

Università Degli Studi Di Messina

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

Progetto Basi di dati 2

Database ibrido Hospital Information System

Di Alessandro Russo, matricola 421412

Anno Accademico 2015/2016

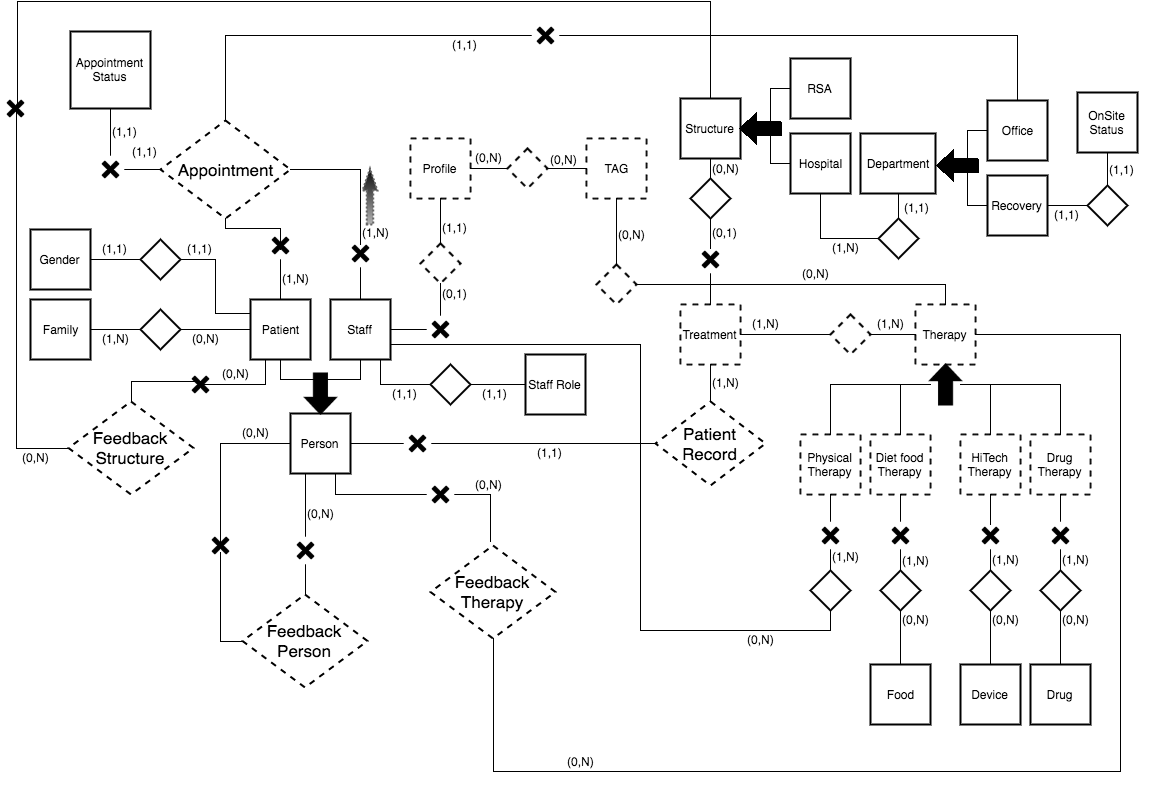
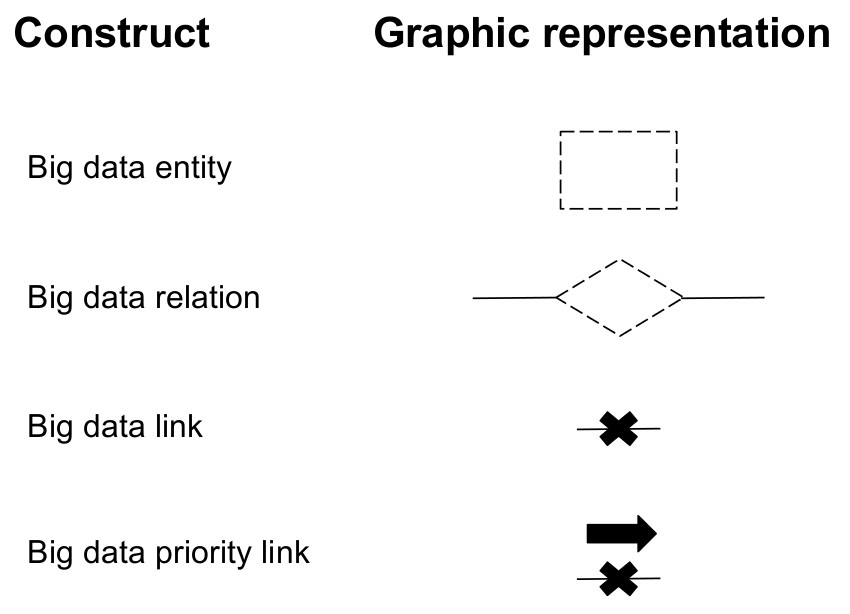
Introduzione

Lo sviluppo di questo progetto ha come scopo la sperimentazione dell’uso di database ibridi come soluzioni in casi di esigenze di alte performance nell’inserimento e selezione dei dati, e allo stesso tempo mantenere relazione tra i dati. Vedremo come progettare e implementare un database ibrido e la comparazione tra le performance di un database SQL (MySQL) e un database NoSQL (MongoDB).

