Laurea Magistrale in Informatica A.A. 2020/2021
Università degli Studi di Milano-Bicocca

Appunti Architetture dati

Marta Pelusi
@Marta629

Indice

1 DBMS centralizzati

Un database è un componente software che deve gestire dati che sono:

- grandi
- persistenti
- condivisi
- software affidabile

1.1 Struttura

I database possono essere:

- centralizzati, ovvero basi di dati isolate
- distribuiti, ovvero db che hanno dati organizati in più basi di dati e che comunicano tra loro

Esistono uno o più sistemi di storage, un solo componente logico (DBMS) e più applicazioni.

Esiste uno schema fisico (come un DBMS ha deciso di implementare la tabella) e uno schema concettuale (ad esempio il modello relazionale dei dati).
Si ha uno schema logico e un unico schema fisico collegati tra loro e si ha un'unica base di dati. Non c'è nessuna forma di eterogeneità e di distribuzione dei dati.

Esiste un unico linguaggio di interrogazione per ogni singolo DBMS, c'è un unico sistema di gestione e un'unica modalità di ripristino. Il DBA (amministratore dei dati) è unico.

I database sono grandi e persistenti

La persistenza richiede una gestione in memoria secondaria. La grandezza, invece, richiede che tale gestione sia sofisticata.
Il contenuto della memoria secondaria è trasferito nella memoria centrale tramite un buffer.
Un buffer è è un componente software che cerca di mettere in memoria centrale i dati presenti nella memoria secondaria secondo una logica di vicinanza.

I database sono condivisi

Il fatto che una base di dati è condivisa porta problemi di sicurezza, ad esempio si possono avere due prenotazioni quasi temporanee sullo stesso posto a sedere di un aereo. Di fatto l'ultimo che scrive prende il posto, ma questo non è corretto, soprattutto per una base di dati. Il posto spetterebbe al primo che lo prenota.
Bisognerebbe, quindi, organizzare il tutto in logiche seriali, come per esempio la coda in posta: (finisco io e vai tu), ma nel web è impossibile, quindi si gestisce il

controllo della concorrenza.

Le basi di dati vengono interrogate tramite query che è necessario ottimizzare.

Normalmente si vuole accedere a file nella memoria secondaria tramite query. Per farlo si potrebbe fare una scansione lineare dei dati, ma questo sarebbe un metodo estremamente oneroso per via della grande quantità di dati.
Se si avessero tabelle ordinate si potrebbe avere tempo di accesso logaritmico, ma questo non è sempre efficiente. Per questo è necessario eseguire operazioni di ottimizzazione sulle interrogazioni.
La selezione riduce la dimensione della tabella nella tabella risultato mentre il join produce una tabella risultato grande $m \times n$, con m, n dimensioni delle due tabelle di partenza.

I database sono affidabili

In caso di malfunzionamento la base di dati dev'essere affidabile, ovvero deve conservare per sempre i dati. Un esempio può essere il computer che crasha mentre si sta effettuando un pagamento alla banca.

Ottimizzazione delle interrogazioni

L'accesso a un DBMS tramite interrogazioni, cioè tramite accessi a dati nella memoria secondaria, deve avvenire in poco tempo, per questo si tende, come già detto, ad ottimizzare le query.

Esistono due oggetti che aiutano l'ottimizzazione delle interrogazioni: il data catalog e lo statistics. Il data catalog è un database che contiene informazioni sugli altri database, ovvero si avranno informazioni su tabelle, attributi, etc... di tutte le tabelle del database.
Lo statistics è un database che per ogni attributo contiene informazioni numeriche, tipo numerosità, minimo, massimo, etc...

Esempio: una query SQL può essere ottimizzata tramite diverse tecniche:

- query tree con algebra relazionale
- query plan logico ottimizzato con regole sintattiche sensate, ad esempio eseguire prima SEL di JOIN

1.2 Transazioni

Le transazioni sono un insieme di istruzioni di lettura e scrittura sulla base di dati che ne garantiscono la corretta e coerente esecuzione.

Le transazioni hanno due step fondamentali:

- commit, eseguito quando la transazione termina correttamente

- rollback, eseguito per abortire la transazione e in questo caso il sistema viene ripristinato allo stato precedente

Proprietà ACID

Le proprietà ACID sono 4 proprietà fondamentali di cui godono tutte le transazioni. Nello specifico sono:

- Atomicità: una transazione è una unità atomica di elaborazione, ovvero il db non può essere lasciato in uno stato intermedio. Si effettuano:
  - REDO, ovvero non c'è nessuna conseguenza in caso di errore dopo il commit e se necessario vanno ripetute le operazioni
  - UNDO, ovvero l'annullamento di operazioni svolte se si verifica un errore prima del commit
- Consistenza: se una transazione inizia col db in uno stato consistente, essa deve terminare col db ancora in uno stato consistente, cioè non sono ammesse soluzioni intermedie
- Isolamento: indipendentemente dal numero di operazioni è garantito il controllo di concorrenza sui dati

Gestore della concorrenza

Il gestore della conoscenza garantisce la possibilità di poter eseguire quasi in parallelo più operazioni. Si vengono a creare problemi, però, quando si vuole accedere a dati in scrittura.

Lo schedule è un insieme di transazioni che può essere generato in base all'ordine di arrivo delle operazioni.
Uno schedule è seriale se la transazione T2 inizia dopo che l'intera transazione T1 è finita.
Uno schedule è serializzabile se l'esito della sua esecuzione è lo stesso che avrebbe se le transazioni in esso contenute fossero eseguite in una (qualsiasi) sequenza seriale.

Proprietà dell'isolamento: ogni transazione dev'essere eseguita come se non ci fosse concorrenza.

Esistono diversi sistemi per il controllo della concorrenza:

- conflict equivalence

* 2PL (garantisce la serializzabilità)

* shared locks

* gestione dei deadlocks

+ timestamp

C'è bisogno di introdurre il concetto di lock, ovvero la richiesta di una risorsa in modo esclusivo), e di unlock, ovvero il rilascio di una risorsa.

2 Sistemi DEA - Distribuiti Eterogenei Autonomi

Un'architettura centralizzata prevede tante postazioni di lavoro e tante applicazioni che accedono ad un unico DBMS, il quale può gestire uno o più dischi sul suo server.
Le applicazioni inviano in modo parallelo richieste di interrogazioni, transazioni e aggiornamenti.
Questa è un'architettura ANSI/SPARC, ovvero rispecchia l'architettura delle basi di dati centralizzate. Ad esempio un'azienda con vari sedi (Milano, Roma, Padova) che ha un unico DBMS centralizzato a Milano. Questo tipo di soluzione, però, ha pro e contro. Si è preferito, poi, creare database distribuiti tra le varie sedi collegati dalla rete; questi vengono chiamati sistemi DEA o più comunemente database distribuiti.
Oltre alla frammentazione delle varie parti del database tra le sedi è importante anche replicarne i dati: ciò che vedo a Padova devo poterlo vedere anche a Milano.

Nel caso di database distribuiti, quindi, si hanno applicazioni cooperanti che risiedono su nodi diversi, e un archivio dati distribuito su più nodi.
È importante, inoltre, che i db distribuiti siano anche trasparenti: chi scrive una query può non essere consapevole che il db è distribuito, oppure può sapere che deve interrogare una tabella interrogando in realtà due tabelle risiedenti in due macchine diverse.

2.1 DEA

Come già detto i database distribuiti sono anche chiamati sistemi DEA perchè godono di tre importanti proprietà, che sono tre livelli di integrazione indipendenti tra loro:

- Distribuzione
- Eterogeneità
- Autonomia

Distribuzione

Esistono tre livelli di distribuzione:

- nessuna distribuzione
- distribuzione client/server, ovvero i nodi sono tutti su un nodo e pochi su un altro

- distribuzione peer to peer, ovvero tutti i nodi hanno lo stesso livello di identità

Eterogeneità

La stessa informazione la si può rappresentare in diversi modi:

- rispetto al modello dei dati (formato relazionale o non relazionale)
- rispetto al linguaggio
- rispetto alla gestione delle transazioni
- rispetto al sistema concettuale e schema logico

Autonomia

Indipendenza tra i vari nodi è valutata rispetto a diverse forme di autonomia. C'è la forma di autonomia rispetto a:

- progetto, ovvero ogni nodo adotta un proprio progetto
- condivisione, ovvero la scelta di porzione di dati da condividere
- esecuzione, ovvero la decisione del modo di esecuzione delle transazioni

Esistono, poi, tre tipi di basi di dati classificate in base alla loro autonomia:

- DBMS integrati che non sono autonomi, ovvero hanno dati centralizzati localmente
- sistemi semi-autonomi, ovvero ogni DBA è autonomo
- sistemi peer to peer, i quali sono totalmente autonomi

2.2 Database Distribuiti omogenei - DDBMS

Non si ha più un unico schema logico, ma ho uno schema logico globale che è associato a tanti schemi logici globali che sono delle viste (Local As View - LAV), cioè sono sottoinsiemi dello schema logico globale. Lo schema globale è sviluppato prima dello schema locale secondo, appunto un approccio LAV. In questo senso, dunque, prima della progettazione logica devo pensare a come distribuire e posizionare i dati.
Ad ogni schema logico locale corrisponde lo schema fisico locale, sebbene lo schema logico sia uno.

Per garantire l'interoperabilità tra db che forniti da diversi fornitori ma sotto lo stesso responsabile sono stati introdotti diversi metodi, come ad esempio ODBC - Database Connectivity, DTP - X-Open Distributed Transaction Processing.

Esistono diversi tipi di database distribuiti che si sviluppano in diversi paradigmi per la distribuzione dati:

- db DEA
- db parallele
- db replicate
- data warehouse

3 Frammentazione e replica

Le architetture dei database sono divise in tre categorie:

- shared everything, ovvero non c'è una distribuzione e risiede tutto in un unico disco
- shared disk, ovvero diversi componenti agiscono sullo stesso storage e questo comporta gravi problemi di controllo della concorrenza
- shared nothing, ovvero il db e il disco vivono in modo indipendente; questo permette di aggiungere indistintamente tanti nodi db-disk. Questa è una proprietà chiamata scalabilità orizzontale, infatti questo tipo di db garantisce maggiore scalabilità

3.1 Proprietà di un DDBMS

Un database DDBMS gode di quattro principali proprietà:

- località, ovvero è un vantaggio avere i dati più vicino possibile all'applicazione
- modularità, ovvero avere la possibilità di scalare i dati orizzontalmente
- prestazioni
- efficienza

Località

Avere i dati più vicini alle applicazioni è un grandissimo vantaggio. È garantita, comunque, la raggiungibilità globale, cioè ad esempio un utente da Padova può fare un'interrogazione sul nodo di Milano. Sono i dati a spostarsi verso dove le applicazioni li richiedono più frequentemente.

Modularità o flessibilità

La distribuzione è una distribuzione incrementale e progressiva dei dati, ovvero la configurazione si adatta alle esigenze delle applicazioni.

In questo caso subentra un problema: ogni componente introdotto in più (consentito in quanto è garantita la scalabilità orizzontale) può fallire e quindi

denotare maggiore fragilità del db
La presenza di ridondanza denota una maggiore resistenza a guasti di dati e applicazioni chiamati "fail soft".

Un esempio di ridondanza è espresso tramite la seguente rappresentazione: un generale A vuole inviare un messaggio ad un generale B tramite un messaggero. La situazione che si verifica, quindi, è che A invia un messaggio a B ma non è sicuro che il messaggio è arrivato. B risponde ad A, ma anch'egli non è sicuro che il messaggio è arrivato. Entrambi allora si inviano un messaggio in cui chiedono conferma che il messaggio precedente sia effettivamente arrivato. Allo stesso modo, però, non sono sicuri che questo messaggio di conferma sia arrivato ... e così via...

Prestazioni

Grazie alle elevate prestazioni una DDBMS:

- è più semplice da ottimizzare localmente
- ogni nodo può essere ottimizzato indipendentemente dagli altri
- il carico totale distribuito sui nodi
- c'è parallelismo tra le transazioni locali

Tutto questo, però, ha un costo perchè c'è bisogno di coordinamento tra i nodi e quindi c'è un'elevata presenza di traffico di rete.

Funzionalità specifiche

L'architettura utilizzata è un'architettura shared nothing, quindi ogni server ha una buona capacità di gestire in modo indipendente le applicazioni.

Il traffico di rete per i DDBMS è così organizzato:

- interrogazioni: le query provengono dalle applicazione e i risultati provengono dal server
- transazioni: le richieste di transazioni provengono dalle applicazioni e i dati di controllo per il coordinamento
- ottimizzazione: è l'elemento critico

C'è l'esigenza di distribuitre i dati per accelerare il funzionaento del sistema e si vuole che le transazioni avvengano in un record di un singolo nodo.

Le funzionalità sono:

- trasmissione di transizioni o di dati di controllo
- frammentazione/replicazione

- query processor, ovvero bisogna interrogare il db globale per poi passare ad effettuare query locali
- controllo della concorrenza
- gestione guasti globali e di singoli nodi

Caratteristiche globali

La frammentazione è la possibilità di allocare pezzi di tabelle su nodi diversi.
Esistono due tipi di frammentazione:

- orizzontale (SEL - σ), ad esempio se ho una tabella di 20 righe e le prime 10 righe vanno a destra e le altre 10 righe vanno a sinistra; si ottiengono quindi due tabelle più piccole con lo stesso schema
- verticale (JOIN, PROJ - Π), ad esempio se ho una tabella di 10 colonne e le prime 5 colonne vanno a destra e le altre 5 colonne vanno a sinistra; si ottiengono quindi due tabelle più piccole con lo stesso schema. Bisogna prestare attenzione in quanto, in questo caso, è necessario portarsi dietro anche la colonna contenente l'id della tabella

Si parla di replicazione quando un frammento o l'intera tabella sono replicati su tutti i nodi del db.
Un'applicazione è trasparente quando accede ai dati attraverso una query senza sapere dove sono allocati.

Regole di correttezza

Data R una tabella e $R_1,\ R_2$ due suoi frammenti si hanno le seguenti regole di correttezza:

- completezza: ogni nodo di R dev'essere almento in uno dei frammenti R_1 o R_2
- ricostruibilità: devo poter ricostruire R attraverso dei JOIN partendo dai ruoi frammenti R_i
- disgiunzione: ogni record dev'essere rappresentato in un solo frammento
- replicazione

3.2 Frammentazione

Frammentazione orizzontale

La frammentazione orizzontale consiste in:

- partizionare R in n relazioni/frammenti $\{R_1, ..., R_n\}$ tali che:
  - $schema(R_i) = schema(R),\ \forall i$

  - ogni R_i contiene un sottoinsieme dei record di R
  - una frammentazione è definita come $R_i = \sigma_{C_i}(R)$

- completezza: $R_1 \cup ... \cup R_n = R$

- ricostruibilità: è sempre garantita dall'unione

Frammentazione verticale

La frammentazione verticale consiste in:

- partizionare R in n relazioni/frammenti $\{R_1, ..., R_n\}$ tali che:
  - $schema(R) = L = (A_1, ..., A_m),\ schema(R_i) = L_i = (A_{i1}, ..., A_{ik})$

- completezza: $L_1 \cup ... \cup L_n = L$

- ricostruibilità: $L_i \cap L_j \supseteq$ chiave primaria di $R,\ \forall i \neq j$

Esempio: frammentazione orizzontale
$R = EMPLOYEE(Empnum, Name, Deptnum, Salary, Taxes)$
F1: $EMPLOYEE1 = \sigma_{Empnum \leq 3}(EMPLOYEE)$,
$EMPLOYEE2 = \sigma_{Empnum > 3}(EMPLOYEE)$
$EMPLOYEE = EMPLOYEE1 \cup EMPLOYEE2$

Esempio: frammentazione verticale
$R = EMPLOYEE(Empnum, Name, Deptnum, Salary, Taxes)$
F1: $EMPLOYEE1 = \Pi_{Empnum,Name}(EMPLOYEE)$,
$EMPLOYEE2 = \Pi_{Empnum,Deptnum,Salary,Tax}(EMPLOYEE)$
$EMPLOYEE = EMPLOYEE1\ join\ EMPLOYEE2$

Nota: in caso di frammentazione verticale a partire da R, se modifico la chiave primaria di R_2 perdo il collegamento con la chiave primaria di R_1. In questo ho necessità di coordinamento tra le due partizioni e sono obbligato a imporre la propagazione delle modifiche della chiave primaria sia in R_2 che in R_1.

3.3 Replicazione

Il meccanismo di replica consente a più persone di accedere (in lettura) agli stessi dati replicando gli stessi su nodi diversi.
Questo si complica quando, oltre agli accessi in lettura si hanno anche accessi in scrittura in quanto, coinvolgendo più nodi, bisogna garantire la coerenza tra i dati. Un esempio può essere un prodotto della Mediaworld di Milano che viene venduto che dev'essere comunicato ai db delle altri sedi Mediaworld.

Sono introdotte complicazioni come gestione di transazioni, updates di copie multiple, scelta di cosa replicare, quante copie mantenere, dove allocare le varie copie e come gestire le copie.

Un buon motivo per fare replica è il backup. Questo è strettamente legato alla persistenza dell'informazione (lettera D delle proprietà ACID).

Allocazione frammenti

Ogni frammento può essere allocati su un nodo diverso. È necessario, quindi, avere un catalogo, ovvero una tabella relazionale che consente di conoscere dov'è il partizionamento nei db locali. Ma a che livello il programmatore può conoscere ciò che contiene un database? È qui introdotto il concetto di trasparenza.

3.4 Trasparenza

La trasparenza indica il livello di conoscenza che un programmatore può avere del contenuto di un database.

Esistono due tipi di trasparenza:

- logica, ovvero posso creare un'applicazione che utilizza un frammento che non viene alterato se vengono modificati degli altri frammenti. Questo denota l'indipendenza dell'applicazione da modifiche sullo schema logico
- fisica, ovvero si ha una modifica sullo schema fisico (ad esempio cambiare DBMS in DDBMS). Questo denota l'indipendenza dell'applicazione da modifiche sullo schema fisico

Per poter applicare modifiche bisogna in primis accedere allo schema logico globale e poi, tramite un middleware, accedere allo schema logico locale. Il risultato della modifica lo si troverà su tutti i nodi del db.

Livelli di trasparenza

Esistono tre livelli di trasparenza:

- frammentazione
- replicazione o allocazione
- linguaggio

La trasparenza di frammentazione ignora l'esistenza di frammenti. Questo rappresenta lo scenario migliore. È importante garantire l'efficienza e la correttezza delle interrogazioni.

La trasparenza di allocazione indica che il programmatore sa che ci sono altri frammenti ma non è in grado di individuarli stabilendo dove sono collocati.

La trasparenza di linguaggio indica che l'applicazione deve specificare sia i frammenti sia il loro nodo, ovvero il nodo in cui sono allocati. Questo è il livello minimo di trasparenza.

