Realizzazione database di un Hospital Information System (HIS) basato su Cassandra

Salvatore Bertoncini, Nazzareno di Pietro

Università degli Studi di Messina

Dipartimento di Scienze Matematiche e Informatiche,
Scienze Fisiche e Scienze della Terra

Contents

Richiesta

Si richiede la realizzazione di un applicativo che simuli il funzionamento e le prestazioni di un database NoSql su uno schema preesistente. Il DBMS richiesto è nella fattispecie Cassandra. L'applicativo richiede di generare in modo casuale i dati che andranno in seguito a popolare il database iniziale da uno schema preesistente, con dataset di 10, 100, 1000, 10000 e 100000 elementi. Si richiede di analizzare le differenze in termini di costi ed efficienza con altri gruppi che stanno effettuando lo stesso progetto su database di tipo NoSql differenti (MongoDB, HBase, Neo4j). Si richede infine di plottare i risultati ottenuti e di confrontarli.

Ambiente di Sviluppo

IDE: JetBrain PyCharm

L'applicazione è stata implementata utilizzando il linguaggio Python 2.7, con IDE PyCharm.

Pandoc

Per la stesura della relazione è stato utilizzato Pandoc, un strumento che permette di scrivere documenti in linguaggio markup e che converte successivamente gli stessi in LaTeX, PDF e qualsiasi altro formato desiderato. Vi è inoltre la possibilità di utilizzare direttive LaTeXdirettamente all'interno del documento stesso.

Architettura dell'Applicativo

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

Conessione al DBMS

Per l'utilizzo di Cassandra su PyCharm è stato necessario implementare la libreria apposita, liberamente scaricabile dall'IDE stesso. Una volta implementata nel progetto, è possibile accedere al proprio keyspace andando a specificarlo attraverso il seguente codice:

```
from cassandra.cluster import Cluster

cluster = Cluster()
session = cluster.connect('db2_project')
```

Specifichiamo successivamente due funzioni che ci permettono di velocizzare il processo di querying.

```
def update(query):
    session.execute(query)


def query(query):
    return session.execute(query)
```

Json

Per prelevare i dati dai vari dataset in json e determinare il dataset adatto è stata importata la libreria json e sviluppato il seguente codice:

```
import json

def settaData(scelta):
    if(scelta == 0):
        with open('ds10.json') as data:
            record = json.load(data)
    elif (scelta == 1):
        with open('ds100.json') as data:
            record = json.load(data)
    elif (scelta == 2):
```

```
        with open('ds1000.json') as data:
            record = json.load(data)
    elif (scelta == 3):
        with open('ds10000.json') as data:
            record = json.load(data)
    elif (scelta == 4):
        with open('ds100000.json') as data:
            record = json.load(data)
    return record

def getAllData(scelta):
    record = settaData(scelta)
    all = record
    return all
```

