Scalabilità

Claudio Biancalana

Fonti e riferimenti

- Abbot 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.
- Abbot, M.L. and Fisher, M.T. Scalability Rules: 50 Principles for Scaling Web Sites. Addison-Wesley, 2011.

Scalabilità (1)

- La capacità del sistema di gestire volumi di elaborazione crescenti nel futuro, se richiesto
- Molti sistemi sono infatti soggetti a un qualche tipo di incremento del carico di lavoro
  - Un incremento di carico può riguardare, ad es., un incremento del numero di utenti del sistema, oppure del numero di richieste fatte dagli utenti al sistema, oppure della mole di dati che il sistema deve gestire
- La scalabilità riguarda la capacità del sistema di accettare questi incrementi di carico, senza un impatto negativo nei confronti di altre qualità (in particolare, di prestazioni e disponibilità)

Scalabilità (2)

- Un aumento del carico di lavoro, se non viene opportunamente gestito, può infatti avere un impatto negativo
  - **Dapprima sulle prestazioni**
    - All'aumentare del carico di lavoro, aumenta il livello di carico del sistema
    - Quando il livello di utilizzo della risorsa del sistema più critica (il cosiddetto collo di bottiglia) si avvicina alla sua capacità (ovvero, quando quella risorsa è satura), allora il comportamento della risorsa peggiora, più o meno improvvisamente, e fa degenerare le prestazioni dell'interno sistema
  - **Poi sulla disponibilità**
    - La saturazione delle risorse può provocare anche un'interruzione di servizio, che, se il carico non diminuisce, può ripresentarsi anche se si tenta di riavviare il servizio

Requisiti di scalabilità

- I requisiti di scalabilità vengono di solito espressi in termini di incremento del carico di lavoro che il sistema deve essere in grado di assorbire in particolari periodi di tempo, mentre i suoi requisiti relativi a prestazioni e disponibilità vengono ancora soddisfatti
  - Un incremento di carico può riguardare, ad esempio
    - Il numero degli utenti (concorrenti) del sistema
    - Il numero delle richieste, delle transazioni o dei messaggi da gestire (nell'unità di tempo)
    - Il volume dei dati da gestire
    - La complessità dei compiti da eseguire

Progettare per la scalabilità

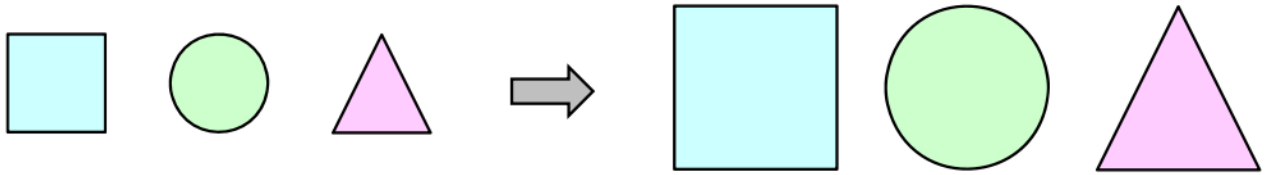
- In queste slide, la trattazione sulla progettazione per la scalabilità si basa soprattutto su **The Art of Scalability** [Abbott and Fisher]
  - Progettare per le prestazioni è in qualche modo un prerequisito per la progettazione per la scalabilità
  - La scalabilità è correlata fortemente anche alla disponiblità, e diverse tattiche per la disponibilità possono sostenere anche la progettazione per la scalabilità
  - Qui ci concentriamo soprattutto sui principi e le regole di progettazione più specifiche per la scalabilità

Scalabilità orizzontale e verticale

- Due approcci principali alla scalabilità
  - Scalabilità verticale (scaling up)
    - Sostituire i componenti hardware esistenti con dei componenti hardware simili ma più potenti, ad esempio, sostituire un server con un server con più processori, più memoria, ecc.
  - Scalabilità orizzontale (scaling out)
    - Aggiungere più componenti simili a un insieme di componenti già in uso nel sistema, ad esempio, aggiungere altri server ad un cluster

