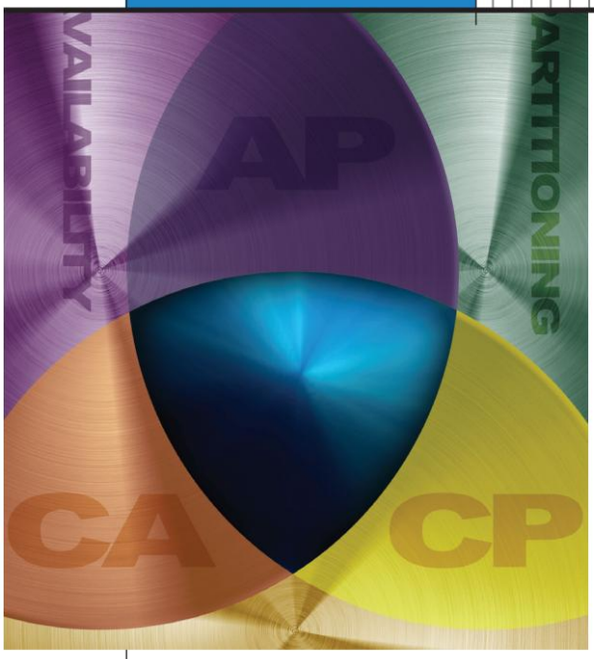

Coerenza Compromessi in Moderno distribuito Sistema di database Progetto

Daniel J. Abadi, *Università di Yale*

L'impatto del teorema CAP sulla progettazione di sistemi di database distribuiti moderni è più limitato di quanto spesso si percepisca. Un altro compromesso, tra coerenza e latenza —ha avuto un'influenza più diretta su diversi DDBS ben noti. Una nuova formulazione proposta, PACELC, unifica questo compromesso con CAP.

Sebbene la ricerca sui sistemi di database distribuiti tems sono iniziati decenni fa, solo di recente l'industria ha iniziato a fare ampio uso di DDBS. Ci sono due fattori principali che determinano questa tendenza. In primo luogo, le applicazioni moderne richiedono un throughput di dati e transazioni maggiore, il che ha portato a un desiderio di sistemi di database scalabili in modo elastico. In secondo luogo, la crescente globalizzazione e il ritmo del business hanno portato alla necessità di posizionare i dati vicino ai clienti che sono sparsi in tutto il mondo. Esempi di DDBS creati negli ultimi 10 anni che tentano di raggiungere un'elevata scalabilità o accessibilità mondiale (o entrambe) includono SimpleDB/Dynamo/ DynamoDB,1 Cassandra,2 Voldemort (http://project-voldemort.com), Sherpa/PNUTS,3 Riak (http://wiki.basho.com), HBase/BigTable,4 MongoDB (www.mongodb.org), VoltDB/H-Store,5 e Megastore.6

I DDBS sono complessi e realizzarli è difficile. Pertanto, qualsiasi strumento che aiuti i progettisti a comprendere i compromessi implicati nella creazione di un DDBS è utile. Il teorema CAP, in particolare, è stato estremamente utile nell'aiutare i progettisti a ragionare attraverso un sistema proposto capacità e nell'esporre l'esagerato clamore di marketing di molti DDBS commerciali. Tuttavia, sin dalla sua dimostrazione formale iniziale,7 il CAP è diventato sempre più frainteso e mal applicato, causando potenzialmente danni significativi. In particolare, molti progettisti concludono erroneamente che il teorema impone determinate restrizioni a un DDBS durante il normale funzionamento del sistema e quindi implementano un sistema inutilmente limitato. In realtà, il CAP pone solo limitazioni di fronte a determinati tipi di guasti e non limita alcuna capacità del sistema durante il normale funzionamento.

Ciononostante, i compromessi fondamentali che inibiscono le capacità dei DDBS durante il normale funzionamento hanno influenzato le diverse scelte progettuali dei sistemi più noti. In effetti, un particolare compromesso, tra coerenza e latenza, è stato presumibilmente più influente sulla progettazione DDBS rispetto ai compromessi CAP. Entrambi i set di compromessi sono importanti; unificare CAP e il compromesso coerenza/latenza in un'unica formulazione, PACELC, può di conseguenza portare a una comprensione più approfondita della moderna progettazione DDBS.

