10. Il problema del consensus

Introduzione

La risposta alla domanda del perché i provider di servizi riescono a garantire un'affidabilità di server parziale e non totale è legata all'**impossibilità del consensus**. Si pensi a un gruppo di server che:

- Intendono assicurarsi che ciascuno di essi riceva gli stessi aggiornamenti nel **medesimo ordine**. Ciò è detto **Reliable Multicast**;
- Intendono mantenere le liste locali in cui ciascuno di essi conosce lo stato dell'altro, così come quando crashano o abbandonano, con aggiornamenti simultanei. Ciò è detto **Membership/Failure Detection**;
- Intendono eleggere un leader. Ciò è detto **Leader Election**;
- Intendono assicurarsi che un solo processo alla volta possa accedere a un file critico. Ciò è detto **Mutual Exclusion**.

Pensiamo quindi a ciascun server come un **processo;** tutti questi processi, in gruppo, provano a **coordinarsi** tra loro per raggiungere un accordo circa:

- L'ordine dei messaggi;
- Lo stato dei processi;
- Il leader;
- L'accesso critico.

Deve esserci quindi un **consenso** da parte di questi processi.

Le variabili dei processi

Ciascuno degli **n** processi **p** dispone di due variabili, la prima di input Xp che assume valori booleani, ossia 0 o 1, e rappresenta la decisione iniziale presa dal singolo processo. La seconda variabile invece, prende il nome di Yp, è anch'essa booleana e rappresenta l'output, ossia, la decisione finale del processo **p** la quale non può essere modificata.

Il problema del consensus prevede di progettare un protocollo, per il quale **alla fine** tutti i processi impostano le loro variabili d'uscita a 0 o in alternativa ad 1. In poche parole, devono essere tutti d'accordo nel decidere il medesimo valore, e come già detto, una volta che tale decisione viene intrapresa, questa non può essere cambiata.

Cosa deve garantire il consenso

Devono essere rispettati:

- **Validità:** se tutti i processi propongono il **medesimo** valore, la decisione è quella definitiva;
- **Integrità:** il valore deciso deve essere proposto dai processi;
- **Non-trivialità:** deve esserci almeno uno stato di sistema iniziale che conduca al risultato finale.

IVAN SOLLAZZO - ANDREA PISCOPO - NICOLAS BRUNO
PROFESSORE PIERLUIGI GALLO - INGEGNERIA CIBERNETICA
UNIVERSITA' DEGLI STUDI DI PALERMO

I benefici del consenso

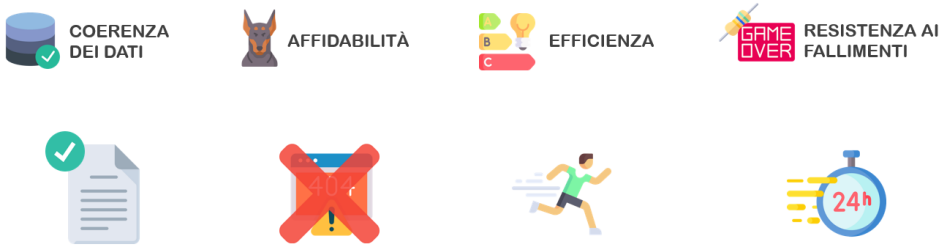
Arrivare al consenso è la base di partenza per:

1. Coerenza dei dati: Quando tutti i nodi di un sistema distribuito raggiungono il consenso su un determinato valore o stato, si può essere sicuri che i dati presenti in ogni nodo siano coerenti e aggiornati.

2. Maggiore affidabilità: Raggiungere il consenso può aiutare a ridurre la probabilità di errori o conflitti nel sistema distribuito, aumentando la sua affidabilità.

3. Efficienza: Un sistema distribuito che raggiunge rapidamente il consenso è più efficiente perché i nodi non devono continuamente scambiarsi informazioni per cercare di raggiungere un accordo.

