RELAZIONE PROGETTO FINALE DI BIG DATA

Gruppo Miel Pops

A cura di Gaetano Bonofiglio e Veronica Iovinella

Sommario

Il progetto: SQL vs NoSQL vs NewSQL

Il progetto svolto riguarda l'analisi di diversi DBMS, in particolare si concentra sul confronto tra le tecnologie SQL, NoSQL e le emergenti NewSQL.

Con l'avvento dei Big Data si sono poste nuove sfide da affrontare per quanto riguarda la sostenibilità delle performance al crescere della base di dati. L'enorme mole di dati e la richiesta intensiva di operazioni al secondo ha portato alla naturale evoluzione dei classici server verso una scalabilità orizzontale, portando alla nascita dei cluster di calcolatori e la costruzione di data center dedicati.

Come si pone la tecnologia NewSQL in questo contesto? I classici database relazioni continuano ad avere una certa attrattiva per via del modello, adattissimo alla rappresentazione a oggetti, e le proprietà ACID delle transazioni garantite che restano una necessità per diverse applicazioni di casi d'uso reali.
L'idea alla base dei sistemi NewSQL è quella di portare il classico database relazionale su cluster, sfruttando i nodi non solo per la ridondanza dei dati, ma dividendo effettivamente il carico e il dataset in blocchi.
Gli approcci possibili sono diversi, e vanno da architetture shared-nothing completamente nuove, ad ottimizzazioni di engine già esistenti per il trasporto su cluster (es: MemSQL, MySQL Cluster).

Trovando il database frammentato tra i diversi nodi, le possibili problematiche riguardo ad operazioni tipicamente relazionali (i.e. join) restano teoricamente le stesse dei già ben noti sistemi NoSQL.

Il nostro esperimento si svolgerà quindi nel seguente modo:

- Proponiamo un'analisi dei sistemi presi in esame, verificando le proprietà delle architetture.
- Verifica delle prestazioni su operazioni quali:
    - Inserimenti
    - Ricerca non indicizzata (ove possibile)
    - Ricerca indicizzata
    - Aggregazioni (ove possibile)
    - Join (ove possibile)
    - Throughput di operazioni (letture e scritture) al secondo
- Le operazioni di cui sopra verranno eseguite:
    - Su nodo singolo
    - Su cluster (ove possibile)

I sistemi presi in esame sono:

- Sistemi **SQL**:
    - Postgres
    - MySQL
- Sistemi **NoSQL**:
    - MongoDB
    - Redis

    - Neo4j
    - OrientDB
    - Cassandra
- Sistemi **NewSQL**:
    - MemSQL
    - Voltdb
    - CockRoach
    - NuoDB

Per eseguire un'analisi precisa delle prestazioni, utilizzeremo come piattaforma **Docker**, avviando ogni sistema in un container, in modo da dedicare le stesse risorse di calcolo a tutti i sistemi.

CAP Theorem

È bene spendere qualche parola a proposito del CAP Theorem. Ciò che sostengono i creatori di molti sistemi NewSQL sembra contravvenire al teorema, dal momento che promettono scalabilità, fault tolerance, consistenza e disponibilità, caratteristiche che non dovrebbero coesistere tutte insieme. Tuttavia in realtà i 3 elementi del CAP Theorem non dovrebbero essere interpretati in modo assoluto e integrale, dal momento che è possibile scegliere un'area che include parzialmente i vari elementi, trovando compromessi ed espedienti di vario genere. Una tecnica utilizzata da tutti i sistemi NewSQL non basati su DBMS SQL pre-esistenti è ad esempio utilizzare dimensioni estremamente ridotte per i singoli database (solitamente 64 MB) in modo da renderne semplice la gestione.

Architetture dei DBMS

Sistemi SQL

Postgres

Postgres è un classico DBMS relazionale, probabilmente il più performante. Utilizza un'architettura client/server ed un sistema di transazioni su connessione TCP/IP per l'esecuzione delle operazioni. Le operazioni del nostro esperimento sono tutte supportate, tuttavia non esiste una versione in cluster ufficiale di tale sistema, per cui non varrà preso in esame per quel frangente. C'è da dire tuttavia che Cockroach, un DBMS NewSQL che abbiamo esaminato, prende Postgres come target, emulandone tutte le funzionalità.

MySQL

MySQL è il secondo DMBS relazionale che prendiamo in esame ed il più famoso e diffuso.
L'architettura è sempre di tipo client/server, e le operazioni sono chiaramente tutte supportate.

A differenza di Postgres, esiste una versione cluster di MySQL supportata ufficialmente.
L'architettura è di tipo shared-nothing, per cui ogni nodo opera in maniera indipendente.
I nodi si distinguono tra SQL nodes, che eseguono MySQL server, e i Data Nodes, che contengono diversi blocchi del dataset. Questa divisione permette ad ogni SQL node di avere accesso in ogni momento all'intero database, garantendo le proprietà delle transazioni.
Questo tipo di architettura inoltre permette di non avere single point of failure, in quanto ogni nodo

resta indipendente, mentre per aumentare la robustezza basta introdurre un fattore di replicazione dei data nodes.

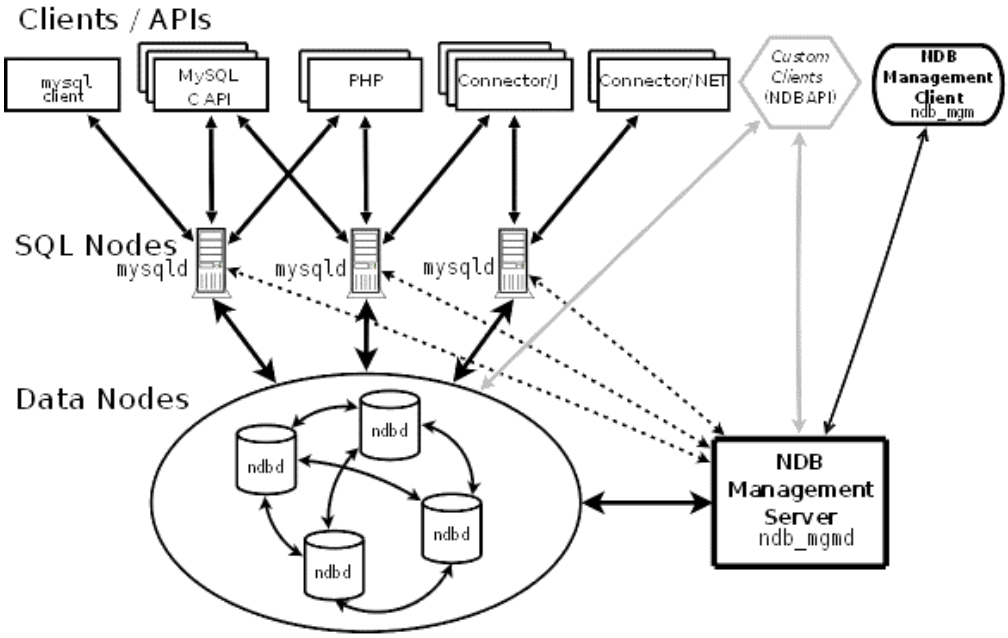


Figura 1 Architettura di MySQL Cluster

Sistemi NoSQL

MongoDB

MongoDB è un document-store database, open source e con una community estremamente attiva, che permette di esprimere attraverso la definizione dello schema anche relazioni tra oggetti, offrendo una gran varietà di query possibili per interrogare i documenti. Data la sua natura, non supporta tuttavia le operazioni di join (eccetto lookup), alle quali si può tuttavia ovviare con una apposita definizione dello schema e inserendo un certo grado di ridondanza. Le funzioni di aggregazione sono tuttavia implementate, attraverso il paradigma MapReduce.