Main

Il Main contiene un contatore chiamato "scelta", che permette di selezionare il dataset adatto e di passarlo prima alla funzione eliminaTuttiDati() che consente di inizializzare il keyspace e successivamente inserisciAllData(scelta) che permette di trasferire all'interno del keyspace il dataset scelto. Una volta riempito il keyspace col dataset corretto, una funzione si preoccuperà di eseguire le query e di stampare il tempo totale necessario all'esecuzione della query indicata. Da notare l'utilizzo della libreria *matplotlib.pyplot* che consente di plottare i grafici del tempi in una immagine con estensione *png*. Di seguito il codice utilizzato:

```
import Connection
import getJson
import time
import matplotlib.pyplot as plt


def eliminaTuttiDati():
    tab = ["patient"]
    i = 0
    while (i < len(tab)):
        query = "truncate " + tab[i] + ";"
        Connection.update(query)
        i = i + 1

def inserisciAllData(scelta):
    people = getJson.getAllData(scelta)
    step = ''
    for x in people:
```

```
            step = ''
            query = "insert into patient (id, name, width, height, l_shank, l_thigh, lokomat_sha
                    "lokomat_recorded, version, legtype, lwalk_training_duration, lwalk_distanc
                    "values ("+ str(x['id'])+", '" + str(x['name']) + "', " + str(x['width']) +
                    + str(x['lokomat_shank'])+", "+ str(x['lokomat_thigh'])+", "+ str(x['lokomat
                    + str(x['lwalk_training_duration'])+", "+ str(x['lwalk_distance'])+", "
            step += "["
            for s in x['step_datas']:
                step += "{step_value: "
                step += "["
                for a in s:
                    if (a == None):
                        step += "'None', "
                    else:
                        step += "'"+str(a)+"', "
                step = step[:-2]
                step += "]}, "
            if(len(x['step_datas']) > 0):
                step = step[:-2]
            step += "]"
            query += step
            query += ");"

            Connection.update(query)


    def funzione(scelta):
        tempi = []
        for _ in range(0,31):
            tPrima = time.time()
            risultato = Connection.query("select * from patient;")
            tTotale = time.time() - tPrima
            tempi.append(tTotale)

        plt.plot(tempi)
        plt.ylabel("Query n " + str(1))
        plt.xlabel(scelta)
        plt.savefig('doc/' + str(scelta) + 'Query' + str(1) + '.png')
        plt.clf()

        print "Il tempo per fare la query 1 e': " + str(tTotale)

        tempi = []
        for _ in range(0,31):
            tPrima = time.time()
            risultato = Connection.query("SELECT * FROM patient WHERE name = 'SIVV33W0' allow fi
```

```
            tTotale = time.time() - tPrima
            tempi.append(tTotale)

        plt.plot(tempi)
        plt.ylabel("Query n " + str(2))
        plt.xlabel(scelta)
        plt.savefig('doc/' + str(scelta) + 'Query' + str(2) + '.png')
        plt.clf()

        print "Il tempo per fare la query 2 e': " + str(tTotale)

        tempi = []

        for _ in range(0,31):
            tPrima = time.time()
            risultato = Connection.query("select * from patient where lwalk_training_duration >
            tTotale = time.time() - tPrima
            tempi.append(tTotale)

        plt.plot(tempi)
        plt.ylabel("Query n " + str(3))
        plt.xlabel(scelta)
        plt.savefig('doc/' + str(scelta) + 'Query' + str(3) + '.png')
        plt.clf()

        print "Il tempo per fare la query 3 e': " + str(tTotale)

    scelta = 0
    while (scelta < 4):
        print scelta


        print "elimino elementi dal DB"
        eliminaTuttiDati()
        print "inserisco gli elementi dal DB"
        inserisciAllData(scelta)
        print "inizio le query"

        funzione(scelta)

        scelta = scelta + 1
```

Statistiche

Di seguito sono riportati i grafici che descrivono le performance sui vari dataset.

