mokabyte.it

Il teorema CAP... in Brewer - MokaByte

Luca Vetti Tagliati

29-39 minuti

In questo quarto articolo della serie dedicata al teorema CAP, iniziamo l'esplorazione di MongoDB non prima però di aver ricapitolato alcune informazioni sui database NoSQL. In coerenza con gli obiettivi della serie, lo scopo di questo articolo non è la presentazione dettagliata di MongoDB, ma fornire una serie di informazioni finalizzate alla trattazione dettagliata dei compromessi derivanti dal teorema del CAP. Discussione che però avrà luogo nel corso del prossimo articolo.

Come siamo arrivati al Not Only SQL

Non più di qualche anno fa, l'egemonia dei database relazionali era di fatto incontrastata. In passato, intorno agli anni Ottanta, i database Object Oriented (**OODB**) avevano provato a contrastarne il dominio: sebben fossero state realizzate diverse soluzioni decisamente interessanti e di ottima fattura (**Gemstone**, **Gbase**, **Vbase**, **Jasmine**, **Objectivity/DB**, **ObjectStore**, **O2**, **POET**, **Versant**, **JADE** giusto per citarne alcuni), queste non riuscirono a fare breccia, rimanendo confinate in settori marginali, di nicchia [1].

La spinta del Web 2.0

Tale situazione ha subito un'interessante evoluzione grazie all'affermazione del Web 2.0. In particolare, importanti servizi Internet si sono trovati a far fronte a una crescita esponenziale dei dati da manipolare, dovuta essenzialmente all'espansione di Internet sia in termini di nuovi siti web e servizi offerti sia in termini della **popolazione utenti**. Basti pensare all'enorme **quantità** di pagine web che i **motori di ricerca** si sono trovati a dover immagazzinare per poter eseguire le proprie ricerche. L'espansione vertiginosa di Internet ha finito per generare **volumi di transazioni** tali da evidenziare tutti i limiti delle soluzioni tradizionali basate sui database relazionali. Intorno al 2008-9, **Google** con **Bigtable** e **Amazon** con **Dynamo** si sono trovati a far da apripista a un nuovo movimento denominato successivamente **NoSQL** (Not Only SQL).

Questo nome si deve a Carlo Strozzi [2] il quale lo utilizzò per primo, nel 1998, per denominare il suo database open-source. In particolare, il termine **NoSQL** evidenziava una delle sue caratteristiche primarie: l'assenza di un'interfaccia SQL. Tale nome fu poi ripreso da Eric Evans agli inizi del 2009 a seguito dell'iniziativa di Johan Oskarsson di organizzare un evento per discutere dei nuovi database distribuiti open source [3].

Da allora, nella comunità informatica hanno cominciato a comparire molte altre soluzioni operanti nello stesso segmento di mercato. Questa generazione iniziale di database ha trovato motivazione ed è cresciuta all'ombra del teorema CAP di Brewer.

Fine dei relazionali?

A questo punto ci si potrebbe domandare se il dominio dei **database relazionali** è veramente sotto attacco? Si tratta ovviamente di un'interrogazione retorica. Tuttavia nelle organizzazioni è sempre più frequente trovare database NoSQL che affiancano i tradizionali database relazionali in aree caratterizzate da pressanti requisiti non funzionali. Ad oggi, i database **NoSQL** sembrano essere un **completamento** dell'offerta di soluzioni per la manipolazione dati, **piuttosto che un rimpiazzo** dei database relazionali.

Le motivazioni per i database NoSQL

Il termine NoSQL è correntemente utilizzato per catalogare tutta una serie di **differenti** tecnologie di database distribuiti i quali, come visto poc'anzi, sono stati sviluppati per riuscire a far fronte a un'eccezionale aumento del numero e della frequenza di transazioni (figura 1). Tale aumento non solo ha portato ad elevatissimi volumi di dati da manipolare e memorizzare, ma ha anche generato stringenti requisiti non funzionali soprattutto nello spazio della **scalabilità** e **performance**. In particolare, si è assistito alla necessità di dover far fronte ad **elevatissime frequenze** di **accesso** ai **dati**, abilità di selezionare opportuni sottoinsiemi di informazioni da un vastissimo insieme in frazioni di secondo, ad esigenze notevoli relative all'elaborazione, e così via.