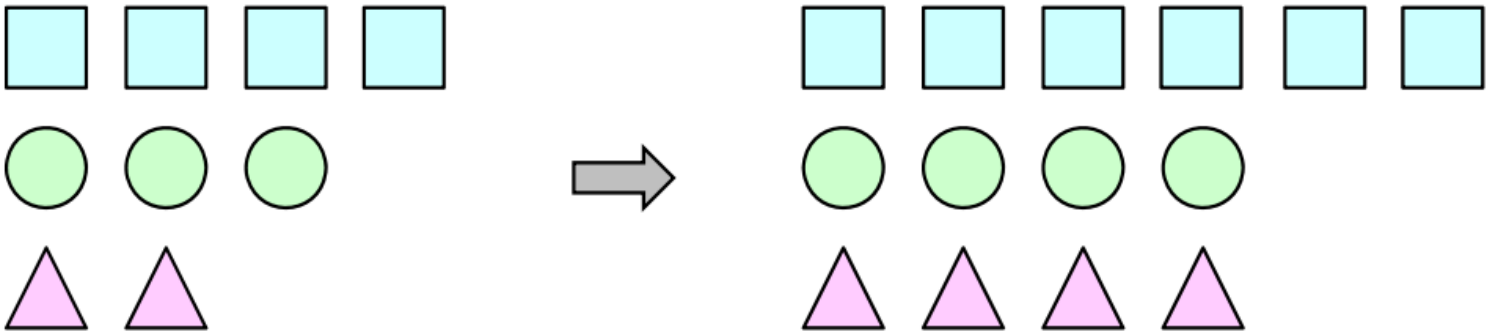
Scalabilità orizzontale e verticale

- Due approcci principali alla scalabilità
  - Scalabilità **verticale** (scaling up)
    - Sostituire i componenti hardware esistenti con dei componenti hardware simili ma più potenti, ad esempio, sostituire un server con un server con più processori, più memoria, ecc.



Scalabilità orizzontale e verticale

- Due approcci principali alla scalabilità
  - Scalabilità **orizzontale** (scaling out)
    - Aggiungere più componenti simili a un insieme di componenti già in uso nel sistema, ad esempio, aggiungere altri server ad un cluster



Scalabilità orizzontale e verticale

- Va notato che questi due approcci (migliorare le risorse oppure aggiungere risorse), da soli, non sufficienti a garantire la scalabilità
  - L'efficacia di una soluzione dipende da come il sistema software è in grado di utilizzare i componenti più potenti oppure aggiuntivi per accomodare l'incremento nel carico di lavoro
    - Se il sistema software non è stato progettato per questo, allora non trarrà vantaggio dalla presenza delle nuove risorse
  - In generale, ciascuno scenario per la scalabilità dovrà descrivere come ciascun incremento nel carico possa essere effettivamente gestito utilizzando degli opportuni componenti sostituitivi o aggiuntivi

Scalabilità orizzontale o verticale?

- In pratica, l'approccio della scalabilità orizzontale è in generale più efficace di quello della scalabilità verticale
  - La **scalabilità verticale,** ovvero sostituire un componente monolitico con un componente simile ma più potente, può essere inefficace a fronte di incrementi molto rapidi del carico
    - Inoltre, in pratica può essere efficace solo nei casi in cui l'incremento atteso del carico sia inferiore all'evoluzione tecnologica attesa nello stesso periodo di riferimento (ad esempio, in relazione alla legge di Moore)
  - Viceversa, la **scalabilità orizzontale**, basata sull'aggiunta di componenti simili aggiuntivi ai componenti esistenti, consente di perseguire una scalabilità «quasi infinita»

Elasticità

- Un altro vantaggio della scalabilità orizzontale è che è compatibile con l'elasticità (elasticity) o scalabilità elastica
  - Una forma di scalabilità orizzontale legata all'aggiunta di risorse (di solito virtuali) acquisite rapidamente da un pool di risorse condivise, soprattutto nel cloud