Scenario di riferimento

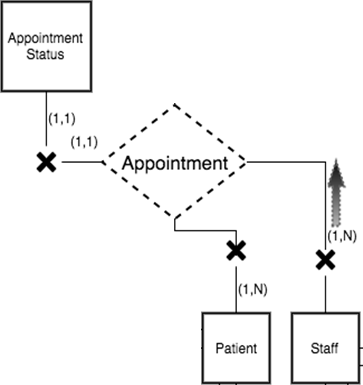
Il progetto cui si riferirà questa tesina è il **Progetto 3**, lo scenario di riferimento e il seguente:

Considerando il seguente schema ER di un Hospital Information System (HIS)

Realizzare un sistema di archiviazione ibrido considerando soluzioni MySQL e MongoDB. Le entità con linea continua dovranno essere memorizzate in MySQL, mentre quelle con linea tratteggiate in MongoDB.

Misurare i tempi di risposta al variare del numero di server del cluster MongoDB e considerando 100, 1000, 10000, 100000, 1000000 di record simulati. In caso di degrado delle prestazioni fermarsi ad un numero di record consentito. Sincronizzare gli esperimenti con i responsabili dei progetti 4 e 5, considerando anche lo stesso input. Graficare i risultati ottenuti per mezzo di istogrammi.

Nello specifico mi concentrerò sulla sezione dello schema ER con le relazioni Appointment Status, Patient, Staff e la tabella generata dalla relazione molti a molti Appointment:

Strumenti utilizzati

Per lo sviluppo del progetto mi sono servito degli AWS in particolare del servizio EC2 di Amazon per la gestione dei nodi e della loro connessione e messa in sicurezza. Ho utilizzato altri strumenti di Amazon per la configurazione dei nodi come l’Elastic IP, Security Group, Key Pairs che vedremo nel dettaglio più avanti. Mi sono servito dei programmi Putty e PuttyGen per la gestione delle chiavi di accesso SSH. Ho utilizzato Photoshop CS6 per le illustrazioni della tesina.

Ho sviluppato la parte applicativa utilizzando il linguaggio di programmazione Python 3 su gedit, connettendomi a MySQL con il driver per python *pymsql* e per la connessione a MongoDB *pymongo*. Per la parte relazionale del progetto ho utilizzato MySQL come RDBMS, invece per la parte NoSQL ho utilizzato MongoDB.

Ho progettato e implementato tutto il progetto su una macchina con Arch Linux.

Per i grafici e la tesina ho utilizzato i programmi Microsoft Word e Microsoft Excel.

Progettazione

AWS

Per realizzare il progetto ed eseguire i test con un differente numero di nodi nel cluster di MongoDB è necessario progettare l’architettura del cluster e dei vari nodi. Per far questo mi sono servito degli AWS (Amazon Web Service). Sono una collezione di servizi di cloud computing (IaaS) on demand forniti da Amazon.

Per utilizzare i servizi Amazon ho creato un account IAM (Ale), per creare un gruppo (Administrator) cui associare le politiche di AministrationAccess. In seguito ho creato un utente (Ale\_user) cui ho assegnato questo gruppo che permette l’accesso a tutti i servizi di Amazon.

EC2

Per creare i miei nodi in particolare ho utilizzato il servizio EC2 (Elastic Compute Cloud) che permette la gestione di macchine virtuali on demand configurando sia l’hardware sia il software.

Key Pairs

Per accedere alle macchine istanziate, ho dovuto creare una coppia di chiavi che mi serviranno come certificato per l’accesso SSH alle mie macchine con S.O. Linux. Le chiavi sono associate a una determinata regione in questo caso ho scelto Francoforte, quindi ho generato la mia coppia di chiavi con il nome di “ale-key-pair-frankufurt”. Tramite il programma PuttyGen ho estratto la mia chiave privata “ale-key-pair-frankufurt.ppk” che utilizzerò per accedere tramite SSH con Putty alle macchine.

Security Groups

Ho creato un Security Group che è possibile associarlo alle istanze per controllare il traffico in entrata e uscita. Serve come firewall di sicurezza, e deve essere configurato correttamente per permettere una corretta comunicazione tra le istanze. Ho chiamato il mio gruppo “ale-sg-frankfurt”, che conterrà la configurazione delle comunicazioni in uscita “Outbound” e in entrata “Inbound”. Ho configurato il traffico in entrata come di seguito:

| **Type** | **Protocol** | **Port Range** | **Source** | |
| --- | --- | --- | --- | --- |
|  | | | | |
| HTTP | TCP | 80 | 0.0.0.0/0 |  |
| All traffic | All | All | 52.29.104.184/32 |  |
| All traffic | All | All | 52.29.250.145/32 |  |
| All traffic | All | All | 52.58.98.149/32 |  |
| All traffic | All | All | 52.29.127.254/32 |  |
| SSH | TCP | 22 | 87.12.198.190/32 |  |
| Custom TCP Rule | TCP | 27017 | 87.12.198.190/32 |  |
| HTTPS | TCP | 443 | 0.0.0.0/0 |  |

E in uscita:

| **Type** | **Protocol** | **Port Range** | **Destination** | |
| --- | --- | --- | --- | --- |
|  | | | | |
| All traffic | All | All | 0.0.0.0/0 |  |

Elastic IP Addresses

Com’è possibile notare nella tabella del traffico in entrata sono presenti gli indirizzi IP che possono comunicare con determinati protocolli con ciascuna macchina che appartiene al gruppo “ale-sg-frankfurt”. Ho creato e associato degli Elastic IP per ciascun nodo del cluster. Un Elastic IP è un indirizzo IP permanente e statico, differente da quello che viene generato alla creazione dell’istanza che può mutare in caso di caduta dell’istanza o se viene fermata. Ho configurato i nodi e gli Elastic IP nel seguente modo:

| **Elastic IP** | **Instance** |  |
| --- | --- | --- |
|  | | |

|  |  |  |
| --- | --- | --- |
| 52.29.104.184 | Node 1 |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| 52.29.250.145 | Node 2 |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| 52.58.98.149 | Node 3 |  |
|  |  |  |

|  |  |
| --- | --- |
| 52.29.127.254 | Node 4 |

Dopo aver correttamente configurato tutti i servizi necessari per la comunicazione tra i nodi e per potervi accedere provvedo alla creazione delle istanze EC2.

