDati(db,id,name,width,height,l_shank,l_thigh,lokomat_shank,lokomat_thigh,
                       lokomat_recorded,version,legtype,lwalk_training_duration,lwalk_distance,step_datas):
    collection = db.Patients
    collection.insert({
        "id" : id,
        "name" : name,
        "width" : width,
        "height": height,
        "l_shank" : l_shank,
        "l_thigh" : l_thigh,
        "lokomat_shank" : lokomat_shank,
        "lokomat_thigh" : lokomat_thigh,
        "lokomat_recorded" : lokomat_recorded,
        "version" : version,
        "legtype" : legtype,
        "lwalk_training_duration" : lwalk_training_duration,
        "lwalk_distance" : lwalk_distance,
        "step_datas" : step_datas
    })
```

- Eseguire le Query:
  Per l'esecuzione delle query si è pensato di andare a creare una lista con tutte le query al suo interno e per ogni iterazione eseguire la query corrente.
  Per ogni query viene utilizzata la connessione e la libreria 'Datetime' per poter ricavare i tempi d'interrogazione. Il metodo utilizzato per interrogare il database è 'find()' a cui è possibile passare i campi da dove verranno presi i dati, dei parametri per poter filtrare i dati. Le query che sono state scelte sono le seguenti:

  - primaQuery: ricava tutti i dati di tutti i pazienti

```
def primaQuery(db):
    collection = db.Patients
    patients = collection.find({})
    return patients
```

- secondaQuery: ricava tutti i dati dei pazienti che hanno uno specifico nome

```
def secondaQuery(db):
    collection = db.Patients
    patients = collection.find({'name' : 'KRVYRSKX7'})
    return patients
```

- terzaQuery: ricava tutti i dati dei pazienti che hanno un 'lwalk_training_duration' minore a 2400, una 'width' diversa da 0.80 e che hanno effettuato almeno 5 misurazioni.

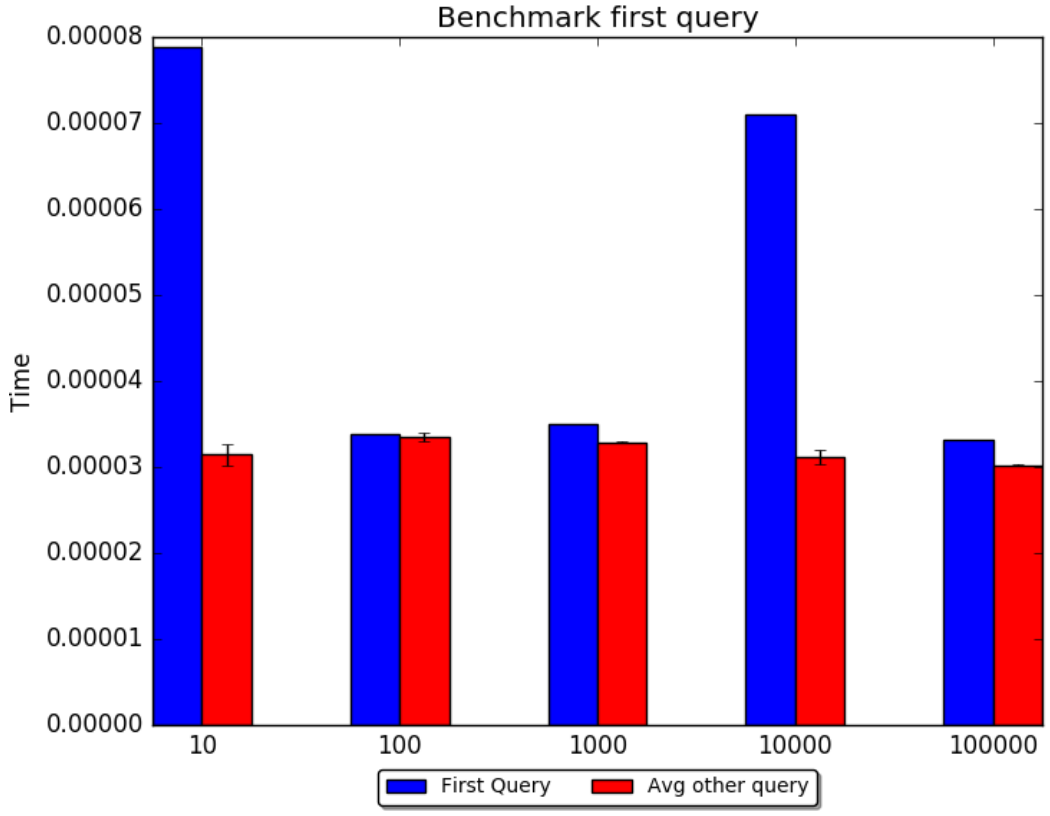
```
def terzaQuery(db):
    collection = db.Patients
    patients = collection.find({'lwalk_training_duration' : { '$lt' : 2400} , 'width' : { '$ne' : 0.80 }, 'step_datas' : { '$size' : 5} })
    return patients
```

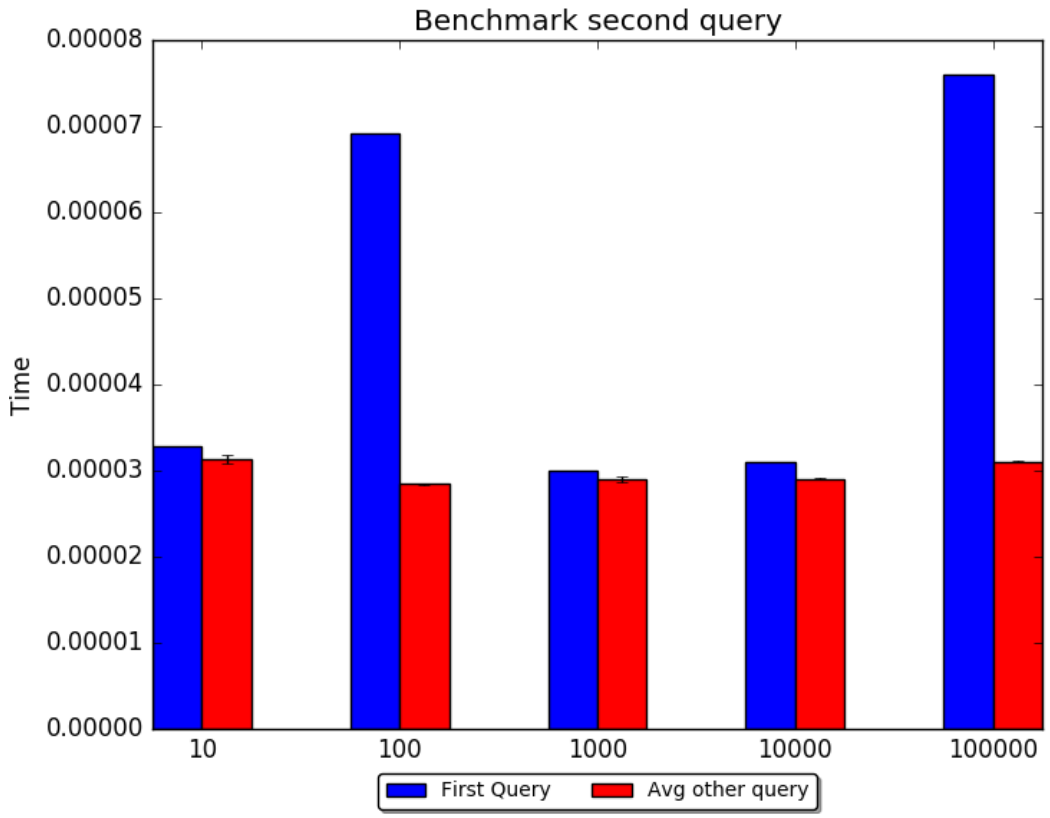
- Main

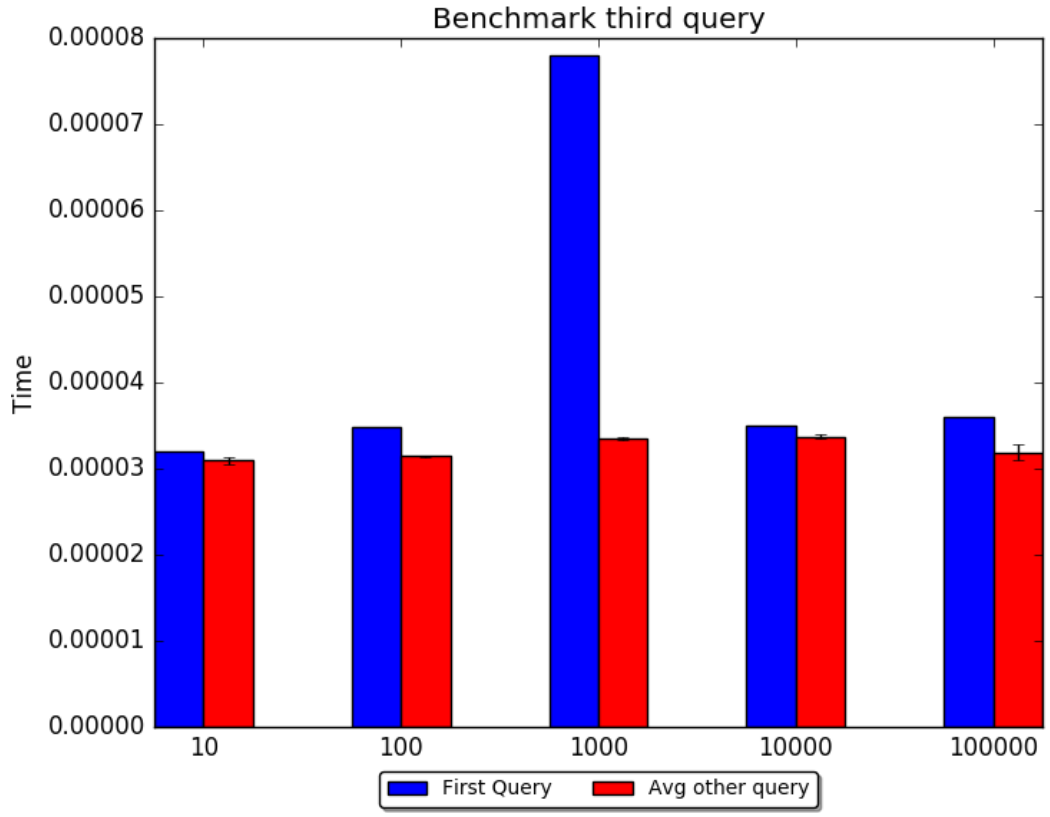
