

7. DataBase Management System

RDBMS vs NRDBMS

I RDBMS (Relational Database Management Systems) e i NRDBMS (Non-Relational Database Management System) sono due tipi di sistemi che permettono la gestione dei database offrendo approcci diversi per l'organizzazione e la manipolazione dei dati.

I RDBMS seguono il modello relazionale, strutturando i dati in tabelle con relazioni definite tra di esse. Questi sistemi, come MySQL, Oracle e SQL Server, utilizzano il linguaggio SQL per interrogare e gestire i dati in modo efficiente.

I NRDBMS adottano invece modelli di dati flessibili come i document stores, key-value stores, graph databases e column-family stores.

- Document Stores: questi sistemi **gestiscono i dati mediante documenti**, solitamente in formato JSON o BSON, che possono essere archiviati e recuperati tramite un identificatore univoco, come un ID di documento.
- Key-value stores: come suggerisce il nome, **gestiscono i dati come coppie chiave-valore**. Possono essere utilizzati per archiviare e recuperare dati strutturati o non strutturati.
- Graph databases: sono **utilizzati per gestire dati che hanno relazioni complesse tra di loro**, come le reti sociali o le mappe. Questi sistemi sono progettati per eseguire operazioni di ricerca di cammini e algoritmi di analisi di grafi in modo efficiente.
- Column-family stores: sono progettati per **gestire grandi volumi di dati**, suddividendoli in **colonne e famiglie di colonne**, invece di fare uso di righe e colonne come in un RDBMS.

Questi sistemi non utilizzano solamente linguaggi SQL, ecco perché prendono il nome di NoSQL (Not Only SQL). Appartengono a questa categoria di DBMS: MongoDB, e Cassandra che, sono progettati per soddisfare requisiti specifici, come la scalabilità orizzontale, la gestione di dati complessi e la flessibilità. I database non relazionali consentono di lavorare con grandi volumi di dati distribuiti su più nodi e possono offrire alte prestazioni e elevate velocità di accesso ai dati.

NOTA

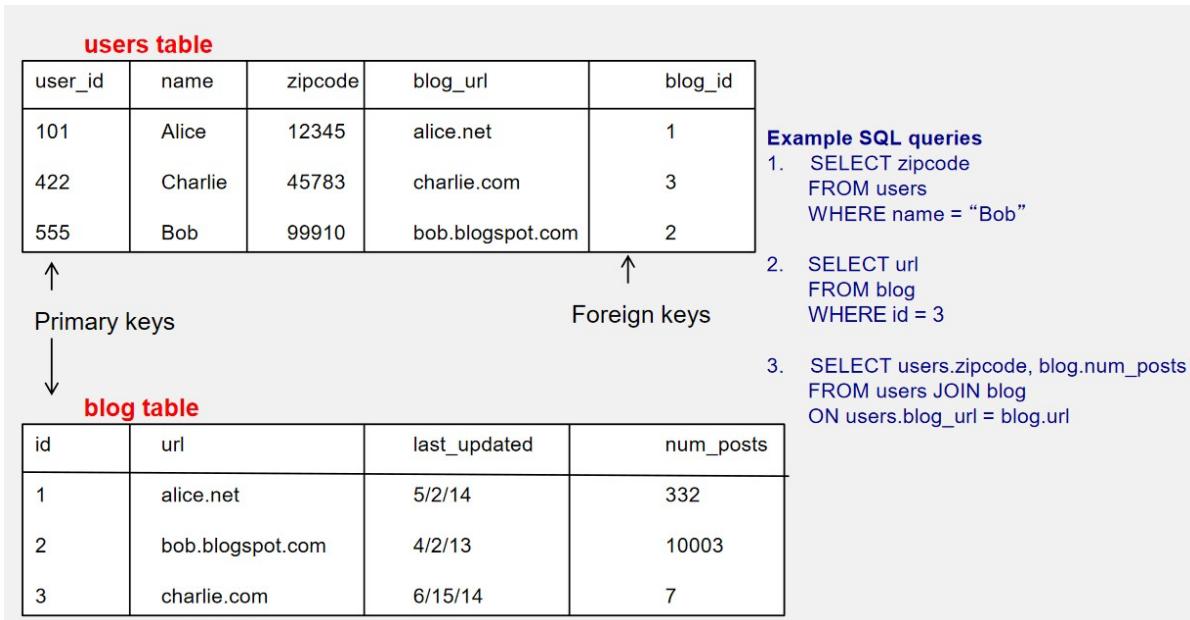
Il termine "NoSQL" è spesso usato in modo intercambiabile con NRDBMS, ma ci sono alcune differenze tra i due. I database NoSQL includono sia gli NRDBMS che altri sistemi che potrebbero utilizzare modelli di dati diversi o un approccio diverso alla gestione dei dati rispetto ai RDBMS tradizionali.

In sintesi, "NRDBMS" si riferisce specificamente ai sistemi di gestione dei database non relazionali, mentre "NoSQL" è un termine più ampio che abbraccia una varietà di sistemi di gestione dei database che possono o meno essere non relazionali, ma che condividono l'obiettivo di offrire alternative o integrazioni ai sistemi RDBMS tradizionali.

Corso di Tecnologie per il cloud - Memorizzazione chiave-valore RDBMS

Si definiscono **relazionali** i database SQL come MySQL, i quali sono stati i principali protagonisti della scena per anni. Proprio MySQL è uno dei più popolari, grazie alla sua struttura a tabelle, dove ogni **riga** corrisponde a un oggetto, che ha una **chiave primaria univoca**.

In un database SQL si eseguono le query utilizzando il linguaggio SQL (Structured Query Language) e si possono effettuare i cosiddetti **join**, cioè unire più tabelle con dei campi in comune.



I database relazionali, tuttavia hanno delle caratteristiche in contrasto con la realtà odierna:

- Oggi i dati sono di dimensioni notevoli e NON strutturati, motivo per il quale di un database strutturato se ne fa sempre a meno;
- Per usare database strutturati sono necessari notevoli quantità di letture e scritture, motivo per il quale i database SQL sono lenti;
- Ad oggi i join delle tabelle non sono così frequenti.

Un database moderno deve approcciarsi con le seguenti caratteristiche odierne:

- Deve essere **veloce**;
- Deve evitare i **single point of failure** (deve essere distribuito);
- Deve avere un basso costo nelle operazioni di lookup e inserimento;
- Deve autogestirsi;
- Deve essere scalabile.

Proprio per quanto concerne la scalabilità, si possono avere due approcci differenti:

- **Scale up:** la capacità del cluster viene incrementata rimpiazzando le attuali macchine con macchine più potenti. Si tratta di un approccio non conveniente dal punto di vista dei costi e che richiede una sostituzione periodica delle stesse;
- **Scale out:** la capacità del cluster viene incrementata aggiungendo più macchine economiche nel lungo periodo. Questa operazione consente di ammortizzare i costi ed è utilizzata oggi da diverse aziende.