Verificare la scalabilità

- **I test di carico sono utili per fornire una base quantitativa nella progettazione per le prestazioni e la scalabilità**
  - I test per le prestazioni sono basati sulla simulazione del carico di lavoro attesto
    - Durante l'esecuzione del test vengono misurati sia parametri per le prestazioni (come i tempi di risposta e il throughput) che il livello di utilizzo delle diverse risorse computazionali
    - Questi test richiedono un ambiente di esecuzione il più possibile simile all'ambiente di produzione

Verificare la scalabilità

- **I test di carico sono utili per fornire una base quantitativa nella progettazione per le prestazioni e la scalabilità**
  - I test per la scalabilità sono analoghi, ma misurano le prestazioni e il livello di utilizzo delle risorse anche a fronte di carichi di lavoro crescenti (ad esempio, nel numero di utenti concorrenti o nel volume delle transazioni).
    - Questi test possono essere eseguiti in un ambiente di esecuzione simile a quello di produzione, per determinare i limiti di quell'ambiente
    - Inoltre, questi test andrebbero ripetuti a fronte di ambienti di esecuzione di capacità via via crescente, per determinare l'efficacia della soluzione di scalabilità scelta (verticale o orizzontale)
  - Inoltre, i testi di carico andrebbero ripetuti (come test di regressione) anche a fronte di ogni nuova versione del software

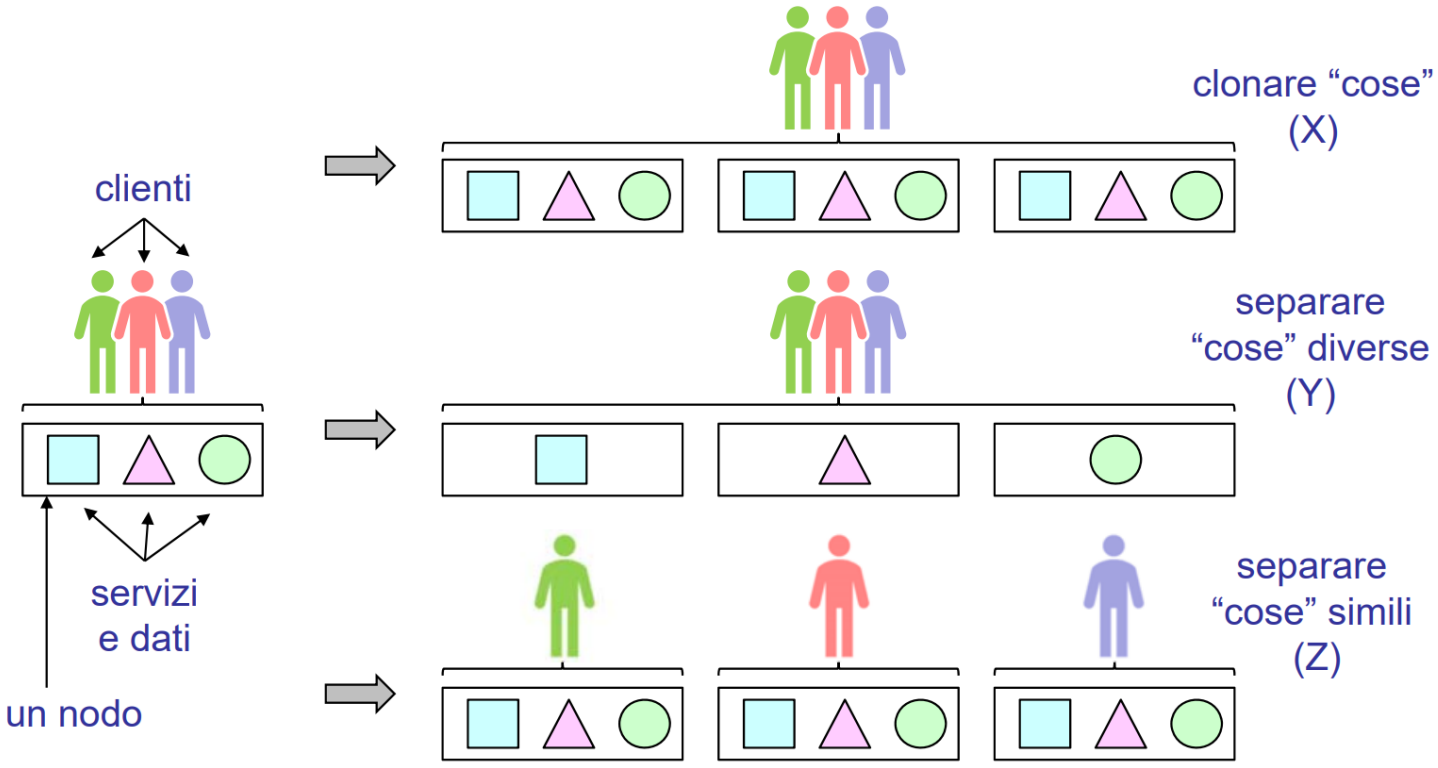
Principi architetturali per la scalabilità