Figura 1 – Diagramma sul trend esponenziale della crescita dati [4]. Da notare che la crescita esponenziale, evidenziata dal semispazio verde riguarda prevalentemente dati semi-strutturati o non strutturati.

Del tema “CAP e Web 2.0” avevamo già fatto menzione nel primo articolo di questa serie [5]. Questi sempre più pressanti requisiti non funzionali hanno finito per evidenziare tutti limiti delle soluzioni tradizionali basate su database relazionali. I RDBMS, ad onor del vero, sebbene siano in grado di fornire eccellenti performance e di immagazzinare notevoli quantitativi di dati, non sono stati progettati per far fronte a requisiti di **scalabilità** e **flessibilità** dell’ordine di grandezza richiesto dalle recenti evoluzioni dei sistemi informatici.

Inoltre, i database relazionali non sono stati progettati neanche per fornire archiviazioni a **basso costo** e potenza di elaborazione richieste dalla crescita esponenziale del volume dati a cui si assiste oggi. In estrema sintesi, i vantaggi offerti dai database NoSQL rispetto a quelli relazionali possono essere raggruppati nelle seguenti quattro categorie: gestire elevati dati di volumi, maggiore scalabilità, maggiore flessibilità evolutiva, riduzione del gap tra paradigma Object Oriented e database relazionale.

Volume dati

I DB NoSQL si caratterizzano per la capacità di gestire **elevati volumi** di dati in varie forme: **strutturati**, **semi-strutturati** e **non strutturati**. Da notare che l’emergere di nuovi tipi di applicazioni, soprattutto **social** e **mobile**, hanno creato una grande domanda di gestione di dati semi- e non strutturati.

Maggiore scalabilità

Questa tipologia di database facilita il disegno di architetture efficienti a **maggiore scalabilità** soprattutto se paragonate con le classiche architetture dei database relazionali le quali, nonostante

la loro evoluzione, presentano tuttora un carattere fortemente **monolitico**. Per i nuovi database **NoSQL** si parla spesso della proprietà di elasticità (**elasticity**): poter aggiungere e rimuovere nodi dinamicamente in funzione del carico di lavoro.

Maggiore flessibilità evolutiva

Altro aspetto importante è quello della maggiore **flessibilità** e **predisposizione** all'**evoluzione incrementale** richiesta da processi di sviluppo del software fortemente iterativi. In questo caso, si parla della proprietà **softness** (letteralmente "morbidezza") ed è una caratteristica necessaria per modelli di sviluppo fortemente agile.

Riduzione del gap tra OO e relational

Non va infine trascurata la riduzione del tradizionale **gap** tra paradigmi di programmazione **Object Oriented** e database **relazionale**. Molte soluzioni NoSQL permettono una **diretta programmazione orientata** agli **oggetti**.

Tipologie di database NoSQL

I database NoSQL, in funzione della modalità con cui i dati sono strutturati e manipolati, possono essere raggruppati nelle seguenti categorie: **Document**, **Graph Store**, **Key-Value Store**, **Wide-Column.**

Document

Si tratta di banche dati che **associano** ad ogni **chiave** una

struttura di **dati** complessa definita **documento**. Un **documento** può contenere numerose **coppie chiave-valore** (**key-value pairs**), o coppie **chiave-array** o anche **documenti** annidati. Sebbene esistano diverse soluzioni di database **document oriented**, tutte assumono che le strutture e i dati siano rappresentate per mezzo di una codifica standard come per esempio: **JSON**, **XML** e **YAML**.

Si tratta di una organizzazione dati definita **semi-strutturata** proprio perche' non c'è separazione tra i dati e il corrispondente schema (figura 2): la quantità di struttura definita dipende dallo specifico utilizzo. In sintesi, i database appartenenti a questa categoria presentano importanti **similarità** con i **database OO**. **MongoDB** appartiene a questa categoria, così come **CouchDB** e **Riak**.

