Luca Cabibbo

Architettura dei Sistemi Software

Disponibilità

dispensa asw240

ottobre 2024

Anything that can go wrong, will go wrong.

Murphy's law

- Riferimenti

- Luca Cabibbo. **Architettura del Software**: Strutture e Qualità. Edizioni Efesto, 2021.
  - Capitolo 10, Disponibilità
- Scott, J. and Kazman, R. Realizing and Refining Architectural Tactics: Availability. Technical report CMU/SEI-2009-TR-006. 2009.
- Abbott, M.L. and Fisher, M.T. **The Art of Scalability**: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise, second edition. Addison-Wesley, 2015.
- Nygard, M. **Release It!** Design and Deploy Production-Ready Software, second edition. Pragmatic Bookshelf, 2018.
- Microsoft. Microsoft Application Architecture Guide: patterns & practices, second edition. Microsoft Press, 2009.

- Obiettivi e argomenti

- Obiettivi
  - presentare la qualità della disponibilità
  - illustrare alcune attività e tattiche per la progettazione per la disponibilità
- Argomenti
  - disponibilità
  - progettare per la disponibilità
  - discussione

* Disponibilità

- Intuitivamente, la disponibilità di un sistema software riguarda
  - la possibilità che, durante l'uso del sistema, si verifichino dei guasti negli elementi (hardware o software) del sistema
  - in corrispondenza, il modo con cui il sistema risponde a questi guasti
    - il sistema potrebbe continuare a funzionare, oppure si potrebbe verificare un fallimento

È legata al fatto che durante l'esecuzione di un sistema software si possono verificare guasti, guasti che sono di diversa natura, sia software (processo va in crash, errore durante l'esecuzione di un programma) che hardware (danneggiamento di qualche componente, va via la corrente). Un errore software è considerato un guasto. Ci sono diverse possibilità a seguito di un guasto: si ferma tutto, il guasto viene tollerato (l'utente non si accorge di nulla), il sistema si ferma e in breve tempo viene ripristinato.

Guasto: è interno
Fallimento: l'utente esterno del sistema se ne accorge
Non sempre un guasto si trasforma in un fallimento, mentre un fallimento significa che c'è stato (almeno) un guasto

Disponibilità: alcuni esempi recenti

- Alcuni recenti problemi di disponibilità
  - febbraio 2023

GERMANIA

Lufthansa, tutti i voli nazionali cancellati per un guasto al sistema informatico

di Redazione Online

Tutti i voli nazionali in Germania sono stati cancellati, ai passeggeri è stato chiesto di viaggiare in treno

  - agosto 2023

IN AGGIORNAMENTO

Gran Bretagna, centinaia voli in ritardo a causa di un guasto informatico: ripercussioni in tutta Europa

Ripercussioni sul traffico di tutta Europa. Migliaia di passeggeri coinvolti. L'agenzia di controllo dei cieli invita i passeggeri a contattare le compagnie per maggiori informazioni

Disponibilità: alcuni esempi recenti

- Alcuni recenti problemi di disponibilità
  - marzo 2024

! ULTIM'ORA

Instagram e Facebook down, problemi in tutta Europa alle piattaforme Meta

di redazione LOGIN

Dalle 16 sono iniziati i disservizi: non si aggiornano i feed di Instagram e moltissimi utenti sono stati scollegati dal proprio account di Facebook o non riescono ad utilizzare WhatsApp

SOCIAL

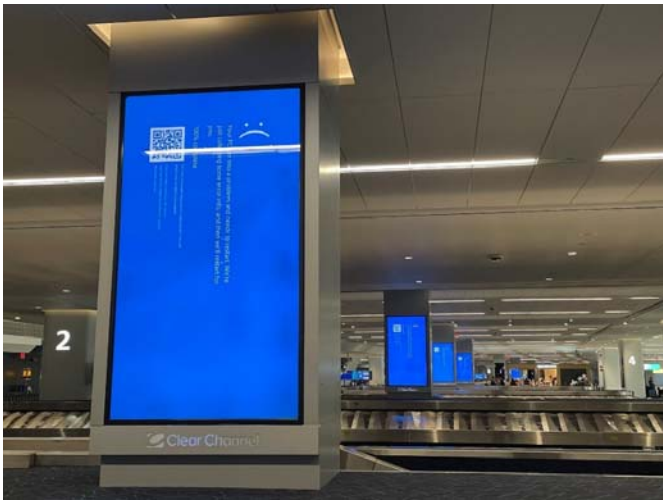
Instagram, Facebook e WhatsApp down: due ore di problemi in tutto il mondo per Meta

di redazione LOGIN

Dalle 16 alle 18, ora italiana, disservizi in tutto il mondo per centinaia di migliaia di utenti: i feed di Instagram non sono stati aggiornati e moltissimi utenti sono stati scollegati dal proprio account

Disponibilità: alcuni esempi recenti

- Alcuni recenti problemi di disponibilità
  - luglio 2024



  - l'incidente CrowdStrike – che qualcuno ha definito "il più grande incidente informatico della storia" – ha interessato circa 8.5 milioni di computer di compagnie aeree, aeroporti, banche, hotel, ospedali, mercati azionari e emittenti televisive – ad es., globalmente ci sono stati più di 5000 voli cancellati – con un danno finanziario complessivo di circa 5.4 miliardi di dollari

Disponibilità

- **Disponibilità** (*availability*)
  - la capacità di un sistema di essere completamente o parzialmente funzionante come e quando richiesto

Come richiesto vuol dire fa quello che si vuole, quando richiesto vuol dire che la disponibilità è anche strettamente legata al momento in cui voglio il servizio. Se richiedo il servizio nell'unico momento in cui non è funzionante, per me è comunque non disponibile

- **Tolleranza ai guasti**/**Resilienza** (*fault tolerance*/*resilience*)
  - la capacità di un sistema di operare come richiesto, anche a fronte di guasti di componenti hardware o software del sistema

Normalmente legato al fatto che alcuni componenti sono replicati, quindi quando si guastano delle copie di un servizio generalmente rimangono utilizzabili le altre copie

- **Capacità di recupero** (*recoverability*)
  - la capacità di recupero di un sistema dai fallimenti – ovvero, di recuperare i dati e rendere il sistema nuovamente operativo dopo un fallimento, entro tempi predefiniti e accettabili

Nota: non solo rendere il sistema di nuovo operativo ma anche recuperare i dati

Questi tempi dipendono dal dominio

- **Alta disponibilità** (*high availability*, *HA*)
  - la combinazione di disponibilità, tolleranza ai guasti e capacità di recupero

Noi ci riferiamo a questa disponibilità

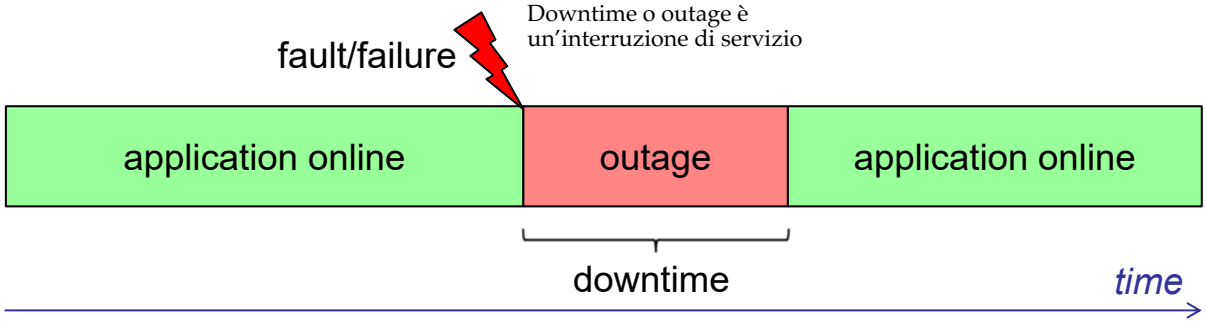
Disponibilità

- La disponibilità è la capacità di un sistema software di essere pronto a erogare i propri servizi, quando vengono richiesti dai suoi utenti
  - una qualità complessa, che riguarda il modo con cui il sistema affronta i *guasti* (*fault*) – per evitare o minimizzare i *fallimenti* (*failure*)
    - *guasto* – una problematica all'interno del sistema
    - *fallimento* – il sistema non eroga più, all'esterno, un servizio in modo coerente con la sua specifica
  - un sistema potrebbe essere *tollerante ai guasti* (*fault tolerance*) – oppure potrebbe essere in grado di effettuare un *ripristino* (*recovery*) a fronte di guasti
  - complessivamente, la disponibilità riguarda il tempo in cui il sistema è attivo e disponibile (appunto) a erogare i propri servizi

Interruzioni di servizio e RTO

Conseguenze di un guasto che diventa un fallimento

- Un *fallimento* è quando il sistema non eroga più un servizio in modo coerente con la sua specifica
  - una possibile conseguenza è un'*interruzione di servizio* (*service outage*)