Instances

Dalla sezione instances è possibile configurare e lanciare la propria istanza.

Nel primo step della configurazione è possibile scegliere il S.O. dell’istanza da lanciare, ho scelto di creare dei nodi con S.O. Ubuntu Server 14.04 LTS (64bit). Nel secondo step è possibile scegliere la tipologia delle prestazioni della propria macchina (ottimizzate per il calcolo, la memoria o l’immagazzinamento dati). Ho scelto in questo caso una macchina di tipo “t2.micro”, con 1 vCPU, 1 GB RAM, 8 GB SSD e performance di rete Moderate, questa tipologia di macchina è idonea per rimanere nel piano gratuito di Amazon. E’ possibile utilizzare gratuitamente queste istanze per 750 ore al mese che si dimezzeranno per ogni istanza aggiuntiva.

Nei successivi terzo e quarto step è possibile associare direttamente il Security Group creato e la coppia di chiavi all’istanza, cosi da essere pronta all’uso non appena lanciata.

Accesso all’istanza

Dopo aver lanciato l’istanza bisogna attendere qualche secondo perché sia disponibile. Quando nel pannello delle istanze notiamo che il suo stato è “Running” allora è possibile accedere all’istanza. Per accedervi ho utilizzato il programma per accesso SSH Putty, con la seguente configurazione:

Host Name / IP Address : *(Indirizzo Elastic IP del nodo)*

Port: *22*

Saved session: *(Nome nodo)*

Auto-login username: *ubuntu*

Private key file for authentication: *ale-key-pair-frankufurt.ppk*

Dopo aver configurato correttamente la sessione di Putty cliccando su open si avvierà Il terminale dell’istanza creata:

*ubuntu@(ip nodo):~$*

*(Ho riscontrato alcuni problemi di accesso con utente ubuntu, risolto accedendo come root e creando l’utente)*

A questo punto il nodo è pronto per essere configurato per diventare un nodo del cluster MongoDB.

Configurazione nodo

Per organizzare i nodi in modo da identificarli correttamente ho modificato il file “/etc/hostname”. Nel caso del primo nodo il file hostname si presenta cosi:

*nodo1*

con il comando: *sudo nano /etc/hostname*

Ovviamente a ciascun nodo ho associato il relativo nome.

Successivamente ho modificato il file “/etc/hosts” in modo che ogni nome del nodo sia associato al proprio indirizzo IP cosi da essere anche facilmente identificabili. Il file hosts si presenta cosi:

*127.0.0.1 localhost*

*52.29.104.184 node1*

*52.29.250.145 node2*

*52.58.98.149 node3*

*52.29.127.254 node4*

*# The following lines are desirable for IPv6 capable hosts*

*::1 ip6-localhost ip6-loopback*

*fe00::0 ip6-localnet*

*ff00::0 ip6-mcastprefix*

*ff02::1 ip6-allnodes*

*ff02::2 ip6-allrouters*

*ff02::3 ip6-allhosts*

con il comando: *sudo nano /etc/hosts*

Configurazione MongoDB

Per configurare e installare MongoDB nel nodo mi sono servito di alcuni servizi e comandi utilizzati nel terminale.

Dato che MongoDB non è presente nell’Ubuntu’s package management tool è necessario copiare e validare la chiave GPG di MongoDB per utilizzare e installare il pacchetto di MongoDB.

Si effettua copia e validazione tramite questo comando:

*sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10*

Dopo è possibile aggiungere la repository di MongoDB scaricandola e inserendola in un file con una pipe:

*echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo tee /etc/apt/sources.list.d/mongodb.list*

Prima di effettuare l’installazione di MongoDB sul nodo ho effettuato un aggiornamento del sistema con i comandi:

*sudo apt-get update*

*sudo apt-get upgrade*

Ho provveduto a creare la cartella dove risiederanno i database di MongoDB prima della sua installazione cosi, non appena l’installazione sarà completata, MongoDB sarà avviato automaticamente come servizio in background senza nessun errore, utilizzando i seguenti comandi:

*cd /*

*sudo mkdir –p data/db*

Affinché MongoDB possa creare file all’interno delle cartelle create è necessario cambiare i permessi e visualizzare che siano correttamente cambiati, utilizzando i comandi:

*cd/*

*sudo chown –R mongodb:mongodb data*

e controllando con:

*ls -la*

Dopo questi passaggi ho effettuato un installazione di MongoDB con il comando:

*sudo apt-get install mongodb-10gen*

Cluster

Affinché il nodo faccia parte del replica set ho provveduto a modificare il file di configurazione di MongoDB con I seguenti comandi:

*sudo nano /etc/mongodb.conf*

modificandolo con questa configurazione:

*port: 27017*

*replSet = rp0*

e riavviato il servizio con il comando:

*sudo service mongodb restart*

A questo punto avendo un’istanza correttamente configurata ho creato un immagine AMI con i servizi di Amazon e ho avviato altre 3 istanze con la medesima configurazione hardware del primo nodo.

Mi è bastato modificare per ogni nodo il proprio file /etc/hostname con il relativo nome avendo cosi pronti all’uso 4 nodi con MongoDB configurato correttamente e avviato.

Ho modificato solo un nodo per far avviare MongoDB nella porta 30000 per utilizzare il nodo come arbiter.

