CORSO I.F.T.S

“TECNICHE PER LA PROGETTAZIONE E LA GESTIONE DI DATABASE ”

Ing. Mariano Di Claudio

Lezione del 03/10/2014

Indice

1. **Aspetti Architetturali**
   - Riak
   - eXist DB

1. **Aspetti *di Accesso ai Dati e di Rappresentazione***
   - RDF HDT Library
   - Neo4j

2. **Aspetti di Ingestion e Mining**
   - Tecnologie di acquisizione dati
   - Twitter API
   - Facebook API


riak
riak
EnterpriseDS
riak

- Database **NoSQL** *Open Source*, scritto in *Erlang con C/C++*

- E' un **DB di tipo Chiave-Valore** che implementa i principi descritti nel paper *Amazon's Dynamo*.

- E' un sistema **Masterless**, basato sul principio ***"eventually consistent"***

- I **dati** sono **distribuiti automaticamente** sui vari **nodi** utilizzando un ***"consistent hashing"***.

- Mette a disposizione diversi **strumenti** e **componenti aggiuntivi**.

Riak

Riak Control

- E' una **console grafica open source** per il *monitoraggio* e la *gestione* del cluster Riak.

RiakCS

- E' un semplice **sistema sw open source di storage cloud** costruito su un *database distribuito Riak*.

Riak Caratteristiche

Scalabilità

- I **dati** sono ***"ribilanciati"*** **in maniera automatica**, senza tempi di latenza, quando viene **eliminata o aggiunta una macchina**. I **dati** sono **distribuiti** sul cluster e le performance crescono in modo lineare con l'aggiunta di nuova capacità.

Disponibilità

- Quando un **nodo non è più disponibile** un suo ***"vicino"*** si assume le responsabilità delle operazioni di scrittura o degli aggiornamenti.

Semplicità operazionale

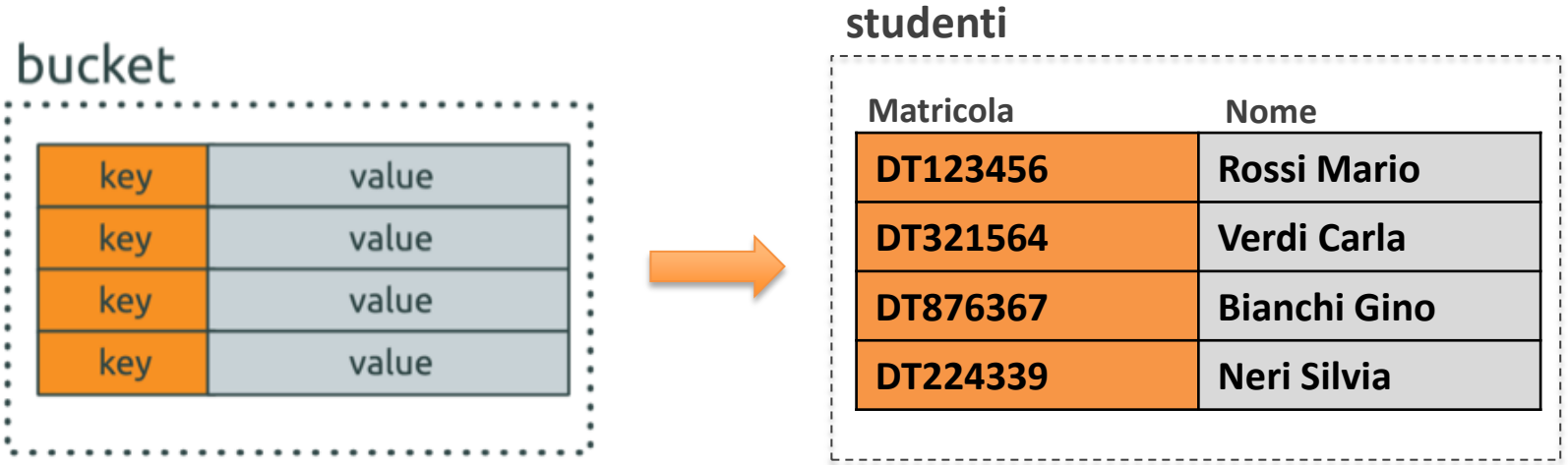
- **Aggiungere una nuova macchina** ad un *cluster Riak* è **un'operazione semplice** senza un eccessivo carico operazionale.

Riak Caratteristiche

Modello Dati Semplice

- I dati sono memorizzati come **coppie chiave-valore** all'interno di una struttura denominata **bucket** (*secchio*).

- E' accettato *qualsiasi tipo di dato*, gli *oggetti sono memorizzati su disco in formato binario*, secondo la **coppia bucket-chiave.**



Sviluppare codice per un modello di questo tipo è più **semplice ed efficiente**, adatto in particolare per applicazioni che richiedono *interazioni rapide e frequenti*.

Riak Caratteristiche

- Gli **oggetti** sono *l'unica unità di memorizzazione dei dati in Riak*, sono cioè strutture a cui è possibile accedere mediante la ***coppia Bucket-Chiave***.

- I ***Buckets*** *sono uno spazio virtuale* in cui *archiviare chiavi che fanno riferimento a dati simili*, questa logica ovviamente aumenta la velocità di esecuzione delle query.

- I ***Buckets*** possono essere pensati come ***tabelle se li si compara ai database relazionali*** oppure a delle ***cartelle se comparati ad un file system***.

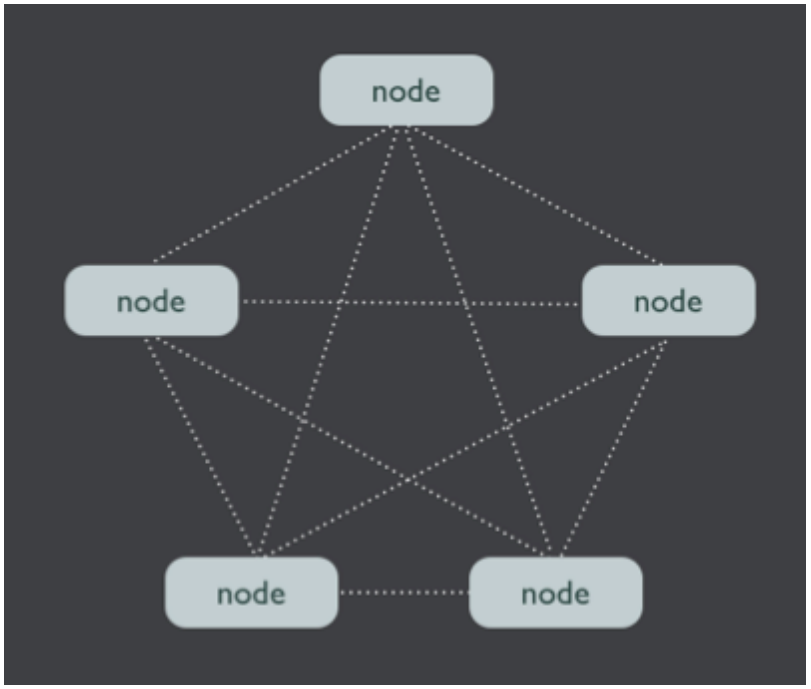
Riak Caratteristiche - Parametri

- Un **attributo** molto potente è ***n_val***, che permette di *indicare il numero di copie che ogni oggetto deve avere dentro il cluster*.
- Per specificare se un *oggetto può essere una copia* di uno già esistente si usa **allow_mult**.
- L'attributo ***Precommit*** può contenere una *lista di funzioni* (Scritte in Erlang e Javascript) *da eseguire **prima** di scrivere un oggetto*.
- L'attributo ***Postcommit*** può contenere una *lista di funzioni* (Javascript, Erlang) *da eseguire dopo aver scritto un oggetto.*

Riak Caratteristiche

Masterless Design

- **Non esistono ruoli *master/slave***. Ogni nodo può inoltrare o rispondere a qualsiasi richiesta in ingresso poiché i dati sono replicati su nodi multipli (*è consigliato un cluster di almeno 5 nodi*).



Questa configurazione garantisce:

- **Uguaglianza tra i nodi** del cluster.
- **Scalabilità** e **disponibilità nelle operazioni di scrittura** (non c'è bisogno dell'autorizzazione da parte del master).

Riak Caratteristiche

Il ***modello key-value*** *di Riak*, è ritenuto da molti utenti **flessibile** e **molto veloce per lo sviluppo di applicazioni**.
Ci sono però dei *compromessi* relativamente a **query** e i **tipi di dati disponibili**, infatti:

- Riak **non dispone** di ***contatori, operatori insiemistici e transizioni***.
- **Non supporta** le operazioni di **Join**, perché *non esistono i concetti di riga e colonna.*
- **Non esiste SQL** o un **linguaggio SQL-like** (le **interrogazioni** sono realizzate tramite *richieste HTTP*, le *API del protocollo buffer*, o tramite *varie librerie client*).

Riak offre comunque **funzionalità aggiuntive** sulla base del modello key-value.

Riak Caratteristiche

MapReduce

- Può essere utilizzato per **operazioni** di ***filtraggio di documenti tramite tag***, ***conteggio delle parole in un documento***, ***estrazione di collegamenti a dati correlati***. Offre inoltre supporto a javascript e Erlang.

Riak Ricerca

- E' presente un **motore di ricerca full-text distribuito** che fornisce il supporto a diversi ***MIME type*** **e** ***query robuste*** (*matching esatto, range query, ricerche di prossimità, wildcard*).

Indice Secondario (2i)

- Ogni **oggetto** memorizzato in Riak può essere **etichettato** con ***uno o più valori ricercabili***. Gli ***indici*** sono sia **interi** che **stringhe**, e possono essere *interrogati sia per matching esatto che per range query*.

Riak Caratteristiche

Tolleranza ai guasti

- Grazie a **meccanismi di replicazione automatica dei dati** nel cluster, anche **perdendo l'accesso ad uno o più nodi** (*per partizionamento della rete o fallimenti hw*), **i dati non vanno persi**.

Consistent Hashing

- E' un meccanismo che assicura che i **dati** siano **distribuiti uniformemente in modo automatico nel cluster**. Nuovi **nodi** possono essere **aggiunti** con un rimescolamento dei dati minimo e automatico.

Anti-entropia attiva

- **Proprietà di *"self-healing"* eseguita in background**. Utilizzo costante di un **hash tree di scambio** per *comparare le repliche* di un oggetto e *riparare/aggiornare in modo automatico eventuali divergenze*.

Riak – Operazioni CRUD

Create

- La **chiave** può essere **predeterminata o auto-generata** da Riak.

Read

- Accesso diretto per chiave.
- Indici secondari.
- Indici invertiti (ricerca testuale).
- Elenco delle chiavi del bucket.

Update

- Le update devono sempre contenere **tutto il dato**.

Delete

- Per **eliminare un bucket** bisogna cancellare ogni valore in esso contenuto.

Riak - Approfondimenti

- **Riak** è stato sviluppato per essere **utilizzato in un cluste**r, cioè un **insieme di nodi** che fisicamente rappresentano degli host.

- **Ogni nodo** ha al suo interno un **set di nodi virtuali** (***vnodes***), ognuno dei quali *si occupa dell'archiviazione dei dati in una partizione dello spazio delle chiavi* (*spazio di interi bucket-chiave*)

- **I nodi non sono dei cloni**, e non tutti partecipano per soddisfare una stessa richiesta. Il loro *modello comportamentale è configurabile*, infatti ad esempio si possono settare **tramite le API Riak**:

  - Il ***valore W***, che identifica il *numero di nodi che devono intervenire affinchè una richiesta di* ***Update sia completata***.
  - Il ***valore R***, che indica il *numero di nodi che devono rispondere positivamente per considerare una* ***lettura valida***.

Riak - Approfondimenti (Consistent Hashing)

Questo meccanismo rende possibile la **ridistribuzione automatica** dei dati sui nodi, assicura che i **dati** siano **equamente distribuiti** utilizzando una **struttura** ad ***"anello"*** *(ring)***.**

- Ogni **coppia bucket/chiave** viene ***mappata in un "anello" ordinato*** tramite una **funzione hash da 160-bit** (2^{160} valori).

- L'**anello circolare (*ring*) è diviso** in un certo **numero di partizioni**. Le *partizioni sono di uguale dimensione e ciascuna di esse corrisponde ad un intervallo di valori sull'anello*.

- Ogni ***vnodes*** è **responsabile di una partizione**, e i **nodi del cluster** cercano di **avere lo stesso numero di vnodes**.

Riak - Approfondimenti (Consistent Hashing)

Questo meccanismo rende possibile la **ridistribuzione automatica** dei dati sui nodi, assicura che i **dati** siano **equamente distribuiti** utilizzando una **struttura** ad ***"anello"*** *(ring)***.**

- *Un Nodo nel cluster è responsabile di*:

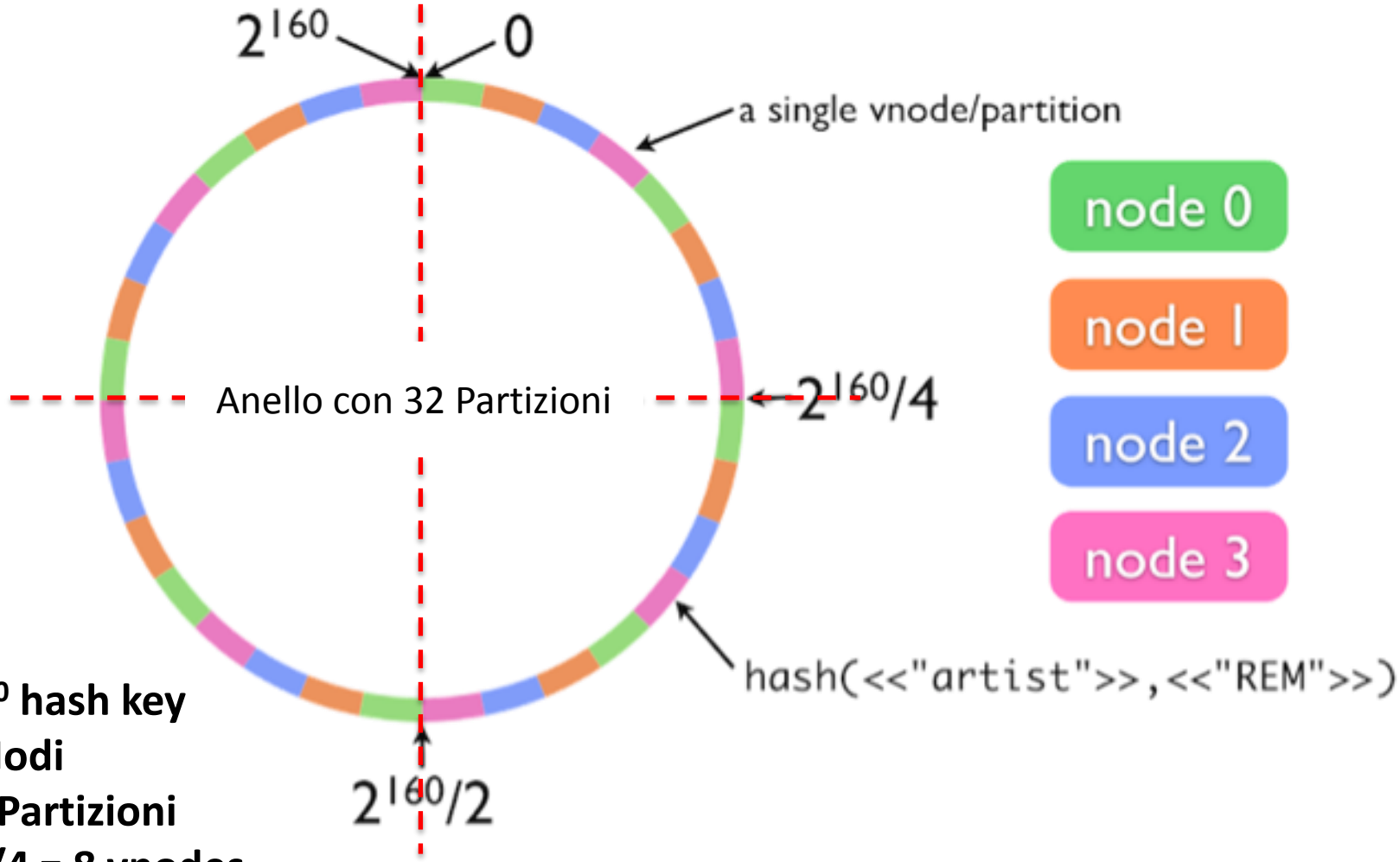
  1/(numero di nodi) dell'anello

- *Un Nodo nel cluster sarà composto da:*

  (numero di partizioni)/(numero nodi) vnodes

Ad **intervalli regolari ogni nodo rivendica (controlla) le sue partizioni** nel cluster, in modo da **mantenere una distribuzione uniforme** dei dati e **evitare** che un nodo sia **responsabile di più di una replica di una chiave**.

