Alla fine coerente - rivisitato

23 dicembre 2008 • 3261 parole

Ho scritto una prima versione di questo post sui modelli di coerenza circa un anno fa, ma non ne sono mai stato soddisfatto perché è stato scritto in fretta e l'argomento è abbastanza importante da meritare un trattamento più approfondito. ACM Queue mi ha chiesto di rivederlo per utilizzarlo nella loro rivista e ho colto l'occasione per migliorare l'articolo. Questo è il nuovo versione.

Eventualmente coerente: la creazione di sistemi distribuiti affidabili su scala mondiale richiede compromessi tra coerenza e disponibilità.

Alla base del cloud computing di Amazon ci sono servizi infrastrutturali come Amazon S3 (Simple Storage Service), SimpleDB ed EC2 (Elastic Compute Cloud) che forniscono le risorse per costruire piattaforme di elaborazione su scala Internet e una grande varietà di applicazioni. I requisiti imposti a questi servizi infrastrutturali sono molto rigorosi; devono ottenere punteggi elevati nelle aree di sicurezza, scalabilità, disponibilità, prestazioni ed efficacia dei costi e devono soddisfare questi requisiti mentre servono milioni di clienti in tutto il mondo, ininterrottamente.

Dietro le quinte, questi servizi sono enormi sistemi distribuiti che operano su scala mondiale. Questa scala crea ulteriori sfide, perché quando un sistema elabora trilioni e trilioni di richieste, eventi che normalmente hanno una bassa probabilità di verificarsi sono ora garantiti e devono essere considerati in anticipo nella progettazione e nell'architettura del sistema. Data la portata mondiale di questi sistemi, utilizziamo tecniche di replicazione ovunque per garantire prestazioni costanti e alta disponibilità. Sebbene la replicazione ci avvicini ai nostri obiettivi, non può raggiungerli in modo perfettamente trasparente; in una serie di condizioni, i clienti di questi servizi si troveranno di fronte a

conseguenze dell'utilizzo di tecniche di replicazione all'interno dei servizi.

Uno dei modi in cui ciò si manifesta è nel tipo di coerenza dei dati fornita, in particolare quando il sistema distribuito sottostante fornisce un

modello di coerenza finale per la replicazione dei dati. Quando progettiamo questi sistemi su larga scala in Amazon, utilizziamo un set di principi guida e astrazioni correlati alla replicazione dei dati su larga scala e ci concentriamo sui compromessi tra elevata disponibilità e coerenza dei dati. In questo articolo presento alcuni degli elementi di base rilevanti che hanno informato il nostro approccio alla fornitura di sistemi distribuiti affidabili che devono funzionare su scala globale. Una versione precedente di questo testo è apparsa come post sul weblog All Things Distributed nel dicembre 2007 ed è stata notevolmente migliorata con l'aiuto dei suoi lettori.

Prospettiva storica

In un mondo ideale ci sarebbe un solo modello di coerenza: quando viene effettuato un aggiornamento, tutti gli osservatori vedrebbero quell'aggiornamento. La prima volta che questo si è rivelato difficile da realizzare è stato nei sistemi di database della fine degli anni '70. Il miglior "pezzo d'epoca" su questo argomento è "Notes on Distributed Databases" di Bruce Lindsay et al. 5 Espone i principi fondamentali per la replicazione del database e discute una serie di tecniche che riguardano il raggiungimento della coerenza. Molte di queste tecniche cercano di raggiungere la trasparenza della distribuzione, ovvero, all'utente del sistema sembra che ci sia un solo sistema invece di una serie di sistemi collaboranti. Molti sistemi durante questo periodo hanno adottato l'approccio secondo cui era meglio far fallire il sistema completo piuttosto che rompere questa trasparenza.2

A metà degli anni '90, con l'ascesa di sistemi Internet più grandi, queste pratiche furono rivisitate. A quel tempo le persone iniziarono a considerare l'idea che la disponibilità fosse forse la proprietà più importante di questi sistemi, ma si stava lottando con ciò con cui avrebbe dovuto essere scambiata. Eric Brewer, professore di sistemi presso l'Università della California, Berkeley, e all'epoca a capo di Inktomi, riunì i diversi compromessi in un discorso programmatico alla conferenza PODC (Principles of Distributed Computing) nel 2000.1 Presentò il teorema CAP, che afferma che di tre proprietà dei sistemi di dati condivisi (coerenza dei dati, disponibilità del sistema e tolleranza alla partizione di rete), solo due possono essere raggiunte in un dato momento. Una conferma più formale può essere trovata in un articolo del 2002 di Seth Gilbert e Nancy Lynch.4