Un database **NON RELAZIONALE** si differenzia da uno relazionale per le seguenti caratteristiche:

- Può essere **non strutturato**: nonostante possa avere delle tabelle, può essere non strutturato, non viene imposto uno schema e possono mancare righe e colonne;
- Non sempre supportano funzioni quali **join**;
- Possono avere indici di tabella come i database **relazionali**

NRDBMS - Key-value stores

I sistemi chiave-valore sono **un tipo** di database NoSQL (non relazionale) che utilizza un modello di dati semplice, basato su una struttura di dati chiave-valore. In questi sistemi, i dati vengono memorizzati in coppie di chiavi e valori, dove ogni chiave è unica e identifica univocamente il valore corrispondente.

Nella sua forma più semplice, un sistema chiave-valore è simile a un dizionario, dove è possibile recuperare il valore associato a una chiave specifica. Tuttavia, a differenza di un dizionario tradizionale, i sistemi chiave-valore sono generalmente progettati per gestire grandi quantità di dati e garantire prestazioni elevate.

Il funzionamento di un sistema chiave-valore è abbastanza diretto. Quando si desidera memorizzare un dato, viene specificata una chiave univoca e viene associato ad essa un valore. Successivamente, per accedere al valore, è sufficiente fornire la chiave corrispondente. Il sistema effettua una ricerca basata sulla chiave e restituisce il valore associato, se presente.

Le operazioni di base supportate dai sistemi chiave-valore (CRUD) includono:

1. Inserimento (Create): Consente di memorizzare una coppia chiave-valore nel database.
2. Recupero (Retrieve): Permette di recuperare il valore associato a una chiave specifica.
3. Aggiornamento (Update): Consente di modificare il valore associato a una chiave esistente.
4. Cancellazione (Delete): Permette di rimuovere una coppia chiave-valore dal database.



C.R.U.D

I sistemi chiave-valore sono spesso progettati per garantire alte prestazioni e scalabilità orizzontale. Poiché il modello dati è molto semplice, le operazioni di lettura e scrittura sono veloci e efficienti. Inoltre, i sistemi chiave-valore possono essere facilmente distribuiti su più nodi, consentendo di gestire grandi volumi di dati e di scalare orizzontalmente per soddisfare le esigenze di crescita.

Tuttavia, a causa della semplicità del modello dati, i sistemi chiave-valore possono risultare meno adatti per scenari che richiedono query complesse o strutture dati relazionali. Pertanto, è importante valutare attentamente le esigenze dell'applicazione e i requisiti di accesso ai dati prima di optare per un sistema chiave-valore.

Differenza tra RDBMS e NRDBMS di tipo key-value: A differenza dei database relazionali, i database di tipo chiave-valore non hanno una struttura specifica. I database relazionali memorizzano i dati in tabelle in cui **ogni colonna ha un tipo di dati assegnato**. I database di tipo chiave-valore sono una collezione di coppie chiave-valore che vengono memorizzate come record individuali e non hanno una struttura dati predefinita. La chiave può essere qualsiasi cosa, ma poiché rappresenta l'unico modo per recuperare il valore associato ad essa, la denominazione delle chiavi dovrebbe essere fatta in modo strategico.

NRDBMS - Column-family stores

I database Column-family sono un tipo di database NoSQL che **organizza i dati in gruppi di colonne anziché nelle tradizionali righe e colonne dei database relazionali**. Questo modello offre flessibilità nello schema dei dati, scalabilità orizzontale e prestazioni elevate, rendendoli adatti per la gestione di grandi volumi di dati non strutturati o semi-strutturati.

Le famiglie di colonne possono essere considerate come gruppi di colonne correlate tra loro. Ogni famiglia di colonne può contenere un numero arbitrario, ed è possibile aggiungerle o rimuoverle senza dover modificare lo schema dell'intero database. Questa flessibilità nel modello di dati rende i database Column-family adatti per applicazioni che richiedono una rapida evoluzione dello schema o una gestione dinamica dei dati.

Esempio

Comando: *Seleziona tutti i blog_id dalla tabella **blog** che sono stati aggiornati nell'ultimo mese.*

Si va a indagare sulla colonna *last_updated*, e facendo comparire la corrispondente colonna *blog_id*. Non è necessario andare a cercare nelle altre colonne.

Cassandra

Cassandra è un **NRDBMS** ibrido tra (**key-value** e **family-column**), progettato per gestire grandi volumi di dati distribuiti su cluster di server. È un database management system altamente scalabile, affidabile e ad alta disponibilità, che offre prestazioni elevate per applicazioni che richiedono una latenza ridotta e un'elevata tolleranza ai guasti.

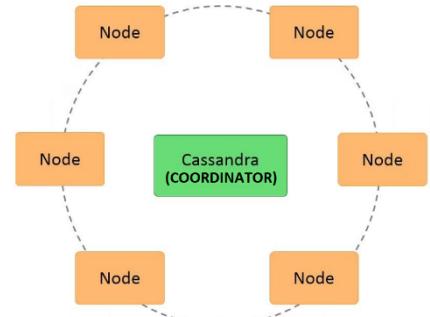
Cassandra è stato sviluppato per affrontare i problemi di scalabilità e affidabilità dei database tradizionali. Utilizza un modello di dati distribuito e senza point of failure, che consente di archiviare grandi quantità di dati su numerosi nodi all'interno di un cluster. Questo modello di distribuzione garantisce che i dati siano replicati su più nodi, garantendo così l'affidabilità dei dati anche in caso di guasti hardware o interruzioni di rete.

Server mapping - Inserimento nodi

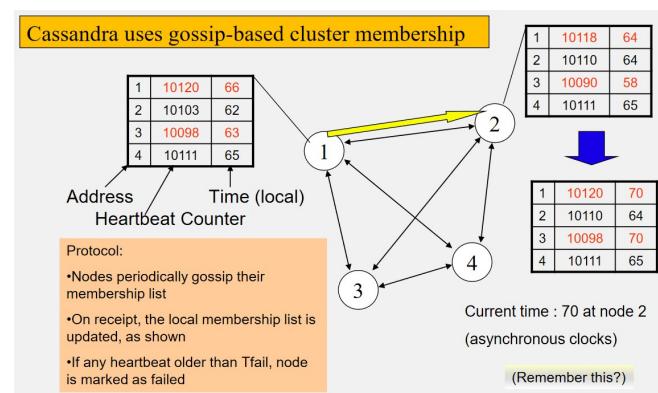
Per determinare quale server debba contenere un'associazione chiave-valore nel database management system Cassandra, viene utilizzato un approccio basato su un "ring" virtuale di 2^m nodi. Questo ring consiste in un insieme di server, chiamati "endpoint", che vengono disposti **"logicamente"** in modo circolare. Le chiavi vengono memorizzate nei server del ring in base al loro ID, in modo simile a come avviene nell'algoritmo Chord. **ATTENZIONE RICORDIAMO CHE CASSANDRA E' UN NRDBMS, MENTRE CHORD E' UN PROTOCOLLO P2P !!!**

Collocazione file

Ogni data center contiene il proprio ring, e all'interno di esso vengono collocate le **associazioni chiave-valore**. Tuttavia, a differenza dell'algoritmo Chord, il ring virtuale di Cassandra non utilizza una "finger table"; Bensì, è presente un coordinatore che gestisce le richieste provenienti dai client e sa dove trovare le chiavi e gli eventuali duplicati all'interno del ring.