CAP È PER I FALLIMENTI

CAP afferma fondamentalmente che nella creazione di un DDBS, i progettisti possono scegliere due delle tre proprietà desiderabili: coerenza (C), disponibilità (A) e tolleranza alle partizioni (P). Pertanto, sono possibili solo sistemi CA (coerenti e altamente disponibili, ma non tolleranti alle partizioni), sistemi CP (coerenti e tolleranti alle partizioni, ma non altamente disponibili) e sistemi AP (altamente disponibili e tolleranti alle partizioni, ma non coerenti).

Molti DDBS moderni, tra cui SimpleDB/Dynamo, Cassandra, Voldemort, Sherpa/PNUTS e Riak, non

0018-9162/12/$31.00 © 2012 IEEE Pubblicato dalla IEEE Computer Society

per impostazione predefinita garantiscono la coerenza, come definito da CAP. (Sebbene la coerenza di alcuni di questi sistemi sia diventata regolabile dopo il rilascio delle versioni iniziali, l'attenzione qui è rivolta al loro design originale.) Nella loro dimostrazione di CAP, Seth Gilbert e Nancy Lynch7 hanno utilizzato la definizione di atomico/coerenza linearizzabile: "Deve esistere un ordine totale su tutte le operazioni in modo che ogni operazione sembri completata in un singolo istante. Ciò equivale a richiedere che le richieste della memoria condivisa distribuita agiscano come se fossero eseguite su un singolo nodo, rispondendo alle operazioni una alla volta".

Dato che le prime ricerche DDBS si concentravano su sistemi coerenti, è naturale supporre che CAP abbia avuto un'influenza importante sugli architetti di sistemi moderni, i quali, durante il periodo successivo alla dimostrazione del teorema, hanno costruito un numero crescente di

È sbagliato supporre che i DDBS che riducono la coerenza in assenza di partizioni lo facciano a causa del processo decisionale basato su CAP.

di sistemi che implementano modelli di consistenza ridotta.
Il ragionamento alla base di questa ipotesi è che, poiché qualsiasi DDBS deve essere tollerante alle partizioni di rete, secondo CAP, il sistema deve scegliere tra alta disponibilità e coerenza. Per le applicazioni mission-critical in cui l'alta disponibilità è estremamente importante, non ha altra scelta che sacrificare la coerenza.

Tuttavia, questa logica è imperfetta e non coerente con ciò che CAP afferma effettivamente. Non è semplicemente la tolleranza della partizione che richiede un compromesso tra coerenza e disponibilità; piuttosto, è la combinazione di

- tolleranza di partizione e
- l'esistenza stessa di una partizione di rete.

Il teorema afferma semplicemente che una partizione di rete fa sì che il sistema debba decidere tra la riduzione della disponibilità o della coerenza. La probabilità di una partizione di rete dipende fortemente dai vari dettagli dell'implementazione del sistema: è distribuita su una rete WAN (Wide Area Network) o solo su un cluster locale? Qual è la qualità dell'hardware? Quali processi sono in atto per garantire che le modifiche ai parametri di configurazione della rete vengano eseguite con attenzione? Qual è il livello di ridondanza?

Tuttavia, in generale, le partizioni di rete sono piuttosto rare e spesso meno frequenti di altri gravi tipi di eventi di guasto nei DDBS.8

Poiché CAP non impone alcuna restrizione di sistema nel caso di base, è sbagliato supporre che i DDBS che riducono la coerenza in assenza di partizioni lo facciano a causa del processo decisionale basato su CAP. Infatti, CAP consente il sistema per realizzare il set completo di garanzie ACID (atomicità, consistenza, isolamento e durata) insieme all'alta disponibilità quando non ci sono partizioni. Pertanto, il teorema non giustifica completamente la configurazione predefinita di DDBS che riduce la consistenza (e solitamente diverse altre garanzie ACID).

COMPROMESSO COERENZA/LATENZA

Per comprendere la progettazione moderna dei DDBS, è importante comprendere il contesto in cui questi sistemi sono stati costruiti. Amazon ha originariamente progettato Dynamo per fornire dati ai servizi principali nella sua piattaforma di e-commerce (ad esempio, il carrello della spesa). Facebook ha creato Cassandra per alimentare la sua funzione di ricerca Inbox. LinkedIn ha creato Voldemort per gestire gli aggiornamenti tenute da varie funzioni ad alta intensità di scrittura sul suo sito web. Yahoo ha creato PNUTS per archiviare dati utente che possono essere letti o scritti in ogni visualizzazione di pagina web, per archiviare dati di elenchi per le pagine di shopping di Yahoo e per archiviare dati per servire le sue applicazioni di social network. Casi d'uso simili al Riak motivato di Amazon.

