Teorema della PAC di Brewer

Il kool aiuta Amazon ed Ebay hanno bevuto

11 gennaio 2009 • 18 minuti di lettura

Etichette: L'architettura - Il business - La strategia

Conosciamo tre accordi, ma puoi sceglierti solo due

Venerdì 4 giugno 1976, in una piccola sala al piano superiore lontano dall'auditorium principale, i Sex Pistols (http://en.wikipedia.org/wiki/Sex_Pistols) hanno dato il via al loro primo concerto alla Lesser Free Trade Hall (http://en.wikipedia.org/wiki/Free_Trade_Hall) di Manchester. C'è un po' di confusione su chi fosse esattamente lì tra il pubblico quella notte, in parte perché c'era un altro concerto solo sei settimane dopo, ma soprattutto perché è considerato un concerto che ha cambiato (http://www.bbc.co.uk/print/manchester/content/articles/2006/05/11/110506_sex_pistols_gig_feature.shtml) per sempre la cultura della musica occidentale (http://www.bbc.co.uk/print/manchester/content/articles/2006/05/11/110506_sex_pistols_gig_feature.shtml). Così iconico e importante è diventato quell'aspetto che David Nolan ha scritto un libro, I Swear I Was There: The Gig That Changed the World (http://www.amazon.co.uk/gp/product/0954970497?ie=UTF8&tag=julibrow-21&linkCode=as2&camp=1634&creative=19450&creativeASIN=0954970497), indagando su chi fosse la pretesa di essere presente era giustificata. Perché il 4 giugno è generalmente considerato la genesi del punk rock [1]1.

Prima di questo (in realtà dal 1971 circa) c'erano stati un certo numero di band protopunk (http://en.wikipedia.org/wiki/Protopunk), come i New York Dolls (http://www.punk77.co.uk/punkhistory/newyorkdolls.htm) e i Velvet Underground (http://en.wikipedia.org/wiki/The_Velvet_Underground), ma è stato questo setto dai Sex Pistols che nel folklore musicale ha iniziato la rivoluzione che ha messo in moto le chitarre trainanti dei Buzzcocks (http://www.buzzcocks.com/), la lamento lamento lamento lamento lamento lamento eclettico degli Smiths (http://en.wikipedia.org/wiki/The_Smiths), le eclettiche sincopazioni della Falla (http://www.dcs.ed.ac.uk/home/cxl/fall/index.html), la maestà

emergente di Joy Division (http://en.wikipedia.org/wiki/Joy_Division) (I Joy Division (http://en.wikipedia.org/wiki/Joy_Division)).

Mercoledì 19 luglio 2000, potrebbe non scendere nella cultura popolare con la stessa portata, ma ha avuto un impatto simile sul business su scala Internet come i Sex Pistols hanno fatto sulla musica un quarto di secolo prima, perché quello era il discorso di (http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf) apertura (http://www.cs.berkeley.edu/~brewer/) di Eric Brewer (http://www.cs.berkeley.edu/~brewer/) al Simposio ACM sui principi del Distributed Computing (http://www.podc.org/podc2000/) (PODC).

I Sex Pistols avevano dimostrato che la furia a malapena vincolata era più importante per i loro contemporanei che per lo strutturalismo della scuola d'arte, dando a chiunque con tre accordi e qualcosa per dire il permesso di avviare una band. Eric Brewer, in quella che è diventata nota come Congettura di Brewer, ha affermato che man mano che le applicazioni diventano più basate sul web dovremmo smettere di preoccuparci della coerenza dei dati, perché se vogliamo un'elevata disponibilità in queste nuove applicazioni distribuite, allora la coerenza garantita dei dati è qualcosa che *non* possiamo *avere*, dando così a chiunque con tre server e un occhio attento all'esperienza del cliente il permesso di avviare un'attività in scala Internet. I discepoli di Brewer (presente quel giorno o più convertito) includono artisti del calibro di Amazon (http://www.infoq.com/news/2008/01/consistency-vs-availability), eBay (http://www.infoq.com/articles/ebay-scalability-best-practices) e Twitter (http://highscalability.com/scaling-twitter-making-twitter-10000-percent-faster).

Due anni dopo, nel 2002, Seth Gilbert (http://lpd.epfl.ch/sgilbert/) e Nancy Lynch (http://people.csail.mit.edu/lynch/) del MIT, dimostrarono formalmente (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.20.1495&rep=rep1&type=pdf) Brewer di essere corretto e così nacque il Teorema di Brewer.