4 Ottimizzazione query distribuite

L'immagine seguente rappresenta un sommario delle architetture di DDBMS:

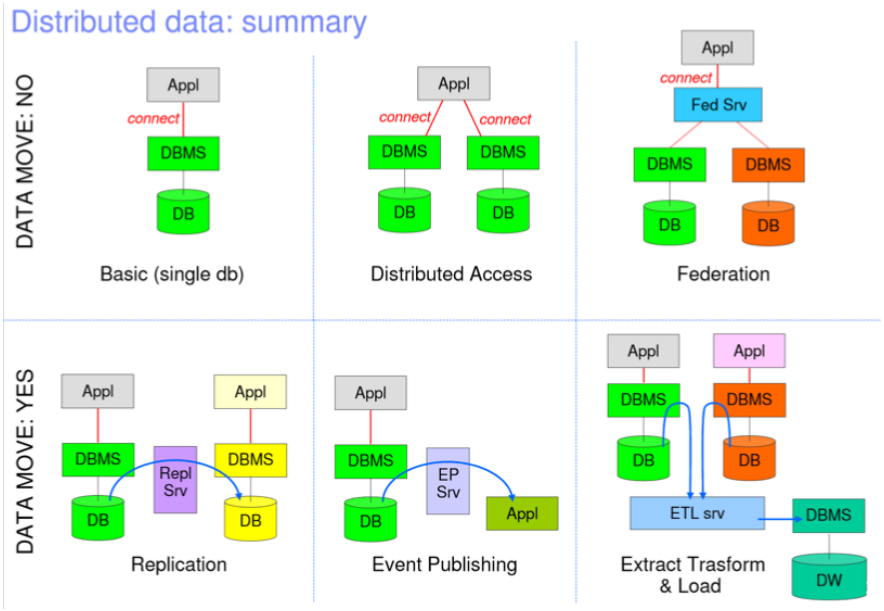

Figura 1: Distributed data: summary

Nota: la replica rende più efficiente le interrogazioni; questo funziona se e solo se esiste un componente funzionale che è in grado di tenere aggiornate tutte le repliche.

4.1 Query processing

Il query processing è il processo di interrogazione nei DDBMS.
Le interrogazioni possono essere svolte in lettura o in scrittura. Nel caso di interrogazioni in sola lettura si vuole cercare di rispondere alle interrogazioni in modo rapido ed efficace, nel caso di interrogazioni in sola scrittura, invece, si vuole evitare l'insorgere di problemi.

Quando l'utente fa un'interrogazione sul sistema globale il database deve decomporre la query dello schema globale nei singoli frammenti degli schemi locali. A questo punto va eseguita un'ottimizzazione globale.
Quando si fa una query, però, bisogna tenere in considerazione il fatto che le informazioni che sto interrogando potrebbero non essere nello stesso nodo. Nel caso in cui fossero su nodi diversi, si avrebbe bisogno di un ulteriore nodo con il quale combinare in AND le interrogazioni sui nodi diversi.

Per ottimizzare una query è necessario fare tre passaggi principali:

1. query decomposition
2. data localization
3. global query optimization

1 - Query decomposition

La query decomposition opera sullo schema logico globale e cerca di costruire un albero di interrogazione centralizzato utilizzando le proprietà dell'algebra relazionale.
L'output sarà un query tree non ottimizzato.

2 - Data localization

La data localization deve ottimizzare l'operazione rispetto alla frammentazione, ad esempio la tabella studente frammentata in modo orizzontale con alcune righe su una tabella e altre su un'altra. L'utente fa, quindi, l'interrogazione sul sistema globale e a quel punto il DBMS deve decomporre la query dallo schema globale ai singoli frammenti degli schemi locali.
L'output sarà una query non ancora ottimizzata che però opera in modo efficiente sui frammenti.

3 - Global query optimization

Per effettuare al meglio l'ottimizzazione si ha bisogno di avere informazioni sulle statistiche dei vari frammenti.
Oltre agli operatori dell'algebra relazionale vengono aggiunti anche operatori di comunicazione, ad esempio spedisci dati - ricevi dati. Alcune operazioni possono essere eseguite in parallelo.

Le decisioni più rilevanti da prendere riguardando l'operatore JOIN in quanto, date due tabelle di dimensione n ed m, questo aumenta la dimensione della tabella da trattare a $n \times m$, a differenza della SEL che riduce il numero di righe e della PROJ che riduce il numero di colonne. Il JOIN, quindi, mi costringe a confrontare tutti gli elementi di tabelle diverse. Quindi è necessario scegliere quale sia il JOIN migliore da utilizzare tra:

- ordine di join n-ari
- scelta tra join e semijoin

L'obiettivo è trovare l'ordinamento migliore delle operazioni definite nella query.
Esiste una fase di re-ottimizzazione a runtime delle query nella quale viene adattato il query plan in base ai meccanismi della rete che lo esegue.
Per query semplici eseguite su tanti nodi bisogna essere in grado di rielaborare continuamente le informazioni ed è molto importante la trasparenza.

Esempio: voglio trovare i nomi dei dipendenti che sono manager di progetti sul seguente schema:
$EMPLOYEE(eno, ename, title),\ |EMP| = 400$
$ASSIGN(eno, projectno, resp, dur),\ |ASS| = 1000$
$|\sigma_{resp="manager"}(ASS1)| = |\sigma_{resp="manager"}(ASS2)| = 10$

Query:

```
select ename
from EMPLOYEE e join ASSIGN a on e.eno=a.eno
where resp="manager"
```

In algebra relazionale diventa:
$\Pi_{ename}(EMP >< _{eno} (\sigma_{resp="manager"}(ASS)))$

A questo punto entra in gioco la distribuzione dei dati e supponiamo ci sia stata una frammentazione orizzontale:
$ASS1 = \sigma_{eno} \leq e3(ASS)$, *nodo* 1
$ASS2 = \sigma_{eno} \leq e3(ASS)$, *nodo* 2
$EMP1 = \sigma_{eno} \leq e3(EMP)$, *nodo* 3
$EMP2 = \sigma_{eno} \leq e3(EMP)$, *nodo* 4
$risultato$, *nodo* 5

Si possono applicare diverse strategie di ottimizzazione:

- strategia 1: calcolare $EMP1$ e $EMP2$ e poi calcolare $EMP1 \cup EMP2 = result$ nel nodo 5
- strategia 2: inviare tutte le tabelle al nodo 5 nel quale viene calcolato il risultato

Il costo/velocità di esecuzione dipende dai tempi di trasporto.

Confronto dei costi tra le due strategie:

- strategia 1: comportamento sequenziale
  - calcolo $ASS1$ *e* $ASS2 = 10 + 10 = 20$
  - trasferimento $ASS1$ *e* $ASS2 = 20 \cdot 10 = 200$
  - calcolo $EMP1$ *e* $EMP2$: join single-loop $= (10 + 10) \cdot 2 = 40$
  - trasferimento $EMP1$ *e* $EMP2 = 20 \cdot 10 = 200$
  - TOTALE: 460 (400 per il trasferimento e 60 per il calcolo)
- strategia 2: comportamento in parallelo (non si trasferiscono gli indici)
  - trasferimento $EMP1$ *e* $EMP2$ sul nodo 5: $= 400 \cdot 10 = 4000$
  - trasferimento $ASS1$ *e* $ASS2$ sul nodo 5: $= 1000 \cdot 100 = 10000$
  - calcolo ASS (selezione): 1000 (no indice)
  - join nested-loop ASS *e* $EMP = 20 \cdot 400 = 8000$ (non si ricostruiscono gli indici)
  - TOTALE: 23000 (14000 per il trasferimento e 9000 per il calcolo)

Costo di comunicazione

Il costo totale di una query è composto dai costi di operazioni/calcolo + i costi di comunicazione/trasferimento.

Il costo di comunicazione è così definito:

$$\text{costo di comunicazione} = C_{MSG} * \#msgs + C_{TR} * \#bytes$$

- C_{MSG}=costo fisso di spedizione/ricezione di un messaggio
- C_{TR}=costo fisso (rispetto alla tipologia) di trasmissione dati

Nota: il costo di comunicazione è molto maggiore rispetto al costo di I/O.

Nel caso di due nodi che mandano due unità X e Y in un unico nodo, vedi figura sotto:

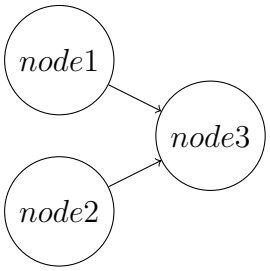

Figura 2: Esempio di due nodi che trasferiscono in parallelo unità in un terzo nodo.

Costo di comunicazione $= 2 \cdot C_{MSG} + C_{TR} * (x + y)$
Tempo di risposta $= \max\{C_{MSG} + C_{TR} * x,\ C_{MSG} + C_{TR} * y\}$
Se si vuole minimizzare il tempo di risposta si deve tener conto del parallelismo ma il tempo totale peggiora perchè c'è un maggior numero di trasmissioni.
Se invece si vuole minimizzare il costo totale non si deve tener conto del parallelismo e c'è un migliore utilizzo delle risorse con un aumento di throughput.

4.2 Join e semijoin

Date due tabelle $R,\ S$ si suppone di voler fare $R\ join\ S$, con R posizionata nel nodo 1 e S nel nodo 2. Definiamo R^* come l'insieme di tutti gli attributi di R.

Il semijoin è così definito:

$$R\ semijoin_A\ S\ \equiv \Pi_{R^*}(R\ join\ S)$$

Il semijoin $R\ semijoin_A\ S$ è la proiezione sugli attributi di R della operazione di join.

Nota: il semijoin non è commutativo!

Valgono le seguenti equivalenze:

- $R\ join_\theta\ S \Leftrightarrow (R\ semijoin_\theta\ S)\ join_\theta\ S$
- $R\ join_\theta\ S \Leftrightarrow R\ join_\theta\ (S\ semijoin_\theta\ R)$
- $R\ join_\theta\ S \Leftrightarrow (R\ semijoin_\theta\ S)\ join_\theta\ (S\ semijoin_\theta\ R)$

Esempio: $R\ join_A\ S \Leftrightarrow (R\ semijoin_A\ S)\ join_A\ S$

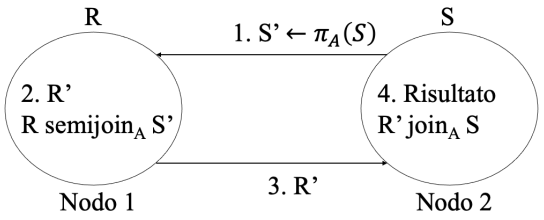

Figura 3: Esempio di semijoin

Nota: in un DDBMS per rete geografica si vogliono ridurre i costi di ottimizzazione, mentre in un DDBMS locale si vuole aumentare il parallelismo.

5 Repliche

Il processo di replica è un processo che consente di creare e mantenere istanze dello stesso db allineate tra loro. Questo permette la condivisione dei dati e tiene in considerazione cambiamenti progettuali nel db, di cui tutte le parti devono vedere le modifiche.

La sincronizzazione consente di avere tutte le copie allineate, sapendo che la replica può essere fatta sia in modo sincrono che in modo asincrono.

Si parla di repliche sincrone quando tutte le repliche vengono aggiornate contemporaneamente tramite un meccanismo che ricorda il protocollo ROWA - Read Once Write All. Di conseguenza quando si applicano modifiche tutti i nodi devono essere aggiornati prima di poter fare qualsiasi altra cosa.
Garantisce:

- alta disponibilità
- si riduce la perdita di dati, e se si dovessero perdere dati non si avrebbero particolari danni
- ci sono problemi di scalabilità
- ci sono problemi di costo

Si parla di repliche asincrone quando le repliche possono essere aggiornate in un secondo momento rispetto a quando viene apportata una modifica sul db.
Garantisce:

- basso costo

- alta scalabilità
- alta flessibilità
- efficienza degli accessi e accessi online
- sussiste il problema della perdita dei dati nel tempo che intercorre tra quando carico i dati nello storage principale e quando aggiorno in un secondo momento la replica

5.1 Contesti di replica

Esistono diversi contesti di replica.

Shared data

Il contesto shared data indica la condivisione di dati da parte di utenti scollegati tra loro.
Per comprenderlo al meglio consideriamo l'esempio del commesso viaggiatore: un commesso va da cliente a cliente per cercare di vendere i suoi prodotti. Questo è un caso in cui l'aggiornamento continuo della replica dei dati (supponendo che il commesso abbia una replica del suo archivio di prodotti) non è vantaggiosa in quanto i dati non farebbero in tempo ad aggiornarsi nel poco tempo in cui il commesso cambia cliente.

Con la merge replication si cerca di rispondere al caso in cui i nodi non sono sempre connessi alla rete e si ha necessità di effettuare una replicazione con merge.

Consolidation

Le aziende con catene di negozi possono avere dati in ogni sede loro sede.
Le modifiche vengono fatte nel livello più basso dei dati e c'è bisogno di riportarle tutte nel livello centrale. Ogni sera, infatti, le aziende aggiornano i dati e replicano le loro informazioni sul db centrale per tenere traccia di merci disponibili, merci vendute etc...

Data distribution

Data distribution è il caso più critico in cui si deve aumentare il più possibile l'accesso ai dati e c'è una sincronizzazione bidimensionale in real time, ad esempio il db di amazon. È importante riuscire a tenere aggiornati tutti i dati, nel caso di amazon, ad esempio, è importante per tenere aggiornati tutti i prodotti disponibili in qualsiasi momento.
Lo svantaggio di questo sistema è che è necessario rispondere immediatamente al click di un cliente che acquista e dicendo se il prodotto è ancora disponibile oppure no.

Data accessibility

Se nella soluzione che si vuole costruire non sono necessari tanti update e non c'è bisogno che siano immediati, allora la replica è perfetta per garantire l'accessibilità.
Si ha, quindi, un server secondario che aggiorna i dati, migliora i tempi di risposta tramite un'amministrazione non centralizzata e garantisce un miglior accesso ai dati.

Load balance

Load balance è una variante della modalità di replica: se troppi utenti vogliono accedere contemporaneamente ai dati, si utilizza una maggior potenza di CPU, quindi invece che riprogettare tutto sposto gli utenti nelle diverse macchine.

Esempio: su Facebook una persona per account scrive e tutti vogliono leggere subito quello che è stato scritto.
Esempio: nel caso della banca si hanno diverse persone che possono accedere contemporaneamente al proprio conto e le richieste devono essere soddisfatte subito, ad esempio prelievo, bonifico, etc...

Disponibilità

La replica è utilizzata per aumentare la disponibilità di un db; questo è proprio di sistemi disaster recover.
È necessario fare planning e testing di tutte le soluzioni disponibili, non basta solamente utilizzare una rete di backup.

Separating data entry and reporting

Se un db subisce una grande quantità di data entry e gli stessi dati sono usati anche per il reporting può essere utile separare questi due elementi.
Se si vuole fare un reporting continuo dei dati è perchè si vuole leggere quei dati. Se i dati, però, sono bloccati io rischio di bloccare troppo i report. Copiando i dati e disaccoppiando data entry e reporting la situazione può migliorare.
Per separare i dati è importante evitare i locking.

Application co-existence

Per comprendere meglio application co-existence facciamo un esempio: sia data una nuova versione di una applicazione che richiede cambiamenti e/o aggiornamenti. In qualche modo bisogna impostare in modo diverso i sistemi passando dalla v1.0 alla v1.1. Non si può spegnere il sistema trasferendo tutti i dati dalla vecchia alla nuova app e poi riaccenderlo. Per esempio durante un aggiornamento di s3 sarebbero da trasferire una marea di dati, ma non si possono interrompere per più giorni le attività didattiche per lo switch dei dati. Questa situazione la si gestisce facendo coesistere i vecchi db con i nuovi db. Questo, però, è complicatissimo e costosissimo in quanto richiede una migrazione di dati perchè questi

vanno copiati dalla applicazione vecchia a quella nuova.

Un database replicato non dovrebbe essere utilizzato se:

- sono presenti frequenti update su più copie
- la consistenza è fondamentale

I database delle banche rispecchiano queste due caratteristiche ma, per sicurezza, va ugualmente replicato.

La garanzia della consistenza dei dati costa molto in termini di performance e in termini economici.
In alcuni sistemi, per garantire alta disponibilità e partizionamento di reti distribuite, si rinuncia alla consistenza.

All'aumentare del numero delle righe inserite durante gli update si creano dei conflitti che poi si devono andare a risolvere.

Esempio: nel database di amazon è presente un conflitto quando in magazzino ci sono 5 prodotti ma se ne sono venduti 7. Amazon a questo punto decide di risolvere il conflitto o ritardando la consegna oppure mandando una mail dicendo che il prodotto è terminato e che verranno restituiti i soldi. In questo caso il conflitto è risolto manualmente.

La consistenza delle proprietà ACID diventa quindi la componente fondamentale. Solitamente viene imposto un protocollo ROWA (Real Once Write All) che riduce le performance ma garantisce la consistenza dei dati.

Esistono diverse tipologie di repliche:

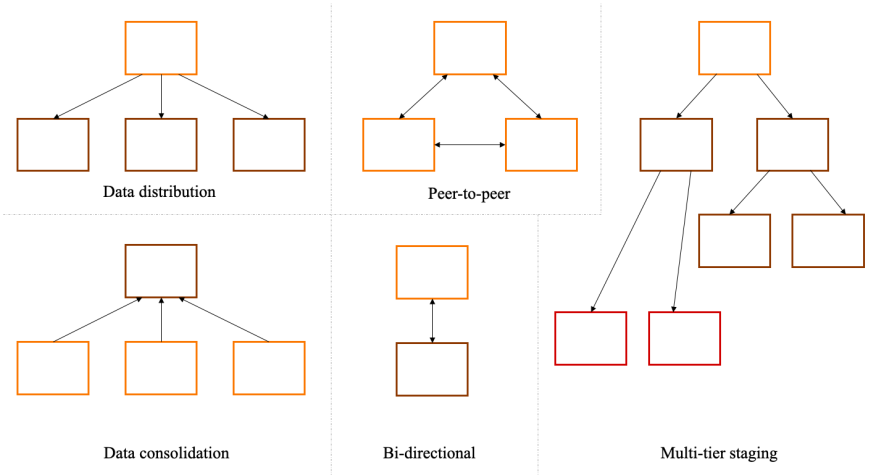

Figura 4: Tipologie di repliche.

5.2 Come realizzare una replica

Una replica si può realizzare in diversi contesti.
Un possibile contesto di replica è fare un backup dei miei dati (replica), copiarlo su un altro file, staccarlo dal mio server e trasportarlo manualmente altrove.
Un altro contesto di replica, invece, viene chiamato replica incrementale: viene eseguito un primo full backup tra la copia principale e la copia secondaria e viene trasportato altrove solo il file di log transazionale. Questo ha il vantaggio di catturare i cambiamenti su una tabella guardando solamente le ultime modifiche sul file di log.