Figura 2 – Esempio di una rappresentazione semi-strutturata con l'equivalente struttura di un DB relazionale.

Graph Store

In questo caso i dati sono memorizzati attraverso **strutture a grafo**, ossia una delle strutture dati più **generiche** ed **eleganti**. Questi database sono utilizzati per memorizzare informazioni organizzate in reti complesse, come per esempio le **interconnessioni sociali**. Sebbene i social network rappresentino solo una minima parte delle applicazioni che si possono realizzare con i database Graph Store, sono anche le rappresentazioni più intuitive e quindi più facili da comprendere. Esempi di questo tipo di database sono **Neo4J** e

HyperGraphDB.

Gli elementi strutturali fondamentali di un database a grafi sono tre: nodi, relazioni, proprietà (figura 3).

Figura 3 – Rappresentazione logica di un Graph store.

I **nodi** rappresentano il **componente** principale per la **memorizzazione** delle informazioni attraverso opportune **proprietà**. La versione più semplice di questo database è formata da un solo nodo. Per esempio (figura 4), il nodo “Arsene Lupin” possiede le proprietà:

- Name = Arsene Lupin
- Alias = Lupin III
- Role = Gentle thief

Figura 4 – Esempio di utilizzo di un graph store.

Le **relazioni** organizzano i nodi. I nodi sono **organizzati** secondo determinate **relazioni** che a loro volta posseggono delle **proprietà** e quindi possono memorizzare delle informazioni. Le relazioni permettono di rappresentare strutture complesse come per esempio **liste**, **alberi** e **mappe**. Nel diagramma di riferimento (figura 4) il nodo “Arsene Lupin” e quello “Daisuke Jigen” sono collegati per mezzo di una **relazione** che ha la proprietà **tipo** con valore **Teams up**.

Infine, le **proprietà** sono le **informazioni** che possono essere **associate** a nodi e relazioni.

Va da se' che strutture come quelle presentate in figura 4 possono essere rappresentate **anche** attraverso database relazionali, molto probabilmente con una **complessità superiore**; tuttavia esiste una grande differenza. Nei database relazionali le **strutture** devono essere **conosciute a priori** e sono "cementate" nello schema. Cambiamenti dello schema non sono sempre agevoli.

I database **Graph Store** (così come tutti i NoSQL) presentano invece l'interessante proprietà nota con il nome di **softness** ("morbidezza"): gli schemi di dati sono o definiti **genericamente** o **non** sono definiti del tutto. Ciò permette un elevato grado di flessibilità e quindi ne semplifica l'evoluzione a basso costo. Nel caso in questione, le **relazioni** tra i **nodi** possono evolvere, le **proprietà** dei **nodi** e delle **relazioni** possono **cambiare**. In poche parole, il grafo cambia in funzione dell'utilizzo e acquista un elevato grado di **polimorfismo**.

Key-value Store

Questa è sicuramente la variante **meno complessa** dei database NoSQL. Si tratta ancora di un database non strutturato in cui ogni singolo **elemento** è **persistito** come una coppia nome attributo (**key**), corrispondente a valore (**value**). Per esempio, **Riak** utilizza una struttura denominata **bucket** (letteralmente "secchio"), ossia un **namespace flat** ("piatto") che contiene coppie **key/value** simili (una sorta di **tabella**). Per esempio, un'applicazione potrebbe creare un **bucket** denominato **User** (figura 5) ove, a fronte di un **Id** univoco, come per esempio il codice **user** o l'indirizzo **email**, potrebbero essere associate varie informazioni dell'utente. Considerando l'API **RestFul** [6] di Riak, un particolare user potrebbe essere raggiunto con un indirizzo del tipo:

.../riak/users/lvt

Figura 5 – Esempio di organizzazione key-value in Riak.

Analogamente al caso di Coherence, la **scalabilità** di questi database è strettamente legata all'hashing delle chiavi che permette di distribuire i dati tra i vari nodi e quindi di scalare in maniera lineare. Alcuni esempi di database **key-value** sono **Riak** e **Voldemort**. Alcune implementazioni di questo modello, come **Redis**, permettono di associare ad ogni valore un tipo (**String**, **Integer**, etc.). Ciò ovviamente aggiunge valore semantico a questa tipologia.