- The Art of Scalability [Abbot e Fisher] presenta diversi principi di progettazione per la scalabilità, i principi più importanti sono
  - La **scalabilità orizzontale** va in genere preferita alla scalabilità verticale
    - Questo richiede di progettare per una distribuzione del carico su più nodi computazionali
  - Inoltre, affinché il carico di lavoro possa essere distribuito su più nodi computazionali in modo efficace ai fini della scalabilità, il sistema deve essere preferibilmente basato su
    - **Interazioni asincrone**
    - **Servizi stateless**
  - Anche il **caching** è utile ai fini della scalabilità

Distribuzione del carico di lavoro

- Un'idea fondamentale nella progettazione per la scalabilità orizzontale di un sistema software è suddividere i suoi «compiti di lavoro» in più parti per poterli distribuire tra più «worker»
  - l'intuizione è che un sistema scalabile orizzontalmente deve essere formato da diversi "lavoratori" su cui viene distribuito il "lavoro" complessivo che il sistema deve svolgere – in modo tale che, se la quantità di "lavoro" da svolgere aumenta, allora il sistema possa scalare aumentando il numero di "lavoratori"
    - il "lavoro" da decomporre è formato da servizi e dati
    - i "lavoratori" su cui viene decomposto il "lavoro" sono genericamente chiamati nodi (di solito sono dei server) ogni nodo è responsabile di alcuni servizi e/o alcuni dati
  - ci sono più modi per decomporre, ai fini della scalabilità, un insieme di servizi e di dati su più nodi

