ISSN/0393-3016

M. PUCCINI
Evodevo s.r.l

F. IANNONE
Dipartimento Tecnologie Energetiche
Divisione Sviluppo Sistemi per l'Informatica e l'Ict,
Laboratorio Infrastrutture per il Calcolo Scientifico
Centro Ricerche, Frascati

S. MIGLIORI, **M. MONGELLI**
Dipartimento Tecnologie Energetiche
Divisione Sviluppo Sistemi per l'Informatica e l'Ict,
Laboratorio Infrastrutture per il Calcolo Scientifico
Sede Centrale, Roma

SURVEY DELLE MODERNE SOLUZIONI PER DATABASE

RT/2019/13/ENEA

AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE,
L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE

M. PUCCINI
Evodevo s.r.l

F. IANNONE
Dipartimento Tecnologie Energetiche
Divisione Sviluppo Sistemi per l'Informatica e l'Ict,
Laboratorio Infrastrutture per il Calcolo Scientifico
Centro Ricerche, Frascati

S. MIGLIORI, M. MONGELLI
Dipartimento Tecnologie Energetiche
Divisione Sviluppo Sistemi per l'Informatica e l'Ict,
Laboratorio Infrastrutture per il Calcolo Scientifico
Sede Centrale, Roma

SURVEY DELLE MODERNE SOLUZIONI PER DATABASE

RT/2019/13/ENEA

AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE,
L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE

SURVEY DELLE MODERNE SOLUZIONI PER DATABASE

M. Puccini, F. Iannone, S. Migliori, M. Mongelli

Riassunto
Nel corso del Progetto EcoDigit (Ecosistema Digitale per la fruizione e la valorizzazione dei beni e delle attività culturali del Lazio), finanziato nell'ambito del Distretto Tecnologico per i Beni e le Attività Culturali della Regione Lazio, è stato svolto uno studio di valutazione delle moderne soluzioni di database per la componente middleware, cui ad ENEA spetta la progettazione. Valutare una tecnologia per la gestione di dati è un'operazione che richiede la definizione di una metodologia di impronta bottom-up. Primo passo è la definizione, il più possibile dettagliata, delle proprie specifiche esigenze. Che tipo di dati dovremo gestire, quale utilizzo ne dovremo fare, sono i punti di partenza per orientarsi nel complesso mondo dei database e dei sistemi distribuiti. Un approccio top-down difatti, rischia di portare all'errore più diffuso in ambito tecnologico, ovvero adattare una tecnologia alle proprie esigenze, invece di ricercare quella più indicata che le soddisfi. In questo documento vengono presentati alcuni elementi indispensabili con i quali poter procedere ad una valutazione, strutturando una metodologia basata sull'analisi dei requisiti, un'analisi delle tecnologie ed infine un'analisi della loro diffusione. L'obiettivo del documento è quello di offrire una panoramica ragionata sullo stato dell'arte in materia di database.

Parole chiave: Database, SQL, NoSQL, Data Science, Big Data, Sistemi distribuiti, Tecnologie

Abstract
This study is developed for the EcoDigit Project (Digital Ecosystem for the use and enhancement of Lazio's cultural assets and activities), funded under the Technological District for Cultural Heritage and Activities of the Lazio Region as an evaluation of the modern database technologies for the middleware which design is under responsibility to ENEA. Evaluate a technology for data management needs to define a methodology bottom-up. First step is the definition, as detailed as possible, of the own specific necessities. Which kind of data we will have to manage, which use we'll do with them are the starting points to find your bearing into the complex world of the databases and distributed systems. A topdown approach risks to carry to the most common error in technologic field, that is adapt a technology to our needs instead of searching for the most advisable to satisfy them. In this document are shown some essential elements with which proceeds with the evaluation, organizing a methodology based on a requirements analysis, a technologies analysis and at least an evaluation on their diffusion. The goal of the document is to give a reasoned review on the state of the art in the database topic.

Keywords: *Database, SQL, NoSQL, Data Science, Big Data, Distributed systems, Technologies*

INDICE

1 Introduzione

Il volume di applicazioni web, in particolare quelle riferite al mondo dell'e-commerce e dei social network, è cresciuto esponenzialmente negli ultimi 10-15 anni. Parallelamente, lo sviluppo di hardware a basso costo e programmabile, ha portato al moltiplicarsi di progetti che rientrano nell'acronimo *IoT*, Internet of Things. Tutte queste tecnologie, molto diverse tra loro, condividono tuttavia la produzione di enormi quantità di dati disomogenei e non strutturati. Siano essi *tweets* relativi ad un determinato *hashtag*, il numero delle visite al sito di un determinato prodotto finanziario, le prenotazioni ad un certo albergo, i messaggi scambiati tra due persone attraverso applicazioni di messaggistica istantanea o ancora i dati da sensori a fibra ottica per il monitoraggio di una struttura. La necessità di governare questa crescente mole di dati disorganizzati ha portato allo sviluppo ed al moltiplicarsi di differenti soluzioni tecnologiche. A partire dalla metà degli anni 90, con il diffondersi di infrastrutture distribuite e dal 2009, con il moltiplicarsi su internet di alcuni tipi di servizi, hanno iniziato a svilupparsi diverse tipologie di database, strutturalmente differenti rispetto alla categoria dominante fino a quel momento, quella dei database cosiddetti relazionali. Tutte queste nuove tipologie possono essere raggruppate all'interno della macro-categoria dei database genericamente detti non-relazionali (o anche *NoSQL*, ossia *Not only SQL*).

In questo documento viene illustrata una panoramica delle principali soluzioni di database, delle loro caratteristiche, la loro diffusione ed individuando degli esempi di utilizzo comuni e le relative scelte di database più opportune. Dato che le tecnologie sono in continua evoluzione, alcuni aspetti tecnici non sono stati affrontati nel dettaglio poiché possono essere oggetto di tale evoluzione e modificarsi nel tempo. Rimangono invece invariate alcune caratteristiche fondamentali sulle quali questo lavoro si concentra. Per questo motivo e per l'estrema eterogeneità delle singole necessità di progetto, questo rapporto tecnico deve intendersi come punto di partenza e non come una guida per tutti i casi.

2 Metodologia di valutazione delle tecnologie

Per poter operare una scelta riguardo la migliore tecnologia adatta alle proprie esigenze, dobbiamo anzitutto metterle a fuoco. Successivamente potremo procedere alla valutazione, prendendo in considerazione alcuni concetti fondamentali. Nelle sezioni seguenti verranno dunque presentati brevemente le principali nozioni con le quali costruire criteri di valutazione.

Anzitutto si devono distinguere i modelli di dati con cui questi possono essere rappresentati. Ognuno ha strutture diverse e risponde a precise esigenze di organizzazione, gestione e rappresentazione. In base ai dati in possesso, alle loro caratteristiche ed ai propri specifici bisogni su come questi debbano essere trattati (eventualmente anche processati), un primo passo è quello di individuare il modello di dati che meglio rappresenta questo quadro di esigenze. Essendo sistemi distribuiti, ci sono delle scelte da fare rispetto ad alcune caratteristiche proprie di queste infrastrutture. Anche in questo caso, quello che deve guidare sono le necessità specifiche che faranno propendere per una caratteristica piuttosto che per un'altra, tenendo conto che esistono dei vincoli entro i quali restare. La concorrenzialità tra disponibilità e consistenza dei dati ad esempio, vincolata dal *teorema di Brewer*, rappresenta probabilmente la scelta più discriminante, immediatamente successiva alla definizione del modello dei dati da adottare.

Particolare attenzione va posta anche rispetto alla *popolarità* e alla *diffusione* della specifica soluzione tecnologica cui ci stiamo indirizzando. Per popolarità si intende una serie di indicatori quali: la quantità e la tipologia di soggetti che hanno adottato questa tecnologia, le *communities* di utilizzatori e sviluppatori, la diffusione di software di interfacciamento (connettori, API, ...), presi in considerazione anche e soprattutto in termini delle loro tendenze nel tempo, al fine di stimarne la capacità di sopravvivenza nella competizione con l'alto numero di competitor. Per quest'ultimo compito, inevitabilmente, entrano in gioco valutazioni di carattere non necessariamente tecnico. Ad esempio, nella valutazione della tecnologia più appropriata ad un progetto di ricerca scientifica, sarà utile uno studio di soluzioni e *best practices* in contesti analoghi, quali altri enti ed istituti che operano nel settore scientifico.

2.1 Definire le proprie necessità

Il primo passo è focalizzare quello di cui si ha bisogno. Per questo scopo, è utile rispondere alle seguenti domande:

1. Che tipo di utenti dovrà usare il sistema?
2. Come dovrà essere usato il sistema?
3. Quanti utenti si preveda che avrà il sistema (ordini di grandezza)?
4. Cosa dovrà fare esattamente il sistema?
5. Quali saranno gli input e output del sistema?
6. Quanti dati ci aspettiamo che dovrà gestire (all'inizio e con quale ritmo di crescita)?