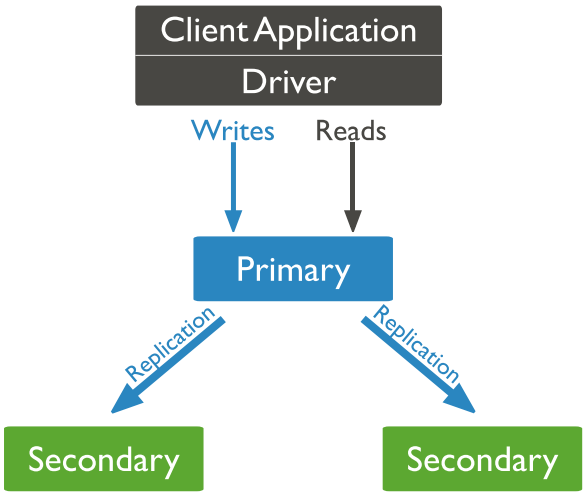


Figura 2 MongoDB Replica Set

Mongo ha la possibilità di scalare orizzontalmente, dividendo il dataset in shard secondo un'architettura master/slave. Prevede inoltre la possibilità di indicare un Replica set, ovvero un fattore di replicazione dei dati.

Nonostante sia un sistema NoSQL, Mongo ammette un certo livello di transazioni ACID, ovvero a livello di documento (anche se è in fase di sviluppo la consistenza a livello di collezione).

Redis

Redis è un particolarissimo database in-memory che opera su coppie key-value. Permette all'occorrenza di persistere i dati in memoria secondaria.

Per via della particolare architettura, Redis non ammette parecchie delle operazioni che sono invece tipiche degli altri sistemi: tra queste la ricerca non indicizzata, alla quale si può ovviare solo aggiungendo un livello di indicizzazione sui valori, e le operazioni di join, poiché si tratta di un'operazione che coinvolge i valori e non le chiavi.

Esiste una versione cluster (sharding) di Redis, che può scalare fino a 1000 nodi, ma porta con sé una forte limitazione perché non supporta le operazioni multi-key nel caso di resharding manuale, ma solo le operazioni single-key. Ciò è dovuto all'architettura in-memory del DBMS.

La replicazione è ottenuta attraverso l'architettura master/slave con un meccanismo publisher/subscriber, e può essere sviluppata su più livelli, permettendo ad uno slave di essere master di un altro slave.

Nonostante le limitazioni, Redis trova ampio utilizzo in particolari casi d'uso, ad esempio implementazioni di code o caching, e proprio grazie a queste limitazioni offre prestazioni elevatissime (è probabilmente il DBMS più performante al momento, anche secondo i nostri test).

Neo4j

Neo4j è un graph database e permette tutte le operazioni previste dal nostro esperimento e tipiche dei modelli relazionali e garantisce transazioni ACID.

Nella versione cluster, l'architettura è di tipo master/slave shared-nothing con replicazione totale. Le letture vengono distribuite sui nodi da un load balancer, mentre il master è l'unico nodo adibito alle letture. Sebbene il dataset non sia distribuito ma replicato sui nodi, la struttura a grafo permette di sfruttare principi di località dei nodi adiacenti.
La tolleranza ai guasti è garantita dalla ridondanza dei nodi, e nel caso del fault di un nodo master questo viene sostituito da un nodo slave attraverso un meccanismo di elezione.

OrientDB

OrientDB è un database multimodello: supporta grafi, documenti, oggetti e coppie key-value, può essere schema-less, mixed-schema e schema-full. Garantisce l'acidità delle transazioni. I dataset relazionali vengono rappresentati esclusivamente attraverso la struttura a grafo, per la quale esistono ottimizzazioni nella navigazione. Non è un DBMS scelto solitamente per le performance ma per le funzionalità speciali che offre (ad esempio notifiche push) e la generalità del modello.

L'architettura del cluster è di tipo multi-master, con un meccanismo di auto discover tramite messaggi broadcast per far partecipare i server al cluster. All'interno del singolo cluster ogni nodo è un master, ovvero contiene una replica dei dati ed è perciò indipendente. Ad un livello superiore si possono definire più cluster indipendenti in sharding, ovvero in ogni cluster abbiamo uno shard di una classe di oggetti (ogni classe viene frammentata su diversi cluster).

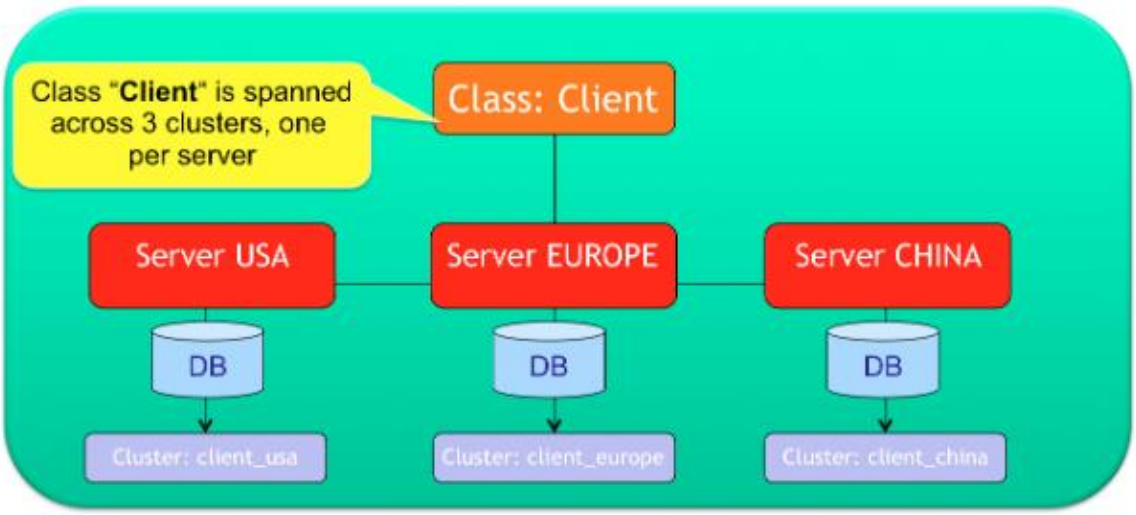


Figura 3 OrientDB Cluster sharding

Cassandra

Cassandra offre un data model ibrido tra key-value e column-based. I record sono identificati da row, che vengono organizzati in "tabelle" dette column-family. Ogni row ha una primary key composta, e una parte di questa è la partition key, che viene utilizzata per la distribuzione su cluster.

Le operazioni supportate sono tutte quelle previste dal nostro esperimento, ad eccezione del join che non è compatibile con il modello di Cassandra: si suggerisce infatti di ridondare e denormalizzare l'informazione.

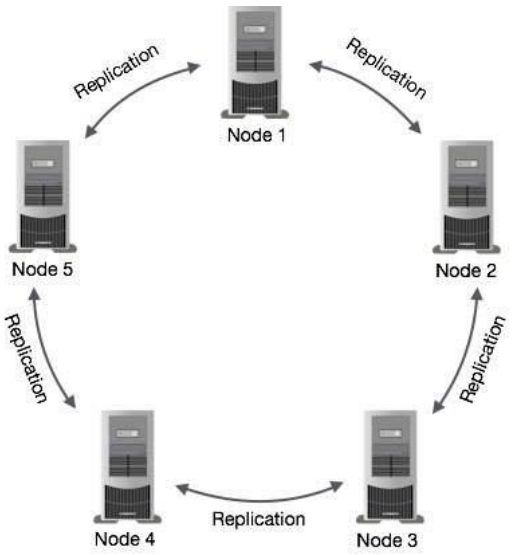


Figura 4 Cassandra cluster architecture

La struttura del cluster è di nodi di replicazione in peer-to-peer. Ogni nodo contiene l'intero dataset e può ricevere richieste dal client. Quando riceve una richiesta prende il ruolo di coordinatore del cluster, e la esaurisce inviandola al nodo più adatto, per bilanciamento di carico o per data di aggiornamento. Se un nodo si trova con dati non aggiornati, viene eseguita una operazione detta read repair ed il dato viene aggiornato alla versione più recente.

La struttura non ha single point of failure, ed eventuali problemi o fault vengono rilevati attraverso il gossip protocol.

Sistemi NewSQL

MemSQL

MemSQL è un database multi modello (relazionale, chiave-valore, documento, geospaziale) in-memory distribuito, con interfaccia ad alto livello basata su MySQL (ne condivide perfino la shell). Essendo relazionale, supporta tutte le operazioni del nostro esperimento. Inoltre al suo interno possiede engine per streaming e data warehousing e supporta esecuzione in parallelo mediante più worker.