6 Transazioni distribuite

Esistono principalmente tre tipi di transazioni per l'accesso a database distribuiti:

- transazioni read only
- remote transaction (hanno un numero arbitrario di operazioni di scrittura, select, insert, delete e update)
- distributed transaction (ogni operazione viene svolta in un unico server, i quali possono modificare più di un db)

Come per tutte le transazioni anche qua valgono le proprietà ACID, e:

- la consistenza non dipende dalla distribuzione dei dati perchè i vincoli descrivono le proprietà logiche dello schema
- la durabilità è garantita localmente

Isolamento e atomocità diventano molto importanti.

6.1 Controllo della concorrenza

Esempio: ogni transazione può essere scomposta in sottotransazioni che possono essere associate a più nodi:
$T_1 : \ r_{11}(x) \ w_{11}(x), \ r_{12}(y), \ w_{12}(y)$
$T_2 : \ r_{22}(y) \ w_{22}(y), \ r_{21}(x), \ w_{21}(x)$
La transazione 1 sulla risorsa x avviene sul nodo 1, mentre la transazione 2 sulla risorsa y avviene sul nodo 2.

C'è una condizione di serializzabilità: uno schedule serializzabile è uno schedule che ha comportamento equivalente ad uno schedule seriale.
C'è un conflitto globale: i due schedule sembrano localmente serializzabili ma in realtà globalmente c'è un problema. I due schedule singolarmente sono seriali, ma nel loro complesso ci sono dei conflitti in quanto la risorsa x è "prenotata" da w_{11} sul nodo 1, quindi T_2 non può andare avanti nella sua esecuzione con r_{21} perchè T_1 non è ancora terminata. Allo stesso modo però non si può eseguire r_{12} perchè T_2 non è terminata in quanto sta aspettando che termini T_1.
Uno schedule globale è serializzabile se gli ordini di serializzazione sono gli stessi per tutti gli schedule coinvolti.

6.2 Controllo delle repliche

T_1 e T_2 sono due transazioni che fanno due scritture che possono avvenire sulla stessa replica, ma ciò diventa un problema se scrivono su risorse diverse, come nell'esempio precedente:
$S_1 : \ r_{11}(x)1 \ w_{11}(x), \ r_{21}(x) \ w_{21}(x)$ al nodo 1
$S_2 : \ r_{22}(x) \ w_{22}(x), \ r_{12}(x) \ w_{12}(x)$ al nodo 2
In questo caso w_{11} è inficiata perchè sul nodo 2 stiamo leggendo la stessa risorsa con w_{22} avendola prima modificata con r_{22}.
I due schedule singolarmente sono seriali, ma globalmente viene violata la mutua consistenza per la quale al termine della transazione tutte le copie devono avere lo stesso valore.
Si ha, quindi, bisogno di un protocollo di controllo delle repliche: ROWA (Read Once Write All).

Protocollo ROWA

Se si ha un oggetto X con copie $x_1, x_2, ..., x_n$ il protocollo consente la lettura da qualsiasi copia ma si ha l'obbligo di scrivere su tutte le copie.

È utile avere delle repliche quando si spostano i dati in modo che qualsiasi nodo può leggere da qualsiasi db lo stesso dato. Questo si realizza imponendo la scrittura su tutte le copie e finchè anche l'ultima copia non ha finito la scrittura la transazione non può terminare. Questo meccanismo, inoltre, semplifica l'operazione di lettura in quanto si legge dal nodo più vicino al client. Tutto ciò, però, comporta una perdita in performance ma è garantita la consistenza.

6.3 Protocollo two Phases Locking in ambiente distribuito - Protocollo 2PL

Il protocollo 2PL prevede che quando si ha una transazione, questa deve chiedere tutti i lock prima di rilasciarne anche solo uno. Un lock viene rilasciato solo quando la transazione ha fatto il commit.

Facciamo l'elenco delle figure fondamentali:

- lock manager LM, ovvero il coordinatore che gestisce tutto
- transaction manager TM, ovvero il nodo da cui inizia la transazione
- slave, ovvero coloro che eseguono ciò che fa il LM

Strategia 2PL centralizzato:

1. il TM coordinatore formula al LM coordinatore le richieste di lock
2. il LM concede/assegna i lock utilizzando un 2PL
3. il TM comunica i lock e l'accesso ai dati ai database partecipanti

4. i database partecipanti comunicano la fine delle operazioni

5. il TM rilascia i lock al LM coordinatore

Questo crea un problema in quanto si crea un collo di bottiglia verso il LM coordinatore perchè gli arrivano tutte le richieste di lock. Per evitare questo problema si ricorre alla strategia della copia primaria del 2PL. Questo è un meccanismo che individua una copia primaria di ogni risorsa prima dell'assegnazione dei lock, e quindi per ogni risorsa il TM comunica le richieste di lock al LM responsabile solo della copia primaria che assegna il lock.

6.4 Deadlock distribuito

Il deadlock è un'attesa circolare tra due o più nodi.
Questo causa problemi, dunque è necessario definire un algoritmo asincrono e distribuito (peer-to-peer) che rilevi i deadlock.
Notazione: t_{ij} è la sottotransazione della transazione t_i al nodo j.

Assumiamo che le sottotransazioni siano attivate in modo sincrono, ovvero t_1 attende t_2.
Questo può generare due tipi di attesa:

- attesa da remote procedure call, ovvero t_{11} sul nodo 1 attende t_{12} sul nodo 2 perchè aspetta la sua terminazione

- attesa da rilascio di risorsa, ovvero t_{11} sul nodo 1 attende t_{21} sullo stesso nodo perchè attende il rilascio di una risorsa

La composizione di queste due condizioni può dar luogo a un deadlock generalizzato.

Ciascun nodo deve capire quali sono le transazioni che sono in attesa.
La sequenza di attesa generale al nodo k è: $EXT_i < t_{ik} < t_{jk} < EXT_j$, dove EXT è una chiamata all'esecuzione della transazione sul nodo j-esimo, mentre con $x < y$ si intende che x sta aspettando che y rilascia una risorsa.

Esempio: deadlock distribuito

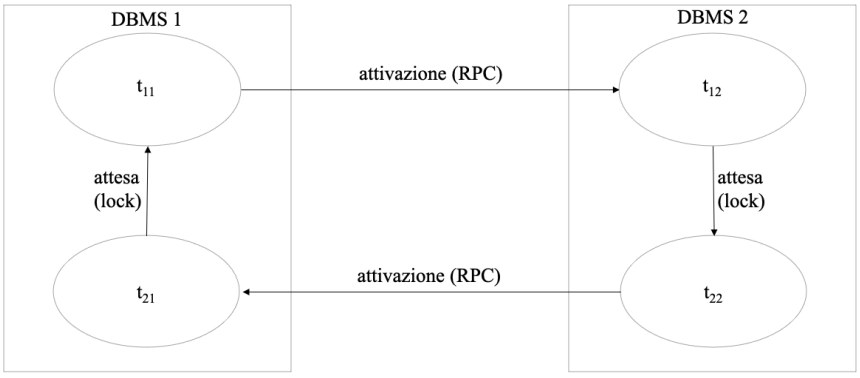

Figura 5: Esempio deadlock distribuito.

Nota: un deadlock è rilevato in presenza di cicli.

Esiste un algoritmo per risolvere il problema del deadlock distribuito dove ogni nodo deve:

1. integrare la sequenza di attesa con le condizioni di attesa locale degli altri nodi (legati da EXT)
2. analizzare le condizioni di attesa e rilevare eventuali deadlock locali
3. comunicare le sequenze di attesa ed altre istanze dell'algoritmo

6.5 Recovery management - atomicità

Un sistema può essere soggetto a guasti locali o perdite di messaggi sulla rete.

Esistono diversi tipi di guasti:

- guasti nei nodi
- perdita di messaggi
- partizionamento della rete, ovvero nodi che si vedono tra loro ma non vedono più il resto della rete

Protocollo Two phases commit - 2PC

I protocolli di commit consentono ad una transazione di giungere ad una decisione di abort/commit su ciascuno dei nodi che partecipano ad una transazione. L'idea è quella di decidere di fare commit o abort tra due o più partecipanti coordinata da un ulteriore partecipante.

I partecipanti sono:

- transaction manager TM, ovvero il nodo da cui inizia la transazione
- resource manager RM, ovvero i server

Il protocollo 2PC si basa sullo scambio di messaggi tra TM coordinatore e RM, i quali mantengono ognuno il proprio file di log.

- prima fase: il TM chiede a tutti i nodi come intendono terminare la transazione e ogni nodo decide autonomamente se fare commit o abort
- seconda fase: se tutti i nodi decidono di fare commit allora il TM comunica la decisione di fare commit e si chiude la transizione; se anche solo un nodo decide di fare abort allora il TM coordinatore comunica l'abort

In assenza di guasti il file di log registra:

- record di transizione

- record di sistema, come per esempio eventi di checkpoint o di dump

Ogni decisione presa va scritta nel file di log perchè, in caso di guasto, si può risalire all'ultimo stato in cui il sistema si trovava prima del guasto.

I record da aggiungere al file di log sono:

- da parte del TM:
  - prepare: contiene l'identità di tutti i RM (nodi+transizioni)
  - global commit/abort: decisione globale, ovvero la decisione del TM diventa esecutiva quando il TM scrive "global commit/abort" sul proprio file di log
  - complete: scritto alla fine del protocollo
- da parte dei RM:
  - ready: disponibilità irrevocabile del RM a partecipare alla fase di commit
  - not ready: indisponibilità del RM a fare commit

Sono inseriti dei **timeout** per fissare un tempo di risposta da parte dei RM.

Prima fase 2PC

Nella prima fase del protocollo 2PC il TM raccoglie le risposte dei RM.
Se tutti gli RM dicono commit, allora TM comunicherà (nella seconda fase) di fare global commit, altrimenti anche se un solo RM comunica di voler fare abort o not-ready, allora il TM comunicherà (nella seconda fase) di fare global abort.

Seconda fase 2PC

Nella seconda fase del protocollo 2PC il TM comunica la decisione globale e fissa un nuovo timeout. Questo passaggio continua finchè tutti i RM non mandano un ack/risposta.
Il TM raccoglie tutte le risposte. Se scatta il timeout il TM ne fissa uno nuovo e ripete la trasmissione a tutti i RM dai quali non ha ancora ricevuto un ack. Quando tutti i RM hanno risposto, allora il TM scrive sul suo file di log "complete".

Ci sono tre diversi paradigmi di comunicazione tra TM e RM, ovvero:

- centralizzato (quello spiegato sopra), ovvero la comunicazione avviene solo tra TM e RM e non c'è comunicazione tra RM
- lineare, ovvero gli RM comunicano tra loro secondo un ordine prestabilito e il TM è il primo nell'ordine
- distribuito

  - nella prima fase: TM comunica con gli RM, gli RM inviano le decisioni agli altri partecipanti. Ogni RM decide in base ai voti che "ascolta" dagli altri partecipanti
  - nella seconda fase: non serve!

In caso di guasto le cose si complicano un po'.

2PC in caso di guasto

Un RM nello stato ready perde la sua autonomia e attende la decisione del TM. Se dovesse esserci un guasto nel TM l'RM viene lasciato in uno stato di incertezza. L'intervallo tra la scrittura di "ready" nel file di log dei RM e la scrittura di commit/abort è detta finestra di incertezza. Uno degli obiettivi del protocollo 2PC è ridurre al minimo la finestra di incertezza.

Guasti di componenti

In caso di guasti di componenti devono essere utilizzati protocolli con due diversi compiti:

- protocolli di terminazione: assicurano la terminazione della procedura
- protocolli di recovery: assicurano il ripristino

Guasti di componenti - guasto di un resource manager

L'RM può cadere in più situazioni:

- all'inizio: comporta un abort, infatti il TM avvia un timeout nell'attesa della risposta del RM. Se non riceve risposta può decidere solo un global-abort
- dopo il ready: si va a controllare nel file di log la risposta, oppure la si domanda al TM e si attende la sua risposta, infatti in questo caso il TM deve continuare a trasmettere la decisione presa di global-commit/abort fino alla ricezione del consenso. Quando l'RM con guasto si riprende manda un messaggio di consenso al TM

Guasti di componenti - guasto di un transaction manager

Ipotizziamo che anche i RM possano attivare un timeout (non è possibile, è solo per capire meglio la situazione).
Il TM può cadere in diverse situazioni:

- stato iniziale
  - l'RM attende il messaggio "prepare"
  - l'RM può fare abort
- stato ready

  - l'RM ha deciso di fare commit e attende la decisione del TM
  - l'RM resta bloccato in uno stato di attesa
  - quando il TM si riprende si vota per fare abort
  - se gli RM comunicano tra loro è possibile sbloccare la situazione decidendo tutti insieme (paradigma lineare o distribuito)

Ripristino del guasto per la caduta di un resource manager

Se cade il RM si possono avere diversi casi:

- l'ultimo record nel log è azione, commit o abort, in questo caso si usa un warm restart per fare abort o azione (undo della transazione) oppure commit (redo della transazione)

- l'ultimo record nel log è ready, e in questo caso:

  - il RM si blocca perchè non conosce la deicisione del TM
  - si fa un warm restart
  - sol 1: quando il TM coordinatore si riprende il partecipante gli chiede cos'è successo (richiesta di remote recovery)
  - sol 2: quando si riprende il TM coordinatore riesegue la seconda fase del 2PC

Ripristino del guasto per la caduta del transaction manager

Se cade il TM coordinatore si possono avere diversi casi:

- l'ultimo record nel log è "prepare", quindi il guasto del TM può aver bloccato alcuni RM, di conseguenza si può decidere global abort e procedere con la seconda fase del 2PC oppure riprendere la prima fase sperando di arrivare ad un global commit

- l'ultimo record nel log è "global-commit/abort", quindi alcuni RM potrebbero non essere informati della decisione e di conseguenza il TM deve ripetere la seconda fase

- l'ultimo record nel log è "complete", cioè la caduta del TM non ha effetti

Perdita di messaggi e partizionamento della rete

Il TM non è in grado di distinguere tra la perdita di messaggi in stato "prepare" o "ready" nella prima fase del 2PC. In entrmabi i casi, per questo motivo, la decisione è global-abort a seguito di timeout. In entrambi i casi, poi, va ripetuta la seconda fase sempre a seguito di timeout.
Un partizionaento della rete non causa molti problemi dato che una transazione può avere successo solo se il TM e tutti gli RM appartengono alla stessa partizione.

Ottimizzazioni

Esistono due tipi di ottimizzazione, il cui obiettivo è quello di ridurre il numero di messaggi trasmessi tra TM coordinatore e RM partecipanti e ridurre il numero di scritture nei file di log.

- ottimizzazione read-only: quando un RM sa che la propria transizione non contiene operazioni di scrittura non influenza l'esito finale della transazione, quindi risponde "read-only" al messaggio di prepare e termina l'esecuzione del protocollo. Nella seconda fase il TM ignora i partecipanti read-only, infatti questi, per cui è noto a priori che siano di sola lettura, possono essere esclusi dal protocollo.

- ottimizzazione presumed abort: il TM abbandona la transazione subito dopo aver deciso per un abort e non scrive sul suo file di log "global-abort". Questo implica che non aspetta la risposta dei RM. Perciò quando il TM riceve una richiesta di remote recovery da parte di un RM che non sa come procedere in seguito a un guasto, e il TM non trova un abort nel file di log, quindi il TM decide sempre per global-abort. Gli unici record che vengono scritti sono "blobal commit", "ready" e "commit".

7 Blockchain

Una blockchain è un registro pubblico (tutti possono vedere chi possiede cosa e la storia delle proprietà seguendo le transazioni) condiviso (gestito da più persone) decentralizzato (una delle persone che lo gestisce non ha poteri particolari di amministrazione, ovvero tutti possono fare le stesse cose) che memorizza la proprietà di beni digitali. Memorizzano anche le transazioni, cioè come cambiano le proprietà.

Questo registro è organizzato in blocchi. I blocchi neri formano una catena di blocchi (blockchain). I blocchi sono legati da funzioni crittografiche chiamate funzioni di hash crittografico.
Il collegamento di due blocchi è dato dal fatto che il valore di hash del blocco è collegato al blocco successivo. Questo significa che è molto difficile alterare il contenuto di un blocco (il blocco contiene transazioni). Se voglio modificare una transazione in un blocco ricalcolo l'hash e vado a vedere se questo valore corrisponde a quello nel blocco successivo - validità blockchain (se non combaciano c'è qualcosa che non va).

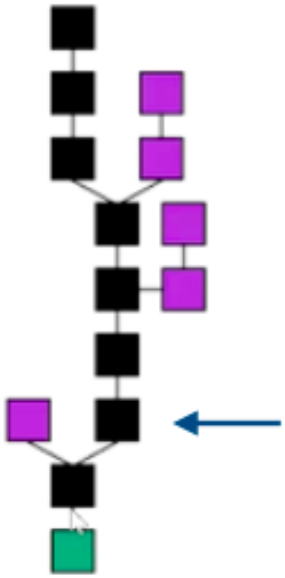

Figura 6: Esempio di blockchain.

Gli utenti sono considerati nodi di una rete peer-to-peer, ovvero rete in cui ogni pc fa sia da client che da server (contiene lui stesso delle informazioni da fornire ad altri pc).
La rete peer-to-peer osserva le proposte delle transazioni fatti da utenti, verificano che le transazioni siano valide (es che un "oggetto" appartenga ad una sola persona) e valutano se l'"oggetto" posseduto è già stato dato ad altri utenti.

I nodi della rete peer-to-peer devono eseguire un protocollo di consenso perchè i nodi potrebbero essere chiunque, anche degli impostori. Una volta raggiunto il consenso il blocco col consenso viene aggiunto in cima alla blockchain. Questo blocco viene poi memorizzato all'interno della copia della blockchain di ogni nodo. Il consenso è valido se almeno il 50% dei nodi è onesto.

Possono esserci dei casi in cui ci sono nodi che la comunità dice che sono entrambi validi ed entrambi collegati allo stesso nodo (es freccia blu nell'immagine sopra). Ad un certo punto qualcuno dei nodi sarà più veloce rispetto all'altro e quel ramo vincerà, gli altri saranno dei rami morti (nodi viola). Le transazioni all'interno dei nodi viola non sono più valide! Verranno considerate come proposte di transazioni da inserire all'interno degli altri nodi.

Pseudo-anonimato: banconote/criptovalute - quando ricevo una banconota non so chi l'ha ricevuta prima di me. La crittografia si occupa di anonimato e di mantenere visibili solamente alcune informazioni. Si parla di pseudo-anonimato perchè alcuni metodi riescono bene e altri no (qualcuno fa qualche errore, ad es pagare con bitcoin all'interno di un sito rivelando la propria identità).