Distribuzione del carico di lavoro

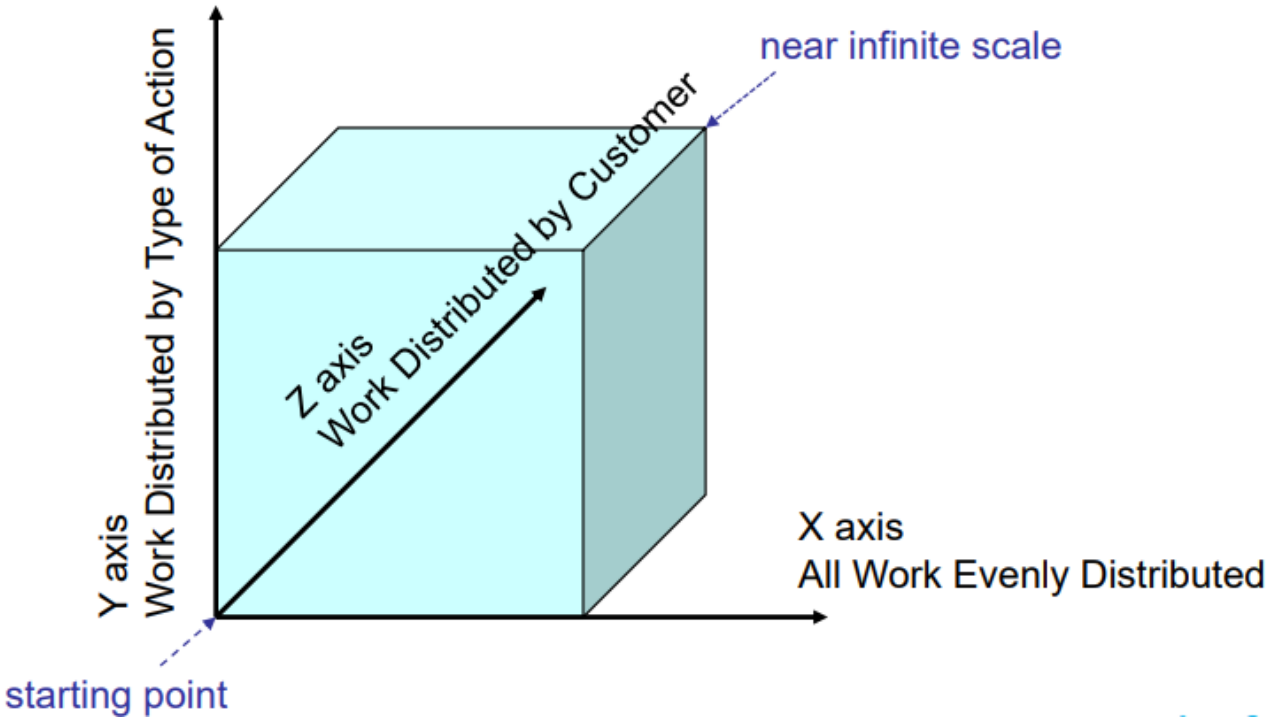


Cubo della scalabilità

- Il cubo della scalabilità [Abbott and Fisher] è un metodo per aiutare a suddividere e distribuire i servizi e i dati di un sistema software in più parti, per sostenere la scalabilità orizzontale del sistema
  - gli assi del cubo della scalabilità (chiamati X, Y e Z) rappresentano modi diversi di decomporre servizi e dati su più nodi – e costituiscono altresì dei criteri per l'identificazione dei nodi da usare nel sistema e delle loro responsabilità
  - il cubo della scalabilità è anche chiamato AKF Scale Cube – dove AKF è il nome della società fondata da Abbott, Keeven e Fisher

Cubo della scalabilità

- Il cubo della scalabilità è un modello che rappresenta tre diversi modi per decomporre un sistema per la scalabilità
  - ciascuna modalità di decomposizione è rappresentata da un diverso asse di scalabilità (X,Y,Z)



Cubo della scalabilità

- Il cubo della scalabilità è un modello che rappresenta tre diversi modi per decomporre un sistema per la scalabilità
  - l'origine del sistema – il punto di coordinate (0,0,0) - rappresenta un sistema monolitico (non decomposto)
    - è di solito il punto di partenza nella progettazione – è un progetto che non è scalabile orizzontalmente
  - ogni asse rappresenta un possibile modo per effettuare una decomposizione "elementare" del sistema – per sostenerne la scalabilità
- la decomposizione può anche avvenire lungo più assi (che sono indipendenti tra loro) – con un effetto benefico moltiplicativo sulla scalabilità
  - [Abbott and Fisher] affermano che queste tre modalità di decomposizione (se applicate correttamente) consentono di scalare quasi qualunque sistema

Asse X del cubo – clonare «cose»

- L'asse X del cubo (clonare "cose") rappresenta la clonazione di tutti i servizi e i dati su più nodi – senza nessuna preferenza o parzialità
  - è una forma di duplicazione orizzontale, in cui ogni nodo eroga gli stessi servizi degli altri nodi
    - ogni richiesta per un servizio da parte di un client può essere girata a un nodo qualunque del sistema
    - è dunque necessario anche un load balancer
  - anche i dati (o la base di dati) sono replicati – dunque ci sono più copie dei dati, e ogni nodo contiene una copia di tutti i dati
    - un'operazione di lettura può essere svolta da un nodo qualunque
    - e operazioni di scrittura vanno invece propagate a tutti i nodi
    - è dunque necessario anche un meccanismo di replicazione dei dati