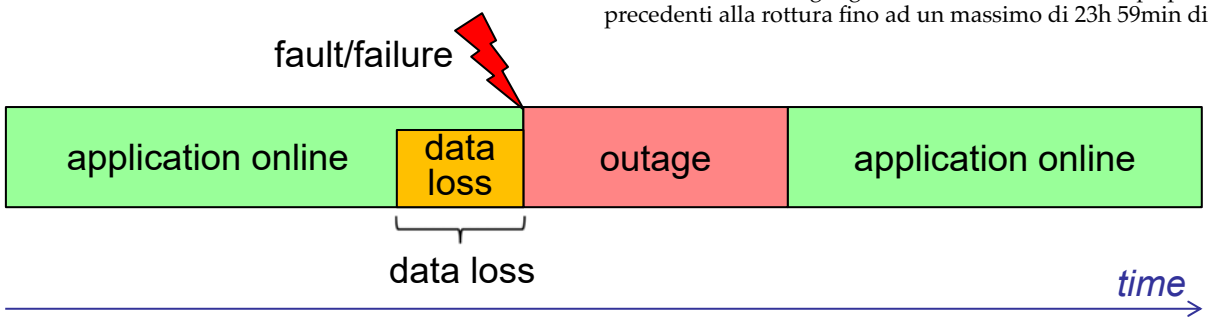


  - un possibile obiettivo di disponibilità di un sistema
    - limitare/minimizzare il tempo delle interruzioni di servizio a fronte dei possibili guasti (*Recovery Time Objective*, *RTO*)

Perdita di dati e RPO

- Un *fallimento* è quando il sistema non eroga più un servizio in modo coerente con la sua specifica
  - un'altra possibile conseguenza è una *perdita di dati* (*data loss*)

La perdita dei dati va "indietro nel tempo", pensa ad un backup automatico alle 12 ogni giorno, se l'hard disk si rompe perdi i dati precedenti alla rottura fino ad un massimo di 23h 59min di lavoro



  - un possibile obiettivo di disponibilità di un sistema
    - limitare/minimizzare la perdita di dati a fronte dei possibili guasti (*Recovery Point Objective*, *RPO*)

Che cosa è la disponibilità

- La *disponibilità* può essere definita anche come [SSA]
  - la capacità di un sistema di essere completamente o parzialmente funzionante, come e quando richiesto
  - anche a fronte di guasti di componenti del sistema
    - in modo che eventuali guasti non comportino un fallimento totale dell'intero sistema
    - oppure che comportino un fallimento dal quale è possibile un recupero – preferibilmente entro tempi concordati
    - in alcuni casi può essere accettabile un funzionamento "parziale" del sistema

Che cosa è la disponibilità

- Un'altra caratterizzazione della *disponibilità* [SAP]
  - la capacità di un sistema di mascherare o riparare guasti
  - in modo tale che la durata complessiva delle interruzioni di servizio non ecceda un valore richiesto nell'ambito di un certo intervallo di tempo specificato

Non dice nulla sulla perdita dei dati. Anche noi ci concentreremo più sul ripristino delle funzionalità

Disponibilità come misura

- La "disponibilità" come misura
  - la **disponibilità** (o *uptime*) è una misura della quantità di tempo in cui un sistema (o servizio o componente) è disponibile in un certo periodo di tempo – ovvero, è operativo ed è in grado di erogare i propri servizi

È una misura del passato, mentre nella progettazione di un sistema vogliamo guardare al futuro, progettare il sistema in modo che sia altamente disponibile

$$\text{disponibilità} = \frac{\text{periodo di tempo in cui il sistema è operativo}}{\text{periodo di riferimento}}$$

È al più 1, oppure espressa come percentuale, oppure come numero di nove che vuol dire 3 nove, 4 nove, 5… che sono rispettivamente 9,99, 9,999, 9,9999 ecc

- di solito espressa come un percentuale o come "numero di 9"
- attenzione, talvolta include i downtime pianificati, ma talvolta no

Sono periodi di tempo in cui il sistema è reso down per fare manutenzione

Disponibilità come misura

	Uptime (%)	Downtime (%)	Downtime per year	Downtime per week
Un 9	90%	10%	36.5 days	16:48 hours
Due 9	99%	1%	3.65 days	1:41 hours
Tre 9	99.9%	0.1%	8:45 hours	10:05 minutes
	99.99%	0.01%	52:30 minutes	1 minute
	99.999%	0.001%	5:15 minutes	6 seconds
	99.9999%	0.0001%	31.5 seconds	0.6 seconds

Disponibilità come misura

Sono interessata alla disponibilità come misura di probabilità, cioè mi interessa di più sapere quanto è probabile che il sistema sia disponibile in futuro di quanto non lo sia stato in passato perché sto facendo progettazione

- La disponibilità di un sistema (o componente) può essere stimata come la probabilità che esso fornisca i servizi specificati durante un certo periodo di tempo

$$\text{disponibilità} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

- *MTBF* – *Mean Time Between Failures* – è il tempo medio tra guasti

  È una misura che ci offre il fornitore del dispositivo (es un disco si guasta una volta ogni tre anni, nell'ottica in cui si stia parlando di guasti hardware. Parleremo poi dei guasti software)
- *MTTR* – *Mean/Maximum Time To Repair or Resolve* – è il tempo medio/massimo di riparazione/ripristino

  Dipende dal fatto che il pezzo da sostituire sia immediatamente disponibile o meno, quanto vicino sono l'impianto e la persona che risolve il guasto ecc.. Dipende da chi fa il ripristino
- come è possibile agire sulla disponibilità?

Posso avere alta disponibilità se l'MBTF è molto alto o se il MTTR è molto basso

Quando si parla di guasti software, l'MTFB diminuisce se faccio tanti/abbastanza test ben formulati (quindi per aumentare la disponibilità e quindi ridurre l'MTFB serve aumentare il testing). L'MTTR dipende dalla capacità di correggere l'errore.

Nota che il MTTR coinvolge anche il tempo che mi serve per rendermi conto del gusto, per accorgermene.

Affidabilità

- L'affidabilità è una qualità correlata alla disponibilità
  - l'*affidabilità* (*reliability*) è la probabilità che un sistema (o componente) fornisca i servizi specificati per tutto un certo periodo di tempo T
  - l'affidabilità è correlata al tempo medio tra guasti (MTBF)
    - per i più curiosi L'MTTR non compare

$$\text{affidabilità}(T) = e^{-\frac{T}{MTBF}}$$

  - l'affidabilità e la disponibilità sono due qualità correlate ma distinte

Affidabile non vuol dire che non si verifichino guasti ma che non si verifichino fallimenti

- Pianificare per la disponibilità

- La specifica dei requisiti di disponibilità di un sistema software è in genere molto complessa
  - il sistema può offrire molti servizi – che possono fallire in modo indipendente tra loro
  - i requisiti di disponibilità dei diversi servizi possono essere differenti tra di loro – e possono riguardare
    - una limitazione delle interruzioni di servizio
    - la possibilità di fallire in modo parziale – "classi di servizio" e modalità di erogazione "ridotta"
    - requisiti diversi in momenti differenti (Es il giorno e la notte)
    - la possibilità di downtime pianificati – per la manutenzione
    - una limitazione della perdita dei dati

Normalmente si diversificano i requisiti di disponibilità rispetto ai vari servizi

Pianificare per la disponibilità

- La specifica dei requisiti di disponibilità di un sistema software è in genere molto complessa
  - è necessario identificare e analizzare i possibili tipi di guasti
    - errori umani, errori nel software, guasti nell'hardware o nelle reti e disastri naturali
    - per ciascun tipo di guasto va valutata la probabilità, il possibile impatto sui servizi e la gravità
  - per ciascun tipo di guasto, va specificato l'impatto desiderato (per ottenerlo bisognerà prendere delle contromisure)

    Va definito come e quanto ci si vuole difendere rispetto a quali guasti
  - nei sistemi più critici può essere richiesta una soluzione di *disaster recovery*

Pianificare per la disponibilità

- La specifica dei requisiti di disponibilità di un sistema software è in genere molto complessa
  - le considerazioni fatte finora riguardano solo la *disponibilità tecnologica* di un sistema – che però è solo una componente della *continuità del business* – la capacità di un'organizzazione di continuare a esercitare il proprio business a fronte di guasti o altri eventi, anche catastrofici

    Disponibilità tecnologica interessa a noi, disponibilità di business interessa all'azienda/cliente
  - in pratica, la quantità di disponibilità richiesta per un sistema va determinata sulla base di un'opportuna *valutazione economica*
    - valuta il costo delle conseguenze delle possibili interruzioni di servizio – che può avere conseguenze dirette e indirette
    - sulla base di questa informazione, determina quanto sei disposto a spendere per proteggere il sistema da queste possibili interruzioni di servizi

      Dirette vuol dire che se in un'ora fatturo cento, in un'ora di failure perdo cento. Indirette vuol dire che i cento che ho perso dai clienti che in teoria avrebbero acquistato da me possono anche voler dire che quei clienti si sono spostati da altri venditori e quindi ho perso anche soldi su lungo termine

* Progettare per la disponibilità

- Alcune attività nella progettazione per la disponibilità [SSA]
  - identifica i guasti che possono verificarsi
  - per ciascun guasto o combinazione di guasti
    - valuta la gravità e l'impatto sui servizi
    - identifica un'opportuna strategia per la gestione di tale guasto
  - fai anche delle considerazioni complessive sulla disponibilità (sia tecnologica che di business)

Progettare per la disponibilità

- Ci concentriamo soprattutto sulle tattiche architetturali per aumentare la disponibilità di un servizio – nel senso di minimizzare le interruzioni di quel servizio – a fronte di alcuni possibili guasti del sistema
  - per impedire ai guasti di provocare fallimenti del servizio
  - oppure, per limitare l'effetto del fallimento e rendere possibile il ripristino del servizio

Migliorare la disponibilità vuol dire:
Minimizzare le interruzioni di servizio
Massimizzare la velocità di ripresa del servizio

- Tre categorie principali di tattiche
  - rilevare guasti
  - gestire il ripristino da guasti
  - prevenire guasti