4. Resistenza ai fallimenti: Il consenso può aiutare un sistema distribuito a tollerare la caduta dei nodi senza interrompere il servizio



Consensus nei modelli di sistemi distribuiti

Si fa distinzione tra due modelli di sistemi distribuiti differenti:

- **Sistemi sincroni:** ciascun messaggio viene ricevuto entro un certo tempo limite e il clock drift di ciascun processo ha un suo limite. **Il consensus è sempre raggiungibile** (a patto che non vi siano della condizioni avverse)
- **Sistemi asincroni:** non ci sono limiti nell'esecuzione dei processi e l'orologio locale può avere un drift arbitrario. **Il consensus non è mai raggiungibile**, si possono però trovare delle soluzioni "probabilistiche" facendo uso di algoritmi come **Paxos** o derivati

Risolvere il consensus in sistemi sincroni

Dato un sistema con **n** processi, se **p** processi crashano (processi sincronizzati che operano in round) avremo un vettore dei valori proposti chiamato $values^r_i$ dove **r** sta per il numero di round e **i** per ID processo. Ad esempio al round 0: $values_i^0 = \{\}$, al round successivo invece avremo $values_i^1 = \{v\}$ dove v rappresenta la decisone presa dall'i-esimo nodo. **In altre parole ognuno propone un valore e lo diffonde all'interno del sistema, così che tutti possano riempire il proprio vettore values con i valori ricevuti dagli altri.** Alla fine dei round, ciascun processo avrà il suo vettore compilato ma non tutti i vettori potrebbero essere della stessa lunghezza dato che alcuni processi, strada facendo, potrebbero non avere inviato/ricevuto il valore della decisione; in questo modo solo i processi che sono rimasti sempre online avranno gli stessi valori e quindi potranno impostare la loro variabile **Yp** al valore concordato.

Risolvere il consensus in sistemi asincroni con Paxos

Come detto in precedenza, nei sistemi asincroni non è possibile raggiungere il consenso (dimostrato da FLP Fischer, Lynch, and Paterson proof), ciò è legato alla natura stessa dei sistemi asincroni, dato che la mancanza di sincronizzazione rende **difficile distinguere un processo lento da un processo fallito**, qua entra in gioco Paxos che non risolve il problema del consenso ma ne argina gli effetti, garantendo sicurezza (**garantisce che due processi non falliti abbiano lo stesso valore**) e che il processo sia effettivamente attivo e non fallito , esistono differenti versioni di Paxos come Zookeeper di apache e Google chubby.

Il funzionamento di Paxos

l protocollo di Paxos è un protocollo di consenso distribuito che viene utilizzato per raggiungere un accordo su un valore comune in un sistema distribuito. Il protocollo si divide in tre fasi principali: elezione del leader, fase di proposta (o fase Bill) e fase di accettazione (o fase Law).

1. Elezione di un Leader (o fase di Election): Inizialmente, tutti i nodi del sistema sono in uno stato non inizializzato e non conoscono l'identità del leader. L'elezione del leader è il processo che determina quale nodo agirà come leader nel sistema. **Il protocollo prevede che venga eletto un leader tra i nodi del sistema in modo casuale o con una specifica politica di elezione. Una volta eletto, il leader inizia a ricevere le proposte dai nodi e coordina il processo di consenso.**

2. Fase di proposta (o fase Bill): In questa fase, i nodi propongono un valore che vogliono far accettare al sistema. Un nodo propone un valore al leader, che lo propaga agli altri nodi. **Il leader riceve queste proposte, le valuta e le combina, se possibile, in un'unica proposta.** Questa proposta combinata viene poi inviata a tutti i nodi del sistema. Se il leader si blocca o smette di funzionare, il protocollo prevede una nuova elezione del leader.

3. Fase di accettazione (o fase Law): In questa fase, **i nodi del sistema accettano o rifiutano la proposta (composta) del leader**. I nodi possono accettare una proposta solo se la maggioranza dei nodi nel sistema accetta la stessa proposta. Se la proposta non viene accettata, si torna alla fase Bill e il leader può formulare una nuova proposta. Se la proposta viene accettata, il leader informa tutti i nodi del sistema del valore accettato.