Il **bitcoin** è una criptovaluta proposta nel 2008. Ci sono stati tanti tentativi di creare il sostituto digitale del denaro. Questo crea un problema chiamato *double spending*. Se creo una sequenza di bit per i 10€, posso copiare questa sequenza

di bit infinite volte e renderla indistinguibile dall'originale duplicando denaro. Il double spending si può evitare usando la banca: questa mi dà la banconota da 10€e se la voglio dare ad un'altra persona lo comunico alla banca che si segna il cambio di proprietà. La banca diventa un collo di bottiglia perchè tutte le richieste vanno a lei e viene a sapere di tutti i trasferimenti di denaro. Questo non piace ai crittografia.
Satoshi Nakamoto lavora nel 2008 ai bitcoin mettendo al posto di una banca un registro condiviso in cui scrivere tutte le transazioni di cambio proprietà dei soldi, in modo da rendere tutto controllato ed evitare il double spending.

Le proprietà memorizzate sulla blockchain di bitcoin sono bitcoin (BTC) oppure frazioni di essi.

Le transazioni sono parecchio complicate: le transazioni contengono transazioni in ingresso, campi particolari per generare nuovi bitcoin, script (scritti in un linguaggio particolare) di transazione in output.
Se Alice vuole mandare a Bob un bitcoin usando il proprio client (wallet - borsellino), specifica la quantità che vuole mandare e l'indirizzo (sequenza di numeri). L'indirizzo si ottiene ottenendo una chiave pubblica (da una copia chiave pubblica-chiave privata) da Bob che utilizza per cifrare il messaggio e a quel punto Bob usando la chiave privata può decifrare il messaggio.
L'indirizzo è una sorta di identità di Bob, ma lui può generare tutte le copie di chiavi pubbliche e private che vuole, quindi utilizzare ogni volta indirizzi diversi.
Bob, inoltre, può combinare le transazioni che riceve combinando i bitcoin.
Come fa Alice a dimostrare che è la proprietaria di questo bitcoin per mandarlo a Bob?
Semplicemente Alice firma digitalmente la transazione attraverso una chiave segreta. I nodi della rete peer-to-peer verificano che la firma di Alice è valida, verificano che Alice non ha mandato questo bitcoin ad altri e infine se tutto è corretto mettono la transazione nel prossimo blocco della blockchain.

Il fatto di possedere un bitcoin significa che posso avviare una transazione in cui posso inviare quel bitcoin ad un altro. Esistono solo le catetere di trasferimenti (transazioni) fatte a ciascun bitcoin e tutti può seguire queste transazioni per verificarne la correttezza.
Possedere qualcosa significa conoscerne la chiave segreta (corrispondente alla chiave pubblica) per firmare la transazione e renderla corretta.

I **miners** sono i blocchi della rete peer-to-peer. Esiste un pull di transazioni proposte dai vari client/wallet e i miners osservano le transazioni, ne scelgono 1 migliaio e cercano di creare un nuovo blocco. Essi dimostrano di aver fatto una certa quantità di computazione (proof-of-works) per avere loro il diritto di aggiungere il prossimo blocco alla catena.
Il puzzle crittografico viene reso via via più difficile se il tempo di computazione è troppo breve (meccanismo di autoregolazione).

Le **funzioni di hash** sono funzioni crittografiche che prendono in input una

sequenza di bit arbitrariamente lunga e ne produce una sequenza univoca di poche centinaia di bit.
Idea: prendo un file e ne calcolo l'hash e ne esce fuori un valore di x bit ottengo un valore univoco per quel file. Siccome il dominio è molto più grande del codominio non è possibile avere due sequenze diversi in input che producono lo stesso output. Se questo dovesse accadere si parla di collisione.
Un altro problema consiste nel trovare due input che producano lo stesso hash.

Proprietà funzione di hash:

- data un'impronta è difficilissimo trovare un input che produca quell'impronta

- anche se cambio un solo bit dell'input ottengo un output che è completamente diverso

Nella proof-of-work bisogna dimostrare di aver svolto una certa quantità di lavoro; per farlo il miner deve prendere il dato di cui deve calcolare l'hash, gli aggiunge una quantità casuale (nonce) e calcolo l'hash del mio dato concatenato alla quantità casuale e vado a vedere se il risultato ha un certo numero di bit significativi uguali a zero. In pratica sto riducendo il codominio riducendo il numero di output validi. (Diminuendo il numero di bit uguali a zero si rende, invece, più facile il lavoro). Se il miner ha successo allora ok, altrimenti spara a caso un altro nonce, etc...
Una volta che il miner ce l'ha fatta manda il blocco nella rete peer-to-peer e dice agli altri del nuovo blocco. Gli altri controllano la validità del blocco e delle sue transazioni, controllano la transazione di hash che abbia quel numero prefissato di bit iniziali uguali a zero e se tutto torna attaccano quel blocco all'ultimo blocco della blockchain.

Se contemporaneamente un altro mi dice che esiste un nuovo blocco ed è effettivamente valido, butto via le transazioni che compaiono anche nel nuovo blocco e, sapendo che l'hash dell'ultimo blocco dev'essere contenuto dell'intestazione del nuovo blocco, devo ricalcolare l'hash e metterla nell'intestazione del nuovo blocco.

Il momento in cui si possono creare dei nuovi bitcoin è solo nel momento in cui si aggiunge un nuovo blocco alla catena. Chi riesce ad agganciare un nuovo blocco ha nuovi bitcoins.

All'inizio si diceva che la blockchain era una rete decentralizzata su cui non si aveva il controllo da tutte le parti del mondo.

Ogni miner deve vincere la gara contro tutti gli altri per avere il diritto di mettere il proprio blocco in cima alla blockchain. Se riesco a possedere il controllo del 51% (¿50%) della potenza di calcolo di tutti i miners, allora vinco sempre io. Tutti contro tutti è molto inefficiente e si crea il problema dello spreco di energia. Motivo per cui si cercano algoritmi diversi più efficienti, ad es la ricerca del consenso. Chi mette più soldi ha più probabilità di essere scelto e di avanzare più di

altri.

Il fatto che ci sia un numero massimo di bitcoin creabili e che sia sempre più difficili (perchè riduci il codominio ogni volta con gli zeri iniziali) creare bitcoin, ha fatto paragonare i bitcoin all'oro (oro digitale).

Transazioni: Alice dà un bitcoin a Bob.
Alice deve creare due transazioni: una in cui a partire da (ad es 10 bitcoins) produce un output di 1 bitcoin verso Bob e un'altra in cui a partire dai suoi 10 bitcoins produce un input a se stessa di 9 bitcoins. L'importante è che la somma totale data in input non sia minore della somma totale data in output.

I miners non scelgono in modo casuale le transazioni ma le mettono in ordine in base a quelle con cui guadagnerebbero maggiormente quelli che hanno fee (=tassa) maggiore.

Se due blocchi arrivano simultaneamente potrebbe accadere perchè ad esempio negli USA si dice di aver trovato un nuovo blocco e l'informazione si propaga. Idem succede in Cina.
Allo stesso tempo in Italia un miner riceve sia il blocco dagli USA che il blocco dalla Cina. A questo punto li aggiungo entrambi e alla fine vincerà quello con la catena più lunga.

Problemi:

- transazioni fantasma con fee=0
- per essere sicuri che una transazione sia andata a buon fine bisogna aspettare 6 blocchi e se c'è un blocco ogni 10 minuti bisogna aspettare un'ora.

Esistono diversi tipi di blockchain. Esiste uno schema famoso per capire quale blockchain è più adatta a me e soprattutto capire se ho bisogno effettivamente di una blockchain.

Blockchain di tipo permission: i nodi si conoscono gli uni con gli altri ma non si fidano. I nodi sono in conflitto gli uni con gli altri, ovvero se le cose vanno male a uno ci guadagna un altro.
Siccome non c'è fiducia ogni cosa che si fa la si deve scrivere pubblicamente. Possono accederci solamente gli scrittori che ne fanno parte, quindi non può partecipare chiunque.
Esiste un distributore di credenziali che dicono se ho il diritto di scrivere o leggere informazioni sulla blockchain. Controlla l'amministratore.

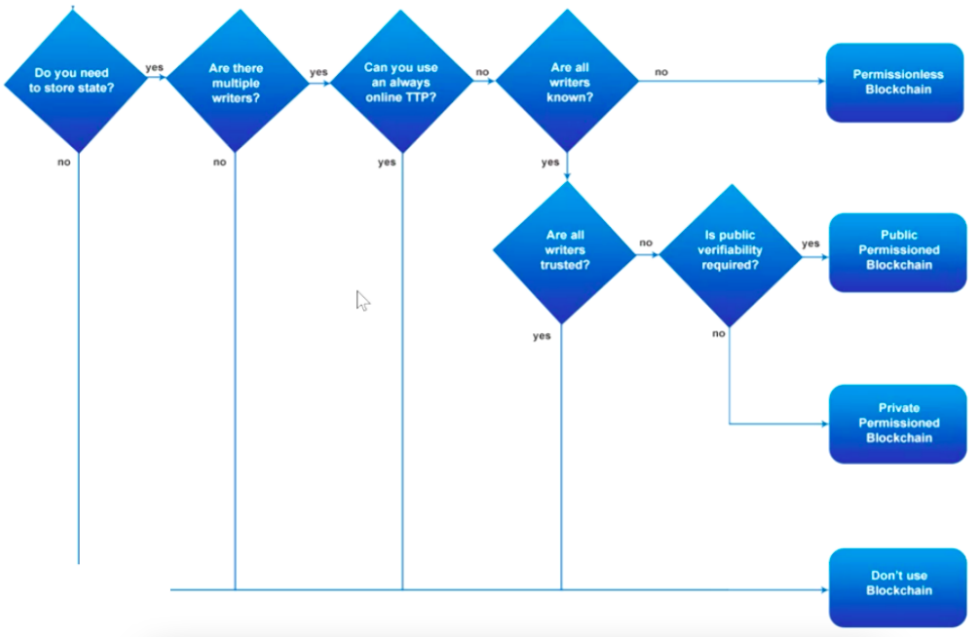

Figura 7: Immagine guida per scelta di utilizzare o meno una blockchain.

La blockchain consente di implementare politiche di trust, ovvero di fiducia dei partecipanti nei confronti di altri.
Il fatto che la blockchain sia inalterabile fa sì che io posso usarla come un servizio di notarizzazione dichiarando cose che non posso più smentire.

8 Introduzione a NoSQL

Frase chiave: lo spazio non è più un problema. La cosa importante è rispondere velocemente alle interrogazioni.

Aspetti positivi del modello relazionale:

- ben definito
- principio della minimizzazione
- tutto ciò che l'applicazione deve sapere è inserito nel database
- sono stati svolti 35 anni di ricerca sulla sicurezza, ottimizzazione e standardizzazione
- proprietà ACID
- è ben conosciuto
- ottima scelta per tantissime applicazioni
- porting dei dati molto pericoloso per la quantità di dati che risiede in un dbms

Limiti del modello relazionale:

- principio di minimizzazione + tutto ciò che serve è all'interno del database rende il modello relazionale molto stretto; è rallentata la velocità di interrogazione
- per ogni attributo ci può essere un solo valore, non sono ammessi array di valori
- i linguaggi di prgrammazione ragionano ad oggetti mentre il modello relazionale no
- non supportano comportamenti ciclici
- sono difficili da modificare perchè bisognerebbe alterare lo schema del db ed è difficile
- per garantire le proprietà ACID e per garantire il 2PC viene imposto un limite alla scalabilità

Esistono due tipi di problemi alla scalabilità:

- scale-up: ci si accorge che un pc portatile non è sufficiente per fare qualcosa che si vuole fare, allora si prova con un pc fisso e ci si accorge che non è ancora sufficiente. L'unico strumento che potrebbe essere utile è chiamato exadata
- scalabilità orizzontale, server infilato orizzontalmente in un armadio con almeno 2 alimentatori, ventole, CPU, etc... quando riempio questo server, ne inserisco un altro all'interno dell'armadio

8.1 NoSQL

- tutti i modelli non relazionali sono schema free o schemaless
- valgono le proprietà del CAP theorem
- modello base, basi arability, soft state, eventually consistence

8.1.1 Schema free o schemaless

Nel modello relazionale prima si definisce lo schema e poi si definiscono i dati. Se ad esempio bisogna aggiungere un dato al db è necessario prima modificare lo schema e poi aggiungere il dato mancante.
Nei sistemi NoSQL non esiste un modello fisso, ma il modello è inserito nei dati che vengono inseriti veramente. Si possono avere, ad esempio, dei modelli per cui per ogni "riga" si possono avere schemi diversi; questo comporta il rischio di inconsistenza di dati, ma vantaggi di sviluppo. Se quindi ci si accorge che manca un dato, lo si aggiunge, semplicemente, senza aggiungere dati NULL da altre parti.

Nota: se il dato non c'è non si scrive NULL, ma semplicemente non si mette il dato: lo schema è flessibile!

8.1.2 CAP Theorem

CAP è l'acronimo di Consistency, Avaiability, Partition tolerance.

- **Consistency**: tutti i nodi hanno gli stessi dati nello stesso istante
- **Avaiability**: impone la garanzia che ogni richiesta deve ricevere sempre una risposta, sia di successo che di fallimento
- **Partition tolerance**: il sistema deve continuare ad operare anche se ci sono fallimenti di parti del sistema

Il CAP Theorem, quindi, dice che: prendendo un sistema distribuito di nodi, si possono garantire contemporaneamente solamente 2 di queste 3 proprietà.

Tipicamente RDBMS tipicamente sono CA e NoSQL sono CP o AP.

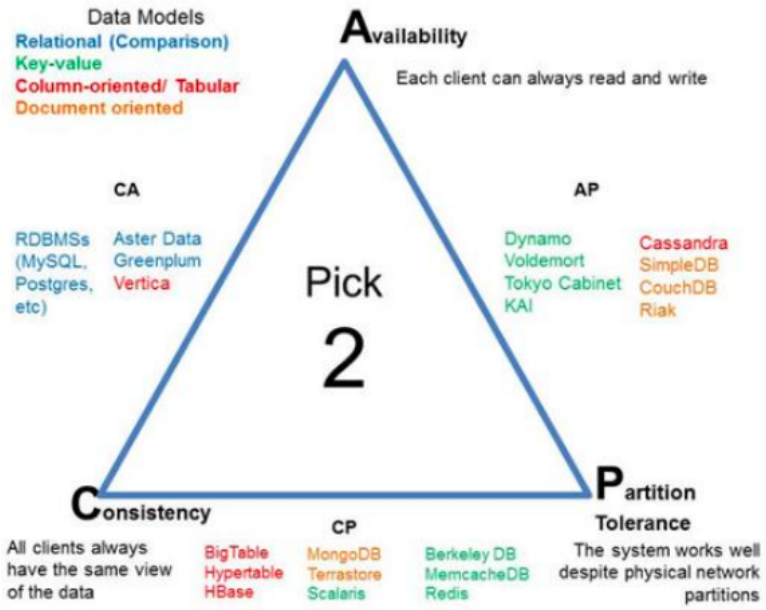

Figura 8: Schema del CAP Theorem.

8.1.3 BASE principale

L'acronimo BASE sta per Base Avaiability, Soft state, Eventual consistency.

- Base Avaiability: la risposta viene sempre data, anche se c'è consistenza parziale
- Soft state: si rinuncia alle proprietà ACID delle transazioni
- Eventual consistency: in un futuro i dati saranno allineati, non è garantita una sicurezza ma prima o poi il sistema arriverà a convergere

9 Document Based System

I modelli documentali sono object identified tramite un hash e i documenti sono memorizzati sotto forma di file .json.

Differenze col modello relazionale

Il modello documentale è un albero e l'accesso ai dati avviene partendo dalla radice e andando verso le foglie. È organizzato per "argomento" e il modo con cui è costruito è più simile a ciò che fa abitualmente l'essere umano quando organizza i dati.
Lo schema è flessibile ovvero, ad esempio, si possono avere due documenti della stessa collezione che contengono un numero diverso di attributi.
Si richiedono molti meno indici per poter effettuare le query migliorando le performance; questo può accadere perchè i documenti sono tutti annidati in uno solo. Questo comporta il rischio di replicare i dati, ma non importa perchè lo spazio non è più un problema.
Non esiste un singolo linguaggio di interrogazione ma ne esistono diversi in base al dbms che si sta utilizzando.

Il modello relazionale, invece, è piatto e non ha una tabella più importante rispetto alle altre.

9.1 MongoDB

MongoDB rientra nella categoria CP rispetto al CAP Theorem.
In MongoDB i dati sono memorizzati in un sistema chiamato binary json. Non è consentito effettuare join.

Notazioni:

RDBMS	Document based
Tabelle	Collezioni
Riga	Documento
Colonna	Chiave

Figura 9: Notazioni a confronto: database relazionali - document based.

9.2 Replicazione

La replica è realizzata tramite l'architettura master slave, ovvero tutte le scritture si fanno nel nodo primario e successivamente nel nodo secondario (master=primary, slave=secondary).

Inizialmente il nodo fa un sync prendendo i dati dal nodo primario (versione peggiorata dei dati) e poi prende dal file di log del nodo primario le ultime n operazioni e le riesegue. Questo è il meccanismo di utilizzo delle repliche.

La tolleranza (P del CAP Theorem) è garantita tramite il meccanismo di elezione.

Stati dei nodi

I nodi si possono trovare in diversi stadi:

- attivo: appena lanciato (non è in nessuna replica)
- primario: accetta operazioni di lettura (unico nodo nella replica)
- secondario: non si può scriverci sopra ma si ricevono delle copie dal dato primario (effettua solo la replicazione dei dati) - può essere promosso a primario
- recovering: recuperato a seguito di rollback
- startup2: entrato in un set
- arbitro: prende parte alle elezioni per passare a nodo primario
- down/offline: nodo irraggiungibile; se un primario è irraggiungibile gli altri nodi si mettono d'accordo per eleggere un nuovo primario
- rollback: durante le elezioni le scritture di interrompono

9.3 Elezioni

Ogni replica manda un bit a tutti gli elementi impostando un timeout di 10 secondi. Se non riceve risposta assume che il nodo è morto. Se è morto il nodo primario si indice un'elezione per eleggerne un altro. La priorità più alta ce l'ha chi sta in cima alle liste dei candidati, ed è lui che viene eletto come nuovo nodo primario (sarà il secondario con livello di priorità più alto).

Nel caso di partizionamento della rete può essere che ci sia un problema di riconciliazione tra il nodo primario eletto e gli altri nodi; in questo momento sono bloccate le scritture.

I nodi secondari con priorità zero sono nodi passivi, ovvero possono eleggere ma non essere eletti.