Il **coordinatore** utilizza un componente chiamato "**Partitioner**" per assegnare una chiave al server appropriato; questo processo consente al coordinatore di determinare quale server debba gestire le operazioni di lettura e scrittura per una particolare chiave nel sistema distribuito di Cassandra.
 NOTA: Il coordinatore in Cassandra è un singolo nodo nel cluster che si occupa della gestione delle richieste dei client e del coordinamento delle operazioni nel sistema distribuito. **Anche se il coordinatore rappresenta un punto di potenziale fallimento, è possibile configurare nodi di backup che agiscono come coordinatori di riserva nel caso in cui il coordinatore principale fallisca; a tale scopo, ciascun nodo mantiene una lista di tutti gli altri nodi che fanno parte del cluster.** Questa lista deve essere aggiornata automaticamente ogni volta che un nodo esegue il **join**, **leave** e/o **fail**. Tuttavia, **in ogni momento, solo un coordinatore sarà attivo** e gestirà le richieste dei client. L'obiettivo è mitigare i rischi associati a un singolo punto di fallimento.



Strategie di memorizzazione e repliche

Ci sono due differenti strategie per memorizzare i dati all'interno di un ring Cassandra:

- **SimpleStrategy:** fa uso del Partitioner per memorizzare le informazioni nel data center, e può essere di due tipi:
 - **RandomPartitioner:** Utilizza un sistema simile a Chord per assegnare i dati alle diverse partizioni all'interno del ring. In questa strategia, **ogni partizione ha un determinato fattore di replica che indica il numero di copie che devono essere mantenute. Le copie vengono distribuite in modo uniforme tra i nodi nel cluster.** Questo metodo di assegnazione casuale può essere utile in alcuni scenari.
 - **ByteOrderedPartitioner:** Assegna un intervallo di chiavi ai server. Ad esempio, se abbiamo un sistema come Twitter, potrebbe essere utile per recuperare tutti gli utenti il cui nome inizia con 'a' e termina con 'b'. In questo modo, i dati con chiavi adiacenti saranno memorizzati vicini tra loro nel ring.
- **NetworkTopologyStrategy:** Questa strategia viene tipicamente utilizzata in **sistemi che operano su più data center**. Consente di definire il numero di repliche dei dati per ogni data center.
Ad esempio:
 - Due repliche per data center: In questo caso, ogni data center avrà due copie dei dati.
 - Tre repliche per data center: Ogni data center avrà tre copie dei dati.

Per un singolo data center, le repliche vengono posizionate nel ring in base al Partitioner utilizzato. La prima replica viene posizionata utilizzando il Partitioner, mentre le repliche successive vengono posizionate in senso orario all'interno del ring fino a quando non si raggiunge un rack (un insieme di server) diverso. Questo garantisce la distribuzione dei dati su server diversi all'interno del data center.

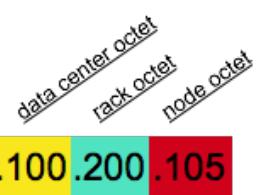
Snitches – localizzazione nodi

In Apache Cassandra, il termine "snitch" si riferisce a un componente che informa il cluster sulla topologia della rete, consentendo ai nodi di comunicare tra loro in modo efficiente. **Il ruolo principale dello snitch è quello di determinare la posizione di un nodo all'interno del cluster**, consentendo a Cassandra di bilanciare il carico e mantenere la coerenza dei dati.

Esistono diverse modalità di funzionamento degli Snitches:

- **SimpleSnitch:** È la modalità predefinita usata da Cassandra; **i nodi sono organizzati in un unico data center senza considerare la loro posizione fisica o la topologia di rete.** Questo snitch è adatto per configurazioni in cui non è richiesta una suddivisione geografica o un bilanciamento del carico tra data center separati.
- **RackInferring Snitch:** Questo snitch è utile quando la topologia di rete del data center segue uno schema prevedibile nella distribuzione dei nodi all'interno dei rack (gruppi fisici di nodi all'interno di un data center). Quando si utilizza il RackInferring Snitch, Cassandra esegue una mappatura degli indirizzi IP dei nodi per stabilire la posizione dei nodi all'interno dei rack. Poiché i nodi all'interno dello stesso rack condividono un prefisso di indirizzo IP comune, il RackInferring Snitch analizza gli indirizzi IP dei nodi e assegna loro una posizione all'interno dei rack in base a tale prefisso.

L'obiettivo del RackInferring Snitch è semplificare la configurazione del cluster, **eliminando la**



110.100.200.105

CORSO DI TECNOLOGIE PER IL CLOUD - Memorizzazione chiave-valore

necessità di specificare manualmente la posizione dei nodi all'interno dei rack. Tuttavia, è importante notare che questo snitch richiede una distribuzione coerente degli indirizzi IP dei nodi all'interno del data center per inferire correttamente la posizione dei nodi.

Scritture

Il client Cassandra, quando deve scrivere, parla col coordinatore che fa parte di quel cluster. I coordinatori possono essere selezionati in base a tre criteri: per chiave, per client o per query. Il coordinatore per chiave viene scelto in base alla chiave primaria specificata nella richiesta, il coordinatore per client viene scelto in base all'origine della richiesta del nodo client, e il coordinatore per query viene scelto in base al tipo di query inviata. **Il coordinatore fa uso del componente “Partitioner” per inviare una query a tutti i nodi replica responsabili di quella chiave.** Quando tutti i nodi replica rispondono, il coordinatore ritorna un ACK al client. Cassandra adotta un approccio specifico per garantire la sua funzionalità e affidabilità in scrittura. Questo approccio comprende due caratteristiche chiave:

Mechanismo di Hinted Handoff: è una caratteristica fondamentale di Cassandra, che garantisce la scrittura costante dei dati. **Quando un nodo di replica smette di funzionare, il coordinatore del sistema scrive i dati sugli altri nodi replica disponibili e mantiene in attesa la scrittura per il nodo guasto,** finché non torna online. In questo modo, anche se alcune repliche sono temporaneamente offline, i dati vengono comunque scritti e mantenuti in attesa fino a quando la replica guasta non ritorna operativa. **Nel caso in cui tutte le repliche siano offline contemporaneamente, il coordinatore memorizza temporaneamente le scritture in un buffer** per un determinato periodo di tempo, solitamente un paio d'ore.