In ogni caso, il sistema in genere fornisce dati per pagine web costruite al volo e inviate a un utente attivo del sito web, e riceve aggiornamenti online. Gli studi indicano che la latenza è un fattore critico nelle interazioni online: un aumento di appena 100 ms può ridurre drasticamente la probabilità che un cliente continui a interagire o torni in futuro.9

Sfortunatamente, c'è un compromesso fondamentale tra coerenza, disponibilità e latenza. (Si noti che disponibilità e latenza sono presumibilmente la stessa cosa: un sistema non disponibile fornisce essenzialmente una latenza estremamente elevata. Ai fini di questa discussione, considero i sistemi con latenze maggiori di un tipico timeout di richiesta, come pochi secondi, come non disponibili e latenze minori di un timeout di richiesta, ma comunque prossime a centinaia di millisecondi, come "latenza elevata". Tuttavia, alla fine abbandonerò questa distinzione e consentirò al requisito di bassa latenza di assorbire entrambi i casi.
Pertanto, il compromesso è in realtà solo tra coerenza e latenza, come suggerisce il titolo di questa sezione.)

Questo compromesso esiste anche quando non ci sono reti partizioni, e quindi è completamente separato dai compromessi descritti da CAP. Tuttavia, è un fattore critico nella progettazione dei sistemi sopra menzionati. (È irrilevante per questa discussione se un singolo guasto della macchina sia trattato o meno come un tipo speciale di partizione di rete.)

Il motivo di questo compromesso è che un requisito di elevata disponibilità implica che il sistema debba replicare i dati.
Se il sistema funziona abbastanza a lungo, almeno un componente del sistema alla fine fallirà. Quando si verifica questo guasto, tutti i dati controllati da quel componente diventeranno non disponibili a meno che il sistema non abbia replicato un'altra versione dei dati prima del guasto. Pertanto, la possibilità di guasto, anche in assenza del guasto stesso, implica che il requisito di disponibilità richieda un certo grado di

replicazione dei dati durante il normale funzionamento del sistema. (Si noti l'importante differenza tra questo compromesso e i compromessi CAP: mentre il *verificarsi* di un errore provoca i compromessi CAP, la *possibilità* stessa di errore determina questo compromesso.)

Per raggiungere i massimi livelli possibili di disponibilità, un DDBS deve replicare i dati su una WAN per proteggersi dal guasto di un intero data center dovuto, ad esempio, a un uragano, un attacco terroristico o, come nel famoso blackout del cloud Amazon EC2 dell'aprile 2011, a un singolo errore di configurazione di rete. I cinque sistemi a consistenza ridotta menzionati sopra sono progettati per una disponibilità estremamente elevata e solitamente per la replica su una WAN.

REPLICA DEI DATI

Non appena un DDBS replica i dati, si verifica un compromesso tra coerenza e latenza. Ciò si verifica perché ci sono solo tre alternative per implementare la replicazione dei dati: il sistema invia gli aggiornamenti dei dati a tutte le repliche contemporaneamente, prima a un nodo master concordato o prima a un singolo nodo (arbitrario). Il sistema può implementare ciascuno di questi casi in vari modi; tuttavia, ogni implementazione comporta un compromesso tra coerenza e latenza.

(1) Gli aggiornamenti dei dati vengono inviati a tutte le repliche contemporaneamente

Se gli aggiornamenti non passano prima attraverso un livello di pre-elaborazione o un altro protocollo di accordo, potrebbe verificarsi una divergenza di replica, ovvero una chiara mancanza di coerenza (ipotizzando che vengano inviati contemporaneamente più aggiornamenti al sistema, ad esempio da client diversi), poiché ogni replica potrebbe scegliere un ordine diverso in cui applicare gli aggiornamenti.
(Anche se tutti gli aggiornamenti sono commutativi, in modo tale che ogni rep-lica alla fine diventi coerente, nonostante il fatto che le repliche possano eventualmente applicare gli aggiornamenti in ordini diversi, la definizione rigorosa di consis-tency7 di Gilbert e Lynch non regge ancora. Tuttavia, Paxos10 generalizzato può fornire una replica coerente in un singolo round-trip.)