L'architettura del cluster è di tipo shared-nothing con sharding del dataset sui nodi. I nodi del cluster possono essere di due tipi: aggregator nodes e leaf nodes. Gli aggregator nodes sono i nodi che ricevono le richieste dal client e conoscono la distribuzione sul resto del cluster, per cui inviano le richieste ai leaf nodes in maniera opportuna. I leaf nodes sono i nodi che contengono effettivamente gli shard del database, e possono rispondere alle richieste. Le risposte dei diversi leaf nodes vengono raccolte ed aggregate dagli aggregator nodes.
Data la struttura, non abbiamo single point of failure, in quanto si può introdurre un fattore di replicazione sia sugli aggregator nodes che sui leaf nodes.

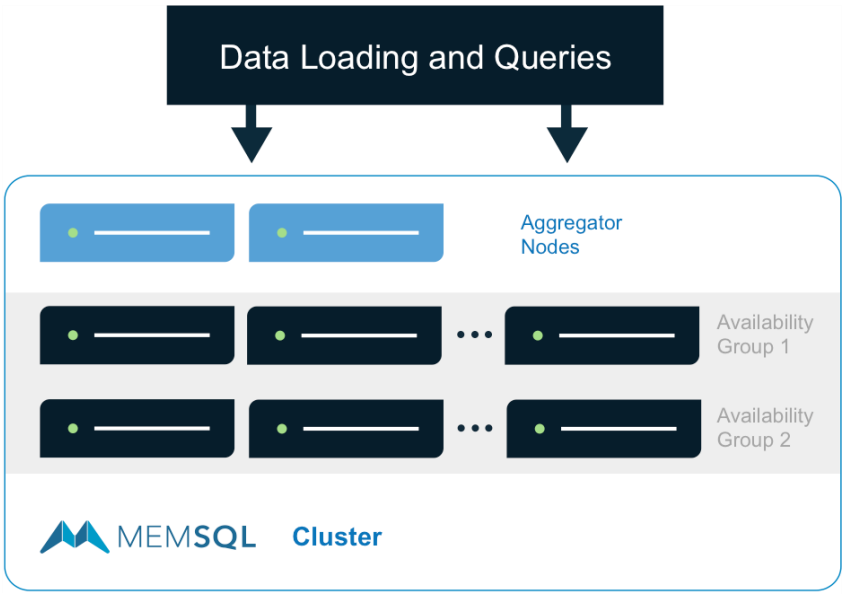


Figura 5 MemSQL Cluster architecture

VoltDB

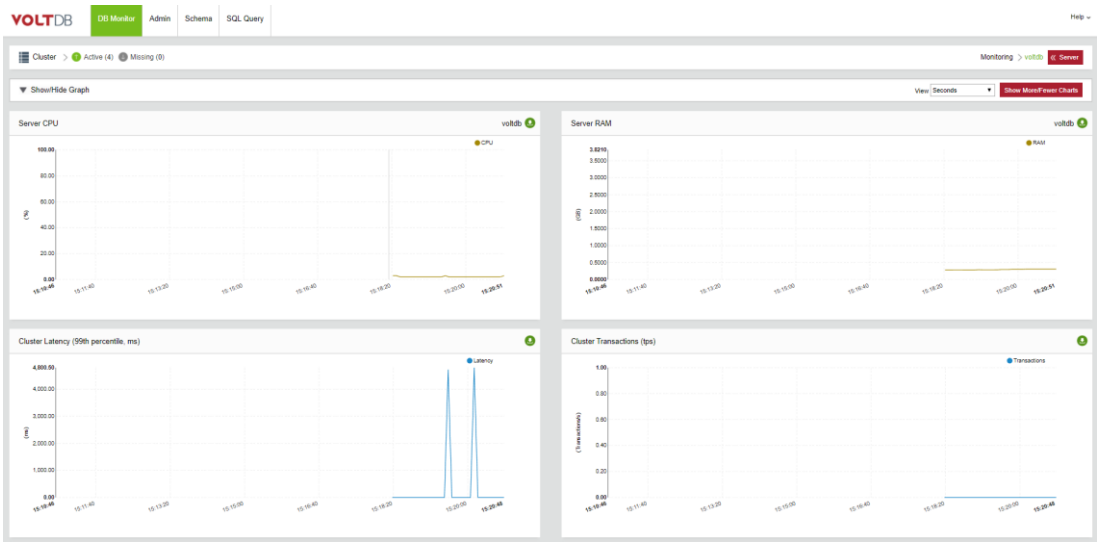


Figura 6 VoltDB Web Interface

VoltDB è un database relazionale in-memory distribuito con architettura shared-nothing. Le operazioni SQL classiche e le proprietà ACID sono supportate.

Il cluster è composto da diversi nodi in sharding, in una configurazione multi-master. Ogni nodo è indipendente e, all'occorrenza, può essere prevista una replicazione dei dati su più nodi, in particolare se si tratta di tabelle piccole che subiscono molte letture. Le operazioni vengono svolte sul nodo contenente la partizione di interesse. Nel caso di operazioni multi-partizione, uno dei nodi coinvolti viene elevato a coordinatore dell'operazione, divide la query sugli altri nodi e raccogli le risposte, aggregandole.

Grazie alla recente aggiunta della funzionalità di replicazione, è stato eliminato anche l'inconveniente del single point of failure.

Va specificato che VoltDB non è pensato per cluster di piccole dimensioni, e scala linearmente fino a 120 nodi, per poi avere cali di performance se i nodi superano questa soglia.

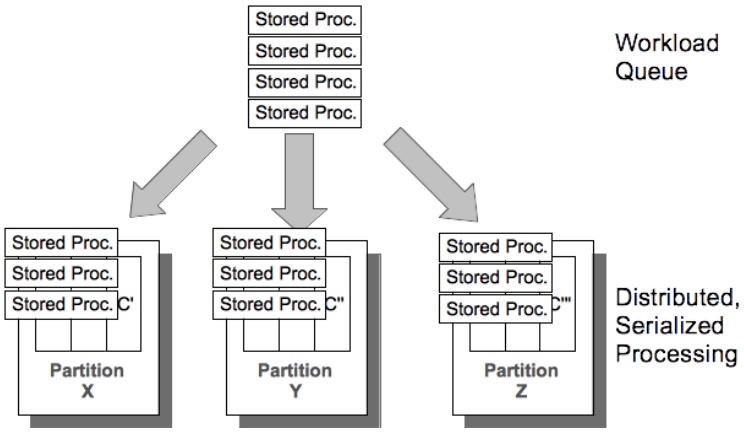


Figura 7 VoltDB Serialization

Come mostrato in figura, VoltDB offre una Web UI per il monitoring del database e del cluster.

