Stabilità

- Una transazione è una certa unità di lavoro che un sistema elabora
- Una unità di lavoro può comprendere vari aggiornamenti del database
- Per esempio una transazione è "Un cliente fa un ordine"
- Questa operazione include tanto codice e persino l'integrazione con sistemi esterni per la verifica della carta di credito
- Il sistema è un insieme di componenti hardware (host e rete), applicazioni e servizi che supportano la transazione.
- Un sistema **resiliente** continua a elaborare transazioni persino quando ci sono guasti nei componenti che disturbano l'elaborazione. Questo è spesso inteso come **stabilità**. Non solo che server e applicazioni sono in esecuzione, ma che l'utente può svolgere del lavoro

Usa timeout

- Oggi tutte le applicazioni non banali usano la rete in qualche modo, esponendo l'applicazione al possibile fallimento della rete
  - Qualsiasi dispositivo della rete può smettere di funzionare in qualunque momento
- Quando ciò capita, il nostro codice non può aspettare indefinitamente una risposta che potrebbe non arrivare mai, prima o poi deve rinunciare
- Il timeout è un semplice meccanismo che permette di smettere di aspettare una risposta quando si sospetta che non arriverà
- Timeout ben posizionati permettono di isolare i guasti
  - Un problema in un sistema, sottosistema o dispositivo non si propaga

Usa timeout

- Purtroppo molte API di alto livello non hanno impostazioni per timeout
  - Varie librerie si basano sulla comunicazione tramite socket, tuttavia nascondono l'implementazione e non permettono l'impostazione di timeout
- Anche per una singola applicazione i timeout sono rilevanti
  - Un pool di risorse può esaurirsi
  - Generalmente, il thread chiamante si dovrebbe bloccare finché una risorsa diventa disponibile
- Un pool di risorse che blocca i thread dovrebbe avere un timeout per assicurare che i thread siano sbloccati sia quando le risorse diventano disponibili che quando non lo diventano

Usa timeout

- Quando si usa il metodo `wait()` di `Object`, usare la versione che prende un timeout come parametro
- Quando si usa la libreria *java.util.concurrent* (tipo `BlockingQueue` e sottotipi) usare `poll()`, `offer()`, e per `Lock` usare `tryLock()` che prendono un timeout come parametro
- Il codice dell'applicazione può essere invaso da codice per la gestione di timeout, ovvero error-handling. Questo codice tuttavia mischia codice relativo alle funzionalità e codice necessario per ottenere resilienza, indispensabile per andare in produzione
- Un modo per gestire i timeout è organizzare operazioni che prendono molto tempo in un set di primitive che sono riusate in vari posti

Usa timeout

- Esempio: si vuol prendere un id di connessione al database da un pool, eseguire una query, organizzare in oggetti i risultati della query, restituire l'id di connessione al pool
- In tre punti potrebbe capitare che le chiamate attendono per un tempo infinito
- Anziché codificare la sequenza di interazioni in vari posti dell'applicazione, tutti associati con la gestione dei timeout, si può creare un QueryObject che rappresenta la parte di interazione che cambia
- Usare un generico Gateway che fornisce un template per la gestione di connessione, errori, esecuzione di query e organizzazione dei risultati
- Inserire tutte le interazioni in una singola classe rende più facile applicare il design pattern Circuit Breaker

Usa timeout

- Spesso si riprova ad eseguire l'operazione che ha provocato il timeout
- Riprovare immediatamente dopo il fallimento l'operazione potrebbe farla fallire nuovamente. Se il problema è transitorio, dall'altra parte della connessione, riprovare dopo un po'
- Da ricordare
  - Applicare il pattern timeout nei punti di integrazione, in modo che le chiamate a tali punti non diventino thread bloccati. Si evitano così fallimenti a cascata
  - Quando un'operazione prende troppo tempo, alcune volte bisogna comunque andar avanti, il pattern timeout ci permette di far questo
  - Riprovare immediatamente potrebbe generare lo stesso problema, e lo stesso timeout. Il più delle volte si dovrebbe accodare l'operazione e riprovare dopo

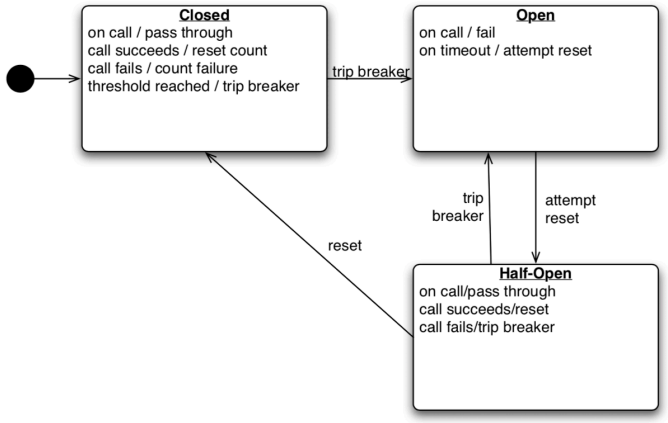
Interruttore di Circuito

- Nel caso di circuiti con fili elettrici, il riscaldamento prodotto è proporzionale alla corrente assorbita al quadrato per la resistenza (I^2R). Più dispositivi collegati, più corrente assorbita, quindi più calore, quindi si ha il rischio che certe parti prendano fuoco (case in legno)
- La soluzione è stata il fusibile, è un componente che si brucia prima (fallisce prima) che il calore diventa eccessivo per la casa, interrompendo il circuito
- In modo più astratto, il dispositivo di interruzione del circuito esiste per permettere ad un sottosistema di fallire (per l'eccessiva corrente assorbita) senza distruggere l'intero sistema. Una volta che il pericolo è passato, il dispositivo di interruzione può essere reimpostato per ripristinare il pieno funzionamento del sistema