D'altro canto, se gli aggiornamenti passano prima attraverso un livello di pre-elaborazione o tutti i nodi coinvolti nella scrittura utilizzano un protocollo di accordo per decidere l'ordine delle operazioni, allora è possibile garantire che tutte le repliche concorderanno sull'ordine in cui elaborare gli aggiornamenti. Tuttavia, ciò porta a diverse fonti di latenza aumentata. Nel caso del protocollo di accordo, il protocollo stesso è la fonte aggiuntiva di latenza.

Nel caso del preprocessore, ci sono due fonti di latenza. Innanzitutto, il routing degli aggiornamenti tramite un componente di sistema aggiuntivo (il preprocessore) aumenta la latenza.
In secondo luogo, il preprocessore è costituito da più macchine o da una singola macchina. Nel primo caso, è necessario un protocollo di accordo per decidere l'ordinamento delle operazioni tra le macchine. Nel secondo caso, il sistema forza tutti gli aggiornamenti, indipendentemente da dove siano avviati, potenzialmente in qualsiasi parte del mondo, a essere instradati fino alla singola macchina.

prima il preprocessore, anche se un'altra replica dei dati è più vicina alla posizione di avvio dell'aggiornamento.

(2) Gli aggiornamenti dei dati vengono inviati prima a una posizione concordata

Farò riferimento a questa posizione concordata come "nodo master" (elementi di dati diversi possono avere nodi master diversi). Questo nodo master risolve tutte le richieste di aggiornamento dell'elemento di dati e l'ordine in cui sceglie di eseguire questi aggiornamenti determina l'ordine in cui tutte le repliche eseguono gli aggiornamenti. Dopo che il nodo master risolve gli aggiornamenti, li replica in tutte le posizioni delle repliche.

Non appena un DDBS replica i dati, si verifica un compromesso tra coerenza e latenza.

Esistono tre opzioni di replicazione:

a. La replica è sincrona: il nodo master attende che tutti gli aggiornamenti siano stati inviati alla/e replica/e.
   Ciò garantisce che le repliche rimangano coerenti, ma le azioni sincrone tra entità indipendenti, in particolare su una WAN, aumentano la latenza a causa della necessità di passare messaggi tra queste entità e del fatto che la latenza è limitata dall'entità più lenta.
b. La replica è asincrona: il sistema tratta l'aggiornamento come se fosse stato completato prima di essere replicato. In genere, l'aggiornamento è almeno arrivato in un archivio stabile da qualche parte prima che l'iniziatore dell'aggiornamento venga a conoscenza del suo completamento (nel caso in cui il nodo master fallisca), ma non vi è alcuna garanzia che il sistema abbia propagato l'aggiornamento. Il compromesso coerenza/latenza in questo caso dipende da come il sistema gestisce le letture:

   i. Se il sistema indirizza tutte le letture al nodo master e le serve da lì, non vi è alcuna riduzione.

   zione in coerenza. Tuttavia, ci sono due problemi di latenza con questo approccio:

   1. Anche se c'è una replica vicina al letto-richiedente, il sistema deve comunque instradare la richiesta al nodo master, che po-potenzialmente potrebbe essere fisicamente molto più lontano.
   2. Se il nodo master è sovraccarico di altre richieste o è fallito, non c'è alcuna opzione per servire la lettura da un nodo diverso. Piuttosto, la richiesta deve attendere che il nodo master diventi libero o si riprenda. In altre parole, la mancanza di opzioni di bilanciamento del carico aumenta il potenziale di latenza.

ii. Se il sistema può servire letture da qualsiasi nodo, la latenza di lettura è molto migliore, ma ciò può anche comportare letture incoerenti dello stesso elemento dati, poiché posizioni diverse hanno versioni diverse di un elemento dati mentre il sistema è ancora in fase di propagazione.
aggiornamenti, e potrebbe inviare una lettura a una qualsiasi di queste posizioni. Sebbene il livello di coerenza ridotta possa essere limitato tenendo traccia dei numeri di sequenza degli aggiornamenti e utilizzandoli per implementare la coerenza sequenziale/temporale o la coerenza read-your-writes, queste non sono opzioni di coerenza ridotte. Inoltre-
Inoltre, la latenza di scrittura può essere elevata se il nodo master per un'operazione di scrittura è geograficamente distante dal richiedente la scrittura.

c. È possibile una combinazione di (a) e (b): il sistema invia aggiornamenti a un sottoinsieme di repliche in modo sincrono e al resto in modo asincrono. La coerenza/