Cockroach

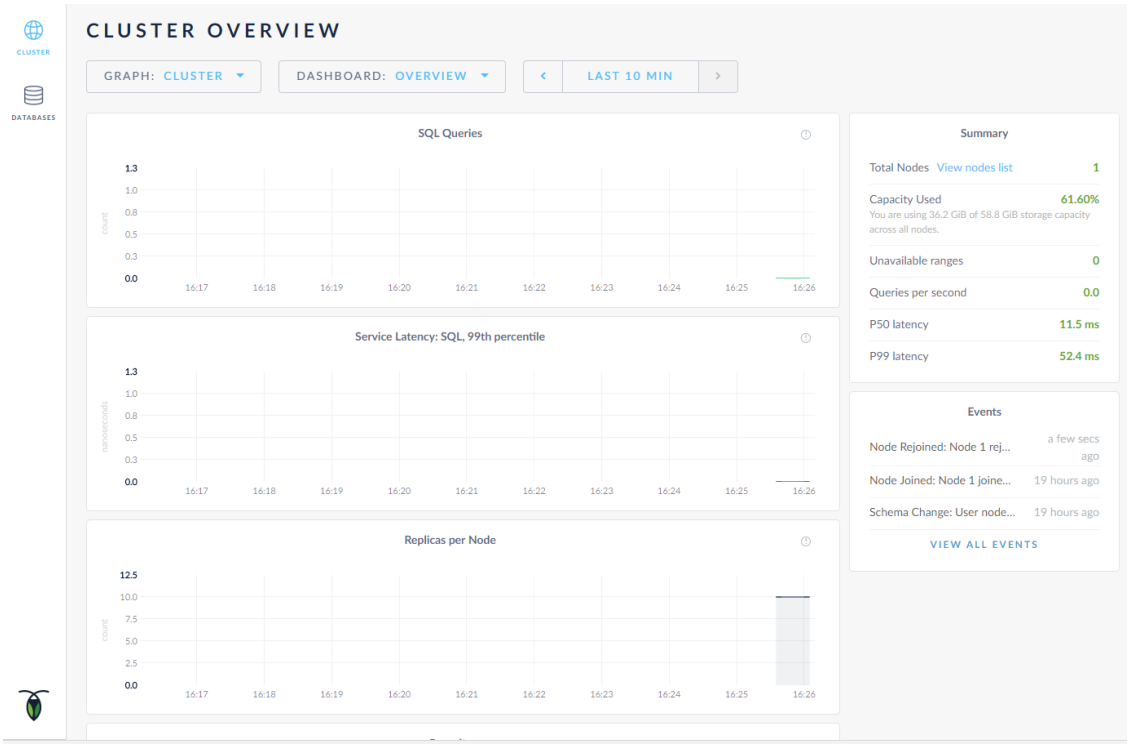


Figura 8 Cockroach web ui

Progetto open-source basato su Google Spanner, Cockroach è un DBMS il cui obiettivo primario è la survivability (da cui il nome). Ad alto livello offre tutte le funzionalità di Postgres (abbiamo utilizzato l'eseguibile Pgql per il bulk import nel nostro esperimento) ma vi aggiunge un particolare sistema di replicazione.

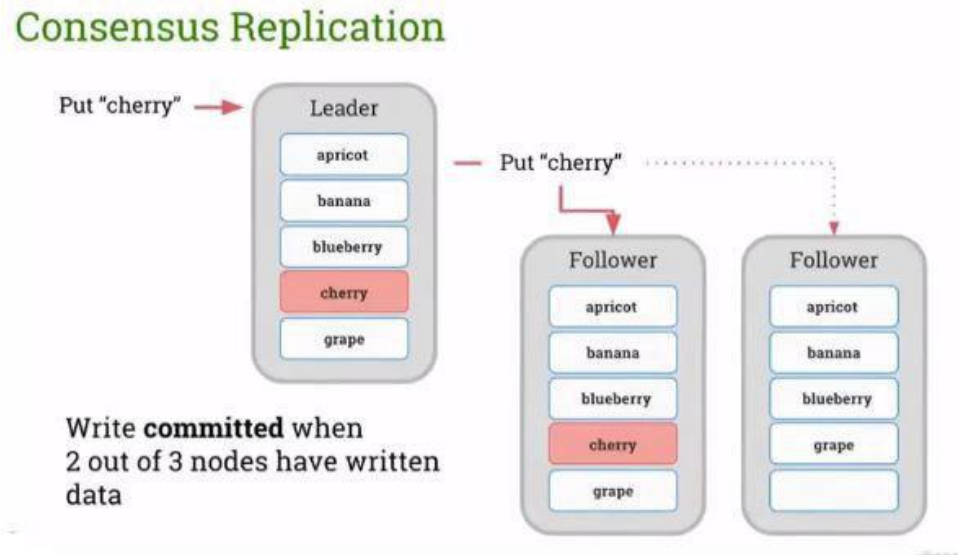


Figura 9 Cockroach Consensus Replication

Come Spanner, BigTable e HBase, Cockroach utilizza un sistema di distribuzione basato su ordinamento piuttosto che hashing. Le shard di Cockroach, chiamate “range” vengono divise quando raggiungono la dimensione massima di 64MB. Ogni nuovo dato inserito viene replicato sui vari nodi e la replicazione termina quando è ricevuto un numero di ack pari alla metà più uno dei nodi.

Quando un nodo ha un guasto si avvia il processo di riparazione, che ridistribuisce i dati perduti, e replicati, sui vari nodi in modo da mantenere costante il tasso di replicazione.

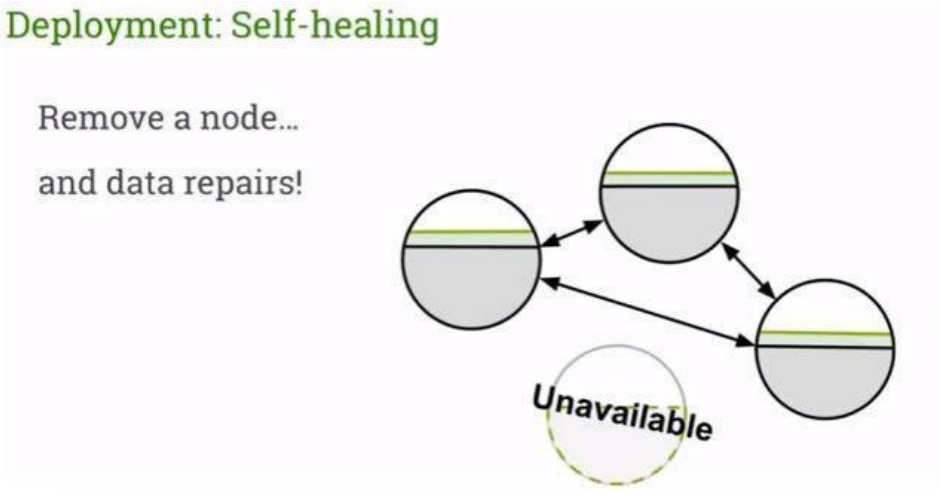


Figura 10 Cockroach Self-healing

Come mostrato in figura, Cockroach offre una Web UI per il monitoring del database e del cluster.

NuoDB

NuoDB è un database relazionale distribuito con replicazione totale. L’architettura del cluster è di tipo shared-nothing, con comunicazione peer-to-peer.

Ogni host contiene un transaction engine (TE) e/o uno storage manager (SM). I TE sono in-memory e si occupano di eseguire le query, vengono replicati su più host per aumentare il throughput. Replicare gli SM aumenta la durability. L’aviability richiede che siano presenti almeno un host con TE ed un host con SM.

Analisi delle performance

Il framework

Per eseguire i test descritti nell’introduzione abbiamo realizzato un framework di test in Python basato su Docker.

Innanzitutto abbiamo creato le immagini Docker (ove necessario) o abbiamo utilizzato le immagini ufficiali fornite dagli sviluppatori, ed abbiamo automatizzato il deployment attraverso il tool Docker-Compose stilando opportuni file di configurazione. Purtroppo il tool Docker Swarm, che avrebbe

reso semplice avviare cluster, non è ancora supportato pienamente dalla maggior parte delle immagini, che richiedono ancora configurazioni manuali.

Tramite Python abbiamo richiamato comandi di sistema per interfacciarci con Docker ed eseguire i container. Tramite Docker abbiamo comunicato con i container per effettuare le operazioni sui DBMS.

Il framework calcola in automatico i tempi di esecuzione, generando i relativi grafici riguardanti tempi e throughput.

Test e Performance

Di seguito riportiamo per ogni test svolto i risultati sotto forma di grafici e osservazioni.