9.4 Operazioni di scrittura - writeConcern

writeConcern ha il compito di dire cosa fare durante le repliche.
Esistono 3 parametri fondamentali:

- w: numero di nodi in cui il dato dev'essere replicato prima di essere considerata conclusa l'operazione di replica:
  - $w = 0$, non c'è nessuna certezza dell'inserimento

  - $w = 1$, bisogna scrivere su un nodo primario
  - $w = n$, bisogna scrivere su almeno n nodi
  - $w = majority$, almeno la metà più uno dei nodi deve aver scritto prima di avere conferma dell'operazione di scrittura

- j: esprime l'intenzione di scrivere sul file di log prima che l'operazione sia stata eseguita

- $wtimeout$: è il tempo limite che bisogna aspettare nel caso in cui $w > 0$; se $wtimeout = 0$ non si attende la risposta prima di concludere l'operazione

9.5 Frammentazione

È garantita la scalabilità orizzontale.
Il meccanismo è dinamico, ovvero è MongoDB che si occupa di bilanciare automaticamente lo sharding.

Mongos è un componente software che rappresenta il punto di accesso alle query. Appena viene lanciato si va a leggere il **config server** che si basa su un file di configurazione che conosce perfettamente la struttura degli shard (=frammenti), ovvero va a vedere quali sono i nodi frammentati e quali sono quelli replicati (nella versione 4.0 ogni nodo distribuito è anche replicato).

Il **ruter**, quando riceve una query, fa una decomposizione della query in modo da capire dove sono i frammenti chiedendolo al config server e poi fa:

- dati su uno shard: target query

- dati su tutti i frammenti: broadcast query

sia gli shard che il config server sono replicati su *replica set* per garantire la disponibilità in caso di guasti e non avere un unico point of failure.

Il primary shard contiene tutto il database. Gli shard secondari contengono frammenti diversi e repliche di altri frammenti.

9.6 Operazioni di lettura - readConcern

Il readConcern è diviso in tre:

- local: se arriva una richiesta locale si fa una query locale (default). Questo crea problemi se l'interrogazione viene fatta nel nodo più vicino e quello stesso nodo non è primary

- avaiable: si vede se il dato è disponibile e consistente

- majority: quando la metà più uno dei nodi hanno ricevuto un ack in scrittura sul dato allora si può dare una risposta

9.7 Transazioni

L'uso dei lock è tutto gestito nel nodo primario:

- S: lock condiviso usato per le letture
- X: lock esclusivo usato per le scritture
- IS: intenzione di bloccare in modo condiviso uno dei nodi che discende dal nodo corrente
- IX: intenzione di bloccare in modo esclusivo uno dei nodi che discende dal nodo corrente

Sono garantite transizioni a livelli di collezioni. L'atomicità è quindi garantita a livello di collezioni e non più a livello di singole transizioni.
La gestione delle transizioni funziona solamente se i dati sono distribuiti in quanto il motore che le gestisce è distribuito.
Nella replica dei dati bisogna garantire che non ci siano scritture concorrenti su nodi diversi.

10 Graph database

In un database a grafo $G = (V, E)$ i concetti sono rappresentati tramite i nodi mentre le relazioni sono rappresentate tramite gli archi. I nodi sono in relazione tra loro.
Degli esempi di applicazioni che utilizzano database a grafo sono i social network, come ad esempio twitter.

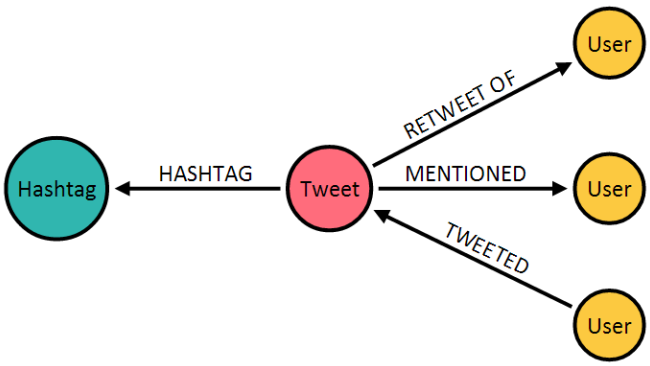

Figura 10: Esempio database a grafo con implementazione di un tweet di twitter.

I database a grafo sono particolamente utili per risolvere il problema del join bombing. Il join bombing è un fenomeno che si verifica quando in un RDBSM, per rappresentare un tipo di relazione, è necessario fare un numero esorbitante di join; un esempio di questa relazione è "amicizia", ovvero se si è interessati a trovare la relazione "amici degli amici".
Nei sistemi document based questo tipo di relazione può essere rappresentata

tramite meccanismi di embedded o tramite reference verso altri documenti. Il referencing, però, non evita i problemi del join bombing perchè farebbe una riproduzione di un modelli relazionale.

Un altro vantaggio del modello a grafo è l'immediatezza di leggere le relazioni in quanto non bisogna andare a cercare l'id nelle tabelle e poi "unirle" con dei join, ma si ha tutto visivamente disponibile da subito.

Property graph

Il modello a grafo segue il modello a "lavagna" che è simile ad una mappa concettuale.
Sia i nodi che le relazioni possono avere delle proprietà espresse in chiave-valore.

Anche per i database a grafo esistono tantissimi linguaggi di interrogazione. Quello utilizzato in questo corso è Neo4j con Cypher.

È possibile memorizzare i dati all'interno dei grafi in molteplici modi e poi in seguito processarli. Si possono memorizzare dati in un grafo tramite un processo basato su un modello relazionale. In questo modo si avrebbe un'interfaccia che simula il modello a grafo ma si interrogherebbe il grafo attraverso query SQL.

Nel caso di questo tipo di database non è detto che il risultato della query sia sempre un grafo.
Quando si interroga un modello relazionale, come risultato della query si ottiene ancora una tabella e la stessa cosa succede quando si interroga un modello documentale, ovvero che come risultato della query si ottiene ancora un modello documentale.
Se si prende RDF db a grafo come esempio, quando lo si interroga, si ottiene come risultato una tabella.

Graph database

I database a grafo devono supportare le operazioni CRUD (Create, Read, Update, Delete).
Le interrogazioni sono degli attraversamenti di un grafo a partire dalla radice e arrivando fino alle foglie tramite gli archi; questo è l'equivalenza coi join di un DBMS. Questi sono più performanti rispetto ai rdbms perchè non fanno join.

Esistono due tipi di db a grafo:

- native graph storage: sono ottimizzati e progettati per la progettazione e gestione dei grafi
- non-native graph storage: trasformano i dati dal formato a grafo al formato:
  - relazionale
  - object oriented database

– altri

I grafi non scalano bene, ovvero non possono essere facilmente frammentati. Se si partiziona un grafo i due sottografi hanno per forza degli archi tagliati e quindi si andranno a perdere delle relazioni. È a questo che è dovuta l'esistenza dei grafi non nativi, cioè per garantire la trasformazione da modello a grafo ad altri modelli per poi poter frammentare i dati.

Native graph storage

I grafi di tipo nativo memorizzano i dati in modo intelligente. Si cercano di memorizzare i dati mettendo i nodi graficamente vicini con i propri archi, in modo tale che per seguire un cammino è necessario seguire semplicemente l'ordine in cui sono stati memorizzati i dati.

Neo4j è molto performante perchè conosce tutti i riferimenti in memoria tramite puntatori ad aree di memoria che contengono altri nodi. Questo meccanismo, però, è poco performante in scrittura, infatti Neo4j è lentissimo a scrivere/caricare i dati ma velocissimo a leggerli.

10.1 Neo4j

Garantisce le proprietà ACID con locking, consente disponibilità e sulla scalabilità garantisce l'alta disponibilità.

È possibile definire un cluster di nodi e alcuni di essi sono nodi per cui tutte le scritture sono fatte in contemporanea, ovvero questi nodi sono sincronizzati. Esistono *replica server*, ovvero server di sola lettura che ricevono dati con meccanismi di replica che possono essere utili per più attività specifiche.

L'interrogazione normalmente viene eseguita sul nodo più vicino e non viene distribuita.

10.2 ArangoDB

In un'applicazione può succedere che i dati vengono interrogati/archiviati tramite diversi modelli di database. In questi casi c'è bisogno di avere più conoscenze e viene quindi in aiuto ArangoDB che introduce il sistema di database poliglotta.

È un DBMS che consente di memorizzare i dati in formato document based potendoci associare anche una chiave-valore, gestisce inoltre i modelli a grafo e la ricerca full text. Esso è un modello poliglotta perchè è in grado di parlare più linguaggi, ovvero posso memorizzare i dati in .json e fare l'interrogazione come se si stesse lavorando su un graph db, ma può anche interrogare un grafo in documenti.

ArangoDB offre diverse modalità di scalabilità:

- single instance: funzione in locale e non fa repliche

- master/slave: scritture sul master e poi meccanismi di replica sullo slave)
- active failover
- cluster
- multiple datacenter

Active failover

Se nell'architettura master-slave il master fallisce, lo slave non può essere sostituito.
Esistono istanze chiamate leader (per scrittura e lettura) e nodi di tipo single server in modalità follower.
Il leader invia al follower operazioni di write, ovvero il leader prima di salvare su disco scrive sul file di log (replica).
L'agency, poi, va a vedere le configurazioni del leader e del follower: se si rende conto che un leader non è più disponibile, promuove il primo follower a leader (elezione).

Cluster

Si ha più di un coordinatore, i quali si connettono con i client, cioè: i client fanno query ai coordinatori, i coordinatori fanno le query sui DBMS server e gli agent si occupano della sincronizzazione dei dati e della elezione di un nuovo coordinatore.
Se un db server di tipo slave fallisce non c'è problema, se invece cade un coordinatore c'è un meccanismo di elezione per definire un nuovo coordinatore.

In questo caso si applica il protocollo ROWA (Read Once Write All), ovvero il coordinatore deve scrivere su tutti i nodi.
Con il protocollo one shard, invece, il coordinatore scrive solo sul nodo in cui c'è la copia primaria e poi, attraverso le repliche, si scrive anche sugli altri nodi.

11 Key Value Store

È la formulazione più semplice dei modelli chiave-valore. È possibile accedere ad un valore conoscendo la sua chiave. È possibile associare un tipo al valore. È possibile utilizzare funzioni di hash per accedere ai dati rapidamente aggiungendo indici. La funzione di hash limita le collisioni perchè è implementata insieme ai bucket.

I key value store sono usati per tenere in memoria grandi quantità di dati, infatti sono detti sistemi in memory.

Notazione:

RDBMS	Key-value
Database instance	Riak cluster
Tabella	Backet
Riga	Key-value
Chiave	Key

Figura 11: Notazioni a confronto: database relazionale - key-value.

In generale, data una rete, si privilegia sempre lo **scale-out**: si preferisce sempre avere una macchina più piccola replicata più volte piuttosto di una molto grande. Il vantaggio dello scale-out è lo spazio che possiamo pensare tenda all'infinito: con una singola macchina grande lo spazio prima o poi finisce.

Le operazioni con questo modello sono molto limitate:

- inserimento coppia
- cancellazione coppia
- modifica coppia
- trovare un value associato a una key

11.1 Redis

Redis è un tipo di key-value store creato da Salvatore Sanfilippo nel 2009 perchè si accorse che MySQL era troppo lento per una soluzione real time web analytic a cui stava lavorando.

Risiede totalmente in memoria centrale, quindi è più rapido prendere i dati.

Redis può gestire una chiave (stringa in ascii) e i valori primitivi sono le stringhe. Non è sempre possibile fare una ricerca per valore.

Sharding

I problemi, principalmente, sono due:

- come avviene la policy, cioè come si possono dividere i dati che sono organizzati per chiavi: si può fare il partizionamento delle chiavi in modo automatico oppure creare una funzione di cluster
- dove sono distribuiti i dati: di default lo sharing è gestito con un meccanismo incrementale, ovvero quando la macchina si riempie si crea un nuovo shard, oppure si può stabilire a priori in quale nodo mettere i dati

Alta disponibilità

1. Redis Replica Of Setup: esistono nodi su due cluster, ovvero si scrive solo su un cluster e poi si sincronizzano i dati sul secondo cluster (soluzione master/slave)

2. Acive-Active Setup: si può scrivere e leggere su più cluster: sul cluster in cui si hanno i nodi e le loro repliche e sul cluster in cui si hanno ancora i nodi e le loro repliche. Ci sono poi soluzioni che permettono di aggiornare i dati (protocolli ROWA)

Redis ha la possibilità di rendere persistenti i dati, ma il suo punto di forza è costruire sistemi che gestiscono tantissimi dati in memory.

12 Wide Column Store

Il modello wide column store è propriamente chiamato modello colonnare.

Non è una evoluzione del modello key-value e l'idea è che accanto alla chiave si ha una riga che contiene degli attributi monovalore.
Si posiziona a metà strada tra il key-value e il modello documentale; si può, quindi, interrogare ma è meno strutturato.

Alcuni esempi di wide column store sono:

- bigTable

- Hbase

- Cassandra

12.1 bigTable

bigTable è una mappa multidimensionale, ordinata, persistente e sparsa.
È indicizzata da 3 informazioni:

- chiave riga

- colonna della riga

- timestamp

Con bigTable si può garantire implicitamente un controllo della concorrenza bassato sul multiversioning tramite le proprietà ACID delle transazioni.
Dato un insieme di righe si ha una chiave, una column family con diversi qualifier e, per una particolare chiave, un column family e qualifier con valori diversi in base al timestamp. L'utilizzo dei timestamp è un metodo molto efficiente per gestire il versioning di un sistema: per accedere a un certo dato, infatti, si deve conoscere la column family, il timestamp e la key.

In una tabella, pur avendo le stesse column family, si può associare un numero di qualifier variabili; questo rappresenta la variabilità dello schema.

I modelli colonnari sono molto vicini ai modelli relazionali, ma a differenza di questi non sono pensati per fare join su valori perchè tutto ciò di cui si ha bisogno è dentro la colonna. Non si possono rappresentare array di valori, ma si può, ad esempio, avere una column family con tanti numeri di telefono divisi in tante colonne.
Il modello colonnare viene spesso utilizzato per gestire modelli più complicati come i grafi.

12.2 Hbase

Hbase è la versione distribuita dei bigTable.
Ogni table è composta da colonne organizzate in column-families (che possono avere un numero variabile di colonne, qualifier in bigTable).

Il consiglio è quello di fare pochi join e di denormalizzare i dati, ovvero replicarli su più tabelle e poi fare join su tabelle di dimensioni minori.
Lo schema è caratterizzato solo dalle column-families. È anch'esso un multiversioned, tramite il quale riesce a garantire uno storico dei dati, e una mappa sparsa.

L'architettura è un'architettura master/slave.

Componenti

I suoi componenti sono:

- region: sottoinsieme delle righe partizionate orizzontalmente
- region server: gestisce un insieme di righe di una o più tabelle e la sincronizzazione avviene tramite i file di log
- master: coordina gli slave, assegna le regioni, riconosce se uno slave non funziona più, ...

Architettura distribuita

L'architettura è un'architettura distribuita; HDFS si occupa di fare la replica dei dati, trasferire i dati e sincronizzare i dati.

La memorizzazione dei dati avviene in modo orizzontale: prima scrivo sul file di log e poi scrivo sul disco vero e proprio, quindi è garantita la ricostruibilità dei dati.

ZooKeeper

Si occupa di coordinare il master con i region nelle operazioni di scrittura e lettura ed è HDFS che tiene allineati i dati.

12.3 Cassandra

Cassandra è un possibile esempio di modelli colonnari, è l'evoluzione open source del database NoSql utilizzato da facebook.
Column families: insieme di chiave-valore definita all'interno dei key spaces e ogni column families ha un row id. Il column family è una tabella.
Riga: collezione di colonne con un nome.
Ogni riga contiene almeno sempre una colonna in modo obbligatorio.
Il linguaggio di interrogazione di Cassandra è CQL-Cassandra Query Language.

Durante la fase di scrittura dei dati si può scrivere su un certo numero di nodi e attendere che questi diano l'ok; si può imporre la scrittura in tutti i nodi ottenendo così la consistenza in stile relazionale.
Si può chiedere un certo numero di repliche dovendo aspettare che i nodi diano l'ok.
Cassandra ha un sistema di aggiornamento interno, potrebbe tuttavia metterci molto tempo (keep in mind the eventualy consistency).

Cassandra ha un'architettura peer-to-peer distribuita.
Cassandra garantisce l'elasticità in modo trasparente e un'alta disponibilità.

Cassandra parte da un modello chiave-valore.
All'interno ha una struttura (spazio di indirizzamento) ad anello in cui ci sono dei nodi che garantiscono delle regole di accesso ai dati tramite valori di hash.

Esempio: se inserisco una funzione di hash compresa tra 0 e A allora il valore corrispondente andrà nel nodo A, se inserisco una funzione di hash compresa tra B e C il valore corrispondente andrà nel nodo C. Se invece voglio aggiungere un nodo D i valori tra B e C verranno modificati in modo che i valori da D a B vanno su B e quelli da D a C vanno su C.
Posso aggiungere quanti nodi voglio e la crescita (aggiunta di nodi) ha modifiche solo a livello locale (modifico solo il nodo precedente e successivo).
La replica viene fatta solamente su 3 nodi: se ad esempio voglio replicare A la replica andrà fatta solo su F e B perchè se uno di questi fallisce (es il nodo A), il nodo E si accorge e chiede ad F di copiare tutti i dati di A prendendone il suo posto.

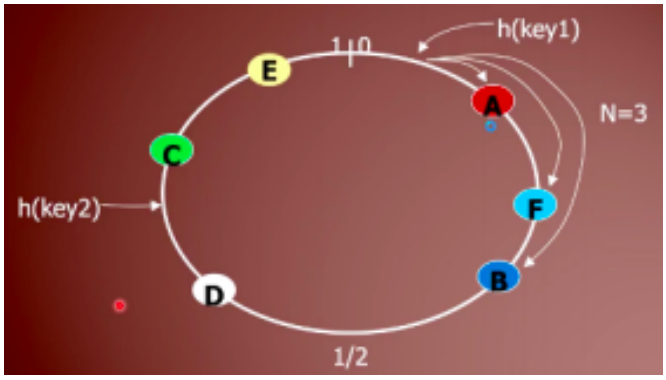

Figura 12: Esempio funzionamento Cassandra.

Protocolli di gossip

Ogni nodo chiede al successivo se va tutto bene fino a quando non riesce ad avere la comunicazione completa per capire la situazione.

Consistenza

La consistenza dei dati è offerta tramite operazioni di lettura e scrittura. La lettura e la scrittura sono consistenti e date per certe se:

- almeno un nodo mi risponde; consistency one
- tutti i nodi mi rispondono
- la maggioranza mi risponde: consistency quorum

La maggioranza è data dal fattore di replica. Ad esempio: se il fattore di replica è 3, il quorum (maggioranza) è 2, quindi basta che 2 elementi mi rispondono.