Per la replicazione dei dati su una WAN, non esiste modo di aggirare il compromesso tra coerenza e latenza.

il compromesso di latenza in questo caso è nuovamente determinato dal modo in cui il sistema gestisce le letture:

i. Se indirizza le letture ad almeno un nodo che è stato aggiornato in modo sincrono, ad esempio quando $R + W > N$ in un protocollo di quorum, dove R è il numero di nodi coinvolti in una sincronizzazione-nous read, W è il numero di nodi coinvolti in una scrittura sincrona e N è il numero di repliche, quindi la coerenza può essere preservata.
Tuttavia, i problemi di latenza di (a), (b)(i)(1) e (b)(i)(2) sono tutti presenti, sebbene in misura leggermente minore, poiché il numero di nodi coinvolti nella sincronizzazione è inferiore e più di un nodo può potenzialmente servire richieste di lettura. ii. Se è possibile per il sistema servire letture da nodi che non sono stati aggiornati in modo sincrono, ad esempio, quando $R + W \not> N$, allora sono possibili letture incoerenti, come in (b)(ii).

Tecnicamente, il semplice utilizzo di un protocollo di quorum non è sufficiente a garantire la coerenza al livello definito da Gilbert e Lynch. Tuttavia, le aggiunte di protocollo necessarie per garantire la coerenza completa11 non sono rilevanti qui. Anche senza queste aggiunte, la latenza è già insita nel protocollo di quorum.

(3) Gli aggiornamenti dei dati vengono inviati prima a una posizione arbitraria

Il sistema esegue gli aggiornamenti in quella posizione e poi li propaga alle altre repliche. La differenza è-
tra questo caso e (2) è che la posizione in cui il sistema invia gli aggiornamenti per un particolare elemento di dati non è sempre la stessa. Ad esempio, due diversi aggiornamenti per un particolare elemento di dati possono essere avviati simultaneamente in due diverse posizioni.

Il compromesso tra coerenza e latenza dipende ancora una volta da due opzioni:

a. Se la replica è sincrona, allora sono presenti i problemi di latenza di (2)(a). Inoltre, il sistema può incorrere in una latenza extra per rilevare e risolvere i casi di aggiornamenti simultanei allo stesso elemento dati avviati in due posizioni diverse.

b. Se la replica è asincrona, allora sono presenti problemi di coerenza simili a (1) e (2)(b).

ESEMPI DI COMPROMESSI

Indipendentemente da come un DDBS replica i dati, è chiaro che deve bilanciare coerenza e latenza. Per una replica attentamente controllata su brevi distanze, esistono opzioni ragionevoli come (2)(a) perché la latenza della comunicazione di rete è ridotta nei data center locali; tuttavia, per la replica su una WAN, non c'è modo di aggirare il compromesso coerenza/latenza.

Per comprendere più a fondo il compromesso, è utile considerare come quattro DDBS progettati per un'elevata disponibilità (Dynamo, Cassandra, PNUTS e Riak)
replicare i dati. Poiché questi sistemi sono stati progettati per interazioni a bassa latenza con client Web attivi, ognuno sacrifica la coerenza per una latenza migliorata.

Dynamo, Cassandra e Riak utilizzano una combinazione di (2)(c) e (3). In particolare, il sistema invia aggiornamenti allo stesso nodo e quindi li propaga in modo sincrono ad altri W nodi, ovvero il caso (2)(c). Il sistema invia letture in modo sincrono a R nodi, con $R + W$ in genere impostato su un numero inferiore o uguale a N, il che comporta la possibilità di letture incoerenti. Tuttavia, il sistema non invia sempre aggiornamenti allo stesso nodo per un particolare elemento dati, ad esempio, ciò può accadere in vari casi di errore o a causa del reindirizzamento da parte di un bilanciatore di carico. Ciò porta alla situazione descritta in (3) e potenzialmente a carenze di coerenza più sostanziali. PNUTS utilizza l'opzione (2)(b)(ii), ottenendo un'eccellente latenza a consistenza ridotta.