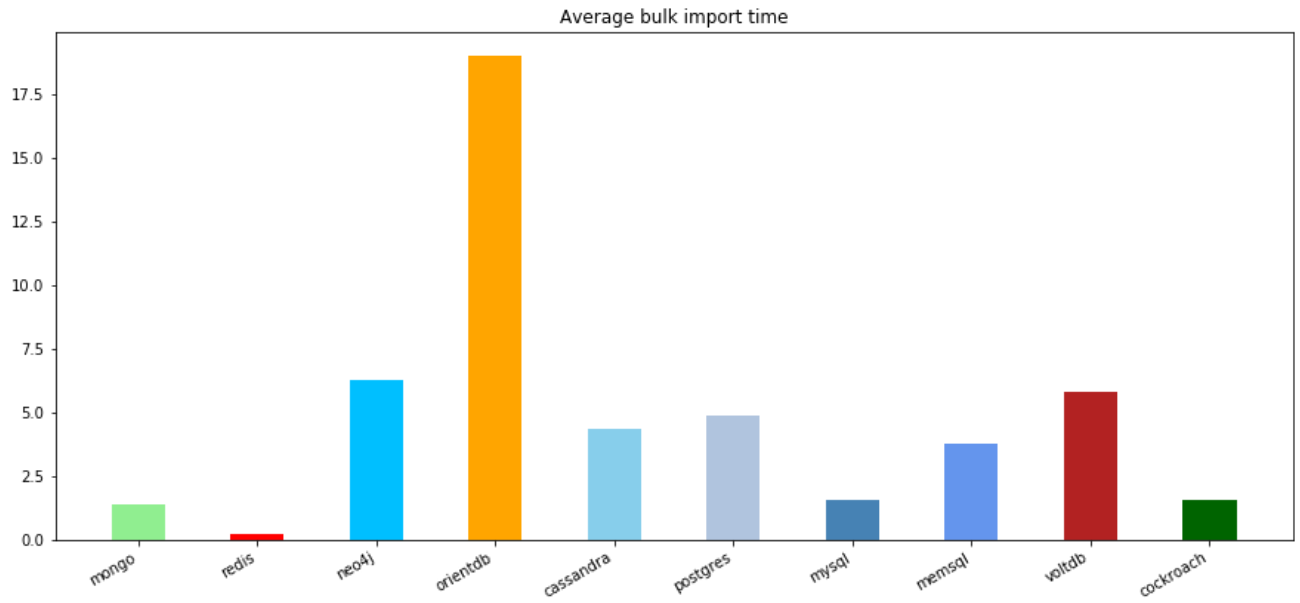
Per ulteriori informazioni sui test e i valori numerici rimandiamo al Python Notebook disponibile sulla repository pubblica di Github, indicata nella consegna sul portale Moodle.

Single Node

Inserimento

Per gli inserimenti abbiamo utilizzato i tool di bulk import disponibili sui diversi sistemi.

Per quanto riguarda i sistemi NoSQL, hanno tutti fornito un sistema di importazione da file csv, diretto o indiretto mediante uno script (Neo4j, Cassandra), oppure mediante ETL integrato (OrientDB).



Nei sistemi SQL e MemSQL abbiamo utilizzato Python con il modulo SQLAlchemy, che si interfaccia direttamente con i container tramite indirizzo IP. È importante notare come MemSQL offra una completa compatibilità con i sistemi SQL tradizionali.

Cockroach stesso, come accennato precedentemente, utilizza per l'import la SQL shell di Postgres (Psql), mentre VoltDB è l'unico NewSQL esaminato che offre una sua shell SQL (Sqlcmd) per il bulk import.

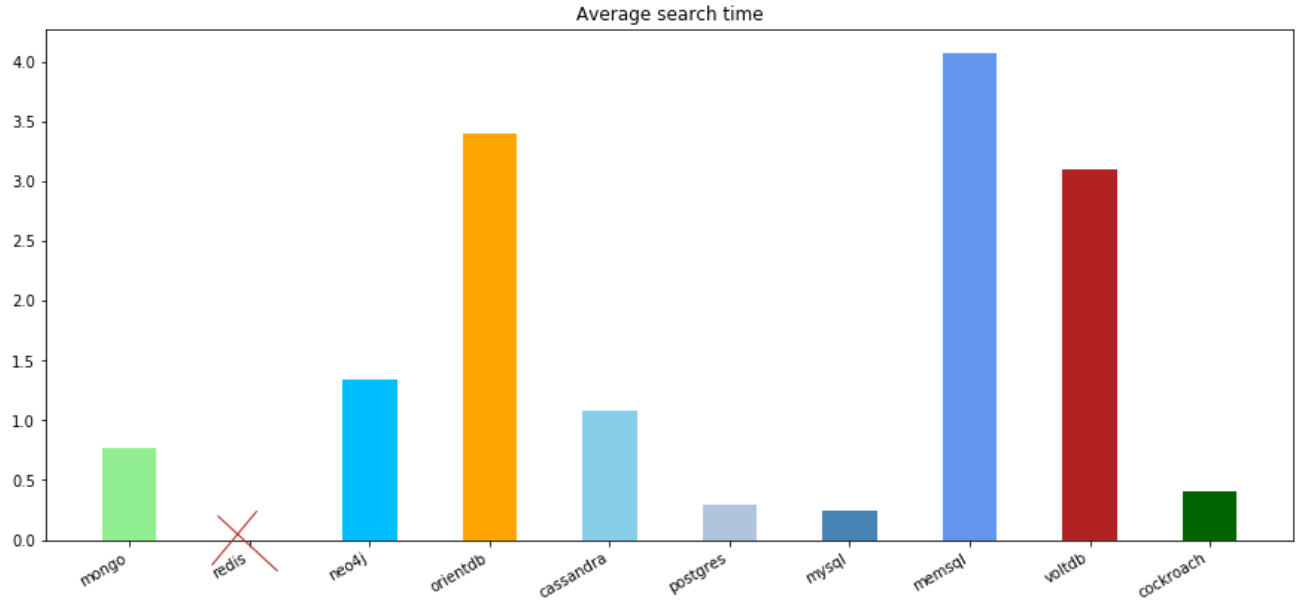
Come si evince dal grafico, i risultati sono quelli che ci aspettavamo, con Mongo e Redis i più performanti in assoluto, seguiti dai sistemi NewSQL. I meno performanti sono risultati essere i database a grafo, in particolare OrientDB (dato che risulta consistente con gli altri test). Questo è probabilmente dovuto alla generazione di una serie di metadati e a varie ottimizzazioni e conversioni sui dati che OrientDB effettua tramite il tool di ETL. Inoltre OrientDB genera eventi (utili ad esempio a inviare notifiche push) ogni volta che un dato è inserito.

Queste proporzioni sono risultate costanti anche al variare delle dimensioni del file di input fino a 500000 righe, confermando la minimalità di tempi di ovehead e la scalabilità lineare fino a dimensioni relativamente piccole.

Unindexed search

Il secondo test effettuato è la ricerca non indicizzata di una stringa all'interno del database.

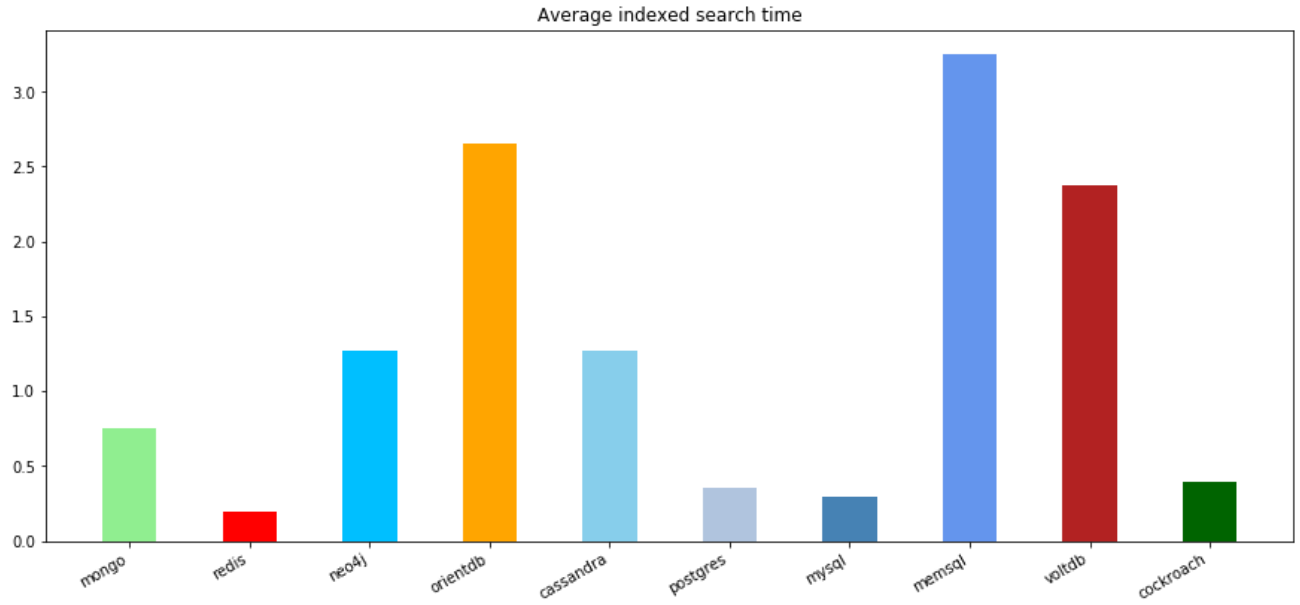
La maggior parte dei tempi è risultata consistente con i tempi di inserimento, eccetto per quanto riguarda VoltDB e MemSQL. La ricerca non indicizzata all'interno di questi DBMS è risultata estremamente inefficiente, anche ripetendo l'operazione 10 volte (ovviamente resettando la cache). Questo non dipende dal fatto che i due DBMS sono in-memory (dal momento che il dato è consistente con gli altri test in cui Redis è risultato il migliore in assoluto), ma forse è da attribuire a overhead di comunicazione in MemSQL, che funziona con più worker anche su nodo singolo (ipotesi dimostrata dai test sul throughput), e al fatto che VoltDB è pensato esclusivamente per cluster fino a 120 nodi (vedremo dai test in cluster che i risultati migliorano linearmente). Tuttavia a difendere l'onore dei NewSQL ci ha pensato Cockroach, che fra tutti gli esperimenti è sempre risultato il più efficiente e consistente fra essi, anche in single node, pur non essendo in-memory e senza avere la possibilità di sfruttare la cache (per via di come sono stati impostati gli esperimenti).