Un sistema che non tollera le partizioni di rete può ottenere coerenza e disponibilità dei dati, e spesso lo fa utilizzando protocolli di transazione. Per far funzionare questo, i sistemi client e storage devono essere parte dello stesso ambiente; falliscono nel loro insieme in determinati scenari e, come tali, i client non possono osservare le partizioni. Un'osservazione importante è che nei sistemi distribuiti su larga scala, le partizioni di rete sono un dato di fatto; pertanto, la coerenza e la disponibilità non possono essere raggiunte a

allo stesso tempo. Ciò significa che ci sono due scelte su cosa eliminare: rilassare la coerenza consentirà al sistema di rimanere altamente disponibile nelle condizioni di partizionamento, mentre rendere la coerenza una priorità significa che in certe condizioni il sistema non sarà disponibile.

Entrambe le opzioni richiedono che lo sviluppatore client sia consapevole di ciò che il sistema offre. Se il sistema enfatizza la coerenza, lo sviluppatore deve gestire il fatto che il sistema potrebbe non essere disponibile per accettare, ad esempio, una scrittura. Se questa scrittura fallisce a causa dell'indisponibilità del sistema, lo sviluppatore dovrà gestire cosa fare con i dati da scrivere. Se il sistema enfatizza la disponibilità, potrebbe sempre accettare la scrittura, ma in determinate condizioni una lettura non rifletterà il risultato di una scrittura completata di recente. Lo sviluppatore deve quindi decidere se il client richiede l'accesso all'ultimo aggiornamento assoluto in ogni momento. Esiste una gamma di applicazioni in grado di gestire dati leggermente obsoleti e sono ben servite da questo modello.

In linea di principio, la proprietà di coerenza dei sistemi di transazione come definita nelle proprietà ACID (atomicità, coerenza, isolamento, durata) è un diverso tipo di garanzia di coerenza. In ACID, la coerenza è correlata alla garanzia che quando una transazione è terminata il database sia in uno stato coerente; ad esempio, quando si trasferisce denaro da un conto a un altro, l'importo totale detenuto in entrambi i conti non dovrebbe cambiare. Nei sistemi basati su ACID, questo tipo di coerenza è spesso responsabilità dello sviluppatore che scrive la transazione, ma può essere assistito dal database che gestisce i vincoli di integrità.

Coerenza: client e server

Ci sono due modi di guardare alla coerenza. Uno è dal punto di vista dello sviluppatore/cliente: come osservano gli aggiornamenti dei dati. Il secondo modo è dal lato server: come gli aggiornamenti fluiscono attraverso il sistema e quali garanzie possono dare i sistemi rispetto agli aggiornamenti.

Coerenza lato client

Il lato client ha i seguenti componenti:

- **Un sistema di storage.** Per il momento lo tratteremo come una scatola nera, ma si dovrebbe supporre che sotto le coperture sia qualcosa di grande e altamente distribuito, e che sia costruito per garantire durata e disponibilità.
- **Processo A.** Questo è un processo che scrive e legge dal sistema di archiviazione.
- **Processi B e C.** Questi due processi sono indipendenti dal processo A e scrivono e leggono dal sistema di archiviazione. È irrilevante se questi sono

in realtà sono processi o thread all'interno dello stesso processo; ciò che è importante è che siano indipendenti e abbiano bisogno di comunicare per condividere informazioni.

La coerenza lato client ha a che fare con come e quando gli osservatori (in questo caso i processi A, B o C) vedono gli aggiornamenti apportati a un oggetto dati nei sistemi di archiviazione. Negli esempi seguenti che illustrano i diversi tipi di coerenza, il processo A ha apportato un aggiornamento a un oggetto dati:

- **Forte coerenza.** Dopo il completamento dell'aggiornamento, qualsiasi accesso successivo (tramite A, B o C) restituiranno il valore aggiornato.
- **Debole coerenza.** Il sistema non garantisce che gli accessi successivi restituiranno il valore aggiornato. È necessario che siano soddisfatte alcune condizioni prima che il valore venga restituito. Il periodo tra l'aggiornamento e il momento in cui è garantito che qualsiasi osservatore vedrà sempre il valore aggiornato è denominato *finestra di incoerenza.*