Ring: l'utilizzo di un anello per datacenter e l'elezione di un coordinatore per ciascun datacenter (del quale abbiamo già parlato in precedenza) permettono una gestione efficiente delle scritture. Ogni coordinatore di datacenter si occupa di coordinare le attività all'interno del proprio datacenter, inclusa la scrittura dei dati sui nodi replica. I coordinatori dei diversi datacenter collaborano tra loro per garantire che i dati vengano correttamente replicati e distribuiti in tutto il sistema.

Scritture in un nodo replica

Nel momento in cui un nodo replica riceve una scrittura:

1. Inserisce tale evento, nel **commit log**, in modo tale da poter recuperare un eventuale fallimento;
2. Effettua delle modifiche nelle **memtable** appropriate. **La memtable è una cache che permette la rappresentazione in memoria delle associazioni chiave-valore;**
3. **Quando la memtable è piena, il contenuto viene scritto su disco, in strutture chiamate SSTable** (Sorted String Table), in particolare la memtable si divide in due SSTable:
 - a. SSTable dati contenente tutte le associazioni chiave-valore;
 - b. SSTable indici contenente le chiavi e la posizione nella SSTable dei dati;

Le SSTable sono file di dati immutabili composti da un indice e i dati effettivi. L'indice tiene traccia della posizione delle righe di dati all'interno del file. I dati sono organizzati in base alla chiave, consentendo un accesso rapido. Le SSTable sono ottimizzate per la lettura e la scrittura efficienti. Le modifiche ai dati vengono effettuate creando nuove SSTable e le SSTable vengono compattate periodicamente per eliminare le righe obsolete. In sintesi, le SSTable sono fondamentali per le prestazioni elevate e la scalabilità di Cassandra. Per rendere le operazioni più efficienti, si fa uso del **Bloom filter**.

Lettura dei dati

La lettura dei dati è simile a quanto visto in scrittura, eccezion fatta per il discorso che:

- **Il coordinatore può contattare X nodi replica** nel medesimo rack e inviare la richiesta di lettura a quei nodi che hanno risposto più velocemente;
- Quando tali nodi rispondono, **il coordinatore ritorna il dato più recente tra queste repliche**;

Ad ogni modo, il coordinatore fa riferimento anche ai valori ottenuti da altri nodi replica e **verifica l'eventuale consistenza**. Se i dati non sono consistenti (nel senso che non sono uguali in tutti i nodi), avvia in background un processo di riparazione.

Inoltre, dato che una riga può essere suddivisa in più SSTable, la fase di lettura implica che tali tabelle debbano essere unite, e ciò comporta una lettura più lenta della scrittura, anche se comunque veloce.

Bloom filter - ricerca dei file

Il **Bloom filter** è impiegato per ottimizzare le query in Cassandra. Si tratta di una struttura dati probabilistica che **consente di effettuare velocemente una verifica sulla presenza di un elemento specifico in un insieme di elementi, riducendo così la necessità di ricerche costose**. Nell'ambito di Cassandra, i Bloom filter vengono utilizzati per evitare la ricerca su disco di dati inesistenti, migliorando notevolmente le prestazioni delle query e alleggerendo il carico sui nodi del cluster. Quando un client invia una query a Cassandra, il Bloom filter viene utilizzato per determinare se i dati richiesti esistono o meno, consentendo a Cassandra di rispondere tempestivamente alla query senza dover effettuare la ricerca su disco. Pertanto, i Bloom filter svolgono un ruolo fondamentale nell'ottimizzazione dell'efficienza delle query in Cassandra.

Il bloom filter consiste sostanzialmente in un vettore chiamato bitmap; la dimensione massima di una bitmap del Bloom filter dipende da diversi fattori, tra cui la quantità di dati da gestire, la probabilità accettabile di falsi positivi e le risorse disponibili nel sistema.

In generale, la dimensione di una bitmap del Bloom filter viene definita come il numero totale di bit nel vettore; per calcolare la dimensione massima del Bloom filter, si può utilizzare la seguente formula approssimativa:

$$m = \frac{-n * \log(p)}{\log(2)^2}$$

Dove:

- **m** è la dimensione del Bloom filter in bit.
- **n** è il numero di elementi da inserire nel Bloom filter.
- **p** è la probabilità accettabile di falsi positivi.

Ad esempio, se si desidera inserire 1.000 elementi nel Bloom filter con una probabilità accettabile di falsi positivi del 1%, la dimensione approssimativa del Bloom filter sarà:

$$m = \frac{-1000 * \log(0.01)}{\log(2)^2} \approx 9586 \text{ bit}$$

Quindi, in questo caso, la dimensione massima del Bloom filter sarebbe di circa 9586 bit. Tuttavia, è

Corso di **Tecnologie per il cloud - Memorizzazione chiave-valore**

importante considerare che i valori calcolati sono approssimativi e che potrebbero essere necessarie ottimizzazioni o regolazioni in base alle specifiche esigenze del sistema.

Determinare se una chiave è presente nel Bloom filter

A questo punto, come si fa a determinare se una chiave sia presente nel **Bloom filter** o meno?

1. Si prende la chiave e se ne calcola di nuovo gli hash;
2. Si confrontano gli slot restituiti dagli hash;
3. Se uno di questi è zero, allora significa che la chiave NON è presente;
4. Se tutti questi sono pari a 1, allora significa che la chiave è presente.

Questo tuttavia non è una garanzia, poiché può capitare che metà dei bit di tale chiave siano impostati a true da una chiave e l'altra metà da una chiave ancora. Facendo poi il confronto, si avrebbe tutto impostato a true e quindi risulterebbe un falso positivo. Tuttavia c'è una probabilità molto bassa che questo possa accadere.

APPRAFONDIMENTO Bloom Filter

NASCE PER STABILIRE SE UN ELEMENTO APPARTIENE
O NO AD UN DETERMINATO SET DI DATI

SI STABILISCE:

1) DIMENSIONE $m = 5$

2) FUNZIONI DI HASHING

$$h_1(x) = x \text{ modulo } m$$

$$h_2(x) = (2x+3) \text{ modulo } m$$

→ VAL DA AGGIUNGERE

x	h_1	h_2	Bloom										
9	4	1	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	0	1	0	0	1	0	1	2	3	4
0	1	0	0	1									
0	1	2	3	4									

x	h_1	h_2	Bloom											
11	1	0	<table border="1"><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	1	0	0	1	0	1	2	3	4	→ RIHALE 1 DALLO STATO PRECEDENTE
1	1	0	0	1										
0	1	2	3	4										

SE VOGLIATE SAPERE: IL 15 È NEL SET? $15 \ h_1=0 \ h_2=3$

DATO LO STATO 11001 VISTO PRIMA:

- SE IN POSIZ. 0 e 3 ABBIAHOO DUE 1 PROBABILMENTE
È GIA' PRESENTE NEL SET
- SE HO 1 e 0 OCCORRENNE VAL ≠ IL 15 NON È PRESENTE

PROVATO CON 16 $h_1=1 \ h_2=0$ TROVIAMO DUE 1 MA
16 NON LO ABBIAHOO MAI HESSO!!! E' UN FALSO POSITIVO

Compattazione

I dati nel tempo si accumulano, dunque è opportuno compattare i log e le SSTables. A tal proposito:

- Il processo di compattazione comporta il merging delle tabelle;
- Il processo viene eseguito periodicamente e localmente su ciascun nodo;

Eliminazione di un elemento

Quando si intende eliminare un'associazione chiave-valore dai server, non si va direttamente ad eliminare l'elemento, ma si aggiunge quella che è chiamata **tombstone**, ossia un marcatore che marca tale elemento come "da eliminare" alla prossima compattazione delle SSTable.