Uno studio recente di Jun Rao, Eugene Shekita e Sandeep Tata12 fornisce ulteriori prove del compromesso tra coerenza/latenza nell'implementazione di base di questi sistemi. I ricercatori hanno valutato sperimentalmente due opzioni nel compromesso tra coerenza/latenza di Cas-sandra. La prima opzione, "letture deboli", consente al sistema di gestire le letture da qualsiasi replica, anche se tale replica non ha ricevuto tutti gli aggiornamenti in sospeso per un elemento dati. La seconda opzione, "letture del quorum", richiede al sistema di verificare esplicitamente l'incoerenza tra più repliche prima di leggere i dati.

la seconda opzione aumenta chiaramente la coerenza al costo di una latenza aggiuntiva rispetto alla prima opzione. La differenza di latenza tra queste due opzioni può essere un fattore di quattro o più.

Un altro studio di Hiroshi Wada e colleghi13 sembra contraddire questo risultato. Questi ricercatori hanno scoperto che richiedere una lettura coerente in SimpleDB non aumenta significativamente la latenza rispetto all'opzione di lettura predefinita (potenzialmente incoerente). Tuttavia, i ricercatori hanno eseguito questi esperimenti in una singola regione di Amazon (Stati Uniti occidentali) e ipotizzano che SimpleDB utilizzi la replica master-slave, che è possibile implementare con un modesto costo di latenza se la replica avviene su una breve distanza. In particolare, Wada e colleghi hanno concluso che SimpleDB forza tutte le letture coerenti ad andare al master incaricato di scrivere gli stessi dati. Finché la richiesta di lettura proviene da una posizione fisicamente vicina al master e finché il master non è sovraccarico, la latenza aggiuntiva della lettura coerente non è visibile (entrambe queste condizioni erano vere nei loro esperimenti).

Se SimpleDB avesse replicato i dati tra le regioni di Amazon e la richiesta di lettura provenisse da una regione diversa dalla posizione del master, il costo di latenza della lettura coerente sarebbe stato più evidente. Anche senza la replica tra le regioni (SimpleDB al momento non supporta la replica tra le regioni), la documentazione ufficiale di Amazon avvisa gli utenti di una latenza aumentata e di una produttività ridotta per le letture coerenti.

Tutti e quattro i DDBS consentono agli utenti di modificare i parametri predefiniti per scambiare una maggiore coerenza con una latenza peggiore, ad esempio rendendo *R* + *W* più di *N* nei sistemi di tipo quorum. Tuttavia, il compromesso coerenza/latenza si verifica durante il normale funzionamento del sistema, anche in assenza di partizioni di rete. Questo compromesso è amplificato se c'è una replicazione dei dati su una WAN. La conclusione ovvia è che la ridotta coerenza è attribuibile alla latenza di runtime, non a CAP.

PNUTS offre la prova più chiara che la CAP non è un motivo principale per i livelli di coerenza ridotti in questi sistemi. In PNUTS, un nodo master possiede ogni elemento dati. Il sistema indirizza gli aggiornamenti a quell'elemento al nodo master, quindi propaga questi aggiornamenti in modo asincrono a repliche su una WAN. PNUTS può servire letture da qualsiasi replica, il che colloca il sistema nella categoria (2)(b)(ii): riduce la coerenza per ottenere una migliore latenza. Tuttavia, nel caso di una partizione di rete, in cui il nodo master diventa non disponibile all'interno di una partizione di minoranza, il sistema per impostazione predefinita rende l'elemento dati non disponibile per gli aggiornamenti. In altre parole, la configurazione predefinita di PNUTS è in realtà CP: nel caso di una partizione, il sistema sceglie la coerenza rispetto alla disponibilità per evitare il problema di risolvere gli aggiornamenti in conflitto avviati su diversi nodi master.

Pertanto, la scelta di ridurre la coerenza nel caso di base è più ovviamente attribuibile alla continua compromesso coerenza/latenza rispetto al compromesso coerenza/disponibilità in CAP che si verifica solo su una partizione di rete. Naturalmente, la mancanza di coerenza di PNUTS nel caso di base è utile anche nel caso di partizione di rete, poiché i dati masterizzati in una partizione non disponibile sono comunque accessibili per le letture.

Si può sostenere che CAP abbia maggiore influenza sugli altri tre sistemi. Dynamo, Cassandra e Riak passano più completamente all'opzione di replica dei dati (3) in caso di partizione di rete e gestiscono i problemi di coerenza risultanti utilizzando uno speciale codice di riconciliazione che viene eseguito al rilevamento della divergenza della replica. È quindi ragionevole supporre che questi sistemi siano stati progettati tenendo presente la possibilità di una partizione di rete. Poiché si tratta di sistemi AP, il codice di riconciliazione e la capacità di passare a (3) sono state integrate nel codice fin dall'inizio. Tuttavia, una volta che quel codice era lì, è conveniente riutilizzare parte di quella flessibilità di coerenza per scegliere un punto anche nel compromesso di base coerenza/latenza. Questa argomentazione è più logica delle affermazioni secondo cui i progettisti di questi sistemi hanno scelto di ridurre interamente la coerenza a causa di CAP (ignorando il fattore latenza).

