Big Data Analytics

Big Data

Marta Caffagnini
Matricola: 168007
307930@studenti.unimore.it

20/11/2021

CAP theorem: paradigmi e osservazioni

Indice

Abstract

Con lo sviluppo del Web è cresciuta la mole di dati disponibili e di conseguenza la necessità di gestire database sempre più grandi e processare big data in modo efficace. Per non perdere performance, è cresciuto il bisogno di uno scaling orizzontale dei dati. Occorreva un tipo di database in grado di gestire un grande numero di richieste contemporaneamente. Così si inseriscono i database NoSQL che guadagnano sempre più popolarità con l'impiego nei sistemi distribuiti, portando alla nascita di nuovi paradigmi (BASE).

Nel 2000 Eric Brewer ha introdotto il CAP theorem che esprime il trade-off tra consistenza, disponibilità, tolleranza alle partizioni. In questa relazione sono raccolte le riflessioni nate dal CAP theorem ed è descritto il contesto dei sistemi distribuiti in cui è posto. Non essendo possibile evitare le partizioni in un sistema, sono esposte delle strategie di gestione delle stesse. Vengono inoltre mostrati ulteriori parametri e possibilità che il CAP theorem non ha preso in considerazione.

1. Introduzione

Il CAP theorem afferma che un sistema distribuito può garantire solo 2 delle 3 proprietà desiderabili: consistenza, disponibilità, tolleranza alle partizioni.
Un sistema distribuito è una rete con più nodi che girano in modo indipendente l'uno dall'altro.
Tutti i siti sono a conoscenza dell'esistenza degli altri e se necessario cooperano e comunicano.
Il CAP theorem è anche chiamato teorema di Brewer, nome di colui che lo ha formulato nel 2000 e che ha coniato l'acronimo BASE.
Il teorema è stato poi formalizzato da Seth Gilbert e Nancy Lynch. [1]

1.1 Definizione delle tre proprietà

Le 3 proprietà sono:

Coerenza (C). Un sistema è coerente se tutti i siti hanno gli stessi dati contemporaneamente e rispondono alla stessa interrogazione allo stesso modo. Questo è realizzabile se a ogni modifica dei dati tutti i nodi sono aggiornati istantaneamente prima che la scrittura sia definita conclusa.

Disponibilità (A). Un sistema è disponibile se letture e scritture sono sempre eseguite e i nodi devono essere sempre disponibili a richieste da parte di utenti.

Tolleranza alle partizioni (P). Una partizione è un'interruzione delle comunicazioni all'interno del sistema. Nonostante la partizione, il sistema, per essere definito tollerante, deve continuare a funzionare e le operazioni devono essere eseguite, anche se alcuni componenti non sono disponibili. [3]

1.2 Tipi di database NoSQL

I database NoSQL (non relazionali) sono ideali per le applicazioni di rete distribuite, perché sono più adatti allo scaling orizzontale, rispetto ai database SQL relazionali. Possono quindi essere scalati velocemente aggiungendo piccoli e più economici server in una rete formata da più nodi connessi tra loro.
Lo scaling verticale consiste nel trasferire l'applicazione a computer più potenti. Questa strategia ha dei

limiti: è economicamente molto costosa e occorrono computer troppo potenti con una grande capacità di memoria. [3]

Lo scaling orizzontale offre più flessibilità, ma è più complesso. Un grande numero di computer vengono collegati in rete formando un cluster con tante CPU e memoria.
Il database NoSQL rende più semplice lo scaling orizzontale. Nei database relazionali è importante la consistenza, di conseguenza tutti i nodi dovrebbero essere sincronizzati istantaneamente a ogni modifica.
I database NoSQL si focalizzano sull'availability e sono progettati per gestire errori di questo tipo. [2]

I database NoSQL sono classificati in base alle due caratteristiche CAP che supportano:

- Database CP: un database CP garantisce coerenza e tolleranza alle partizioni, ma non la disponibilità. Quando si verifica una partizione tra due nodi qualsiasi, il sistema rende non disponibile il nodo non coerente fino a quando la partizione non viene risolta. [4]
Un esempio di database CP è MongoDB, un sistema di gestione del database NoSQL. [2]