Teorema di Brewer (CAP)

Quindi cos'è esattamente il Theorem di Brewer, e perché merita il confronto con un concerto punk del 1976 a Manchester?

Il discorso di Brewer del 2000 si basava sul suo lavoro teorico alla UC Berkley e alle osservazioni di Esing Inktomi (http://en.wikipedia.org/wiki/Inktomi), anche se Brewer e altri stavano parlando di decisioni di compromesso che devono essere prese in sistemi altamente scalabili anni prima (ad es. " Cluster-Based Scalable Network Services" (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1.2034&rep=rep1&type=pdf) da SOSP nel 1997 e "Rilascio, resa e sistemi tolleranti (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.24.3690&rep=rep1&type=pdf) scalabili" nel 1999) quindi i contenuti della presentazione non erano nuovi e, come molte di queste idee, erano

opera di molte persone intelligenti (come sono sicuro che Brewer stesso sarebbe stato veloce a sottolineare).

Quello che ha detto era che ci sono tre requisiti sistemici fondamentali che esistono in una relazione speciale quando si tratta di progettare e distribuire applicazioni in un ambiente distribuito (ma si parlava specificamente del web, ma così tante aziende aziendali sono multi-sito / multi-paese in questi giorni che gli effetti potrebbero ugualmente applicarsi al tuo accordo data-center / LAN / WAN).

I tre requisiti sono: ***Consistenza***, ***disponibilità*** e ***tolleranza alla partizione***, dando al teorema di Brewer il suo nome - **CAP**.

Per dare a questi alcuni significati del mondo reale, usiamo un semplice esempio: vuoi comprare una copia di Tolstoy (http://en.wikipedia.org/wiki/Leo_Tolstoy)Tolstoj's War & Peace (http://en.wikipedia.org/wiki/War_and_Peace) per leggere una vacanza particolarmente lunga che inizierai domani. La tua libreria web preferita ha una copia rimasta in magazzino. Fai la tua ricerca, controlla che possa essere consegnato prima di partire e aggiungilo al tuo carrello. Ti ricordi che hai bisogno di alcune altre cose per sfogliare il sito per un po '(hai mai comprato *una* cosa online? Devo massimizzare il dollaro del pacco). Mentre stai leggendo le recensioni dei clienti di un prodotto di lozione abbronzante, qualcuno, da qualche altra parte nel paese, arriva al sito, aggiunge una copia al suo carrello e va direttamente al processo di checkout (hanno bisogno di una soluzione urgente per un tavolo traballante con una gamba *molto* più corta delle altre).

- **La coerenza**

  Un servizio che è *coerente* funziona pienamente o non funziona affatto. Gilbert e Lynch usano la parola "atomico" invece che coerente nella loro prova, il che ha più senso tecnicamente perché, in senso stretto, coerente è il C in ACID (http://en.wikipedia.org/wiki/ACID) applicato alle proprietà ideali delle transazioni di database e significa che i dati non saranno mai persistenti che rompe determinati vincoli preimpostati. Ma se si considera un vincolo preimpostato di sistemi distribuiti che più valori per lo stesso pezzo di dati non sono consentiti, penso che la perdita nell'astrazione sia tappata (più, se Brewer avesse usato la parola atomico, si chiamerebbe il teorema AAP e saremmo tutti in ospedale ogni volta che abbiamo cercato di pronunciarlo).

  Nell'esempio di acquisto del libro puoi aggiungere il libro al tuo carrello o fallire. Acquistalo o no. Non puoi aggiungere a metà o a metà di acquistare un libro. C'è una copia in magazzino e solo una persona lo otterrà il giorno successivo. Se entrambi i clienti possono continuare attraverso il processo di ordinazione fino alla fine (cioè effettuare il pagamento) la mancanza di coerenza tra ciò che è in magazzino e ciò che è nel sistema causerà un problema. Forse non è un problema enorme in questo caso - qualcuno si annoierà in vacanza o versando la zuppa - ma scala questo a migliaia di incongruenze e dà loro un valore monetario (ad esempio, le negoziazioni su uno scambio finanziario in cui c'è un'incoerenza tra ciò che pensi di aver comprato o venduto e quali stati record di scambio) ed è un problema enorme.