```
listaDataSet = ['ds10.json','ds100.json','ds1000.json','ds10000.json','ds100000.json']
listaDatiPrimaQuery = []
listaTempiOtherQuery = []
listTitoli = ['Benchmark first query', 'Benchmark second query', 'Benchmark third query']
#listaDataSet = ['ds10.json','ds100.json']

directory = "result"
if not os.path.exists(directory):
    os.makedirs(directory)

listaQuery = [primaQuery, secondaQuery, terzaQuery]
tempototaleIniziale = time.time()
#Eseguo il tutto per ogni dataset
for dataset in listaDataSet:
    queryEliminaCollection(db)
    data = utility.caricaDatiDaJson("../../../dataset_lokomat/" + dataset)
    #Inserisco dataSet
    for i in data:
        queryInserisciDati(db,i['id'],i['name'],i['width'],i['height'],i['l_shank'],i['l_thigh'],i['lokomat_shank'],i['lokomat_thigh']
                           ,i['lokomat_recorded'],i['version'],i['legtype'],i['lwalk_training_duration'],i['lwalk_distance']
                           ,i['step_datas'])
    #Eseguo query
    contatoreQuery = 0
    for query in listaQuery:
        #Stessa query 31 volte
        sommaTempi = 0
        tempiOtherQuery = []
        for _ in range(0, 31):
            tempo = time.time()
            query(db)
            tempoFinale = time.time() - tempo
            if (_ == 0):
                listaDatiPrimaQuery.append(tempoFinale)
            else:
                tempiOtherQuery.append(tempoFinale)
        listaTempiOtherQuery.append(tempiOtherQuery)
#Grafici
i = 0
for __ in range(0, 3):
    numDataset = 0
    for _ in range(0,5):
        print i

        istogrammaPrimaQuery = plt.bar(numDataset, (listaDatiPrimaQuery[i]), 0.5, color='b')

        #dati per la confidenza d'intervallo del 95%
        mean = np.mean(listaTempiOtherQuery[i])
        stddev = np.std(listaTempiOtherQuery[i])
        conf = 0.95 * (stddev / np.math.sqrt(len(listaTempiOtherQuery[i])))

        numDataset = numDataset + 0.5
        istogrammaOtherQuery = plt.bar(numDataset, mean, 0.5, color='r', yerr=conf, ecolor='black')
        numDataset = numDataset + 1.5
        i = i + 1
    plt.ylabel('Time', fontsize=12)
    plt.xlabel('Dataset', fontsize=12)