7. Quante richieste al secondo ci aspettiamo?
8. Qual è la proporzione tra le letture e le scritture dei dati?

Dal momento che parliamo di tecnologie relative a sistemi distribuiti è utile avere le risposte alle precedenti domande al fine di progettare le caratteristiche del sistema, in particolare la sua architettura, le sue dimensioni (numero nodi, capacità di storage, ...), le risorse (CPU, GPU, RAM, storage, ...), le tecniche di scalabilità e replicazione dei dati, le *policies* di accesso e sicurezza. Vediamo di comprendere meglio le singole domande.

Che tipo di utenti dovrà usare il sistema? Ossia quali sono le caratteristiche dell'utente o degli utenti. Sono utenti che avranno bisogno di interfacce di gestione o sapranno interagire direttamente con la tecnologia? Sono di tipologie diverse, magari per necessità?

Come dovrà essere usato il sistema? Dovrà essere interrogato molto frequentemente da una web app oppure ogni tanto da un team di ricercatori? Dovrà raccogliere slot di dati ad ogni fine campagna sperimentale, o dovrà rimanere collegato in acquisizione real-time per tutta la durata di un progetto di ricerca?

Quanti utenti si preveda che avrà il sistema (ordini di grandezza)? Chiaramente qui si deve fare una stima, ma porsi questa domanda in questo momento, evita di ritrovarsi con sistemi sottodimensionati nel mezzo del loro utilizzo, quando potrebbe essere molto complicato se non impossibile correggere la situazione. D'altro canto, anche un sistema sovradimensionato rappresenta un inutile impiego di risorse, che può significare un inutile aggravio per chi deve gestire la tecnologia (dba, sistemisti, ...).

Cosa dovrà fare esattamente il sistema? Questa domanda è strettamente collegata ad altre, come quelle sull'utenza e quelle sulle capacità di lettura e scrittura. Si vuole capire quale esigenza/e dovrebbe soddisfare il sistema: dovrà assistere un sistemista a monitorare le prestazioni di un cluster, o dovrà accogliere dati relativi al personale di un'organizzazione? I dati che deve gestire dovranno poter essere pre o post processati prima o dopo averli immessi nel database? Dovremo fare particolari *queries*, magari aggregando valori relativi ad intervalli temporali? Avremo necessità di incrociare ricerche su più blocchi di dati, oppure possiamo pensare dati differenti completamente isolati gli uni dagli altri?

Quali saranno gli input e output del sistema? Abbiamo bisogno di gestire uno streaming di dati provenienti da un social network, oppure sono file di testo inseriti una tantum? Come ci serve che ci vengano restituiti?

Quanti dati ci aspettiamo che dovrà gestire (all'inizio e con quale ritmo di crescita)? Stesso discorso relativo al numero di utenti. Necessario per disegnare in modo opportuno l'architettura.

Quante richieste al secondo ci aspettiamo? Quante letture al secondo ci aspettiamo che avrà il nostro database? Una web app che interroga ogni 3 secondi un database per farsi restituire i dati, ad esempio, di un sensore di temperatura, obbliga ad orientarsi verso tecnologie che garantiscano periodi di latenza inferiori alla frequenza delle letture. Contrariamente, è inutile puntare a sistemi ad alte prestazioni di lettura se dovremo interrogare il database solo una volta al mese per calcolare gli stipendi di un'azienda.

Qual è la proporzione tra le letture e le scritture dei dati? Abbiamo necessità di un sistema in grado di ricevere scritture e letture nella stessa misura? C'è uno sbilanciamento in questo rapporto a favore di una delle due?

2.2 Modello di dati

Un modello di dati è la struttura formale astratta mediante la quale questi vengono organizzati. Tale organizzazione riguarda il modo in cui i dati vengono descritti e rappresentati, ma anche le possibili operazioni che possiamo avere necessità di fare, come ad esempio quella di aggregare temporalmente ed ottenere una media su quell'intervallo. In Tabella 2.1 sono elencati i modelli di dati più diffusi, sia i tradizionali relazionali che quelli emersi nell'ultimo decennio, che sono descritti poi nel seguito.

Principali modelli di dati
Relational
Key-Value
Document
Wide-Column
Graph
Time series

Tabella 2.1: Lista dei principali modelli di dati trattati in questo documento.

2.2.1 Relational model

Il modello relazionale [1] è quello più datato ma tuttavia anche quello più utilizzato ancora oggi. Questo modello rappresenta i dati mediante i concetti astratti di *entità* e *relazioni* organizzati in *tabelle* ed identificati attraverso una *chiave* univoca. Prende il nome dall'*algebra relazionale*, il linguaggio procedurale che descrive le procedure per ottenere un determinato risultato nell'interazione con un database. Su questo modello dati si è sviluppato il linguaggio SQL, ad oggi ancora il più diffuso linguaggio per database.

2.2.2 Key-Value stores

Questo modello è molto semplice e consiste in un set di coppie chiave-valore definite da chiavi univoche. Data la sua struttura semplice, supporta solo operazioni di tipo *get* e *put* ossia, rispettivamente, di richiesta e di inserimento, dunque non rispetta una completa tabella CRUD. Questo è un primo esempio di dati *schemaless*, o meglio è definibile come *schema-on-read*, ossia con una struttura implicitamente codificata nella logica dell'applicazione. Al contrario dei dati strutturati, nei quali lo schema viene definito prima di scriverli (*schema-on-write*), in questo caso i dati vengono inseriti come sono per poi definire lo schema in fase di lettura. Il principale vantaggio di questo modello è, chiaramente, la semplicità. Questo rende possibili il partizionamento e la ricerca dei dati, consentendo basse latenze ma anche alte capacità di trasmissione.

2.2.3 Document stores

Questo modello rappresenta i dati similmente ai chiave-valore, solo che in questo caso ad ogni chiave non è associato un singolo valore ma viene annidata una collezione o lista di coppie attributi/valore. Dunque ad ogni chiave viene associato un documento semi-strutturato, come ad esempio i documenti JSON. In Figura 2.1 è rappresentato questo tipo di modello.

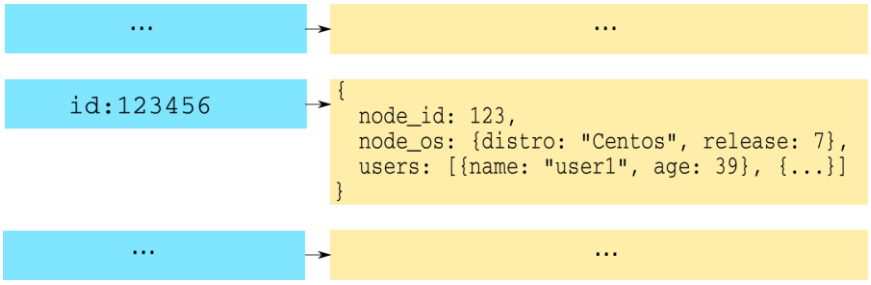

Figura 2.1: Esempio di modello dati a documento.

Di seguito viene riportato invece il documento JSON relativo alla Figura 2.1:

```
{
   "id": 123456,
   "node_id": 123,
   "node_os": {"distro": "Centos", "releaase: 7},
   "users": [
      {"name": "user1", "age": 39},
      {"name": "user2", "age": 56}",
      ...
   ]
}
```

2.2.4 Wide-Column stores

É il modello apparentemente più simile a quello relazionale, avendo anche in questo caso, delle tabelle strutturate in righe e colonne. La differenza è nella possibilità, per le colonne, di variare nome e formato. In realtà la struttura interna è ancor più complessa, considerando ad esempio che molti database basati su questo modello di dati, consentono anhce un sistema di *versioning* dei valori delle chiavi. Diverse tecnologie che implementano questo modello, si differenziano per altre caratteristiche. Bigtable di Google ad esempio, prodotto pioniere per questo tipo di modello, si differenzia per diversi aspetti dal prodotto più affermato di questo settore che, al momento (maggio 2019) è uno dei possibili elementi dell'ecosistema Hadoop di Apache, ossia Cassandra. Da sottolineare il fatto che, in questo modello di dati, molta dell'informazione è contenuta all'interno della chiave stessa più che nei valori delle relative colonne.

2.2.5 Graph

Quello a *grafo* è un modello di dati mutuato dalla *teoria dei grafi*. Secondo questo modello i dati sono rappresentati mediante un insieme finito (ma potenzialmente mutabile) di oggetti astratti definiti come *vertici* o *nodi*, collegati a coppie da *grafi* che ne rappresentano le relazioni. Questi grafi possono definirsi *archi* (o *spigoli*) nel caso di coppie non ordinate, nel caso si parla di *grafi non orientati*. Se invece le coppie sono ordinate, nei *grafi orientati*, prendono il nome di *frecce* (o *archi diretti*). In Figura 2.2 sono rappresentati questi due tipi di grafi (orientati in Figura 2.2a e non orientati in Figura 2.2b). In Figura 2.3 viene invece mostrato,, a titolo esemplificativo, il grafo per rappresentare il network di editori (nodi) di Wikipedia per le differenti lingue che concorrono alle versioni dell'enciclopedia nelle diverse lingue in ogni mese.
Il modo per archiviare questo tipo di dati varia da database a database. In generale, rimane un modello che consente ricerche molto veloci.

(a) Grafo orientato (b) Grafo non orientato

Figura 2.2: Rappresentazione di grafi: in blu i vertici, i segmenti sono gli archi o le frecce a seconda del tipo di grafo.