- Disponibilità e ridondanza

La prima soluzione intuitiva è replicare tutti i componenti che possono guastarsi, per eliminare i punti di fallimento singolo, il cui fallimento comporta il fallimento di tutto il sistema e che non sono dotati di copie

- La progettazione per la disponibilità è di solito basata sulla *ridondanza* – ovvero, sulla presenza di più "copie" o "repliche" di uno o più elementi
  - per eliminare i *punti di fallimento singoli*
  - gli elementi ridondati possono essere elementi hardware oppure elementi software (componenti o dati)
    - Può essere considerata un'applicazione delle tattiche:
      - *Maintain multiple copies of computations*/*Maintain multiple copies of data*
  - è spesso necessario "ridondare" molti elementi
  - oltre agli elementi ridondati, sono spesso necessari anche degli ulteriori elementi o delle responsabilità aggiuntive
- Qui consideriamo soprattutto la ridondanza di nodi (computer usati come server) che implementano servizi
  - i nodi vengono organizzati in cluster e in "gruppi di protezione"

Il software è soggetto ad errori di programmazione, quindi se ho due copie dello stesso pezzo di codice l'errore che ho in una copia sarà presente anche nell'altra copia, quindi la copia "cieca" può non essere la soluzione migliore. Per questo in alcuni casi la diversificazione delle copie può essere una strategia migliore. In casi estremi si suggerisce una ridondanza tripla, di cui si valutano i risultati ricercando l'omogeneità dei risultati

Si può anche avere ridondanza dei dati, e si possono avere più copie dei dati o anche una copia dei dati e una copia di "come si è arrivati a quei dati", ad esempio una lista di transazioni che hanno portato ad avere quel database

Ridondanza e affidabilità

- Un po' di matematica
  - supponiamo che un elemento E abbia un'affidabilità R – ovvero, ha probabilità 1-R di guastarsi nell'unità di tempo
  - consideriamo poi un sistema composto da N repliche di E (al posto del singolo elemento E), nelle seguenti ipotesi
    - il sistema è disponibile se almeno una delle repliche di E lo è
    - i guasti tra le repliche di E sono indipendenti tra loro
  - sotto tali ipotesi, il sistema si guasta (non è disponibile) se tutte le repliche di E si guastano contemporaneamente
    - questo avviene con una probabilità $(1-R)^N$ e l'affidabilità complessiva del sistema è dunque $1-(1-R)^N$
  - ad es., se l'affidabilità di un singolo elemento non replicato è R=90% (0.9), allora l'affidabilità di un sistema con 2 repliche è 99% (0.99)

Cluster

Il sistema complessivo sarà organizzato in un cluster, in un insieme di nodi che offre tanti servizi:
Primario: viene usato primariamente per erogare un certo servizio
Secondario: potrebbe non essere usato attivamente per erogare quello stesso servizio.
Nota che in due servizi diversi i ruoli dei nodi potrebbero essere diversi/invertiti

- Un cluster è un gruppo di più nodi interconnessi, che lavorano assieme per offrire un insieme di servizi come un sistema singolo
  - ad es., un cluster potrebbe offrire due servizi A e B, mediante due nodi X e Y, in cui
    - X è un nodo "primario/attivo" per il servizio A e "secondario/di riserva" per il servizio B
    - Y è un nodo "primario/attivo" per il servizio B e "secondario/di riserva" per il servizio A
  - i cluster hanno in genere configurazioni complesse
  - è più semplice ragionare inizialmente in termini di "gruppi di protezione"

Un gruppo di protezione per un servizio S è l'insieme dei nodi utilizzati per erogare e proteggere un certo servizio S, quindi sia primari che secondari

- Gruppi di protezione

- Un **gruppo di protezione** per un servizio S è un insieme di nodi dedicati all'erogazione del servizio S
  - in un gruppo di protezione, in un dato momento
    - uno o più nodi sono *attivi* – ovvero, sono utilizzati per erogare effettivamente il servizio ai client del servizio
    - ci possono essere anche delle *riserve* ridondanti
    - di solito servono anche altri elementi oltre a questi nodi
  - il caso più semplice di gruppo di protezione è la ridondanza 1+1
  - attenzione, le riserve per un servizio S possono essere utilizzate in modi diversi (anche in altri gruppi di protezione)

Ripristino dai guasti

Quanto tempo passa per il ripristino di un servizio?

- In un gruppo di protezione, il ripristino di un nodo attivo (a seguito di un suo guasto) viene gestito in genere come segue
  - rilevamento del guasto del nodo – automaticamente o mediante un intervento manuale
  - preparazione di un nodo di riserva per sostituire il nodo che si è guastato
  - attivazione di questo nodo – l'elemento di bilanciamento del carico gli inizia ad assegnare richieste
  - il tempo richiesto per eseguire queste attività ha influenza sulla disponibilità del servizio – ma anche sul costo della soluzione

Tempi più variabili

Potrei non avere proprio il nodo sostitutivo, potrei averlo ma non già predisposto, potrei averlo già collegato. Poi devo prepararlo nel senso di renderlo attivo e col software aggiornato, e poi trasferirvi i dati per renderlo operativo in modo da sostituire l'altro

È normalmente il tempo più breve

- Tattiche per la disponibilità

- Categorie principali di tattiche per la disponibilità di [SAP]
  - *detect faults*
    - hanno lo scopo di rilevare guasti
  - *recover from faults*
    - hanno a che fare con il ripristino del sistema a fronte di guasti
  - *prevent faults*
    - hanno lo scopo di prevenire guasti
  - *exceptions* – una categoria "trasversale", che riguarda la gestione dei guasti nel software

Osservazioni

Di solito si tratta di capire quali tattiche voglio implementare e COMPRARE servizi software che mi offrano servizi per impiegare le tattiche che desidero

- Queste tattiche per la disponibilità
  - fanno riferimento soprattutto alla disponibilità dei nodi – ma spesso possono essere applicate anche ad altri elementi
  - sono di solito presenti in molti ambienti di esecuzione standard – come sistemi operativi, infrastrutture software e middleware
    - è importante comprendere tali tattiche e il loro effetto
  - il lavoro dell'architetto sarà poi spesso quello di scegliere e valutare (piuttosto che implementare) le tattiche per la disponibilità da applicare nella progettazione di un sistema

- Detect faults

Si parla di monitoraggio "minimale", monitoraggio di nodi, non di funzionalità specifiche

- Prima di poter intraprendere qualunque azione che riguardi un guasto (come un ripristino automatico), la presenza del guasto deve essere rilevata
- Una tattica generale per il rilevamento automatico dei guasti
- ④ *Monitor*

In un gruppo di protezione gli elementi possono monitorarsi a vicenda oppure può esserci un elemento aggiuntivo che fa solo da monitor

  - un elemento utilizzato per effettuare il monitoraggio dello stato di salute delle altre parti di un gruppo di protezione o di un sistema
  - il monitoraggio può essere effettuato mutuamente dai nodi stessi che appartengono al gruppo di protezione – oppure da un elemento specializzato

Stiamo parlando di monitor impiegati per i nodi ma si possono anche utilizzare per dischi, processi, ecc…

Detect faults

- Due ulteriori tattiche principali per il monitoraggio Sono due varianti del monitor

2 - *Ping/echo*
  - il monitor invia dei messaggi *ping* al monitorato, che risponde con dei messaggi *echo*
  - lo scambio di messaggi, di solito asincrono, consente di determinare la raggiungibilità tra elementi e il tempo di roundtrip del canale di comunicazione

3 - *Heartbeat*
  - l'elemento monitorato emette periodicamente un messaggio *heartbeat* – mentre il monitor è in ascolto Qui il monitor è passivo

- Come distinguere tra il guasto di un nodo e un guasto della rete?

Normalmente la rete è replicata, in modo da poter escludere che il mancato arrivo in un echo/heartbeat sia da imputare ad un guasto della rete

Detect faults

- Esistono anche altre tattiche per il rilevamento dei guasti, per valutare in modo più preciso lo stato di salute del sistema – ad esempio
  - per rilevare tentativi di frodi o attacchi DoS (Denial of Service)
  - usare nella comunicazione checksum, timestamp o timeout
  - verificare la "ragionevolezza" dei messaggi (richieste e risposte) scambiati tra gli elementi
  - richiedere l'esecuzione di una stessa operazione a più elementi software distinti – e confrontare le loro risposte (voting)
    - se gli elementi software sono identici, una risposta differente dalle altre consente di rilevare un problema nell'hardware
    - elementi software sviluppati indipendentemente possono consentire di rilevare errori nel software

- Recover from faults

Una volta rilevato il guasto vogliamo effettuare il ripristino. Ci sono due sottocategorie, noi vedremo soprattutto la prima

- Due categorie di tattiche per il ripristino da guasti
  - tattiche che preparano ed eseguono il ripristino di un nodo che si è guastato (tattiche per il *failover*)
  - tattiche relative alla reintroduzione di un nodo che è stato "riabilitato" dopo che si era guastato

Recover from faults

- Nel ripristino da guasti, un nodo di riserva può subentrare a un nodo che si è guastato solo dopo che
  1) è stato rilevato il guasto di un nodo
  2) il nodo di riserva è stato "preparato" per sostituire questo nodo
     - acquisizione e configurazione del nodo
     - installazione e configurazione del software sul nodo, e suo avvio
     - sincronizzazione dello stato del nodo No se è stateless

     Alcuni di questi passaggi potremmo averli già fatti
  3) il nodo preparato è stato reso attivo
  - come si può aumentare la disponibilità? ma a quale costo?