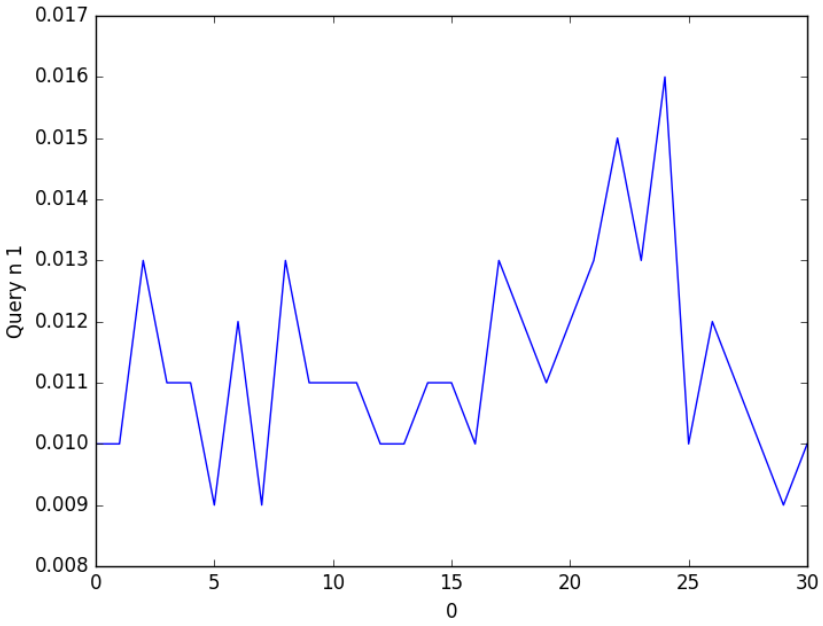

Figure 1: ds10.json query 1

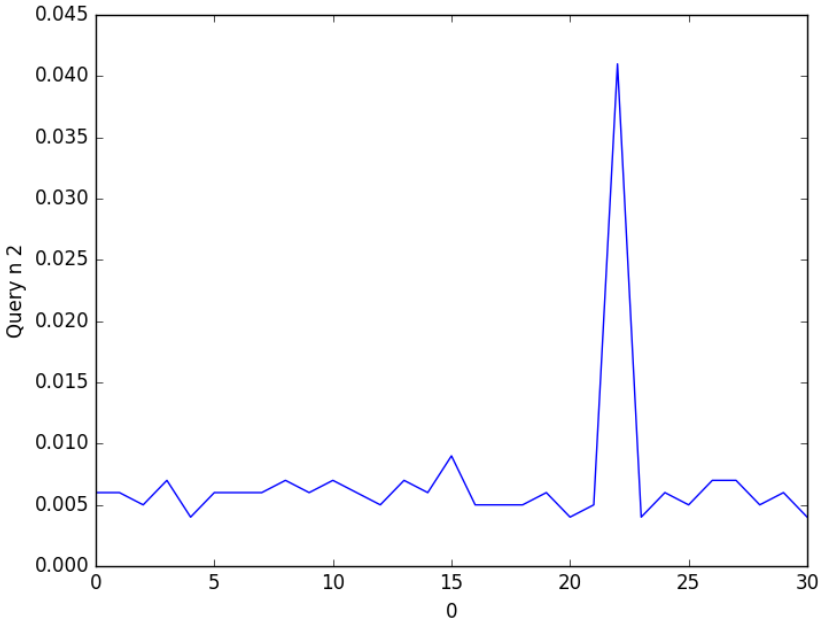

Figure 2: ds10.json query 2

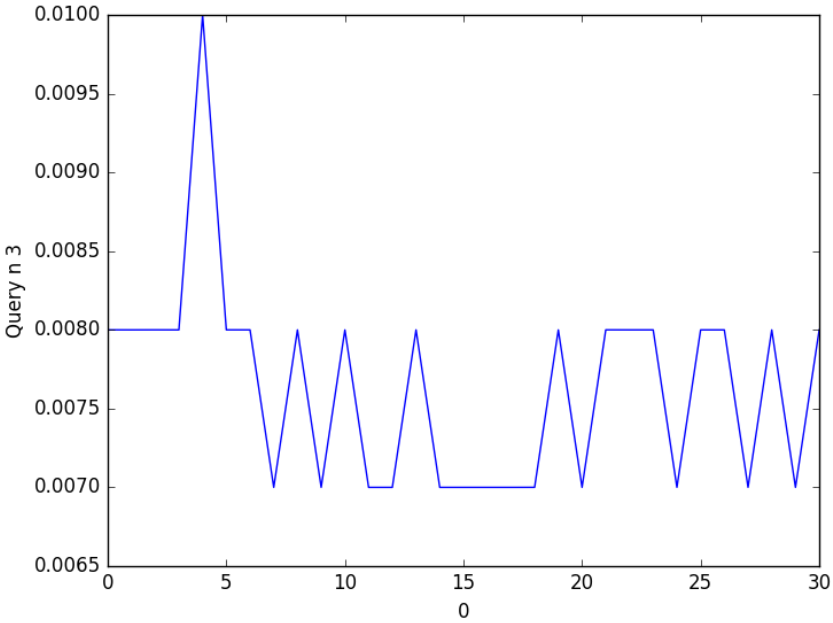