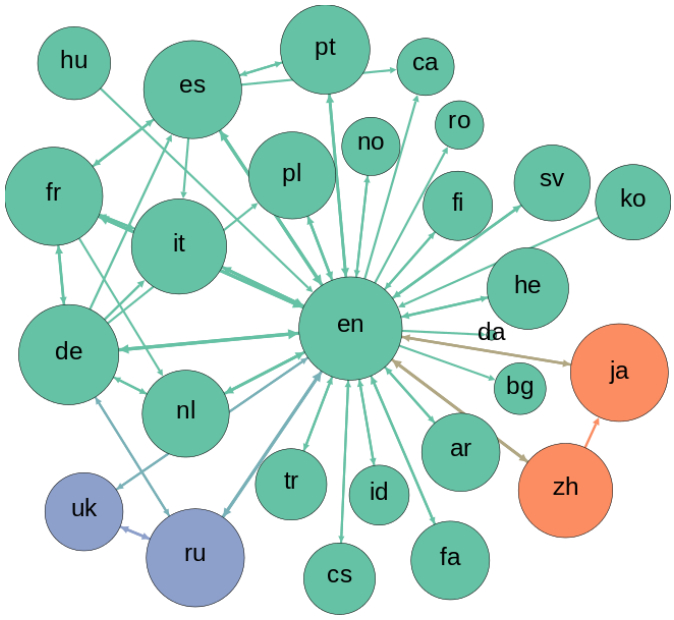

Figura 2.3: Il grafo delle diverse edizioni di Wikipedia in un mese.

Fonte: https://en.wikipedia.org/wiki/Graph_theory

2.2.6 Time series

Le serie temporali, più che un modello di dati, sono una tipologia di acquisizione di questi ultimi. Sono il classico dato preso ad intervalli definiti di tempo, strutturato in *n* colonne. Per ogni riga, il primo valore è quello temporale (tipicamente un timestamp), i successivi (da 2 a *n*), sono quelli relativi alle grandezze misurate. Possono riferirsi alle acquisizioni di sensori, a valori di stock sul mercato ma anche a variabili di sistemi informatici quali, ad esempio,

quelli di logging. Sono il tipo di dato che tipicamente interessa l'IoT, la matematica finanziaria, l'econometria, la meteorologia, l'astronomia, la sismologia, l'ingegneria dell'automazione e quella delle telecomunicazioni. Sono stati inseriti nel capitolo relativo ai modelli di dati poiché, in seguito alla loro massiccia diffusione, hanno dato luogo a database ottimizzati per la loro gestione.

2.3 Disponibilità e Consistenza

Come anticipato, in un sistema distribuito, il bilancio tra disponibilità e consistenza rappresenta una scelta discriminante per orientarsi verso una tecnologia piuttosto che un'altra. Come verrà mostrato di seguito, sono due condizioni impossibili da rispettare contemporaneamente, dunque si deve obbligatoriamente scegliere quale, tra le

due, risponde meglio alle proprie esigenze. Questo vincolo è definito da uno dei più importanti risultati della teoria sui sistemi distribuiti, il Teorema CAP.

2.3.1 Teorema di Brewer

Il teorema di Brewer, noto anche come Teorema CAP, afferma che:

In un sistema distribuito si possono soddisfare contemporaneamente al massimo due delle seguenti garanzie (Figura 2.4):

(C)onsistency ogni lettura riceve la più recente scrittura o al massimo un errore

(A)vailability ogni richiesta riceve una risposta, senza però garanzia che questa contenga la scrittura più recente.

(P)artition tolerance il sistema continua a funzionare anche in caso di guasti, nonostante un partizionamento arbitrario.

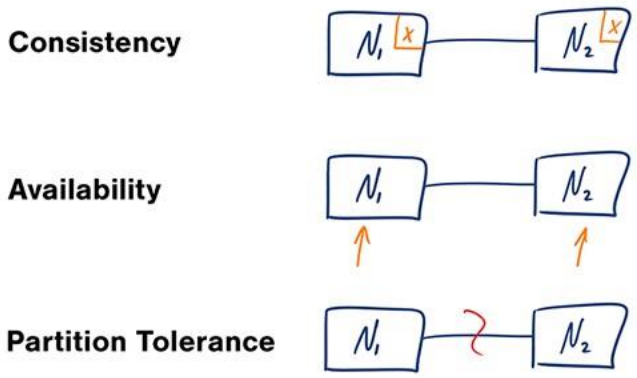

Figura 2.4: Garanzie per un sistema distribuito.

Fonte: http://robertgreiner.com/2014/08/cap-theorem-revisited/

Per partizionamento ci si riferisce, nel caso di problemi di comunicazione fra nodi o nel caso in cui uno o più nodi siano spenti, alla creazione di una sotto rete costituita dalla parte o dalle parti del network ancora funzionanti. Nei sistemi non distribuiti, non essendoci network di comunicazione tra nodi differenti, non avremo problemi di partizionamento. Dunque potremo riuscire a garantire contemporaneamente sia la disponibilità che la consistenza. In un sistema distribuito, dovendo necessariamente garantire la partition tolerance (P), senza la quale avremmo un sistema distribuito non funzionante non appena anche solo una sua parte dovesse avere problemi, siamo costretti a scegliere tra due scenari:

- **CP - Consistency and Partition tolerance**, l'attesa di una risposta dal nodo partizionato potrebbe causare un errore di timeout.
- **AP - Available and Partition tolerance**, le risposte restituiscono la versione più recente dei dati disponibili su un nodo, ma potrebbe non essere l'ultima.

Dunque, sostanzialmente, dato per assodato che si debba garantire P, dobbiamo scegliere se ci interessa avere garantita C o A. Questo ci apre a due possibilità, i due insiemi di proprietà relative alle transazioni di un database, che rappresentano le due opposte filosofie di design nello spettro C-A:

ACID Acronimo coniato nel 1983, da Andreas Reuter e Theo Härder [2], che indica le seguenti proprietà:

- **Atomicity**, ogni transazione è indivisibile dal punto di vista logico, dunque nessun'altra transazione può coesistere fintanto che non sia ultimata la prima.
- **Consistency**, ogni transazione porta il DB da uno stato valido ad un altro stato valido.
- **Isolation**, il risultato è il medesimo se le transazioni vengono eseguite in modo concorrenziale o seriale.
- **Durability**, una volta che la transazione è stata commessa, rimane tale.

BASE Definito in un articolo del 2008 [3], è l'acronimo di:

- **Basically available**, il sistema garantisce la disponibilità.
- **Soft state**, lo stato del sistema può cambiare nel tempo, anche senza ricevere input.
- **Eventual consistency**, il sistema diventa consistente entro un determinato periodo di tempo assumendo di non ricevere input in tale periodo.

Una corretta interpretazione del teorema CAP è stata avanzata dallo stesso autore [4] dodici anni dopo la sua formulazione. Il documento citato, cui si rimanda in bibliografia per approfondimento, consente di introdurre il prossimo teorema, poiché sottolinea la necessità di tenere in conto anche la latenza.

2.3.2 Teorema PACELC

Questo teorema [5] è un'estensione del precedente ed afferma che:

> *In un sistema distribuito nel quale sia garantita la partition tolerance (**P**), si deve scegliere tra disponibilità (**A**) e consistenza (**C**), altrimenti (**E**-else), anche quando il sistema è normalmente in funzione, si deve scegliere tra latenza (**L**) e consistenza (**C**).*

Il teorema CAP viene esteso dunque, considerando anche il bilanciamento tra latenza e consistenza, che si presenta necessariamente dal momento che in un sistema distribuito i dati sono replicati. Ossia, in un sistema distribuito che assicura P, in assenza di guasti, si potrà comunque avere la condizione per la quale una transazione abbia dei ritardi. In quel frangente il sistema dovrà decidere se aspettare comunque la risposta prima di proseguire le operazioni, mettendo a rischio la consistenza. Oppure se cancellare l'operazione e proseguire senza, minando la disponibilità. In Figura 2.5 viene mostrato uno schema con il quale poter collocare diverse soluzioni di database rispetto al teorema CAP, indicandone anche il relativo modello di dati.

2.4 Altri fattori fondamentali

Oltre ai modelli dati ed alla scelta di puntare sulla disponibilità o sulla consistenza, ci sono altri fattori che si devono prendere in considerazione all'atto di una completa valutazione della migliore tecnologia per le proprie necessità. Tuttavia, in questo documento verrà fatto solo un accenno, illustrando brevemente i fattori di cui è utile

prendere in considerazione l'esistenza. La scalabilità, ad esempio, è una condizione ormai assicurata da qualsiasi tecnologia: anche i database relazionali possono scalare orizzontalmente come quelli non relazionali. Le differenze risiedono nel come questo viene fatto, dalla particolare tecnica scelta dagli sviluppatori della tecnologia per questo scopo.