Per creare il replica set ho avviato il client MongoDB sul nodo1 e utilizzato i comandi per inizializzare un replica e aggiungere i nodi al cluster con i seguenti comandi:

Inizializza il replica set:

*rs.initiate()*

Aggiunge il nodo2:

*rs.add("node2:27017")*

Aggiunge un arbiter nei casi in cui i nodi sono dispari per effettuare la votazione per scegliere il PRIMARY:

*rs.addArb("node3:30000")*

Aggiunge il nodo4:

*rs.add("node4:27017")*

Infine lo status del replica set aveva la seguente configurazione:

*rs.status()*

*{*

*"set" : "rp0",*

*"date" : ISODate("2016-06-08T16:33:27Z"),*

*"myState" : 1,*

*"members" : [*

*{*

*"\_id" : 0,*

*"name" : "node1:27017",*

*"health" : 1,*

*"state" : 1,*

*"stateStr" : "PRIMARY",*

*"uptime" : 288247,*

*"optime" : Timestamp(1465378111, 10000),*

*"optimeDate" : ISODate("2016-06-08T09:28:31Z"),*

*"self" : true*

*},*

*{*

*"\_id" : 2,*

*"name" : "node3:30000",*

*"health" : 1,*

*"state" : 7,*

*"stateStr" : "ARBITER",*

*"uptime" : 27628,*

*"lastHeartbeat" : ISODate("2016-06-08T16:33:26Z"),*

*"lastHeartbeatRecv" : ISODate("2016-06-08T16:33:26Z"),*

*"pingMs" : 0*

*},*

*{*

*"\_id" : 3,*

*"name" : "node2:27017",*

*"health" : 1,*

*"state" : 2,*

*"stateStr" : "SECONDARY",*

*"uptime" : 26747,*

*"optime" : Timestamp(1465378111, 10000),*

*"optimeDate" : ISODate("2016-06-08T09:28:31Z"),*

*"lastHeartbeat" : ISODate("2016-06-08T16:33:26Z"),*

*"lastHeartbeatRecv" : ISODate("2016-06-08T16:33:26Z"),*

*"pingMs" : 0,*

*"syncingTo" : "node1:27017"*

*},*

*{*

*"\_id" : 4,*

*"name" : "node4:27017",*

*"health" : 1,*

*"state" : 2,*

*"stateStr" : "SECONDARY",*

*"uptime" : 25890,*

*"optime" : Timestamp(1465378111, 10000),*

*"optimeDate" : ISODate("2016-06-08T09:28:31Z"),*

*"lastHeartbeat" : ISODate("2016-06-08T16:33:25Z"),*

*"lastHeartbeatRecv" : ISODate("2016-06-08T16:33:26Z"),*

*"pingMs" : 0,*

*"syncingTo" : "node1:27017"*

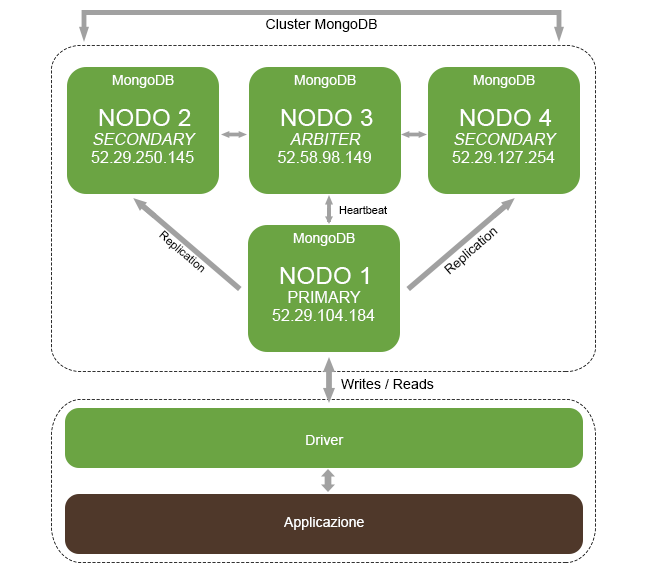
*}*

*],*

*"ok" : 1*

*}*

L’architettura del cluster e delle comunicazioni con l’applicazione sono descritte nella seguente immagine:



Avendo progettato un database distribuito con MongoDB con le caratteristiche di fault tollerance, replica dei dati , alta disponibilità dei dati e potendo gestire i big data sono passato all’installazione in locale di MySQL per la gestione della parte relazionale dello schema ER.

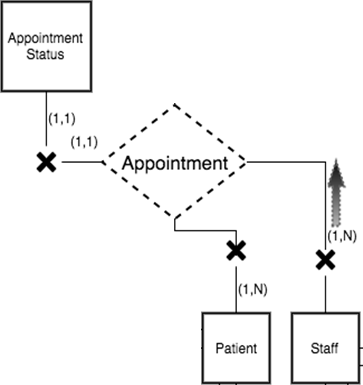
MySQL

Ho installato MySQL in locale sulla mia macchina ArchLinux installando il package MariaDB che è l’implementazione di default di MySQL su Arch Linux con il comando:

*sudo pacman –S mariadb*

Schema ER ibrido

La parte dello schema da prendere in esame comprende 3 entità di cui una è un’entità nata per una relazione molti a molti che dovrà essere memorizzata nel cluster MongoDB mantenendo le relazioni con le entità che saranno memorizzate su MySQL.