Riak - Approfondimenti (Consistent Hashing)



- **2^{160} hash key**
- **4 Nodi**
- **32 Partizioni**
- **32/4 = 8 vnodes**

Riak - Approfondimenti (Consistent Hashing)

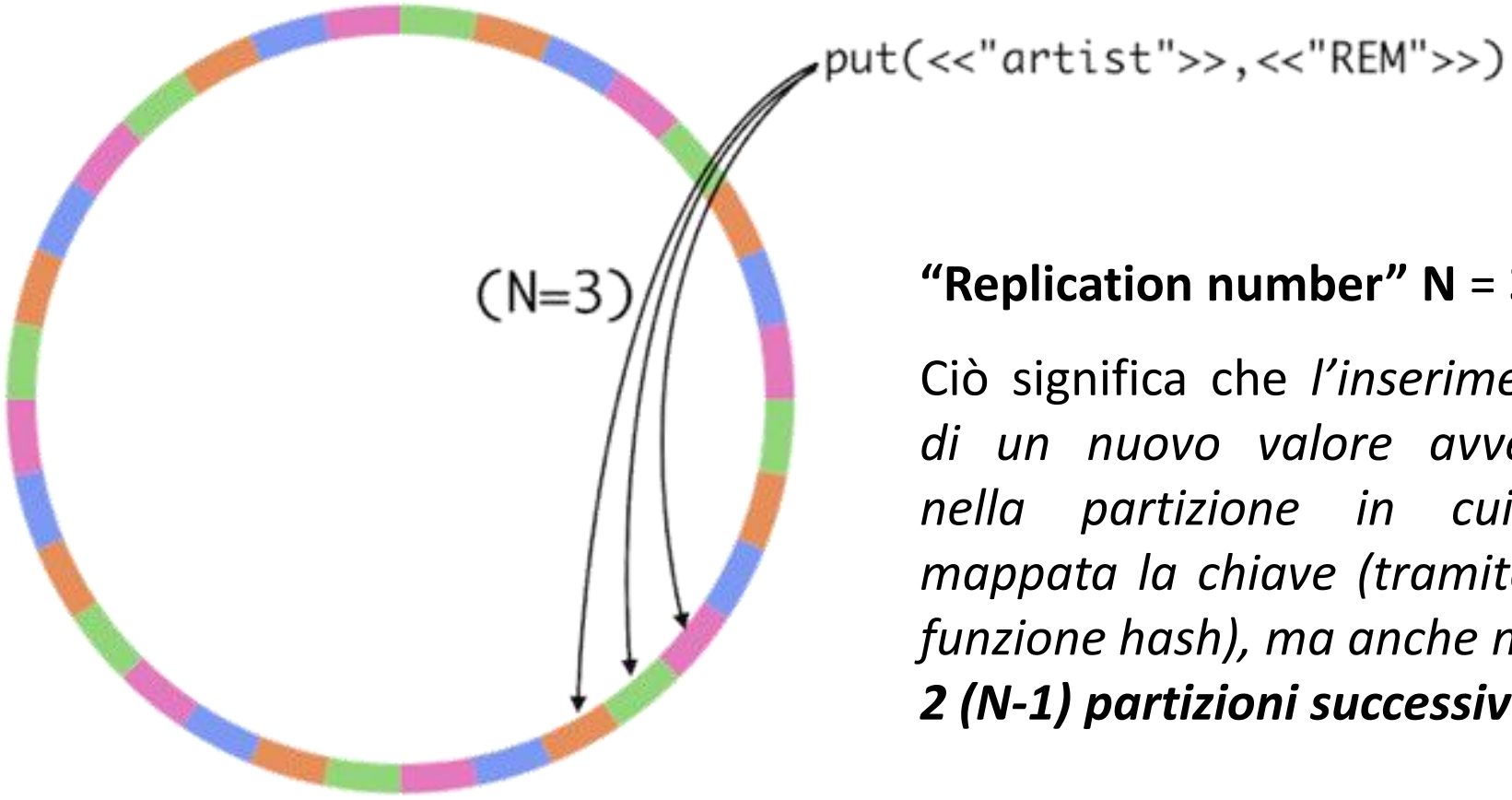
Quando si **memorizza un nuovo valor**e nel cluster **ogni nodo può partecipare come coordinatore.**

- Il **nodo di coordinamento** legge lo **stato del cluster** per determinare quale ***vnode* è responsabile della partizione relativa alla chiave** del valore da inserire.

- La richiesta di ***"put"*** viene inviata prima al **vnode responsabile**, e poi anche ad altri **N-1 vnodes**, responsabili delle **"prossime" N-1 partizioni sul ring**.

- **N** è un **parametro configurabile** del bucket che indica il **numero di copie (repliche) del valore da memorizzare** (in questa richiesta è possibile specificare anche il parametro W).

Riak - Approfondimenti (Consistent Hashing)



“Replication number” N = 3.

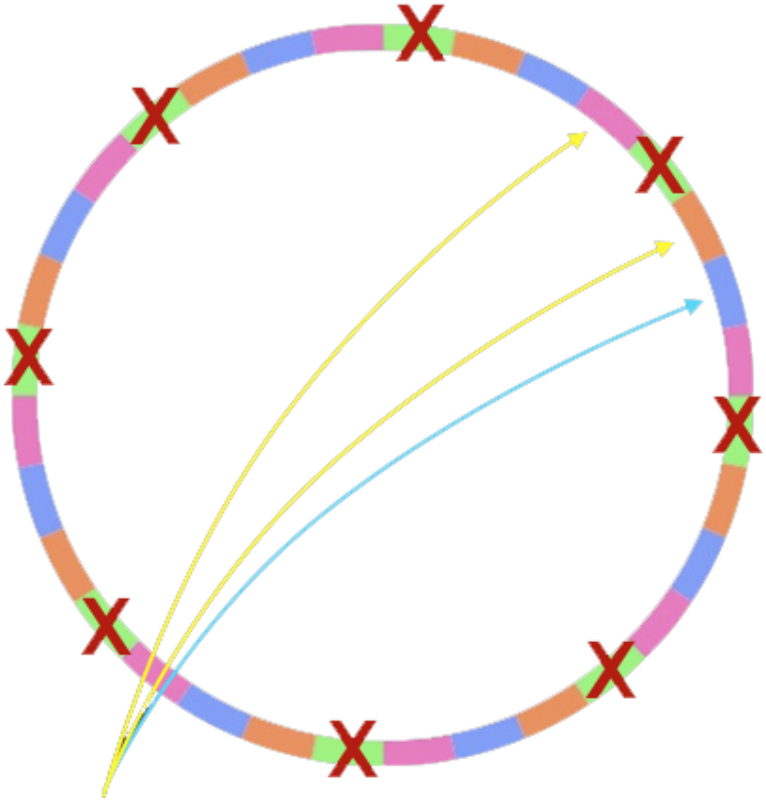
Ciò significa che *l’inserimento di un nuovo valore avverrà nella partizione in cui è mappata la chiave (tramite la funzione hash), ma anche nelle* ***2 (N-1) partizioni successive.***

Riak - Approfondimenti (Fallimento Nodo)

- Quando **un nodo fallisce**, le **richieste** vengono **inviate ad una *"replica secondaria"***, in cui sono stati **copiati i dati** dalle altre **repliche primarie ancora attive**.

- Se il **nodo viene ripristinato**, tramite l'***handoff*** i dati vengono nuovamente ritrasferiti sul **nodo originale**.

- ***Il sistema recupera il suo normale funzionamento***.



Riak - Aggiunta/Rimozione nodo

- Un **nuovo nodo** può essere **aggiunto ad un cluster esistente**. Ciò si realizza mediante una **richiesta join**:

  riak-admin cluster **join** riak@192.168.15.2

- Se l'operazione ha **successo**, il **nodo** riceve lo stato dell'anello e **comincia ad essere attivo** all'interno del cluster.

- Lo stato dell'anello è sempre conosciuto da tutti i nodi del cluster tramite un **"gossip protocol".** *Quando un nodo subisce una modifica (es. sul relativo spazio delle chiavi) la comunica ad una latro nodo e così via a cascata.*

- Dopo l'***handoff*** (*trasferimento di dati da un nodo esistente a quello appena aggiunto*), il **nuovo nodo è pronto** a soddisfare le richieste di interrogazione.

Riak - Vector Clock

- E' un **metodo per tener traccia della versione più recente di un dato.**
- Quando un **nuovo valore** è memorizzato in Riak, viene **etichettato con un "*vector clock*"** che ne stabilisce la sua **versione iniziale**.

```
a85hYGBgzGDKBVIcR4M2cgczH7HPYEpkzGNlsP/VfYYvCwA=
```

- Ad ogni **aggiornamento** il **vector clock è esteso** con uno specifico meccanismo *in modo tale che Riak possa comparare due repliche di un oggetto e determinare*:
  - **Se un oggetto è discendente diretto di un altro oggetto.**
  - **Se più oggetti sono discendenti diretti di un oggetto comune.**
  - **Se più oggetti sono scorrelati tra loro.**

Riak - Vector Clock

- Tramite il ***vector clock*** è possibile **tracciare quali modifiche sono state applicate** ad un oggetto e chi le ha realizzate.

- E' come avere una **sorta di array in cui sono conservati le modifiche** dei vari client e *l'ordine con cui sono state eseguite*.

- In questo modo **è possibile capire se un oggetto è aggiornato** oppure se si sono verificati dei conflitti in fase di scrittura.

- Usando **questa conoscenza**, Riak può applicare, quando possibile, dei ***meccanismi di auto-riparazione di dati non sincronizzati***; o almeno fornire ai client con la possibilità di riconciliare i cambiamenti divergenti con meccanismi legati alle specifiche applicazioni.

Riak - Query data

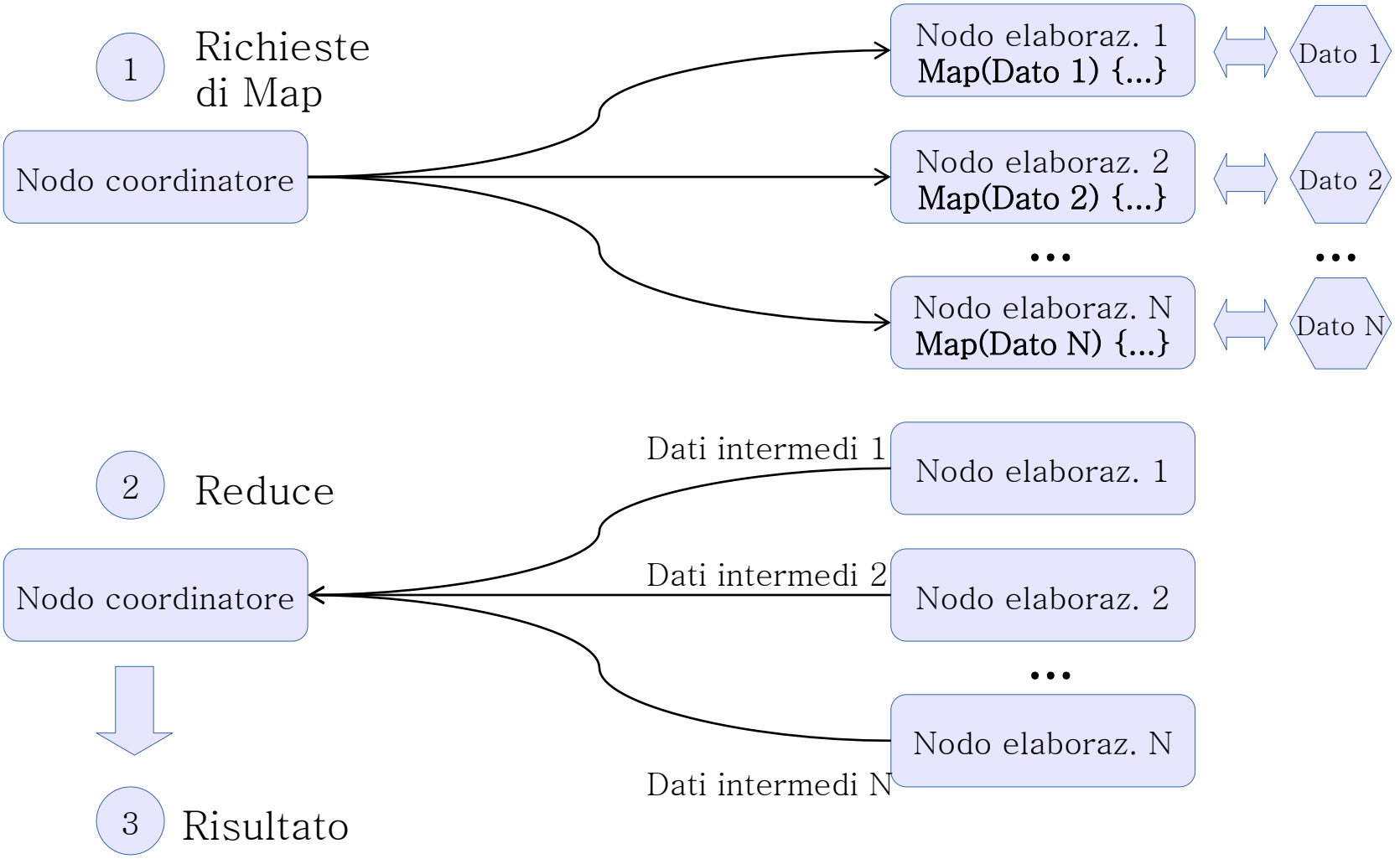
Chiave Primaria

- **GET, PUT, UPDATE, DELETE**

MapReduce

- Permette di realizzare **query più efficienti** sui dati memorizzati.
- Fa si che l'elaborazione tragga **vantaggio dal processamento dei dati in parallelo**.
- Divide le **query in diversi step** e i **dataset in diversi blocchi**, successivamente queste coppie step/blocchi vengono eseguiti su host separati.
- E' più efficiente realizzare l'elaborazione dove sono i dati che nel portare i dati all'elaborazione (**10KB Vs 10GB**).

Riak - MapReduce



Riak - Query data

Indici Secondari

- Per ricercare i dati in modo differente dalla ricerca basata sulle coppie bucket/chiave.
- Sono costruiti sui metadati, è più una sorta di "***tagging***".

Riak Ricerca

- Supporta **diversi tipi MIME** (JSON, XML, testo normale, Erlang) per l'estrazione automatica dei dati.
- **Punteggi e classifica** per catalogare i risultati più interessanti.
- **Corrispondenza esatta** (caratteri jolly, range query, raggruppamento, corrispondenza del prefisso, query di prossimità).

Riak - Query Esempio

- **Ricerca Riak via riga di comando**

bin/search-cmd search books “title:\”pinocchio””

- **Interfaccia HTTP Solr-compatibile**

http://hostname:8098/solr/select

Parametri:
index=INDEX - q=QUERY - df=fieldname - start=N - rows=N - wt=FORMAT - sort=fieldname…

Riak Ricerca:

- **Ricerca per Caratteri Speciali: “bus*” “bus?”**
- **Ricerca per Prossimità: “pinocchio”~20**
- **Ricerca per range: ”campo:{red TO rum}"**

Riak - Caso d'uso *Monitoraggio Server*

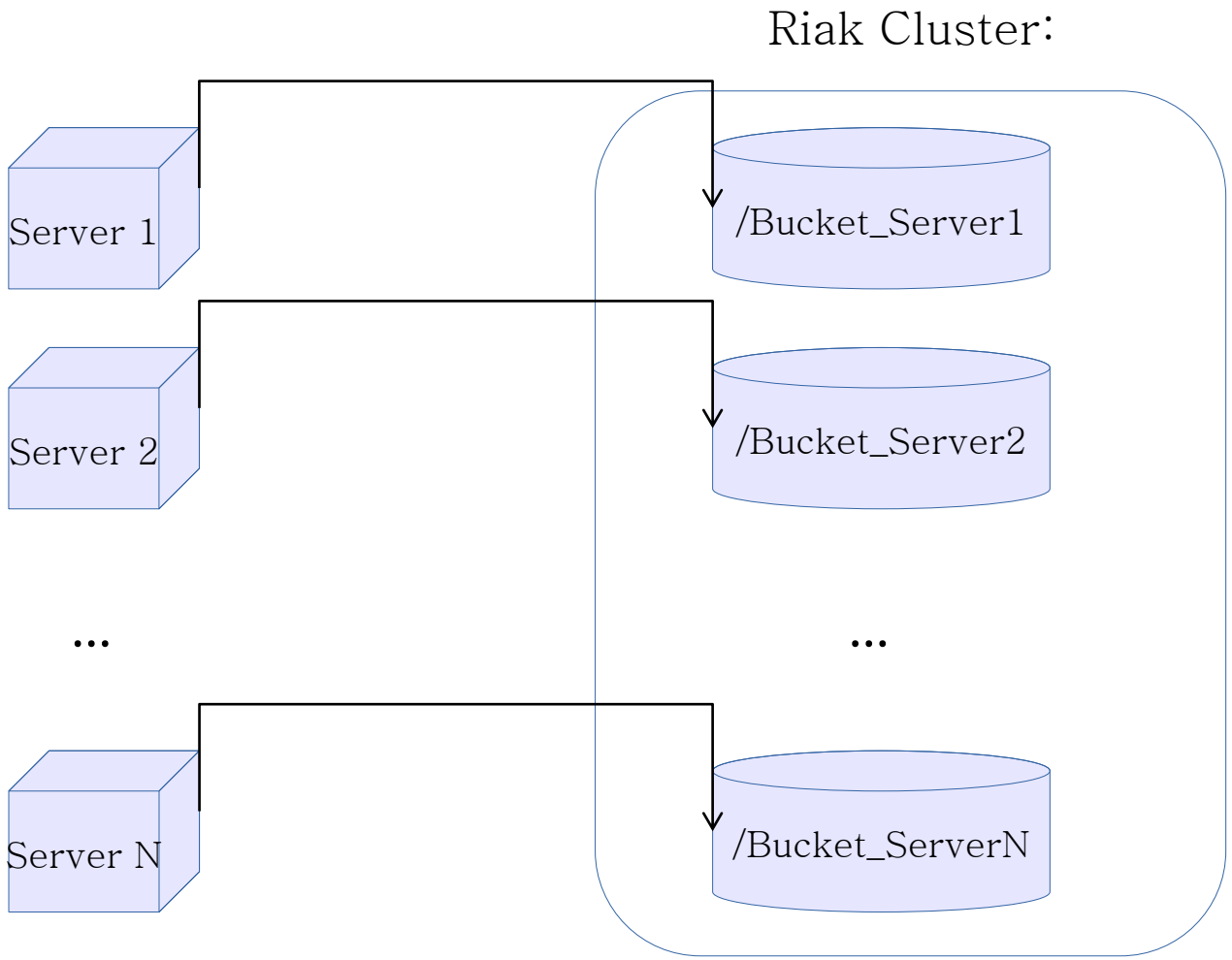
- Raccolta di dati di monitoring come *utilizzo di memoria, CPU, etc*.