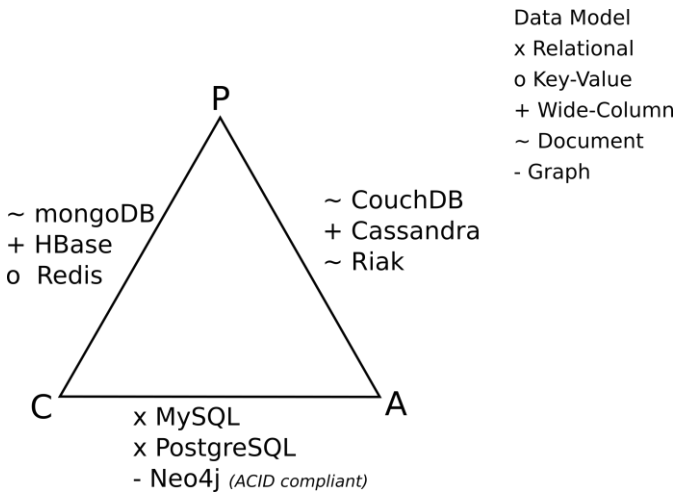

Figura 2.5: Schema a triangolo per illustrare graficamente la collocazione di diversi database rispetto al teorema CAP.

Prestazioni e Scalabilità Definiamo un sistema *scalabile* quando, aumentandone le risorse, possiamo misurarne un proporzionato aumento delle prestazioni. Per aumento delle prestazioni intendiamo sia la capacità di servire un numero maggiore di unità di lavoro, sia la capacità di gestire unità di lavoro più grandi. Possiamo parlare di scalabilità verticale se ottenuta aumentando le risorse della singola macchina, mentre invece parliamo di scalabilità orizzontale quando ad aumentare sono il numero di nodi (macchine). Mentre nel primo caso possiamo ottenere prestazioni di calcolo maggiori, aumentando le capacità di CPU o di GPU per numero di core o per tipo di architettura del processore, ma con un maggiore investimento di risorse economiche; nel secondo otteniamo un aumento delle capacità del sistema (ad esempio relativamente allo storage), anche con hardware modesto e dunque più economico. Spendere molti soldi per avere l'ultimo processore, o per aumentare i GB di memoria, potrebbe essere del tutto inutile se le operazioni di cui abbiamo bisogno, possono essere svolte da hardware più datati. Potremmo dover disporre di una quantità di spazio di archiviazione crescente, per cui sarebbe sufficiente aggiungere altre macchine, anziché comprare hard disk più grandi. Ovviamente, anche in questo caso, si può capire l'importanza di progettare prima l'architettura del sistema, per non ritrovarsi con la necessità di espandere la memoria o lo spazio di archiviazione in corso d'opera. A seconda dell'architettura, potrebbe essere infatti un'operazione rischiosa per i nostri dati, o che comunque richiede passaggi di backup intermedi dispendiosi in termini di tempo e risorse.

Latenza e Capacità di trasmissione La latenza (*latency*) è il tempo che intercorre tra una richiesta al sistema (ad esempio una query nel caso di database) e la risposta dello stesso. In altri termini è la misura del tempo che passa tra uno stimolo al sistema e la relativa risposta allo stimolo. Questo intervallo dipende chiaramente dall'infrastruttura, dal tipo di comunicazione (lan, wifi, radio, ...), ma anche dalle tecniche adottate dalle singole soluzioni di database per gestire le transazioni. Avere bassi o alti valori di latenza, può dipendere infatti da quanto il compito che chiediamo ad una determinata tecnologia, si discosti da quello per cui è stata pensata. Ad esempio, se un database si basa su un modello di dati di tipo serie temporale, fare un'operazione come la media su un intervallo di tempo implica un determinato valore di latenza, mentre richiedere di mostrare un intervallo di valori, non essendo tale database strutturato per questo tipo di operazione, comporta valori di latenza molto maggiori.
La capacità di trasmissione (*troughput*) definisce invece le dimensioni massime *effettive* del pacchetto di dati che si riesce a trasmettere in una transazione. Tecnicamente è il numero di dati trasmessi nell'unità di tempo e viene misurato in bit al secondo (bit/s). Poiché i protocolli di trasmissione prevedono generalmente una transazione alla volta, prima di trasmettere un altro pacchetto di dati, dovremo aspettare che finisca la trasmissione precedente e che si ottenga dal destinatario la conferma di ricezione. Solitamente, se una rete ha una capacità di K bit/s, il troughput sarà di $K/2$ bit/s. Metà del tempo è infatti impiegata per trasmettere il pacchetto di dati, l'altra metà per attendere la conferma di ricezione.
La relazione teorica che lega queste due grandezze è la *Legge di Little* [6], un risultato della *teoria delle code*, una branca della matematica delle probabilità. Essa afferma che Il numero medio di clienti L in un sistema è uguale al tasso medio di arrivo λ moltiplicato per il tempo medio W nel sistema:

$$L = W \cdot \lambda \tag{2.1}$$

Identificando con la 2.1 la latenza con il tempo medio di attesa W e il troughput con il tasso λ, otteniamo una relazione vincolante fra le due grandezze considerando che L è fissato dalle capacità che desideriamo per il nostro sistema.
Per evidenziare questa relazione è utile fare un esempio particolare: il protocollo *IPoAC* [7], regolarmente sottoposto all'IEFT (Internet Engineering Task Force) come *RTF 1149* (Request for comments). Si tratta di una proposta di protocollo che utilizza piccioni viaggiatori per trasportare pacchetti di dati archiviati in memorie flash (IPoAC sta infatti per IP over Avian Carriers). Nonostante la proposta avesse chiari intenti goliardici, evidenzia in modo efficace la relazione sopra citata. Il protocollo, seppure vantando alti valori di troughput, potendo trasferire grosse quantità di dati mediante le memorie flash, ha ovviamente pessime prestazioni in termini di latenza. Il lavoro svolto è quello effettivamente richiesto, ossia la trasmissione di molti dati. Tuttavia, la *tecnologia* scelta, non è evidentemente quella più indicata per il compito.

Consenso Nei sistemi distribuiti, essendo costituiti da più nodi che comunicano tra loro, nel momento in cui uno o un numero maggiore di questi non siano raggiungibili per qualche motivo, si deve poter scegliere con quale parte del sistema comunicare (quella non isolata e funzionante chiaramente). Il sistema dovrebbe poter determinare autonomamente come riorganizzarsi. Le macchine funzionanti lanciano un *appello* aspettando la risposta delle

altre, definendo così il numero di nodi funzionanti. La mancata comunicazione con un nodo può avvenire sia in seguito ad un fallimento alla macchina, sia perché questa si è spenta, sia per problemi nella trasmissione con questa. In ogni caso, la o le altre macchine, non riescono a comunicare con la prima. Le altre macchine a questo punto dovranno essere in grado di prendere una decisione su quale deve essere il nuovo assetto del network. Un algoritmo in grado di svolgere questo compito risponde ad un problema teorico più generale. Non entreremo nel dettaglio della trattazione teorica che anzitutto richiederebbe di definire il problema del *consenso* all'interno del sistema. Dovremmo capire se il sistema lavora in un regime di scambio di informazioni sincrono o asincrono e che tipo di fallimento lo interessa. Possiamo infatti prendere in considerazione *crash failures* nei quali, sostanzialmente, assumiamo che la macchina non risponde semplicemente perché si è fermata. In generale potremmo avere altri tipi di comportamenti, nei quali la macchina non comunica in modo veritiero la sua condizione. Si tratta dei cosiddetti *Byzantine failures* dal nome di un noto problema teorico: il problema dei generali bizantini. A seconda del sistema dunque, possiamo avere restrizioni o condizioni differenti. É stato dimostrato [8], ad esempio, che il problema del consenso può essere risolto solo per sistemi sincroni, poiché per questi esiste un limite superiore al tempo di risposta delle macchine. In un sistema asincrono una macchina che non risponde potrebbe essere ancora funzionante ma per qualche motivo sta impiegando molto tempo a trasmettere la sua risposta all'appello. Nel caso di guasti di tipo bizantino, esiste [9] un limite inferiore al numero di macchine per poter risolvere il consenso. Per un sistema di n nodi, siano f quelli non funzionanti, non esiste alcun algoritmo in grado di risolvere il problema del consenso se:

$$n \leq 3f \tag{2.2}$$

Oltre a questo, si deve considerare il tema del rispetto delle garanzie del teorema CAP. Questo, seppur non rappresenta un vincolo vero e proprio, è una forte indicazione sul come orientarsi per l'architettura del proprio sistema.

Volendo infatti garantire la consistenza dei dati ad esempio, è facile dimostrare che questo è possibile con un numero dispari di nodi. Per un sistema distribuito in grado di tollerare un numero n_f di guasti, avremo che:

$$n_f = \frac{\lceil N \rceil}{2} + 1 \tag{2.3}$$

Dove N è il numero di nodi di cui abbiamo bisogno. Possiamo partire invece dalla necessità di garantire la disponibilità del dato, prendendo in considerazione il numero di nodi in funzione dei guasti tollerabili:

$$N = 2n_f + 1 \tag{2.4}$$

Ad esempio, volendo un sistema in grado di tollerare almeno un fallimento, avremo bisogno di 3 nodi. Un sistema con un numero pari di macchine ha lo stesso numero di guasti tollerabili. Dunque, quella macchina in più non garantirebbe nulla in termini di replicazione del dato, rimanendo tuttavia una spesa in termini di risorse. Ovviamente a questo tema si lega quello del consenso, che si basa sulle macchine ancora funzionanti in seguito ad un fallimento.