Recover from faults

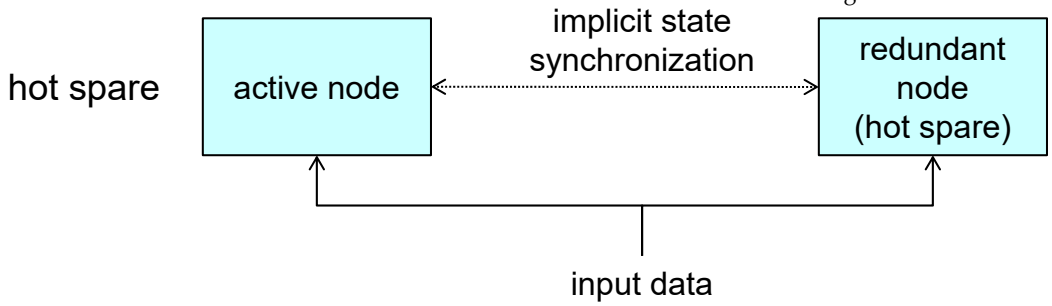
- Tre tattiche principali per il ripristino da guasti (per preparare ed eseguire il ripristino di un nodo che si è guastato)
  - *hot spare*
  - *warm spare*
  - *cold spare*
  - queste tattiche sono caratterizzati da tre livelli differenti di disponibilità (e di costo)

Recover from faults

Il nodo di riserva è acceso ed elabora tutte le richieste degli utenti; non è considerato attivo perché le risposte alle richieste degli utenti vengono dal nodo primario, tuttavia lo stato dei due nodi è identico perché hanno eseguito le stesse cose. È il metodo più costoso perché il nodo secondario è impegnato nelle elaborazioni che lo mantengono sincronizzato col nodo primario

④

- *Active redundancy (hot spare)*

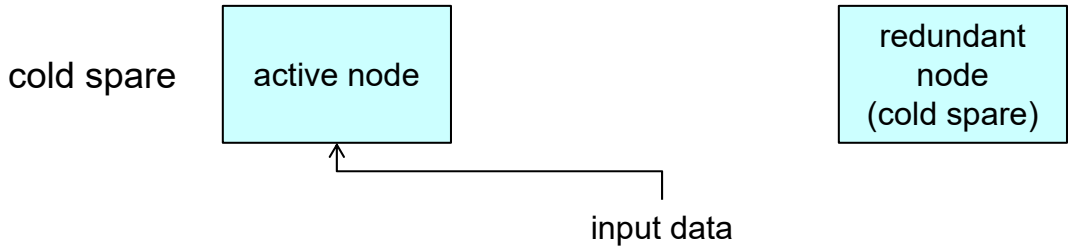


  - tutti i nodi (attivi e di riserva) in un gruppo di protezione sono preparati e avviati, ed elaborano tutti gli eventi di input – sono sincronizzati in modo continuo
  - quando si guasta un nodo attivo, un nodo di riserva lo può sostituire immediatamente
    - il downtime potrebbe essere nell'ordine dei millisecondi
  - la ridondanza può essere anche nel canale di comunicazione (rete) e nelle unità disco (condivise)

Recover from faults

② ❑ *Spare (cold spare)*

Estremo opposto, prevede che il nodo di riserva sia completamente staccato dalla computazione svolta dal sistema e solo quando si rileva il guasto sul nodo attivo si iniziano le attività per iniziare ad usare il nodo secondario. Normalmente è necessario un ripristino dei dati prima della sincronizzazione, per questo il downtime potrebbe essere lungo, tuttavia questa soluzione potrebbe essere preferibile se il servizio è stateless.

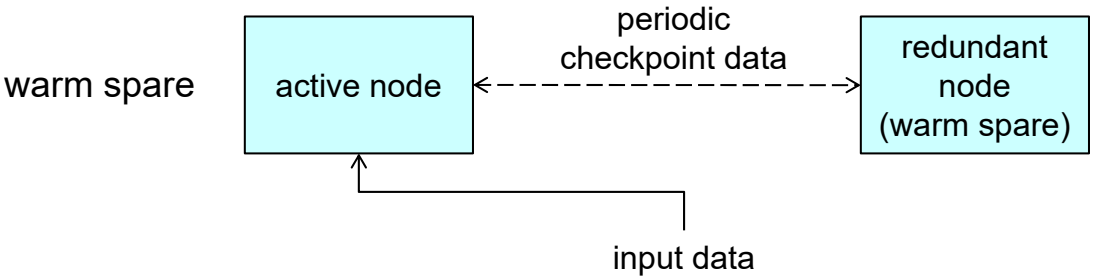


- i nodi di riserva di un gruppo di protezione sono tenuti fuori servizio fino a quando non è necessario sostituire un nodo
  - i nodi di riserva sono spesso solo preparati (configurati) per poter sostituire altri nodi – ma non sincronizzati
- quando si guasta un nodo attivo, un nodo di riserva viene configurato e avviato, il suo stato viene sincronizzato (vedi *Rollback*) – poi avviene la sostituzione
  - il downtime potrebbe essere nell'ordine dei minuti o delle ore
  - ma può essere anche più lungo o più breve – in quali casi?

Recover from faults

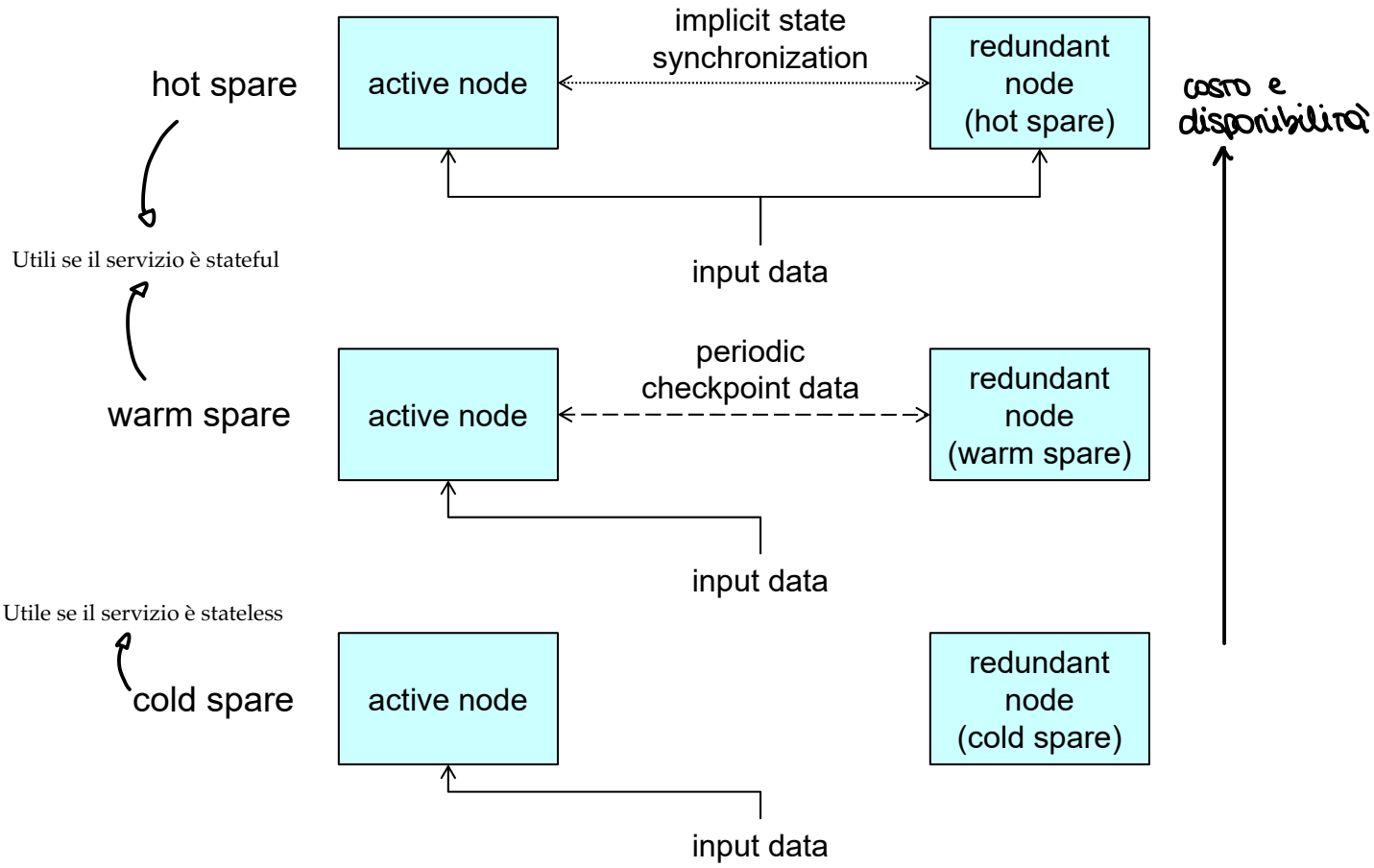
③ ❑ *Passive redundancy (warm spare)*

Tecnica intermedia, prevede che il nodo attivo gestisca tutti gli input, il nodo di riserva sia preparato e in esecuzione senza però ricevere ed elaborare gli input. Periodicamente il nodo primario comunica al nodo secondario il proprio stato. È necessaria una sincronizzazione prima che il nodo secondario prenda il posto del primario, ma anche qui se il servizio è stateless questo non è necessario