Wide-Column

Questa tipologia di database NoSQL trae le proprie origini dall'implementazione **BigTable** realizzata presso i laboratori **Google** [7] ed è correntemente utilizzata in diverse applicazioni Google tra cui il web indexing, Google Earth, Google Maps e Google Finance, giusto per citarne alcune. Si tratta di applicazioni che necessitano di scalare linearmente al fine di poter gestire dati nell'ordine di grandezza Petabytes (10^15 bytes) in ambienti distribuiti che includono migliaia di server.

Per comprendere il modello dati di **BigTable** si consideri il motore di ricerca Google il quale esegue ricerche su una propria copia/immagine dell'**intero** WWW (!): un'immensa raccolta di pagine

web, con incluse ulteriori informazioni, memorizzate nei propri "database". Si provi a immagine che tipo di crescita abbiano conosciuto questi dati nell'arco degli ultimi 5-10 anni! Non appena il WorldWide Web ha iniziato la sua crescita, necessità di questo tipo hanno cominciato a scontrarsi con i limiti delle soluzioni tradizionali.

Tornando al caso del **web indexing** (figura 6), si immagini di poter disporre di una tabella in cui gli indirizzi URL, memorizzati in forma inversa, hanno il ruolo di **chiavi**, mentre le altre colonne sono utilizzate per memorizzare il **contenuto** della pagina web, e i vari **link** ad essa. Inoltre, è necessario mantenere il **dato temporale** di quando ogni elemento è stato catturato/memorizzato. L'esempio di figura 6 rappresenta un possibile segmento della tabella riferito alla pagina principale del sito di MokaByte, che nel caso in questione è referenziata da una pagina di LinkedIn e dal sito dell'Università degli Studi di Roma "La Sapienza". Tutti i vari link hanno una sola versione, mentre la colonna utilizzata per il contenuto ha tre versioni (timestamp: **t3**, **t5** e **t6**).

Figura 6 – Esempio di un possibile frammento della tabella per il web indexing (fonte: [7]).

Il database BigTable è una **mappa multidimensionale dispersa**, **ordinata**, **distribuita** e **persistente**. La mappa è indicizzata da una chiave equivalente alle chiavi primarie dei record, da una chiave per colonna, e da un timestamp. Ogni elemento/cella della

mappa è un **array** di byte non interpretato. Quindi:

(row key: String, column key: String, time: timestamp) -> String

Da notare che **Cassandra**, il database che sta alla base di Facebook, è stato disegnato e implementato a partire dal modello **BigTable**. Nel corso degli anni, come è lecito attendersi, ha subito alcune evoluzioni.

Quando si parla di questa categoria di database NoSQL bisogna porre attenzione a non confondere le soluzioni **Wide-Column** (alcuni esempi tra i più famosi sono probabilmente **Cassandra** e **HBase**) con database "orientati alle colonne" (**Column Oriented database** come **Vertica**, **Greenplum**, **Aster Data** and **InfiniDB**) che, pur utilizzando strutture interne diverse, restano a tutti gli effetti database **SQL**.

La tabella 1 riporta le varie tipologie di database NoSQL con alcune implementazioni particolarmente famose.

Tabella 1 – Principali database NoSQL.

Introduzione a MongoDB

MongoDB deve il suo nome al termine "hu**mongo**us" che si presta ad essere tradotto con i termini "enorme" o "colossale". Come visto in precedenza, si tratta di un database **NoSQL document-oriented** e **multipiattaforma** implementato in **C++**.

MongoDB, come tutti i database appartenenti alla categoria NoSQL, prende le distanze della struttura classica dei database relazionali basati su tabelle e si orienta su una struttura a **documenti** rappresentati per mezzo di una versione del linguaggio **JSON** (JavaScript Object Notation) con schemi dinamici, definita **BSON** (Binary JSON, descritto di seguito). La ragione alla base di questa scelta è semplificare l'integrazione dei dati per sistemi di nuova generazione rendendola più immediata e veloce possibile.