CAP Theorem

CAP è acronimo di (**Consistency**, **Availability** e **Partition-tolerance**):

- **Consistency**: tutti i nodi hanno gli stessi dati;
- **Availability**: il sistema è operabile sempre e velocemente;
- **Partition-tolerance**: il sistema non è afflitto dalle variazioni della topologia di rete.

Perchè la disponibilità è importante?

La presenza di un'elevata disponibilità consente di avere letture e scritture affidabili e veloci.

Perchè la consistenza è importante?

Tutti i nodi, in tal modo, vedono gli stessi dati in qualunque momento, oppure le letture ritornano gli ultimi valori scritti da ciascun client. Questo è importante soprattutto in sistemi bancari o in sistemi di volo (si pensi alla prenotazione di un posto).

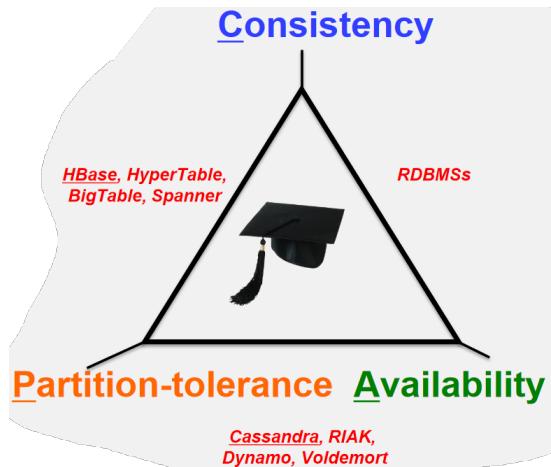
Perchè la partition-tollerance è importante?

E' la capacità di un sistema distribuito di continuare a funzionare correttamente nonostante la presenza di guasti di rete o frammentazioni della stessa che isolano alcuni nodi del sistema.

Problematiche del CAP theorem

Il **CAP Theorem**, proposto da Eric Brewer afferma che: in un **sistema distribuito** è possibile soddisfare e garantire **al massimo** due di queste tre condizioni. Se si decide di soddisfare Consistenza e Disponibilità, non si può avere un'elevata tolleranza al partizionamento. Per esempio:

- **Cassandra** offre una **consistenza eventuale**, per favorire l' **availability** e la **partition-tolerance**;
- I **databases relazionali** favoriscono la **consistency** e l' **availability**;



Consistenza eventuale

Cassandra si basa sulla filosofia **BASE** (Basically Available Soft-state Eventual Consistency), che preferisce l'availability alla consistency.

Quindi Cassandra si fonda sulla premessa che i sistemi distribuiti non possono garantire sempre una **coerenza immediata** dei dati a causa dei **ritardi** nella comunicazione asincrona tra i nodi.

Secondo questo modello, i dati vengono replicati in modo asincrono su più nodi e potrebbero esistere momentaneamente versioni leggermente divergenti dei dati. Durante le letture, il coordinatore restituisce la versione più recente disponibile dei dati, ma non è garantito che tutti i nodi abbiano la stessa versione dei dati in ogni momento. **Nel tempo, attraverso il processo di scambio di informazioni tra i nodi, le copie dei dati convergono a una versione coerente, garantendo la coerenza eventualmente consistente.**

Mentre i **database relazionali** si basano sulla filosofia **ACID** ovvero un acronimo che sta per **A**tomicità, **C**onsistenza, **I**solamento e **D**urabilità:

- **Atomicità:** ogni transazione nel database deve essere considerata come una singola unità di lavoro che viene eseguita completamente o non viene eseguita affatto. Ciò significa che se una transazione non può essere eseguita completamente, tutte le modifiche apportate fino a quel momento devono essere annullate e il database deve essere ripristinato allo stato precedente.
- **Consistenza:** dopo il completamento di una transazione, il database deve essere in uno stato coerente e valido. Ciò significa che tutte le regole di integrità dei dati, come i vincoli di chiave esterna, devono essere rispettati e tutte le modifiche apportate alla base di dati devono essere conformi alle regole e ai vincoli di validità.
- **Isolamento:** ogni transazione deve essere eseguita in modo completamente isolato dalle altre transazioni in esecuzione contemporaneamente. Ciò significa che le modifiche apportate da una transazione non devono essere visibili ad altre transazioni fino al completamento della transazione in corso.
- **Durabilità:** dopo il completamento di una transazione, tutte le modifiche apportate devono essere permanenti e resistere a eventuali guasti del sistema. Ciò significa che le modifiche apportate devono essere scritte su un supporto di memorizzazione permanente, come un disco rigido o una memoria flash, per garantire che sopravvivano a eventuali interruzioni di corrente o guasti hardware.



In sintesi, la filosofia **ACID** dei database relazionali mira a garantire che i dati vengano gestiti in modo affidabile e consistente, senza la possibilità di perdere dati o incorrere in situazioni di incoerenza o ambiguità. Ciò è particolarmente importante per le applicazioni aziendali che richiedono un'alta affidabilità e precisione dei dati.

Livelli di consistenza di Cassandra

Cassandra permette di impostare dei livelli di consistenza **sui dati replicati**, i quali si applicano contemporaneamente alle letture e le scritture. Al client è consentito scegliere il livello di consistenza tra:

- **ANY**: si basa in particolare sul fattore di replica (quanti nodi devono replicare un dato contenuto). Se il numero di nodi che risponde è minore del fattore di replica, sarà possibile solamente scrivere sul database, ma non leggere;
- **ALL**: alle richieste devono rispondere tutti i server. Se anche uno solo dei nodi presenti non è più raggiungibile, il database non sarà più utilizzabile, né in lettura, né in scrittura;
- **ONE**: almeno una replica deve rispondere alle richieste, sia in lettura, che in scrittura;
- **QUORUM**: almeno la maggioranza deve rispondere alle richieste, sia in lettura, che in scrittura.

