Università degli studi di Messina Dipartimento di Scienze Matematiche, Informatiche, Scienze Fisiche e Scienze della Terra (MIFT)

Corso di Laurea Triennale in INFORMATICA

Progetto Basi di Dati NoSQL

Federico Sciuto

PROBLEMATICA AFFRONTATA

Si calcola che in Italia vengano effettuate in media 160 milioni di telefonate al giorno. Una mole di dati esorbitanti, dunque, è immagazzinata nei tabulati telefonici, soprattutto se si pensa che le informazioni riguardanti ogni singola chiamata vengono cancellate solo dopo 24 mesi.

In ambito giudiziario l'acquisizione dei tabulati telefonici è una pratica che può rivelarsi fondamentale per lo svolgimento di un caso; tuttavia, ciò è possibile solamente entro le 48 ore successive all'attuazione della telefonata.

I database utilizzati per il confronto prestazionale sono MongoDB e Neo4j.

- MongoDB memorizza i dati in documenti flessibili JSON-like, il che significa che i campi possono variare da un documento all'altro ed è possibile modificare nel tempo la struttura dei dati.
  Il modello di documento mappa gli oggetti del codice applicativo, semplificando il lavoro sui dati.
  Le query ad hoc, l'indicizzazione e l'aggregazione in tempo reale offrono efficaci modalità di accesso e analisi dei dati.
  MongoDB è concepito in origine come database distribuito; l'alta disponibilità, la scalabilità orizzontale e la distribuzione geografica sono quindi native e facili da usare.

- Neo4j è un DBMS di tipo NoSQL orientato ai grafi (graph-oriented), open-source, che fornisce un back-end transazionale conforme ad ACID, disponibile pubblicamente dal 2007.
  È offerto come servizio gestito tramite AuraDB, ma è anche possibile eseguire personalmente Neo4j con la Community Edition o la Enterprise Edition.
  Neo4j è scritto in Java e Scala e il codice sorgente è disponibile su GitHub.
  Essendo un database a grafi nativo, implementa un vero modello grafico fino al livello di archiviazione; i dati non vengono archiviati come "astrazione dei grafi" in cima ad un'altra tecnologia.

SOLUZIONE DB CONSIDERATA

I database sono stati creati tramite l'utilizzo del linguaggio di programmazione JavaScript, eseguito tramite "node.js" (ovvero l'ambiente di runtime per l'esecuzione di JavaScript), e popolati con dati fittizi generati grazie alla libreria "Faker.js" e salvati in file JSON.

dati75.mjs

```
import { faker } from '@faker-js/faker';
import fs from 'fs';

faker.locale = 'it';

var person = [];
var call = [];

for (var i=1; i<=5000; i++) {

    var First_name = faker.name.firstName();
    var Last_name = faker.name.lastName();
    var Full_name = First_name + Last_name;
    var Phone_number = faker.phone.number('### ### ####');

    var First_name2 = faker.name.firstName();
    var Last_name2 = faker.name.lastName();
    var Full_name2 = First_name2 + Last_name2;
    var Phone_number2 = faker.phone.number('### ### ####');

    person.push({
        "Full_name": Full_name,
        "First_name": First_name,
        "Last_name": Last_name,
        "Phone_number": Phone_number,
        "Calls_done": i
    });
    person.push({
        "Full_name": Full_name2,
        "First_name": First_name2,
        "Last_name": Last_name2,
        "Phone_number": Phone_number2,
        "Calls_done": i
    });

    var Call_start = Math.floor(Math.random() * 31535399) + 1640991600;
    var Duration = Math.floor(Math.random() * 600);
    var Call_end = Call_start + Duration;

    var City = faker.address.city();
    var Address = faker.address.streetAddress(false);
    var Cell_site = Math.floor(Math.random() * 999) + 1;

    call.push({
        "ID": i,
        "Call_start": Call_start,
        "Call_end": Call_end,
        "Duration": Duration,
        "City": City,
        "Address": Address,
        "Cell_site": Cell_site
    });
}

let dataObj = person;

fs.writeFile('people75.json', JSON.stringify(dataObj, null, '\t'), function(err, result) {
    if(err) console.log('error', err);
    else console.log('File saved');
});

let dataObj2 = call;

fs.writeFile('calls75.json', JSON.stringify(dataObj2, null, '\t'), function(err, result) {
    if(err) console.log('error', err);
    else console.log('File saved');
});
```

PROGETTAZIONE

MongoDB

I dati all'interno di MongoDB sono stati gestiti con dei documenti incorporati (embedded documents), una funzionalità specifica del DBMS in questione che permette la gestione dei dati in gruppi e come singoli elementi.

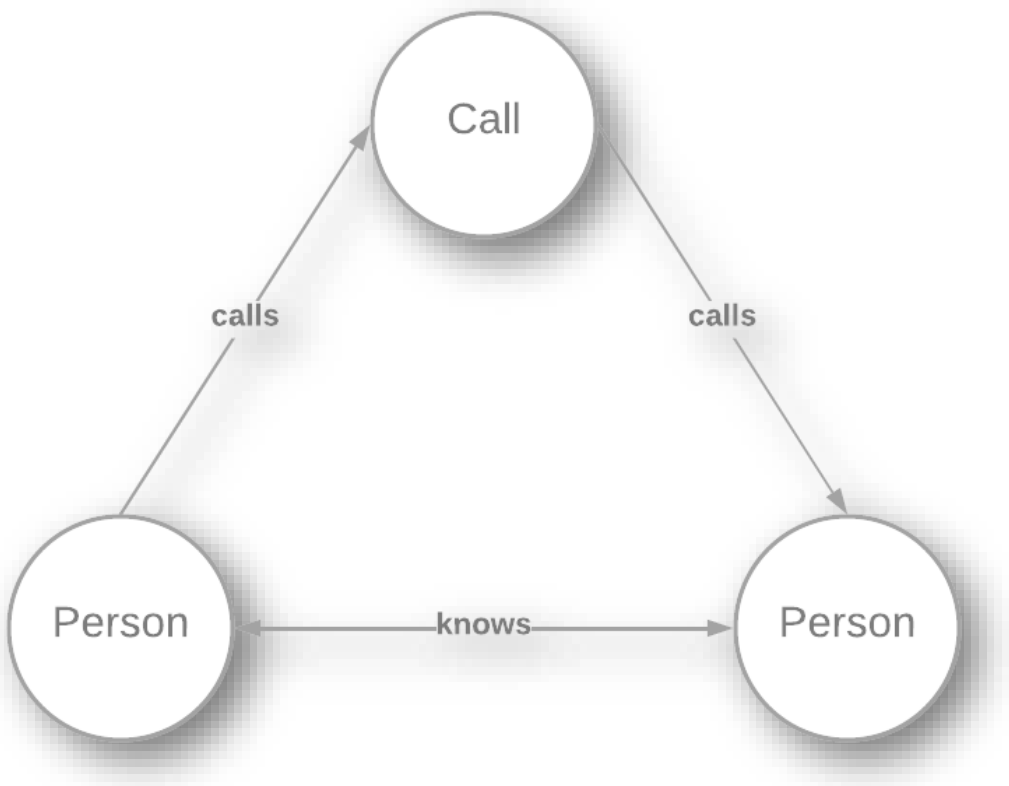
Di seguito è presente un esempio di come si presentano i documenti gestiti da MongoDB

e i rispettivi attributi:

- **"FULL_NAME":** Nome e cognome della persona (stringa);
- **"FIRST_NAME":** Nome della persona (stringa);
- **"LAST_NAME":** Cognome della persona (stringa);
- **"PHONE_NUMBER":** Numero di telefono della persona (stringa).
- **"CALLS_DONE":** Insieme delle chiamate effettuate dalla persona (array di documenti), ciascuna con:
  - **"ID":** L'identificativo unico per ciascuna chiamata (numero intero);
  - **"CALL_START":** Data e ora dell'inizio della chiamata, rappresentata in UNIX timestamp (numero intero) (1/1/2022 0:00:00 – 31/12/2022 23:49:59);
  - **"DURATION":** Durata della chiamata in secondi (numero intero);

- **“CALL_END”:** Data e ora della fine della chiamata, rappresentata in UNIX timestamp (numero intero);
- **“CITY”:** Città nella quale è avvenuta la chiamata (stringa);
- **“ADDRESS”:** Indirizzo in cui è avvenuta la chiamata (stringa);
- **“CELL_SITE”:** Cella telefonica della chiamata (numero intero);

Neo4j



Questo modello grafico rappresenta le relazioni che le persone (“Person”) hanno con la telefonata (“Call”) e tra di loro (chi telefona e chi riceve la telefonata).

Per ogni “Call” abbiamo sempre due “Persone” che, essendo state in contatto, sono l’una a conoscenza dell’esistenza dell’altra, questa relazione è fondamentale per avere una rete di contatti durante le indagini.

Ogni entità “Call” è composta dai campi:

- **“ID”:** L’identificativo unico per ciascuna chiamata (numero intero);
- **“CALL_START”:** Data e ora dell’inizio della chiamata, rappresentata in UNIX timestamp (numero intero) (1/1/2022 0:00:00 – 31/12/2022 23:49:59);
- **“DURATION”:** Durata della chiamata in secondi (numero intero);
- **“CALL_END”:** Data e ora della fine della chiamata, rappresentata in UNIX timestamp (numero intero);
- **“CITY”:** Città nella quale è avvenuta la chiamata (stringa);
- **“ADDRESS”:** Indirizzo in cui è avvenuta la chiamata (stringa);
- **“CELL_SITE”:** Cella telefonica della chiamata (numero intero);

Ogni entità “Person” è composta dai campi:

- **“FULL_NAME”:** Nome e cognome della persona (stringa);
- **“FIRST_NAME”:** Nome della persona (stringa);
- **“LAST_NAME”:** Cognome della persona (stringa);
- **“PHONE_NUMBER”:** Numero di telefono della persona (stringa).
- **“CALLS_DONE.ID”:** Insieme degli ID delle chiamate effettuate dalla persona (numero intero)

IMPLEMENTAZIONE

Dopo aver creato i file JSON contenenti i dati da inserire nei vari database, si effettua la connessione a MongoDB tramite JavaScript grazie alla libreria “MongoClient” e si inseriscono i dati letti dal file grazie al comando “.insertMany”.

inserimento75.mjs

```
import { MongoClient } from 'mongodb';
import fs from 'fs';

const dbName = 'CallsRecord75';
const client = new MongoClient('mongodb://127.0.0.1:27017');

var startTime = performance.now();

client.connect(function(err) {

  console.log('Connected successfully to server');
  const db = client.db(dbName);
  const data = fs.readFileSync('calls75.json');
  const docs = JSON.parse(data.toString());

  db.collection('calls')
  .insertMany(docs, function(err, result) {
    if (err) throw err;
    console.log('Inserted docs:', result.insertedCount);
    client.close();

    var endTime = performance.now();

    let mia = `${endTime - startTime} msec\n`;

    console.log(mia);

    fs.appendFile('Inserimento75.txt', mia, (err) => {
      if (err) throw err;
    });

  });

});
```

Per quanto riguarda Neo4j, per prima cosa sono stati creati due indici per velocizzare l'inserimento dei dati grazie ai seguenti comandi Cypher (query language di Neo4j):

```
neo4j$ CREATE INDEX Person FOR (nodeP:Person) ON (nodeP.Full_name)
neo4j$ CREATE INDEX Call FOR (nodeC:Call) ON (nodeC.ID)
```

La connessione al database avviene grazie al driver ufficiale fornito da Neo4j per JavaScript "neo4j-driver".

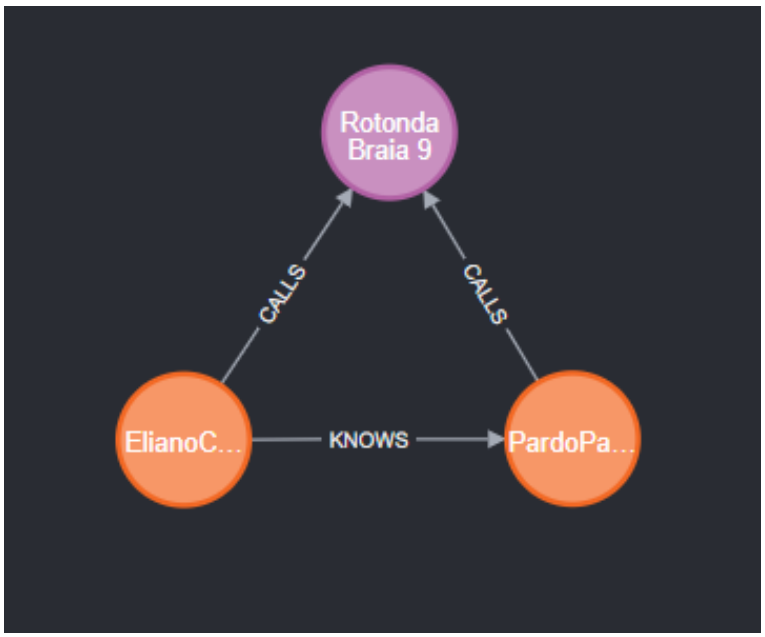
L'inserimento dei dati prelevati dai file avviene tramite comandi Cypher con il comando "apoc.load.json".