  Potremmo risolvere la coerenza utilizzando un database. Nel momento giusto nel processo di ordine del libro il numero di libri in magazzino per la guerra e la pace è decrementato da uno. Quando l'altro cliente raggiunge questo punto, l'armadio è nudo e il processo di ordinazione li avviserà senza *without*continuare a pagare. Il primo funziona completamente, il secondo per niente.

  I database sono ottimi in questo perché si concentrano sulle proprietà ACID e ci danno coerenza anche dandoci isolamento, in modo che quando il cliente One sta riducendo i libri in magazzino di uno, e contemporaneamente aumentando i libri in cestino di uno, tutti gli stati intermedi sono isolati dal Cliente Due, che deve aspettare pochi millisecondi mentre il data store è reso coerente.

- **Disponibilità di**

  Disponibilità significa proprio questo - il servizio è disponibile (per operare completamente o meno come sopra). Quando acquisti il libro vuoi ricevere una risposta, non un messaggio del browser sul sito web che non è comunicativo. Gilbert & Lynch nella loro dimostrazione di Teorema CAP fanno il buon punto che la disponibilità più spesso ti abbandona quando ne hai più bisogno - i siti tendono a scendere nei periodi di punta proprio perché sono occupati. Un servizio disponibile ma non accessibile è di alcun beneficio per nessuno.

- **Tolleranza di partizione**

  Se l'applicazione e il database vengono girati su una casella, quindi (ignorando i problemi di scala e assumendo che tutto il codice sia perfetto) il tuo server agisce come una sorta di processore atomico in quanto funziona o non funziona (cioè se si è schiantato non è disponibile, ma non causerà nemmeno l'incoerenza dei dati).

  Una volta che si inizia a diffondere dati e logica intorno a diversi nodi, c'è il rischio di formazione di partizioni. Una partizione si verifica quando, ad esempio, un cavo di rete viene tagliato e il nodo A non può più comunicare con il nodo B. Con il tipo di capacità di distribuzione che il web fornisce, le partizioni temporanee sono un evento relativamente comune e, come ho detto prima, non sono nemmeno così rare all'interno delle aziende globali con più data center.

  Gilbert & Lynch ha definito la tolleranza della partizione come:

  > *Nessun set di guasti è consentito meno di un errore di rete totale che induca il sistema a rispondere in modo errato*

  Il commento di Brewer e notato che una partizione a un nodo è equivalente a un crash del server, perché se nulla può connettersi ad esso, potrebbe anche non essere lì.

Il significato del teorema

Il teorema della PAC prende vita come una scala di applicazione. A bassi volumi transazionali, le piccole latenze per consentire ai database di ottenere una coerenza non hanno alcun effetto evidente sulle prestazioni complessive o sull'esperienza dell'utente. Qualsiasi distribuzione di carico che si intraprende, quindi è probabile che sia per motivi di gestione dei sistemi.

Ma con l'aumentare dell'attività, questi pizzico-punti nel throughput inizieranno a limitare la crescita e a creare errori. Una cosa è dover aspettare che una pagina web torni con una

risposta e un'altra esperienza per inserire i dettagli della tua carta di credito per essere soddisfatta con "HTTP 500 java.lang.schrodinger.purchasingerror" e si chiede se hai appena pagato per qualcosa che non otterrai, non pagato affatto, o forse l'errore è irrilevante per questa transazione. - Chi lo sa? È improbabile che continui, più probabile che acquisti altrove e molto probabilmente telefoni la tua banca.

In ogni caso questo non è un bene per gli affari. Amazon afferma (http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it) che solo un decimo in più di un secondo sui tempi di risposta costerà loro l'1% nelle vendite. Google ha detto (http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html) di aver notato che solo mezzo secondo aumento della latenza ha causato un calo del traffico di un quinto.

Ho già scritto un po' di scalabilitàbefore, quindi non ripeterò tutto questo qui se non per fare due punti: il primo è che mentre affronti i problemi di scala potrebbe essere una preoccupazione architettonica, le discussioni iniziali non lo sono. Sono delle decisioni di business. Mi stanco molto di sentire, dai tecnici, che un tale approccio non è garantito perché i volumi di attività corrente non lo giustificano. Non è che abbiano torto; il più delle volte sono abbastanza corrette, è che limitare la scala fin dall'inizio è quello di prendere implicitamente decisioni sulle entrate - un fattore che dovrebbe essere reso esplicito durante l'analisi aziendale.

Il secondo punto è che una volta che ti imbarcherai in discussioni su *come* scalare al meglio la tua applicazione, il mondo cade ampiamente in due campi ideologici: la folla del database e la folla non-database.