Funzionamento del quorum

Il quorum in Cassandra è un meccanismo utilizzato per garantire la coerenza dei dati in un ambiente distribuito. Si basa sul fattore di replica, che determina il numero di copie dei dati mantenute nel cluster. Il quorum viene utilizzato per le operazioni di lettura e scrittura e non è strettamente legato al 50% dei nodi. La formula per calcolare il quorum in Cassandra è la seguente:

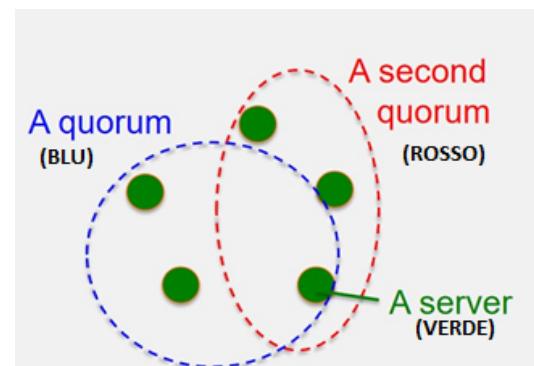
$$\text{quorum} = \frac{\text{somma dei fattori replica dei DC}}{2} + 1$$

Dove "somma dei fattori di replica" rappresenta il numero di copie dei dati da creare nel cluster.

Ad esempio, se il fattore di replica è impostato su 3, il quorum sarà calcolato come $(3 / 2) + 1 = 2$. Quindi, per garantire la coerenza dei dati, sarà necessario ottenere la risposta da almeno 2 nodi.

Supponiamo di avere un gruppo di 5 nodi replica e una coppia chiave-valore. Per garantire la replica di tale coppia, è necessario che almeno 3 dei 5 nodi abbiano una copia dei dati. Abbiamo quindi due quorum, in cui un server è presente in entrambi.

Ora immaginiamo che il Client 1 effettui una scrittura nei server del quorum rosso: i nodi replica all'interno del quorum rosso vengono aggiornati e viene restituito un ACK (acknowledgement) al Client 1.



Successivamente, il Client 2 effettua una lettura nel quorum blu. Poiché un quorum rappresenta almeno il 50% dei nodi, ogni coppia di quorum avrà almeno un server in comune. Questo server restituirà sempre il valore più aggiornato, poiché viene richiamato ogni volta. Nel caso specifico, il server appartiene al quorum rosso e restituirà il valore più recente. Il livello di consistenza QUORUM è più veloce di ALL, nonostante mantenga un alto livello di consistenza.

Il quorum in Cassandra

In **Cassandra**, il quorum è impostabile nel dettaglio come:

- **EACH_QUORUM**: il quorum deve essere raggiunto in **ciascun** data center;
- **LOCAL_QUORUM**: il quorum deve essere raggiunto da almeno il data center dove risiede la chiave;
- **QUORUM**: il quorum può essere raggiunto a livello globale.

Lettura

1. Il client imposta un livello di consistenza in lettura, indicato come R, che è inferiore o uguale al numero di nodi replica, indicato come N, associati a una specifica chiave.
2. Il coordinatore invia una richiesta ai R nodi per ottenere il dato e, se riceve risposte da tutti i nodi, invia un ACK (acknowledgement) al client confermando la lettura avvenuta con successo.
3. Nel frattempo, il coordinatore verifica gli altri N - R nodi replica associati alla stessa chiave in background. Se necessario, avvia delle operazioni di riparazione per sincronizzare e ripristinare eventuali discrepanze tra i dati presenti su diversi nodi replica.

Scrittura

1. Il client imposta un livello di consistenza in scrittura, indicato come W (write), e richiede al coordinatore di scrivere il dato su un numero specifico di nodi replica.
2. Il coordinatore invoca i W nodi replica designati per la specifica chiave. Se tutti i nodi rispondono, il coordinatore procede con l'operazione di scrittura e restituisce un ACK (acknowledgement) al client confermando il successo della scrittura.

Tuttavia, se alcuni nodi replica non rispondono, il comportamento dipende dall'impostazione configurata:

- a. Se è configurato per continuare la scrittura nonostante la mancata risposta di alcuni nodi, il coordinatore procede comunque con la scrittura in memoria e restituisce un ACK al client. In questo caso, potrebbe esserci una potenziale inconsistenza dei dati tra i nodi replica che non hanno risposto e quelli che hanno ricevuto l'operazione di scrittura.
- b. Se è configurato per bloccare le scritture fino a quando non si raggiunge un quorum (un numero minimo di risposte coerenti), il coordinatore sospende le scritture finché non ha ricevuto risposte da un numero sufficiente di nodi replica per formare il quorum. Solo allora procede con l'operazione di scrittura e restituisce l'ACK al client. Questo assicura una maggiore coerenza dei dati tra i nodi replica.

La scelta tra le due opzioni dipende dalle esigenze dell'applicazione e dalle considerazioni sulla consistenza dei dati e la tolleranza ai guasti.

Elevata consistenza - RDBMS

Affinchè si raggiunga un'**elevata consistenza**, la somma del **livello di consistenza in lettura** e del **livello di consistenza in scrittura** deve essere maggiore del numero totale di nodi replica associati a una data chiave. Inoltre, il livello di consistenza in scrittura deve essere maggiore della metà dei nodi replica.

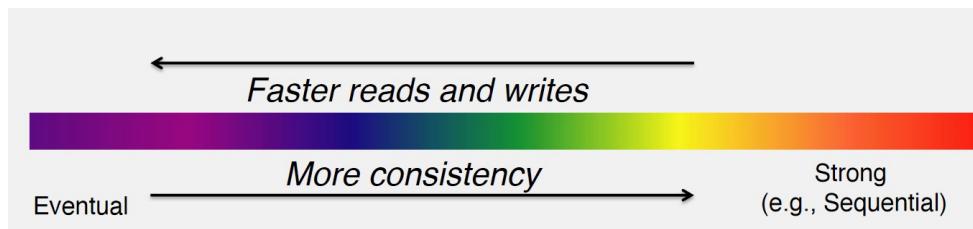
$$W + R > N \quad ; \quad W > \frac{N}{2}$$

Esempio:

- $W = 1, R = 1$: poche letture e scritture;
- $W = N, R = 1$: ottimo per carichi pesanti in lettura, dunque faccio fare tutto ad 1 solo nodo per non sovraccaricare la rete;
- $W = N/2 + 1, R = N/2 + 1$: ottimo per carichi pesanti in scrittura;
- $W = 1, R = N$: ottimo per carichi pesanti in scrittura con almeno un client che scrive per chiave.

Nota: Con R si intende il numero di nodi interpellati in lettura

Lo spettro di consistenza



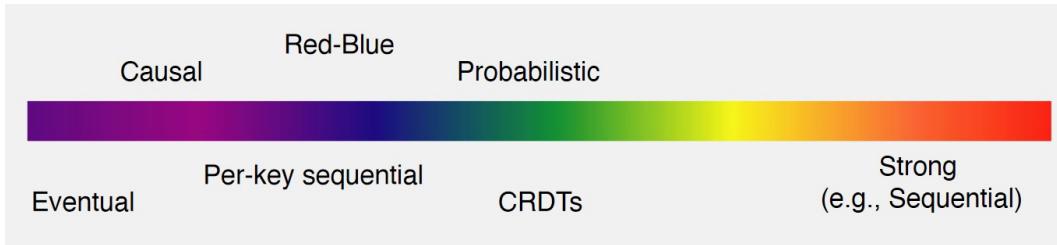
La figura sovrastante mostra quello che è lo **spettro di consistenza**, e si può constatare che:

- Un livello di consistenza definito come **eventuale** comporta dei tempi di accesso in lettura e scrittura di gran lunga inferiori, visto che i coordinatori non devono effettuare numerose verifiche;
- Un livello di consistenza definito come **strong** (duro) garantisce maggior affidabilità, a discapito tuttavia dei tempi di accesso.