- Database AP: un database AP fornisce la disponibilità e la tolleranza alle partizioni, ma non la coerenza. Quando si verifica una partizione, tutti i nodi rimangono disponibili, ma quelli alle estremità della partizione potrebbero restituire una versione meno recente di dati rispetto agli altri. Quando la partizione viene risolta, solitamente i database AP sincronizzano i nodi per risolvere tutte le incoerenze nel sistema. [4]
Un esempio di database AP è Cassandra Apache, un sistema di gestione del database NoSQL. [2]

- Database CA: un database CA fornisce coerenza e disponibilità su tutti i nodi, ma non assicura tolleranza agli errori. [4]

MongoDB

MongoDB è un sistema che usa il meccanismo di replicazione Master-Slave: viene definito un master e uno o più slave. Quindi ogni set di repliche ha un solo nodo primario che riceve tutte le operazioni di scrittura e lettura. Tutti gli altri nodi dello stesso set di repliche sono nodi secondari che replicano le operazioni del nodo primario e lo applicano al loro dataset.
Quando il nodo primario diventa non disponibile, il nodo secondario con i dati più recenti viene promosso come nuovo nodo primario. Dopo che tutti gli altri nodi secondari si saranno aggiornati con il nuovo master, il cluster tornerà disponibile. Poiché non si possono effettuare richieste di scrittura durante questo intervallo, i dati rimangono coerenti in tutta la rete. [2] [4]

Cassandra

A differenza di MongoDB, Cassandra ha un'architettura senza master e, di conseguenza, ha più punti di errore, invece di uno solo.
Cassandra, essendo un database AP, garantisce la disponibilità e la tolleranza alle partizioni ma non può garantire sempre la coerenza. Dato che Cassandra non ha un nodo master, tutti i nodi hanno lo stesso ruolo e devono essere disponibili continuamente.
I dati sono replicati su tutti i nodi del cluster e diventano incoerenti solo in caso di una partizione di rete. Le incoerenze vengono risolte rapidamente, perché Cassandra offre una funzionalità che aiuta i nodi ad aggiornarsi all'ultima versione. [2] [4]

2. BASE e ACID

Alla fine degli anni ’90 Eric Brewer conia l’acronimo BASE che si fonda sul CAP theorem e definisce proprietà che si concentrano sulla disponibilità. Questa si ottiene tollerando fallimenti parziali senza che il sistema fallisca del tutto. [3]

Le proprietà indicate da BASE sono:

- Basically available: il sistema deve garantire la disponibilità dei dati, come specificato dal CAP theorem; quindi ci sarà una risposta a ogni richiesta. I database NoSQL duplicano i dati su vari storage per ottenere un alto livello di replicazione e di conseguenza un alto livello di disponibilità.

- Soft state: non c’è garanzia di consistenza, passato un certo transitorio c’è la possibilità di poter conoscere lo stato. Lo stato del sistema può cambiare nel tempo, anche se non ci sono input.

- Eventually consistent: se il sistema sta funzionando ed è passato abbastanza tempo dall’ultimo input, si può sapere lo stato del sistema e si possono avere garanzie di consistenza. Quando sono effettuati dei cambiamenti ai dati, le modifiche si propagano in modo graduale, un nodo alla volta, fino a far diventare consistente l’intero sistema. [2]

Il paradigma BASE si pone in opposizione al paradigma ACID, tipicamente applicato alle transazioni dei database relazionali.
Una transazione è una sequenza di operazioni raggruppate che può concludersi o con un successo o un insuccesso.
I principi ACID si focalizzano sulla coerenza, al contrario di BASE che dà maggiore importanza alla disponibilità. [2]

Le quattro proprietà ACID sono le seguenti:

- Atomicità. L’atomicità è garantita se o tutte le operazioni della transazione vengono completate oppure nessuna.

- Consistenza. Il database è in uno stato di consistenza quando le transazioni iniziano e finiscono.

- Isolamento. L’isolamento è rispettato se la transazione si comporta come se fosse l’unica a essere eseguita nel database.

- Durata. Le modifiche al database compiute dalla transazione devono essere persistenti anche se si verificano guasti software o hardware. [3]

Con l’aumento della quantità di dati, la crescente diffusione dei sistemi distribuiti delle infrastrutture e l’avvento dei database NoSQL, ACID risulta difficile da rispettare. Per questo i database NoSQL utilizzano il principio BASE.
BASE è più flessibile rispetto a ACID. Se è importante garantire la consistenza, un database relazionale può essere la soluzione più adatta, ma se si hanno molti nodi in un sistema distribuito la consistenza è difficile da raggiungere [2].