- solo i nodi attivi del gruppo di protezione ricevono ed elaborano gli eventi di input – i nodi di riserva vengono sincronizzati in modo parziale e asincrono (ad es., periodico)
- quando si guasta un nodo attivo, un nodo di riserva lo può sostituire, ma solo dopo aver completato la sincronizzazione del suo stato
- è un compromesso (affidabilità/complessità) tra le due tattiche precedenti – il downtime può essere nell'ordine dei secondi

Hot spare, warm spare, cold spare



Recover from faults e stato del servizio

- Lo stato del servizio di interesse per un gruppo di protezione ha un impatto rilevante nella modalità di ripristino dai guasti
  - ad es., se il servizio è stateless, allora non è necessaria nessuna sincronizzazione tra i nodi
  - se il servizio è stateful e lo stato del servizio è ripartito (ad es., in "shard") tra i nodi attivi del gruppo di protezione (anziché essere replicato interamente su tutti i nodi attivi), allora la sincronizzazione va gestita "ad hoc"

Recover from faults e clustering

- Le tattiche per il ripristino da guasti possono essere implementate mediante clustering e failover (discussi più avanti)
  - un cluster può essere considerato un pattern architetturale, basato sull'applicazione di un insieme "complesso" di scelte di progettazione
    - in un cluster è sempre possibile riconoscere l'applicazione di più tattiche per la disponibilità
  - ciascuna tattica corrisponde invece a un'opzione di progettazione "elementare"
  - alcune configurazioni di cluster sono descritte brevemente più avanti

Recover from faults

- Altre tattiche per il ripristino da guasti
  - *Rollback* Periodicamente salvo lo stato, continuamente salvo le transazioni
    - questa tattica ha lo scopo di sostenere il ripristino dello stato del sistema – in combinazione con altre tattiche per la disponibilità
    - ad es., mediante *checkpoint* (dello stato) e *log* (delle transazioni)
  - *Software upgrade* Quando devo aggiornare il software, questo potrebbe provocare un'interruzione di servizio. Per minimizzare queste interruzioni di servizio posso aggiornare prima il nodo principale, usando attivamente quello secondario, e poi quello secondario riportando l'esecuzione sul nodo primario
    - il tempo per effettuare l'aggiornamento di un servizio software può corrispondere a un'interruzione di servizio
    - questa tattica ha l'obiettivo minimizzare o anche evitare le interruzioni di servizio relative agli aggiornamenti del software – anche se non si tratta di guasti

Recover from faults

- Altre tattiche per il ripristino da guasti

6
- in caso di guasti, prova a mantenere la disponibilità delle funzioni del sistema più importanti e critiche, arrestando invece le funzioni meno importanti (*Degradation*) Quindi disponibilità parziale, degrado il servizio ma senza interromperlo

7
- il ripristino cerca di riassegnare responsabilità agli elementi sopravvissuti ai guasti, sempre cercando di mantenere attive le funzioni più importanti (*Reconfiguration*)

8
- prova ad assumere che il guasto sia spurio o transitorio – ignorando il guasto (*Ignore faulty behaviour*), oppure provando a ripetere l'esecuzione dell'operazione in cui si è verificato il guasto (*Retry*)

Recover from faults

- Alcune tattiche relative alla reintroduzione di componenti che sono stati (auspicabilmente) "riparati" – dopo che si erano guastati

9
- *Shadow*
  - esegui temporaneamente il nodo da reintrodurre in "modalità ombra" (come riserva) – per verificare se il guasto è stato effettivamente riparato

10
- *State resynchronization*
  - verifica la correttezza dello stato dell'elemento da reintrodurre – oppure da ripristinare – prima di reintrodurlo

11
- *Escalating restart*
  - consenti di riavviare i servizi in modo granulare – ad es., per evitare di dover riavviare servizi su cui un guasto non ha avuto impatto

- Prevent faults

- Le tattiche in questa categoria hanno l'obiettivo di favorire la prevenzione dei guasti – per evitare che si verifichino e di doverli rilevare e gestire

① *Removal from service*
  - rimuovere temporaneamente un componente del sistema per fargli svolgere attività per mitigare possibili fallimenti di sistema (Es. Un nodo viene soggetto ad antivirus o riavvio)

② *Transactions*
  - una transazione è una sequenza di passi elementari che viene complessivamente considerata un'operazione atomica – da svolgere oppure annullare completamente
  - le transazioni consentono di prevenire inconsistenze nello stato del sistema, nel caso di fallimento di uno dei loro passi elementari

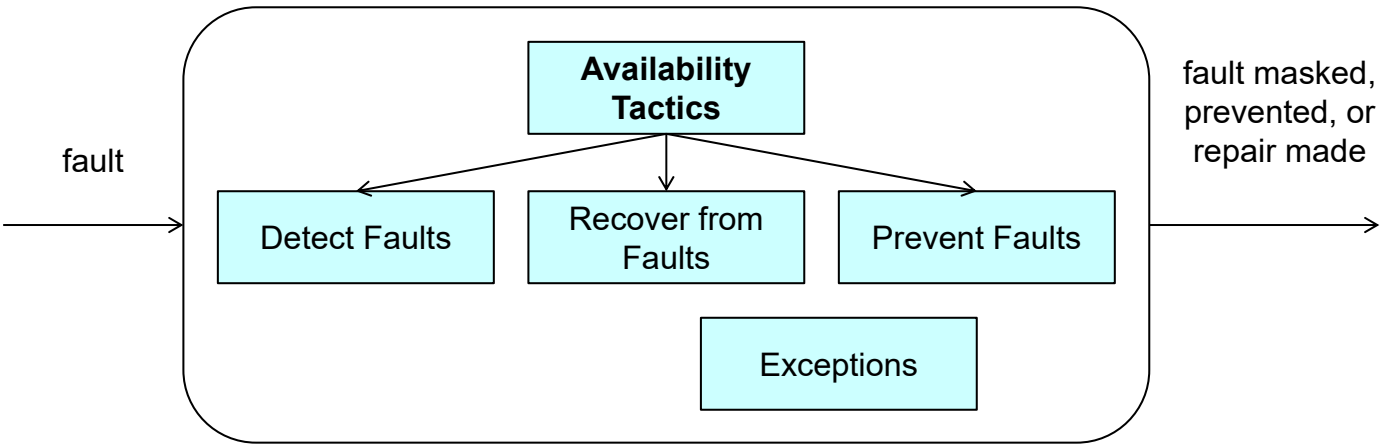
- Exceptions

Strumento usato nel codice per gestire rilevamento di guasti e gestione dei guasti

- Le tattiche per le *eccezioni* riguardano l'applicazione dello strumento linguistico delle eccezioni alle tre precedenti categorie di tattiche per la disponibilità – per gestire guasti nel software

- *Exception detection* – per detect faults
  - il rilevamento delle eccezioni ha lo scopo di identificare le condizioni che devono alterare il normale flusso di esecuzione
- *Exception handling* – per recover from faults
  - la gestione di eccezioni può essere usata per il ripristino da guasti – notificati appunto come eccezioni
- *Increase competence set* – per prevent faults
  - la competenza di un programma è l'insieme di stati in cui il programma ha competenza per operare
- *Exception prevention* – per prevent faults
  - ha lo scopo di prevenire il verificarsi di eccezioni

- Tattiche per la disponibilità



Mini recap a inizio lezione:
La disponibilità è una qualità del sistema, la capacità di un sistema di gestire guasti e fallimenti. I guasti sono problematiche interne del sistema, i fallimenti sono una manifestazione all'esterno del sistema del malfunzionamento (di uno o più guasti) del sistema stesso.
Un sistema può essere tollerante ai guasti, le conseguenze di un guasto possono essere interruzione di servizio e/o perdita di dati, per cui si progetta per migliorare la disponibilità.
La disponibilità è una qualità complessa.
Abbiamo poi parlato delle tattiche per la disponibilità: l'idea fondamentale è quella di replicare i componenti hardware e software. Ci siamo concentrati in particolare sulla duplicazione dei nodi e sulle tecniche di recover from fault.

DOMANDA: UN NODO È SOFTWARE O HARDWARE ?

- Discussione

Si possono usare varie tattiche, in diversi nodi e per diversi servizi

- Riguardo alle tattiche per la disponibilità di [SAP]
  - l'applicazione di una tattica può richiedere anche l'applicazione di altre tattiche – in altre categorie
  - l'applicazione di una tattica non impedisce l'applicazione di altre tattiche – anche nella stessa categoria
  - l'applicazione di una tattica per la disponibilità può essere efficace a fronte di un certo tipo di guasto – ma non lo è necessariamente con tutti i possibili guasti (allo stesso modo)
  - molte configurazioni richiedono delle considerazioni speciali

- Fault isolation

- L'*isolamento dei guasti* (*fault isolation*) [Abbott and Fisher] è un altro importante principio di progettazione per la disponibilità
  - i fallimenti a cascata e le reazioni a catena sono tra i principali problemi alla stabilità e alla disponibilità dei sistemi software [Nygard]
    - la causa sono spesso elementi troppo accoppiati e integrati tra loro anziché isolati
  - l'isolamento dei guasti ha lo scopo di impedire la *propagazione dei guasti* – da un servizio ad altri servizi o all'intero sistema
    - per evitare fallimenti a cascata, e per fare in modo che il sistema sia parzialmente funzionante a fronte di guasti
    - ci sono diversi modi di realizzare l'isolamento dei guasti

Non voglio che un servizio importante sia mandato in fallimento dal fallimento di un servizio meno importante

Supponiamo che ci sia un componente A che chiama un componente B che chiama un componente C. Se C va in fault, si guasta, non risponde. Allora se B lo chiama non riesce a risolvere la richiesta, e a catena anche A se chiama B. Così i guasti rischiano di propagarsi.
Esempio se B è dotato di un tempo di timeout e chiama C e C non risponde, ad un certo punto deciderà di procedere in un qualche modo (segnala che c'è stato un problema, fornisce un risultato alternativo che tiene conto del fault di B) in modo da non andare anch'esso in fault.