Va notato che per Redis non ci sono tempi di ricerca non indicizzata dal momento che non permette tale operazione (se non mettendo un indice sui valori, cosa che ovviamente vanificherebbe l'esperimento).

Indexed search

La ricerca indicizzata è stata svolta generando id casuali nelle varie iterazioni. I tempi ottenuti sono stati calcolati facendo le media. Come mostrato dalla figura, i risultati sono i medesimi (in percentuale) della ricerca non indicizzata.



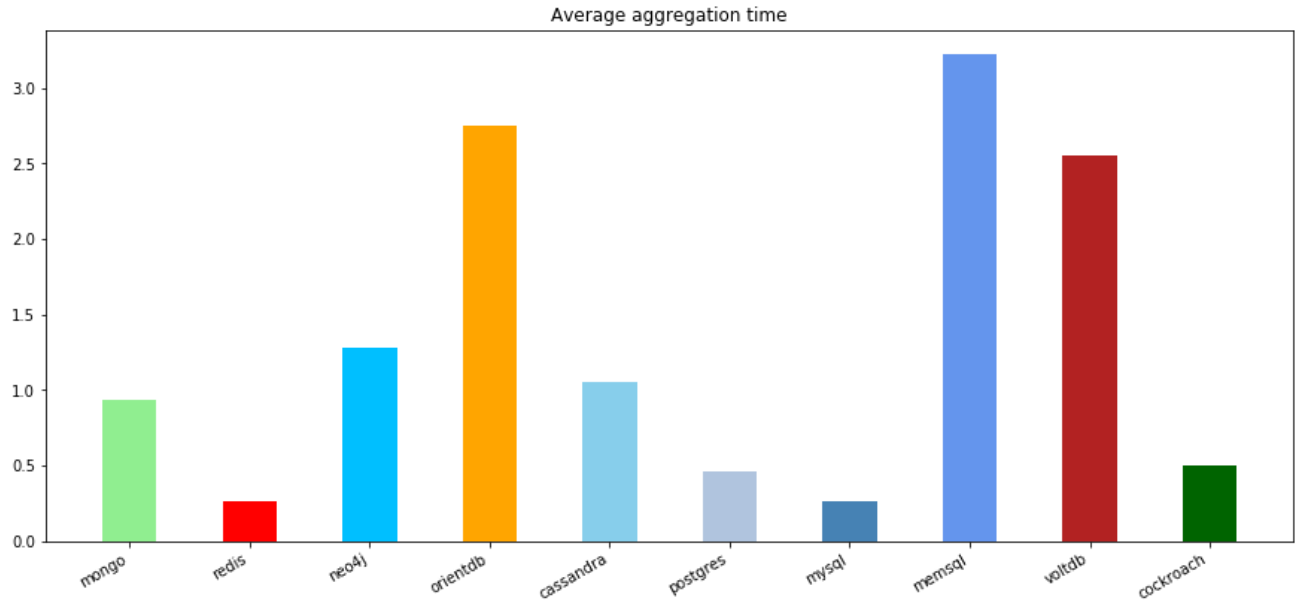
Redis si dimostra il più veloce nella sua operazione principale. Tutte le osservazioni possibili sono le medesime del paragrafo precedente. Va segnalato che per Mongo è stato aggiunto un secondo indice sul campo id numerico importato dal file csv, dal momento che normalmente utilizza un suo campo id autogenerato (che non avrebbe permesso di replicare l'esperimento).

Aggregation

Tutti i DBMS esaminati consentivano un certo tipo di funzioni di aggregazione. Per quanto riguarda Redis, abbiamo utilizzato una funzione di count su un set di elementi della stessa dimensione del dataset. Non potendo svolgere operazioni sui value, questa è stata l'unica operazione di aggregazione che potevamo utilizzare con il dataset autogenerato. Gli altri database hanno svolto un'operazione di average su un campo numerico del dataset.

I risultati confermano il ranking già visto nelle precedenti operazioni, con tempi di overhead non indifferenti per i NewSQL in-memory, che tuttavia non giustificati dal momento che Redis, anch'esso in-memory, ottiene i migliori risultati.

Vedremo con i test sul throughput che l'overhead è presente solo per MemSQL che, come detto, funziona in cluster dentro un unico container, mentre per VoltDB è da ricercarsi in altre cause, probabilmente legate al fatto che è pensato per funzionare in un cluster (vedremo dai test successivi).

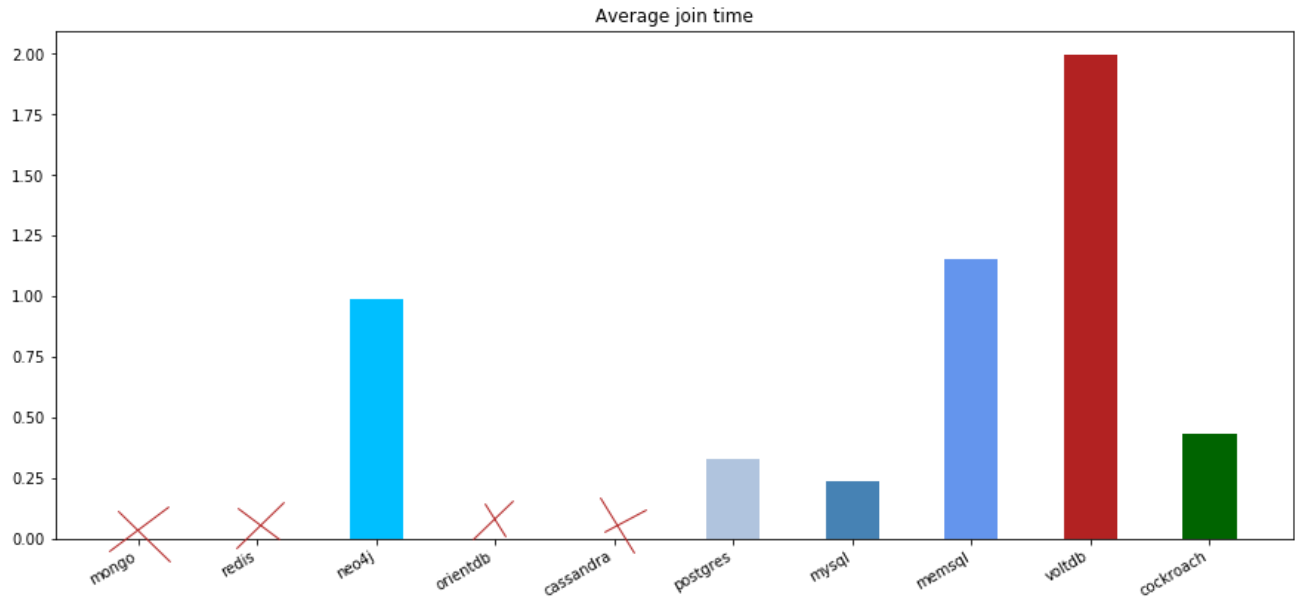


Join

L'operazione di join non è supportata da tutti i sistemi presi in esame, per ovvie ragione esplicitate nelle analisi dei vari DBMS.