3. Osservazioni sul CAP theorem

3.1 La formulazione “2 di 3”

La formulazione "2 di 3" tende a semplificare eccessivamente le proprietà e a incasellare in modo rigido i gestori di database.
I progettisti devono ancora decidere tra consistenza e disponibilità quando sono presenti partizioni, ma ci sono varie possibilità di gestione delle partizioni. [1]

Il "2 di 3" è fuorviante su più fronti. Innanzitutto, dato che le partizioni sono rare, ci sono poche ragioni per perdere C o A quando il sistema non è partizionato. In secondo luogo, la scelta tra C e A può verificarsi molte volte all'interno dello stesso sistema. Non solo i sottosistemi possono fare scelte diverse, ma la scelta può cambiare in base all'operazione o ai dati specifici o l'utente coinvolto.
Infine, la disponibilità è ovviamente continua in una percentuale, ma ci sono anche molti livelli di coerenza e di tolleranza alle partizioni. [1]

3.2 CAP e latenza

Nella sua interpretazione classica, il teorema CAP ignora la latenza, anche se nella pratica, latenza e partizioni sono strettamente legate. [1] [2]

La CAP agisce durante una interruzione di comunicazione, cioè un momento in cui il programma deve prendere la decisione di partizione e scegliere se:

• annullare l'operazione e quindi diminuire la disponibilità,

oppure

• procedere con l'operazione e rischiare l’incoerenza. [1]

Una partizione è un tempo vincolato alla comunicazione. Se non si riesce a raggiungere la coerenza entro il tempo definito, si verifica una partizione e quindi deve essere fatta una scelta tra C e A per questa operazione.
Ritentare la comunicazione per raggiungere la coerenza, significa scegliere C su A.
Si deduce che non esiste una nozione globale di partizione, poiché alcuni nodi potrebbero rilevare una partizione ed entrare in modalità partizione e altri potrebbero non farlo.
I progettisti possono impostare il limite di tempo che definisce la partizione. I sistemi con limiti più stretti probabilmente entreranno nella modalità partition più spesso e in momenti in cui la rete è semplicemente lenta e non partizionata. [1]

Un designer, inoltre, non può scegliere di non avere partizioni. Se la scelta è CA, e poi c'è una partizione, la scelta deve tornare a C o A. È più corretto pensare in modo probabilistico. Se si scegliesse CA, questo significherebbe che la probabilità di una partizione è molto inferiore a quella di altri fallimenti sistemici, come disastri o multipli difetti contemporanei.
Poiché le partizioni sono rare, CAP dovrebbe consentire pienamente C e A per la maggior parte del tempo, ma quando le partizioni sono presenti o percepite, occorre una strategia che rileva partizioni e si comporta di conseguenza.
La strategia dovrebbe avere tre passaggi: rilevare le partizioni, entrare in una modalità di partizione che può

limitare alcune operazioni, e avviare un processo di ripristino per ripristinare la coerenza e rimediare agli errori commessi durante la partizione. [1]

3.3 Gestione delle partizioni

L’obiettivo difficile per i progettisti è quello di limitare gli effetti di una partizione sulla coerenza e sulla disponibilità. L'idea chiave è quella di gestire le partizioni in modo molto esplicito, specificando il rilevamento, il processo di recupero specifico e la pianificazione delle invarianti che potrebbero essere violate durante una partizione. Questo approccio di gestione prevede tre fasi:

• rilevare l'inizio di una partizione

• entrare in modalità partizione esplicita che potrebbe limitare alcune operazioni

• avviare il ripristino della partizione quando la comunicazione è ristabilita. In questa fase si ripristina la coerenza e si sistemano gli errori commessi dal programma mentre nel sistema si verificava la partizione. [1]

Rilevazione partizione

L’esecuzione dell'operazione in una situazione di normalità consiste in una sequenza di operazioni atomiche, e quindi le partizioni iniziano sempre tra operazioni. Una volta che è trascorso il tempo definito nel sistema, viene rilevata una partizione e il nodo entra in modalità partition, perché potrebbe avere operazioni inconsistenti.

Una volta che il sistema entra in modalità partizione, si possono attuare due strategie.

La prima consiste nel limitare alcune operazioni, riducendo la disponibilità del sistema.