La folla del database, non sorprende, come la tecnologia di database e tenderà ad affrontare la scala parlando di cose come il blocco ottimistico (http://en.wikipedia.org/wiki/Optimistic_concurrency_control) e lo sharding (http://en.wikipedia.org/wiki/Shard_\(database_architecture\)), mantenendo il database al centro delle cose.

La folla non-database tenderà ad affrontare la scala gestendo i dati al di fuori dell'ambiente di database (evitando il mondo relazionale) il più a lungo possibile.

Penso che sia giusto dire che l'ex gruppo non è stato portato al Teorema con lo stesso gusto del secondo (anche se ne stanno parlando (http://sqlblog.com/blogs/linchi_shea/archive/2008/12/17/the-cap-theorem.aspx)). Questo perché se devi perdere una di consistenza, disponibilità o tolleranza alla partizione, molti scelgono di rilasciare la coerenza che è la ragion d'essere del database. La logica. senza dubbio, è che la disponibilità e la tolleranza-partizione mantengono viva la tua applicazione per fare soldi, mentre l'incoerenza si sente come una di quelle cose che puoi lavorare con un design intelligente.

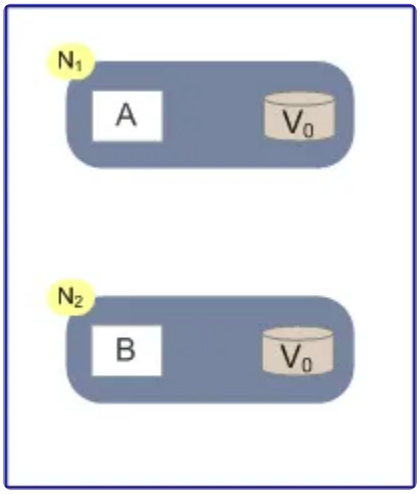
Come molto altro in IT, non è così in bianco e nero come questo. Eric Brewer, nella diapositiva 13 del suo discorso PODC, quando si confronta ACID ed è la controparte informale BASE (http://www.infoq.com/articles/pritchett-latency) dice anche "Penso che sia uno spettro". E se sei interessato a questo come argomento (è leggermente al di fuori di quello di cui voglio parlare qui) potresti fare peggio che iniziare con un articolo intitolato "Design and Evaluation of a Continuous Consistency Model for Replicated Services" (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.7743&rep=rep1&type=pdf) di Haifeng Yu e Amin Vahdat. Nessuno dovrebbe interpretare la PAC come implicante che la banca dati sia morta.

Laddove entrambe le parti concordano, tuttavia, è che la risposta alla scala è distribuita parallelizzazione, non, come si pensava una volta, supercomputer grugnito. L'influenza di Eric Brewer sui progetti della Rete delle Postazioni (http://en.wikipedia.org/wiki/Network_of_Workstations) di lavoro (http://en.wikipedia.org/wiki/Network_of_Workstations) della metà degli anni Novanta ha portato alle architetture che hanno esposto il teorema della CAP, perché come dice in un'altra presentazione su Inktomi e Internet Bubble (http://video.google.co.uk/googleplayer.swf?docid=4168901697847625272&hl=en&fs=true) (flash), la risposta è sempre stata i processori che lavorano in parallelo:

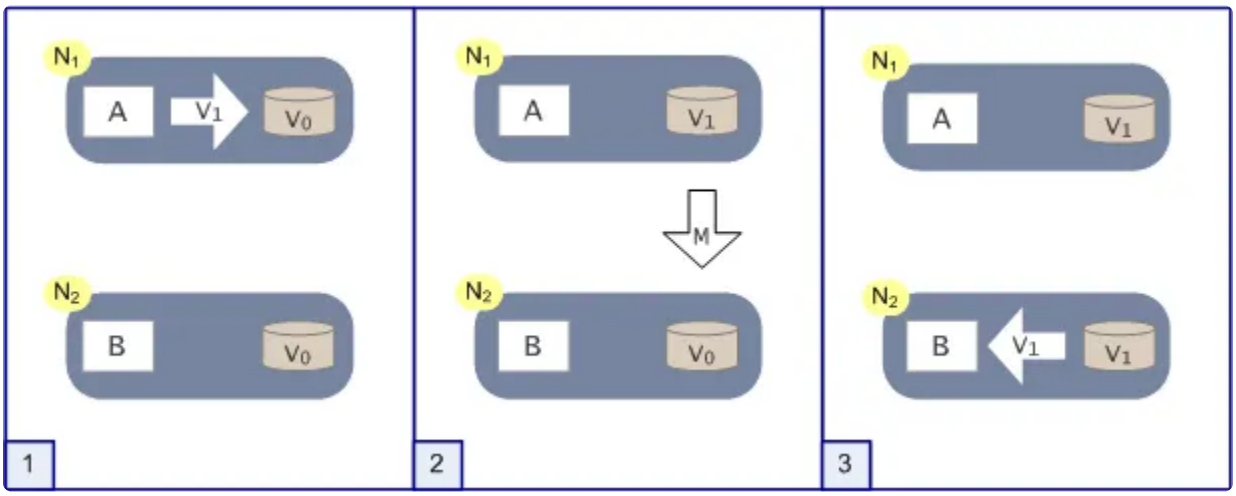
> *Se non stanno lavorando in parallelo non hai alcuna possibilità di ottenere il problema in un ragionevole lasso di tempo. Questo è molto simile a qualsiasi altra cosa. Se hai un lavoro davvero grande da fare, ottieni molte persone a farlo. Quindi, se stai costruendo un ponte, hai molti lavoratori edili. Anche questa è un'elaborazione parallela. Quindi molte di queste cose finiranno per essere "come possiamo mescolare l'elaborazione parallela e Internet?"*

La prova nelle immagini

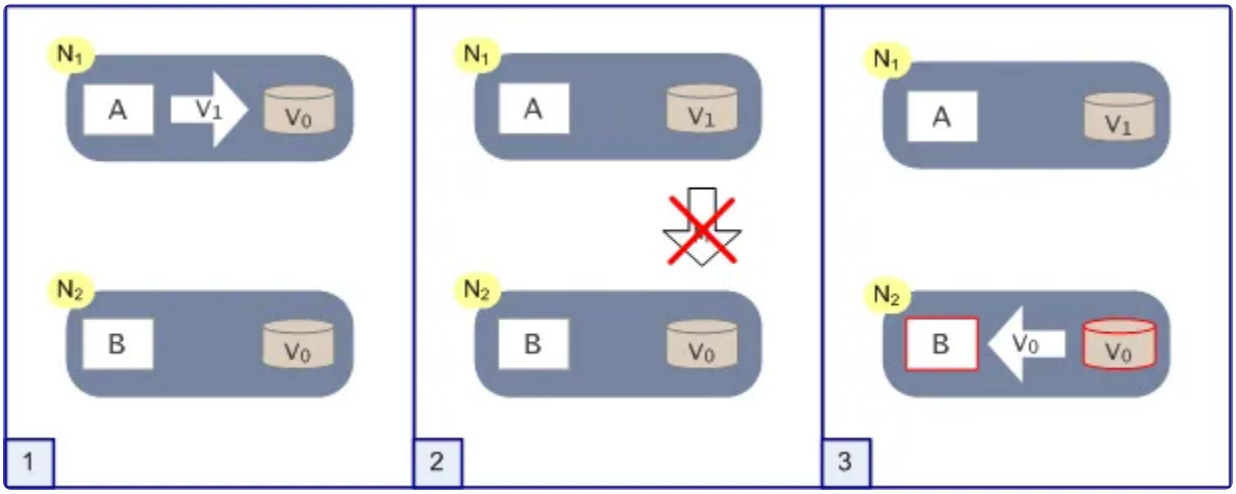
Ecco una prova semplificata, in immagini perché trovo molto più facile capire in questo modo. Ho usato per lo più gli stessi termini di Gilbert e Lynch in modo che questo si lega con il loro articolo.



Il diagramma sopra mostra due nodi in una rete, N $_1$ e N $_2$2. Entrambi condividono un pezzo di dati V (quante copie fisiche di Guerra e Pace sono in magazzino), che ha un valore V $_0$. Correre su N $_1$ è un algoritmo chiamato A che possiamo considerare sicuro, privo di bug, prevedibile e affidabile. L'esecuzione su N $_2$ è un algoritmo simile chiamato B. In questo esperimento, A scrive nuovi valori di V e B legge i valori di V.



In uno scenario di sole giornate questo è ciò che accade: (1) First A scrive un nuovo valore di V, che chiameremo V $_1$1. (2) Quindi un messaggio (M) viene passato da N $_1$ a N $_2$ che aggiorna la copia di V. (3) Ora qualsiasi lettura da B di V tornerà V $_1$1.