Figure 3: ds10.json query 3

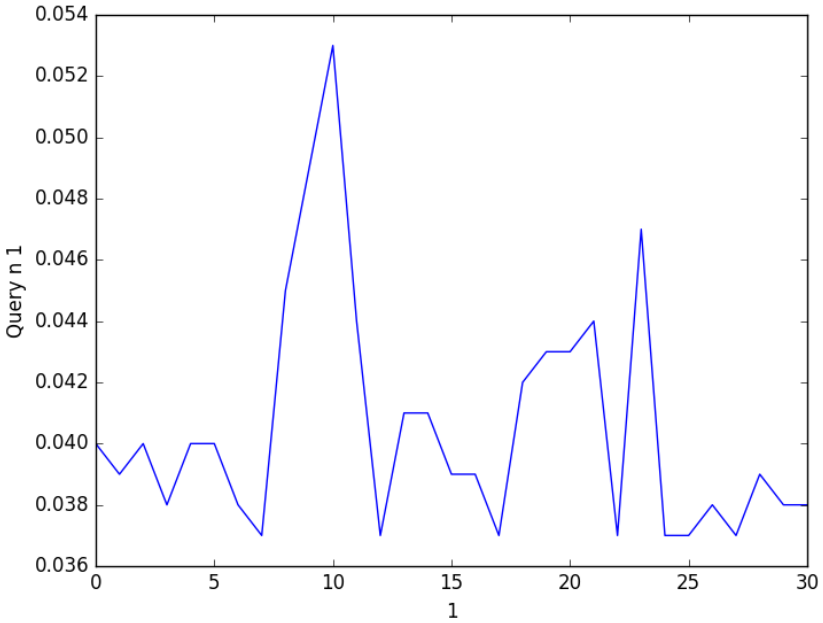

Figure 4: ds100.json query 1

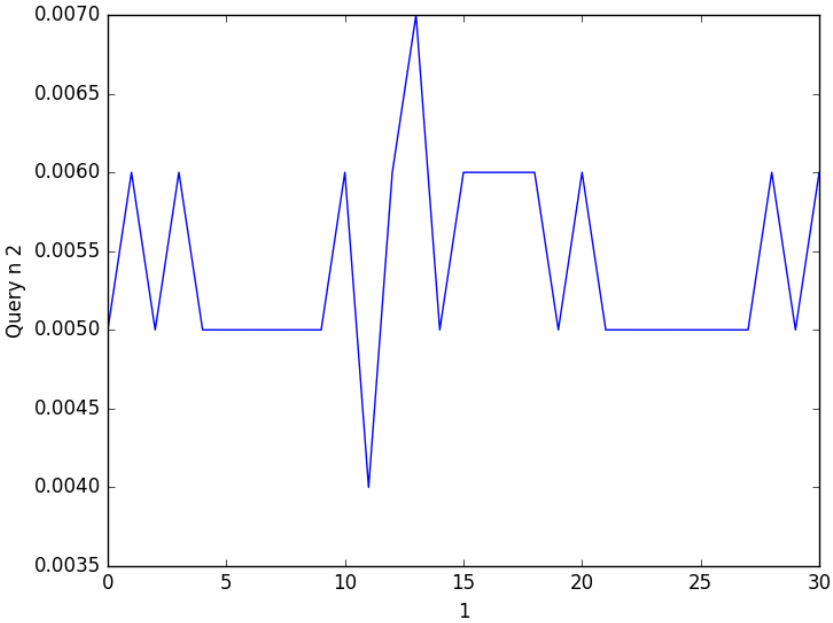

Figure 5: ds100.json query 2