3 Valutazione delle tecniche

Una volta individuate le nostre esigenze, tenendo conto dei fattori illustrati nel capitolo precedente, dobbiamo entrare nel merito delle scelte tecniche che possono essere state fatte dai diversi database. Vogliamo introdurre quelle che possiamo definire generalmente come le tecniche mediante le quali si realizzano le caratteristiche del capitolo precedente. Non esiste una tecnologia migliore in senso assoluto. Esistono diverse tecnologie più o meno adatte alle nostre esigenze. Ogni database può adottare specifiche soluzioni tecniche per svolgere uno stesso compito. Non entreremo in questo dettaglio, lasciandone l'approfondimento al lettore. È importante però sottolineare che, dietro le tecniche esposte nelle sezioni che seguono, esistono ulteriori livelli di complessità che caratterizzano i prodotti sul mercato. Tale complessità è spesso oggetto di modifiche, aggiornamenti, miglioramenti o anche completi stravolgimenti ad opera degli sviluppatori. Un quadro aggiornato e dettagliato della situazione rischierebbe di diventare obsoleto in un tempo relativamente breve rispetto alla vita media di un documento come questo.

3.1 Replicazione

La replica consiste in varie tecniche con le quali un database replica su varie macchine i dati che deve gestire. Attraverso queste tecniche possiamo dunque garantire l'accessibilità al dato, la sua integrità, assicurandocene anche la durabilità nel tempo. Tuttavia, potremmo incorrere in problemi relativi alla sincronizzazione tra macchine differenti e, necessariamente, saremo costretti a ricercare un compromesso tra consistenza da una parte e latenza e disponibilità dall'altra. Da qui nascono le diverse soluzioni offerte nei diversi database. Dunque, lo stesso concetto di replicazione del dato può essere ottenuto in diversi modi. Si possono avere sistemi di replica in modo *sincrono*, pre, quali la replica del dato si propaga in modo sincrono a tutte le macchine, oppure sistemi che svolgono questo lavoro in modo *asincrono*. Mentre il primo premia la consistenza del dato tra le macchine al prezzo di una maggiore latenza, il secondo è sicuramente più veloce, permettendo però che vi possano essere dati diversi su macchine diverse. Ci sono poi sistemi di tipo *master-slave* o sistemi di tipo *master-master* a seconda di come è organizzata la politica di scrittura del dato ed accettazione delle repliche. Ovvero se le modifiche si possono scrivere su di una sola macchina (*master*) per poi essere propagate alle altre (*slave*), oppure se la scrittura è consentita su una qualsiasi delle macchine che poi si riallineano per quanto riguarda le repliche (in entrambe i casi questo può avvenire in modo sincrono o asincrono).

3.2 Sharding

Con *sharding* si indica il partizionamento orizzontale dei dati su un sistema distribuito. Sostanzialmente sono diverse tecniche con le quali i dati vengono suddivisi tra i diversi nodi di un cluster, consentendo di scalare le risorse di archiviazione. Vi si riferisce anche come *shared-nothing architecture* per indicare che non vengono

condivise le risorse delle singole macchine se non in termini di spazio di archiviazione. Questo evita *single points of failure*, garantendo la scalabilità in termini di troughput e volume di dati. Sostanzialmente possono essere tre i metodi per realizzare lo sharding. Il *range-sharding*, o sharding dinamico, nel quale i dati vengono impacchettati in range contigui e ordinati. Questa tecnica ha bisogno di un coordinatore per gestire le assegnazioni dei pacchetti ai diversi *shards*, ossia le macchine che si devono dividere il carico. Tale sistema di coordinazione deve essere anche in grado di accorgersi dell'eventuale formarsi di *hotspot*, riequilibrando i carichi. É un metodo particolarmente utile per modelli dati come il wide-column o i document stores. Altro metodo è l'*hash-sharding* o sharding algoritmico. In questo caso i dati vengono spezzettati in frammenti (*hash*) ai quali viene assegnata una chiave primaria (*partition key*). Successivamente i vari frammenti vengono distribuiti sulle macchine secondo una mappa delle chiavi. L'hashsharding ha il vantaggio di mantenere equilibrata la suddivisione sulle macchine, consentendo controlli veloci ma rendendo quasi impossibili ricerche complete sui dati. Questa tecnica è adottata principalmente da database per key-value e per qualche wide-column. Un grande limite di questa tecnica è quello di dover rimappare le chiavi ogni volta che si vuole aggiungere uno shard. Per questo motivo è stata sviluppata una variante, il *consistent hashing* secondo il quale soltanto una parte dei dati deve essere rimappata in caso di ampliamenti di sistema. L'ultimo metodo è l'*entity-group sharding* nel quale i dati vengono suddivisi raggruppandoli per entità, motivo questo per cui viene spesso usato per lo sharding di database relazionali. I gruppi di identità si possono implementare sia in modo algoritmico che dinamico, usualmente però si usa il metodo dinamico poiché le dimensioni dei gruppi possono variare considerevolmente.

3.3 Altre tecniche fondamentali

Vi sono altre tecniche che andrebbero prese in considerazione. In questo documento verranno solo citate, senza entrare nel dettaglio. Tra queste la principale è sicuramente lo *storage engine*, il motore che gestisce i dati fisici. Semplificando, la scelta si muove su due direttive concorrenti: quella temporale (che mira ad abbattere i tempi di latenza) e quella spaziale (che mira ad estendere lo spazio di archiviazione). La prima, cui ci si riferisce come *inmemory*, gestisce i dati scrivendoli in memoria, ossia nella RAM. Questo aumenta di molto le prestazioni in termini di velocità ma consente di gestire un minore troughput. Per aumentare lo spazio si deve investire in memoria, generalmente più costosa rispetto alle soluzioni di storage (HDD o SSD). La seconda, definita *append-only*, scrive invece i dati su spazi fisici di archiviazione, premiando la disponibilità di spazio al costo di maggiore latenza ma anche minori investimenti.

Anche il sistema di processazione delle interrogazioni è determinante nelle prestazioni del sistema. Se tutti i database prevedono una chiave primaria che consente richieste per identificatori unici, alcune tecnologie prevedono sistemi di chiavi secondarie, come nel caso delle ricerche full-text (si tratta di un caso speciale di chiave secondaria).

3.4 Esempio di una tabella di confronto

Si possono costruire tabelle di raffronto tra diverse soluzioni tecnologiche concorrenti. In queste tabelle conviene partire dalle caratteristiche e tecniche principali per poi addentrarsi sempre più nel merito di specifiche configurazioni. A mero titolo esemplificativo sono mostrate in Tabella 3.1 due soluzioni relative a due soluzioni di database differenti molto diffuse, confrontate relativamente alle diverse tecniche illustrate in questo capitolo.

	mongoDB	Cassandra
Modello Dati	Document	Wide-Column
Teorema CAP	CP	AP
Prestazioni in scrittura	Alte (append-only I/O)[1]	Alte (append-only I/O)
Replicazione	Master-slave, sincrona	Consistent hashing
Sharding	dinamica di default, configurabile come algoritmica	Consistent hashing

Tabella 3.1: Confronto qualitativo tra due soluzioni per database.

[1] Nella versione Enterprise di mongoDB si può configurare uno storage engine di tipo in-memory mentre quello di dafault, WiredTiger, è di tipo append-only.

4 Diffusione, popolarità, supporto e tendenze

Nella valutazione di una tecnologia, oltre agli aspetti tecnici, è opportuno prenderne in considerazione anche altri. Uno tra questi è sicuramente la *diffusione* di tale tecnologia, ovvero quanto questa sia affermata come soluzione tecnologica. Parallelamente alla diffusione, un altro indicatore utile è quella che possiamo definire *popolarità* della tecnologia. Oltre al numero di sistemi adottati è difatti utile capire quanto ampia e vitale sia la comunità di suoi utenti e sviluppatori. Maggiore è l'attività di queste comunità, maggiore è la probabilità di ricevere assistenza o trovare problemi simili e, auspicabilmente le soluzioni a questi. Optare per tecnologie troppo di nicchia potrebbe voler dire condannarsi ad essere gli unici ad avere un determinato problema, essendo la comunità di utilizzatori troppo ristretta. D'altro canto, scegliere una tecnologia solo perché di moda in quel momento, potrebbe portare all'inconveniente di ritrovarsi con un sistema eccessivamente chiuso, che ci obbliga a continue spese per garantirci supporto e aggiornamenti (*lock-in*).