- Applicazione caratterizzata da una *quantità e da un'intensità notevole di dati*.

- Possibile Struttura:
  - **Bucket**: identificativo del server
  - **Key**: timestamp, con intervallo prefissato (ad es. 10 secondi)
  - **Value**: JSON con i valori dei vari sensori

    esempio : *{"CPU": 0.8, "MEM": 0.5}*

Riak - Caso d'uso *Monitoraggio Server*



Riak - Caso d'uso *Monitoraggio Server*

- Si presta ad **analisi massive con *MapReduce***

- **Flessibilità**: possiamo memorizzare qualsiasi valore, anche non inizialmente previsto

- Tolleranza a comportamento non-ACID:
  - *Configurazione write-intensive (W=1)*.
  - Gestione della **perdita di dati.**
  - Non sono necessarie transazioni ACID perché *i dati, una volta scritti, non cambieranno mai*.


existdb

- **eXist** è un **DBMS open source** interamente **basato sulla tecnologia XML**, infatti viene anche chiamato ***XML database nativo***.

- A differenza dei classici database management system relazionali, **eXist utilizza Xquery per la manipolazione dei dati.**

eXistdb - Scheda

- **API e Query Method:** Xpath e Xquery 1.0
- **Protocolli:** HTTP/REST, SOAP, WebDAV, XML-RPC, Atom.
- **Interfacce:** XML:DB, fornisce l'accesso da codice.
- **Scritto in:** Java (open source).
- **Concorrenza:** *Letture concorrenti e blocco in scrittura*.
- **Misc:** Xquery permette di scrivere intere web applications utilizzando XSLT, XHTML, CSS e Javascript (per le funzionalità AJAX). La versione 1.4 ha introdotto un indice di ricerca completamente testuale basato su Apache Lucene.

eXistdb - Caratteristiche

Data Storage

- **Archivio di dati XML** basato sui **B+tree e file di paginazione**.
  I documenti (nodi) sono memorizzati in un DOM persistente.

Collezioni

- I *documenti vengono gestiti in* ***collezioni gerarchiche***, nello stesso modo in cui vengono memorizzati i file in un file system.

Indicizzazione

- Si basa su uno **schema di indicizzazione numerico** che supporta la **rapida identificazione delle relazioni tra i nodi**, come *padre/figlio, antenato/discendente o fratello precedente/successivo*.

eXistdb - Caratteristiche

Creazione dell'indice

- Di default *l'indice è* ***creato in automatico***. E' possibile l'utilizzo di diverse tipologie di indici:
  - ***Indici strutturali -*** per elementi, attributi dei documenti XML.
  - ***Indici testuali -*** per il testo e valori degli attributi (di tipo testuale).
  - ***Indici intervallo*** *(range-index)* - per i valori digitati (immessi).

- ***L'indicizzazione full-text*** *può essere* ***attivata/disattivata per parti di documento*** *selezionate.*

Aggiornamenti

- Tramite **Xquery** e **Xupdate** è possibile realizzare *aggiornamenti di tipo document-level o node-level*.

eXistdb - Caratteristiche

Rilascio

- eXist può essere distribuito come **database server stand-alone**, come una **libreria Java embedded** o come **parte di un'applicazione web** (*in esecuzione su un servlet engine*).

Query Engine

- Il query engine di eXistdb **evita attraversamenti top-down o bottom-up dell'albero**. Invece si basa su ***algoritmi di fast path join*** *per calcolare le* ***relazioni tra i nodi***. I Path join possono essere applicati ***all'intera sequenza di nodi in un singolo passaggio***.

Meccanismo di Autorizzazione

- Esistono dei **meccanismi di autorizzazione degli accessi a collezioni di dati** o **documenti** di tipo *Unix-like per utenti o gruppi*.

eXistdb - Caratteristiche

Sicurezza

- E' presente **eXtensible Access Control Markup Language** (***XACML***) *per il controllo degli accessi tramite XQuery*.

Transazioni e Crash recovery

- Il database è in grado di **supportare il recupero di un crash completo** basato su un ***"journal"*** in cui ***sono loggate tutte le transazioni eseguite***. In caso di crash tutte le operazioni previste vengono rieseguite, quelle incomplete ripristinate.

Funzionalità di Backup/Ripristino

- Fornita tramite *client admin Java* o *script Ant*. Il **backup contiene risorse come un XML leggibile** che permette il *ripristino completo di un database, degli utenti e dei gruppi*.

eXistdb - Limiti

Numero Massimo di documenti

- Essendo un database XML nativo eXist è in grado di memorizzare e interrogare documenti XML di complessità e profondità arbitraria.
- Di default il **numero massimo di documenti** che è possibile archiviare in un database eXist è fissato a $\mathbf{2^{32}}$**,** ma questi limite può essere superato.

Dimensione massima di un documento

- *Teoricamente i documenti possono avere una dimensione arbitraria*. In pratica, alcune **proprietà come il conteggio dei figli di un nodo** sono limitati a un **numero intero (int) di 4 byte**. Inoltre ci sono anche alcune **limitazioni legate al sistema operativo**; ad *esempio la dimensione massima di un file nel filesystem*, può influenzare la modellazione del sistema.

eXistdb – Vecchio schema di indicizzazione

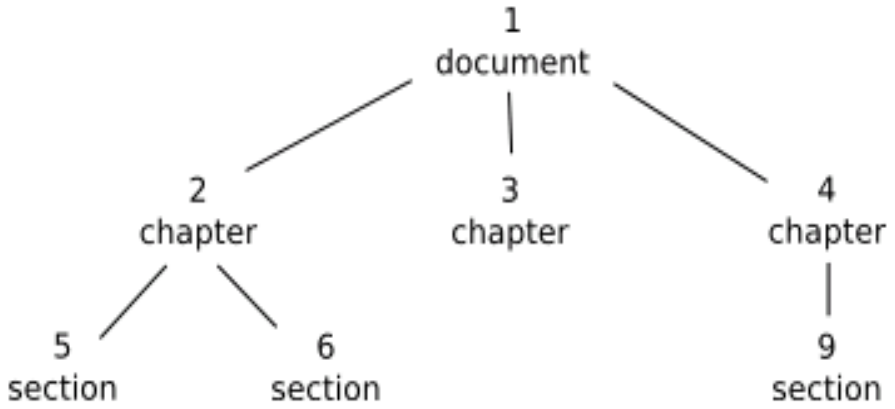
Schema di indicizzazione

- Permette una **rapida identificazione delle relazioni** strutturali **tra i vari nodi**.
- In questo **modo per determinare le relazioni tra i nodi** *non c'è bisogno di* ***caricare i nodi*** *stessi nella* ***memoria principale***.
- Ogni **nodo è etichettato nel *document tree*** da un **identificatore univoco (*node ID*)**.
- **Non** sono necessarie **ulteriori informazioni** per determinare se un **nodo è un antenato**, **un discendente o un fratello** di un altro nodo.
- Lo **schema di numerazione** originario utilizzato in eXist era **basato su una numerazione level-order**.

eXistdb – Vecchio schema di indicizzazione

- La **numerazione level-order** *modella il document tree come un k-tree completo*, ***assumendo che ogni nodo nell'albero abbia k nodi figlio***. Se un nodo ha *meno di k figli*, gli *ID dei figli mancanti vengono lasciati vuoti* prima di riprendere la numerazione.

- **Molti ID non saranno assegnati** ai nodi reali. L'albero risultante è un k-ario completo, *gli ID disponibili si esauriscono velocemente limitando la dimensione massima del documento da indicizzare*. -> modifica



eXistdb – schema di indicizzazione modificato

- Invece di considerare un **numero** fisso di **nodi figli**, **eXist ricalcola questo numero per ogni livello dell'albero**.

- L'indicizzazione di un documento con una struttura regolare delle dimensioni di qualche centinaia di MB non è più un problema (anche in termini di tempi di realizzazione).

- **Lo schema di numerazione/indicizzazione** comunque **non lavora bene se il document tree è piuttosto sbilanciato**.

La numerazione level-order ha alcuni svantaggi

- *L'inserimento, l'aggiornamento o la rimozione di nodi* in un documento memorizzato ***è molto costosa***, infatti *richiede almeno una parziale* ***ri-numerazione*** *e* ***re-indicizzazione*** *dei nodi del documento.*

eXistdb – Nuova soluzione di indicizzazione

- **2006**, viene introdotta ***"dynamic level numbering" (DLN),*** che consiste in **ID gerarchici** (*secondo la classificazione decimale di Dewey*).

- ***Schema di numerazione basato su ID di lunghezza variabile*** che evita di porre un limite concettuale sulle dimensioni del documento da indicizzare.

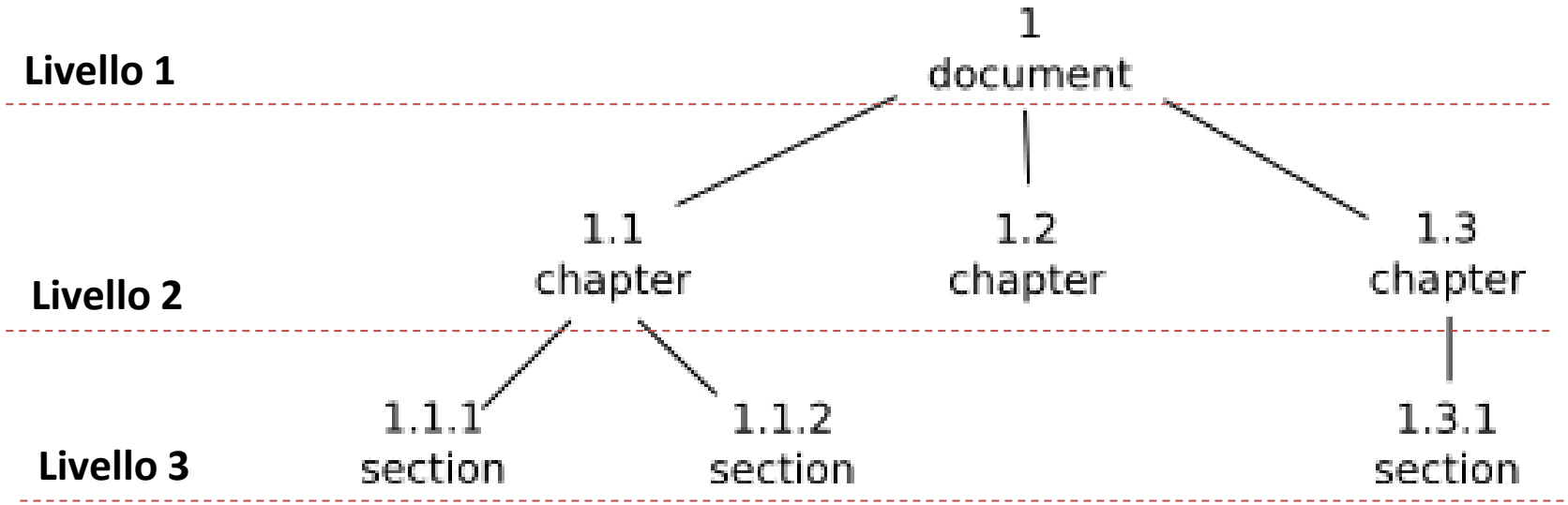
- Ciò porta al **miglioramento** di diversi aspetti, tra cui gli ***aggiornamenti rapidi dei nodi senza re-indicizzazione.***

- **L'ID è una sequenza di valori numerici**, separati da un separatore. **Il nodo radice ha ID=1**. *Tutti i **nodi sottostanti** consistono nell'avere l'**ID** del proprio **nodo padre** usato **come prefisso** a cui va aggiunto un **valore corrispondente al livello.***

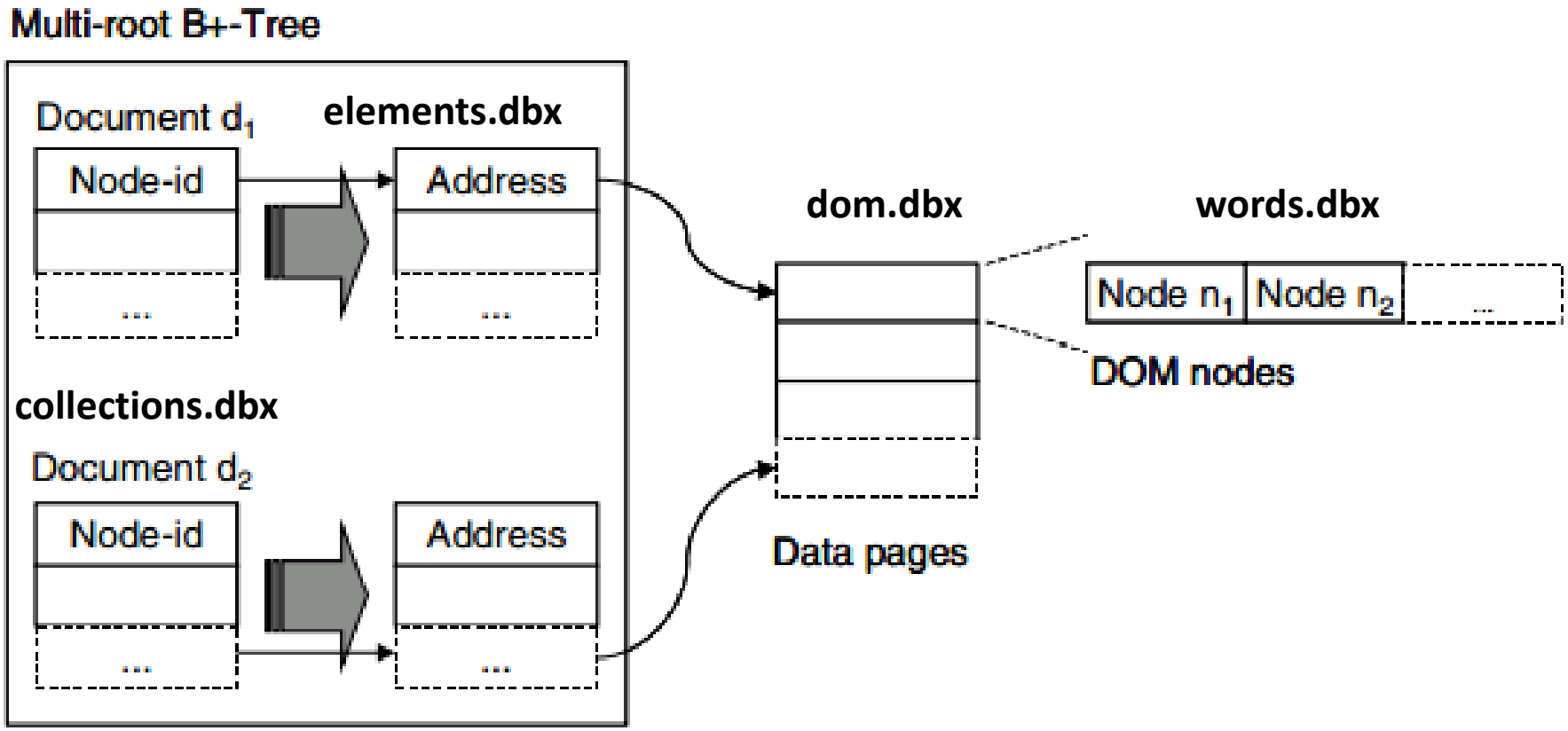
eXistdb – Nuova soluzione di indicizzazione

- ***1** è il **nodo radice*** (livello 1), ***1.1 è il primo figlio*** sul secondo livello, ***1.2 è il secondo figlio*** sul secondo livello…e così via. Si può osservare come il **numero delle cifre di un ID, identifica il numero del livello**.