- **Eventual coerenza.** Questa è una forma specifica di weak coerenza; il sistema di archiviazione garantisce che se non vengono apportati nuovi aggiornamenti all'oggetto, alla fine tutti gli accessi restituiranno l'ultimo valore aggiornato. Se non si verificano errori, la dimensione massima della finestra di incoerenza può essere determinata in base a fattori quali ritardi di comunicazione, carico sul sistema e numero di repliche coinvolte nello schema di replicazione. Il sistema più diffuso che implementa l'eventual coerenza è DNS (Domain Name System). Gli aggiornamenti a un nome vengono distribuiti in base a uno schema configurato e in combinazione con cache a tempo controllato; alla fine, tutti i client vedranno l'aggiornamento.

Il modello di coerenza finale presenta una serie di varianti che è importante considerare:

- **Coerenza causale.** Se il processo A ha comunicato al processo B di aver aggiornato un elemento dati, un accesso successivo da parte del processo B restituirà il valore aggiornato e una scrittura è garantita per sostituire la scrittura precedente. L'accesso da parte del processo C che non ha alcuna relazione causale con il processo A è soggetto alle normali regole di coerenza finale.
- **Coerenza Read-your-writes.** Questo è un modello importante in cui il processo A, dopo aver aggiornato un elemento dati, accede sempre al valore aggiornato e non vedrà mai un valore più vecchio. Questo è un caso speciale del modello di coerenza causale.

- **Coerenza di sessione.** Questa è una versione pratica del modello precedente, in cui un processo accede al sistema di archiviazione nel contesto di una sessione. Finché la sessione esiste, il sistema garantisce la coerenza di lettura-scrittura. Se la sessione termina a causa di un determinato scenario di errore, è necessario creare una nuova sessione e le garanzie non si sovrappongono alle sessioni.

- ÿ **Consistenza di lettura monotona.** Se un processo ha visto un valore particolare per l'oggetto, tutti gli accessi successivi non restituiranno mai alcun valore precedente.
- ÿ **Consistenza di scrittura monotona.** In questo caso il sistema garantisce di serializzare le scritture tramite lo stesso processo. I sistemi che non garantiscono questo livello di coerenza sono notoriamente difficili da programmare.

Un certo numero di queste proprietà può essere combinato. Ad esempio, si possono ottenere letture monotone combinate con coerenza a livello di sessione. Da un punto di vista pratico, queste due proprietà (letture monotone e read-your-writes) sono le più desiderabili in un sistema di coerenza finale, ma non sempre richieste. Queste due proprietà semplificano la creazione di applicazioni da parte degli sviluppatori, consentendo al contempo al sistema di archiviazione di allentare la coerenza e fornire elevata disponibilità.

Come si può vedere da queste varianti, sono possibili diversi scenari. Dipende dalle applicazioni specifiche se si può o meno gestire l' conseguenze.

La coerenza finale non è una proprietà esoterica dei sistemi distribuiti estremi. Molti moderni RDBMS (sistemi di gestione di database relazionali) che forniscono affidabilità primaria-backup implementano le loro tecniche di replicazione sia in modalità sincrona che asincrona. In modalità sincrona l'aggiornamento della replica è parte della transazione. In modalità asincrona gli aggiornamenti arrivano al backup in modo ritardato, spesso tramite log shipping. In quest'ultima modalità, se il primario fallisce prima che i log vengano spediti, la lettura dal backup promosso produrrà valori vecchi e incoerenti. Inoltre, per supportare migliori prestazioni di lettura scalabili, gli RDBMS hanno iniziato a fornire la possibilità di leggere dal backup, che è un caso classico di fornitura di garanzie di coerenza finale in cui le finestre di incoerenza dipendono dalla periodicità del log shipping.

Coerenza lato server

Sul lato server dobbiamo dare un'occhiata più approfondita al modo in cui gli aggiornamenti fluiscono attraverso il sistema per capire cosa guida le diverse modalità che lo sviluppatore che usa il sistema può sperimentare. Stabiliamo alcune definizioni prima di iniziare:

N = il numero di nodi che memorizzano le repliche dei dati

W = il numero di repliche che devono confermare la ricezione dell'aggiornamento prima che l'aggiornamento venga completato

R = il numero di repliche che vengono contattate quando si accede a un oggetto dati tramite un'operazione di lettura

Se W+R > N, allora il set di scrittura e il set di lettura si sovrappongono sempre e si può garantire una forte coerenza. Nello scenario RDBMS primary-backup, che implementa la replica sincrona, N=2, W=2 e R=1. Indipendentemente dalla replica da cui il client legge, otterrà sempre una risposta coerente. Nella replica asincrona con lettura dal backup abilitata, N=2, W=1 e R=1. In questo caso R+W=N e la coerenza non può essere garantita.