Oltre a questi indicatori, che descrivono lo stato dell'arte, è necessario rivolgere l'attenzione anche all'interpretazione e l'estrapolazione di alcune tendenze, per cercare di capire quello che potrebbe essere il destino a medio e lungo termine della tecnologia che stiamo valutando. Popolarità e diffusione sono valori quantificabili. Stimare l'evoluzione di una tecnologia richiede invece di fare previsioni su sistemi altamente complessi che includono, tra le possibili variabili, fattori difficilmente quantificabili quali mode e tendenze culturali, orientamenti politici riguardo la regolamentazione di nuove tecnologie (ad esempio DRM), equilibri geopolitici[1]. Nonostante sia stata oggetto di diverse critiche, l'*Hipe Curve* proposta dalla società di consulenza Gartner, individua efficacemente quelle che possono essere ragionevolmente considerate le fasi del ciclo di vita di una tecnologia. Dunque, seppure non ci consenta di quantificare il fenomeno, ci suggerisce indicazioni sull'approccio al problema, che deve necessariamente tener conto di tali fasi. Parallelamente a questo, altri risultati possono invece fornire basi quantitative che ci autorizzano a fare alcune ipotesi con le quali avanzare previsioni. In particolare, la *legge di Reed*[2] ci autorizza ad ipotizzare che un crescente numero di utenti e sviluppatori organizzati in communities, social, gruppi e sottogruppi di discussione, forum, determinano conseguentemente un elevato successo della tecnologia. Questo non assicura necessariamente la sua longevità. Da questo punto di vista conta ovviamente l'età della tecnologia. Un alto numero di utenti, progetti e sviluppatori relativi ad una tecnologia *anagraficamente* datata è sintomo comunque di una sua certa vitalità che la rende, a dispetto dell'età, una tecnologia ancora interessante. All'inizio della loro vita, le tecnologie hanno tutte un periodo *di innesco* utile a farsi conoscere, durante il quale non ci sono grandi numeri. Successivamente, se per qualche motivo dovesse prendere piede, si assiste ad

[1] Proprio mentre viene scritto questo documento è in atto una guerra dei dazi tra Stati Uniti e Cina che ha avuto, al momento, la conseguenza di modificare il supporto di Google ai sistemi operativi Android su smartphone di marca Huawei, cambiando forse le sorti di alcune tecnologie, almeno all'interno del mercato cinese.

[2] 2 La legge afferma che:

il valore di un network aumenta quando i membri formano sottogruppi per collaborare e condividere.

In altri termini:

Dato un insieme $\mathcal{A}$ di *n* elementi, possiamo creare 2^N sottoinsiemi di $\mathcal{A}$

un’impennata fino ad un massimo, per poi osservare una discesa non appena è passato il periodo d’oro dovuto alla novità. Tecnologia longeve possono condividere questo tipo di comportamento con quelle meno fortunate, tuttavia si differenziano perché poi si riprendono assestandosi su valori asintotici nel tempo.
Dunque, i numeri relativi la diffusione e la popolarità, se letti anche nel tempo, offrono buoni indicatori non solo sullo stato di salute nell’immediato, ma possono fornire utili indicazioni anche sui possibili sviluppi. Oltre a questo, ci sono poi delle considerazioni di carattere qualitativo che possono offrire ulteriori indicazioni ragionevoli. Se ad esempio una determinata soluzione tecnologica, grazie a qualche caratteristica che la contraddistingue, risponde a una domanda che è ragionevole aspettarsi aumenterà, sarà altrettanto ragionevole aspettarsi di osservarne un relativo successo.

4.1 Diffusione

In questa sezione mostreremo alcuni dati relativi alla diffusione delle diverse tecnologie. I dati sono stati raccolti dal sito db-engines.com che si occupa di monitorare le diverse soluzioni di database sul mercato. In Figura 4.1 è mostrata la diffusione in termini di numero di sistemi, suddivisi per categoria. Le categorie mostrate rispecchiano sostanzialmente il modello dati principale che gestiscono, dunque ricalcano quanto illustrato nel Capitolo 2, con l’aggiunta di una voce: *search engines*. Questa categoria, seppur non propriamente un DBMS, è spesso presentata in modo concorrenziale ad altre tecnologie per database poiché consente l’interrogazione di documenti, per esempio in formato JSON. Tuttavia, è bene sottolineare la sostanziale differenza tra un motore di ricerca ed un database, poiché nei primi non è prevista l’interazione con il dato in termini di sua modifica. I motori di ricerca consentono ricerche di tipo *full text*, ossia sull’intero corpo testuale contenuto nel documento, ma non consentono determinate operazioni (a meno di passare per strumenti esterni). Se però l’utilizzo del dato prevede una semplice interrogazione, possono essere pensati come valide alternative ad un database. Come si può vedere chiaramente, ancora oggi la soluzione per database più diffusa è quella dei relazionali. Insieme a questo dato però, dobbiamo andare ad osservare gli andamenti negli ultimi anni se vogliamo capire quali soluzioni si stanno diffondendo maggiormente in relazione a specifiche necessità (dunque ai relativi modelli dati più indicati). Rimanderemo questo approfondimento nella sezione successiva, quando sarà mostrato l’andamento di singoli prodotti.

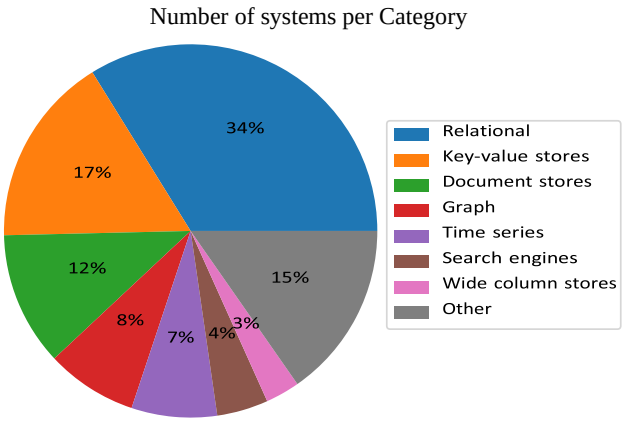

Figura 4.1: Numero di sistemi per categoria di database.

Fonte: www.db-engines.com

4.2 Principali tendenze

Come anticipato, oltre al numero di database in circolazione, è interessante capire quali sono gli andamenti che li interessano. Un primo parametro da mostrare è quello relativo alla tipologia di licenze sotto le quali vengono rilasciati i vari prodotti. Viene proposta qui una suddivisione semplificata tra prodotti Commerciali e prodotti Open Source. All'interno di queste categorie esistono sottoinsiemi definiti dalle varie tipologie di licenze proprietarie o aperte, che esulano tuttavia dallo scopo di questo documento. É importante analizzare questo parametro per capire come si sta orientando il mondo in questo settore. Se ancora alla fine degli anni '90 le soluzioni proprietarie erano quasi una scelta obbligata, oggi si assiste (Figura 4.2) ad un progressivo *switch* verso soluzioni open source. Questo è dovuto al fatto che il crescente mercato di soluzioni di database non relazionali è rappresentato per la stragrande maggioranza da prodotti open source. Ognuna di queste tecnologie prevede versioni totalmente aperte (generalmente indicate con il termine *Community*) e di versioni a pagamento (*Enterprise*) basate su diversi modelli di business (servizi e/o tool aggiuntivi, assistenza, formazione, ...).

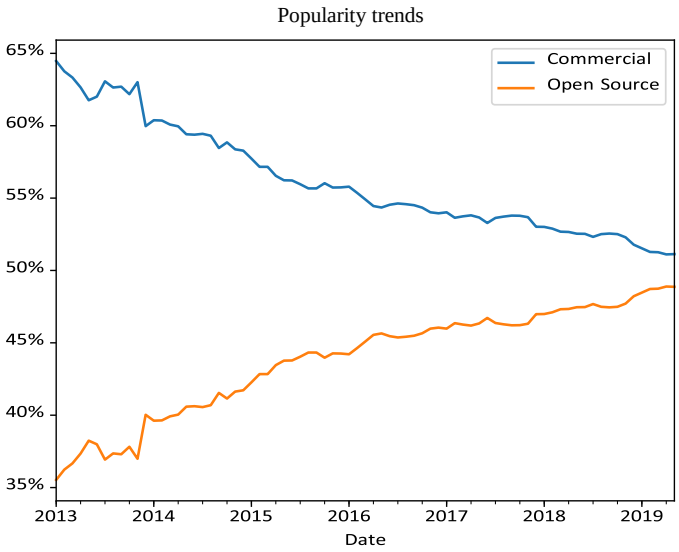

Figura 4.2: Andamento della diffusione per tipologia di licenza.
Fonte: www.db-engines.com

4.2.1 Commerciale ed Open Source

In Figura 4.3 sono mostrate diverse categorie di database suddivise per tipologia di licenza (Commerciale o Open Source). Si può notare come, mentre nel mondo relazionale vi sia una preponderanza di soluzioni commerciali, in quelle non relazionali l'open source sia maggioritario. Ogni categoria di database ha diversi prodotti sul mercato a rappresentarla. Per semplificazione verrà mostrato, per ogni categoria, solo il primo risultato in termini di diffusione, per consentire di avere alcuni punti di riferimento. Chiaramente, se si fosse orientati verso un determinato modello di dati, potrebbe essere utile estendere la valutazione anche a soluzioni minori per diffusione, ma che magari offrono vantaggi in termini di tecniche specifiche adottate. In Tabella 4.1 sono illustrati questi prodotti rappresentativi delle rispettive categorie. Unica eccezione sono tre soluzioni SQL. Per quanto riguarda MySQL ed Oracle poiché quelle che principalmente occupano il mercato. Discorso a parte va fatto per PostgreSQL, per il quale rimandiamo alla sezione finale di questo capitolo.