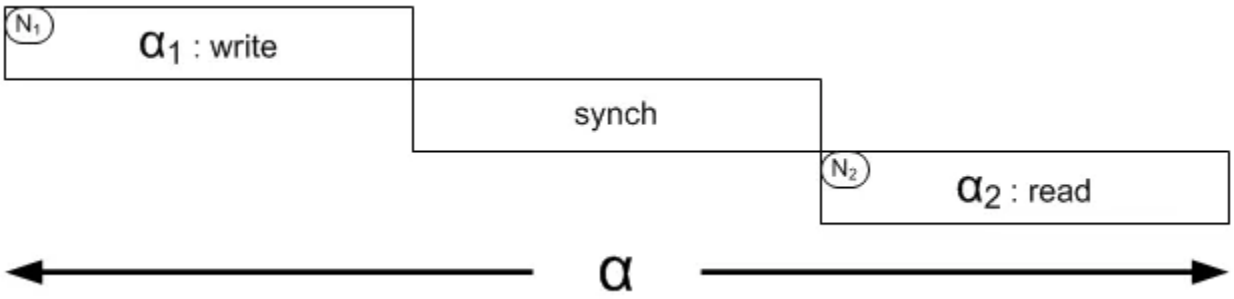
Se le partizioni di rete (cioè i messaggi da N_1 a N_2 non vengono consegnati), N_2 contiene un valore incoerente di V quando si verifica il passaggio (3).

Speriamo che questo sembri abbastanza ovvio. Scalare che si trova fino a poche centinaia di transazioni e diventa un problema importante. Se M è un messaggio asincrono, N_1 non ha modo di sapere se N_2 riceve il messaggio. Anche con la consegna garantita di M, N_1 non ha modo di sapere se un messaggio viene ritardato da un evento di partizione o qualcosa che non funziona in $N_2$2. Rendere M sincrono non aiuta perché questo tratta la scrittura da A su N_1 e l'evento di aggiornamento da $_{N1}$ a N_2 come un'operazione atomica, che ci dà gli stessi problemi di latenza di cui abbiamo già parlato (o peggio). Gilbert e Lynch dimostrano anche, usando una leggera variazione su questo, che anche in un modello parzialmente sincrono (con orologi ordinati su ciascun nodo) l' atomicità non può essere garantita.

Quindi quello che ci dice CAP è che se vogliamo che A e B siano altamente disponibili (cioè lavorando con una latenza minima) e vogliamo che i nostri nodi $_{N1}$ a N_n (dove n potrebbero essere centinaia o addirittura migliaia) rimangano tolleranti delle partizioni di rete (messaggi persi, messaggi non dichiarabili, interruzioni hardware, guasti di processo) allora *a volte* avremo casi in cui alcuni nodi pensano che V_0 (una copia)$_1$- Si tratta di un'altra ciarra.

Vorrebbe davvero che tutto fosse strutturato, coerente e armonioso, come la musica di una band prog rock (http://en.wikipedia.org/wiki/Prog_rock) dei primi anni settanta, ma quello che ci troviamo di fronte è un po' di anarchia in stile punk. E in realtà, anche se potrebbe spaventare le nostre nonne, va bene una volta che lo sai, perché entrambi possono lavorare insieme abbastanza felicemente.

Analizziamo rapidamente questo da una prospettiva transazionale.



Se abbiamo una transazione (cioè un'unità di lavoro basata sull'elemento dati persistente V) chiamata , allora $_1$ potrebbe essere l'operazione di scrittura da prima e $_2$ potrebbe essere la lettura. Su un sistema locale questo sarebbe facilmente gestito da un database con un semplice blocco, isolando qualsiasi tentativo di lettura in ? $_2$ fino a quando $_{\text{non si}}$ completa in modo sicuro. Nel modello distribuito, però, con i nodi N $_1$ e N $_2$ da preoccupare, il messaggio di sincronizzazione intermedia deve anche essere completato. A meno che non possiamo controllare *quando* $_2$si verifica il 2, non possiamo *mai* garantire che vedrà gli stessi valori di dati ? $_1$ scrive. Tutti i metodi per aggiungere il controllo (blocco, isolamento, gestione centralizzata, ecc.) avranno un impatto sulla tolleranza della partizione o sulla disponibilità di 1 $_1$(A) e/o 2 $_2$(B).