MongoDB è rilasciato con una licenza "combinazione" tra GNU Affero General Public License (anche detta Affero GPL o **AGPL**) e **Apache License**: è quindi un software gratuito e open source.

Tabella 2 – La classifica dei primi 20 database più popolari. Fonte: Db-Engine [10].

Le prime implementazioni di MongoDB furono opera dell'azienda newyorkese chiamata **10Gen** che, visto il successo del prodotto, fu ribattezza in **MongoDB Inc.** [9]. I primi rilasci avvennero nell'ottobre del 2007, quando MongoDB era solo un componente di una piattaforma progettata come un prodotto di servizio. Nel 2009 tuttavia **10Gen** cambiò strategia commerciale rilasciando il software come prodotto open source, posizionandosi come azienda di supporto e consulenza dello stesso. Da allora, MongoDB è stato adottato come software di **back end** per una serie di importanti siti web e servizi, tra cui **Craigslist**, **eBay**,

Foursquare, **SourceForge**, il **New York Times** e da molteplici banche di investimento. Da una ricerca effettuata molto di recente (novembre 2013) da DB-Engines risulta che MongoDB è il più popolare database NoSQL tra quelli attualmente disponibili (tabella 2).

Figura 7 – Tendenza della popolarità di Mongo DB. Fonte: Db-Engine [10].

Una panoramica su Mongo DB

MongoDB è un database NoSQL le cui principali caratteristiche sono di essere **document-oriented**, di avere **elevate prestazioni**, di garantire **alta disponibilità** e di presentare una **scalabilità facilitata**.

Database document-oriented

La **struttura** dati è basata su oggetti di tipo "documento" e pertanto si presta ad un'integrazione quasi immediata con i moderni linguaggi di programmazione. L'organizzazione in **documenti** e **array** annidati riduce la necessità di effettuare **join**. La presenza di **schema dinamici** ne semplifica l'evoluzione e supporta processi di sviluppo agili.

Elevate performance

La struttura a **documenti annidati** rende le letture e le scritture più veloci eliminando la necessità di accedere a diversi elementi. È possibile definire **indici** su campi presenti nei vari livelli di annidamento inclusi array. Vi è infine la possibilità di eseguire **scritture** in **streaming** senza **ackwnoledgement**.

Alta disponibilità

Il **deployment** prevede una serie di **server replicati** e meccanismi di **failover** automatico.

Scalabilità facilitata

I dati sono **distribuiti automaticamente** sui diversi server disponibili. Le letture sono **eventually-consitstent** ("consistenti alla fine") sui server replicati.

Le caratteristiche di Mongo DB

Le caratteristiche principali di MongoDB, assolutamente compatibili con gli altri database della categoria NoSQL sono: **flessibilità**, **potenza**, **performance/scalabilità** e **facilità d'uso**.

Flessibilità

MongoDB memorizza i dati in documenti **JSON** (memorizzati in forma binaria: **BSON**). Questa soluzione fornisce un modello dati ricco che si presta a essere immediatamente **integrato** con i tipi di **dati nativi** dei linguaggi di programmazione più utilizzati. Inoltre, lo schema dinamico ne semplifica l'evoluzione soprattutto se paragonato a sistemi con schemi predefiniti come nel caso dei

database relazionali.

Potenza

MongoDB fornisce molte delle caratteristiche tipiche dei tradizionale database relazionali come per esempio: **indici primari** e **secondari**, **query dinamiche**, **ordinamento**, complessi **aggiornamenti**, semplicità di **aggregazione** e così via. Quindi riesce a offrire la ricchezza di funzionalità tipiche di un database relazionale, con la capacità, la flessibilità e la scalabilità tipici dei database NoSQL.