Dopo aver importato tutte le entità sono state create le relazioni che le interessano usando sempre istruzioni Cypher.

inserimento75.mjs

```
import neo4j from 'neo4j-driver'
import fs from 'fs'

const uri = "neo4j://localhost:7687";
const user = "neo4j";
const password = "admin";

const driver = neo4j.driver(uri, neo4j.auth.basic(user, password));
const session = driver.session();

const startTime = performance.now();

await session.run(
  'CALL apoc.load.json("file:///people75.json") YIELD value UNWIND value AS persone MERGE (p:Person{Full_name: persone.Full_name})',
  'SET p.First_name = persone.First_name, p.Last_name = persone.Last_name, p.Phone_number = persone.Phone_number',
);
await session.run(
  'CALL apoc.load.json("file:///calls75.json") YIELD value UNWIND value AS chiamate MERGE (c:Call{ID: chiamate.ID}) SET c.Call_start = chiamate.Call_start',
  'c.Call_end = chiamate.Call_end, c.Duration = chiamate.Duration, c.City = chiamate.City, c.Address = chiamate.Address, c.Cell_site = chiamate.Cell_site',
);

const endTime = performance.now();

await session.run(
  'CALL apoc.load.json("file:///people75.json") YIELD value UNWIND value AS persone MERGE (p:Person{Full_name: persone.Full_name}) WITH p',
  'persone UNWIND persone.Calls_done as chiamate MERGE (c:Call{ ID: chiamate }) MERGE (p)-[:CALLS]->(c)',
);
await session.run(
  'MATCH (a:Person)-[:CALLS]-(c:Call)-[:CALLS]-(b:Person) MERGE (a)-[:KNOWS]-(b)',
);

await session.close();


let mia = `${endTime - startTime} msec\n`;

console.log(mia);

fs.appendFile('Inserimento75.txt', mia, (err) => {
  if (err) throw err
});

await driver.close();
```

Questo è un esempio di come si presentano le entità con le relative relazioni in Neo4j



In entrambi i programmi JavaScript sopra riportati, il tempo di inserimento dei dati nei database è stampato a video, oltre ad essere salvato in un file di testo, grazie a questa porzione di codice.

```
7   var startTime = performance.now();
8
9   // inserimento
10
11  var endTime = performance.now();
12
13  let mia = `${endTime - startTime} msec\n`;
14
15  console.log(mia);
16
17  fs.appendFile('Inserimento75.txt', mia, (err) => {
18    if (err) throw err;
19  });
```

Di seguito troviamo una tabella rappresentante i tempi d'inserimento dei vari dataset per entrambi i DBMS

	MongoDB	*Neo4j*
25%	102,22	162,80
50%	151,52	180,35
75%	396,90	865,73
100%	2.598,78	8.150,76

ESPERIMENTI

Per confrontare le prestazioni dei due DBMS in esame sono state eseguite 5 query ciascuno, calcolando il tempo di esecuzione.

Ogni query è stata eseguita 31 volte prendendo in considerazione:

- Il tempo della prima esecuzione (in millisecondi);
- Il tempo medio delle 30 esecuzioni successive (in millisecondi);
- La deviazione standard;
- L'intervallo di confidenza al 95%.

Tutti i dati sopra citati sono stati immagazzinati in delle tabelle e hanno permesso di creare degli istogrammi.

QUERY 1

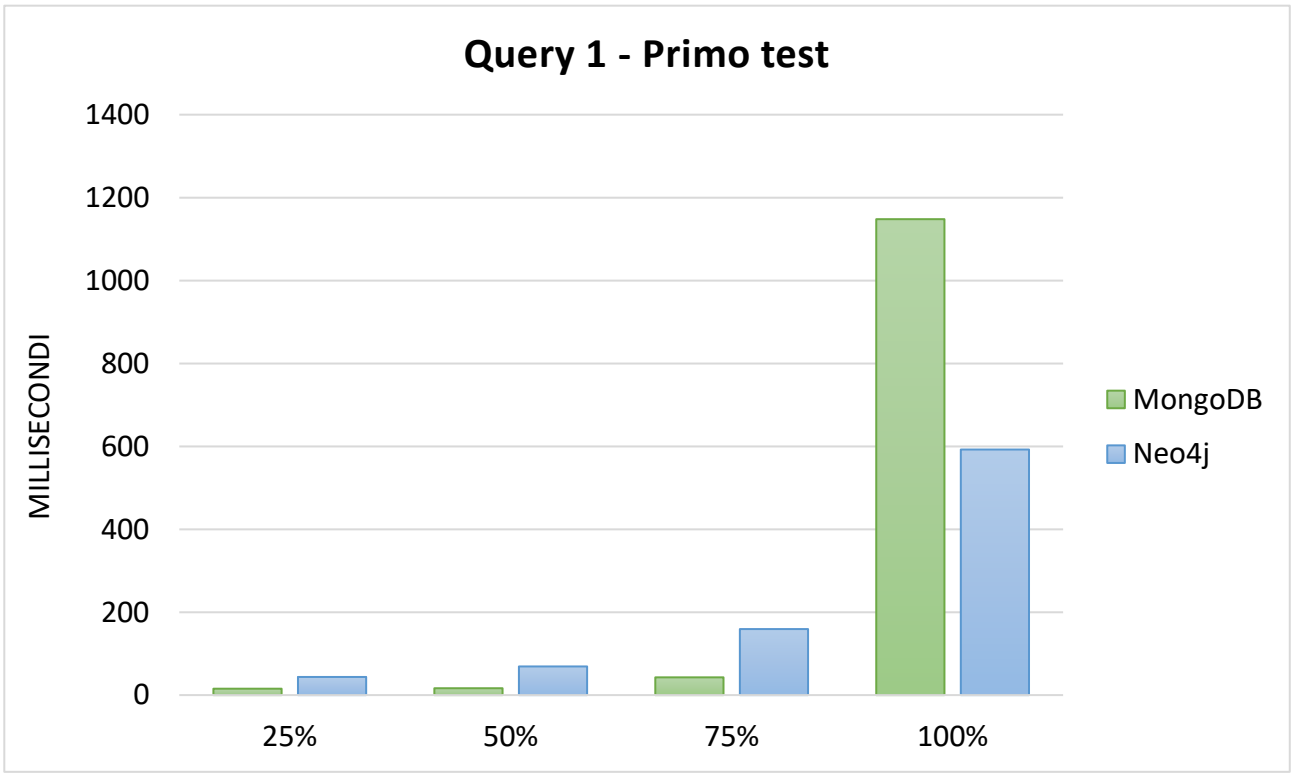
MongoDB:

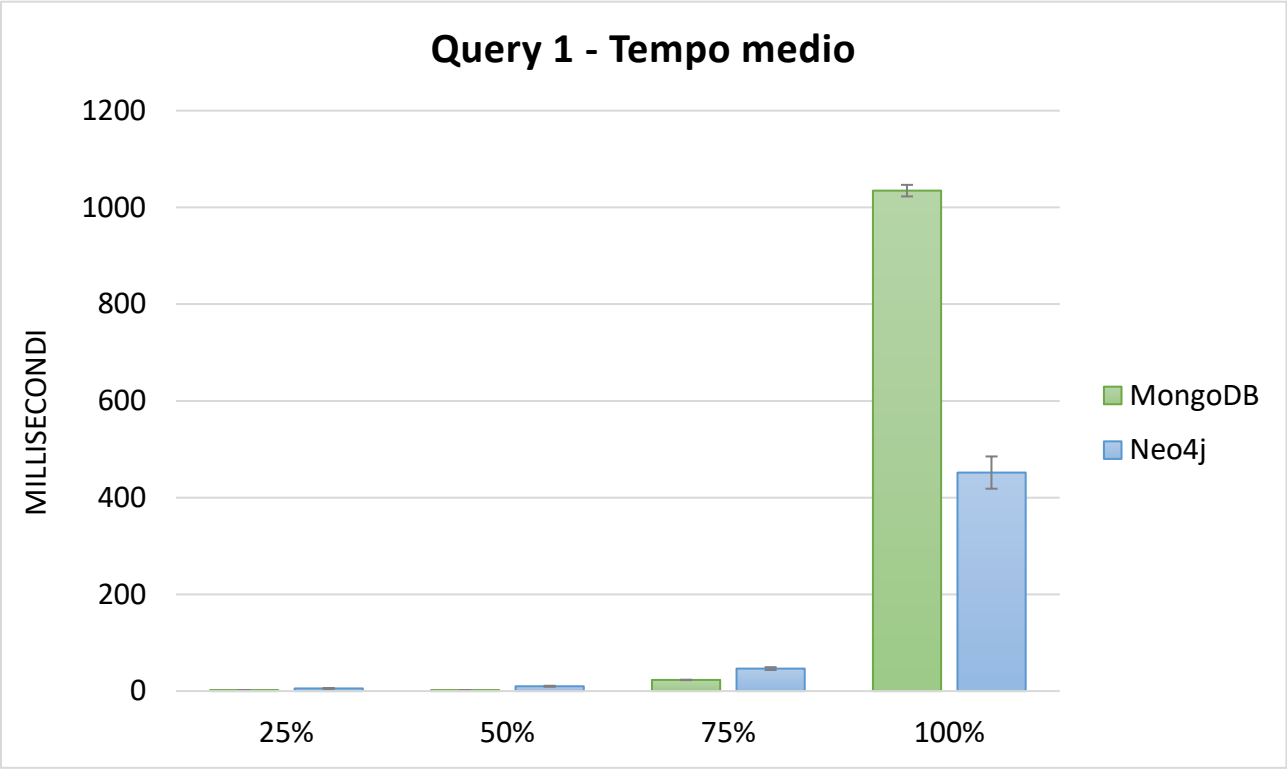
```
const query = { "Calls_done.Cell_site": { $gt: 100 } };
```

Neo4j:

```
await session.run('MATCH (n:Call) WHERE n.Cell_site > 100 RETURN n',
);
```

	MongoDB 25%	*Neo4j 25%*	*MongoDB 50%*	*Neo4j 50%*	*MongoDB 75%*	*Neo4j 75%*	*MongoDB 100%*	*Neo4j 100%*
Primo test	15,42	43,74	16,54	69,10	42,81	159,26	1.147,91	592,33
Tempo medio	2,13	5,46	2,10	9,96	23,05	46,49	1.034,66	451,73
Dev. standard	0,44	0,98	0,34	2,82	1,23	8,15	33,31	93,34
Conf. 95%	0,16	0,35	0,12	1,01	0,44	2,92	11,92	33,40





QUERY 2

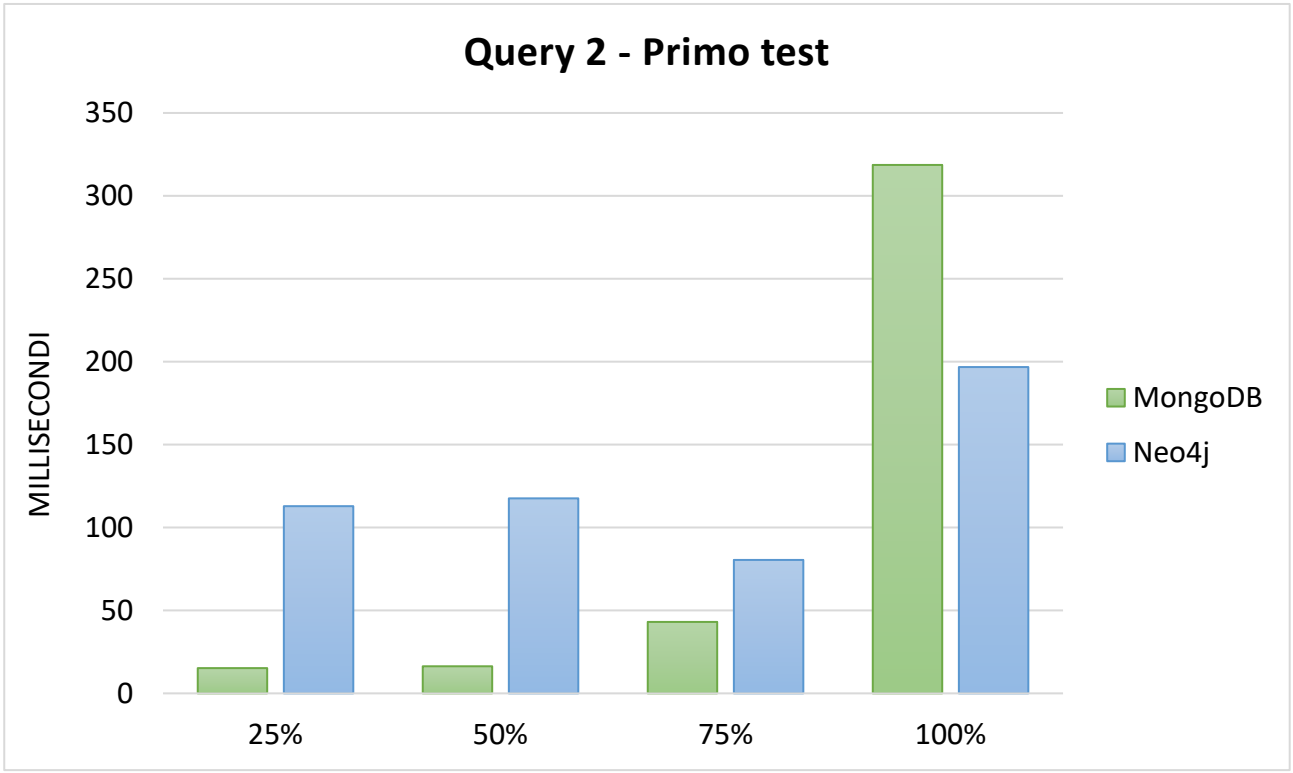
MongoDB:

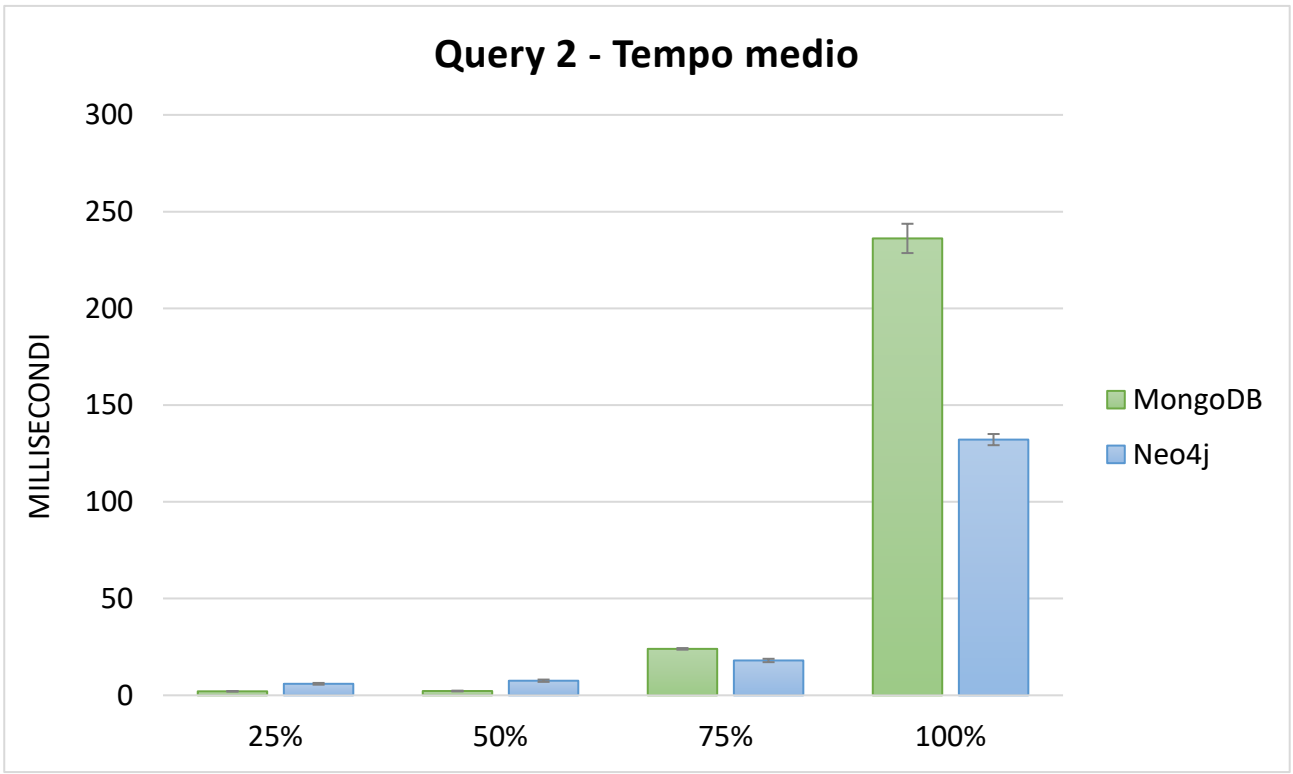
```
const query = { "Calls_done.Cell_site": { $gt: 100 }, "Calls_done.City": { $gt: "V" } };
```

Neo4j:

```
await session.run('MATCH (n:Call) WHERE n.Cell_site > 100 AND n.City > "V" RETURN n',
);
```

	MongoDB 25%	*Neo4j 25%*	*MongoDB 50%*	*Neo4j 50%*	*MongoDB 75%*	*Neo4j 75%*	*MongoDB 100%*	*Neo4j 100%*
Primo test	15,23	112,88	16,33	117,59	43,08	80,46	318,63	196,77
Tempo medio	2,06	5,95	2,22	7,54	24,00	18,01	236,15	132,17
Dev. standard	0,34	1,26	0,54	1,67	1,16	2,43	21,15	8,06
Conf. 95%	0,12	0,45	0,19	0,60	0,42	0,87	7,57	2,88


Query 2 - Primo test
MILLISECONDI
350
300
250
200
150
100
50
0
25%
50%
75%
100%
MongoDB
Neo4j


Query 2 - Tempo medio
MILLISECONDI
300
250
200
150
100
50
0
25%
50%
75%
100%
MongoDB
Neo4j

QUERY 3

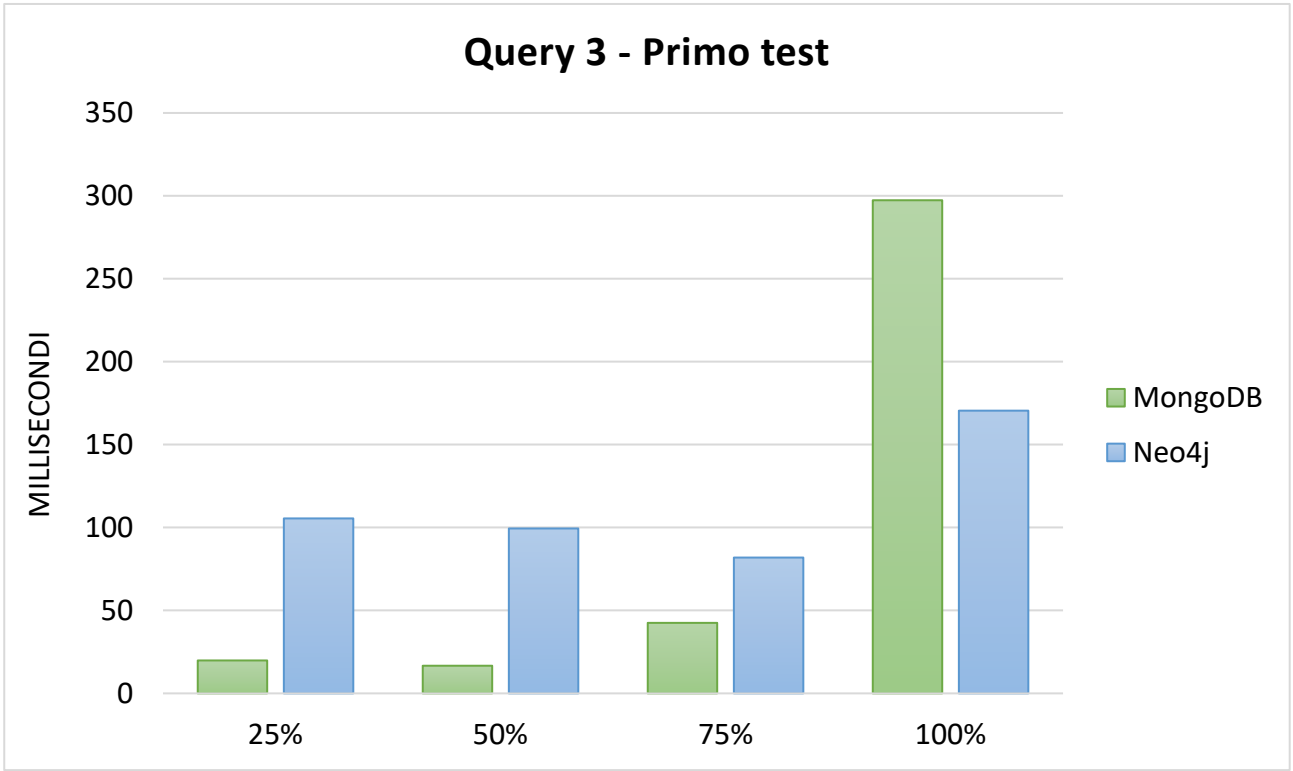
MongoDB:

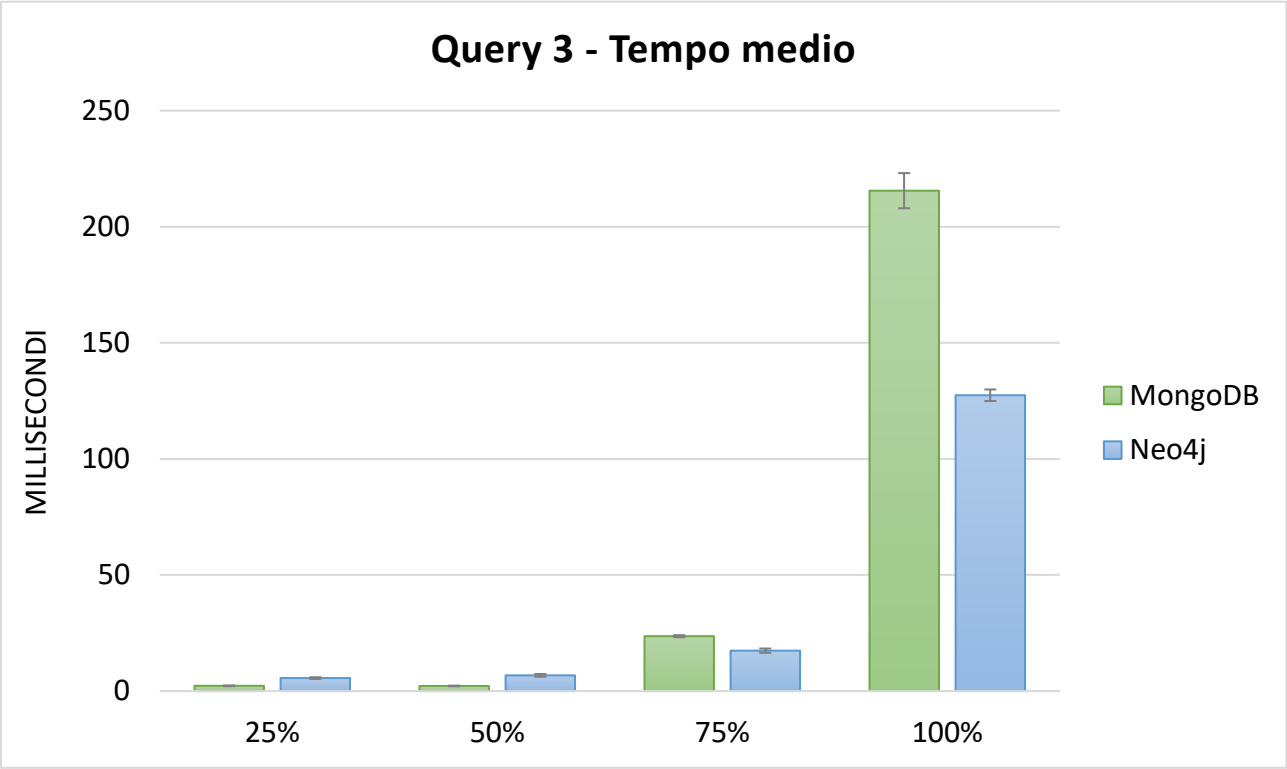
```
const query = { "Calls_done.Cell_site": { $gt: 100 }, "Calls_done.City": { $gt: "V" },
                "Calls_done.Call_start": { $gt: 1660000000 } };
```

Neo4j:

```
await session.run('MATCH (n:Call) WHERE n.Cell_site > 100 AND n.City > "V"',
                  'AND n.Call_start > 1660000000 RETURN n',
);
```

	MongoDB 25%	Neo4j 25%	MongoDB 50%	Neo4j 50%	MongoDB 75%	Neo4j 75%	MongoDB 100%	Neo4j 100%
Primo test	19,85	105,50	16,67	99,44	42,52	81,90	297,36	170,48
Tempo medio	2,28	5,61	2,20	6,78	23,64	17,41	215,53	127,43
Dev. standard	0,40	1,01	0,39	1,67	1,25	2,61	21,15	6,96
Conf. 95%	0,14	0,36	0,14	0,60	0,45	0,93	7,57	2,49





QUERY 4

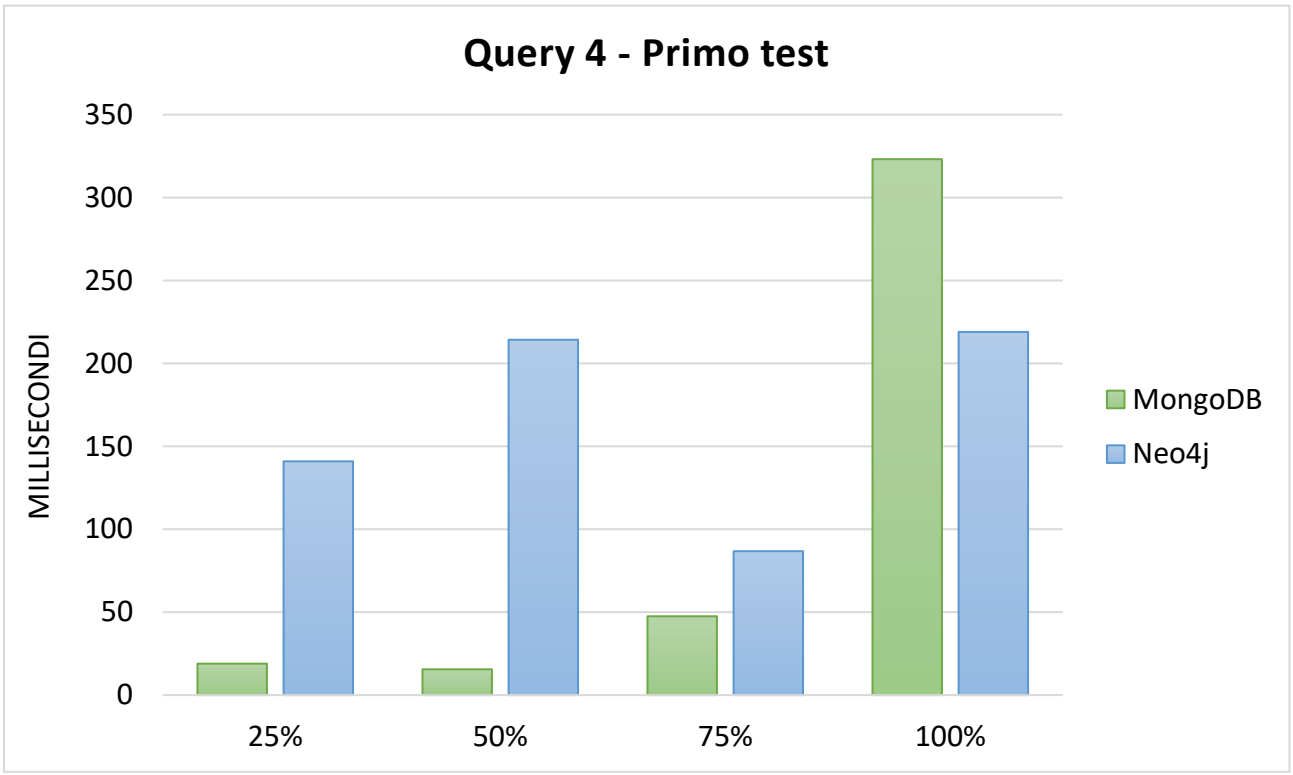
MongoDB:

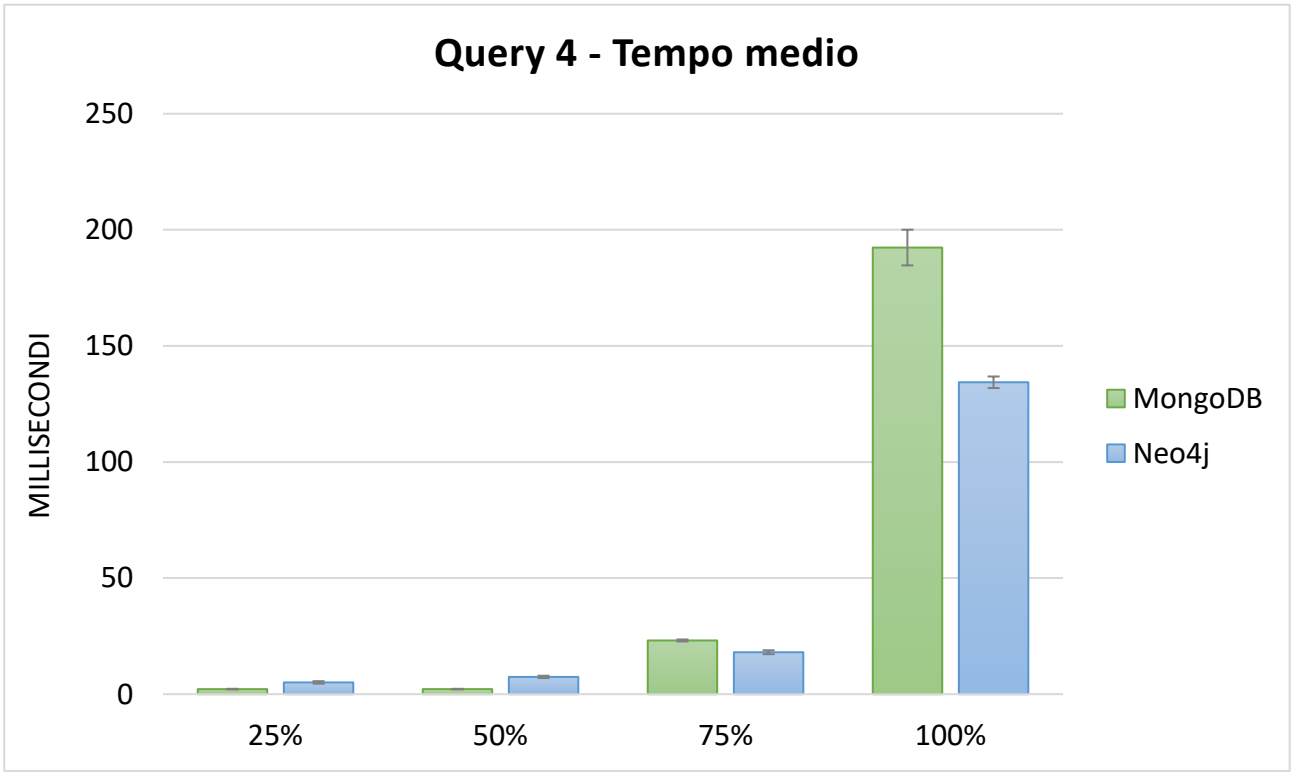
```
const query = { "Calls_done.Cell_site": { $gt: 100 }, "Calls_done.City": { $gt: "V" },
                "Calls_done.Call_start": { $gt: 1660000000 }, "Phone_number": { $gt: "700" } };
```

Neo4j:

```
await session.run('MATCH (n:Call)-[r]-(p:Person) WHERE n.Cell_site > 100 AND n.City > "V"',
                  'AND n.Call_start > 1660000000 AND p.Phone_number > "700" RETURN n',
);
```

	MongoDB 25%	***Neo4j 25%***	***MongoDB 50%***	***Neo4j 50%***	***MongoDB 75%***	***Neo4j 75%***	***MongoDB 100%***	***Neo4j 100%***
Primo test	18,96	140,97	15,55	214,27	47,53	86,76	323,15	218,96
Tempo medio	2,17	5,06	2,18	7,43	23,13	18,06	192,36	134,34
Dev. standard	0,37	1,48	0,36	1,42	1,20	2,35	21,51	6,97
Conf. 95%	0,13	0,53	0,13	0,51	0,43	0,84	7,70	2,49


Query 4 - Primo test
MILLISECONDI
350
300
250
200
150
100
50
0
25%
50%
75%
100%
MongoDB
Neo4j


Query 4 - Tempo medio
MILLISECONDI
250
200
150
100
50
0
25%
50%
75%
100%
MongoDB
Neo4j

QUERY 5:

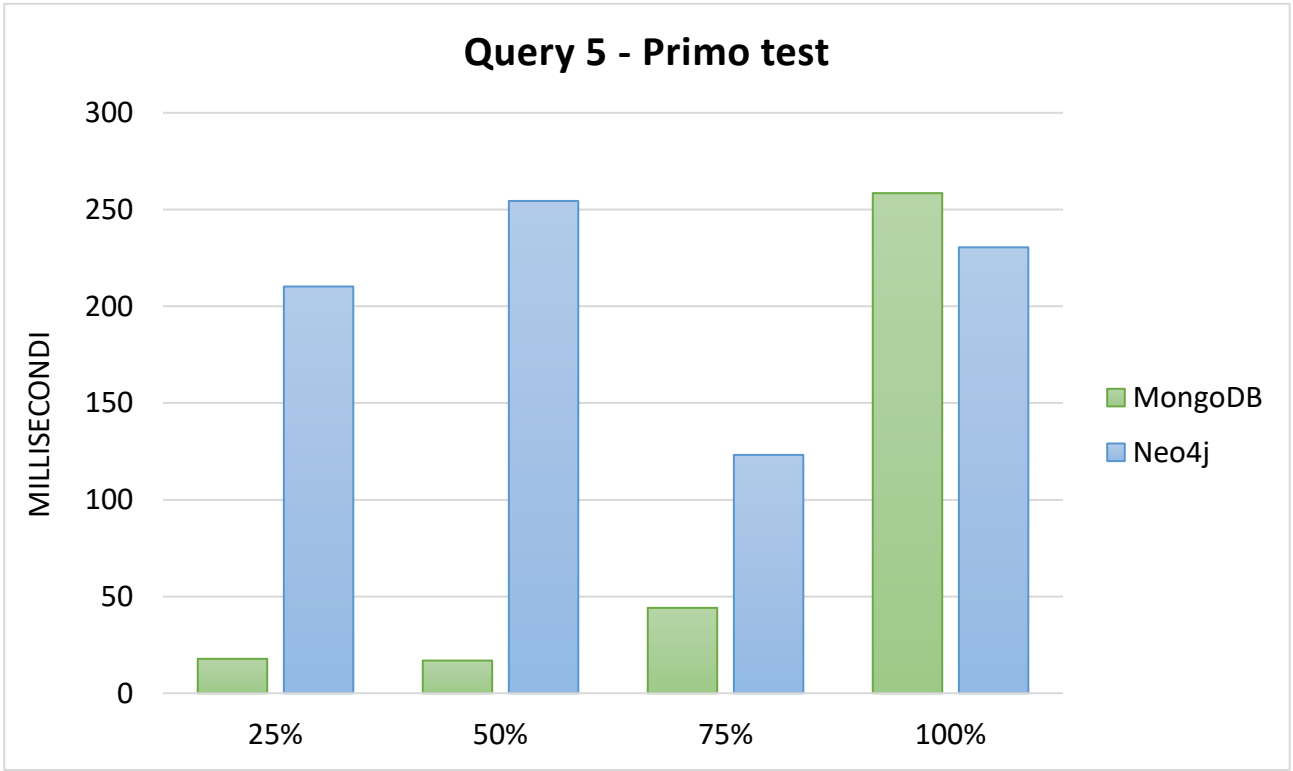
MongoDB:

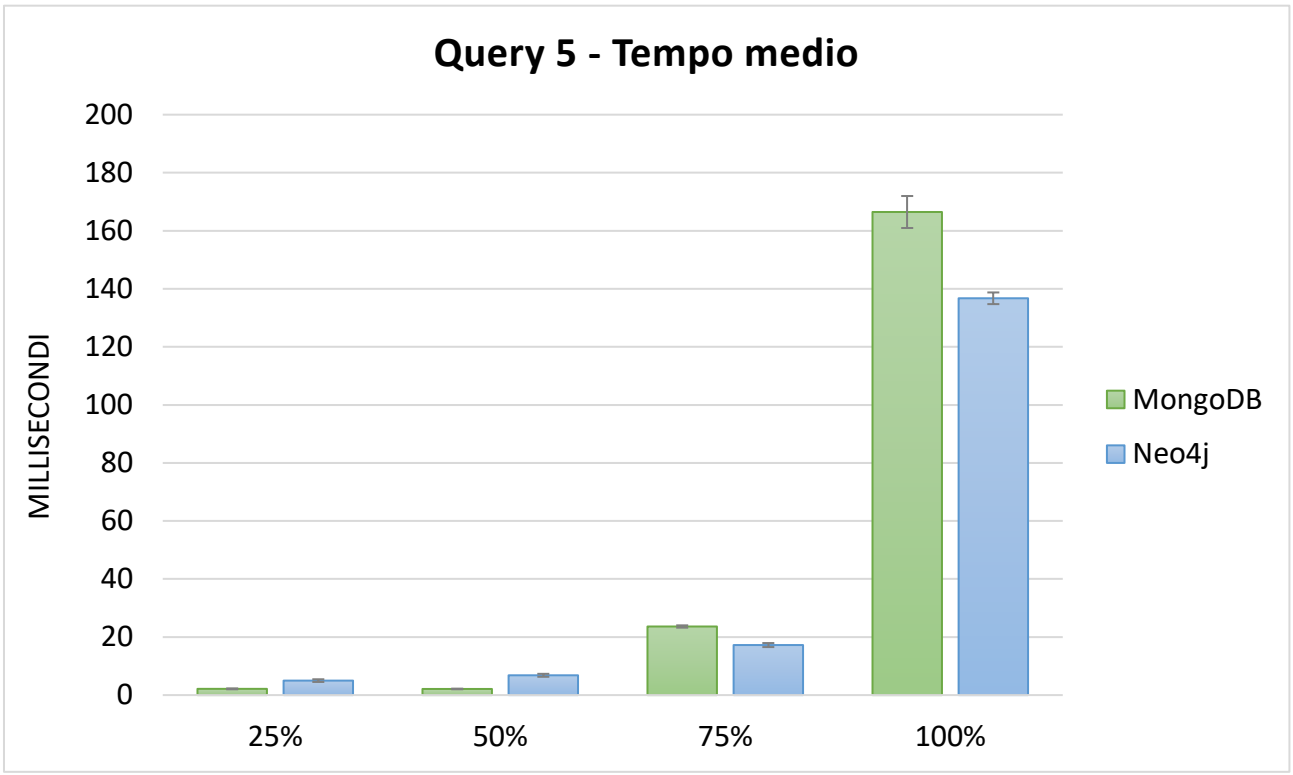
```
const query = { "Calls_done.Cell_site": { $gt: 100 }, "Calls_done.City": { $gt: "V" }, "Calls_done.Call_start": { $gt: 1660000000 },
                $or: [ { "Phone_number": { $gt: "700" } }, { "Last_name": { $gt: "S" } } ] };
```

Neo4j:

```
await session.run('MATCH (n:Call)-[r]-(p:Person) WHERE n.Cell_site > 100 AND n.City > "V" AND n.Call_start > 1660000000',
                  'AND (p.Phone_number > "700" OR p.Last_name > "S") RETURN n',
);
```

	MongoDB 25%	Neo4j 25%	MongoDB 50%	Neo4j 50%	MongoDB 75%	Neo4j 75%	MongoDB 100%	Neo4j 100%
Primo test	17,78	210,23	16,89	254,43	44,13	123,18	258,48	230,46
Tempo medio	2,15	4,99	2,11	6,82	23,63	17,24	166,46	136,73
Dev. standard	0,41	1,30	0,29	1,43	1,10	1,95	15,40	5,62
Conf. 95%	0,15	0,47	0,11	0,51	0,39	0,70	5,51	2,01





CONCLUSIONI

Dai vari test effettuati si può evincere una maggiore efficienza da parte di MongoDB per quanto riguarda la gestione e l'interrogazione di database con piccole quantità di dati; al contrario, in caso di dataset di grandi dimensioni i tempi d'esecuzione delle interrogazioni salgono notevolmente.

Neo4j è più lento nella gestione di database di piccole dimensioni e risulta, grazie alla struttura a grafo che ottimizza le interrogazioni in presenza di dataset di grandi dimensioni, più efficace con grandi quantità di dati.