Supponiamo che ci sia un sistema che deve gestire degli ordini. L'ordine è confermato quando viene confermato il pagamento. C'è un servizio di gestione degli ordini e un servizio di gestione dei pagamenti. Quando arriva l'ordine il servizio di gestione dell'ordine chiede di verificare il pagamento. Se c'è un guasto nel servizio di gestione dei pagamenti e questo non risponde, non risponderà neanche il servizio di gestione degli ordini e quindi il cliente non sa se il suo ordine è confermato. Tuttavia è possibile che il sistema di gestione degli ordini dica al cliente "il tuo ordine è stato registrato. Non è stato possibile verificare il pagamento quindi il tuo ordine non è confermato ma potrai verificarlo quando il servizio sarà ripreso"

Fault isolation e affidabilità

- Un po' di matematica
  - consideriamo un componente A_1 che dipende (direttamente o indirettamente) da altri componente A_2 … A_N
  - ognuno di questi componenti ha un'affidabilità (in isolamento) R – ovvero, ha probabilità 1-R di guastarsi nell'unità di tempo
  - per fruire di un servizio di A_1, tutti i componenti A_1 … A_N devono essere contemporaneamente attivi – ovvero, non sono isolati
  - qual è l'affidabilità combinata del componente A_1?
    - è R^N – ad es., se R=99% e N=10, allora $99\%^{10}=90.4\%$
    - ma se, ad es., R=99% e N=20, allora $99\%^{20}=81.8\%$
    - in assenza di isolamento dei guasti, l'affidabilità combinata diminuisce in modo esponenziale rispetto a N
  - l'isolamento dei guasti cerca invece di impedire questa propagazione dei guasti

Fault isolation

- L'isolamento dei guasti è utile soprattutto
  - per isolare un servizio di alto valore di business da altri servizi meno importanti
  - per isolare servizi particolarmente soggetti a guasti

Fault isolation

L'intuizione è quella di partizionare il sistema in zone. Ad esempio in AWS ci sono AZs, availability zones, ovvero zone che sono indipendenti/separate da altre AZs.

- Progettare per l'isolamento dei guasti
  - identifica nel sistema le zone da isolare rispetto ai guasti – chiamate swim lane, pod, shard o bulkhead
  - linee guida
    - zone distinte non devono comunicare in modo sincrono (che può causare la propagazione dei guasti)
    - zone distinte non devono condividere niente – ad es., una medesima base di dati
  - ecco alcune possibili soluzioni (relative alla comunicazione)
    - comunica in modo asincrono
    - usa timeout stringenti
    - ripeti la comunicazione (utile solo se i guasti sono transienti)
    - fallback – comunica con un componente diverso
    - usa un circuit breaker – combina più tecniche

Se il guasto non è transiente (non passa) questo può diventare invece problematico perché le richieste si accumulano. Ci sono altri approcci per cui non si ripete mai la comunicazione con lo stesso componente, si decide di considerarlo sempre guasto e di non considerare l'opzione di un guasto transiente

Fault isolation

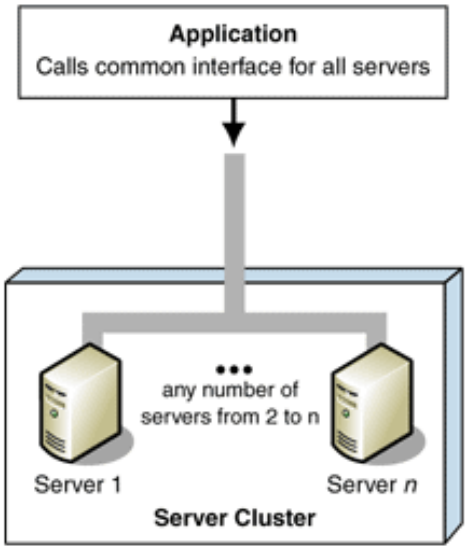
- *Circuit Breaker* [Nygard] è un pattern per l'isolamento dei guasti
  - un "interruttore automatico" o "salvavita"
  - è un intermediario che incapsula la chiamata a un servizio remoto (che potrebbe non essere disponibile)
    - si può trovare, in due stati: "chiuso" o "aperto"
    - se il circuito è "chiuso", allora il circuit breaker chiama (prova a chiamare) il servizio remoto
    - se il circuit breaker rileva dei problemi nella comunicazione, allora (dopo un certo numero di errori) "apre" automaticamente il circuito, in modo che le chiamate successive non provino nemmeno a raggiungere il servizio remoto – il circuit breaker può anche chiamare un servizio alternativo (fallback)
    - dopo un po', il circuito prova a richiudersi automaticamente

- Cluster

- Un **cluster** è un gruppo di due o più nodi interconnessi – chiamati *nodi* o *membri* del cluster – che lavorano assieme per offrire uno o più servizi come un sistema singolo
  - un cluster consente di implementare i gruppi di protezione per uno o più servizi I gruppi di protezione non si sommano e basta, di solito si usa il nodo primario per un servizio, il nodo secondario per un altro servizio ecc, in modo da ridurre il numero di nodi
  - per "sistema singolo" si intende che il gruppo dei nodi interconnessi che forma il cluster viene visto dall'utente/client di un servizio come una singola entità
  - un cluster, oltre ai nodi (server), comprende anche altri elementi, hardware (fisici o virtuali) e software
    - ad es., la rete, un servizio di appartenenza al cluster, un load balancer
  - due motivazioni principali per l'uso dei cluster (anzi tre) – disponibilità e scalabilità E la possibilità di perseguire queste due qualità in modo simultaneo
  - qui consideriamo i cluster per l'alta disponibilità

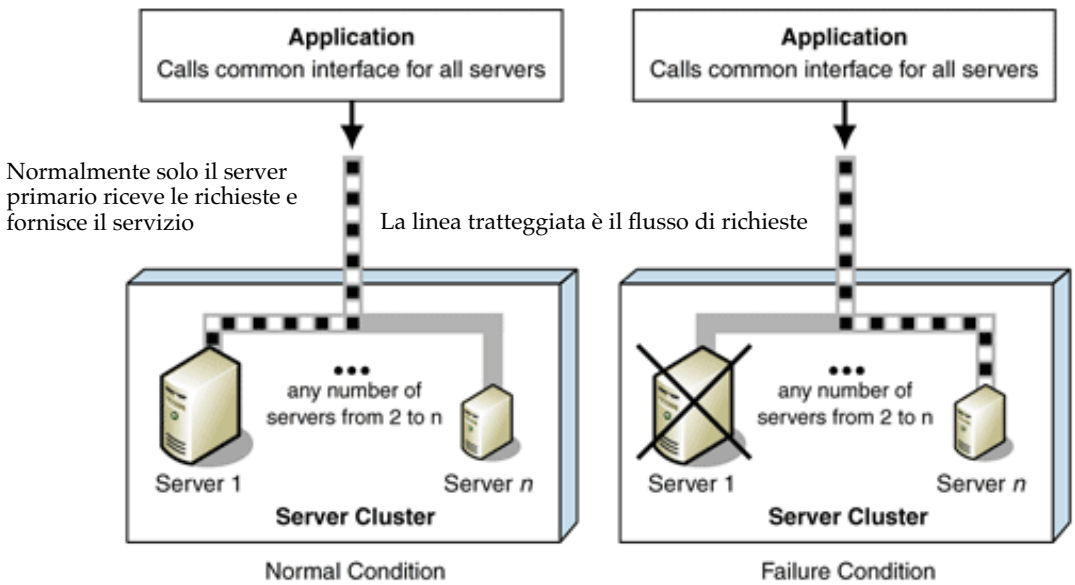
Configurazioni

- Esistono diverse configurazioni comuni per i cluster
  - la configurazione di base è un gruppo di due o più nodi che lavorano insieme per offrire un servizio come un sistema singolo



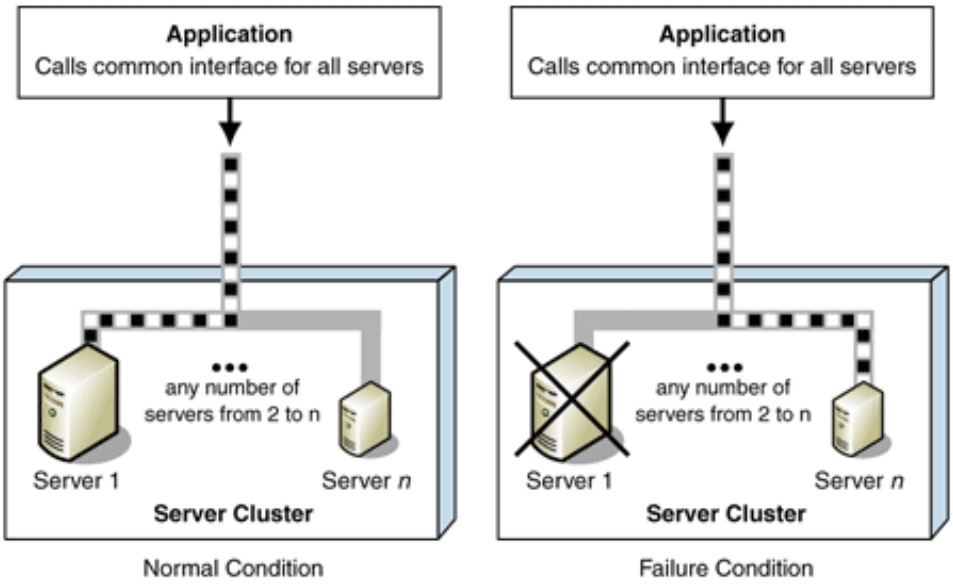
Cluster asimmetrico

- *Cluster asimmetrico*
  - un servizio viene erogato solo dal nodo attivo (*server primario*)
  - il *server secondario* (*standby server*) sostituisce il server primario nell'erogazione del servizio solo nel caso di un suo fallimento