Performance/Scalabilità

La caratteristica di mantenere i dati **correlati** in **documenti** fa sì che le query possano essere eseguite con una latenza minore (molto più velocemente) rispetto a database relazionali in cui i dati sono in genere distribuiti in **più tabelle** e quindi hanno bisogno di essere **uniti all'atto** del **reperimento**. Ciò rende più immediata anche la **scalabilità** della base di dati. In particolare, lo sharing automatico dei dati, come nel caso di **Coherence**, consente di scalare il cluster linearmente con l'aggiunta di nuove macchine. Inoltre, si può aumentare la capacità senza generare alcuna interruzione del servizio, caratteristica molto importante per molte applicazioni web, che possono assistere a variazioni improvvise di carico. In questi scenari, portare giù il sito web per manutenzione potrebbe generare un impatto sul business o semplicemente potrebbe essere un'opzione non attuabile.

Facilità d'uso

Particolare importanza durante lo sviluppo di MongoDB è stata assegnata alla **facilità d'uso** del software durante le varie fasi: **installazione**, **configurazione**, **gestione** e **utilizzo**. A tal fine, MongoDB fornisce alcune opzioni di configurazione, e, ove possibile, cerca di fare automaticamente e di trovare i setting più appropriati. Questo significa che MongoDB è in grado di funzionare fin da subito, evitando lungaggini di installazione e studio dei manuali. Quindi permette di tuffarsi nello sviluppo dell'applicazione immediatamente evitando/minimizzando i tempi richiesti per installazioni, ottimizzazioni, etc.

MongoDB: il modello dati in breve

Un'installazione di MongoDB ospita una **serie** di **basi di dati** ognuna delle quali prevede un **insieme** dedicato di **file**. Un database, in termini MongoDB, è un **contenitore** di un **insieme** di **collezioni**, ove ognuna possiede un gruppo di documenti. Una collezione presenta diverse similitudini con una **tabella** di un classico database relazionale. Tuttavia, **non** esiste un vero **schema** predefinito. Un documento, infatti, è un insieme di coppie **chiave-valore** con schema dinamico. Ciò fa sì che i documenti nella stessa collezione non debbano necessariamente avere lo stesso insieme di campi o la stessa struttura. Anche i campi comuni nei documenti appartenenti alla medesima collezione possono essere di tipi di dato diversi.

BSON

BSON, acronimo derivato dalla fusione dei termini **Binary JSON**, è un protocollo binario utilizzato da MongoDB per memorizzare i documenti e per eseguire chiamate remote al database.

Come JSON, **BSON** supporta l'**annidamento** di **documenti** e **array** all'interno di altri documenti e array. BSON contiene anche meccanismi di **estensione** che consentono la rappresentazione dei tipi di dati che non fanno parte delle specifiche base di JSON. Ad esempio, BSON dispone dei tipi di dato **Date** e **BinData**.

BSON opera nello stesso dominio di altri formati (soprattutto binari) di interscambio dati, come Google Protocol Buffers (detto anche **protobuf**). La caratteristica peculiare di BSON è essere assolutamente libero da schemi predefiniti. Ciò offre grandi vantaggi in termini di flessibilità, a discapito però di una **lieve riduzione di performance** e di un ridotto incremento dell'occupazione della memoria dovuti alla necessità di riportare in testa alle serializzazioni i nomi dei campi dei dati inclusi.

BSON è stato progettato per avere le seguenti tre caratteristiche fondamentali: **leggerezza**, **navigabilità**, **efficienza**.

Leggerezza

Sebbene **BSON** sia un formato assolutamente flessibile, si è investito molto per cercare di minimizzare l'overhead al fine di ridurre l'impatto sull'occupazione di memoria e sulle performance. Il requisito di **leggerezza** (**lightweight**) è chiaramente fondamentale per qualsiasi formato di rappresentazione di dati.

Navigabilità

BSON è stato disegnato per far sì che gli oggetti possano essere **attraversati** semplicemente. Anche questa è una caratteristica fondamentale per il suo ruolo in MongoDB.

Efficienza

Le operazioni di **serializzazione** e **deserializzazione** dei dati in BSON sono eseguite rapidamente nei vari linguaggi. A tal fine, i progettisti hanno deciso di ricorrere all'utilizzo dei tipi di dato propri del linguaggio C, con opportune modifiche volte a standardizzarne l'occupazione di memoria.