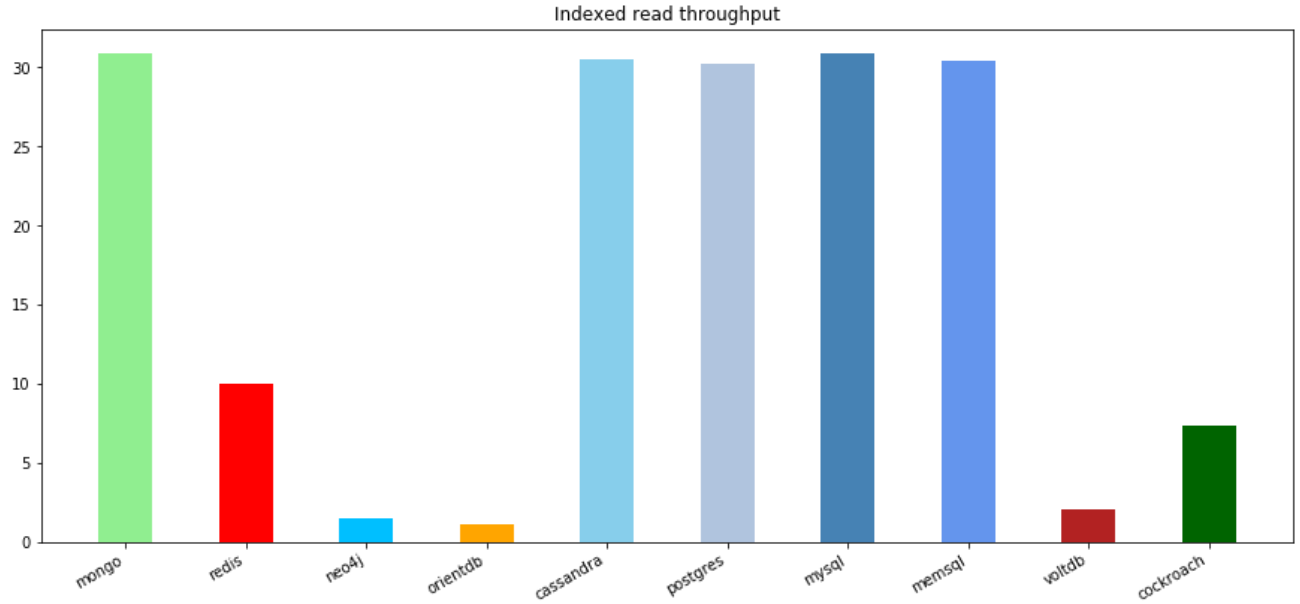
Nei sistemi NoSQL, solo i graph database permettono operazioni di join perché la struttura dati prevede la presenza di relazioni. Tuttavia OrientDB ha un particolare sistema di ottimizzazione per il quale il nostro dataset autogenerato, non contenendo vincoli di integrità referenziale, non ha permesso un esperimento senza l'aggiunta di ulteriori dati, che comunque lo avrebbero falsato rispetto agli altri.

In questo test VoltDB è risultato ancora più inefficiente, anche rispetto a MemSQL, dimostrando ancora una volta che il suo utilizzo non è pensato per il single node.

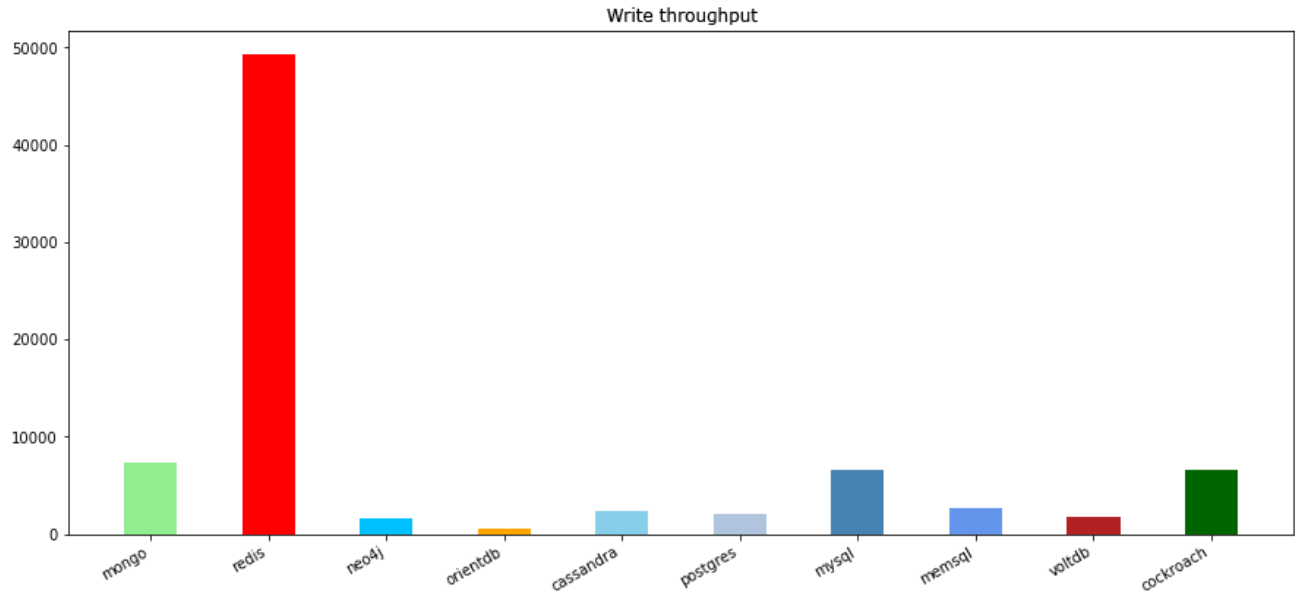


Read and write throughput

Il throughput in lettura è stato calcolato eseguendo indexed search in sequenza su id casuali, e mostra il numero di query al secondo che i DBMS sono in grado di processare.



Il dato è fortemente influenzato da un limite superiore dovuto dal limite del framework in Python e di Docker stesso, entrambi single thread. Tuttavia mostra comunque un dato interessante, ossia che quando abbiamo poca memoria disponibile Redis cala in prestazioni rispetto a Mongo, Cassandra e gli altri DBMS tradizionali. Va notato che in questo caso MemSQL mostra ottime prestazioni, rinforzando l'ipotesi per cui gli scarsi risultati precedenti fossero dovuti a overhead di comunicazione, mentre invece VoltDB si presenta ancora una volta poco efficiente in single node.