Failover

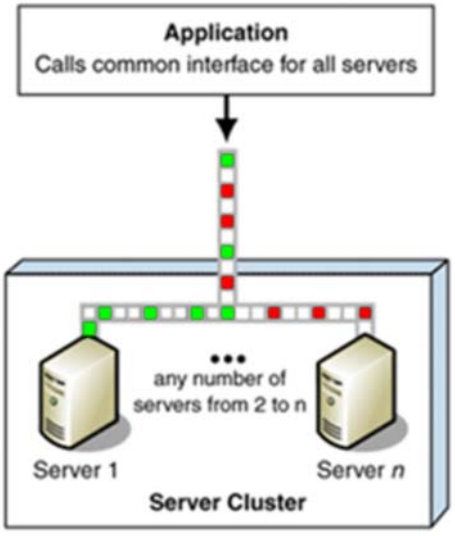
- *Failover*
  - è la sostituzione del server primario con il server secondario



Cluster simmetrico

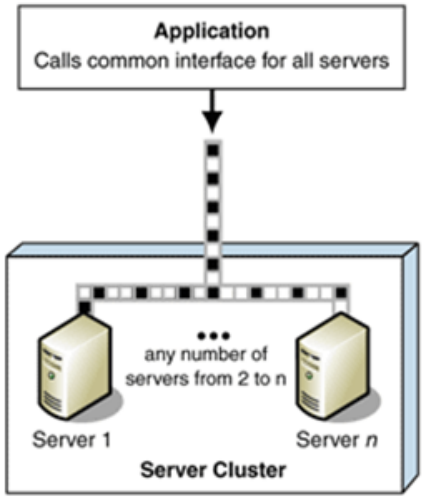
- *Cluster simmetrico*
  - utile quando un sistema che deve erogare più servizi
  - ciascun nodo del cluster svolge del lavoro utile, erogando uno o più servizi
  - un nodo può essere il server primario per uno o più servizi – ma anche il server secondario (standby server) per altri servizi
  - in caso di fallimento di un nodo, ciascuno dei suoi "servizi primari" viene riassegnato a un server secondario per quel servizio

Fa sì che ogni nodo sia primario rispetto a servizi diversi. Nell'esempio sotto, se il server di sinistra si guasta quello di destra potrebbe diventare primario anche per il servizio dell'altro. Questo potrebbe diventare problematico a livello di carico



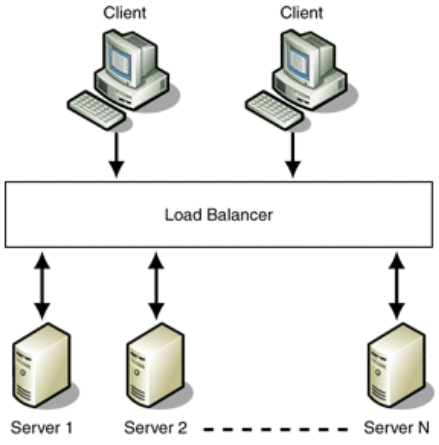
Cluster e load balancing

- *Cluster per load balancing*
  - un cluster di tipo simmetrico, in cui un servizio viene erogato attivamente da più nodi – al limite, tutti i nodi del cluster partecipano attivamente all'erogazione del servizio
  - è necessario un meccanismo aggiuntivo di load balancing
  - può essere necessario anche un meccanismo di sincronizzazione tra i nodi del cluster



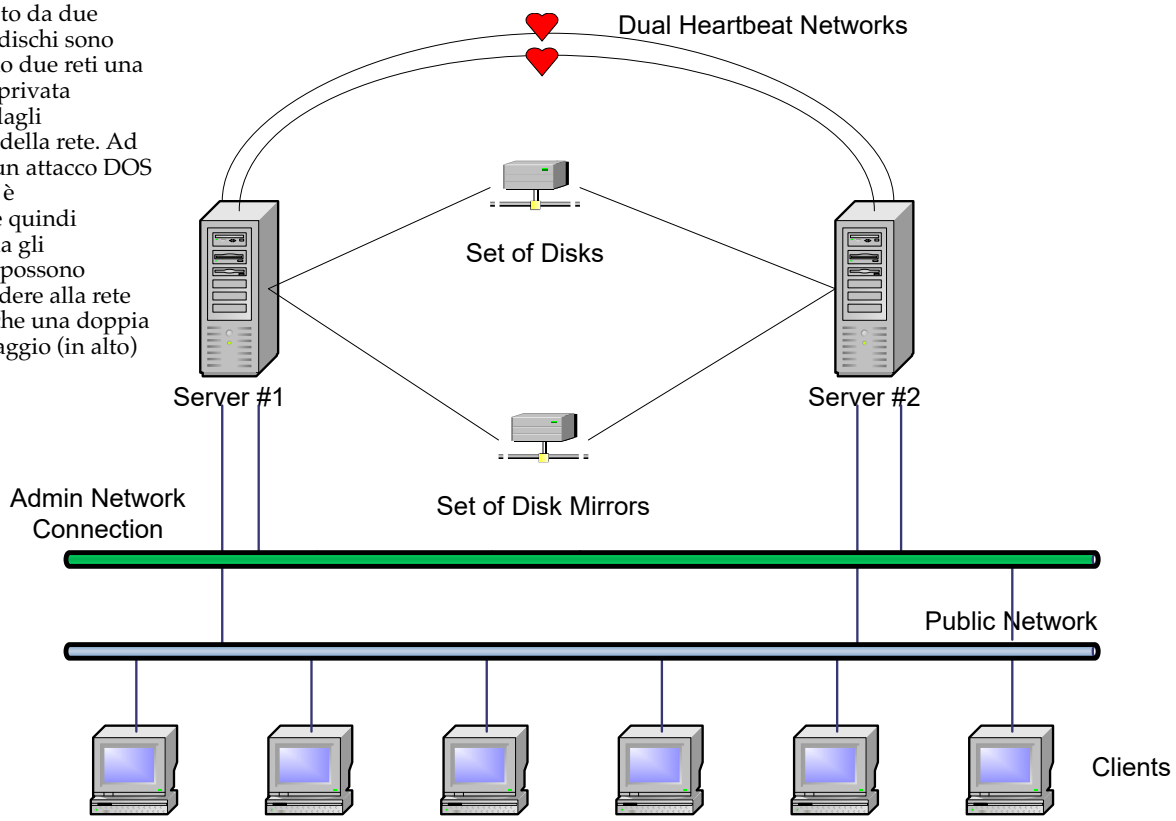
Cluster e load balancing

- *Load balancing*
  - le richieste per un servizio vengono ricevute da un load balancer – che le gira ai nodi del cluster che erogano quel servizio sulla base di un'opportuna politica
  - sostiene la scalabilità orizzontale