- Utilizzando **questo schema**, determinare la ***relazione tra una qualsiasi coppia di ID dati (nodi) diventa un'operazione banale***, così come identificare il rapporto antenato-discendente.



eXistdb – Indexes and Storage Model



eXistdb – Indexes and Storage Model

eXist utilizza **4 file indici** (*basati sui B+-trees*)

- **dom.dbx -** Raccoglie i DOM dei nodi in un file di paginazione, e associa gli identificatori dei nodi ai nodi attuali.
- **collections.dbx** - Gestisce le collezioni gerarchiche, e *mappa **nome collezione – nome documento.*** Durante l'indicizzazione ad ogni **coppia collezione-documento** è assegnato un **ID univoco**.
- **elements.dbx** - indicizza elementi e attributi.
- **words.dbx** - tiene **traccia del numero di occorrenze** di ciascuna parola ed è utilizzato come estensione di ricerca full-text.

eXistdb – Indexes and Storage Model

- **Questa organizzazione** aiuta a **contenere il numero di pagine interne di un B+tree** e produce **migliori performance per le query su intere collezioni.**

- La **creazione di una voce di indice per ogni singolo documento** in una collection porta a ***diminuire le prestazioni per le collezioni contenenti un grande numero di documenti*** (**> 5000**) o ***di dimensioni molto piccole*** (**<50KB**).

eXistdb – Esempi di Query

- **doc()** è ristretta ad un singolo argomento di document-URI.
- **xmldb:document()** accetta path multipli di documenti da includere nel nodo di input.
- **xmldb:document()** senza specificare alcun argomento include tutti i document node presenti nell'istanza corrente del database.
- doc("/db/shakespeare/plays/hamlet.xml")//SPEAKER
- xmldb:document('/db/test/abc.xml', '/db/test/def.xml')//title

Approfondimento – XML:DB

Che cos'è XML:DB?

- **XML:DB** è *un'iniziativa pubblica* (aperta a studenti, ricercatori, professori…) **volta a definire delle API** specifiche per *l'accesso ai database XML nativi.*

- **La struttura delle informazioni** vista attraverso **XML:DB** è *estremamente diversa dalla classica* e nota struttura *schema-tabella-record-campo* che caratterizza i tipici **DBMS relazionali**.

- **Il modello organizzativo** di questo tipo di DBMS è molto più simile a quello di un ***filesystem*** (modello che presenta un'organizzazione ad albero o cartelle).

Il DB visto da XML:DB

- **Le strutture di base** esposte da XML:DB sono 3:
  1. ***Collection,*** corrispondono approssimativamente agli schemi e alla tabelle relazionali.
  2. ***Resource,*** che si possono pensare come i record relazionali.
  3. ***Service,*** é un sistema di accesso alle funzionalità avanzate del DBMS, sostituisce e integra le funzioni dell'SQL nei RDBMS.

- A questi oggetti di base (gestiti dalle API) si aggiungono:
  - ***DatabaseManager***, che gestisce le connessioni ai DBMS XML nativi.
  - ***ResourceSet,*** cioè un insieme di Resource che viene utilizzato come output delle funzioni di query.

Panoramica sulle tecnologie Big Data

Considerando le tecnologie e le soluzioni adatte a lavorare con i **Big Data** bisogna prestare attenzione a **4 classi di aspetti fondamentali**:

- **Aspetti *di Accesso ai Dati e di Rappresentazione***

E' opportuno avere dei tool che consentano una **visualizzazione scalabile dei risultati** ottenuti dall'analisi dei Big Data.

Aspetti di Accesso ai Dati e di Rappresentazione

Accesso Utente

- Grazie al **sistema di gestione degli accessi** utente è possibile definire delle **diverse tipologie di utenti** (*Admin, Utente semplice etc*.).
- In base alla **tipologia di utente** si può consentire **l'accesso a parti differenti del sistema** e **dei Big Data** memorizzati.

Separazione delle funzioni

- Utilizzando una combinazione di *autorizzazione, autenticazione e crittografia*, si possono **separare i compiti dei diversi tipi di utenti**.
- Questa **separazione** fornisce un **forte contributo per salvaguardare la privacy dei dati**, che è una caratteristica fondamentale in alcuni *settori come la sanità e la pubblica amministrazione.*

Aspetti di Accesso ai Dati e di Rappresentazione

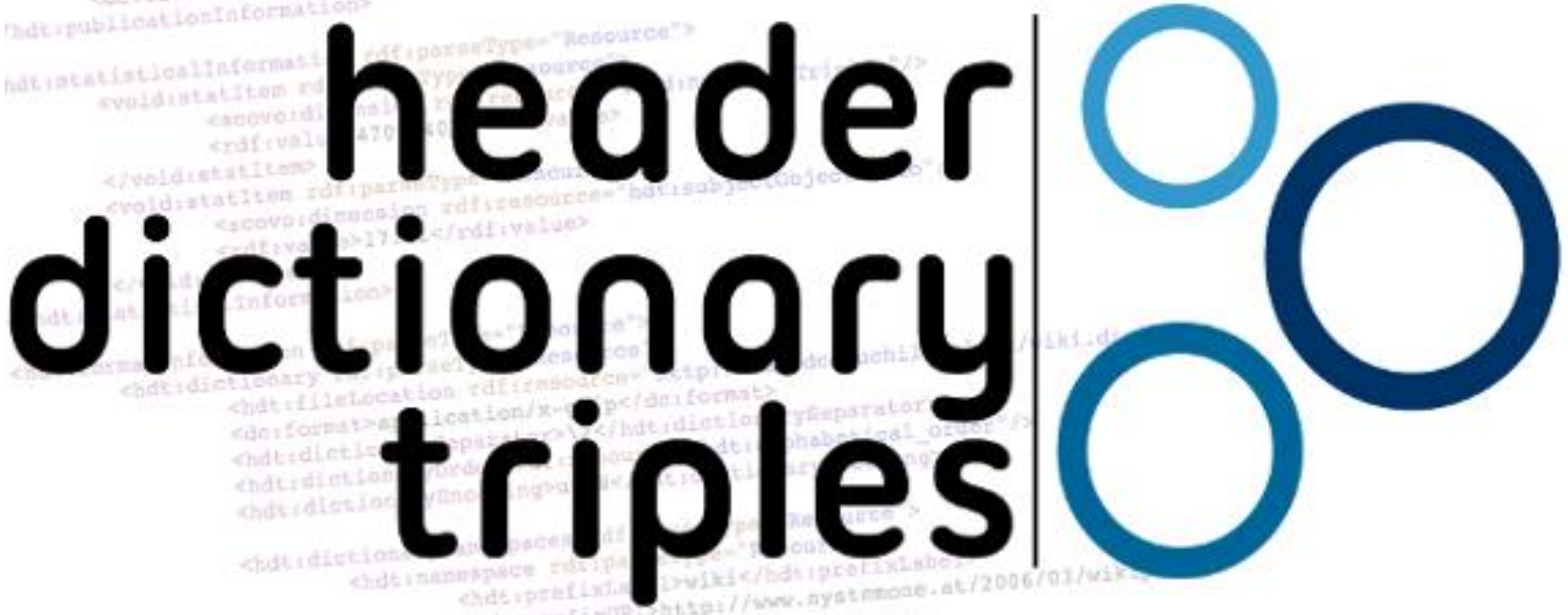
Accesso ai dati efficiente

- Definizione di **interfacce standard** (specie in settori come l'istruzione o quello finanziario).
- **Gestione di problematiche** come **accessi concorrenti**, garantire un **accesso ai dati multi-piattaforma e multi-device**.
- *Ciò significa non diminuire la disponibilità dei dati e migliorare l'esperienza utente*.

Visualizzazione scalabile

- Una *query può restituire un numero piccolo ma anche enorme di risultati*, un buon requisito è quello di avere dei **sistemi/tool che premettano una visualizzazione scalabile** e chiara in entrambe le situazioni (es. la *matrice di adiacenza 3D per RDF*).

RDF-HDT Library



RDF-HDT Library

- **RDF è uno standard per la descrizione delle risorse sul web**, utilizzando delle asserzioni formate da:

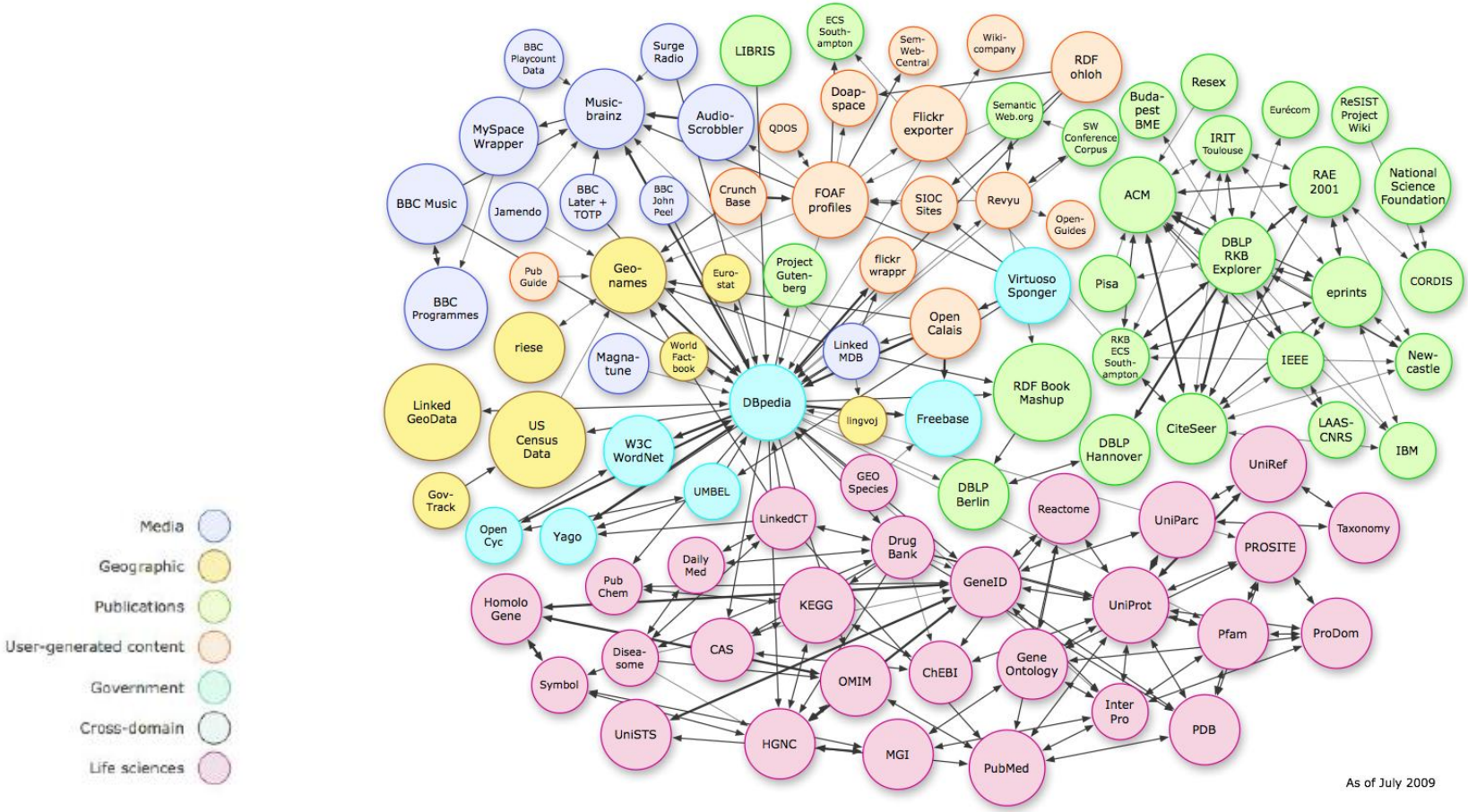
  Soggetto – Predicato – Complemento

- Un **modello RDF** può essere rappresentato da un **grafo diretto**.

- Un grafo RDF è rappresentato fisicamente da una serializzazione **RDF/XML – N-Triple – Notation3**

RDF-HDT Library

- Aumento di dati sul web **con dataset RDF sempre più grandi** relativi a differenti contesti.



RDF-HDT Library – Problemi scalabilità Dati WEB

Pubblicazione

- ***Mancanza di metodologie/norme*** per la *pubblicazione di dati su larga scala.*
- Uso di *vocabolari per dati interconnessi* e di *mappe semantiche*.

Scambio

- **I principali formati RDF** (*RDF/XML – Notation3 – Turtle*) hanno una **impostazione *document-centric*** (“verbosa”)
- Utilizzo di ***“compressori universali”*** (*gzip, targz*) per ridurne le dimensioni e favorirne il trasferimento.

Utilizzazione (query)

- **Costi post-elaborazione** dovuti a **problematiche di decompressione e indicizzazione dei dati** (*RDF store*).

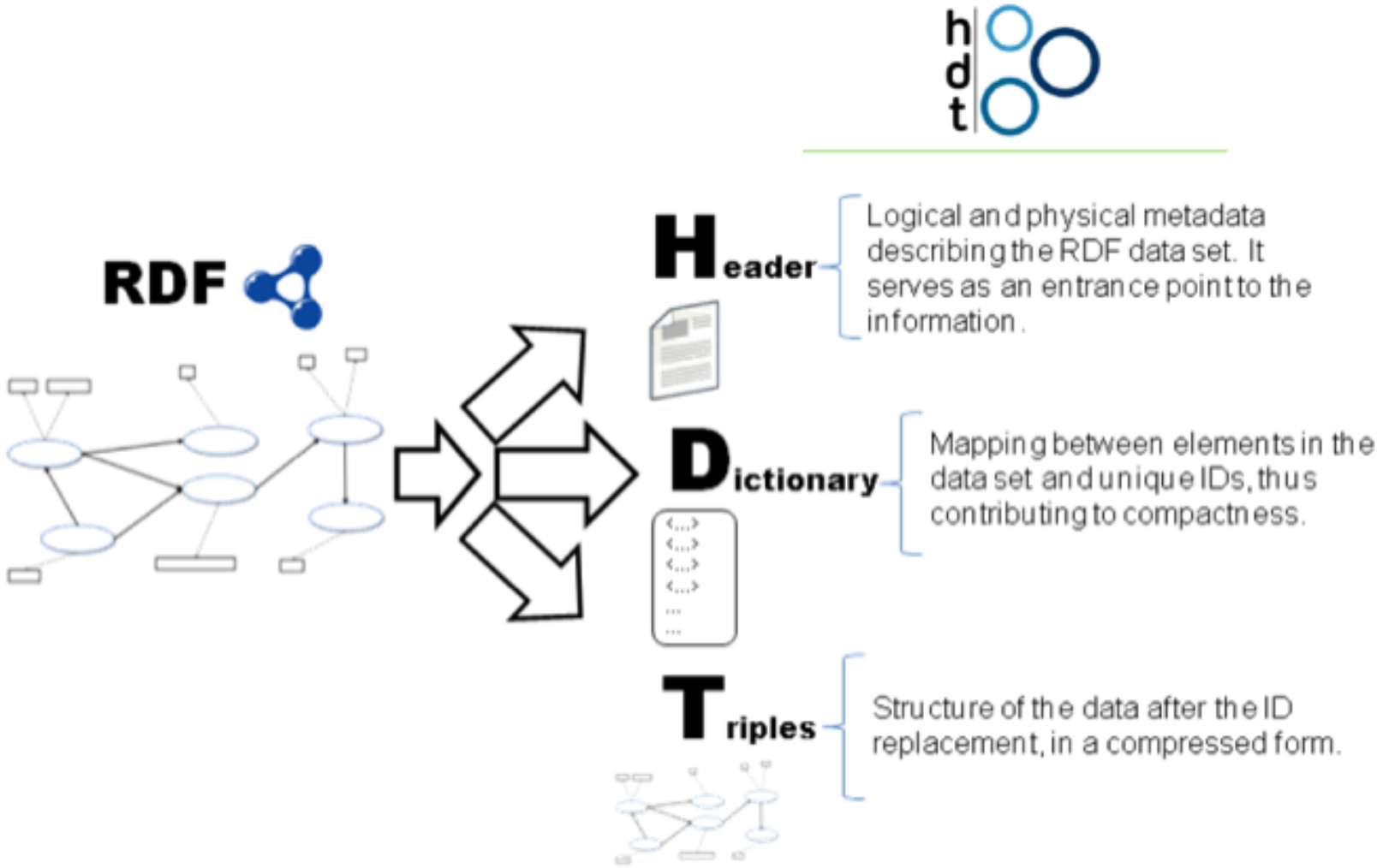
RDF-HDT Library

- **HDT** (***Header, Dictionary, Triples***) è una ***rappresentazione compatta per i dataset RDF*** con l'obiettivo di ridurre la loro verbosità.