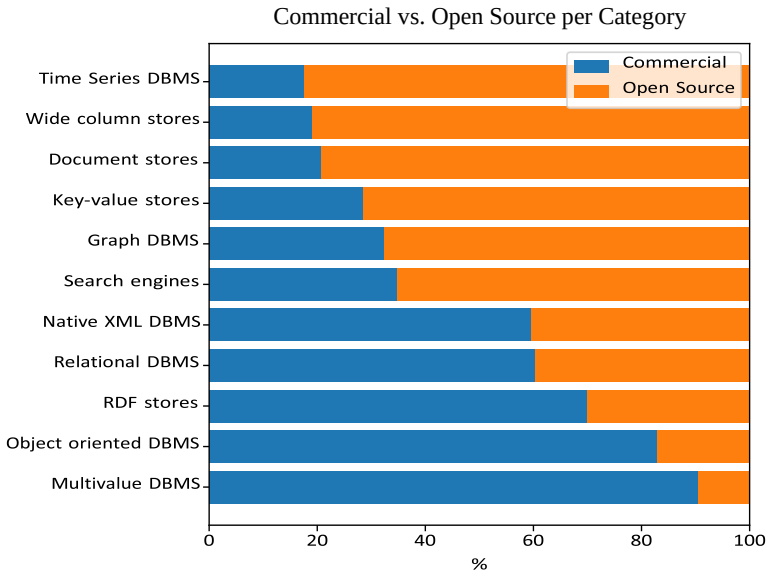

Figura 4.3: Diffusione di sistemi commerciali ed open source per categoria.

Fonte: www.db-engines.com

Prodotto	Categoria
Cassandra	Wide-column
mongoDB	Document store
InfluxDB	Time series
Redis	Key-value
Elasticsearch	Search engine
Oracle	SQL
MySQL	SQL
PostgreSQL	Multi-model

Tabella 4.1: Principale prodotto per differenti categorie di database

4.3 Popolarità

Per misurare le dimensioni della comunità attorno ad una determinata tecnologia, possiamo riferirci ad alcuni indicatori che possiamo ottenere da portali molto diffusi tra utenti e sviluppatori. Ad esempio, il numero di domande relative ad una data tecnologia su portali come Stack Overflow[1] o il numero di progetti (*repositories*)

[1] https://stackoverflow.com/

legati alla tecnologia in esame su Git Hub[1] sono indicatori quantitativi molto rappresentativi. In Tabella 4.2 sono riportati questi valori per alcuni database. Riferendosi a questi indicatori possiamo osservare quanto alcune tecnologie siano ben rappresentate nel mondo degli utenti e degli sviluppatori mentre altre, magari più giovani o meno affermate, siano scarsamente supportate. La tabella precedente vuole essere d'esempio e suggerimento riguardo a questi valori che è bene andare a valutare come elementi utili alla decisione che si dovrà prendere.

Database	Stack Overflow quests	Github repositories
CouchDB	5594	5308
Redis	15868	49398
Cassandra	15929	9791
Elasticsearch	38126	25385
mongoDB	113239	98821
MySQL	561126	130050
PostgreSQL	99246	29571

Tabella 4.2: Alcuni indicatori di *popolarità* sono quelli relativi ai due noti siti, riferimento delle comunità di utilizzatori e sviluppatori. Dati riferiti a maggio 2019.

4.3.1 Ranking da db-engines.com

Sul sito db-engines.com viene riportata una classificazione rispetto ad un indice di ranking che include quanto illustrato fino ad ora. In particolare, l'indice è costruito osservando i seguenti valori:

- Numero di citazioni sui siti (dal numero di risultati nei motori di ricercar quali Google, Bing, ...)
- Interesse generale (dalla frequenza di ricerche su Google)
- Frequenza di discussioni tecniche (dal numero di domande ed utenti interessati su Stack Overflow e DBA Stack Exchange)
- Numero di offerte di lavoro nel quale il sistema è citato (Indeed, Simply Hired)
- Numero di profili in network professionali nei quali viene citato il sistema (LinkedIN, Upwork)
- Rilevanza sui social networks (dal numero di tweets nei quali viene citato il sistema)

In base a questi indicatori, vengono costruiti dei punteggi per ogni prodotto. Riportiamo in Figura 4.4 i risultati relativi ai prodotti della Tabella 4.1 su scala semi-logaritmica[2].

[1] https://github.com/

[2] Questo a causa dell'ampio divario tra prodotti relazionali e quelli non relazionali.

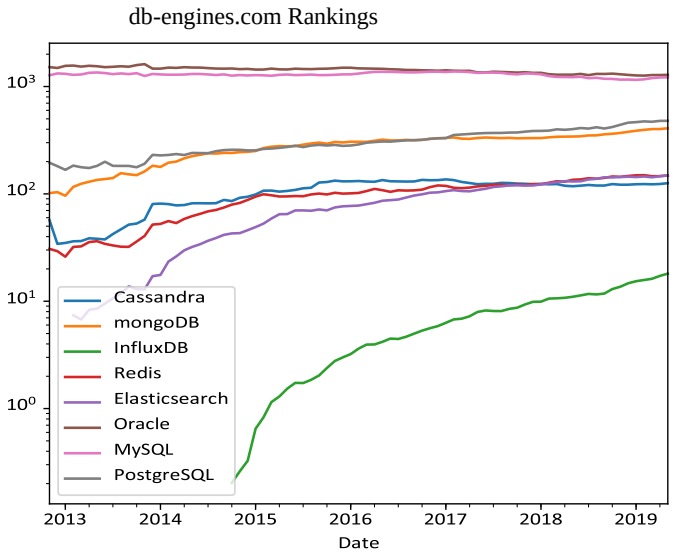

Figura 4.4: Ranking secondo il portale db-engines.com dei principali database per categoria.
Fonte: www.db-engines.com

Come si può osservare, i principali attori sul mercato rimangono Oracle e MySQL. Seguono PostgreSQL e l'unico prodotto non relazionale che si attesta su ordini di grandezza confrontabili: mongoDB. Oltre a questi è da notare la forte crescita di InfluxDB, database specificatamente pensato per serie temporali, probabilmente da imputare alla forte diffusione di progetti nell'IoT, spesso caratterizzati da flussi di dati provenienti da sensoristica. Oltre a questo, dato infatti, è utile mostrare anche quello relativo ai tassi cambiamento di questi punteggi, ossia quanto cambiano di anno in anno. Osservando questi valori in Figura 4.5 è evidente quanto alcune tecnologie, a dispetto dello stato attuale, possiamo aspettarci che abbiano un ruolo nel futuro prossimo e che quindi sappiano mantenersi vive.

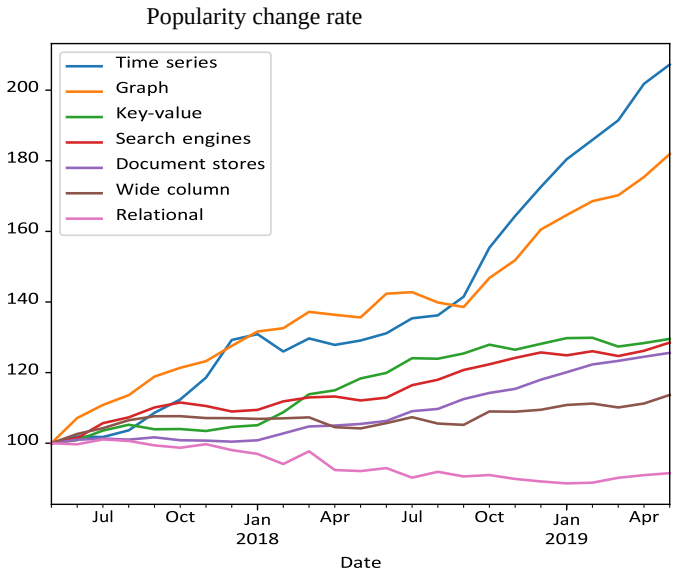

Figura 4.5: Tasso di cambiamento del ranking per categoria di database.
Fonte: www.db-engines.com

In questo ultimo grafico si può osservare l'evidente tasso di crescita con il quale si stanno affermando alcune soluzioni innovative di tipo non relazionale tra i quali spiccano i database per serie temporali e quelli per modelli a grafo, con valori di crescita nettamente superiori alla media. Da notare è anche il corrispettivo tasso in flessione negativa registrato per i database relazionali.

4.4 Polyglot Persistence

Una conferma a quanto illustrato fino ad ora, è rappresentata dall'emergere di un concetto che esprime un nuovo paradigma di approccio alla gestione dei dati, quello che viene definito [10] *Polyglot Persistence*, ovvero l'idea di usare tecnologie di archiviazione diverse per gestire differenti necessità mediante la stessa soluzione software. Questo approccio, dettato dalla crescente esigenza di gestire in modo opportuno e performante dati eterogenei nel medesimo ambiente, ha condotto allo sviluppo di soluzioni definite *multi-modello*. Anziché cercare di forzare una tecnologia mono modello per gestire differenti tipi di dati, tentando di strutturarli secondo quel modello, l'orientamento è quello verso soluzioni che offrano la possibilità di gestire diversi dati con i rispettivi diversi modelli che meglio li descrivono. Anche perché, in uno scenario in continua evoluzione, potrebbero cambiare le proprie esigenze e ci si potrebbe ritrovare nuovamente a dover reinventare il proprio database per gestire i nuovi dati, spesso perdendo in prestazioni, oltre che aggiungendo livelli di complessità che, tra le altre cose, necessiteranno di ulteriore manutenzione. Una scelta multi-modello ha il grande vantaggio di permettere di investire nella stessa tecnologia senza precludersi la possibilità di accogliere l'eterogeneità dei dati presenti e futuri. Di seguito vengono citati brevemente due esempi, simili rispetto questo paradigma, ma che si differenziano per età e modalità con cui hanno sviluppato il concetto.