    plt.title(listTitoli[__])
    plt.xticks([0.5, 2.5, 4.5, 6.5, 8.5], ['10', '100', '1000', '10000', '100000'], rotation='horizontal')
    fontP = FontProperties()
    fontP.set_size('small')
    plt.legend([istogrammaPrimaQuery, istogrammaOtherQuery], ('First Query', 'Avg other query'), prop=fontP, loc='upper center',
               bbox_to_anchor=(0.5, -0.05), fancybox=True, shadow=True, ncol=2)
    plt.savefig('result/' + listTitoli[__] + '.png')
    plt.clf()
```

6. Esperimenti

I risultati ottenuti dalla simulazione sono stati studiati attraverso degli istogrammi.
Ogni query viene eseguita 31 volte al fine di poter studiare il fenomeno di caching. Per ogni query troviamo un istogramma che rappresenta per ogni dataset il tempo di esecuzione della prima volta e la media del tempo di esecuzione delle restanti 30.




Benchmark second query
Time
0.00008
0.00007
0.00006
0.00005
0.00004
0.00003
0.00002
0.00001
0.00000
10
100
1000
10000
100000
First Query
Avg other query

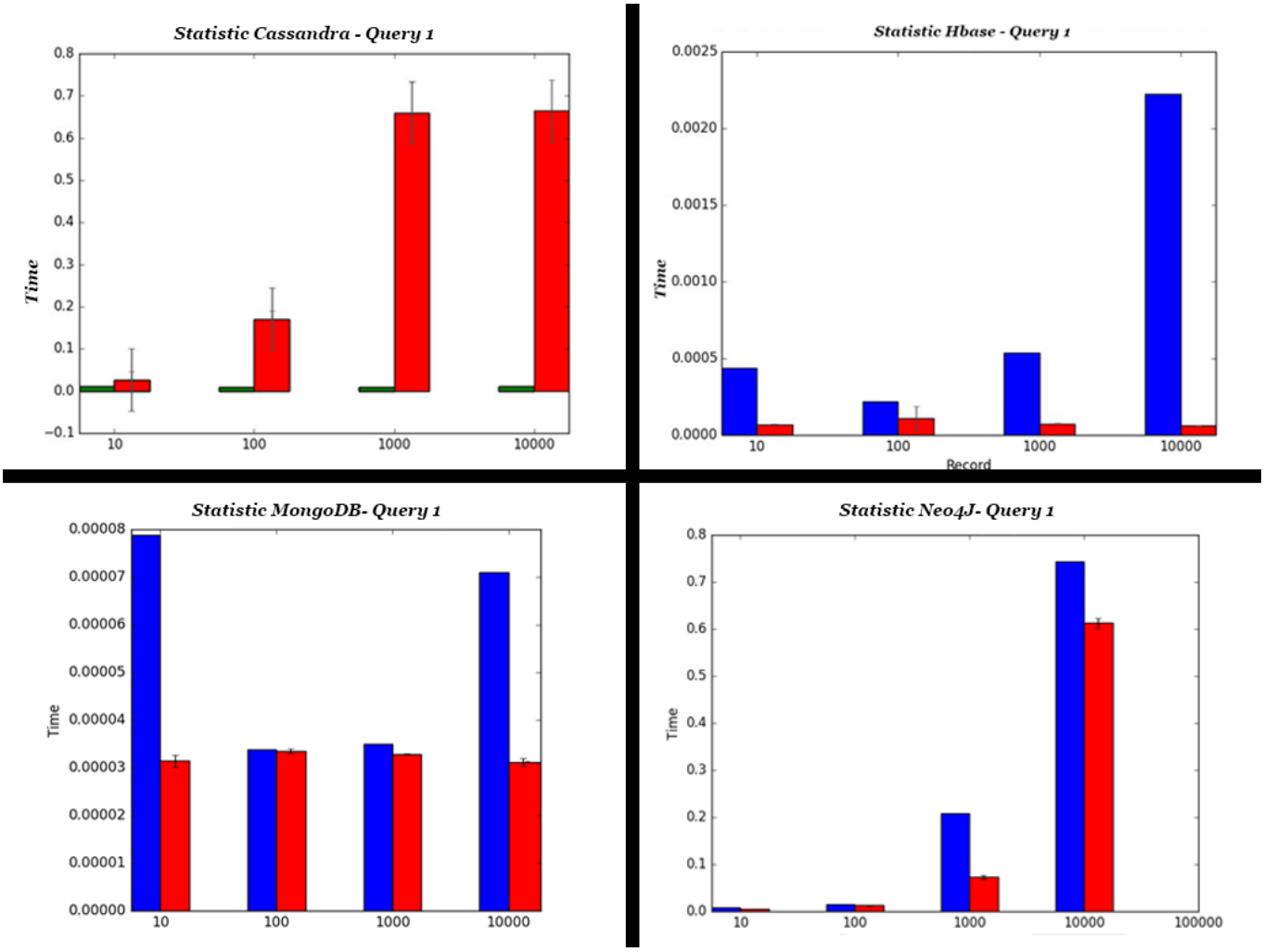

Benchmark third query
Time