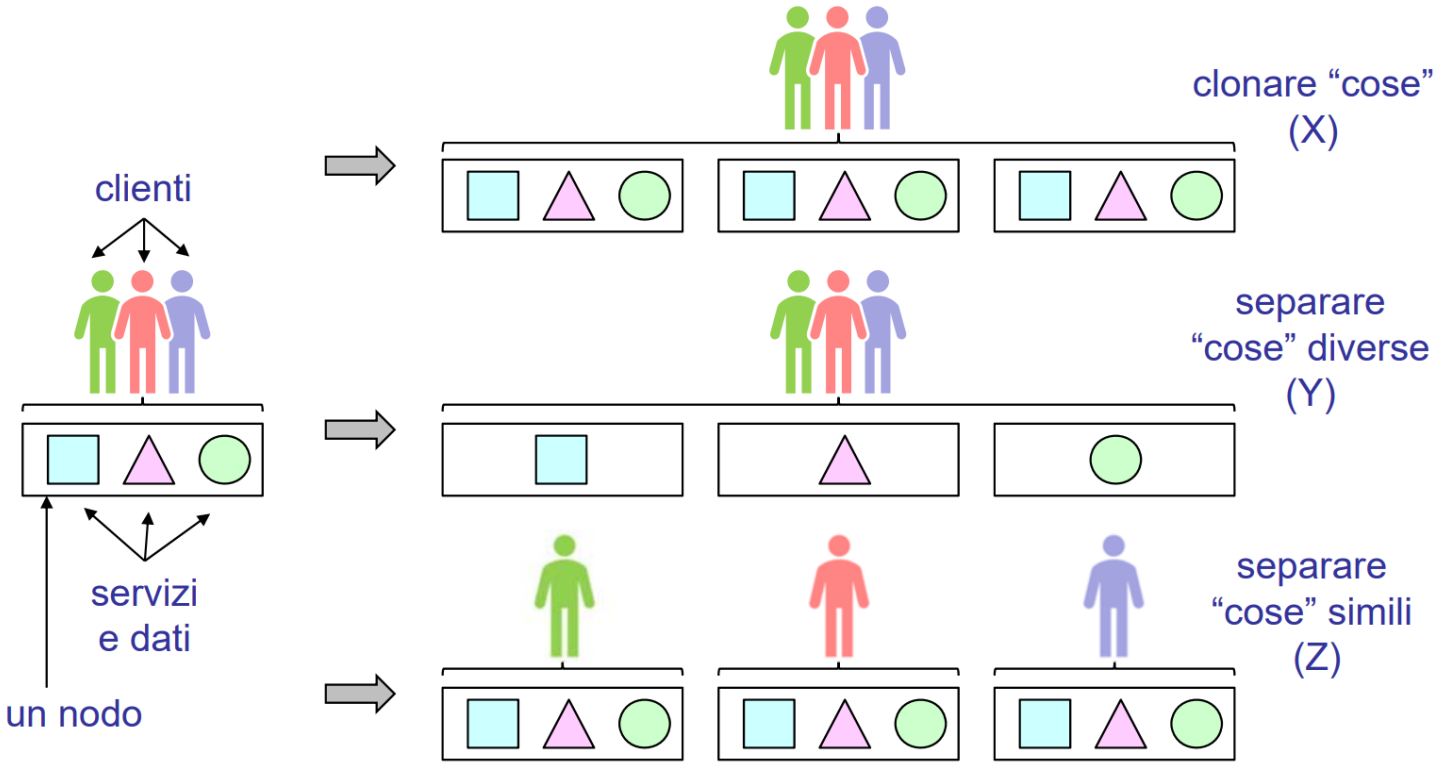
Asse Y del cubo – separare «cose» diverse