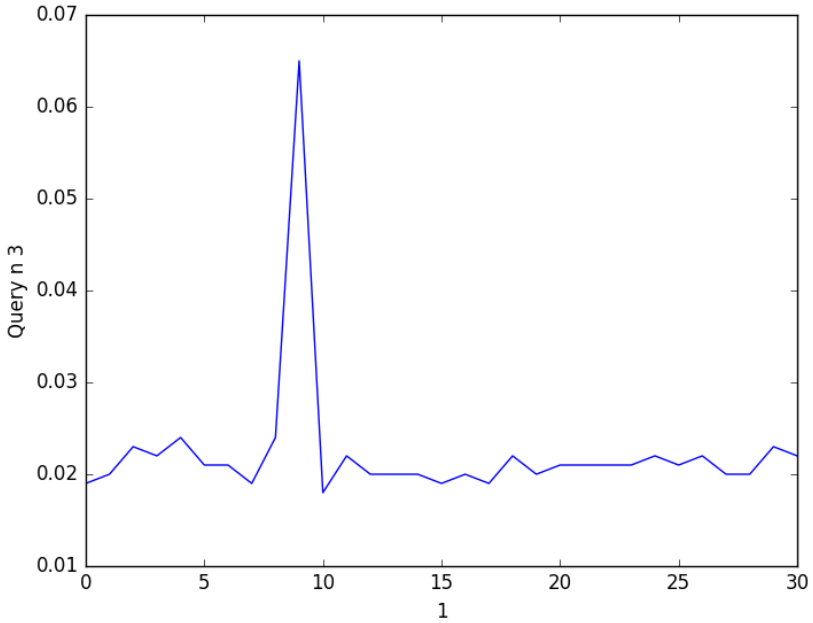

Figure 6: ds100.json query 3

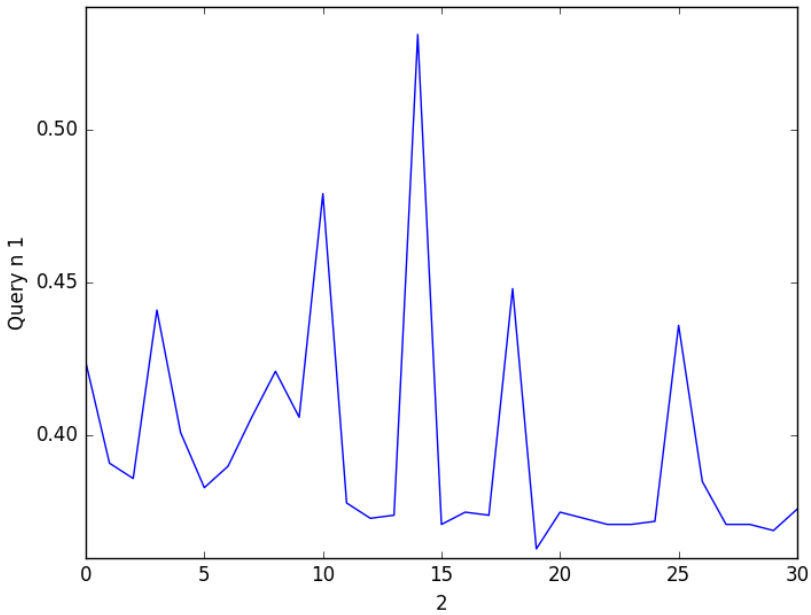

Figure 7: ds1000.json query 1