Tipi di dato

L'insieme dei **tipi di dato BSON** contiene i tipi JSON. Per esempio JSON non dispone di un tipo **Date** ne' di un tipo di **array** di **byte**. Tuttavia esiste un'eccezione data dal tipo **numero** universale (**universal number type**) presente solo in JSON. BSON prevede i seguenti quattro tipi di dati base:

- byte 1 byte
- int32 4 byte
- int64 8 byte
- double 8 byte

Un **documento** è definito come un **int32** che indica il numero totale di byte del documento seguita dalla lista degli elementi (coppie **key/value**). Inoltre BSON supporta i tipi di dato accessibili attraverso l'operatore **$type** nel modo che riportiamo di seguito, specificando **tipo BSON** e **identificatore**, e riportando alcuni commenti

Tipo: **Double** – Identificatore: **1**

Tipo a 64-bit rappresentato secondo lo standard: IEEE 754 floating point number.

Tipo: **String** – Identificatore: **2**

Le stringhe sono rappresentate secondo la codifica UTF-8. I driver per ogni linguaggio di programmazione si occupano di convertire dal formato stringa del linguaggio a UTF-8.

Tipo: **Object** – Identificatore: **3**

Documento

Tipo: **Array** – Identificatore: **4**

Gli array sono rappresentati secondo la notazione JSON. Per esempio l'array con le lettere 'x' e 'y' è rappresentato come: **{'0':'x', '1':'y'}**.

Tipo: **Binary** data – Identificatore: **5**

Dati in formato binario.

Tipo: **ObjectId** – Identificatore: **7**

Questi identificatori sono stati disegnati per essere molto efficienti, piccoli, veloci da generare e ordinare e probabilmente univoci (funzioni random). Questo tipo è costituito da 12-byte, dove i primi quattro byte sono il timestamp della creazione dell'ObjectId.

Tipo: **Boolean** – Identificatore: **8**

true/false.

Tipo: **Date** – Identificatore: **9**

Il tipo Date è un intero con segno di 64-bit che rappresenta il numero di millisecondi a partire da Unix epoch (1 gennaio 1970).

Tipo: **Null** – Identificatore: **10**

Valore null (Null value).

Tipo: **Regular Expression** – Identificatore: **11**

Rappresentate per mezzo di due stringhe. La prima rappresenta il pattern regex e la seconda è la stinga delle opzioni regex.

Tipo: **JavaScript** – Identificatore: **13**

Frammento di codice in JavaScript, rappresentato per mezzo di una stringa.

Tipo: **Symbol** – Identificatore: **14**

Simbolo: si tratta di un tipo deprecato. Era un tipo simile a String con diversa codifica.

Tipo: **JavaScript** (with scope) – Identificatore: **15**

Frammento di codice JavaScript con scope.

Tipo: **32-bit integer** – Identificatore: **16**

Intero di 4 byte

Tipo: **Timestamp** – Identificatore: **17**

BSON ha un tipo timestamp speciale per uso interno (MongoDB), univoco all'interno di ogni processo, e non è associato al normale tipo Date. Valori timestamp sono un valore di 64 bit dove i primi 32 bit sono un valore time_t (in secondi da Unix epoch), e i restanti 32 bit sono un incremento ordinale per le operazioni eseguite nello stesso secondo.

Tipo: **64-bit integer** – Identificatore: **18**

Intero 8 byte

Tipo: **Min key** – Identificatore: **255**

Si tratta di un tipo speciale che è sempre minore rispetto a tutti i valori dei gli altri elementi BSON.

Tipo: **Max key** – Identificatore: **127**

Si tratta di un tipo speciale che è sempre maggiore rispetto a tutti i valori dei gli altri elementi BSON.