- L’asse Y del cubo (separare “cose” diverse) rappresenta una decomposizione delle responsabilità ai diversi nodi per tipo di servizio oppure per tipo di dati oppure per una loro combinazione
  - è una forma di divisione per funzione, servizio o risorsa (dato)
  - in una decomposizione orientata ai servizi, i componenti e i dati necessari per erogare un servizio sono separati dai componenti e dai dati necessari per erogare altri servizi
  - esempi di decomposizioni in un sito di commercio elettronico
    - separare le funzioni di (a) login, (b) consultazione del catalogo prodotti e (c) acquisto – oppure
    - separare i dati relativi ai (i) clienti da quelli relativi ai (ii) prodotti e agli (iii) acquisti
- in generale, questa decomposizione porta a separare sia i servizi che i dati su cui essi operano – anche se la separazione non è sempre completa

Asse Z del cubo – separare «cose» simili

- L’asse Z del cubo (separare “cose” simili) rappresenta una decomposizione delle responsabilità relativa (in genere) ai clienti del sistema
  - viene effettuato un partizionamento dei clienti in base a una qualche loro proprietà – ad es., in genere in base alla loro posizione geografica, oppure in base al loro nome o ad un loro identificatore – ciascun gruppo di clienti (shard) viene assegnato ad un nodo diverso
  - i dati del sistema sono suddivisi sulla base dei gruppi di clienti – anche se talvolta è necessario che alcuni dati vadano replicati su tutti i nodi
  - invece, ciascun nodo deve in genere ospitare tutti i servizi del sistema
  - tutte le operazioni richieste da un certo cliente vengono gestite dal nodo assegnato al suo gruppo

Cubo della scalabilità



Cubo della scalabilità

- Ecco una sintesi dei tre assi di scalabilità
- l'asse X si basa sulla clonazione di un insieme di servizi o dati
    - aiuta a scalare rispetto al volume delle transazioni – ma non scala con la crescita dei dati
    - sostiene la disponibilità e la scalabilità elastica
  - l'asse Y si basa su una separazione delle responsabilità per tipo di servizio o tipo di dato
    - aiuta a scalare rispetto al volume delle transazioni – ma non scala con l'aumento dei clienti
  - l'asse Z si basa su una separazione per cliente o tipo di cliente
    - aiuta a scalare rispetto al volume delle transazioni e all'aumento dei clienti – ma non scala a fronte di una crescita dei dati specifici per una certa funzione o servizio
    - è la soluzione più costosa e difficile da implementare

Comunicazione asincrona e scalabilità

- La **sincronizzazione** si riferisce all'uso e al coordinamento di più unità di esecuzione (thread o processi) simultanei che fanno parte di uno stesso compito complessivo – in cui queste unità di esecuzione devono essere eseguite in ordine corretto al fine di evitare risultati erronei (o altre anomalie)
  - Il login è un esempio di compito in cui ci sono diverse attività da svolgere in modo sequenziale e sincronizzato
    - la cifratura della password inserita dall'utente
    - il confronto tra questa password e la vera password cifrata dell'utente nella base di dati
    - l'utente viene marcato come autenticato nella sua sessione
    - viene generata e presentata la pagina dell'utente
- ma, in un sistema, non tutti i compiti devono essere svolti in modo sincronizzato strettamente

La comunicazione sincrona e asincrona

- La comunicazione in un sistema software distribuito può essere basata su
  - **chiamate sincrone** *(request-reply)* – un processo chiama un altro processo, e rimane in attesa di una risposta
  - **comunicazione asincrona** *(send-and-forget)* – un processo lancia un nuovo processo, e poi termina immediatamente – senza aspettare il completamento dell'operazione richiesta
    - nota: qui ci riferiamo ad una nozione piuttosto generale di comunicazione asincrona – che include meccanismi come, ad es., la comunicazione asincrona basata sullo scambio di messaggi, ma anche le invocazioni remote asincrone
  - chiaramente, l'implementazione di un sistema (o di una sua funzionalità) basata su comunicazione asincrona sarà diversa da un'implementazione basata su chiamate sincrone