Operazioni di cancellazione

Le operazioni di cancellazione rendono semplicemente il dato non disponibile invece che cancellarlo. Questo crea problemi di spazio nelle tabelle fisiche: per questo, periodicamente, si fanno dei merge (compattazione) tra i dati per ridurre lo spazio dei dati utili.

Le operazioni di lettura (read) si svolgono nello stesso modo delle scritture (write). Si può utilizzare un graph DB memorizzando internamente i dati, ad es, in modello wide column.

13 Data integration

Data integration è un processo diverso rispetto a data enrichment.

Data integration

Dati due diversi insiemi di dati che possono avere anche attributi diversi, il processo di integrazione porterà ad avere un unico insieme che conterrà tutti gli elementi del primo e del secondo insieme. Questo equivale a fare un full outer join.

A	M
1	m
2	n
4	o

A	N
2	p
3	q
5	r

$\Longrightarrow$

A	M	A	N
1	m	-	-
2	n	-	p
4	o	-	-
-	-	3	q
-	-	5	r

Figura 13: Esempio full outer join.

Data enrichment

Dati due insiemi diversi di dati, si considerano solamente gli elementi comuni ad entrambe le tabelle e si aggiungono informazioni. Questo equivale a fare un left outer join.

A	M
1	m
2	n
4	o

A	N
2	p
3	q
5	r

$\Longrightarrow$

A	M	A	N
1	m	-	-
2	n	2	p
4	o	-	-

Figura 14: Esempio left outer join.

Esistono due tipi di integrazione:

- spaziale
- temporale: integrazione al variare del tempo (ad esempio andamento azioni di borsa)

Può succedere che in due dataset con diversi schemi, ci sia lo stesso contenuto informativo rappresentato in modo diverso.
Il problema maggiore si ha quando si hanno due dataset che sono rappresentati con due modelli diversi. L'obiettivo in questo caso è capire in che formato si vuole ottenere il risultato del dataset integrato.

13.1 Eterogeneità

Le eterogeneità/conflitti sono tipicamente legate a:

- nome, come sono etichettate le cose
- tipo, come è modellato un pezzo della realtà

Eterogeneità - nome

Le eterogeneità legate al nome sono di diversi tipi:

- sinonimia: rappresento lo stesso concetto in modo diverso
- omonimi: termini usati per descrivere dei concetti che hanno la stessa stringa ma semantiche diverse (ad esempio città inteso come città di residenza in una tabella A e città di nascita in una tabella B)
- iperonimie: utilizzo un concetto più di alto livello rispetto ad altri (ad esempio persona e uomo, donna)

Eterogeneità - tipo

Le eterogeneità legate al tipo possono presentare problematiche diverse:

- domini diversi per gli stessi attributi in schemi diversi (ad esempio città come italiane e francesi)
- attributo in uno schema e un suo valore derivato in un altro schema
- in uno schema un concetto è rappresentato come un'entità e nell'altro come un attributo
- gerarchia di generalizzazione (es persona, uomo, donna)
- in ER/grafo posso avere un concetto come nodo-entità e in un altro schema posso avre lo stesso concetto rappresento come relazione-arco
- differenze di granularità, cardinalià
- conflitti di chiave (es chiave n matricola e nell'altro schema chiave cf)

Fasi di risoluzione eterogeneità

1. trasformazione degli schemi nello schema target
2. controllare le corrispondenze tra gli schemi, ovvero gli attributi che compaiono in entrambi gli schemi
3. definire lo schema integrato, ovvero l'unione degli attributi definiti attraverso le regole di mapping

Nota: se da una parte si hanno due colonne Nome, Cognome e dall'altra parte una sola colonna Nome cognome, conviene definire nella tabella target integrata il nome e il cognome in un'unica colonna perchè può essere molto difficile distinguere il nome dal cognome per separarli in due colonne diverse (ad esempio per nomi stranieri).

13.2 Strategie

Si possono applicare diverse strategie di integrazione:

- approccio binario: prendo uno schema, integrarlo con un secondo schema e il risultato lo integro con un terzo schema, etc...
- approccio bilanciato: integro due schemi (ad esempio perchè sono vicini e simili), integro altri due schemi e i due risultati li integro insieme
- approccio n-ario: integro tutti gli schemi insieme (da evitare)
- meccanismi misti: come il bilanciato ma in un caso ne integro 2, nell'altro 3, etc...

13.2.1 Semantic relativism

Lo stesso concetto può essere rappresentato in modo diverso. Non esiste la soluzione giusta, tutto dipende da cosa è necessario per il lavoro che si deve fare.

Il primo passo quando ci si trova davanti a due schemi diversi è stabilire, dopo averli trasformati nello schema target (modello in cui si vuole ottenere lo schema risultato, ad esempio json, relazionale, etc...), quali attributi matchano, ovvero quali attributi sono rappresentati (soprattutto in modo diverso) nei due diversi schemi. Questo si chiama mapping.

13.3 Conflitti

Descrizione dei conflitti:

- nomi: devo trovare il modo di rappresentare questi concetti
- attributi diversi che descrivono le stesse persone in modo diverso

Conflitti strutturali: oggetto rappresentato da una parte come un'entità e dall'altra come un attributo.
Conflitti frammentati: tabella divisa in due tabelle e nella soluzione si ha la necessità di unire queste due tabelle. Queso tipo di conflitto è quello che dà valore al fatto che si stanno integrando dei dati. Le tabelle si integrano sia per aggiungere informazioni ai dati e sia per unire insieme i dati.

13.4 Mapping

Quando si integrano due insiemi di dati non ci si aspetta che questi siano completamente disgiunti e ci si aspetta che può capitare che uno stesso oggetto sia descritto in modo diverso tra i due schemi.

Instance level conflicts

Gli instance level conflicts sono conflitti del tipo:

- i valori di chiave sono diversi tra le tabelle che voglio integrare (ad esempio da una parte si ha matricola e dall'altra si ha codice fiscale)
- si hanno due valori diversi per lo stesso attributo semanticamente uguale (ad esempio in una tabella john guadagna 1000€e nell'altra guadagna 2000€). In questo caso non si sa effettivamente quale sia il valore dell'attributo corretto da considerare

13.5 Funzione di risoluzione dei conflitti

Dati due valori in conflitto si cerca di restituire il valore più probabile.
Possono esserci più possibilità:

- si guarda/studiano le due sorgenti e si stabilisce se una è più affidabile dell'altra
- funzioni matematiche (minimo, massimo, media, varianza) con la consapevolezza di introdurre potenziali errori (ad esempio in caso di conflitto prendi lo stipendio minimo introduce un errore in quanto si sta sottostimando il dato; non si può sapere se si sta sbagliando oppure no)

In questi casi non esiste LA risposta giusta.

Deduplication: processo in cui si può scoprire se ci sono concetti del mondo reale duplicati nella mia tabella. In questo caso si considera solo una tabella quindi non si ha schema matching. Lo schema matching si ha in caso di integrazione con due tabelle diverse.

Esempio: Siano "Crlo" e "Carlo" due nomi di due righe diverse della stessa tabella. Questi rappresentano la stessa cosa semanticamente ma sintatticamente no; tramite la distanza di edit ci si accorge che l'errore sintattico tra queste due parole è di 1 lettera, quindi si può dire che probabilmente rappresentano lo stesso oggetto del mondo reale.
Per gli indirizzi non si può fare questo tipo di discorso in quanto ad esempio "via giuseppe bologna 16" e "via giuseppe bologna 18" sono due indirizzi diversi, quindi non rappresentano lo stesso oggetto della realtà, sebbene la loro distanza di edit sia pari a 1.

Riduzione dello spazio di ricerca

Per andare a cercare tutti i valori duplicati (deduplica) nella stessa tabella, considerando che le tabelle possono avere tantissimi valori, è necessario ridurre lo spazio di ricerca; in questo modo si hanno meno confronti da fare.

Il blocking method, ad esempio, è un metodo di riduzione dello spazio di ricerca perchè considera solamente colonne significative per effettuare i confronti. Applicando un modello di decisione si possono ottenere:

- match
- possible match
- unmatch

Steps of probabilistic record linkage

1. preprocessing
2. blocking
3. funzione di comparazione delle stringhe
4. costruisco un campione per vedere se ho comparato correttamente
5. decido match o unmatch

Metodo di Hernandez 1995: si prende una chiave e un attributo che si suppone essere più caratterizzante, si ordina la tabella sulla base di questo attributo e si confrontano gli elementi.

Nota: per il confronto tra date la prima domanda che ci si deve fare è se si sta guardando una data italiana o inglese. Poi si può pensare che sia più affidabile il mese piuttosto che il giorno perchè è più facile sbagliare a scrivere il giorno.

14 Data integration - architecture

Data integration ha l'obiettivo di consentire all'utente di accedere in modo univoco a dati eterogenei e autonomi.
Ha il vantaggio di avere la capacità di gestire eterogeneità di schemi superiori rispetto alle architetture federate e gestire eterogeneità a livello di istanze. Si ha un livello di autonomia un po' più basso perchè gli schemi possono venire da più posti (es locale, web, etc...).

14.1 Approcci alla data integrazione

- integrated read-only views: i dati vengono integrati solo per poterli leggere
- integrated read-write views: possibilità anche di scrivere i dati, e quindi poter fare degli update (non è trattata nel corso)

Data integration è il problema di combinare dati su sorgenti diversi e che forniscano all'utente una versione unificata (GS - Global Schema).

Elementi data integration architecture:

- schema globale
- schemi locali
- mappings

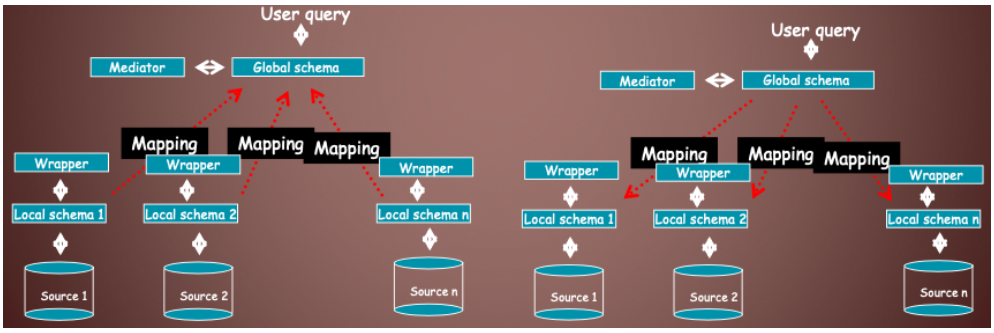

Figura 15: Architettura di data integration.

Wrapper

Wrapper:

- agganciano lo schema locale con lo schema globale
- rappresentano la sorgente che wrappano come una sorgente compatibile
- ricevono in ingresso le query nel linguaggio dello schema globale e restituiscono il risultato

Mediators

Mediator:

- comprendono una query
- riscrivono la query per poter eseguire interrogazioni sui nodi locali grazie ai wrapper
- mettono insieme i dati e li integrano
- risolvono le eterogeneità

L'idea è combinare le informazioni in modo che l'informazione interrogata è come se fosse in un unico sistema.

Il mediator, a partire da uno schema globale, trasforma l'interrogazione scritta dall'utente in un insieme di sottoquery ognuno per sorgenti coinvolte nell'interrogazione.

Riconciliazione delle istanze: una volta che si è effettuata la parte "discendente" dalla query dell'utente alla query locale, i db locali forniscono delle risposte

che vanno messe insieme.
Bisogna evitare la replicazione dei dati e bisogna evitare l'inconsistenza, altrimenti si trasporta un problema di qualità dei dati.

L'elemento fondamentale è il tempo perchè tutto ciò va fatto velocemente.
Due attività/fasi:

1. design-time: il mediator deve o costruire lo schema globale, oppure deve collegare i mapping e costruire lo schema locale (lo schema globale e i mapping devono essere costruiti a partire dagli schemi locali)

2. querying-time: il mediator deve eseguire la query velocemente (dato lo schema globale e dati i mapping si effettua la query)

I ruoli del mediator sono, quindi:

- riformulare interrogazioni

- ottimizzatore

- execution engine (=motore di esecuzione) utilizzando i mapping

Il mapping è l'elemento fondamentale, è costruito a design time ed è collegato a run-time. Il mapping è costruito in base a come si devide di rappresentare lo schema globale.
Lo schema globale è un db virtuale. Qualunque schema si va a costruire sarà una rappresentazione solo virtuale e bisogna solo decidere come associare ogni punto dello schema globale agli schemi locali.
Approcci per la costruzione di mapping:

- GAV

- LAV

- GLAV

Il mapping spiega qual è la relazione tra schema globale e locale rispetto al concetto di vista. La vista è un risultato di una query che costruisce una tabella virtuale.

14.2 GAV - Global as view

Assume che lo schema globale non è nient'altro che una vista degli schemi locali.
Si vogliono considerare tutti i data source contemporaneamente, quindi si ottiene l'unione di tutti gli schemi concettuali che si hanno a disposizione.
Ogni tabella dello schema globale è una vista degli schemi locali.

Il verso del mapping sarà dallo schema locale allo schema globale.
Questo ha il vantaggio di prendere tutti gli elementi disponibili dalle sorgenti, di conseguenza bisogna solo fare un arricchimento dei dati.

Questi sistemi hanno performance basse.
Il mapping dice come andare a trovare i dati da una sorgente reale al posto di quelli virtuali che si stanno considerando.

14.3 LAV - Local as view

Si vuole costruire uno schema globale (che si conosce già) come idea di progetto, quindi si definisce il modello di rappresentazione, e dopo di che, si va a vedere quali sono i pezzi dell'informazione che servono e che sono presenti negli schemi locali.
Può succedere che un concetto dello schema locale non è rappresentato nello schema globale; questo non è importante. Per l'approccio LAV è importante il viceversa, ovvero che ogni concetto del globale è rappresentato in qualche locale.
Il verso del mapping sarà dallo schema globale agli schemi locali.

Bisogna definire un mapping per capire quali concetti del globale possono essere mappati nel locale.
Cambia l'approccio temporale, ovvero se si aggiunge una sorgente locale lo schema globale non cambia perchè è fisso e definito dal problema.

L'approccio LAV dice come dati reali possono contribuire alla costruzione dello schema locale.

14.4 GLAV - Global and Local as view

Si ha un mix, ovvero alcuni concetti sono costruiti come approccio GAV e altri come approccio LAV. Questo approccio è meno frequente.

Recap

Approccio GAV: se gli schemi locali non sono sovrapponibili allora si costruisce lo schema globale facendo l'unione delle due tabelle locali: operazione di unfolding.
Tutti gli elementi dello schema della sorgente locale sono mappati nello schema locale.

$$S_1, S_2 \to S_3 = S_1 \bigcup S_2 \to \text{query su } S_3$$

Approccio LAV: il mediator esegue la query su entrambi gli schemi locali uniti e restituisce i risultati locali. Si basa sull'idea che il contenuto di ogni sorgente locale sia caratterizzato in termini della vista dello schema globale, quindi si ha una progettazione a priori dello schema globale con gestione complicata delle query. Supporta la facilità di estendere il modello.

$$\text{Query su } S_1 \text{ e } S_2 \to \text{unisco il risultato}$$

15 Data quality

Il problema della qualità dei dati nasce dal fatto che i dati sono una rappresentazione della realtà e non sempre la rispecchiano perfettamente. Un esempio può essere una fotografia della strada quando c'è nebbia: questa non rispecchia la realtà perchè nasconde dei dettagli.

Si considera un dato di buona qualità se questo è adatto all'uso che se ne vuole fare.
La qualità di un dato è negli occhi di chi lo usa e non nelle mani di chi lo produce - Cit Maurino

La qualità è la caratteristica di un artefatto che soddisfa necessità o aspettative implicite o esplicite. Essa è rappresentata da dimensioni e sottodimensioni, ovvero caratteristiche particolari che descrivono la qualità dell'informazione associata al dato. Le dimensioni possono essere misurabili o non misurabili; esse possono essere misurate attraverso delle metriche.
Le metriche sono un insieme di procedure che consentono di misurare un determinato valore. Si dividono in:

- misurazioni di procedure (metodi)
- unità di misura

15.1 Dimensioni di qualità dei dati

Le misure di qualità dei datipossono essere:

- misure oggettive (ad es metriche)
- misure soggettive (ad es questionari che dipendono da percezioni umane)

Ci concentriamo principalmente sull'esaminare misure oggettive, ovvero:

- accuratezza
- completezza
- dimensione temporale
- consistenza dei dati
- problemi di tradeoffs

15.1.1 Accuratezza

L'accuratezza impatta su valori di tipo alfanumerico e si può applicare sia a tuple che a relazioni.
L'accuratezza è la vicinanza tra il valore presente nel dataset e il valore considerato come valore vero.
L'accuratezza può essere di due tipi:

- accuratezza sintattica, ovvero studia se del valore/stringa presente in un attributo di una tupla è presente in un reference di valori di riferimento considerati corretti

- accuratezza semantica, la quale è molto difficile da verificare perchè sarebbe necessario avere una sorgente considerata vera da paragonare con i dati del dataset per capire se questi ultimi sono accurati

La funzione distanza è una metrica per misurare l'accuratezza sintattica e serve a comprendere la correttezza sintattica di un valore del dataset. Questa funzione si applica tra due valori: il valore presente nel dataset e un valore di riferimento.
Ad esempio se in un dataset ho un nome Carl come faccio a capire se è Carlo o Carla? Considero come valore di riferimento la distribuzione dei nomi italiani; da qui vedo che per il 42% è utilizzato il nome Carla, mentre per il 58% è utilizzato il nome Carlo. Stando a queste percentuali posso presupporre che Carl voglia dire Carlo, ma non ne ho la certezza assoluta.
Se si ha un problema del genere per un anno di nascita, invece, non si può dire niente in quanto non si hanno valori di riferimento.

Le metriche per la funzione di distanza (distanza di edit - ED) possono essere applicate sia sulle stringhe (es "Architetture dati") sia sui singoli token (es "Architetture", "dati").
La distanza di edit misura la distanza tra due stringhe/token. Nell'esempio precedente si ha $ED(Carl, Carlo) = 1$ perchè differiscono per una sola lettera.
La distanza di edit funziona solo per misurare l'accuratezza sintattica; nel caso dell'accuratezza semantica questa funzione non si può utilizzare. Ad esempio al sud il nome Domenico è abbreviato con il diminutivo Mimmo. Questi due nomi hanno $ED > 1$ sebbene abbiano lo stesso significato semantico.