Interruttore di Circuito

- Applicando la stessa tecnica sul software, si può schermare un'operazione pericolosa con un componente che evita le chiamate quando il sistema non funziona bene
  - Questo non è la sessa cosa che riprovare a chiamare, le operazioni sono evitate piuttosto che eseguite nuovamente
- Nello stato normale di "chiuso", il circuit breaker (interruttore di circuito) esegue le operazioni. Queste possono essere chiamate ad un altro sistema, o chiamate interne soggette a timeout o fallimenti.
  - Se la chiamata ha successo non succede niente di speciale
  - Se la chiamata fallisce, il circuit breaker prende nota, e una volta che un certo numero di fallimenti (o frequenza di fallimenti) eccede una soglia, il circuit breaker "apre" il circuito
- Quando il circuito è "aperto", le chiamate al circuit breaker falliscono immediatamente senza alcun tentativo di eseguire l'operazione

Interruttore di Circuito



Interruttore di Circuito

- Dopo una certa quantità di tempo, il circuit breaker decide che l'operazione potrebbe riuscire e va nello stato di "semi-aperto"
  - La prossima chiamata è abilitata ad eseguire l'operazione a rischio
  - Se la chiamata ha successo il circuit breaker ritorna nello stato di "chiuso"
  - Se la chiamata fallisce il circuit breaker ritorna allo stato di "aperto", fino allo scadere di un altro timeout
- I circuit breaker sono un modo per degradare le funzionalità quando il sistema è sovraccarico
- Alcune scelte coinvolgono il cliente, es. cosa si fa se non si può verificare la carta di credito?

Interruttore di Circuito

- I cambiamenti di stato del circuit breaker dovrebbero essere registrati (log), e lo stato corrente dovrebbe essere mostrato per in caso di monitoring
- Circuit breaker possono essere efficaci per problemi nei punti di integrazione, fallimenti a cascata, capacità non bilanciate e risposte lente
- Da ricordare
  - E' il pattern fondamentale per proteggere il sistema da tutti i tipi di problemi derivanti dai punti di interazione. Quando vi è una difficoltà con i punti di integrazione, non chiamare più
  - Usare insieme al pattern timeout, quest'ultimo indica che c'è un problema con i punti di integrazione
  - Un Circuit Breaker che entra in azione è l'indicazione che c'è un problema significativo. Le relative operazioni dovrebbero essere riportate, registrate, osservate come trend, correlate

Bulkheads (Paratie)

- In una nave, le paratie sono barriere che possono essere chiuse per dividere la nave in compartimenti a tenuta. Le paratie prevengono che l'acqua si sposti da una sezione ad un'altra. Così una singola falla nello scafo non affonda la nave
- La paratia impone il principio di contenimento del danno.
- Partizionando un sistema software si può evitare che il fallimento in una parte distrugga tutto
- La ridondanza fisica è la forma più comune di paratia. Se ci sono server indipendenti, il fallimento di un hardware non coinvolge altri
- Su larga scala, ci potrebbero essere varie farm di server, con alcune farm dedicate a parti mission-critical. Un sistema di biglietti potrebbe avere server dedicati per il check-in, questi non sono influenzati da altri che gestiscono query sullo stato dei voli, che potrebbero essere sovraccarichi da tante richieste

Bulkheads (Paratie)

- Con un pool di server condiviso fra due servizi in esecuzione, considerando che i picchi di richieste ai due servizi non sono coincidenti, la capacità totale necessaria è minore della somma delle singole capacità richieste dai servizi
- L'efficienza nell'uso delle risorse è una buona motivazione per l'adozione di un'architettura orientata ai servizi. Tuttavia come avere sia l'efficienza che la sicurezza data dalle paratie?
- La virtualizzazione permette di ottenere entrambe. Le risorse hardware sono condivise, e ospitano server virtuali dedicati
  - Per cambiare la capacità concessa basta agire a livello amministrativo sulle macchine virtuali
  - Si può fare il boot di macchine virtuali aggiuntive, o le macchine virtuali possono essere migrate su altri server

Bulkheads (Paratie)

- Su scala più piccola, CPU binding è una forma di paratia. Fare il binding di un processo a una CPU evita che un processo fuori controllo usi tutti i cicli di CPU di tutte le CPU e tira giù tutta la macchina
- Da ricordare
  - Il pattern permette di preservare una parte di funzionalità quando capita qualcosa di brutto
  - Il partizionamento dei server fa sì che ogni partizione ha la sua riserva di capacità. Se tutti i server sono condivisi, sarà necessaria meno capacità totale
  - Scegliere la granularità del partizionamento, si possono avere partizioni in thread pool nell'applicazione, di CPU in un server, o di server in un cluster
  - In un ambiente orientato ai servizi, se un'applicazione va giù per una catena di reazioni, grazie alla paratia si evita che vada giù tutto

Sistemi Reattivi

- I sistemi reattivi esibiscono Resilienza e si basano sulla Messagistica
- Resilienza (vedi slide precedenti)
  - Il sistema risponde in tempi appropriati sebbene in presenza di guasti. Ciò è ottenuto tramite repliche, contenimento, isolamento, e delega. I guasti sono isolati all'interno di un componente, in modo che parti che si guastano (bloccano) e si riavviano non compromettono l'intero sistema
  - Il client non è sovraccaricato dalla gestione dei fallimenti del componente
- Messagistica (vedi pattern successivi)
  - Ci si affida ad un sistema di passaggio di messaggi asincroni per stabilire una connessione fra componenti che assicura un lasco accoppiamento, isolamento, e location transparency