Ignorare il compromesso tra coerenza e latenza dei sistemi replicati è una grave svista, poiché è presente in ogni momento durante il funzionamento del sistema.

In conclusione, CAP è solo una delle due principali ragioni per cui i moderni DDBS riducono la coerenza. Ignorare il compromesso coerenza/latenza dei sistemi replicati è una grave svista, poiché è presente in ogni momento durante il funzionamento del sistema, mentre CAP è rilevante solo nel caso, presumibilmente raro, di una partizione di rete. In effetti, il primo compromesso potrebbe essere più influente perché ha un effetto più diretto sulla operazioni di base dei sistemi.

PACELC

Una rappresentazione più completa dello spazio dei potenziali compromessi di coerenza per i DDBS può essere ottenuta riscrivendo CAP come PACELC (pronunciato "pass-elk"): se c'è una partizione (P), in che modo il sistema compensa disponibilità e coerenza (A e C); altrimenti (E), quando il sistema funziona normalmente in assenza di partizioni, in che modo il sistema compensa latenza (L) e coerenza (C)?

Si noti che il compromesso latenza/coerenza (ELC) si applica solo ai sistemi che replicano i dati. Altrimenti, il sistema soffre di problemi di disponibilità in caso di qualsiasi tipo di guasto o nodo sovraccarico. Poiché tali problemi sono solo istanze di latenza estrema, la parte di latenza del compromesso ELC può incorporare la scelta se replicare o meno i dati.

Le versioni predefinite di Dynamo, Cassandra e Riak sono sistemi PA/EL: se si verifica una partizione, rinunciano alla coerenza per la disponibilità e, in condizioni di normale funzionamento, rinunciano alla coerenza per una latenza inferiore. Rinunciare a entrambe le C in PACELC semplifica la progettazione; una volta che un sistema è configurato per gestire le incongruenze, ha senso rinunciare alla coerenza sia per la disponibilità che per una latenza inferiore. Tuttavia, questi sistemi hanno impostazioni regolabili dall'utente per modificare il compromesso ELC: ad esempio, aumentando *R* + W, ottengono maggiore coerenza a scapito della latenza (sebbene non possano raggiungere la piena coerenza come definito da Gilbert e Lynch, anche se *R* + *W* > N).

I sistemi completamente ACID come VoltDB/H-Store e Megastore sono PC/EC: si rifiutano di rinunciare alla coerenza e sono disposti a pagare i costi di disponibilità e latenza per ottenerla. Anche BigTable e i sistemi correlati come HBase sono PC/EC.

MongoDB può essere classificato come un sistema PA/EC. Nel caso di base, il sistema garantisce che le letture e le scritture siano coerenti. Tuttavia, MongoDB utilizza l'opzione di replica dei dati (2) e, se il nodo master fallisce o è partizionato dal resto del sistema, memorizza tutte le scritture che sono state inviate al nodo master ma non ancora replicate in una directory di rollback locale. Nel frattempo, il resto del sistema elegge un nuovo master che rimanga disponibile per letture e scritture. Pertanto, lo stato del vecchio master e lo stato del nuovo master diventano incoerenti finché il sistema non ripara il guasto e utilizza la directory di rollback per riconciliare gli stati, che oggi è un processo manuale.

(Tecnicamente, quando si verifica una partizione, MongoDB non è disponibile secondo la definizione di disponibilità CAP, poiché la partizione minoritaria non è disponibile. Tuttavia, nel contesto di PACELC, poiché una partizione causa più problemi di coerenza che di disponibilità, MongoDB può essere classificato come un sistema PA/EC.)