La "filosofia" di MongoDB

La maniera migliore di presentare la filosofia di MongoDB è verosimilmente quella di riportare la definizione di Eliot Horowitz [11], co-fondatore e CTO di MongoDB:

"MongoDB non è stato progettato in un laboratorio. Abbiamo costruito MongoDB dalle nostre esperienze nell'implementazione di sistemi robusti, su larga scala, ad alta disponibilità. Non siamo partiti da zero, abbiamo davvero cercato di capire che cosa non funzionasse più, e di affrontare solo quello. Così il mio modo di pensare MongoDB è che, se si prende MySql, e si cambia il modello di dati da relazionale a document-oriented, si ottengono molteplici caratteristiche molto importanti come documenti integrati per aumentare la velocità, maneggevolezza, supporto allo sviluppo agile attraverso database senza schemi predefiniti, scalabilità orizzontale semplificata dal momento che le join non sono più necessarie. Ci sono molte caratteristiche che funzionano molto bene nei database relazionali: indici, query dinamiche e aggiornamenti, solo per citarne alcuni, e non abbiamo cambiato molto in queste aree. Per esempio, il modo di progettare gli indici in MongoDB dovrebbe essere esattamente il modo in cui lo si fa in MySql o Oracle: la differenza è che esiste la possibilità di indicizzare campi embedded."

Conclusioni

In questo articolo siamo partiti ricapitolato brevemente le ragioni che hanno portato alla nascita dei database NoSQL e in

particolare l'improvvisa esigenza di dover gestire enormi quantità di dati, che ha evidenziato tutti i limiti delle soluzioni tradizionali basate su database relazionali. Soluzioni, che, ad onor del vero, non sono neanche state progettate per far fronte a queste sfide. I database NoSQL non sono nati nei laboratori, ma nelle varie organizzazioni come Google, Amazon, Facebook, e così via, che per prime si sono trovate a dover far fronte a una crescita esponenziale dei dati da manipolare e della frequenza di operazioni da gestire.

Dopo aver presentato le varie categorie di database NoSQL (document oriented, grafo, key-value e wide column) abbiamo avviato la descrizione di MongoDB: database NoSQL, document-oriented ad elevate performance, ad alta disponibilità in grado di scalare semplicemente. L'obiettivo di questa serie di articoli non è tanto quello di descrivere queste soluzioni in dettaglio, ma di discutere del teorema CAP, quindi le varie soluzioni sono introdotte al fine di poter trattare l'argomento delle scelte architetturali relative al particolare compromesso CAP scelto.

I database NoSQL stanno proseguendo la loro penetrazione del mercato dei database, tuttavia non si tratta di una soluzione alternativa ai database relazionali (Oracle continua a conquistare posizioni), ma di soluzioni particolarmente interessanti per il disegno e lo sviluppo di applicazioni caratterizzatee da stringenti requisiti non funzionali.

Riferimenti

[1] Neal Leavitt, "Whatever happened to object-oriented databases?", IEEE Computer, 33, No. 8:16-19, 2000.

[2] Carlo Strozzi. "NoSQL: A relational database management

system", 2007-2010

http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page

[3] Eric Evans' blog

http://blog.sym-link.com/2009/05/12/nosql_2009.html

[4] NoSQL

http://www.w3resource.com/mongodb/nosql.php

[5] Luca Vetti Tagliati, "Il teorema CAP… in Brewer – I parte: Il teorema CAP (Consistency, Availability, Partition tolerance)", MokaByte 186, Luglio/Agosto 2013

http://bit.ly/19zivg6

[6] Luca Vetti Tagliati, "Architetture REST: un modello di maturità – I parte: Da RESTless a RESTful", Mokabyte 168, Dicembre 2011

https://www.mokabyte.it/cms/article.run?articleId=T5H-H8J-86C-DOC_7f000001_15042247_08e01d0b

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, "Bigtable: A Distributed Storage System for Structured Data", SDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006

[8] La voce MongoDB su Wikipedia

http://en.wikipedia.org/wiki/MongoDB

[9] Derrik Harris, "10Gen embraces what it created, becomes MongoDB Inc.", 27 agosto 2013

http://gigaom.com/2013/08/27/10gen-embraces-what-it-created-

becomes-mongodb-inc/

[10] DB-Engines, “DB-Ranking” novembre 2013

http://db-engines.com/en/ranking

[11] Eliot Horowitz, 2013

http://www.mongodb.org/about/introduction/