Dato n il numero di simboli totali tra le due stringhe/token prese in esame, la distanza di edit può essere normalizzata nel seguente modo:

$$ED_{norm}(s_1, s_2) = 1 - ED(s_1, s_2)/n$$

in modo che:

- $ED = 0$ le due stringhe/token sono diverse

- $ED = 1$ le due stringhe/token sono identiche

Esempio: $ED_{norm}(Mario, Carlo) = 1 - ED(Mario, Carlo)/n = 1 - 2/10 = 1 - 0.2 = 0.8$

La distanza di edit normalizzata non misura, quindi, la distanza tra due stringhe/token ma il suo complemento a 1, cioè l'accuratezza di s_1 rispetto a s_2: accuratezza=1 corrisponde a distanza=0.

Le funzioni di edit funzionano bene se si stanno considerando dei nomi, ma se si vanno a considerare indirizzi, ad esempio, bisogna ragionare in modo diverso. Ad esempio Piazza della Scienza 4 è un indirizzo diverso rispetto a Piazza della Scienza 2, sebbene abbiano distanza di edit pari a 1.

15.1.2 Completezza

La completezza misura il grado di copertura di un fenomeno osservato nell'insieme di dati che si hanno a disposizione. Essa può essere di:

- tupla, ovvero denota la presenza di valori nulli nella tupla
- attributo, ovvero denota la presenta di valori nulli in un attributo/colonna
- tabella, ovvero denota la presenza di valori non nulli in una tabella

Importanti per il concetto di completezza sono anche le ipotesi di mondo chiuso e mondo aperto.
Ipotesi di mondo chiuso: tutto ciò che non è rappresentato nel database non esiste.
Ipotesi di mondo aperto: viene registrato solo quello che si conosce; di quello che non si conosce non si può dire niente. Ne consegue che gli oggetti rappresentabili possono essere di più di quelli che sono rappresentati.

La completezza di un oggetto rappresenta il numero di valori non nulli in riferimento a tutti i valori rappresentati nella tabella.

Esempio: se rappresento solamente 3 impiegati su 4 la completezza è del 75%.

Cognome	**Età**	**Città di nascita**
Rossi	null	null
Verdi	35	Roma
Neri	null	Milano

15.1.3 Dimensione temporali

È complicato dare una definizione di tempo perchè si deve chiarire qual è precisamente la semantica della dimensione di qualità che si sta considerando.

Ci sono due principali proprietà temporali:

- currency o livello di aggiornamento
- tempestività

La currency è la rapidità con cui i dati sono aggiornati rispetto al mondo reale.
Una tipica misura della currency è il ritardo temporale tra quando l'evento avviene nel mondo reale e quando lo stesso viene registrato nel sistema, ad esempio

la registrazione di un voto nel database dell'università.
Un'altra metrica di currency è il capire la differenza tra tempo di arrivo all'organizzaizone e tempo in cui è effettuato l'aggiornamento, ad esempio una lezione pubblicata nel 2020 ma aggiornata al 2019.

La tempestività misura quanto i dati sono aggiornati rispetto al processo che li utilizza.
La tempestività è dipendente dal processo ed è associata al momento temporale in cui dev'essere disponibile per il processo che utilizza il dato.

Esistono dati con elevata currency ma che sono obsoleti per il processo che li usa.

15.1.4 Consistenza dei dati

La consistenza fa riferimento al fatto che i dati hanno dipendenze funzionali, vincoli di integrità referenziale, vincoli di dominio, vincoli di tupla, etc...
Un altro tema della consistenza è la diversa rappresentazione di uno stesso oggetto della realtà.

Esistono diversi formalismi per poter esprimere la consistenza:

- vincoli di integrità nel modello razionale, i quali esprimono dipendenze funzionali o di integrità referenziale
- vincoli di integrtà logica, ad esempio: ogni impiegato deve guadagnare meno del suo capo
- data edit (associato a indagini statistiche), ad esempio: età=15 anni, stato civile=sposato; questo rappresenta palesemente un errore nel caso dell'Italia

Il dato, inoltre, dev'essere accessibile a tutte le categorie di persone, dunque in particolare l'accessibilità dipende da:

- cultura, ad esempio accedere ad un dato in giapponese è difficile per un italiano
- stato fisico, ad esempio dati per non vedenti
- tecnlogie, ad esempio problematiche rispetto alla banda, alla rete, etc...

15.1.5 Problemi di tradeoffs

Consistenza e completezza possono non essere conciliabili se si vuole rispettare l'integrità referenziale.

Nel dominio statistico c'è il problema tra la tempestività e l'accuratezza. Un esempio sono i dati del Covid-19, per cui i dati aggiornati arrivano troppo tardi e quindi si preferisce avere dati sporchi ma analizzarli subito. Il problema si sposta,

quindi, nello stimare quanto un dato è sporco.

Nel web, invece, arrivare primi con una notizia è importante, quindi la tempestività va a discapito della completezza.

16 Quality improvement

L'obiettivo è il miglioramento dei dati: una volta misurata la qualità dei dati bisogna cercare di migliorarne la qualità. La qualità dei dati è un problema di tipo multidimensionale. Si possono migliore i dati sotto diversi aspetti.

Strategie:

- data-driven: si vuole migliorare il dato stesso
- process-driven: migliora i dati migliorando il processo col quale il dato viene acquisito e raccolto

16.1 Data-driven strategies

Il dataset in sè si migliore migliorando tutti i dati al suo interno.
Immaginando i dati come un lago inquinato si possono fare due cose:

- ripulire il lago periodicamente
- ripulire la sorgente che porta l'acqua al lago

Esistono, però, tecniche più puntiforme e puntuali:

- acquisition of new data: si scopre che i dati non sono abbastanza aggiornati, quindi si fa un'acquisizione di nuovi dati più aggiornati (maggiore qualità temporale)
- record linkage: si vede se esiste un altro dataset che contiene la stessa descrizione dell'oggetto che si sta studiando ma più completa e più ricca
- source trustworthiness: si ascquisiscono sorgenti dati da una sorgente affidabile di dati

16.2 Process-driven strategies

Fanno riferimento all'ambito BPR (Business Process Rengineering) ovvero quello di riprogettare i processi, perchè ad esempio si può scoprire che i dati vengono raccolti in modo sbagliato:

- process control: inserire elementi di controllo che verificano se ci sono degli errori e se il dato non è di buona qualità
- process redisign

Queste due tecniche servono se il processo di raccolta dati è continuativo.

Le differenze tra data-driven e process-driven sono che a lungo termine le tecniche process-driven sono più efficaci. Le tecniche data-driven, invece, possono essere più efficienti dal punto di vista economico perché c'è un impegno più ridotto di riscrivere il codice per acquisire dati, funzionano a breve termine e richiedono un tempo per eseguire le loro operazioni. Funzionano bene per app one-time (ovvero se serve il dato una sola volta).

Migliorare elementi specifici

- accuratezza: si cerca il dominio di riferimento e il valore più vicino al valore considerato e si stima un eventuale errore per migliorare il dato
- completezza: tecniche che usano la conoscenza del dato
- consistenza: si identificano le violazioni dei vincoli per andarli a correggere o con sorgenti esterne o acquisendo di nuovo le sorgenti di dati
- error localization and correction: tecnica molto importante presa nel dominio statistico
- deduplica: stessa informazione scritta due volte nello stesso dataset

16.3 Fasi di improvement e assessment

Fase di normalizzazione: si cerca di trasformare le stringhe secondo certi standard per rendere il tutto più uniforme migliorando la qualità del dato e migliorando l'attività di distanza di edit rispetto a tabelle di verità.

Miglioramento del singolo valore o dell'intera tupla per migliorare:

- accuratezza sintattica
- completezza
- concorrenza/aggiornamento temporale
- consistenza

Deduplica:

- raggruppo tuple che descrivono lo stesso oggetto del mondo reale
- raggruppo elementi distinti tra loro
- raggruppo tuple che probabilmente sono simili ma non è detto che siano semanticamente la stessa cosa

16.4 Casi di studio

La prima attività è un assessment perchè bisogna decidere quali sono le dimensioni di qualità per le quali si vuole fare un'analisi di qualità. Si definiscono, quindi, le metriche associate alle dimensioni di qualità.

L'attività di pre-processing si fa per migliorare la qualità e la misura dei dati.

Esempio: per la colonna *mese di nascita* si va a normalizzare decidendo, ad esempio, che il mese di nascita dev'essere sempre un numero.

Si fa, poi, un assessment e un improvement: per i nomi e per i cognomi bisogna trovare una tabella di verità, per gli indirizzi esistono i servizi di postalizzazione che indicano la georeferenziazione e la versione corretta della rappresentazione.

In seguito si può lavorare sul tema dei valori nulli; in alcuni casi si riuscirà a ricostruire, mentre in altri casi no.
I valori nulli hanno un contesto, e quindi si possono acquisire dati nuovi tramite delle euristiche.

16.5 Dati derivati

Un dato derivato è un dato ottenuto tramite un'operazione matematica o di manipolazione di stringhe esistenti.
Si può migliorare la consistenza dei valori nulli andando a vedere se la dipendenza funzionale tra i vari attributi è corretta.

17 Record linkage

17.1 Data integration

Prima del data integration c'è la fase di *schema integration* in cui si cercano di mettere insieme gli schemi. Si hanno diversi approcci, ad esempio un approccio architetturale che prevede un'integrazione virtuale e un consolidamento.
È necessario decidere se utilizzare un approccio GAV o LAV.

Potrebbe capitare, però, che esistono degli elementi della realtà descritti in modalità diversi. Se tutti gli schemi contenessero la stessa chiave primaria non ci sarebbe il problema in quanto basterebbe fare un join per avere tutti i dati.

Il processo di record linkage identifica quali righe nei diversi dataset descrivono lo stesso oggetto ma in modo diverso. Si potrebbe avere anche casi duplicati nella stessa tabella (deduplica).

Referencing

Il referencing è un processo per cui si ha un insieme di concetti o elementi e si ha anche una tabella di verità. Bisogna controllare se alcuni concetti, però, sono già presenti nel db.

Canonicalization

Il processo di canonicalization richiede la fusione dei dati, ovvero è tipico usarlo quando si raccolgono e integrano dei dati mettendo insieme le informazioni. Per farlo, però, bisogna scoprire dove sono le stesse informazioni dell'oggetto del mondo reale sparpagliate per vari dataset.

Record linkage

I risultati del record linkage sono un insieme di tuple che matchano o non matchano tra loro, più un insieme di gruppi di valori con informazioni per immaginare che siano collegati, ma non c'è la certezza per poterlo dire.

17.2 Tecniche

Esistono diverse tecniche per poter effettuare record linkage:

- empirica: si guarda la tabella e si cerca di capire quali sono gli oggetti che matchano
- probabilistico: si ha un campione di elementi da verificare, si ha la frequenza delle distanze tra i vari elementi e identificando soglie si può dire se matchano oppure no
- regole basate sulla conoscenza
- tecniche di machine learning

Per controllare se due tabelle hanno elementi che matchano tra loro bisogna confrontare ogni elemento della prima riga della prima tabella con ogni elemento della seconda tabella. Questo, però, ha una complessità quadratica $m \times n$ che cresce all'aumentare della dimensione delle tabelle; risulta quindi molto costoso e infattibile.

Steps record linkage

- supponiamo dei dataset in input A, B -
- spazio di ricerca $S = A \times B$
- ridurre lo spazio di cerca $S' = A \times B$ (questo è un problema)
- applicare comparazioni e modello di decisione per definire: (problema)
    - match

- unmach
- possible match

Un primo problema, ad esempio, può essere avere dataset in input in formato diverso. Si cerca, quindi, di trasformare un modello nell'altro scegliendo un modello target.
Bisogna tener conto anche della qualità dei risultati in quanto il sistema che si ottiene non dà certezze assolute.

Probabilistic record linkage

1. preprocessing
2. blocking
3. compare (choice of distance function)
4. compare (find a sampe of pair of tuples that are known to be matching or not matching)
5. decide (evaluate the distance for pairs in the universe)
6. decide (for pairs of tuple in the universe)
   - whose distance d is $d < d_{min}$ decide for matching
   - whose distance d is $d > d_{max}$ decide for unmatching
   - whose distance d is $d_{min} < d < d_{max}$ decide for possible match

Preprocessing

La fase di preprocessing ha come obiettivo quello di normalizzare i valori in modo tale che siano rappresentati tutti nello stesso modo.

17.3 Blocking

Ci sono vari algoritmi di blocking per poter raggruppare gli elementi.
L'idea è di prendere un attributo e in qualche modo carcare di ordinare la tabella sugli insiemi degli attributi in modo tale che gli elementi che si vanno a confrontare siano vicini tra loro per ridurre lo spazio di ricerca.
Una volta ordinata secondo la chiave k, si vanno a confrontare gli elementi a sinistra e a destra, tutti con tutti. Questo insieme è molto più compatto in quanto cambia lo spazio di ricerca, ovvero si restringe.

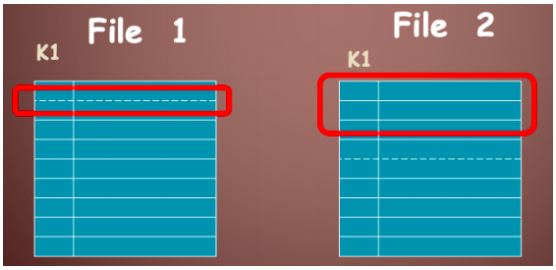

Figura 16: Esempio di blocking tra due tabelle per ridurre lo spazio di ricerca.

La finestra di sinistra scende (a partire dalla linea tratteggiata, non subito andando oltre) per confrontare altri elementi in modo tale da confrontare tutti gli elementi di sinistra.

Con n = numero di record e m = dimensione finestra si risolve tutto in un tempo $O(m^2 \cdot n/m)$.
Siccome $m << n$ quindi si riesce ad ottenere un vantaggio significativo in quanto il tempo è quadratico rispetto alla dimensione della finestra.
Questo aumenta la scalabilità della soluzione.

17.4 Comparison

Esistono diverse tipologie di approcci di confronto tra i dati:

- probabilistico
- supervisionato
- non supervisionato

Approccio probabilistico

L'approccio probabilistico vuole calcolare la probabilità di match tra x, y utilizzando dei vettori di distanza, in particolare la somma pesata dei loro vettori. Per ogni attributo si possono definire funzioni di distanza, come ad esempio la distanza di edit sulle stringhe, e si può dare un peso da 0 a 1 agli attributi con la somma dei pesi che deve fare 1.
È necessario definire degli score per il vettore (espressi anch'essi tra 0 e 1).

- scelta della funzione di distanza per ogni coppia di attributi
- definire per ogni attributo solo quelli rilevanti (ad esempio se si hanno due persone con stesso cognome, l'attributo cognome è rilevante, mentre tra due persone è meno rilevante l'attributo sesso perchè non aggiunge informazioni

Decision

Per decidere quali elementi matchano e quali no si fa una somma pesata. Per calcolare le soglie è suggerito di partire da una coppia e calcolare, a partire dai

dati che matchano e non matchano, la frequenza delle distanze tra questi.

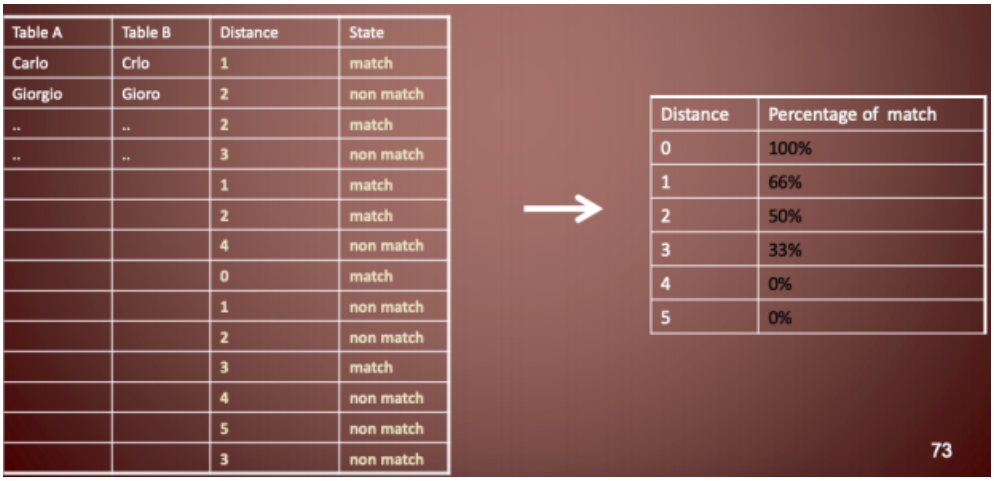

Figura 17: Esempio calcolo soglie tra attributi che matchano e non matchano.

Si prende un sottoinsieme dei due dataset costruendo un istogramma verticale delle distanze andando a vedere i dati che matchano e quelli che non metchano.

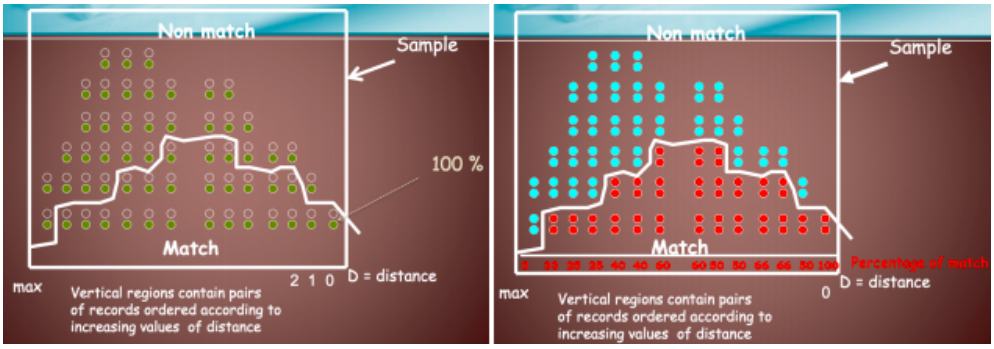

Figura 18: Esempio SVM attributi match o non match.

Il comportamento è quasi come una SVM.
All'aumentare della distanza aumentano i non match e diminuiscono i match.

Errori

Gli errori possono essere:

- falsi negativi
- falsi positivi

Questo dà origine ad un possibile allargamento/restringimento della fascia di incertezza dei possibili match.

Il record linkage è spesso legato al dominio.

Approccio supervisionato

L'idea è di fare comparazione tra coppie di un insieme di elementi.

Approccio non supervisionato

Clustering (ad esmepio con k-means): si prende un punto, si decide quali sono gli elementi più vicini, li si raggruppa, etc...

Spazio di embedding: i concetti simili appaiono frequentemente insieme cioè sono associati a vettori molto simili tra loro. Se si calcola, quindi, la distanza vettoriale tra due insiemi di vettori si scopre che appaiono molto vicini tra loro.

Quality assessment: si hanno falsi negativi e falsi positivi e si definiscono delle misure di recall e precision.