Schema MySQL

In base alle entità la struttura del database MySQL sarà nel seguente modo:

Creazione database his:

*CREATE DATABASE his;*

Creazione tabella entità Appointment\_Status:

*CREATE TABLE Appointment\_Status(*

*id INT NOT NULL AUTO\_INCREMENT,*

*status VARCHAR (100),*

*paid VARCHAR (3),*

*PRIMARY KEY ( id )*

*)*

Creazione tabella entità Staff:

*CREATE TABLE Staff(*

*id INT NOT NULL AUTO\_INCREMENT,*

*name\_member VARCHAR(100),*

*surname\_member VARCHAR(100),*

*birthday\_date DATE,*

*gender VARCHAR(1),*

*phone VARCHAR(30),*

*date\_contract DATE,*

*specialty VARCHAR(150),*

*PRIMARY KEY ( id )*

*)*

Creazione tabella entità Patient:

*CREATE TABLE Patient(*

*id INT NOT NULL AUTO\_INCREMENT,*

*name VARCHAR(100) NOT NULL,*

*surname VARCHAR(100),*

*birthday\_date DATE,*

*gender VARCHAR(1),*

*phone VARCHAR(30),*

*disease VARCHAR(100),*

*date\_recovery DATETIME,*

*PRIMARY KEY ( id )*

*)*

Schema MongoDB

Dato che MongoDB è un database NoSQL, non supporta le relazioni o le chiavi esterne o primarie, ma è possibile modellare lo schema dati in modo che i dati siano relazionati tra loro incapsulando i dati in base alle loro relazioni. In questo caso ho provveduto a creare 3 tipi di collezioni per ottimizzare le query di ricerca. Le collezioni sono:

*Patient*

*Staff*

*Appointment\_Status*

Ogni collezione incapsula al suo interno I dati relativi alle proprie relazioni, come dimostrato in questo dataset di esempio: *{*

*"Staff" : [{*

*"gender" : "m",*

*"date\_contract" : "1945-03-04",*

*"name\_member" : "Karnes",*

*"Appointment" : {*

*"Appointment\_Status" : {*

*"id" : 1,*

*"paid" : "No",*

*"id\_appointment" : 1,*

*"status" : "Wait"*

*},*

*"id" : 1,*

*"Patient" : {*

*"surname" : "Acrey",*

*"name" : "Paek",*

*"gender" : "m",*

*"phone" : "222381564671",*

*"disease" : "Pulmonary embolism",*

*"id" : 1,*

*"date\_recovery" : "2013-04-04 18:05:37",*

*"birthday\_date" : "1984-09-24"*

*},*

*"appointment\_date" : "2011-06-18 19:41:10"*

*},*

*"surname\_member" : "Chojnowski",*

*"specialty" : "Anesthesia",*

*"phone" : "953854136361",*

*"id" : 1,*

*"birthday\_date" : "2000-10-18"*

*}*

*],*

*"Patient" : [{*

*"Appointment" : {*

*"Appointment\_Status" : {*

*"id" : 1,*

*"paid" : "No",*

*"id\_appointment" : 1,*

*"status" : "Wait"*

*},*

*"id" : 1,*

*"Staff" : {*

*"surname\_member" : "Chojnowski",*

*"specialty" : "Anesthesia",*

*"gender" : "m",*

*"phone" : "953854136361",*

*"id" : 1,*

*"date\_contract" : "1945-03-04",*

*"name\_member" : "Karnes",*

*"birthday\_date" : "2000-10-18"*

*},*

*"appointment\_date" : "2011-06-18 19:41:10"*

*},*

*"gender" : "m",*

*"disease" : "Pulmonary embolism",*

*"surname" : "Acrey",*

*"name" : "Paek",*

*"phone" : "222381564671",*

*"id" : 1,*

*"date\_recovery" : "2013-04-04 18:05:37",*

*"birthday\_date" : "1984-09-24"*

*}*

*],*

*"Appointment\_Status" : [{*

*"Appointment" : {*

*"Staff" : {*

*"surname\_member" : "Chojnowski",*

*"specialty" : "Anesthesia",*

*"gender" : "m",*

*"phone" : "953854136361",*

*"id" : 1,*

*"date\_contract" : "1945-03-04",*

*"name\_member" : "Karnes",*

*"birthday\_date" : "2000-10-18"*

*},*

*"id" : 1,*

*"Patient" : {*

*"surname" : "Acrey",*

*"name" : "Paek",*

*"gender" : "m",*

*"phone" : "222381564671",*

*"disease" : "Pulmonary embolism",*

*"id" : 1,*

*"date\_recovery" : "2013-04-04 18:05:37",*

*"birthday\_date" : "1984-09-24"*

*},*

*"appointment\_date" : "2011-06-18 19:41:10"*

*},*

*"id" : 1,*

*"paid" : "No",*

*"id\_appointment" : 1,*

*"status" : "Wait"*

*}*

*]*

*}*

Implementazione

Per automatizzare i processi d’inserimento dei dataset sia su MySQL che su MongoDB ho creato un programma in Python 3 che effettua le connessioni al cluster MongoDB e al client MySQL ed effettua le query di inserimento e ricerca generando degli istogrammi di confronto.

Il programma è suddiviso in 3 file che ho programmato:

Script principale:

*his.py*

Modulo connessione MySQL (Classe *mysqlConn*):

*mysqlConn.py*

Modulo connessione MongoDB (Classe *mongodbConn)*:

*mongodbConn.py*

Per generare i grafici utilizzo il modulo:

*pylab*

Per la connessione a MongoDB:

*pymongo*

Per la connessione a MySQL:

*pymysql*

Il programma è suddiviso in 2 parti, la prima parte effettua le query di inserimento per 100, 1000, 10000, 100000 dati e ricerca su MySQL e la seconda parte effettua le query di inserimento per 100, 1000, 10000, 100000 dati e ricerca su MongoDB.

Nel caso delle query di SELECT e FIND saranno effettuate 30 volte e saranno presi in considerazione i tempi medi.

Nel caso dei test per MongoDB per ogni dataset (100, 1000, 10000, 100000) saranno effettuati test di inserimento e ricerca riconfigurando il replica set ad ogni test, con le seguenti configurazioni:

Il primo test con 1 nodo PRIMARY più il nodo3 ARBITER,

Il secondo test con il nodo1 PRIMARY il nodo2 SECONDARY più un nodo3 ARBITER.