Affrontare la PAC

Hai alcune scelte quando affronti i problemi sollevati dalla CAP. Gli ovvi sono:

1. **Tolleranza della partizione di goccia**

   Se si desidera eseguire senza partizioni è necessario impedire che accadano. Un modo per farlo è mettere tutto (relativo a quella transazione) su una macchina, o in un'unità atomicamente infallibile come un rack. Non è garantito al 100% perché puoi ancora avere guasti parziali, ma hai meno probabilità di ottenere effetti collaterali simili a partizioni. Ci sono, naturalmente, significativi limiti di scala a questo.

2. **Disponibilità di goccia**

   Questo è il rovescio della medaglia della medaglia di caduta-partizione-tolleranza. Nell'incontrare un evento di partizione, i servizi interessati attendono semplicemente fino a quando i dati sono coerenti e quindi rimangono non disponibili durante tale periodo. Controllare questo potrebbe diventare abbastanza complesso su molti nodi, con i nodi ri-disponibili che necessitano di logica per gestire il ritorno online con grazia.

3. **La consistenza di Drop**

   Oppure, come dice Werner Vogels, accetta che le cose diventeranno "a certo (http://www.allthingsdistributed.com/2008/12/eventually_consistent.html) Coerenti" (a sud. Dec 2008). L'articolo di Vogels merita una lettura. In molti più dettagli sulle specifiche operative di quanto non lo faccia io.

   Molte incongruenze in realtà non richiedono tanto lavoro come si potrebbe pensare (il che significa che la coerenza continua probabilmente non è qualcosa di cui abbiamo bisogno comunque). Nell'esempio del mio libro ordinata se vengono *are*ricevuti due ordini per l'unico libro in magazzino, il secondo diventa solo un back-order. Finché il cliente viene informato di questo (e ricorda che questo è un caso raro) tutti sono probabilmente felici.

4. **Il BSE Jump**

   La nozione di accettare l'eventuale coerenza è supportata attraverso un approccio architettonico noto come BASE (**B**Bicamente **A** vailable, **S** oft-state, **E** ventually coerente). BASE, come indica il suo nome, è l'opposto logico dell'ACID, anche se sarebbe del tutto sbagliato implicare che qualsiasi architettura dovrebbe (o potrebbe) essere basata interamente sull'uno o sull'altro. Questo è un punto importante da ricordare, data l'abitudine del nostro settore di adozione della strategia "oooh shiny".

   E qui mi rifergo allo stesso professor Brewer che mi ha inviato alcuni commenti su questo articolo, dicendo:

   > *Il termine "BASE" è stato presentato per la prima volta nell'articolo del SOSP del 1997 che citi. Ho trovato l'acronimo con i miei studenti nel loro ufficio all'inizio di quell'anno. Sono d'accordo che è un po 'artificioso, ma lo è anche "ACID" - molto più di quanto la gente si renda conto, quindi abbiamo pensato che fosse abbastanza buono. Jim Gray e io abbiamo discusso di questi acronimi e ha prontamente ammesso che anche l'ACID era un tratto: l'A e la D hanno un'elevata sovrapposizione e la C è mal definita al massimo. Ma la coppia connota l'idea di uno spettro, che è uno dei punti della lezione PODC, come si sottolinea correttamente.*

   Dan Pritchett di eBay ha una bella presentazione (http://softwaresummit.org/2007/speakers/presentations/PritchettArchitectingForLatency.pdf) su BASE.

5. **Design intorno ad esso**

   Guy Pardon, CTO di atomikos (http://www.atomikos.com/) ha scritto un post interessante che ha chiamato "A CAP Solution (Proving Brewer Wrong) (http://guysblogspot.blogspot.com/2008/09/cap-solution-proving-brewer-wrong.html)", suggerendo un approccio architettonico che avrebbe dato coerenza, disponibilità e tolleranza alla partizione, anche se con alcuni avvertimenti (in particolare che non si ottiene tutti e tre garantiti nello stesso istante).

   Vale la pena leggere come Guy rappresenta eloquentemente una visione opposta in questo settore.