0.00008
0.00007
0.00006
0.00005
0.00004
0.00003
0.00002
0.00001
0.00000
10
100
1000
10000
100000
First Query
Avg other query

Come è possibile visualizzare dai grafici, MongoDB utilizza molto la cache per andare a salvare i risultati precedentemente ottenuti.
Dai grafici risulta che i tempi d'interrogazioni sono molto brevi e grazie al sistema di caching la media dei tempi delle ultime 30 query è minore rispetto alla prima.

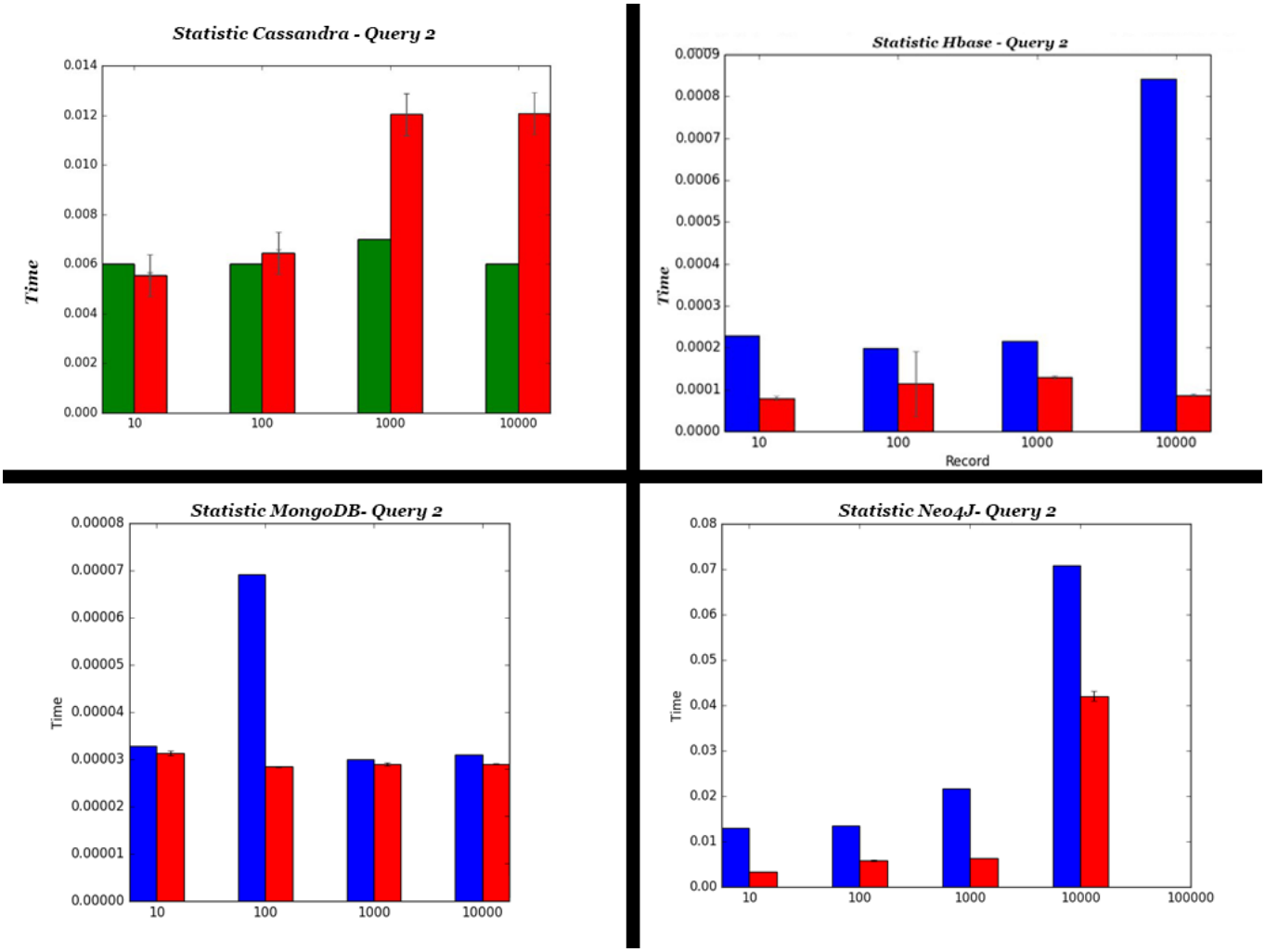
7. Conclusioni

In conclusione possiamo affermare, dopo aver confrontato i vari grafici che riportano i tempi di esecuzione delle diverse query eseguite su tutti e quattro i tipi di DBMS noSql, che MongoDB è risultato essere il più veloce tra i vari DBMS confrontati.
Possiamo infatti notare che le performance sono di un ordine di grandezza inferiore al resto dei DBMS. La maggiore velocità di MongoDB si suppone sia data dal fatto che sia scritto in C++ ed inoltre dalla trasformazione che esso effettua dal file JSON in BSON.
Di seguito vengono riportati i grafici a confronto delle tre query.