La seconda è registrare informazioni aggiuntive sulle operazioni che saranno utili in seguito durante il ripristino della partizione. Un altro comportamento utile è continuare a provare a ristabilire la comunicazione, così il sistema potrà individuare quando termina la partizione. [1]

Modalità partition

Il progettista deve decidere quali operazioni limitare durante la modalità partition. Questa scelta dipende dalle invarianti che il sistema deve mantenere e dalla possibilità di rischiare di violarli e nel caso ripristinarli in seguito durante l’ultima fase di ripristino.

Se un invariante deve essere rispettato durante una partizione, il progettista deve vietare o modificare operazioni che potrebbero violarlo.

In alcuni casi, come la ricarica di una carta di credito, la strategia è registrare l'intento ed eseguirlo dopo il recupero. Il progettista perde A senza che gli utenti se ne accorgano.

Riassumendo, il progettista deve analizzare tutte le operazioni e decidere se queste potrebbero violare l'invariante. In tal caso, si può decidere di vietare, ritardare o modificare l'operazione. [1]

Ripristino della partizione

A un certo punto, la comunicazione riprende e la partizione termina. Durante la partizione, ogni nodo era disponibile e quindi sono stati registrati progressi sul sistema, ma il partizionamento ha ritardato alcune operazioni e violato alcune invarianti.

Il progettista deve risolvere due problemi difficili durante il recupero:

- lo stato del sistema da entrambe le parti deve diventare coerente. Generalmente la strategia più semplice è correggere lo stato corrente iniziando dallo stato al momento della partizione e proseguendo le serie di operazioni, mantenendo coerenti i dati fino ad ottenere uno stesso stato su tutti i nodi.

- eventuali errori commessi durante la modalità partizione devono essere sistemati. Avendo limitato alcune operazioni durante la partizione, si può dedurre quali sono gli invarianti che sono stati violati e si può creare una strategia di ripristino.
Esistono vari metodi per aggiustare gli invarianti, tra cui rendere definitiva e corretta l'ultima scrittura.

I progettisti di sistema non dovrebbero sacrificare ciecamente la consistenza o la disponibilità quando esistono partizioni. Usando l'approccio proposto, si possono ottimizzare entrambe le proprietà attraverso un'attenta gestione delle invarianti durante le partizioni. [1]

4. Conclusioni

I database NoSQL rispondono alle attuali esigenze derivate dai big data.

Per una gestione efficiente di grandi database occorre un sistema distribuito su più nodi, frutto di uno scaling orizzontale. Questo scaling è ben supportato dai database NoSQL.

I database di questa tipologia possono essere classificati in base alle proprietà del CAP che soddisfano.

Il paradigma che governa i database non relazionali è il BASE che si pone in contrapposizione al paradigma ACID, applicato ai database relazionali.

Riguardo il CAP theorem gli studiosi hanno criticato la formulazione rigida secondo cui è possibile soddisfare solo due delle tre proprietà. Si conclude che si possa parlare di vari livelli di tolleranza, disponibilità e consistenza.

Nella tolleranza alle partizioni bisogna anche prendere in considerazione la latenza che ha lo stesso effetto di un'interruzione di comunicazione.

Se non si possono evitare le partizioni bisogna essere in grado di gestirle e definire un piano che coinvolge anche gli invarianti del database.

Le osservazioni sul CAP theorem portano alla conclusione che quando in un sistema distribuito si verifica una partition non bisogna direttamente decidere quale proprietà sacrificare tra la consistenza e la disponibilità. Entrambe le proprietà possono essere ottimizzate gestendo adeguatamente le partizioni e gli invarianti.

5. References

[1] Brewer, E., "CAP twelve years later: How the "rules" have changed," Computer , vol.45, no.2, pp.23,29, Febbraio 2012. doi: 10.1109/MC.2012.37.

[2] Abramova, V.; Bernardino, J., “NoSQL databases: MongoDB vs Cassandra”, International C* Conference on Computer Science & Software Engineering, pp.14,22, Luglio 2013. doi: 10.1145/2494444.2494447.

[3] Pritchett, D., “BASE: An Acid Alternative”, Queue, 6(3), pp.48-55, Maggio/Giugno 2008.

[4] IBM. "Teorema CAP”. https://www.ibm.com/it-it/cloud/learn/cap-theorem/. [Online; accessed 19-November-2021].