Riepilogo di

Che si possa garantire solo due di coerenza, disponibilità e tolleranza per partizione è reale e evidenziato dai siti Web di maggior successo del pianeta. Se funziona per loro non vedo alcun motivo per cui gli stessi compromessi non dovrebbero essere considerati nel design

quotidiano negli ambienti aziendali. Se l'azienda non vuole scalare bene, sono disponibili soluzioni più semplici, ma è una conversazione che vale la pena avere. In ogni caso queste discussioni saranno su progetti appropriati per operazioni specifiche, non sull'intero shebang. Come ha detto Brewer nella sua e-mail "l'unica altra cosa che aggiungerei è che parti diverse dello stesso servizio possono scegliere punti diversi nello spettro". A volte hai assolutamente bisogno di coerenza qualunque sia il costo di scalabilità, perché il rischio di non averlo è troppo grande.

In questi giorni mi direi che Amazon ed eBay non hanno un problema di scalabilità. Penso che ne *avessero* uno e ora hanno gli strumenti per affrontarlo. Ecco perché ne possono parlare liberamente. Qualsiasi ridimensionamento che fanno ora (data la dimensione che già sono) è davvero più o meno lo stesso. Una volta scalati, i tuoi problemi si spostano su quelli di manutenzione operativa, monitoraggio, implementazione di aggiornamenti software ecc. - difficile da risolvere, certamente, ma bello da avere quando hai quei flussi di entrate in arrivo.

Le note

- La maschera di HP su CAP Theorem, un white paper dal titolo "Non c'è pranzo gratuito con dati distribuiti (ftp://ftp.compaq.com/pub/products/storageworks/whitepapers/5983-2544EN.pdf) "
- Note di Informatica sulle Transazioni distribuite (http://www.cogs.susx.ac.uk/courses/dist-sys/node263.html) e le partizioni di rete (http://www.cogs.susx.ac.uk/courses/dist-sys/node286.html) dell'Università del Sussex
- Bellissimo post (http://www.mooseyard.com/Jens/2007/04/twitter-rails-hammers-and-11000-nails-per-second/) di Jens Alfke su database, ridimensionamento e Twitter.
- Il (http://blogs.msdn.com/pathelland/default.aspx) documento di Pat Helland (http://blogs.msdn.com/pathelland/default.aspx) su transazioni distribuite e SOA ha chiamato Data on the Outside versus Data on the Inside (http://www.cidrdb.org/cidr2005/papers/P12.pdf), che in seguito ha riferito al Teorema CAP qui (http://blogs.msdn.com/pathelland/archive/2007/05/20/soa-and-newton-s-universe.aspx)
- Un'altra serie di diapositive (http://cs.gmu.edu/~setia/cs707/slides/replication2.pdf) del corso (http://cs.gmu.edu/~setia/cs707/slides/replication2.pdf) di informatica, questa volta dalla George Mason University in Virginia, su Distributed Software Systems (http://cs.gmu.edu/~setia/cs707/) e in particolare il Teorema CAP e lo scontro tra ACID e ideologia BASE.
- L'opera è del talentuoso Johannes Saurer (http://jsaurer.deviantart.com/). Potete trovare l'originale qui (http://www.deviantart.com/art/Sex-Pistols-125097516) e controllare il suo altro lavoro qui (https://www.deviantart.com/jsaurer/gallery?sort=popularity).

Traduzioni e tras

- Grazie a Hiroshi Yuki (http://www.hyuki.com/), è disponibile una traduzione giapponese (http://www.hyuki.com/yukiwiki/wiki.cgi?BrewersCapTheorem) di questo articolo.

- Grazie a Daniel Cohen (http://www.codeinvain.com/blog/), è disponibile una traduzione in ebraico in due (http://www.codeinvain.com/heblog/169/cap-you-can-have-only-two/) parti (http://www.codeinvain.com/heblog/219/%D7%94%D7%AA%D7%9E%D7%95%D7%93%D7%93%D7%95%D7%AA-%D7%A2%D7%9D-cap/) di questo articolo.

- Grazie a Wang Qi (http://pt.alibaba-inc.com/wp/author/eric/), è disponibile una traduzione cinese (http://pt.alibaba-inc.com/wp/dev_related_728/brewers-cap-theorem.html) di questo articolo.

Le note a piè di pagina

1. Il 4 giugno 1976 è considerato la nascita di Punk Rock nel Regno Unito. Grazie a Charlie Dellacona per aver sottolineato che i Ramones (http://en.wikipedia.org/wiki/The_Ramones) si prendono il merito di aver iniziato il movimento all'inizio del 1974 negli Stati Uniti, anche se le loro registrazioni punk (https://en.wikipedia.org/wiki/Ramones_(album)) ufficiali sono più o meno contemporanee. - Si tratta di un'azione

Creato: 11 gennaio 2009