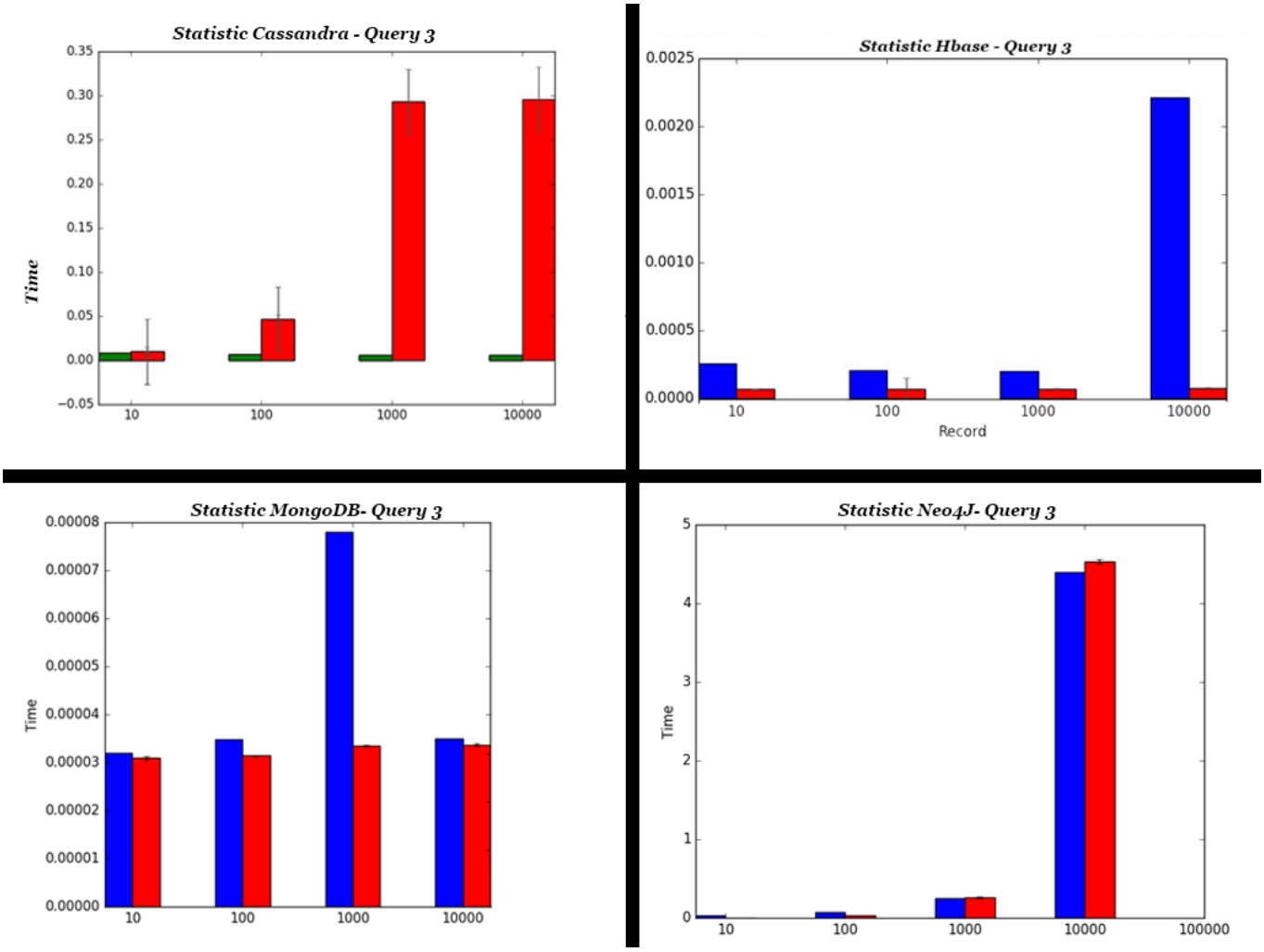
- Query 1


Statistic Cassandra - Query 1
Time
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
10
100
1000
10000
Statistic Hbase - Query 1
Time
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
10
100
1000
10000
Record
Statistic MongoDB- Query 1
Time
0.00008
0.00007
0.00006
0.00005
0.00004
0.00003
0.00002
0.00001
0.00000
10
100
1000
10000
Statistic Neo4J- Query 1
Time
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
10
100
1000
10000
100000

- Query 2


Statistic Cassandra - Query 2
Time
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
10
100
1000
10000
Statistic Hbase - Query 2
Time
0.0009
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
10
100
1000
10000
Record
Statistic MongoDB- Query 2
Time
0.00008
0.00007
0.00006
0.00005
0.00004
0.00003
0.00002
0.00001
0.00000
10
100
1000
10000
Statistic Neo4J- Query 2
Time
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
10
100
1000
10000
100000

- Query 3


Statistic Cassandra - Query 3
Time
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
10
100
1000
10000
Statistic Hbase - Query 3
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
10
100
1000
10000
Record
Statistic MongoDB- Query 3
Time
0.00008
0.00007
0.00006
0.00005
0.00004
0.00003
0.00002
0.00001
0.00000
10
100
1000
10000
Statistic Neo4J- Query 3
Time
5
4
3
2
1
0
10
100
1000
10000
100000