- Un **grafo RDF** è rappresentato da **3 componenti logiche**:
  - ***Header***
  - ***Dictionary***
  - ***Triples***

- *Questa struttura lo rende un **formato ideal**e per memorizzare e condividere **dataset RDF sul web**.*

RDF-HDT Library



RDF-HDT Library - Header

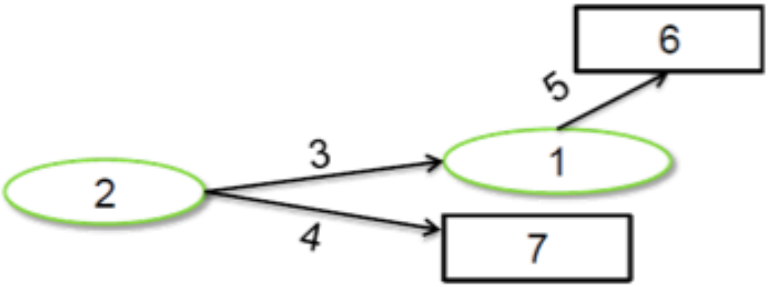
HDT migliora il **valore dei metadati**.

- L'**Header *è esso stesso un grafo RDF*** contenente informazioni relative a:
  - ***Provenienza*** (*versione, provider, data di pubblicazione*).
  - **Statistiche** (*dimensione, qualità, vocabolari*)
  - **Organizzazione fisica** (*sottoparti, dislocazione dei file*).
  - **Altri tipi di informazioni** (*proprietà intellettuale, firma.*)

RDF-HDT Library – Dictionary + Triples

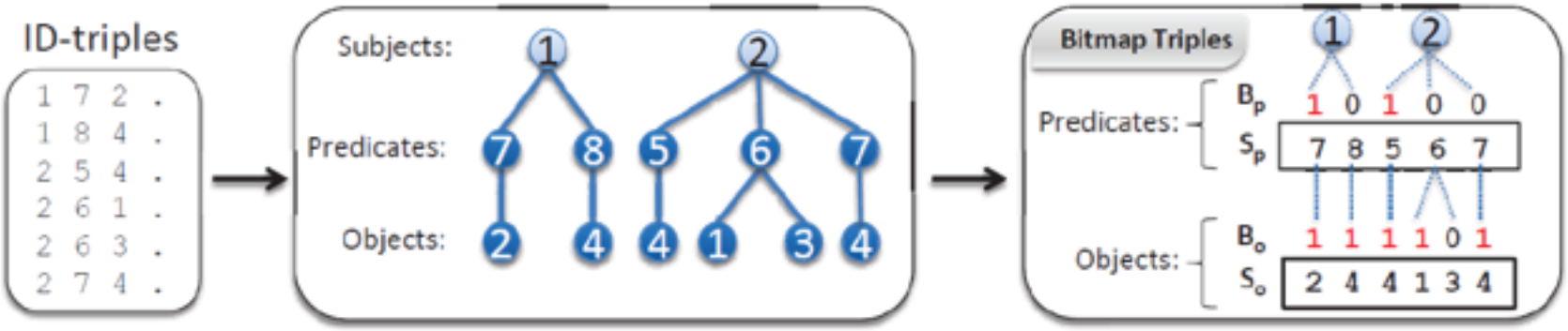
- Viene realizzato un **mapping** tra ogni termine utilizzato in un dataset e un **ID univoco** (*intero*).
- I termini **lunghi/ridondanti** sono rimpiazzati dal loro **identificativo** (ID).
- La **struttura del grafo** può essere ***indicizzata e gestita come un flusso di interi.***
- Con un **dizionario di serializzazione** avanzata si *ottengono compattezza e performance sui consumi*.

1	<http://books/author33>
2	<http://books/book21>
3	dc:author
4	dc:title
5	foaf:name
6	“Pablo Neruda”
7	“Spain in the Heart”



RDF-HDT Library – Triples

- Gli **ID-Triples** *rappresentano in* ***modo compatto*** *il grafo*.
- Gli **ID-Triple** sono la *componente* ***chiave*** *per* ***accedere e realizzare query*** *sul grafo RDF.*
- La **componente Triple** può fornire un **indice compatto** per realizzare delle query base.



- In questo modo si ottiene un'**ottimizzazione spaziale** e **performance efficienti nelle operazioni primitive**.

RDF-HDT Library Pro

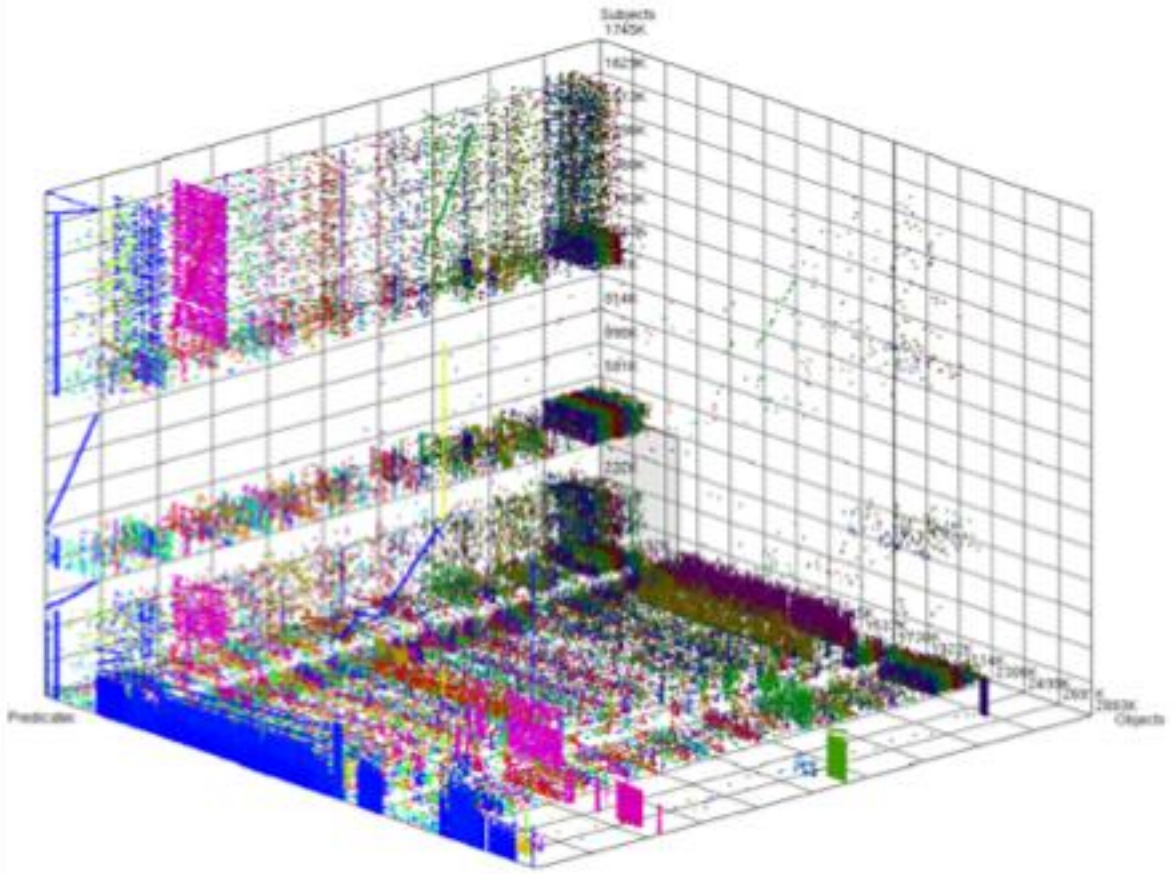
- HDT migliora lo **scambio dei dati**

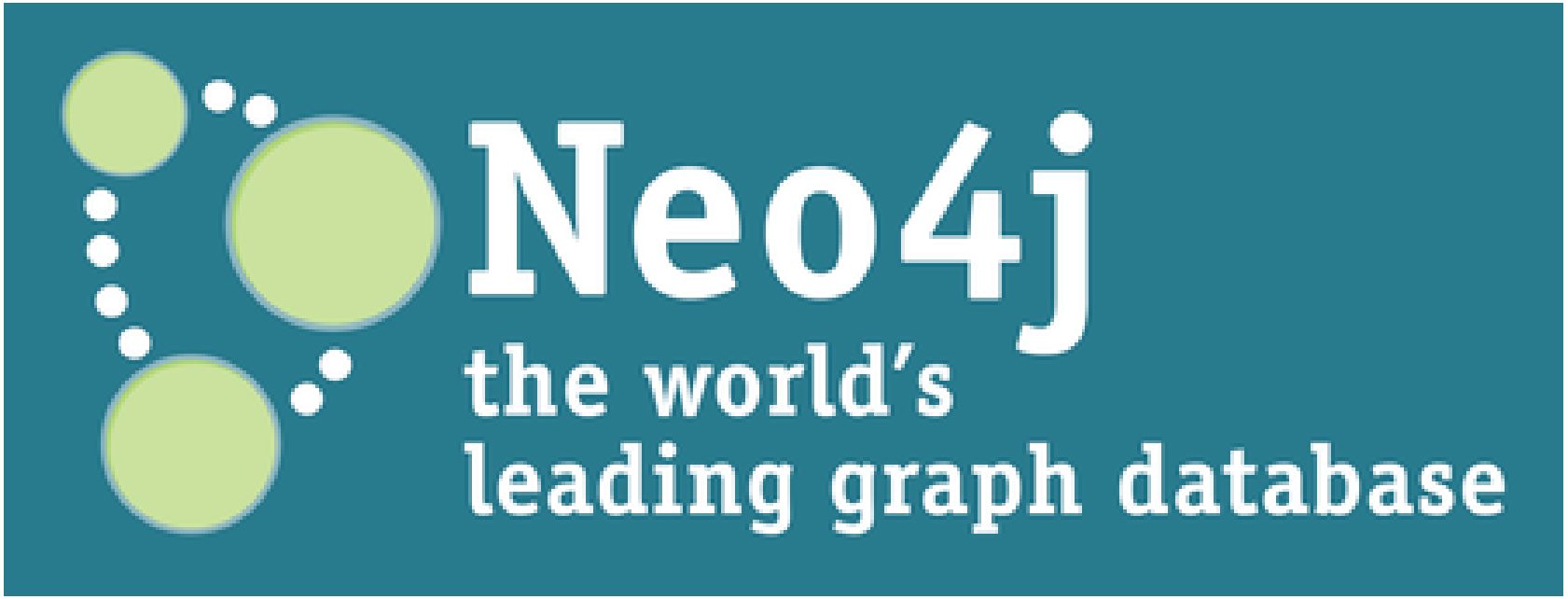
Dataset	Triples (millions)	Size (GB)	Compression (MB)		
			gzip	bzip2	HDT
wikipedia	47.0	6.88	491.04	360.01	**230.48**
dbtune	58.9	9.34	924.85	630.28	**462.31**
uniprot	72.5	9.11	1233.25	739.76	**481.34**
dbpedia-en	232.5	33.12	3513.58	2645.36	**2176.54**

- Con opportune **tecniche di compressione** messe a disposizione di riescono ad **ottenere risultati migliori** che **favoriscono trasferimenti di dati più veloci** ed efficienti.

RDF-HDT Library – Visualizzazione

HDT-it!




Neo4j
the world's
leading graph database

- **Neo4J** è un **Graph Database** *open source*.
- E' disponibile in **3 versioni:**
  - ***community edition*** (versione di base, free).
  - ***advanced*** (provvista di monitoring).
  - ***enterpricse*** (fornisce alta disponibilità grazie ad un cluster).

Le **caratteristiche principali** di *Neo4J* sono:

- ***Garantisce le proprietà ACID delle transizioni***.
- ***Alta scalabilità*** (nell'ordine di *miliardi di nodi*).
- ***Ottime performance nell'attraversare i nodi.***

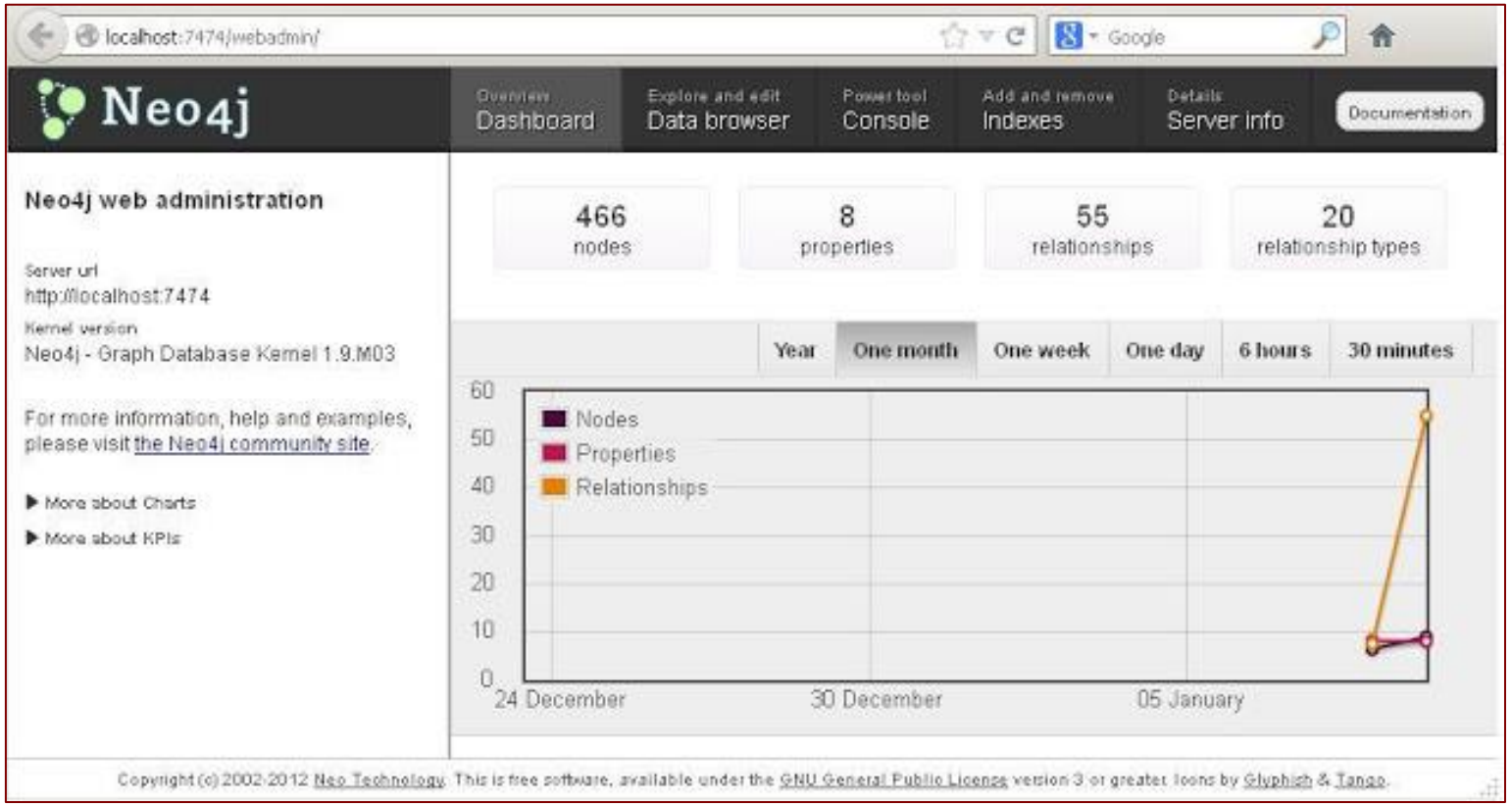
Neo4J – Accesso ai dati

Neo4J fornisce **3 modalità differenti di *accesso al database***:

- ***Interfaccia web***
- ***Console da riga di comando***
- ***API Java***

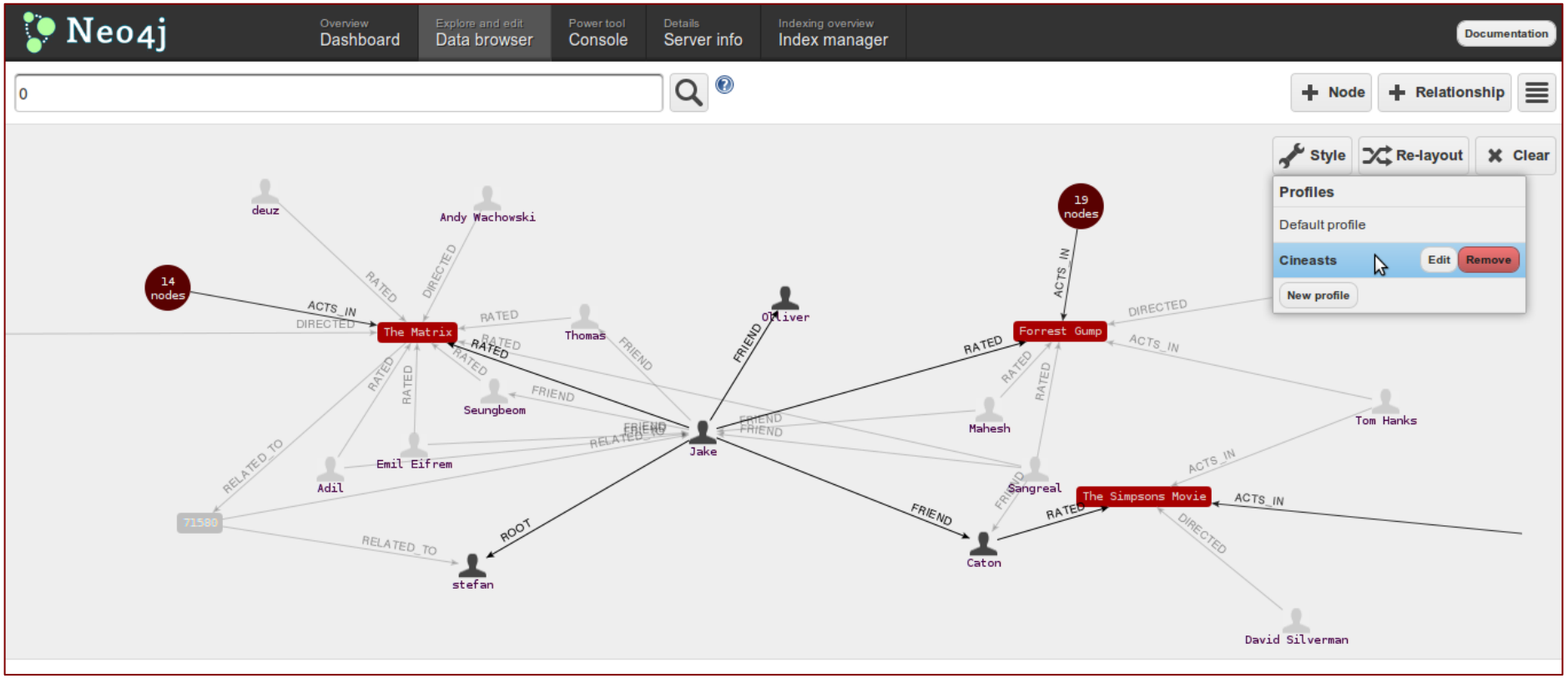
Neo4J – Interfaccia web

- E’ possibile effettuare *attività di monitoraggio* utilizzando la tab ***Dashboard*** oppure la sezione ***Server Info***.



Neo4J – Interfaccia web

- E' possibile accedere alla ***Console***, e tramite il ***Data browser*** realizzare la *navigazione e/o la modifica dei dati* (aggiunta/eliminazione di nodi o relazioni, oppure definizione di proprietà legate ai nodi).



Neo4J – Interfaccia web

- Il ***Data browser*** funziona attraverso le **query Cypher**, cioè il *linguaggio di interrogazione fornito da* ***Neo4J***.

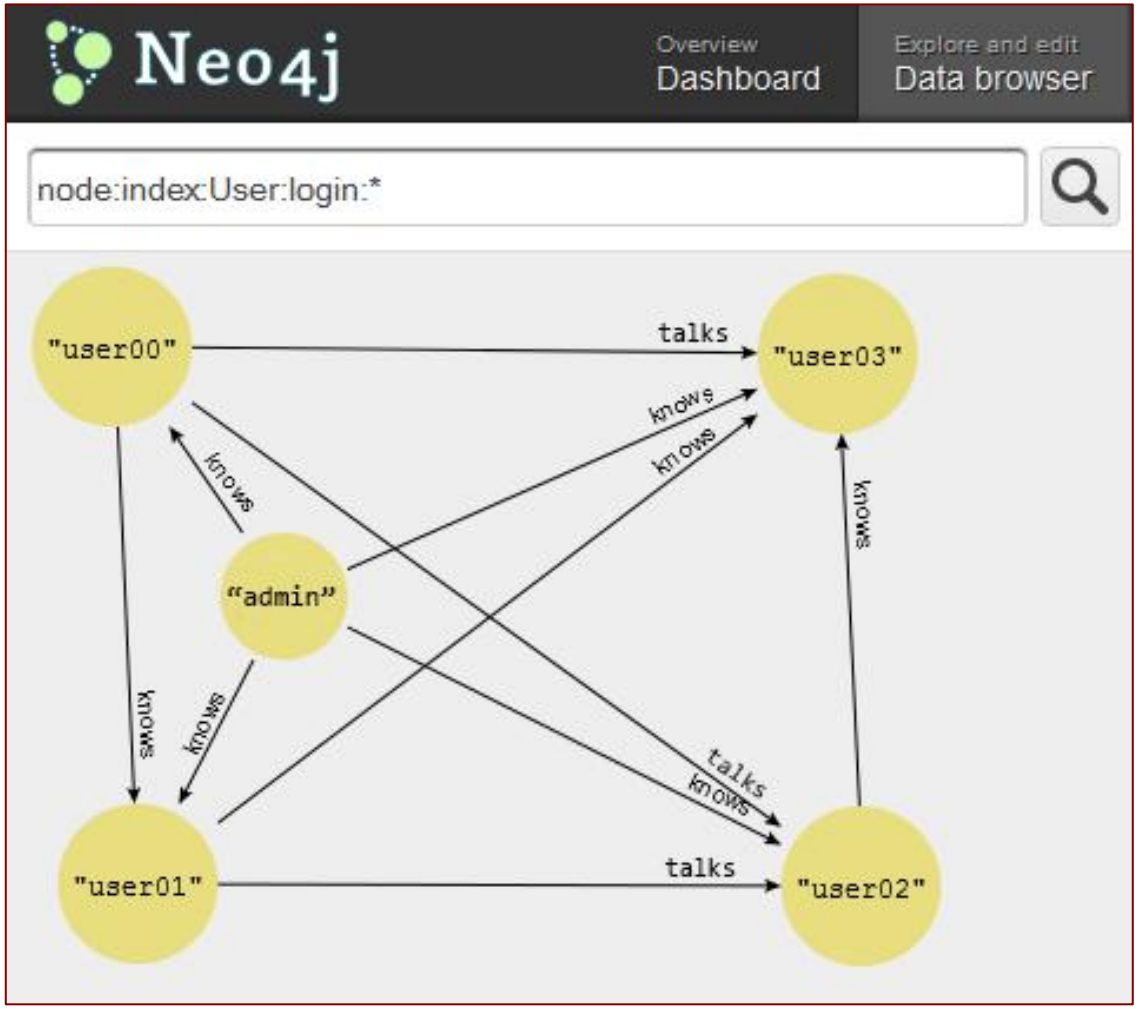
- ***Cypher*** è un **linguaggio dichiarativo**, semplice da apprendere (adatto sia a sviluppatori che analisti), *che restituisce una* ***lista di nodi e relazioni*** *oppure una loro* ***rappresentazione grafica***.

- Le **query Cypher** possono essere **eseguite** da:
  - ***Data browser***
  - ***Riga di comando***
  - ***API HTTP REST***
  - ***API Java***

Neo4J – Query Cypher

Consideriamo un **grafo** visualizzato all'interno del ***Data browser*** di **Neo4J**

*Quali **query** è possibile eseguire?*



Neo4J – Query Cypher

Alcune ***possibili query*** sono:

Query	Descrizione
`START n=node(1)` `RETURN N`	Restituisce il nodo1. Le query cominciano con la parola chiave START che definisce il punto iniziale della query, cioè il nodo di partenza.
`START n=node(*)` `RETURN n`	Restituisce tutti i nodi.
`START n=node:node_auto_index(nome="user01")` `RETURN n`	Restituisce il nodo con proprietà nome uguale a "user01". La query usa un indice automatico attivato nella shell.

`START n=node(1)`
`MATCH n-[r:knows*]->conoscenti`
`RETURN conoscenti`
	Restituisce I nodi legati al nodo 1 dalla relazione "knows". L'uso dell'asterisco fa si che la query sia applicata in modo ricorsivo.

Neo4J – Query Cypher

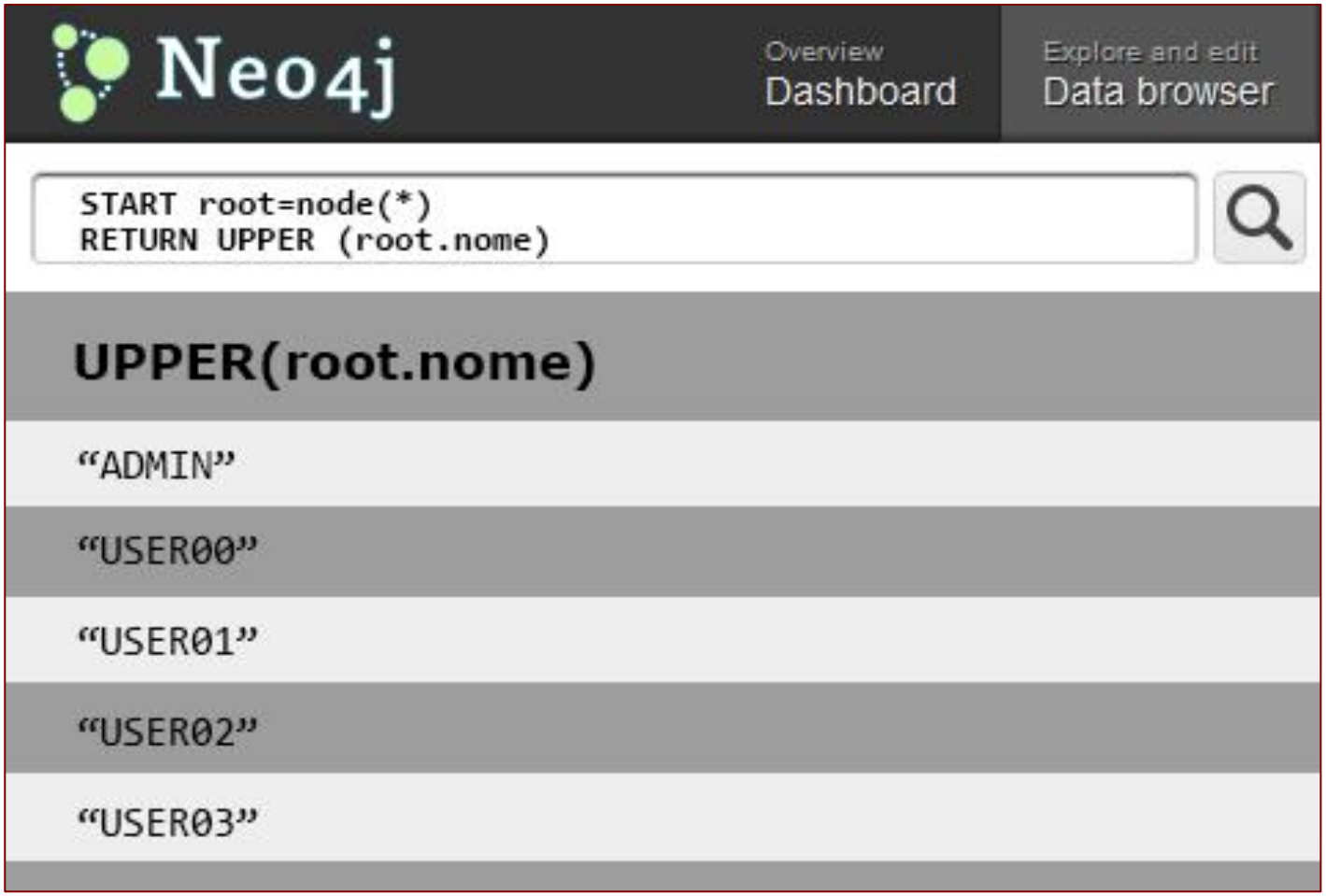
Altre ***possibili query*** sono:

Query	Descrizione

`START n=node(1)`
`SET n.permessi="yes"`
`RETURN N`
	Imposta la proprietà permessi con valore "yes"

`START n=node(*)`
`RETURN UPPER (n.nome)`
`ORDER BY n.nome`
	Restituisce per tutti i nodi la proprietà nome, in maiuscolo e ordinati. Sono offerte molte funzioni predefinite per lavorare con stringhe, numeri nodi e collection di nodi.

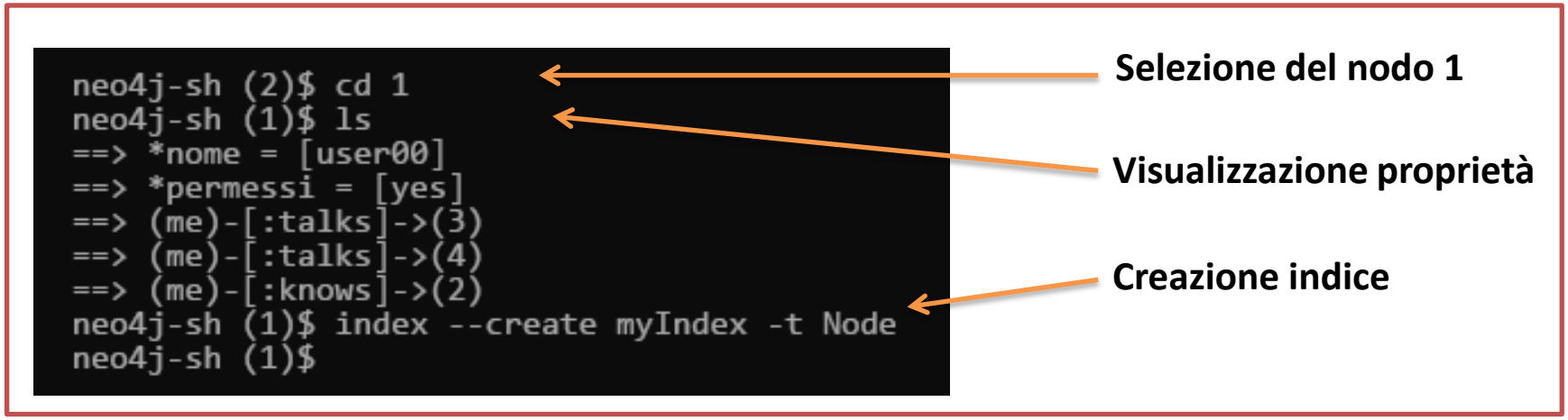
Un esempio di risultato di una query è il seguente



Neo4J – Riga di comando e API

Alcune delle **possibili operazioni** *eseguibili da* ***riga di comando*** sono:

- **Navigazione dei nodi** (*comandi* ***cd*** e ***ls***).
- **Modifica dei nodi** (*comandi* ***set, rm, rmnode, mkrel***…).
- **Esecuzione di query Cypher** (*comando* ***cypher***).
- **Creazione o rimozione di indici** (*comando* ***index***).



- Le **API Java** e **API HTTP REST** *danno l'accesso completo al DB*, consentendo tutte le operazioni di amministrazione e gestione dei dati (*inserimento/rimozione nodi, creazione relazioni…*)

Panoramica sulle tecnologie Big Data

Considerando le tecnologie e le soluzioni adatte a lavorare con i **Big Data** bisogna prestare attenzione a **4 classi di aspetti fondamentali**:

- **Aspetti *di Ingestion e Mining***

E’ opportuno avere dei tool che consentano una **visualizzazione scalabile dei risultati** ottenuti dall’analisi dei Big Data.

Aspetti di Ingestion e Mining

- ***Mining e Ingestion*** sono *due delle caratteristiche chiave* nel campo dei Bid Data, infatti si *tratta di individuare un* ***compromesso*** *tra*:
  - La ***velocità nell'operazione di data ingestion***.
  - La ***capacità di rispondere velocemente alle query***.
  - La ***qualità dei dati*** in termini di *aggiornamento*, *coerenza* e *consistenza*.

- Questo **compromesso impatta sulle scelte di progetto** di questi sistemi (es. ***OLTP vs OLAP***).

- Ad esempio alcuni *filesystem* sono *ottimizzati per le operazioni di lettura, altri per quelle di scrittura*; ma generalmente il carico di lavoro di questi sistemi prevede un **mix di entrambe le operazioni**.

Tipologia di indicizzazione

- Per **velocizzare l'ingestion** dei dati i record si potrebbe pensare di adottare un approccio che preveda di scriverli in una memoria **cache** oppure applicare delle **tecniche di compressione dei dati avanzate**.