Come detto prima, Cassandra offre un livello di consistenza **eventuale**, per cui si va a posizionare totalmente a sinistra nello spettro. In tale contesto, la convergenza avverrà nel momento in cui le scritture su una chiave terminano.

Nuovi modelli di consistenza

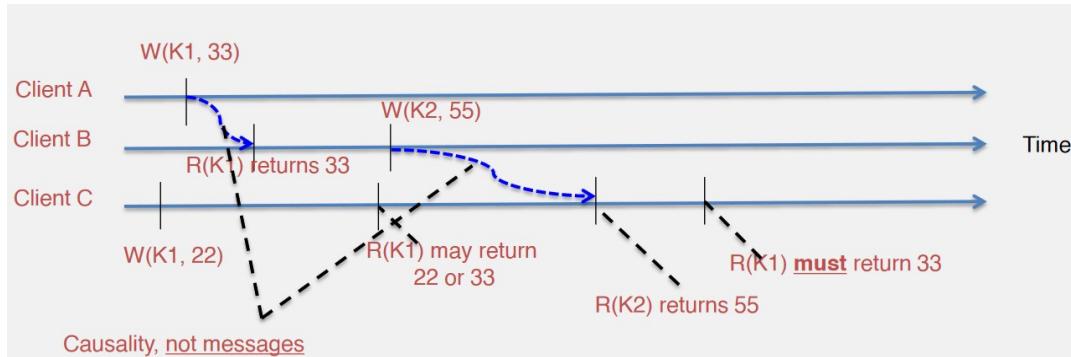
Aldilà di quanto visto finora, ci sono dei nuovi modelli di consistenza, suddivisi come di seguito:



In particolare sono:

- **Per-key sequential:** Per ogni chiave specifica, tutte le operazioni di lettura e scrittura vengono eseguite in un ordine globale determinato dal coordinatore. Ciò significa che tutte le operazioni relative a una determinata chiave vengono eseguite una dopo l'altra in sequenza, garantendo un ordine definito per le operazioni su quella chiave.
- **CRDTs (Commutative Replicated Data Types):** Si tratta di strutture dati distribuite che possono essere combinate (commutate) in modo indipendente da un coordinatore. Ciò consente di eseguire operazioni su dati replicati senza la necessità di coordinazione centrale, poiché le operazioni possono essere commutate in qualsiasi ordine senza compromettere la coerenza.
- **Red-blue consistency:** Questa è una strategia per gestire le transazioni dei clienti che separa le operazioni in due categorie: operazioni "red" e operazioni "blue".
 - Le operazioni "blue" possono essere eseguite e combinate (commutate) in qualsiasi ordine tra i diversi data center. Ciò significa che le operazioni "blue" possono essere eseguite in parallelo e l'ordine in cui vengono eseguite o combinate non influisce sulla coerenza dei dati.
 - Le operazioni "red" devono essere eseguite in ordine all'interno di ciascun data center. Ciò garantisce che le operazioni "red" vengano applicate seguendo un ordine specifico all'interno di ciascun data center, mantenendo la coerenza dei dati.
- **Linearizzabilità:** La linearizzabilità è una proprietà di un sistema che garantisce che ogni operazione effettuata da un client sia immediatamente visibile a tutti gli altri client. In altre parole, l'effetto di un'operazione viene percepito da tutti gli altri client come se fosse avvenuto istantaneamente, creando un ordine globale e coerente delle operazioni.
- **Consistenza sequenziale:** La consistenza sequenziale è una proprietà che assicura che il risultato di ogni operazione sia lo stesso se le operazioni di tutti i processori (o client) vengono eseguite in sequenza, e le operazioni di ogni processore vengono eseguite nell'ordine specificato dal suo programma. Questo significa che, se le operazioni vengono eseguite una dopo l'altra in un certo ordine definito, il risultato complessivo sarà lo stesso indipendentemente dal modo in cui le operazioni vengono suddivise tra i processori.
- **Causal consistency:** Questo tipo di consistenza richiede che le operazioni di lettura rispettino un certo ordine parziale basato sul flusso di informazioni. In altre parole, se un'operazione A dipende da un'operazione B, allora l'operazione A deve leggere i risultati di B o di operazioni successive a B. Questo assicura che le operazioni di lettura riflettano le dipendenze causali tra le operazioni di scrittura, garantendo una consistenza logica dei dati.

A titolo di esempio, guardiamo cosa accade nella figura sottostante. Si hanno tre client **A**, **B** e **C**:



1. Il client **A** esegue una scrittura alla cui chiave **K1** associa il valore **33**;
2. Poco tempo dopo, il client **B** (che non comunica in alcun modo col client A), chiede la lettura di **K1** e questo torna **33**;
3. Poco tempo dopo, il client **B** scrive (**K2, 55**) e il client **C** chiede la lettura di **K2**, che dovrà tornare ovviamente **55**;
4. Poco tempo dopo, il client **C** chiede la lettura di **K1** che **dovrà** tornare **33**.

Si può notare tuttavia che proprio il client **C** ha effettuato su **K1** prima del client **A** una scrittura (**K1,22**). Questo permette di constatare la causalità degli eventi, in quanto 33 è l'ultimo valore scritto su K1 e nonostante **C** non comunichi con **A**, il valore più vecchio non viene più ritornato.