Il terzo test con nodo1 PRIMARY il nodo2 SECONDARY il nodo4 SECONDARY e l’ARBITER con il nodo3.

Alla fine di ogni test verrà visualizzato un istogramma contenente i risultati dei test. Sull’asse delle ordinate verrà visualizzato il tempo, mentre sull’ascisse gli istogrammi in coppia, inserimento e ricerca per ciascun dataset.

Per le operazioni su MySQL di creazione, eliminazione, inserimenti e selezioni nelle tabelle verranno effettuati automaticamente tramite il programma scritto in Python 3 “his.py” rispettivamente con i comandi:

*thisMysqlConn = mysqlConn() # Connessione al DB*

*thisMysqlConn.createHisSchema() # Creo lo schema MySQL*

*thisMysqlConn.resetHisMySQL() # Resetto il DB*

*mySqlInsertTime = thisMysqlConn.insertMySQL(numberInsert) # Inserisco i dataset in MySQL, (numberInsert è la dimensione dataset) e torna su mySqlInsertTime il tempo di esecuzione query*

*mySqlSelectTime = thisMysqlConn.selectMySQL() # Seleziono i dati in MySQL e torna su mySqlSelectTime il tempo di esecuzione medio su 30 query*

Per le operazioni su MongoDB di creazione, eliminazione, inserimenti e selezioni nelle collezioni verranno effettuati automaticamente tramite il programma scritto in Python 3 “his.py” rispettivamente con i comandi:

*thisMongoConn = mongodbConn() # Connessione al DB*

*thisMongoConn.resetHisMongoDB() # Resetto il DB*

*mongoDbInsertTime = thisMongoConn.insertMongoDB(numberInsert) # Inserisco il dataset in MongoDB e torna il tempo do esecuzione query (numberInsert è la dimensione dataset)*

*mongoDbSelectTime = thisMongoConn.selectMongoDB() #Effettuo la selezione e mi faccio tornare il tempo medio 30 query di selezione*

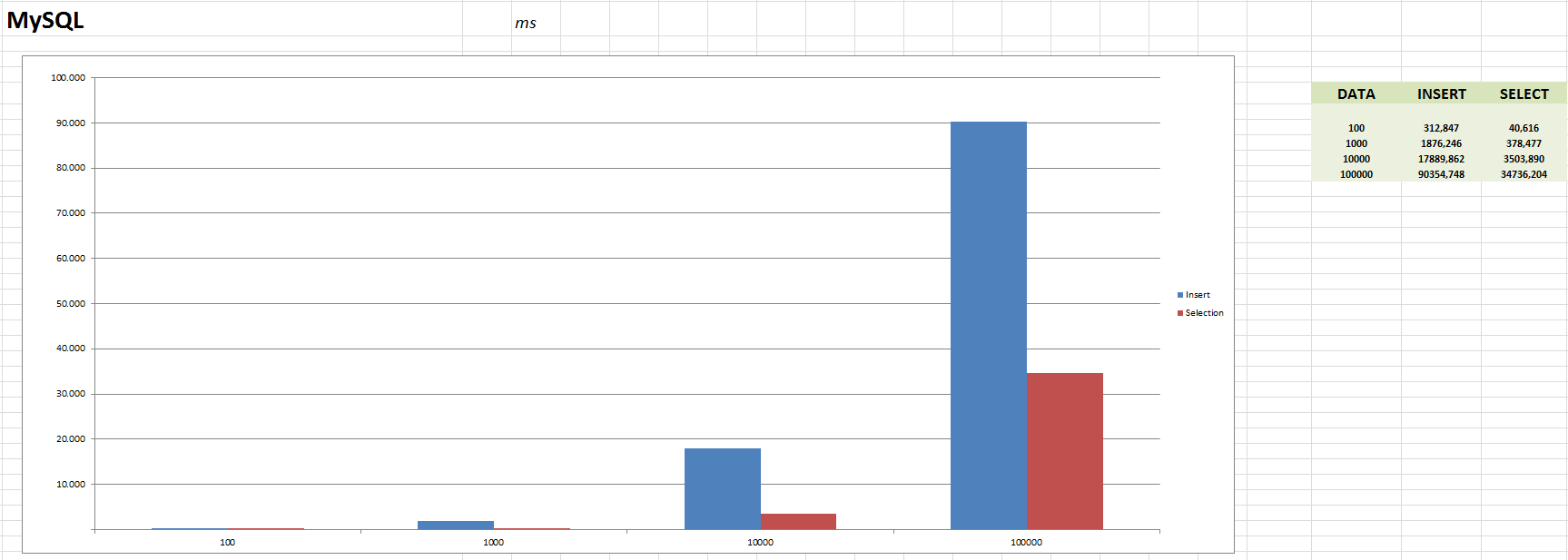
Esperimenti

I risultati ottenuti nei test sono contenuti nel file *test.xlsx*, *test.txt* e nei file *mysql\_test.png,* *mongodb\_test.png*.