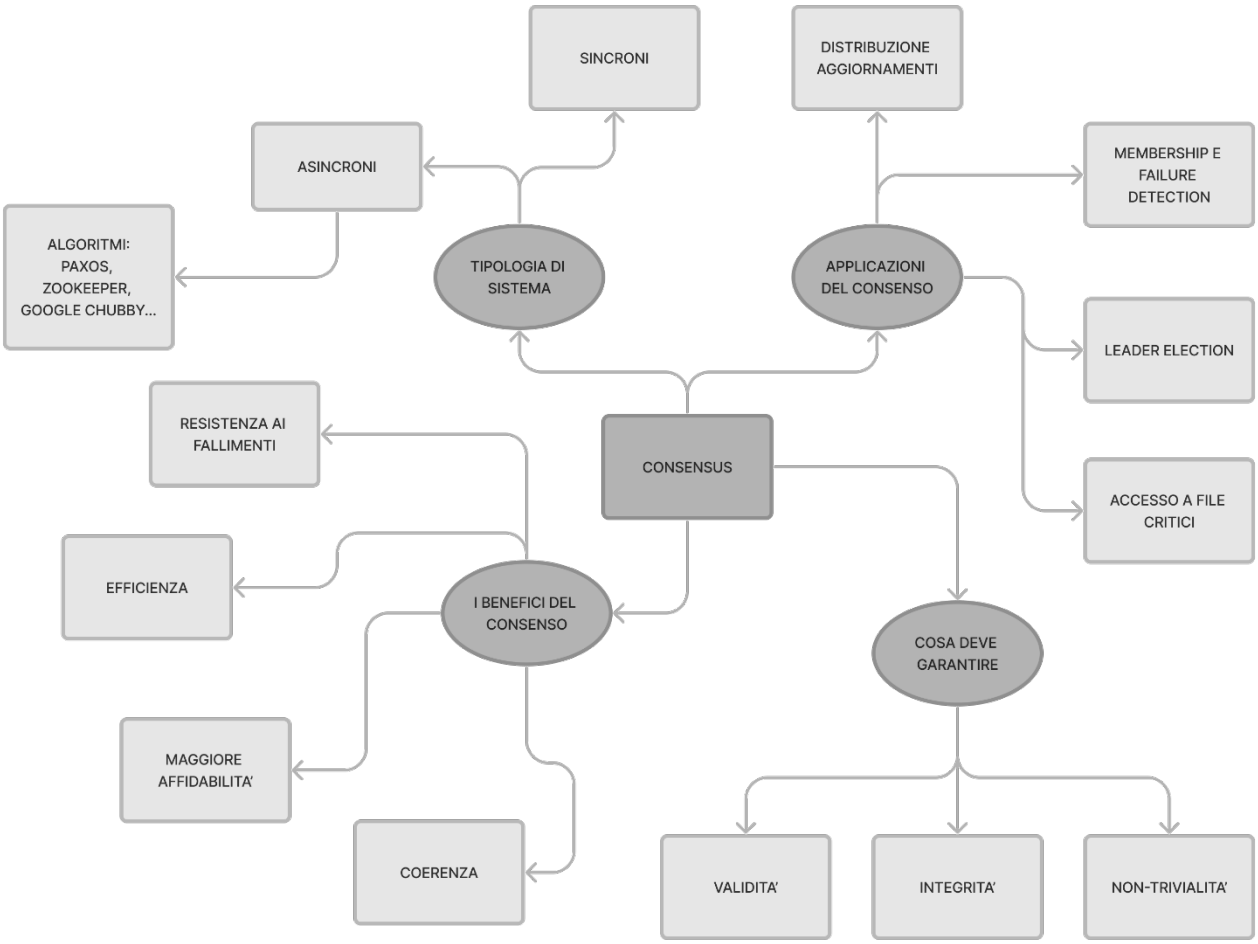
Cosa può andare storto ? I processi possono fallire, il leader può fallire e i messaggi scambiati possono essere persi.

Approfondimento ricevimento

Se un nodo in un sistema asincrono perde pochi round o arriva in ritardo può partecipare ugualmente alla decisione del valore finale, se vengono persi tanti round tale non può più contribuire alla decisione del valore finale.

IVAN SOLLAZZO - ANDREA PISCOPO - NICOLAS BRUNO
PROFESSORE PIERLUIGI GALLO - INGEGNERIA CIBERNETICA
UNIVERSITA' DEGLI STUDI DI PALERMO

Mappa Capitolo



IVAN SOLLAZZO - ANDREA PISCOPO - NICOLAS BRUNO
PROFESSORE PIERLUIGI GALLO - INGEGNERIA CIBERNETICA
UNIVERSITA' DEGLI STUDI DI PALERMO