PNUTS è un sistema PC/EL. In condizioni operative normali, rinuncia alla coerenza in favore della latenza; tuttavia, se si verifica una partizione, baratta la disponibilità con la coerenza. Questo è certamente un po' confusionario: secondo PACELC, PNUTS sembra diventare più coerente su una partizione di rete. Tuttavia, PC/EL non dovrebbe essere interpretato in questo modo. PC non indica che il sistema è completamente coerente; piuttosto indica che il sistema non riduce la coerenza oltre il livello di coerenza di base quando si verifica una partizione di rete, ma riduce la disponibilità.

i sistemi di base sono complessi e né il CAP né
I compromessi implicati nella creazione di dati distribuiti
PACELC può spiegarli tutti. Tuttavia, incorporare il compromesso coerenza/latenza nelle considerazioni di progettazione DDBS moderne è abbastanza importante da giustificare di portare il compromesso più in primo piano nelle discussioni architettoniche.

C

Ringraziamenti

Questo articolo è stato sottoposto a numerose iterazioni e modifiche grazie al feedback estremamente utile e dettagliato di Samuel Madden, Andy Pavlo, Evan Jones, Adam Marcus, Daniel Weinreb e tre revisori anonimi.

Riferimenti

1. G. DeCandia et al., “Dynamo: Amazon's Highly Available Key-Value Store,” *Proc. 21st ACM SIGOPS Symp. Principi dei sistemi operativi* (SOSP 07), ACM, 2007, pp. 205-220.
2. A. Lakshman e P. Malik, "Cassandra: sistema di archiviazione strutturato su una rete P2P", *Proc. 28th ACM Symp. Principi di elaborazione distribuita* (PODC 09), ACM, 2009, articolo n. 5; doi:10.1145/1582716.1582722.
3. BF Cooper et al., “PNUTS: piattaforma di distribuzione dati ospitata di Yahoo!,” *Proc. VLDB Endowment* (VLDB 08), ACM, 2008, pp. 1277-1288.
4. F. Chang et al., "Bigtable: un sistema di archiviazione distribuito per dati strutturati", *Proc. 7th Usenix Symp. Progettazione e implementazione di sistemi operativi* (OSDI 06), Usenix, 2006, pp. 205-218.
5. M. Stonebraker et al., "La fine di un'era architettonica (è tempo per una riscrittura completa)", *Proc. VLDB Endowment* (VLDB 07), ACM, 2007, pp. 1150-1160.
6. J. Baker et al., “Megastore: fornire storage scalabile e altamente disponibile per servizi interattivi”, *Atti 5th Biennial Conf. Innovative Data Systems Research* (CIDR 11), ACM, 2011, pp. 223-234.
7. S. Gilbert e N. Lynch, "La congettura di Brewer e la fattibilità di servizi Web coerenti, disponibili e tolleranti alle partizioni", *ACM SIGACT News,* giugno 2002, pp. 51-59.
8. M. Stonebraker, “Errori nei sistemi di database, coerenza eventuale e il teorema CAP”, blog, *Comm. ACM,* 5 aprile 2010; http://cacm.acm.org/blogs/blog-cacm/83396-errori-nei-sistemi-di-database-coerenza-eventuale-e-teorema-cap.
9. J. Brutlag, “Speed Matters for Google Web Search,” articolo non pubblicato, 22 giugno 2009, Google; http://code.google.com/velocità/files/delayexp.pdf.
10. L. Lamport, *Generalized Consensus e Paxos,* rapporto tecnico MSR-TR-2005-33, Microsoft Research, 2005; ftp://ftp.Research.microsoft.com/pub/tr/TR-2005-33.pdf.
11. H. Attiya, A. Bar-Noy e D. Dolev, “Condivisione robusta della memoria nei sistemi di passaggio di messaggi”, *JACM,* gennaio 1995, pp. 124-142.
12. J. Rao, EJ Shekita e S. Tata, "Utilizzo di Paxos per creare un datastore scalabile, coerente e altamente disponibile", *Proc. Dotazione VLDB* (VLDB 11), ACM, 2011, pp. 243-254.
13. H. Wada et al., "Proprietà di coerenza dei dati e compromessi nell'archiviazione cloud commerciale: la prospettiva dei consumatori", *Atti 5a conferenza biennale. Ricerca sui sistemi di dati innovativi* (CIDR 11), ACM, 2011, pp. 134-143.

***Daniel J. Abadi** è professore associato presso il Dipartimento di Informatica della Yale University. I suoi interessi di ricerca includono l'architettura del sistema di database, il cloud computing e i sistemi scalabili. Abadi ha conseguito un dottorato di ricerca in informatica presso il Massachusetts Institute of Technology. Contattatelo all'indirizzo dna@cs.yale.edu; il suo blog è http://dbmsmusings.blogspot.com e twitta su @daniel_abadi.*