Il problema con queste configurazioni, che sono protocolli di quorum di base, è che quando il sistema non riesce a scrivere su W nodi a causa di guasti, l'operazione di scrittura deve fallire, segnando l'indisponibilità del sistema. Con N=3 e W=3 e solo due nodi disponibili, il sistema dovrà fallire la scrittura.

Nei sistemi di storage distribuiti che devono fornire elevate prestazioni e alta disponibilità, il numero di repliche è in genere superiore a due. I sistemi che si concentrano esclusivamente sulla tolleranza agli errori spesso utilizzano N=3 (con configurazioni W=2 e R=2).
I sistemi che devono gestire carichi di lettura molto elevati spesso replicano i propri dati oltre quanto richiesto per la tolleranza agli errori; N può essere costituito da decine o addirittura centinaia di nodi, con R configurato a 1 in modo che una singola lettura restituisca un risultato. I sistemi interessati alla coerenza sono impostati su W=N per gli aggiornamenti, il che può ridurre la probabilità che la scrittura abbia successo. Una configurazione comune per questi sistemi interessati alla tolleranza agli errori ma non alla coerenza è quella di eseguire con W=1 per ottenere una durata minima dell'aggiornamento e quindi affidarsi a una tecnica pigra (epidemica) per aggiornare le altre repliche.

Il modo in cui configurare N, W e R dipende dal caso comune e dal percorso di prestazioni che deve essere ottimizzato. In R=1 e N=W ottimizziamo per il caso di lettura e in W=1 e R=N ottimizziamo per una scrittura molto veloce. Ovviamente in quest'ultimo caso, la durabilità non è garantita in presenza di guasti e se W < (N+1)/2, c'è la possibilità di scritture in conflitto quando i set di scrittura non si sovrappongono.

La coerenza debole/eventuale si verifica quando W+R <= N, il che significa che c'è la possibilità che il set di lettura e scrittura non si sovrappongano. Se questa è una configurazione deliberata e non basata su un caso di errore, allora non ha molto senso impostare R su un valore diverso da 1. Ciò accade in due casi molto comuni: il primo è la replicazione massiccia per il ridimensionamento della lettura menzionata in precedenza; il secondo è dove l'accesso ai dati è più complicato. In un semplice modello chiave-valore è facile confrontare le versioni per determinare l'ultimo valore scritto nel sistema, ma nei sistemi che restituiscono set di oggetti è più difficile determinare quale dovrebbe essere il set più recente corretto. Nella maggior parte di questi sistemi in cui il set di scrittura è più piccolo del set di replica, è in atto un meccanismo che applica gli aggiornamenti in modo pigro al rimanente

nodi nel set della replica. Il periodo fino a quando tutte le repliche non sono state aggiornate è la finestra di incoerenza discussa in precedenza. Se W+R <= N, il sistema è vulnerabile alla lettura da nodi che non hanno ancora ricevuto gli aggiornamenti.

Che si possano o meno ottenere read-your-write, sessione e coerenza monotona dipende in generale dalla "stickiness" dei client al server che esegue il protocollo distribuito per loro. Se questo è lo stesso server ogni volta, allora è relativamente facile garantire read-your-write e letture monotone. Ciò rende leggermente più difficile gestire il bilanciamento del carico e la tolleranza agli errori, ma è una soluzione semplice. L'utilizzo di sessioni, che sono sticky, rende questo esplicito e fornisce un livello di esposizione su cui i client possono ragionare.

A volte il client implementa read-your-writes e letture monotone. Aggiungendo versioni sulle scritture, il client scarta le letture di valori con versioni che precedono l'ultima versione visualizzata.

Le partizioni si verificano quando alcuni nodi nel sistema non riescono a raggiungere altri nodi, ma entrambi i set sono raggiungibili da gruppi di client. Se si utilizza un approccio di quorum di maggioranza classico, la partizione che ha W nodi del set di repliche può continuare a ricevere aggiornamenti mentre l'altra partizione diventa non disponibile. Lo stesso vale per il set di lettura. Dato che questi due set si sovrappongono, per definizione il set di minoranza diventa non disponibile. Le partizioni non si verificano frequentemente, ma si verificano tra i data center, così come all'interno dei data center.