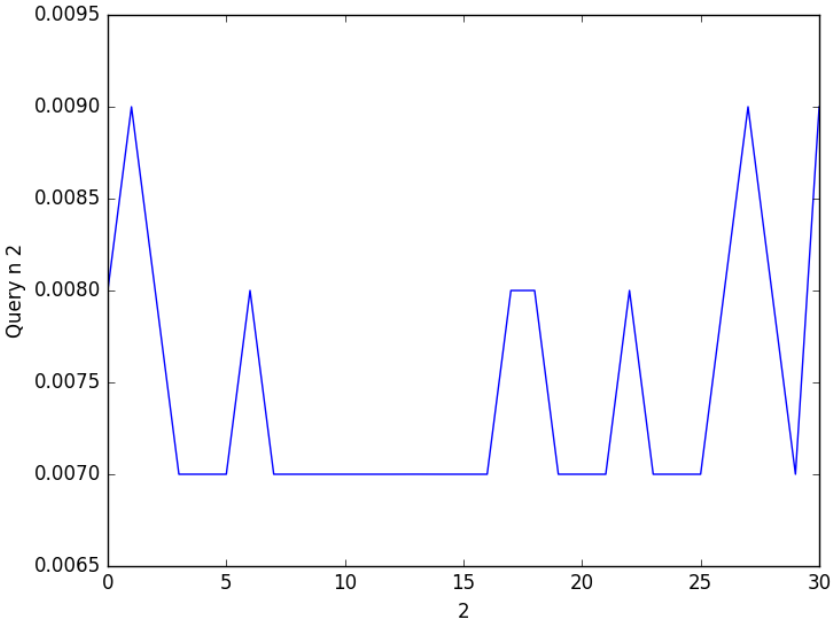

Figure 8: ds1000.json query 2

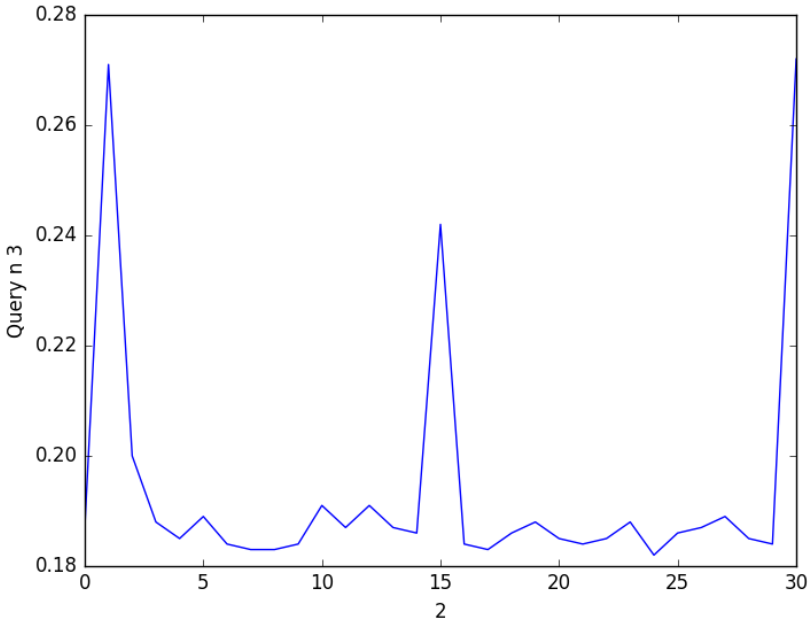

Figure 9: ds1000.json query 3

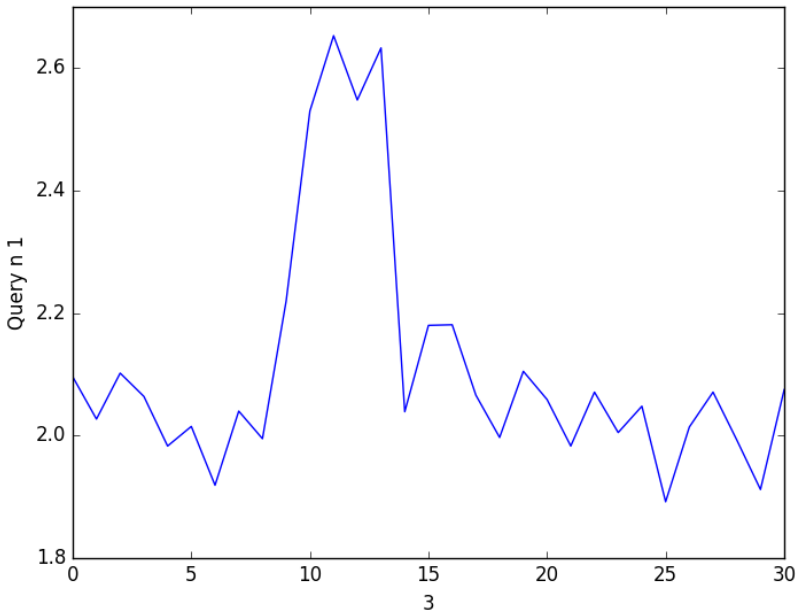