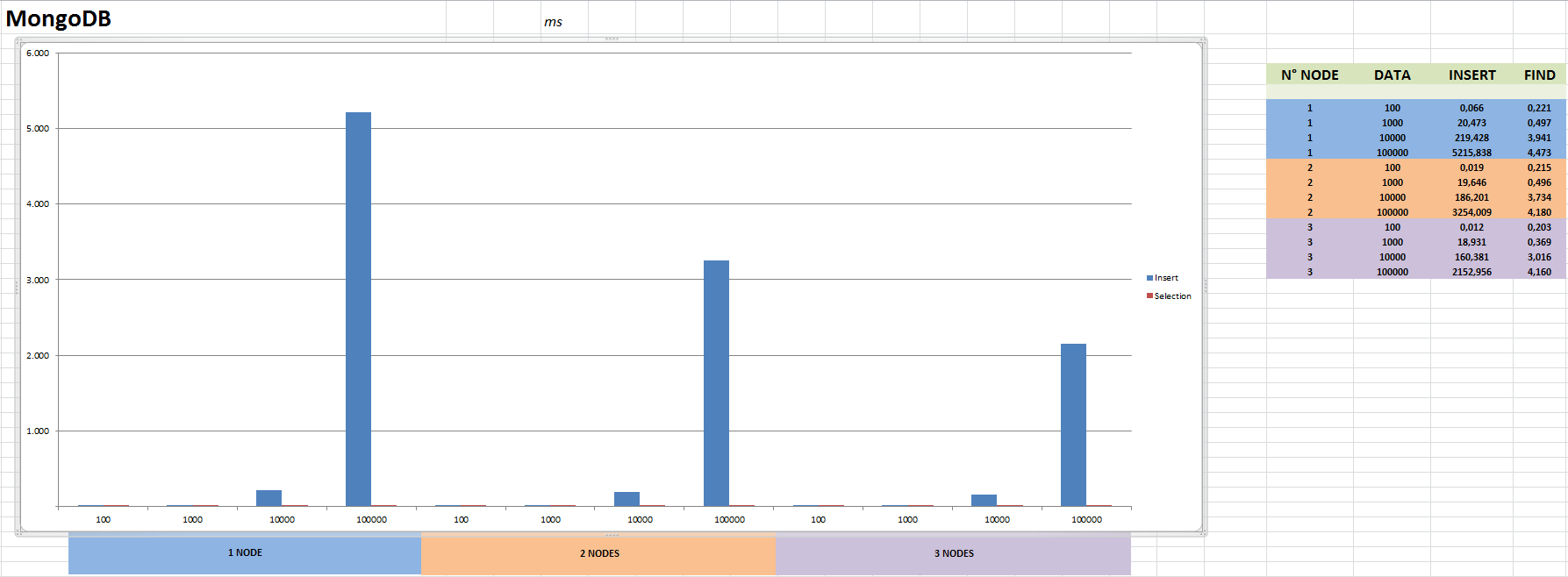
Gli esperimenti sono stati condotti utilizzando l’architettura e i programmi descritti in precedenza, automatizzati tramite un’applicazione Python che effettua i test.

Per i test MySQL ho utilizzato una macchina in locale, non appena l’applicazione viene avviata provvede a pulire (se esistono le tabelle) e creare le tabelle per i test. Successivamente provvede a effettuare gli *INSERT* per ogni dataset su ogni tabella ed effettuare le relative query di *SELECT* per ogni tabella. Per le query di *SELECT* i test sui tempi di esecuzione sono effettuati prendendo i tempi medi di 30 query. Ho eliminato i dati prima di ogni nuovo *INSERT* cosicché i test vengono effettuati sui dati effettivi indicati dal test (ad esempio su 1000 e non su 1100 come accadrebbe).

Di seguito riporto i risultati dei test su MySQL in un grafico con istogrammi:

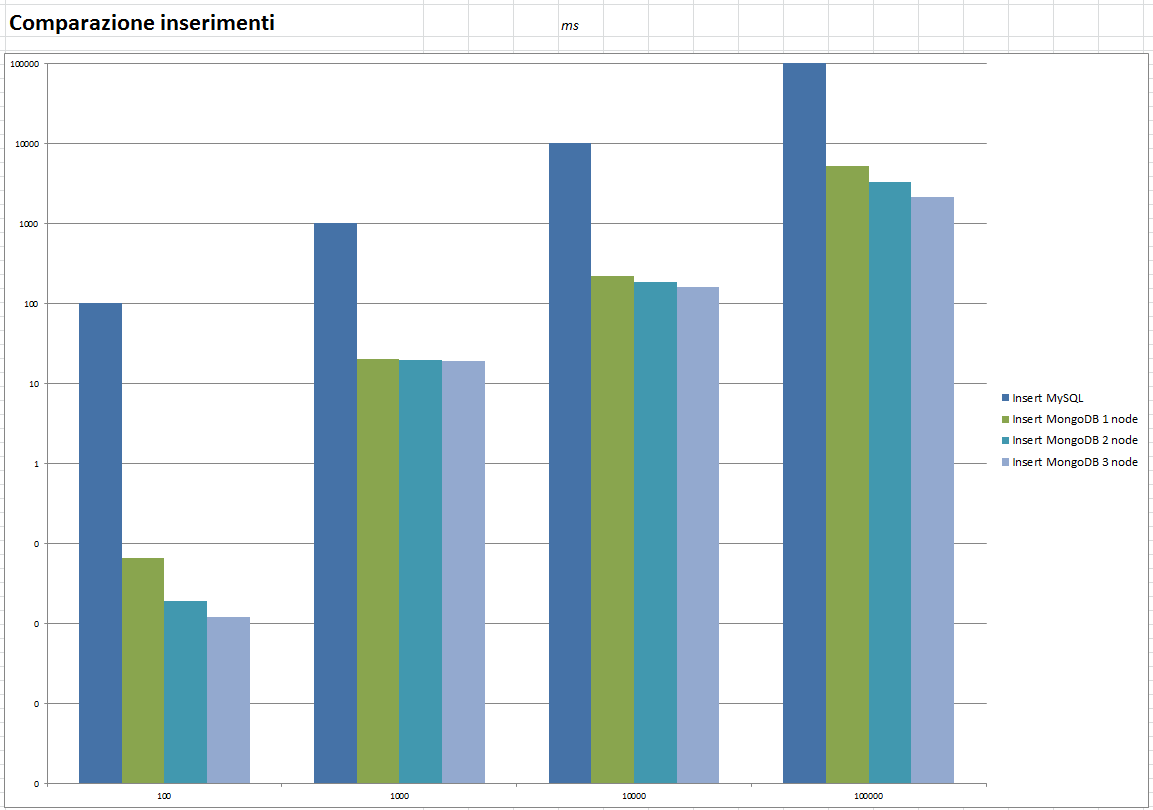
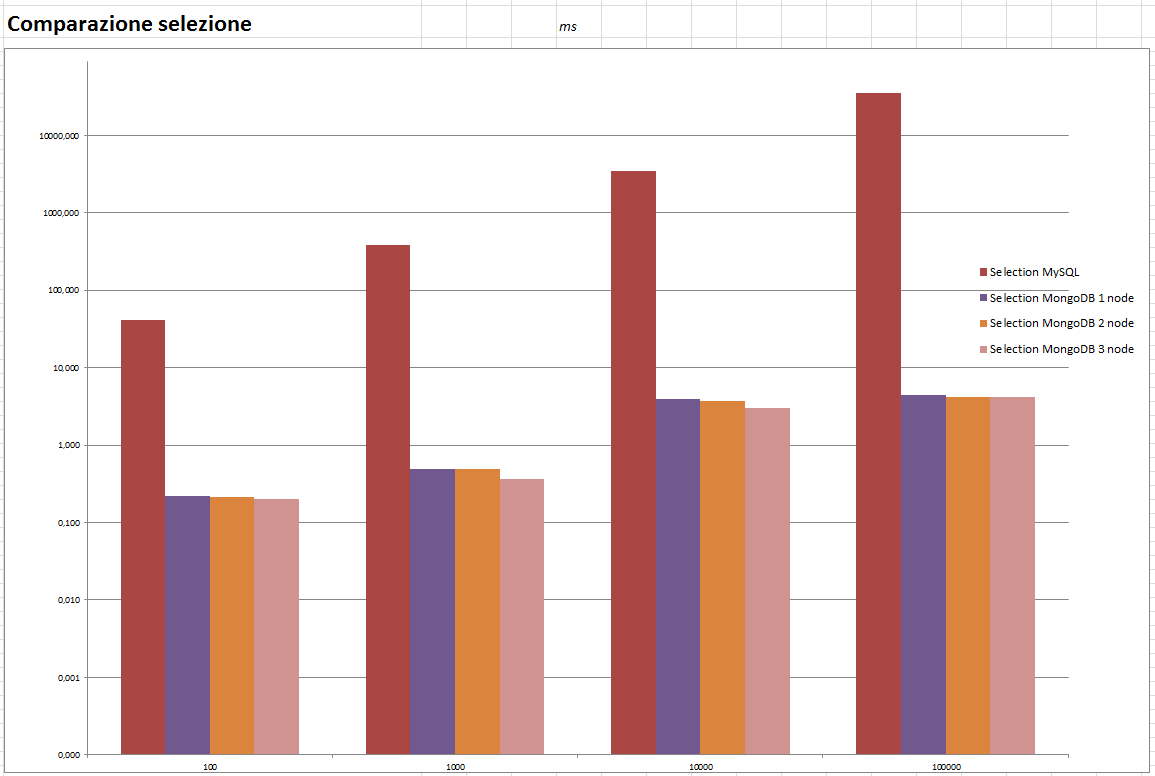


Per i test su MongoDB l’applicazione segue la stessa falsa riga dei test per MySQL, solo che i test effettuati per ogni dataset vengono effettuati su un cluster variando il numero di nodi da 1 a 3 per evidenziare i benefici relativi all’uso di un database distribuito. Come per i test su MySQL elimino i dati prima di ogni test e per le query di find prendo in considerazione i tempi medi su 30 query di find evitando la prima. Per una precisa misurazione dei tempi di query ho utilizzato il metodo *explain()* per ottenere i tempi relativi all’esecuzione delle query.

Di seguito riporto i risultati dei test su MongoDB in cluster di 1,2 e 3 nodi:

Per marcare le differenze di performance tra i database presi in esame, ho inserito i dati in due grafici a istogrammi, uno contenente gli inserimenti di MySQL e MongoDB e il secondo contenente le selezioni di MySQL e MongoDB. Per evidenziare meglio l’andamento degli istogrammi ho configurato il grafico con una scala logaritmica.

Di seguito sono illustrati i grafici di comparazione tra MySQL e MongoDB:



Conclusioni

E’ bene puntualizzare che i test per gli inserimenti e selezione su entrambi i database sono stati eseguiti su macchine con hardware pressoché simile, con l’unica differenza che per MongoDB i test sono stati eseguiti su un cluster di macchine. E’ bene ricordare che la velocità di rete da cui si eseguono i test, può incidere sui risultati (i test sono stati eseguiti con una connessione D 5.73 Mbps U 0.30 Mbps). MongoDB è stato ottimizzato nella stesura dello schema, che facilita l’esecuzione delle query di find. E’ stato ottimizzato anche per gli inserimenti, dato che non era necessario ricevere conferma di avvenuta scrittura è stato disabilitato il messaggio di *acknowledge* con *w = 0,* questo permette di velocizzare ulteriormente gli inserimenti. Da questo si è evinto che all’aumentare dei nodi nel cluster MongoDB i tempi di inserimento e selezione (anche se di poco dovuto al caching delle query di selezione) tendono a diminuire, è possibile apprezzare questo fenomeno in casi di grossi inserimenti di dati, nei quali i tempi non tendono ad aumentare esponenzialmente come nei casi di MySQL ed è per questo che l’utilizzo di database distribuiti, NoSQL come MongoDB sono adatti per la gestione dei Big Data.