HBase

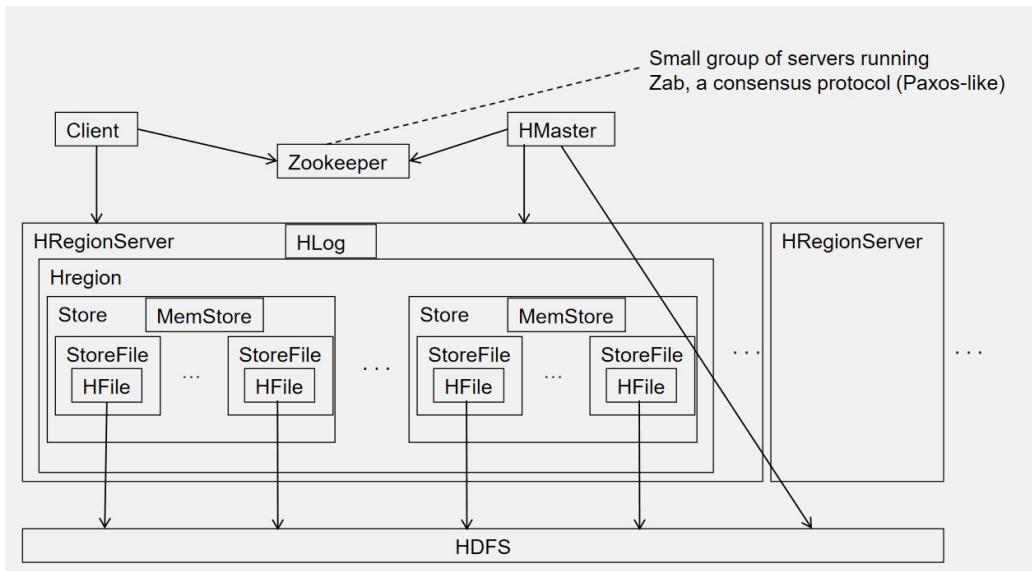
HBase è un database NoSQL (Not Only SQL) ad alte prestazioni, distribuito e scalabile, progettato per archiviare grandi quantità di dati strutturati o semi-strutturati. È un componente chiave della piattaforma Apache Hadoop e offre una soluzione per archiviare dati strutturati su un cluster di macchine. HBase è stato sviluppato da **Google** e si tratta del primo sistema **blob-based** al mondo. Dopo che è passato sotto le mani di Yahoo!, è stato reso open source e rinominato in **HBase** (il primo nome era **BigTable**). Oggi è uno dei progetti più grandi della **Apache Foundation**.

Il sistema HBase è usato internamente da Facebook. A differenza di Cassandra, favorisce la **consistenza** a discapito della disponibilità.

Alcune caratteristiche di HBase:

- **Struttura delle tabelle:** HBase organizza i dati in tabelle, che sono simili alle tabelle in un database relazionale. Tuttavia, a differenza dei database relazionali, le tabelle HBase non hanno uno schema fisso. Ogni riga in una tabella è identificata da una chiave primaria univoca, ed è possibile aggiungere, rimuovere o modificare le colonne di una tabella in qualsiasi momento senza dover definire uno schema anticipatamente.
- **Archiviazione dei dati:** HBase archivia i dati in modo distribuito su disco. I dati vengono suddivisi in "regioni" che sono gestite da diversi nodi del cluster. Ogni regione può contenere una porzione dei dati di una tabella. Ciò consente di distribuire il carico di lavoro tra i nodi del cluster e di migliorare le prestazioni.
- **Struttura delle colonne:** le tabelle HBase possono avere un numero arbitrario di colonne, anche senza uno schema predefinito.
- **Persistenza dei dati:** HBase mantiene i dati in memoria per migliorare le prestazioni, ma garantisce anche la persistenza su disco. Questo log garantisce la durabilità dei dati in caso di guasti o arresti improvvisi del sistema.
- **Accesso ai dati:** HBase offre anche un linguaggio di interrogazione chiamato HBase Query Language (HQL), simile a SQL, per eseguire query complesse sui dati.

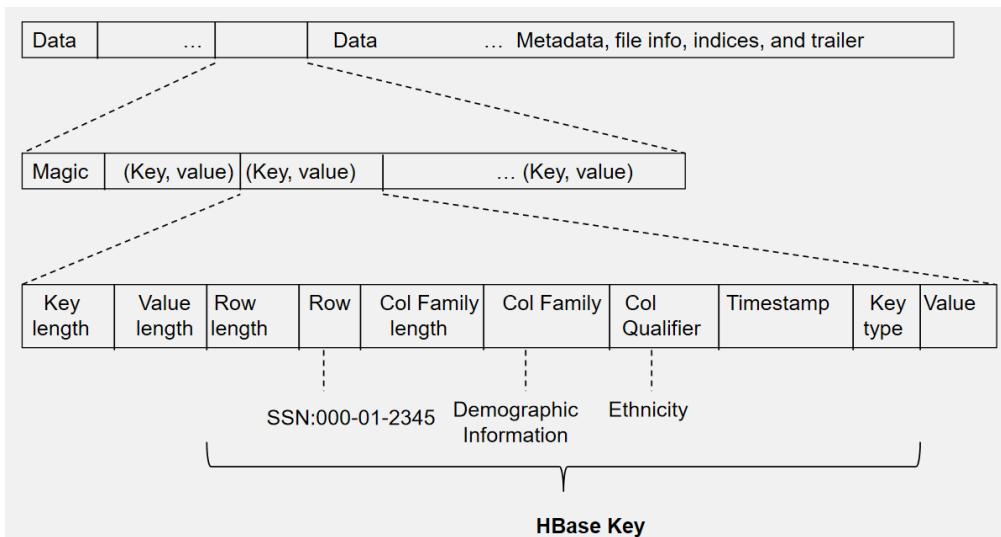
Funzionamento



Lo **schema di funzionamento** comprende:

- L'**HMaster**: è responsabile del coordinamento degli **region server** e di **HDFS**. Consente il bilanciamento del carico e la manutenzione dei server;
- Gli **HRegionServer**: sono i singoli nodi sui quali gira il sistema. Sono dotati di un file log chiamato **HLog**;
- Ciascun **HRegionServer** contiene delle regioni dette **Store**, ognuna delle quali memorizza set di righe e colonne;
- Ciascuno **Store** contiene lo **StoreFile** e un **MemStore**;
 - Lo **StoreFile** è il dato **scritto** su disco. È dotato di un **HFile** per le specifiche di HDFS;
 - Il **MemStore** è una **cache** per la scrittura in RAM, fintantochè questa non si riempie, al fine di velocizzare le operazioni.

Struttura di un HFile



L'HFile è simile come struttura a una SSTable vista in Cassandra ed è strutturato come da figura, con una serie di dati e metadati. A ciascun dato è associato un **magic number** che lo identifica univocamente nell'HFile. Per le associazioni chiave-valore ci sono info come: lunghezza della chiave, lunghezza del valore, ecc. L'intero segmento finale visto in figura viene trattato come **HBase Key**.

Consistenza in HBase

L'alto livello di consistenza in HBase è mantenuto per mezzo dell'HLog in ciascun region server

Come fa HBase a mantenere un alto livello di consistenza? La chiave di tutto è l'HLog per ciascun region server.

1. Per ciascuna chiave che deve essere scritta, viene invocata una determinata **regione**;
2. Le modifiche vengono primis scritte sull'**HLog**, così da per recuperare eventuali fallimenti;
3. Viene scritto il dato sul MemStore;
4. Quando il MemStore si riempie viene scritto il dato sul disco.

Recupero da crash (log replay)

In caso di ripristino da crash, vengono svolti i seguenti passaggi:

1. Si usano i timestamp per determinare dove si trovano i database rispetto ai log;
2. Si replicano tutte le voci a partire dal timestamp;
3. Eventuali modifiche vengono aggiunte al MemStore.

Repliche tra data center

Nel processo di replica tra data center. Si può avere un singolo cluster **master** e altri cluster **slave** che replicano le medesime tabelle.

Mappa Capitolo