In alcune applicazioni, la non disponibilità di una qualsiasi delle partizioni è inaccettabile ed è importante che i client che possono raggiungere quella partizione facciano progressi. In tal caso, entrambe le parti assegnano un nuovo set di nodi di archiviazione per ricevere i dati e viene eseguita un'operazione di unione quando la partizione viene riparata. Ad esempio, in Amazon il carrello della spesa utilizza un sistema di scrittura sempre; nel caso della partizione, un cliente può continuare a mettere articoli nel carrello anche se il carrello originale si trova sulle altre partizioni. L'applicazione del carrello aiuta il sistema di archiviazione a unire i carrelli una volta che la partizione è stata riparata.

La dinamo di Amazon

Un sistema che ha portato tutte queste proprietà sotto il controllo esplicito dell'architettura dell'applicazione è Dynamo di Amazon, un sistema di archiviazione chiave-valore utilizzato internamente in molti servizi che compongono la piattaforma di e-commerce di Amazon, nonché nei Web Services di Amazon. Uno degli obiettivi di progettazione di Dynamo è consentire al proprietario del servizio applicativo che crea un'istanza del sistema di archiviazione Dynamo, che in genere si estende su più data center, di effettuare compromessi tra coerenza, durata, disponibilità e prestazioni a un certo costo

punto.3

Riepilogo

L'incoerenza dei dati nei sistemi distribuiti affidabili su larga scala deve essere tollerata per due motivi: migliorare le prestazioni di lettura e scrittura in condizioni di elevata concorrenza e gestire i casi di partizione in cui un modello di maggioranza renderebbe parte del sistema non disponibile anche se i nodi sono attivi e funzionanti.

L'accettabilità o meno delle incongruenze dipende dall'applicazione client.
In tutti i casi lo sviluppatore deve essere consapevole che le garanzie di coerenza sono fornite dai sistemi di archiviazione e devono essere prese in considerazione quando si sviluppano le applicazioni. Ci sono una serie di miglioramenti pratici al modello di coerenza finale, come la coerenza a livello di sessione e le letture monotone, che forniscono strumenti migliori per lo sviluppatore. Molte volte l'applicazione è in grado di gestire le garanzie di coerenza finale del sistema di archiviazione senza alcun problema. Un caso popolare specifico è un sito Web in cui possiamo avere la nozione di coerenza percepita dall'utente. In questo scenario la finestra di incoerenza deve essere più piccola del tempo previsto per il ritorno del cliente per il caricamento della pagina successiva. Ciò consente agli aggiornamenti di propagarsi attraverso il sistema prima che sia prevista la lettura successiva.

L'obiettivo di questo articolo è quello di aumentare la consapevolezza sulla complessità dei sistemi di ingegneria che devono operare su scala globale e che richiedono un'attenta messa a punto per garantire che possano fornire la durata, la disponibilità e le prestazioni richieste dalle loro applicazioni. Uno degli strumenti a disposizione del progettista di sistema è la lunghezza della finestra di coerenza, durante la quale i client dei sistemi sono potenzialmente esposti alle realtà dell'ingegneria dei sistemi su larga scala.

Riferimenti

1. Brewer, EA 2000. Verso sistemi distribuiti robusti (astratto). In . *Atti del 19° Simposio annuale ACM sui principi del calcolo distribuito* (16-19 luglio, Portland, Oregon): 7 2. Una conversazione con Bruce Lindsay. 2004. ACM Queue 2(8): 22-33.
3. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W. 2007. Dynamo: Amazon's highly available key-value store. In Proceedings of the 21st ACM *Symposium on Operating Systems Principles* (Stevenson, Washington, ottobre).

4. Gilbert, S., Lynch, N. 2002. La congettura di Brewer e la fattibilità di modelli coerenti,

disponibili. servizi Web partizionabili. ACM SIGACT News 33(2).

5. Lindsay, BG, Selinger, PG, et al. 1980. Note sui database distribuiti. In *Basi di dati distribuite, a cura di* IW Draffan e F. Poole, 247-284. Cambridge: Cambridge University Press. Disponibile anche come IBM Research Report RJ2517, San Jose, California (luglio 1979).