- Utilizzare dei **metodi di indicizzazione differenti** può **migliorare la velocità nelle operazioni di ricerca** a *scapito solo di un aumento dei costi dovuti allo storage aggiuntivo* (ovviamente parlando di Big Data, quindi di grandi Volumi di dati, l'occupazione spaziale è un parametro a cui bisogna sempre fare attenzione).

Aspetti di Ingestion e Mining

Dati temporanei

- Alcune **tipologie di analisi** dei dati sono esse stesse *grandi e complesse*: le analisi e le operazioni computazionali **creano *enormi quantità di dati temporanei*** che devono essere gestiti opportunamente per evitare problemi di memoria.

- In altri casi è bene fare alcune **statistiche sulle informazioni a cui si accede più frequentemente**, e quindi creare degli ***appositi sistemi di cache*** con file temporanei al fine di ottimizzare il processo di interrogazione e velocizzare la resa dei risultati all'utente.

Tecnologie di acquisizione dati

Un **elenco non esaustivo** degli **strumenti** che permettono ***l'acquisizione dei big data*** potrebbe essere il seguente:

Categoria	Strumenti
API	**API dei principali motori di ricerca (Google, Bing, Yahoo!)** **Twitter API** **Facebook API**
Web scraping	**cURL** **Apache Tika**
Stream e CEP	**Apache Flume** **Microsoft StreamInsight**
ETL	**Sqoop** **Pentaho Data Integration (PDI) - Kettle**

Nota: *Apache Tika così come gli ETL offrono anche altre funzionalità tanto da partecipare anche fase di integrazione ed elaborazione dei dati.*

Tecnologie di acquisizione dati - Twitter API

- **Twitter** è la ***piattaforma di microblogging*** più utilizzata, basata sullo *scambio/pubblicazione di brevi messaggi* (max 140 caratteri).
- Oltre al **testo** i messaggi possono includere **link a foto**, **video**, **risorse** (anche di altri social network).
- L'**accesso a Twitter** può avvenire da **dispositivi mobile** come *smartphone e tablet*, oppure **tradizionalmente** da *PC e Notebook*.

Parola chiave	Descrizione
Tweet	**Messaggio di max 140 caratteri.**
Follower	**Persona che decide di visualizzare I tweet di un certo utente.**
Retweet	**Tweet con cui utente inoltra il tweet di un altro utente.**
Hashtag	***#topic*** **è un formato per definire un argomento di interesse.**
@reply	**Tweet indirizzato all'utente *@nome_utente***

Tecnologie di acquisizione dati - Twitter API

Gli **sviluppatori tramite le API** (documentate *http://dev.twitter.com*) **interagiscono con la piattaforma** per:

- *Creare nuovi tweet.*
- *Rispondere o effettuare retweet.*
- **Effettuare ricerche** o **intercettare i nuovi tweet relativi ad uno specifico topic.** Funzionalità detta ***streaming***, importante per l'analisi.

OSSERVAZIONE

- Le **normali API di ricerca** si basano su una **connessione REST HTTP**, cioè utilizzano un ***protocollo di richiesta-risposta***.
- Le **Twitter Streaming API** *aprono una **connessione** e la tengono **aperta** raccogliendo i dati man mano che questi sono disponibili*.

Tecnologie di acquisizione dati - Twitter API

Twitter mette a disposizione **3 tipologie di stream**:

- ***User Stream –*** recupera i tweet pubblici di un utente.
- ***Site Tweet –*** recupera i tweet pubblici da più utenti.
- ***Public Stream –*** recupera i tweet pubblici da qualsiasi utente, magari definendo un argomento di ricerca . *E' quindi la più utilizzata per fini analitici.*

La ricerca *Public Stream* può essere eseguita in **3 modalità**:

- ***Firehose*** – senza filtri
- ***Sample*** – a campione
- ***Filter*** – indicando specifici filtri.

Tecnologie di acquisizione dati - Twitter API

La ricerca ***Public Stream*** in modalità ***filter***, è sicuramente la più interessante perché permette di *parametrizzare la raccolta dei tweet.* Alcuni paramentri interessanti sono:

- ***Track –*** con la quale è possibile specificare una o più parole chiave di ricerca (separate dalla virgola). Es track=vino,chianti,rufina.
- ***Location –*** con la quale è possibile impostare una ricerca geografica definendo le coordinate di latitudine e longitudine (sempre separate dalla virgola).
- Una **ricerca** di questo tipo **restituirà** un ***elenco progressivo di tweet***. Sarà visualizzata la ***struttura completa***, cioè oltre al **testo** compariranno una serie di **importanti metadati** come: *id progressivo*, *utente, data e ora di creazione, coordinate* (se abilitate) etc.

Tecnologie di acquisizione dati - Twitter API

- **L'accesso alla Twitter API** è possibile in diversi linguaggi di programmazione, previa **autenticazione** mediante il **modello OAuth.**
- Un'applicazione che utilizza queste API **riceve i tweet** sotto forma di **oggetti JSON**.
- L'**applicazione** avrà al suo interno il **codice necessario a memorizzare i tweet** (risultato della ricerca) in un *database, file o anche su HDFS di Hadoop.*

OSSERVAZIONI

- Esiste un **limite massimo di tweet scaricabili** in un certo intervallo di tempo da un account.
- Per scaricare **tweet real-time** si utilizzano le ***Streaming API***, se invece si vuole recuperare **tweet pubblicati in passato** si devono utilizzare le ***Search API***.

Tecnologie di acquisizione dati - Facebook API

- **Social Network** nato nel **2004**, che oggi conta ***più di 1 Miliardo di utenti*** (*collegati sulla base di una relazione di amicizia o di like*)**.**
- Ogni **utente** ha un proprio **profilo** in cui sono *raccolte informazioni* come *dati anagrafici, interessi, immagini* etc. Il profilo utente può avere **diversi livelli di accesso**:
  - ***Pubblico***
  - ***Amici***
  - ***Privato***
  - ***Personalizzato*** (*Gruppi, liste di contatti etc*).
- I ***dati pubblici*** (cioè visibili a tutti) sono: *nome – nome utente - sesso - immagine del profilo – foto di copertina – ID utente*; per poter visualizzare gli altri occorre che sia soddisfatta la relazione di amicizia.

Tecnologie di acquisizione dati - Facebook API

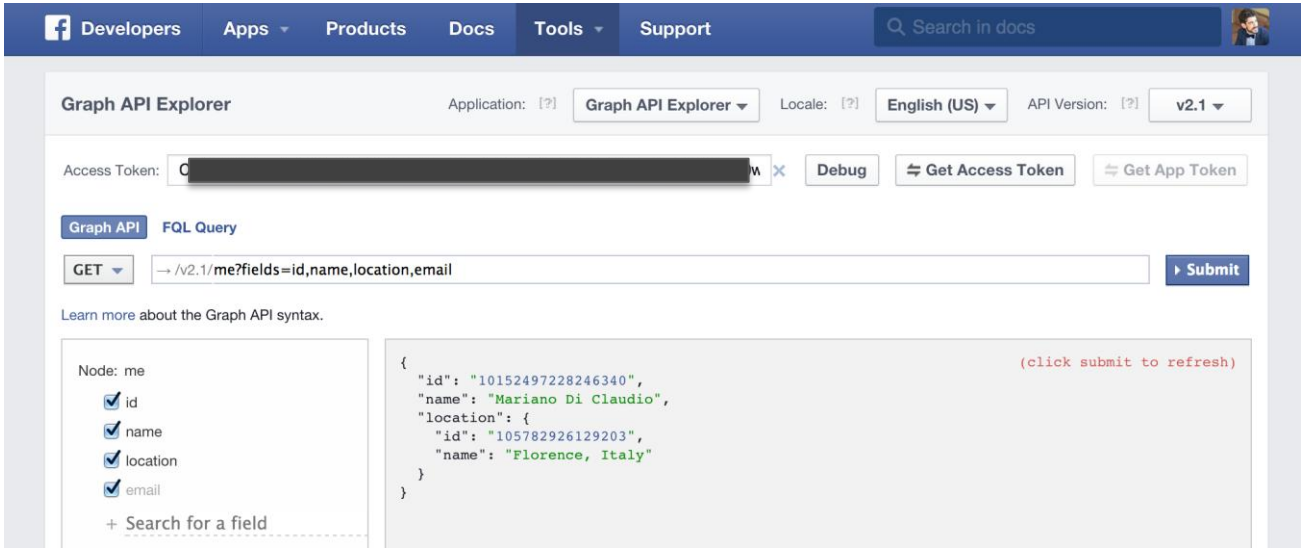
- Una delle **API** più interessanti è la **Graph API**, che permette di **eseguire interrogazioni e raccogliere dati** dal *social graph* attraverso una **serie di funzionalità HTTP**.
- Il ***social graph*** è il *modello di rappresentazione degli "oggetti" che popolano Facebook*, è infatti costituito da un **insieme di nodi interconnessi** che *rappresentano amici, interessi, contenuti etc.*

- La **Graph API** è disponibile in *toolkit per iOS, Android, PHP, JavaScript* forniti da FB e di altre toolkit esterni per altri linguaggi.
- http://developers.facebook.com/tools/explorer è l'indirizzo per testare la Graph API (Explorer).

Tecnologie di acquisizione dati - Facebook API

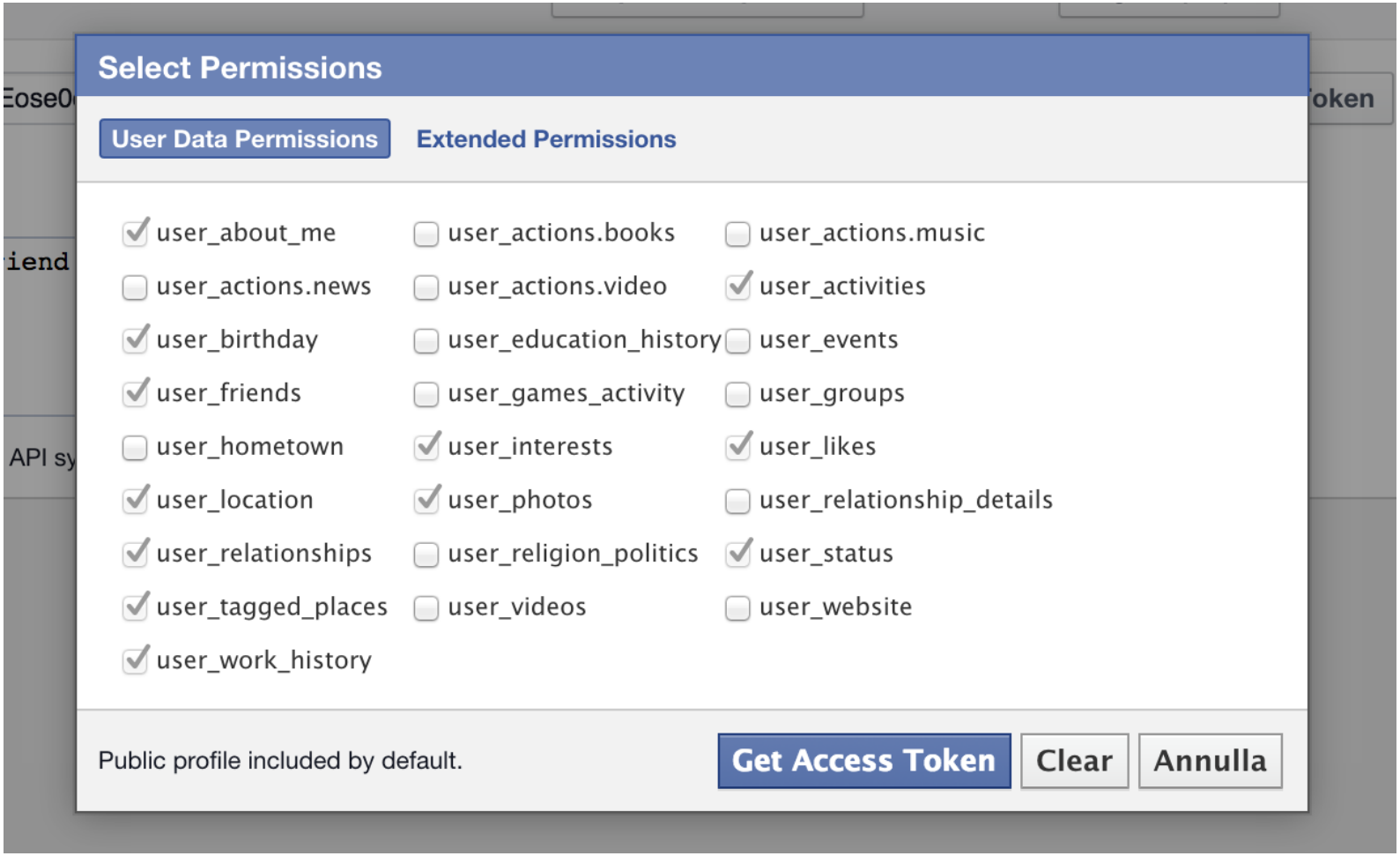
Per eseguire delle interrogazioni attraverso il Grah API Explorer si deve:

- ***Effettuare l'accesso con il proprio account FB.***
- ***Ottenere un Access Token***, tramite il pulsante *"Get Access Token"*.
- **Selezionare** quali **permessi sui dati** si vuole ottenere (*visualizzazione di like, interessi, foto, informazioni di localizzazione, eventi etc*.). Saranno visualizzabili solo le informazioni pubbliche o quelle legate al tipo di relazione con un certo utente.



Tecnologie di acquisizione dati - Facebook API

Un esempio della finestra per selezionare i permessi è il seguente



Tecnologie di acquisizione dati - Facebook API

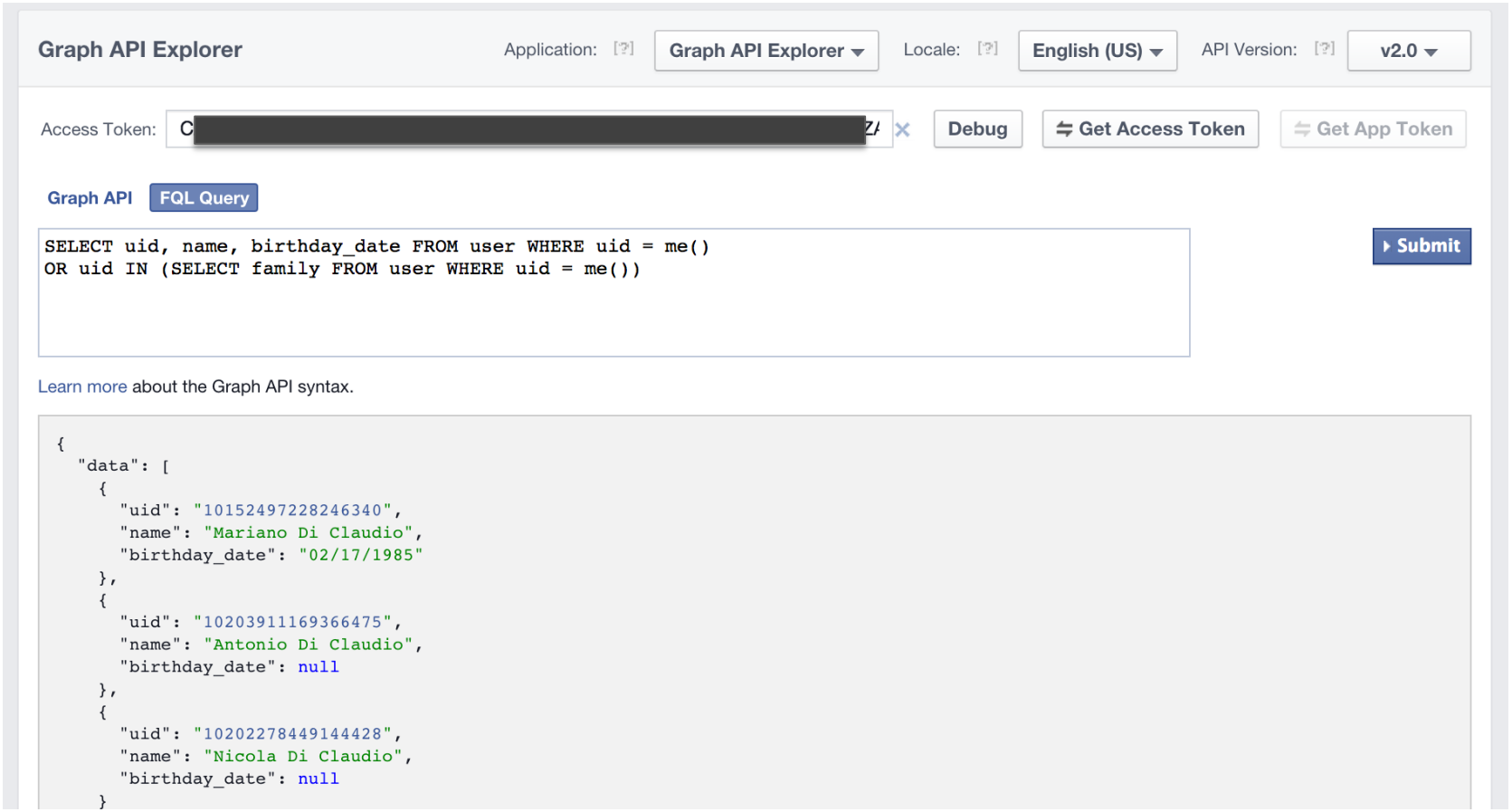
- **Il risultato di un interrogazione** nel Graph API Explorer è un oggetto **JSON**.
- Fuori dall'Explorer **una query Graph API** *si può eseguire andando a digitare nel browser un indirizzo* in cui si specificano nome utente o ID e i paramentri dell'interrogazione, esempio:

  http//graph.facebook.com/nome.cognome?fileds=id,name

- Le **Graph API** possono essere realizzate anche utilizzando uno **specifico linguaggio FQL** (*Facebook Query Language*), **linguaggio SQL-like** (fino alle versione 2.0).
- Le **query FQL** possono essere **eseguite** o **tramite il Graph Explorer**, oppure **direttamente da browser** tramite **una chiamata HTTP.**
- **FQL** non supporta JSON, ma sono permesse le **subquery**, cioè le *query annidate in cui la query più esterna sfrutta i risultati di quella interna.*

Tecnologie di acquisizione dati - Facebook API

- Alcune **tabelle** interrogabili da FQL sono ***user, friend, group, photo, url_like*** (l'elenco completo dovrebbe essere disponibile online).