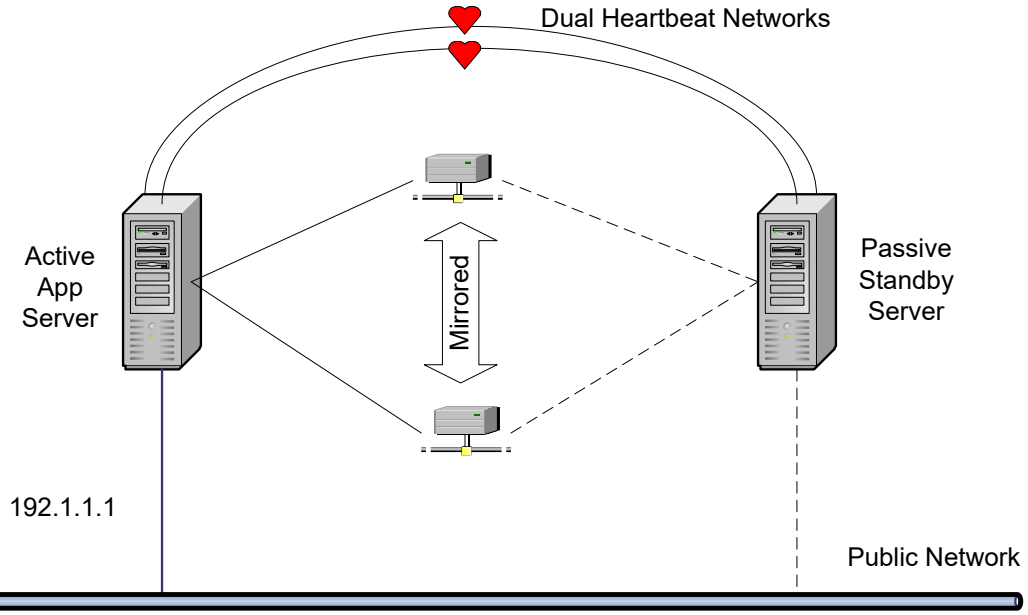


- Esempio - un semplice cluster

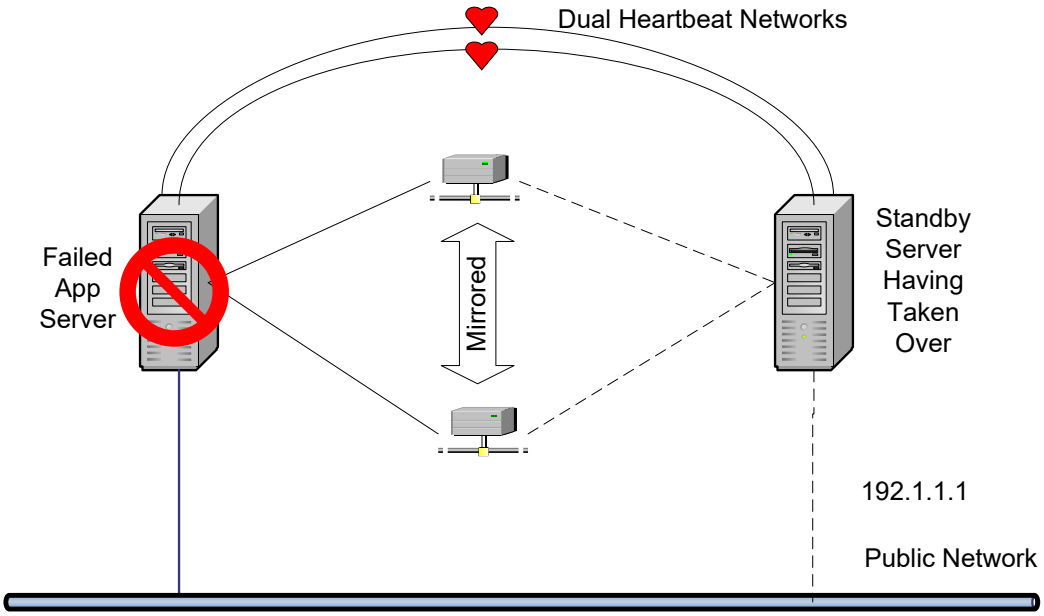
Cluster composto da due server. Anche i dischi sono replicati. Ci sono due reti una pubblica e una privata utilizzata solo dagli amministratori della rete. Ad esempio se c'è un attacco DOS la rete pubblica è sovraccaricata e quindi inutilizzabile ma gli amministratori possono comunque accedere alla rete privata. C'è anche una doppia rete di monitoraggio (in alto)



Cluster asimmetrico – prima di un failove

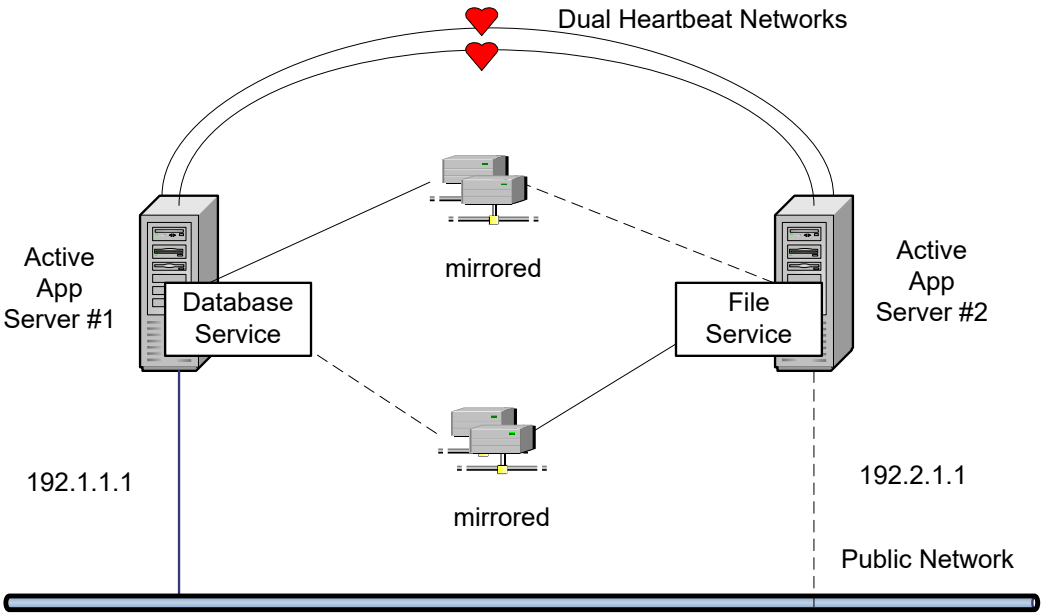


Cluster asimmetrico – dopo un failover

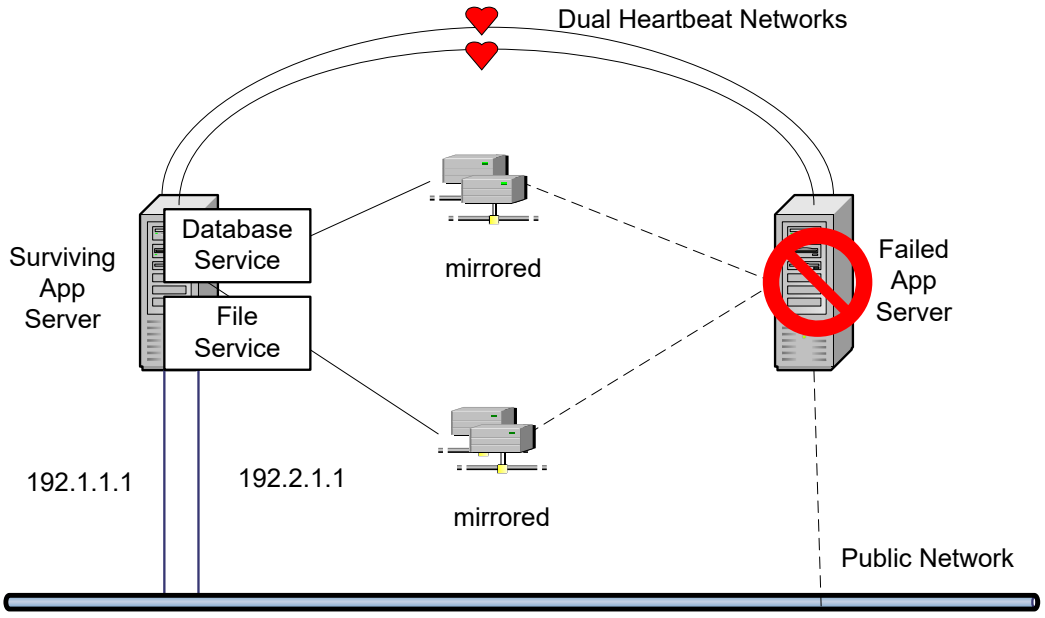


L'indirizzo ip del nodo primario viene assegnato al nodo secondario per indirizzarvi le richieste

Cluster simmetrico – prima di un failover



Cluster simmetrico – dopo un failover



- Monitoraggio

- Il *monitoraggio* riguarda la capacità di osservare e controllare il comportamento di un sistema software mentre è in esecuzione nell'ambiente di produzione
  - l'obiettivo generale del monitoraggio è rendere più semplice la comprensione del comportamento del sistema e dei suoi componenti
  - il monitoraggio ha delle applicazioni importanti proprio nel contesto della disponibilità – per favorire il rilevamento di problemi e per sostenere la loro diagnosi
    - ad es., per capire come (provare a) ripristinare il sistema a seguito di un guasto o fallimento (inizialmente) non previsto
  - ne parleremo meglio nel capitolo sul monitoraggio

Il monitoraggio è molto importante per il supporto alla disponibilità. Supponiamo di avere un sistema software distribuito, ci sono tanti servizi e ogni servizio ha un proprio log in cui registra quello che succede. Se si verifica un errore io devo capire la causa dell'errore per capire come risolverlo, e quindi dovrei andare a vedere tutti i log di tutti i nodi, che potrebbero essere migliaia. Per questo può essere utile un sistema di monitoraggio centrale e automatico che analizza tutta questa mole di dati.

- Altre tattiche per la disponibilità

- Altre tattiche (e attività) per la disponibilità, dalla prospettiva della disponibilità e della resilienza [SSA]
  - seleziona e usa tecnologie hardware e software mature – ad es., hardware tollerante ai guasti
  - usa cluster per l'alta disponibilità e meccanismi di bilanciamento dei carico
  - crea o applica soluzioni per la disponibilità del software
  - consenti la replicazione di componenti
  - applica soluzioni di backup ed effettua il log delle transazioni
  - identifica soluzioni di disaster recovery
  - progetta per il fallimento – come se i guasti e i fallimenti fossero la norma e non le eccezioni
  - verifica in modo pragmatico le soluzioni per la disponibilità

* Discussione

- La disponibilità è una qualità importante in molti sistemi software
  - è importante soprattutto nei sistemi che hanno requisiti stringenti in termini di safety ("sicurezza") per le persone o l'ambiente – oppure che gestiscono informazioni critiche
  - è importante anche nelle situazioni di business in cui le interruzioni di servizio possono causare perdite economiche, danni di immagine e perdita di clienti – e dunque possono avere un impatto significativo per un'organizzazione
  - oggi sono disponibili un numero sempre maggiore di soluzioni tecnologiche per la disponibilità – che però vanno comprese e applicate bene, e vanno spesso anche integrate con l'applicazione di altre opzioni di progettazione e tattiche per la disponibilità
  - torneremo a parlare di disponibilità e affidabilità anche nel contesto della delivery del software