Fusione: due descrizioni di dataset diversi rappresentano lo stesso oggetto del mondo reale, bisogna solo metterli insieme.
Cosa succede però se si hanno dati diversi? Quale dato si deve considerare? Un esempio di questo caso potrebbe essere due tabelle in cui ad una stessa persona sono abbinati due stipendi diversi.

Strategie

Esistono diverse strategie per pensare di risolvere i conflitti:

- ignorare il conflitto
- evitare il conflitto: si sceglie di prendere un dato piuttosto che l'altro (in base a qualità o altro)
- risolvere il conflitto: ci si basa sulla qualità delle sorgenti

Non esiste la soluzione ottima, ma bisogna tener conto di tutti gli aspetti che si vogliono considerare. Ad esempio se tra due valori di stipendi diversi si prende lo stipendio più alto si sta sovrastimando il dato.
Nota: non è detto che il dato più recente sia quello più vero!
L'importante in questi casi è prendere una decisione, rimanerne coerente e tenere traccia del perchè ho preso quella decisione.

17.5 Caso di studio

Consideriamo la seguente tabella:

Tuple #	F.Name	Last Name	Y.Of Birth	M. Birth	D. Birth	Toponym	Name T.oponym	Number T.	City	Zip Code	Country
1	Miroslav	Konecny	1978	7	10	Street	St. John	49	Prague	412776	Czeck
2	Martin	Necasky	1975	7	8	Square	Vienna	null	Bratislava	101278	Slovackia
3	Miroslav	Konecny	null	null	null	Street	Saint John	49	Prague	412776	Czeck
4	Carlo	Btini	1949	6	7	Street	Dessiè	15	Roma	00198	Italy
5	Miroslav	Knecy	1978	7	10	Square	Budapest	23	Wien	k2345	Austria
6	Anisa	Rula	1982	9	7	Street	Sesto	null	Milano	20127	Italy
7	Anita	Rula	1982	9	7	Street	Sesto	23	Milano	20127	Italy
8	Anna	Rla	null	null	null	Street	Sarca	336	Milano	20126	Italy
9	Carlo	Batini	1949	6	7	Street	Beato Angelico	23	Milano	20133	Italy
10	Carla	Botni	1949	6	7	Avenue	Charles	null	Prague	412733	Czeck
11	Marta	Necasky	1976	11	8	null	null	null	Bratislava	112234	Slovakia

Figura 19: Tabella da considerare per il seguente esempio.

Deduplica

Ovviamente si deve decidere un blocking, ad esempio si può decidere di raggruppare gli elementi che hanno lo stesso country.
Bisogna fare tutti i confronti tra i vari blocchi.

Tuple #	Candidate tuples	F.Name	Last Name	Y.Of Birth	M. Birth	D. Birth	Toponym	Name T.oponym	Number T.	City	Zip Code	Country
5		Miroslav	Knecy	1978	7	10	Square	Budapest	23	Wien	k2345	Austria
1		Miroslav	Konecny	1978	7	10	Street	Saint John	49	Prague	412776	Czech
3		Miroslav	Konecny	null	null	null	Street	Saint John	49	Prague	412776	Czech
10		Carla	Botni	1949	6	7	Avenue	Charles	null	Prague	412733	Czeck
4		Carlo	Btini	1949	6	7	Street	Dessiè	15	Roma	00199	Italy
6		Anisa	Rula	1982	9	7	Street	Sesto	null	Milano	20127	Italy
7		Anita	Rula	1982	9	7	Street	Sesto	23	Milano	20127	Italy
8		Anna	Rla	null	null	null	Street	Sarca	336	Milano	20126	Italy
9		Carlo	Batini	1949	6	7	Street	Beato Angelico	23	Milano	20133	Italy
2		Martin	Necasky	1975	7	8	Square	Wien	null	Bratislava	101278	Slovakia
11		Marta	Necasky	1976	11	8	null	null	null	Bratislava	112234	Slovakia

Figura 20: Esempio confronto blocchi con tecnica di blocking.

Es blocco rosso: devo scegliere quali sono gli attributi che diventano rilevanti e decidere quali righe matchano tra loro. È abbastanza probabile che la tupla 1 e 3 rappresentano la stessa persona.
Es blocco blu: le tuple 4 e 9 rappresentano la stessa persona perchè si può assumere che nome, cognome e data di nascita siano gli attributi più significativi e si suppone che una persona possa avere più residenze (giustificato perchè via e città sono completamente diverse).
Sulla tupla 8 non si può dire che matcha con le tuple 6,7 perchè non si hanno abbastanza informazioni per dirlo.

A questo punto si possono assegnare delle misure di distanze.
Si deve capire qual è l'indirizzo corretto e, in base al contesto, tra 4 e 9 si decide

che l'indirizzo corretto è quello di 9 perchè si vede che è aggiornato in una data più recente guardando l'ultimo update.
Ci si accorge che persone completamente identiche hanno cuntry diverso. Probabilmente anche quei dati sono duplicati.

A questo punto si può fare la fusione degli elementi. La tabella finale colmando i valori nulli che si riescono a risolvere a dedurre dal contesto (non tutti) e risolvendo i conflitti è la seguente:

F.Name	Last Name	Y.Of Birth	M. Birth	D. Birth	Toponym	Name T.oponym	Number T.	City	Zip Code	Country
Miroslav	Konecny	1978	7	10	Street	Saint John	49	Prague	412776	Czech
Carla	Botni	1949	6	7	Avenue	Charles	null	Prague	412733	Czeck
Anita	Rula	1982	9	7	Street	Sesto	23	Milano	20127	Italy
Anna	Ria	null	null	null	Street	Sarca	336	Milano	20126	Italy
Carlo	Batini	1949	6	7	Street	Beato Angelico	23	Milano	20133	Italy
Martin	Necasky	1975	7	8	Square	Wien	null	Bratislava	101278	Slovakia
Marta	Necasky	1976	11	8	null	null	null	Bratislava	112234	Slovakia

Figura 21: Tabella risultato dopo aver risolto tutti i conflitti.

18 Big Data

Non c'è una vera definizione dei big data. Si possono intendere i Big data come un grande volume di dati, una grande velocità di arrivo e/o una grande variatà sulle informazioni. Inoltre è molto importante valutarne la veridicità.
Una battuta dice che i big data sono tutto ciò che non ci sta in un foglio excel.

È necessario definire il concetto di scale-in e scale-out:

- scale-in: si ha una macchina con certe caratteristiche, la si vuole migliorare, si prende una macchina più grande/potente. Il costo per tutto ciò è molto alto e si ha un grosso limite fisico perchè ad un certo punto ci si dovrà fermare (concetto di crescita verticale)

- scale-out: si hanno tante macchine in parallelo, quindi i costi sono sicuramente inferiori e tendenzialmente non si hanno limiti di crescita. Un problema può essere l'esistenza di una macchina più lenta che mi vincola tutto (concetto di crescita orizzontale)

I sistemi di big data si basano su uno scale-out: questa soluzione ha una scalarità lineare in quanto se anche si dovesse avere una crescita doppia basterebbe raddoppiare il numero di macchine.

La distribuzione del calcolo su più nodi (sistemi paralleli) ovviamente presenta delle criticità quali:

- sincronizzazione

- deadlock
- bandwidth
- coordinazione
- failure

Esistono una serie di tecnologie che permettono di sistemare questi problemi in maniera completamente trasparente.

Il concetto di Big Data riguarda l'elaborazione di grandi quantità di dati con scalabilità lineare. Si deve prevedere di spostare le applicazioni verso i dati e non viceversa. Inoltre, l'hw deve costare poco così che se si deve aggiungere qualcosa o qualcosa si rompe si può aggiugere/riparare a bassi costi.
I dati sono il nuovo petrolio, bisogna solo capire dove trovarli, come estrarli, come raffinarli e distribuirli.
Ci sono grandi "giacimenti di dati" come i dati presenti sui social, sul web, open data, ci sono piccoli "giacimenti di dati" come i dati reperibili dai sistemi IoT e dai sistemi CRM dove, ad esempio, si possono mantenere una serie di dati riguardanti clienti e performare una serie di azioni (gestione dati, analisi file di log del sito web). Tramite CRM si possono fare analisi di charm e capire quando un client vuole, per esempio, annullare un servizio e intervenire per cercare di evitare la situazione.

Il problema resta sempre la proprietà dei dati: di chi sono? A chi appartengono?

Dove trovare e come scaricare i dati

I dati sono di coloro a cui fanno riferimento. Se i dati sono disponibili su internet non è detto che si possono usare liberamente. Questa è una questione ancora aperta.
L'estrazione dei dati può avvenire da applicazioni, mediante wrapper (script che scaricano i dati), file di log o streaming.
Il fatto che il dato sia disponibile non fa si che sia facile da scaricare. Un altro problema è: dove vengono messi i dati?

Come raffinare i dati

Il raffinamento dei dati può avvenire mediante strumenti ETL di pulizia, ma c'è il solito problema della privacy.
Con i dati si potrà poi fare data mining e analisi.

Come distribuire i dati

È importante la distribuzione dei dati che deve avvenire mediante un formato comune e condiviso e deve prevedere interoperabilità.
C'è anche un problema di trasporto: le piattaforme di big data costruiscono dei

Wallet Garden, ovvero dei giardini murati dove ci si può muovere solamente all'interno (immagina di usare apple e di poter utilizzare solo applicazioni apple) e questo crea delle barriere molto rilevanti. Diventa molto complicato, quindi, il trasporto dei dati da un giardino all'altro.

È importante sottolineare come non sempre bigger is better. Se si hanno troppi dati, non sempre è una cosa positiva: avere troppi dati può compromettere i risultati. Cosa succede se la maggior parte dei dati sono sporchi? Si deve sempre prevedere una fase di features selection.

18.1 HDFS

La prima cosa da fare è analizzare i dati e per fare questo si utilizza un file system distribuito: HDFS. HDFS è un file system fortemente scalabile che consente la replica dei dati anche in caso di fallimento di nodi. Ogni nodo, infatti, è replicato su tre nodi diversi. La sua politica è: Write Once Read Many ed è pensato per analisi statistiche.
Un problema di HDFS è che non si può capire che dati si hanno dentro. Dev'essere pensato come un sistema di governance lato utente dove vengono inserite una serie di metadati aggiuntivi. Una volta che su un nodo è presente il pezzo di dato, bisogna pensare ad implementare le operazioni con la logica di *portare le applicazioni dai dati e non viceversa*; questo è importante perché costa di più spostare tera di dati che mega di applicazione. Ci si trova in un contesto scale-out.

Come già detto ogni nodo viene replicato in 3 nodi diversi, così nel caso in cui un nodo dovesse rompersi si ha una buona tolleranza.

Si ha un unico coordinatore (name node) e tanti slave (data node) che performano una certa operazione, infatti si parla di un'architettura master-slave.
Il coordinatore sa sempre dove sono i dati e li indirizzerà verso la macchina corretta.
I data node (slave) detengono i frammenti di informazione e performano qualche operazione, e si possono aggiungere quanti nodi si vuole.
Il name node è il coordinatore che indirizzerà le richieste verso il corretto data node.

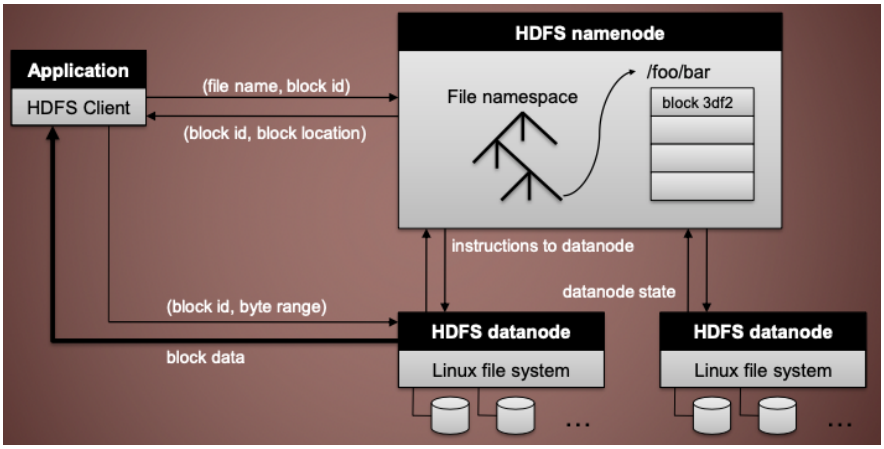

Figura 22: Architettura di HDFS.

Il problema di questa architettura è il name node: se questo cade non si sa più dove siano i dati. È prevista, per fortuna, una replica che rimane in aggiornamento guardando il file log del name node principale. Si ha, quindi, un file system distribuito nel quale ci sono un numero di data node che detengono una certa quantità di dati e vi performano sopra una certa operazione.
Il name node può indirizzare il client a un certo dato in quanto conosce la locazione fisica del dato ricercato e dovrà:

- coordinare tutte le operazioni
- gestire il file system namespace
- tenere nota della salute complessiva del sistema

Per mantenere nota della salute complessiva del sistema avviene uno scambio di messaggi per capire se i datanode sono "vivi" e funzionanti: in caso in cui un data node non dovesse rispondere questo verrebbe sostituito da una replica (lo si darebbe per morto).

Per "spezzettare" i dati bisogna impostare una logica: si può dire ad HDFS che un dato ha una certa struttura e questo verrà "spezzato" di conseguenza (es: non possiamo spezzare a metà un file json).

18.2 Map-reduce

Modello di programmazione e motore di esecuzione parallele per cui i programmi sono realizzati in stile funzionale e eseguiti in parallelo su grandi cluster di macchine poco costose.

Ha due fasi:

- map: si distribuiscono le attività ai nodi e si produce un risultato (partition)
  - $map : (k, v) \rightarrow [(k', v')]$

- reduce: mette insieme i risultati intermedi e genera il file di output (combine) - *reduce* : $(k', [v']) \rightarrow [(k', v')]$

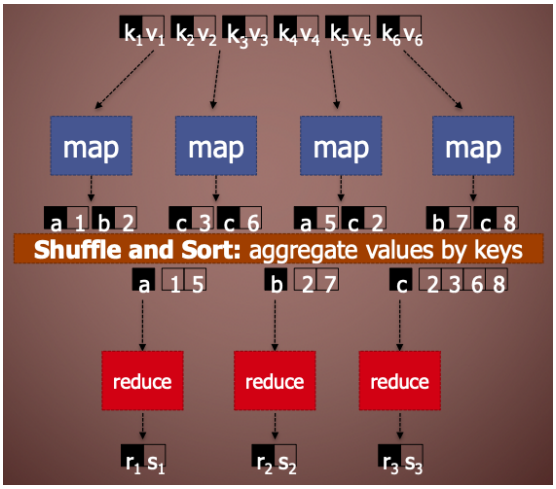

Figura 23: Esempio del funzionamento di Map-Reduce.

Si possono pensare una serie di problematiche, come ad esempio: l'assegnazione dei lavori alle unità di lavoro, cosa succede se ho più work che workers, condivisione risultati parziali, etc... Queste problematiche sono risolte in maniera gratuita da Map-reduce. Bisogna solo occuparsi di scrivere la funzione di Map e quella di Reduce.

Si devono indicare:

- partition: spesso si utilizza una semplice hash della chiave e si divide così lo spazio delle chiavi per l'esecuzione parallela delle operazioni di reduce

- combine: mini-reducers eseguiti in memoria dopo la fase di map, utilizzati come task di ottimizzazione per ridurre il traffico di rete.

Ogni worker fa la stessa cosa: prende una parte di dati e fa qualche operazione producendo un risultato. Tutti i risultati saranno combinati per produrre un output finale.

Un altro elemento fondamentale è il job-tracker, il quale ha una lista dei worker attivi e, per quelli attivi, lancia la macchina virtuale per eseguire o la funzione di map o la funzione di reduce.

18.3 Hadoop

Hadoop è la fusione tra HDFS e Map-reduce.

La sua architettura è sempre master-slave:

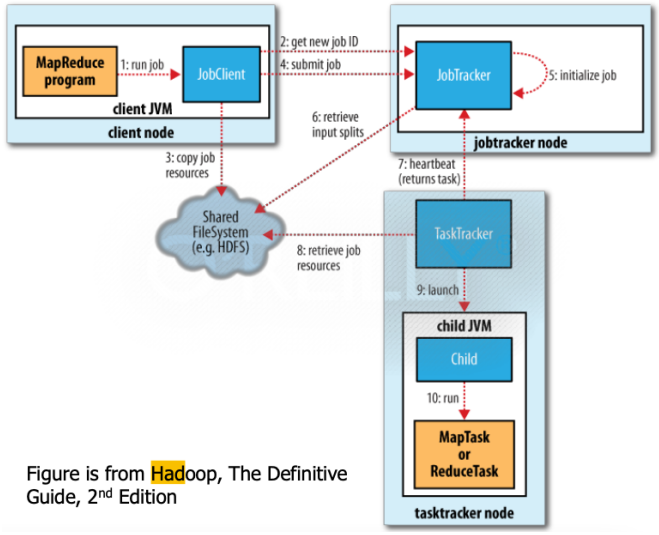

Figura 24: Map-reduce con job-client e job-tracker.

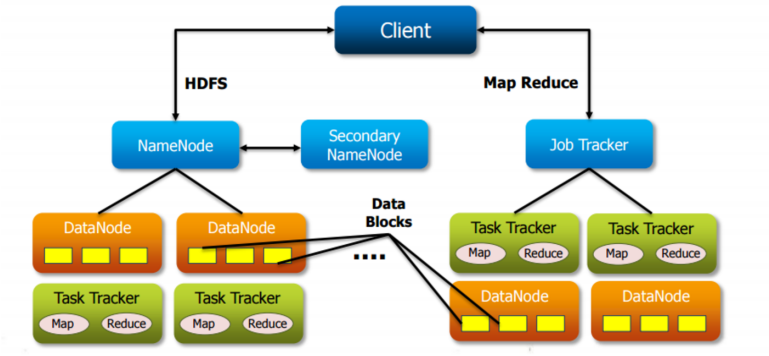

Figura 25: Hadoop architecture.

Si ha un filesystem distribuito con su installato un motore di calcolo distribuito. Su questo, poi, sono inseriti una serie di moduli come Pig (linguaggio di scripting per eseguire problemi di data analysis come un workflow, il tutto poi verrà tradotto in map-reduce), o come Hive per fornire un'interfaccia SQL-Like per i dati memorizzati in HDFS.

18.4 Hive

Il modello Map-reduce, però, presenta dei limiti:

- necessità di comprendere e usare il modello map reduce
- non è riusabile

Hive è molto intuitivo: gestisce i dati come dati relazioni anche se non lo sono permettendo dunque la possibilità di effettuare query SQL sui dati.
Si possono anche generare degli execution plan per le query.

Hive è un sistema di data warehousing per memorizzare i dati strutturati in Hadoop e consente di tradurre query tramite esecuzioni di piani Hadoop Map Reduce.