Figure 10: ds10000.json query 1

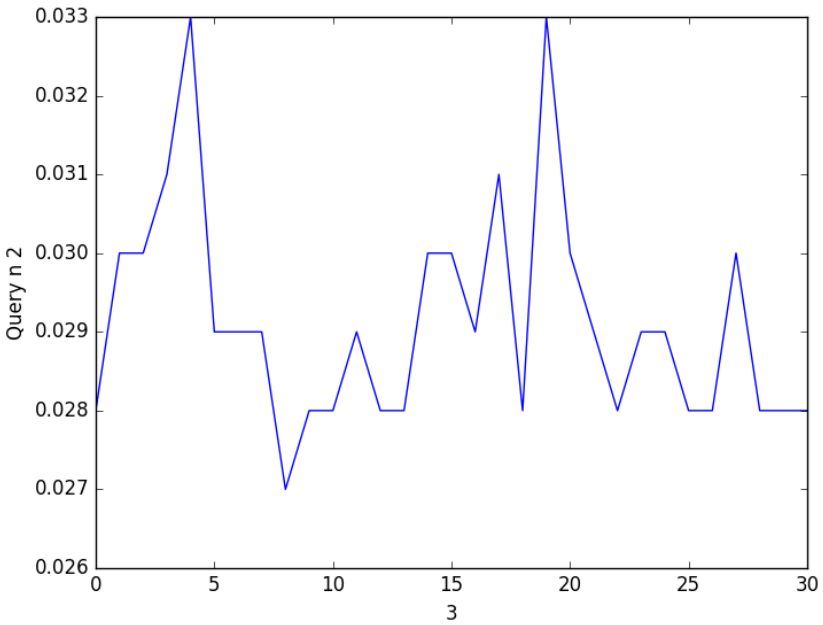

Figure 11: ds10000.json query 2

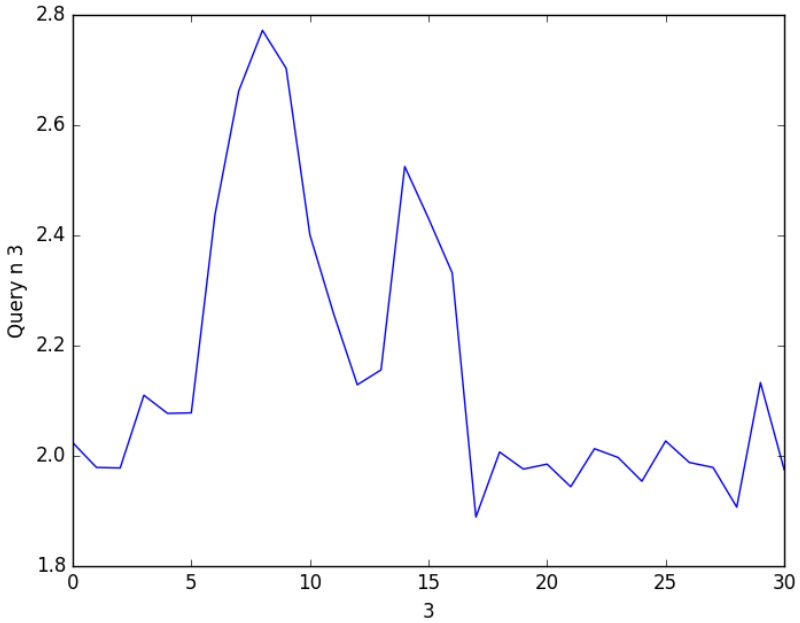

Figure 12: ds10000.json query 3