Tecnologie di acquisizione dati - Web Scraping

- Il termine ***Web Scraping*** si riferisce ad un *insieme di tecniche per* ***l'estrazione automatica del contenuto*** *dai* ***siti web su HTTP*** (senza l'uso di API ufficiali).

- **L'obiettivo finale** è quello di ***trasformare questo contenuto in informazione strutturata*** utilizzabile in altri contesti.

cURL

- ***Utility da riga di comando*** per il **trasferimento dati**; utilizza la sintassi URL.
- **Supporta numerosi protocolli** tra cui FTP, HHTP, HTTPS, POP3, IMAP,FILE etc.

Apache Tika

- Strumento scritto in Java per ***identificare ed estrarre metadati e testo*** da numerose tipologie di documenti come XML, HTML, PDF, MS Office e OpenDocument, EPUB, ZIP etc.
- Adatto sia a fonti esterne (Internet), che fonti interne (documenti aziendali).

Tecnologie di acquisizione dati - Web Scraping

- Con **Stream e CEP** (*Complex Event Processing*) si intendono un i*nsieme di tecnologie ideate per* ***tracciare e analizzare flussi di dati***.

- I **dati** solitamente **provengono da più fonti** e spesso vengono **combinati e analizzati in tempo reale**.

Apache Flume

- E' un sistema con un'architettura semplice e flessibile adatta a trasferire grandi quantità di dati da fonti di vario genere verso HDFS o database.
- Usato per recuperare dati di log da web server, aggregarli e salvarli su HDFS.

Microsoft StreamInsight

- Piattaforma per lo **sviluppo di applicazioni CEP con il framework .NET**
- Consente la **gestione di un numero elevato di eventi**, **l'analisi di dati** e **la definizione di pattern e trend in tempo reale** (*sensor data, clickstream, etc.*).

Tecnologie di acquisizione dati - ETL

- **ETL** identifica *un insieme di tool e tecniche* per realizzare un **processo** di ***Estrazione, Trasformazione e Caricamento*** di dati provenienti da diverse fonti un un sistema di sintesi (database, data warehouse, data mart).

Sqoop

- E' un **progetto di Apache** che permette il **trasferimento di dati tra DB relazionali e Hadoop**.
- Si utilizza Sqoop per **caricare inizialmente i dati** nel filesystem distribuito di Hadoop **(HDFS)**, questi poi **vengono elaborati** con programmi che implementano **MapReduce**, ed infine i risultati vengono **salvati su un RDBMS**.

Pentaho Data Integration (Kettle)

- Progetto open source che ha l'obiettivo di facilitare i processi ETL utilizzando un approccio *"metadata-driven".*
- Uno dei punti di forza è ***Spoon,*** il tool grafico che aiuta la creazione dei processi ETL (job e trasformazioni) con ***il drag&drop***.

Valutazione di alcune soluzioni esaminate

- ArrayDBMS
- CouchBase
- eXist
- Hadoop
- Hbase
- MapReduce
- MonetDB
- Riak
- Objectivity
- OpenQM
- MongoDB
- RdfHdt








Data Management

	ArrayDBM	CouchBase	Db4o	eXist	Hadoop	HBase	MapReduc	MonetDB	Objectivity	OpenQM	Rdf Hdt Library	RDF 3X	MongoDB	Riak
Data Dimension Huge	100 PB	PB	254GB	2^{31} doc.	10PB	PB	10PB	TB	TB	16TB	100mil Triples (TB)	50mil Triples	PB	TB
Traditional / Not Traditional	NoSQL	NoSQL	NoSQL	NoSQL	NoSQL	NoSQL	NoSQL	SQL	XML/ SQL++	NoSQL	NoSQL	NoSQL	NoSQL	NoSQL
Partitioning	blob	blob 20MB	blob	N	chunk	A	blob	blob	blob	Chunk 32KB	(N)	blob	Chunk 16MB	20MB (better 4-6MB)
Data Storage	Multidim. Array	1 document/concept	object database +B-tree	XML document + tree	Big Table	Big Table	N	BAT	Classic Table in which define +models	1file /table (data+dictionary)	3structures,RDF graph for Header	1Table + permutations	Memory-Mapped Storage Engine	modular, extensible local storage system

Data Management

	ArrayDBMS	CouchBase	Db4o	eXist	Hadoop	HBase	MapReduce	MonetDB	Objectivity	OpenQM	Rdf Hdt Library	RDF 3X	MongoDB	Riak
Data Model	Multi dime nsio n	Docum ent Store	Obje ctDB	XML DB	Colum n Famil y	Colum n Famil y	CF, KV, OBJ, DOC	Ext SQL	Graph DB	Multi value DB	RDF	RDF	Document Store	KV docu ment orient ed
Cloud	A	Y	A	Y/A	Y	Y	Y	A	Y	N	A	A	Y	Y
Amount of Memory	redu ces	docum entis+ Bidime nsional index	Obje cts + inde x	Docu ments + index		Optim ized Use	N	Effic ient Use	Like RDBM S	No compr ession	-50% datas et	Less than data set	More than TDB, but hight compressio n with Fractal Tree Index	Googl e Snapp y with Level DB
Metadata	Y	Y	Y	Y	Y	Y	Y/A	opt	Y/A	N	Y	Y	Y	Y

Data Management

- Gestione di PB di dati
- NoSQL
- Chunk di 16MB (BSON)
- Auto-Sharding e Auto-failover
- Semplice il passaggio da SQL
- Supporto di soluzioni Cloud
- Elevata capacità di compressione dati grazie all'uso di Fractal Tree index
- Indici spaziali e supporto di query ad-hoc

Accesso ai dati, Rendering visuale – Mining/Ingestion

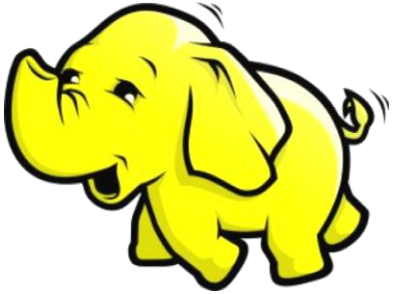
	ArrayDBMS	CouchBase	Db4o	eXist	Hadoop	HBase	MapReduce	MonetDB	Objectivity	OpenQM	Rdf Hdt Library	RDF 3X	MongoDB	Riak
Scalable visual rendering	N	N	N	N	A	N	N	N	N	Y	Y (3d adiacent matrix)	N	A	A
Visual rendering and visualization	N	N	Y	N	A	N	P	N	N	Y	Y	P	A	A
Access Type	Web-interface	Multiple point	Y (Interfaces)	Y (extend XPath sintax)	Sequential	Cache for frequent access	Sequential (no random)	Full SQL interfaces	Y (OID)	Y (concourrency)	Access on demand	SPARQL	http interface	http and solr-like interface

Accesso ai dati, Rendering visuale – Mining/Ingestion

	ArrayDBMS	CouchBase	Db4o	eXist	Hadoop	HBase	MapReduce	MonetDB	Objectivity	OpenQM	Rdf Hdt Library	RDF 3X	MongoDB	Riak
Type of indexing	Multi mensional-index	Y(incremental)	Y	B+-tree (XISS)	HDFS	h-files	Distributed Multileval-tree Indexing	Hash index	Y (function objectivity/SQL++ interfaces)	B-tree based	RDF-graph	Y (efficient triple indexes)	Single Field, Compound, Multikey, Geospatial, Text, Hashed	Primary key, secodndary index, MapReduce, Fractal Tree
Data relationships	Y	N	Y	Y	A	N	Y	Y	Y	Y	Y	Y	Y	Y

Accesso ai dati, Rendering visuale – Mining/Ingestion

- Tool esterni che permettono una visualizzazione scalabile dei dati.
- Accesso sequenziale
- Sistema di indicizzazione intelligente grazie alla presenza di HDFS
- Tool esterni per la gestione delle relazioni tra i dati



Data Analysis

	ArrayDBMS	CouchBase	Db4o	eXist	Hadoop	HBase	MapReduce	MonetDB	Objectivity	OpenQM	Rdf Hdt Library	RDF 3X	MongoDB	Riak
Statistical Support	Y	Y/A	N	N	Y	Y	N	A (SciQL)	Y	A	N	N	Y	Y
Log Analysis	N	N	N	N	Y	P	Y	N	N	N	N	Y	Y	Y
Semantic Query	P	N	P	A	N	N	N	N	N	N	Y	Y	N	N
Statistical Indicator	Y	N	N	N	Y	Y	N	N	Y	A	Y	Y(Performance)	A	Y
CEP (Active Query)	N	Y	N	N	N	N	N	N	N	N	N	N	P	N
Faceted Query	N	N	N	Y	N	A(filter)	Y	N	Y (Objectivity PQE)	N	Y	N	N	Y

Data Analysis

- Non supporta l'analisi dei Log
- Supporto alle query semantiche e alle Facet Query
- Supporta la definizione di indicatori per l'analisi statistica dei dati

Aspetti Architetturali

	ArrayDBMS	CouchBase	Db4o	eXist	Hadoop	HBase	MapReduce	MonetDB	Objectivity	OpenQM	Rdf Hdt Library	RDF 3X	MongoDB	Riak
Distributed	Y	Y	Y	P (hard)	Y	Y	Y	Y	Y	(-)	A	Y	Y	Y
High Availability		Y	Y	N	Y	Y	Y	N	Y	Y			Y	Y
Fault Tolerance	A	Y	Y	Y	Y	Y	Y	N	Y	N	A	A	Y	Y
Workload	Computation intensitive	Auto distribution		So high for update of enteir files	configurable	Write intensitive	configurable	Read dominated (or rapidly change)	More Possibility	Divided among more processes		optimized	read-dominated	high write and read performance
Indexing Speed	more than RDBMS	non-optimal performance	5 – 10 times more than SQL	High speed with B+Tree	A	Y	Y/A	More than RDBMS	High Speed	Increased speed with alternate key	15 volte RDF	aerodynamic	Changing with differet type of index	low latency request times, high throughput

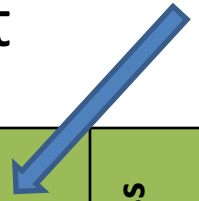
Aspetti Architetturali

	ArrayDBMS	CouchBase	Db4o	eXist	Hadoop	HBase	MapReduce	MonetDB	Objectivitv	OpenQM	Rdf Hdt Library	RDF 3X	MongoDB	Riak
Parallelism	Y	Y	Transational	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y
N. task configurable	N	Y	N	Y	Y	A	Y	N	Y	N	N	N	N	Y
Data Replication	N	Y	Y	Y	Y	Y	A	Y	Y	Y	A	A	Y	Y

Aspetti Architetturali

- Possibilità di effettuare operazioni di letture e scritture anche in condizioni di fallimento
- Fault tolerant
- Scalabilità orizzontale
- Scalabilità single-site molto buona
- Interfacce RESTful e HTTP
- Facilità di aggiungere nuovi nodi al Cluster
- Distribuzione automatica dei dati sul Cluster

Data Management

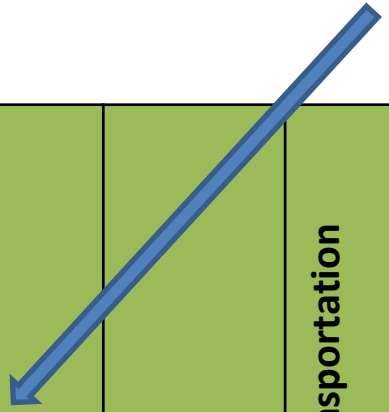


	Healthcare	Educational	Energy/Transportation	Social Network Internet Service Web Data	Financial/Business	Security	Data Analysis Scientific Research (biomedical…)
Data Dimension Huge	PB	300PB	TB	1000EB	PB	1Bil TB	PB
Traditional / Not Traditional	NoSQL	SQL	Both	Both	Both	Both	Both
Partitioning	Blob		Y/ Cluster		Cluster		Chuncks
Cloud	Y	P	Y	Y	P	Y	Y
Metadata	Y	Y	Y	Y	N	Y	Y

Data Management

- Scambio di ExaByte di dati
- Coinvolgono DB Tradizionali e NoSQL
- Supporto a Sistemi Cloud
- Uso elevato di Metadati

Accesso ai dati, Rendering visuale – Mining/Ingestion

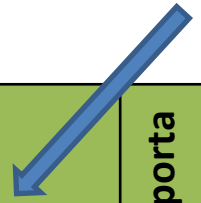


	Healthcare	Educational	Energy/Transportation	Social Network Internet Service Web Data	Financial/Business	Security	Data Analysis Scientific Research (biomedical…)
Scalable visual rendering	P	Y	N	Y	N	Y	Y
Visual rendering and visualization	Y	N	A	Y	A	N	Y
Access Type	Y (Concurrency)	Standard Interfaces	Get API	Open ad hoc SQL access	A	N	AQL (SQL array version)

Accesso ai dati, Rendering visuale – Mining/Ingestion

- Possibilità di integrazione di tool per la visualizzazione dei risultati
- Disponibilità di report visuali
- E’ raccomandata la possibilità di accesso concorrente ai dati

Data Analysis



	Healthcare	Educational	Energy/Transportation	Social Network Internet Service Web Data	Financial/Business	Security	Data Analysis Scientific Research (biomedical…)
Type of Indexing	N	Ontologies	Key Index Distributed	Y (Inverted Index, MinHash)	N	N	Array MultiDim
Data relationships	Y	Y	N	Y	Y	N	Y
Statistical Analisys Tools	Y	Y	A	Y	P	N	Y
Log Analysis	Y	Y	A	Y	Y	Y	A
Semantic Query	N	N	N	Y	N	N	N
Statistical indicator	Y	Y	P	N	P	P	Y
CEP (Active Query)	N	N	N	N	N	Y	N
Facet Query	N	Y	N	Y	Y	P	Y

Data Analysis

- Indicizzazione semantica
- Gestione delle relazioni tra i dati
- Benefit derivanti dall'uso di tool di l'analisi statistica
- Benefit derivanti dall'analisi dei Log
- Uso elevato di Facet Query
- Importanza di Indicatori statistici

Aspetti Architetturali

	Healthcare	Educational	Energy/Transportation	Social Network Internet Service Web Data	Financial/Business	Security	Data Analysis Scientific Research (biomedical…)
Distributed	Y	P	Y	Y	Y	Y	Y
High reliable	Y	Y	N	N	Y	N	Y
FaultTollerance	N	Y	Y	N	Y	Y	N
Indexing Speed	N	N	N	P	N	N	Y
Parallelism	Y	N	Y	Y	N	N	Y

Data Analysis

- Preferenza di un'Architettura Distribuita
- E' richiesta un'elevata Affidabilità
- Elevata velocità di Indicizzazione
- Richiesta l'elaborazione in parallelo

Tabella di comparazione

	ArrayDBMS	CouchBase	Db4o	eXist	Hadoop	HBase	MapReduce	MonetDB	Objectivity	OpenQM	RdfHdt Library	RDF 3X	MongoDB	Riak
Healthcare	Y		Y		Y	Y	Y	Y	Y	Y			Y	Y
Education				Y	Y								Y	
Energy/Transportation		Y	Y	Y				Y	Y					
Social Network	P	Y			Y	Y	Y	Y			Y	Y	Y	Y
Financial/Business	Y	Y	Y		Y			Y	Y				Y	Y
Security			Y	Y					Y	Y				Y
Data Analysis (Astronomy - Biology - Sociology)	Y		Y	Y			Y	Y	Y	Y	Y	Y	Y	Y