4.4.1 Due casi significativi

PostgreSQL Questo database nasce nel 1996 sostanzialmente come soluzione SQL open source. L'approccio degli sviluppatori è stato però immediatamente quello di concentrarsi sul motore, rendendolo robusto ed essenziale. Questo ha fatto si che la sua crescente comunità (è il database open source più diffuso) abbia potuto sviluppare, sullo stesso motore, soluzioni per ogni tipo di modello dati, rendendolo una tecnologia spesso usata da chi fa ricerca in questo settore. Ad oggi Postgres (generalmente senza il diretto riferimento al SQL) ha la possibilità di adattarsi ad ogni esigenza, dal classico database relazionale, a quello per dati georiferiti (PostGIS).

ArangoDB É un database generalmente inquadrato nella categoria a grafo anche se in realtà è un multi-modello a tutti gli effetti. Questo giovane database (la prima stable release è del 2018) risponde perfettamente a questo approccio consentendo la gestione di diversi modelli di dati all'interno dello stesso ambiente.

5 Conclusioni

Sono state valutate diverse tecnologie per la gestione di dati, partendo da un set di use case definito dal Progetto EcoDigit, estendendosi ad altri utilizzi di interesse della Divisione DTE-ICT, al fine di individuare una soluzione che rispondesse ad ampio spettro, in modo robusto ed elastico. Per questo scopo sono stati esposti alcuni fattori chiave che entrano in gioco nella scelta della tecnologia più opportuna per le proprie necessità, seguendo lo schema di una metodologia di analisi. Nel loro lavoro, Gessert e collaboratori [11] propongono un tool di analisi basato su una serie di criteri con i quali costruiscono un albero decisionale, mostrato in Figura 5.1, utile a valutare la soluzione più appropriata al proprio caso. Un approccio simile può essere adottato, considerando di dover sempre svolgere un'attività di aggiornamento su quelle che sono le soluzioni tecnologiche del momento, dato che parliamo di un settore in continua evoluzione. In particolare, nel corso di questo studio, è stato preso in esame il caso rappresentato da mongoDB. Alla luce delle caratteristiche di scalabilità, grazie allo sharding (tra l'altro configurabile in modalità dinamica o algoritmica), della flessibilità offerta dal modello di dati a documento con specifiche configurazioni nel caso di use case tipici quali serie temporali (il database prevede la possibilità di *impacchettare* questo tipo di dati in *bucket*, ossia pacchetti di range temporali definiti dall'utente), delle alte prestazioni in scrittura e lettura oltre all'ampia diffusione e popolarità, è emerso che questa soluzione rappresenta una scelta solida e al tempo stesso capace di adattarsi ad una molteplicità di casi. Nell'ambito del Progetto EcoDigit (attualmente ancora in corso), in particolare nella definizione e progettazione del *proof of concept* della componente middleware, si sta valutando l'utilizzo di mongoDB come database di cache per ottimizzare le ricerche.

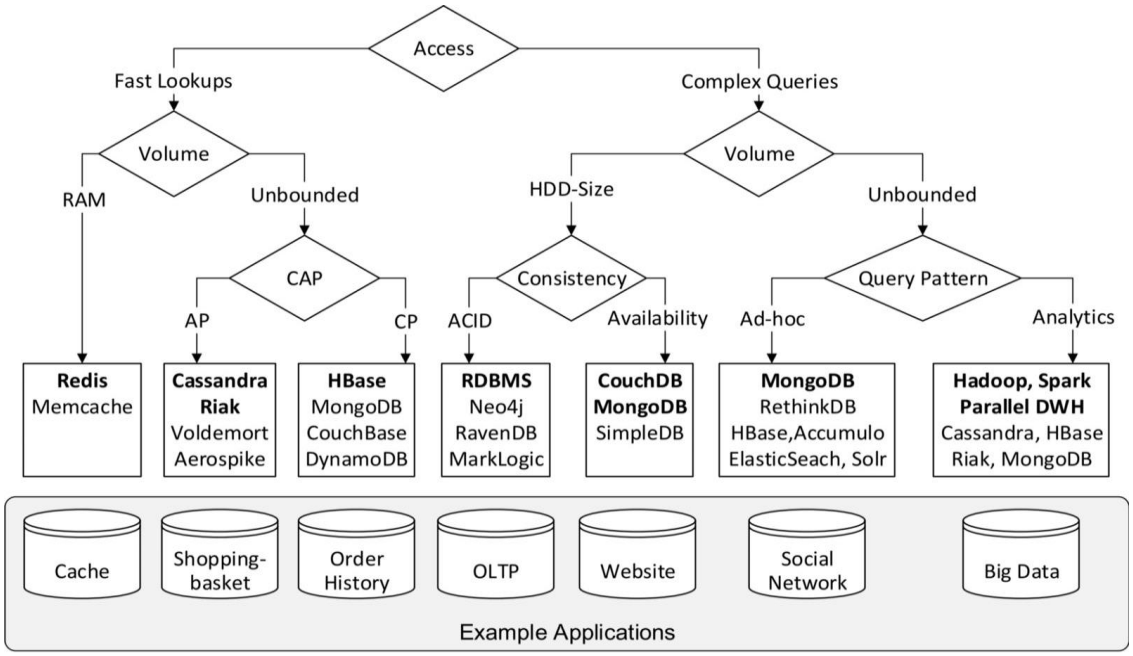

Figura 5.1: Albero delle decisioni per alcune tecnologie di database proposto in [11], relativo a diverse possibili use case.

Bibliografia

[1] D. Waitzman, "Standard for the transmission of IP datagrams on avian carriers".

[2] D. Singh and C. K. Reddy, "A survey on platforms for big data analytics," *Journal of Big Data,* vol. 2, no. 1, p. 8, 9 12 2015.

[3] P. J. Sadalage and M. Fowler, NoSQL distilled : a brief guide to the emerging world of polyglot persistence, Addison-Wesley, 2012.

[4] D. Pritchett and Dan, "BASE: AN ACID ALTERNATIVE," *Queue,* vol. 6, no. 3, pp. 48-55, 1 5 2008.

[5] M. Pease, R. Shostak and L. Lamport, "Reaching Agreement in the Presence of Faults," *JOURNAL OF THE ACM,* vol. 27, pp. 228--234, 1980.

[6] P. Mahajan, L. Alvisi and M. Dahlin, *Consistency , Availability , and Convergence,* 2011.

[7] J. D. C. Little, "A Proof for the Queuing Formula: L = λ W," *Operations Research,* vol. 9, no. 3, pp. 383-387, 1 6 1961.

[8] I. B. Lars George, Jan Kunigk, Paul Wilkinson, Architecting Modern Data Platforms - O'Reilly Media, O'Reilly, 2018, p. 636.

[9] M. Kleppmann, Designing data-intensive applications : the big ideas behind reliable, scalable, and maintainable systems.

[10] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine and D. Lewin, "Consistent hashing and random trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC '97*, New York, New York, USA, 1997.

[11] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Computing Surveys,* vol. 15, no. 4, pp. 287-317, 2 12 1983.

[12] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News,* vol. 33, no. 2, p. 51, 1 6 2002.

[13] F. Gessert, W. Wingerath, S. Friedrich and N. Ritter, "NoSQL database systems: a survey and decision guidance," *Computer Science - Research and Development,* vol. 32, no. 3-4, pp. 353-365, 3 7 2017.

[14] M. J. Fischer, N. A. Lynch and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," 1983.

[15] S. B. Davidson, H. Garcia-Molina and D. Skeen, "Consistency in a partitioned network: a survey," *ACM Computing Surveys,* vol. 17, no. 3, pp. 341-370, 1 9 1985.

[16] E. F. Codd, "Relational Completeness of database," *DATABASE SYSTEMS,* p. 38, 1972.

[17] E. A. Brewer and E. A., "Towards robust distributed systems (abstract)," in *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing - PODC '00*, New York, New York, USA, 2000.

[18] E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer,* vol. 45, no. 2, pp. 23-29, 2 2012.

[19] H. Attiya and A. Bar-noy, "Sharing memory robustly in message-passing systems," *Journal of the ACM (JACM)*, no. 1, pp. 124-142, 1995.

[20] H. Attiya and J. Welch, Distributed Computing, Hoboken, NJ, USA: John Wiley & Sons, Inc., 2004.

[21] D. Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story," *Computer*, vol. 45, no. 2, pp. 37-42, 2 2012.

ENEA
Servizio Promozione e Comunicazione
www.enea.it

Stampa: Laboratorio Tecnografico ENEA - C.R. Frascati
novembre 2019