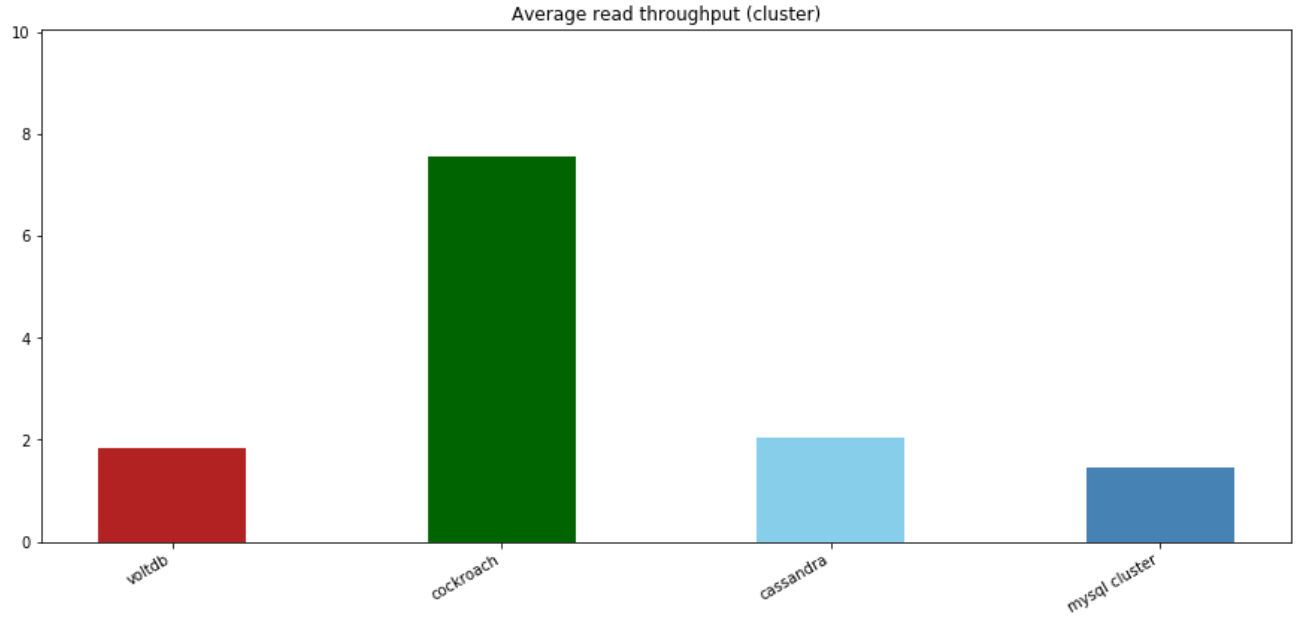


Per il throughput in scrittura, Redis risulta enormemente più efficiente, scrivendo su disco raramente e in batch. Mongo, MySQL e Cockroach sono al secondo posto con prestazioni simili tra loro, ed è singolare notare che ognuno rappresenta una diversa categoria di DBMS.

Cluster

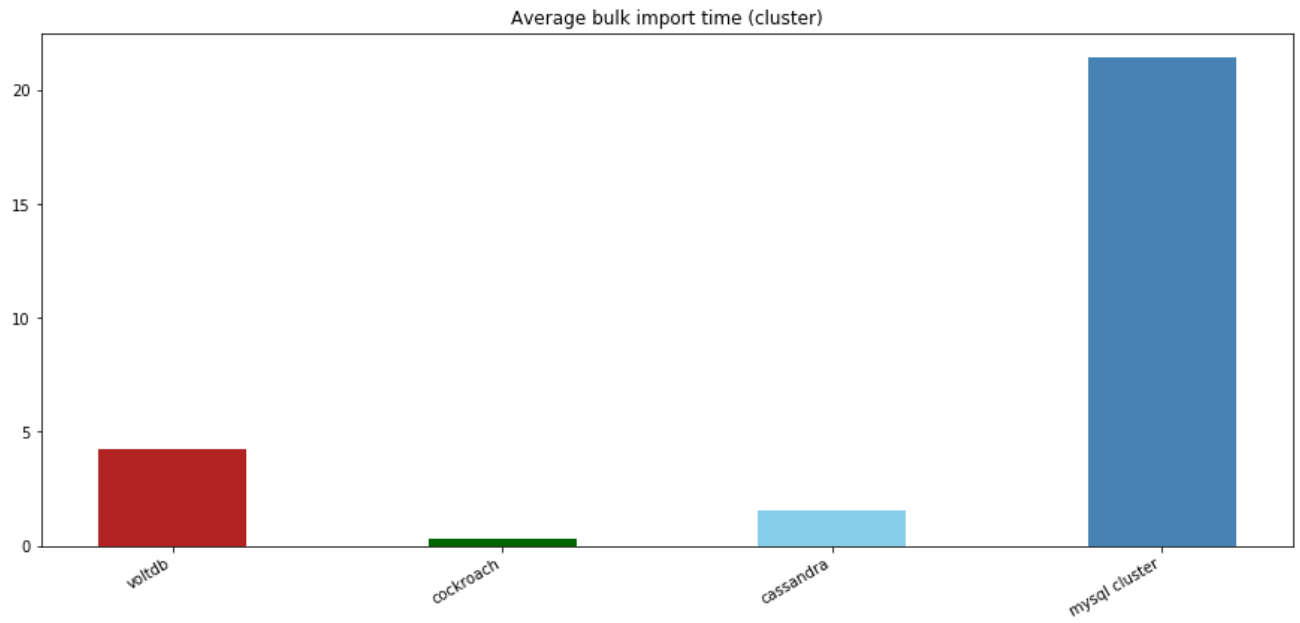
Sono stati realizzati i cluster per i DBMS per i quali lo ritenevamo più significativo. In particolare, confrontando VoltDB e Cockroach, rispettivamente il peggiore e il migliore tra i NewSQL nel single node, Cassandra che ha ottenuto prestazioni medie, e MySQL Cluster, database tradizionale in approccio NewSQL. Realizzare cluster funzionanti per ogni DBMS esaminato avrebbe portato via molto più tempo di quello a disposizione e sarebbe risultato comunque poco più significativo. Inoltre va sottolineato che i cluster realizzati sono molto più piccoli rispetto a quelli consigliati dai produttori (4 noti contro circa 120), e quindi abbiamo valutato una proiezione dei risultati.

Il throughput in lettura mostra come VoltDB, inizialmente diverse volte più lento di Cassandra e MySQL in single node, raggiunga stavolta un risultato quasi equivalente se non migliore. Cockroach, già fra i più efficienti nei test in singolo nodo, vede le sue performance ulteriormente migliorate con il cluster, mentre MySQL è in trend discendente rispetto alla sua implementazione standard.



Per quanto riguarda il tempo di scrittura (basso è meglio) notiamo anche qui un miglioramento netto di VoltDB, le cui performance scalano linearmente al crescere dei nodi, e una netta vittoria in generale dei sistemi NewSQL e NoSQL rispetto a MySQL Cluster, che per import di oltre 10000 righe risulta estremamente inefficiente rispetto agli altri sistemi.

Abbiamo potuto notare che per quanto riguarda i sistemi NoSQL e NewSQL, la tendenza è di scalare linearmente con il numero di nodi fino ad un certo threshold (come accennato 120 nodi per VoltDB), mentre cluster di DBMS SQL sembrano avere alcuni problemi per certe operazioni. Vanno inoltre considerate le funzionalità peculiari dei cluster NewSQL, come le funzioni di recovery messe in piedi da Cockroach, o le UI di gestione del cluster. Va inoltre considerato che per VoltDB è stata presa in esame la versione Community e non Enterprise.



Test di robustezza

Grazie a Docker abbiamo potuto facilmente realizzare test di robustezza e recovery per i cluster. In particolare Cockroach ha dimostrato una ripresa quasi istantanea a un tempo di risposta alle query invariato nonostante fossero effettuate a fronte della chiusura contemporanea di uno dei nodi. MySQL Cluster invece si è rivelato meno efficiente, andando a rilevare solo dopo alcuni secondi il fault e mostrando messaggi d'errore e tentativi di query multipli.

Conclusioni

In conclusione le performance dei DBMS hanno mostrato un ranking altalenante in base all'operazione, con l'unica quasi costante in Redis, che in generale è il più performante, ma ha funzionalità davvero limitate ai pochi casi d'uso per i quali è stato pensato. Per quanto concerne la nostra esperienza, i sistemi NewSQL hanno sicuramente mantenuto le promesse, dimostrando efficienza e retrocompatibilità in nodo singolo, ma soprattutto una effettiva facilità di configurazione del cluster e una scalabilità pressoché lineare delle performance. Per quanto riguarda i sistemi NoSQL in generale, essi offrono sicuramente funzionalità ridotte e meno attenzione alla consistenza, tuttavia garantiscono ancora le prestazioni più elevate, oltre al fatto che esistono tentativi di avvicinamento al paradigma SQL. La community dietro MongoDB ad esempio sta lavorando per portare proprietà ACID a livello di collezione (al momento sono solo a livello di documento), il che lo renderebbe molto simile, se non per l'assenza del join e la differenza del linguaggio di interrogazione, ad un sistema SQL, con tuttavia l'enorme vantaggio di essere schema-less e object oriented.

In conclusione, la scelta del DBMS deve essere il risultato delle funzionalità richieste e del tipo di astrazione del problema, ma per applicazioni che necessitano espressamente le funzioni di un DBMS relazionale o retrocompatibilità con un database SQL, e tuttavia richiedono scalabilità e robustezza, un sistema NewSQL è sicuramente auspicabile, almeno al momento, rispetto ad una implementazione distribuita di un sistema SQL tradizionale.