Comunicazione sincrona e asincrona

- Le chiamate sincrone – se usate in modo eccessivo o non corretto – possono avere un impatto negativo sulla scalabilità del sistema
- e chiamate sincrone possono diffondere rallentamenti e guasti a cascata – di solito con un effetto moltiplicativo
  - ad es., un problema nel chiamato può anche bloccare il chiamante – il tempo di risposta aumenta, ma aumenta anche il tempo in cui sono state allocate le risorse al chiamante (anche quando è in attesa) – dunque il sistema consuma più memoria, più connessioni e altre risorse
  - questo aumento nell'uso delle risorse può a sua volta bloccare altre chiamate fatte nel sistema – con un impatto negativo dapprima sulle prestazioni e poi, eventualmente, sulla disponibilità e la scalabilità del sistema
    - nel caso di chiamate asincrone, invece, un blocco nel chiamato non provoca anche un blocco nel chiamante – il consumo complessivo di risorse è minore

Comunicazione asincrona e scalabilità

- Pertanto, se la scalabilità è importante, la comunicazione asincrona dovrebbe essere preferita a quella sincrona
- l'uso della comunicazione asincrona è un principio di progettazione fondamentale per la scalabilità
- la comunicazione asincrona va usata soprattutto per chiamate tra servizi e nodi diversi, oppure verso sistemi esterni, oppure se relativa all'attivazione di elaborazioni lunghe
  - detto in altro modo, questi sono i casi in cui va soprattutto evitata la comunicazione sincrona
- è spesso comune che un'operazione debba restituire dei risultati – ecco alcune opzioni
  - accedi ai risultati in modo asincrono
  - utilizza delle callback
  - usa chiamate sincrone con timeout stringenti – e una gestione opportuna delle eccezioni

Comunicazione asincrona e consistenza

- La comunicazione asincrona può avere però un effetto negativo sulla consistenza – poiché non può garantire una consistenza forte
  - informalmente, la **consistenza** si riferisce all'esecuzione atomica di un certo insieme di azioni su un certo insieme di dati – più precisamente, al fatto che questa esecuzione venga percepita (ad es., da un client) come se fosse stata atomica
  - se un'operazione deve essere eseguita in modo consistente **(strong consistency)** e i dati su cui opera sono distribuiti su più nodi, allora probabilmente quest'operazione va implementata mediante chiamate sincrone (ed in modo transazionale)

Comunicazione asincrona e consistenza

- La comunicazione asincrona può avere però un effetto negativo sulla consistenza – poiché non può garantire una consistenza forte
- in ogni caso, la comunicazione asincrona supporta alcune forme deboli di consistenza (che sono spesso accettabili in pratica)
  - ad es., l'eventual consistency garantisce che, se non vengono richiesti nuovi aggiornamenti su un certo dato, allora, prima o poi, gli accessi a quel dato restituiranno l'ultimo valore che gli è stato assegnato
  - questa forma di consistenza risulta adeguata, in pratica, per molte applicazioni reali (ma non tutte!)

Discussione

- La scalabilità è la capacità di un sistema di gestire volumi di elaborazione crescenti
- Queste slide hanno presentato alcune opzioni di progettazione principali per la scalabilità – i principi fondamentali sono quelli relativi alla scalabilità orizzontale (la distribuzione del carico e il cubo della scalabilità), alla comunicazione asincrona (e ai servizi stateless – ma anche il caching)
- alcuni aspetti importanti per la scalabilità sono stati solo accennati oppure sono trattati in altre dispense – come la valutazione della scalabilità e della capacità residua, la scalabilità elastica, il monitoraggio e l'automatizzazione dei rilasci (preferibilmente piccoli)
- questi diversi aspetti non vanno considerati in isolamento – piuttosto bisogna considerarli insieme anche ad altre qualità – come le prestazioni, la disponibilità, la consistenza dei dati e la sicurezza – ed identificare e valutare eventuali compromessi