



Luca Cabibbo  
Architettura  
dei Sistemi  
Software

# Scalabilità

**dispensa asw270**  
ottobre 2024

*Rule 50 – Be Competent.*  
*Martin L. Abbott and Michael T. Fisher*



## - Riferimenti

- ❑ Luca Cabibbo. **Architettura del Software: Strutture e Qualità**. Edizioni Efestò, 2021.
  - Capitolo 13, **Scalabilità**
- ❑ [TAoS] 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. Citato come [TAoS]
- ❑ Abbott, M.L. and Fisher, M.T. **Scalability Rules: 50 Principles for Scaling Web Sites**. Addison-Wesley, 2011.



## - Obiettivi e argomenti

### □ Obiettivi

- presentare la qualità della scalabilità
- illustrare alcune attività, principi e regole di progettazione per la scalabilità

### □ Argomenti

- scalabilità
- progettare per la scalabilità
- discussione



## \* Scalabilità

### □ **Scalabilità** (*scalability*)

- la capacità del sistema di gestire volumi di elaborazione crescenti nel futuro, se richiesto
  - molti sistemi sono soggetti a un qualche tipo di incremento nel carico di lavoro
  - la scalabilità riguarda la capacità del sistema di accettare questi incrementi di carico, senza un impatto negativo nei confronti di altre qualità
- La scalabilità è importante soprattutto quando un'organizzazione usa il proprio sistema software per offrire i propri servizi direttamente ai propri clienti finali



# Scalabilità: alcuni esempi recenti

- Alcuni recenti problemi di scalabilità – relativi ai “click day”
  - aprile 2020 – il sito dell’INPS va in crash per il bonus da 600 euro
  - novembre 2020 – il sito del Ministero dell’Ambiente va in crash per il bonus bici
  - dicembre 2020 – stessi problemi per partecipare al cashback di Natale con l’app IO

📶 81% 17:27



Si è verificato un errore temporaneo nel salvataggio di questa carta, riprova.

Annulla

Riprova

📶 81% 17:27

< Salva la carta ?



Qualcosa è andato storto.

Prima di riprovare, controlla la tua casella di post riceverai a breve un'email da no-reply@pagopa.gov.it con l'esito del pagamento.

5

Scalabilità

Luca Cabibbo ASW



# Scalabilità

- Un aumento del carico di lavoro, se non viene opportunamente gestito, può avere un impatto negativo
  - sulle prestazioni
  - sulla disponibilità
  - il problema è dovuto al degrado legato prima all’aumento nel livello di utilizzazione delle risorse – e poi alla loro saturazione

6

Scalabilità

Luca Cabibbo ASW



## Requisiti di scalabilità

- I requisiti di scalabilità possono essere 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
    - 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



## Requisiti di scalabilità

- I requisiti di scalabilità possono essere 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
  - inoltre, un incremento di carico può essere
    - di lungo termine (e anticipato nel suo arrivo)
    - transiente (e anticipato nel suo arrivo)
    - transiente ma non anticipato nel suo arrivo



# Scalabilità ed elasticità

- L'elasticità è una forma particolare di scalabilità
  - per scalabilità si intende di solito la capacità del sistema di gestire incrementi di lungo termine del carico di lavoro
    - la scalabilità viene di solito gestita mediante un aumento “statico” delle risorse computazionali in relazione all'incremento di carico atteso
  - l'**elasticità** (**elasticity**, o **scalabilità elastica**) è invece la capacità del sistema di gestire carichi di lavoro che possono aumentare o diminuire in modo variabile nel corso del tempo, con un uso efficace delle risorse
    - l'elasticità richiede un'allocazione “dinamica” delle risorse computazionali, in relazione alle effettive variazioni di carico
    - per conseguire un uso efficace delle risorse, le risorse che non sono più necessarie devono poter essere anche deallocate, per riutilizzare queste risorse per altri scopi oppure anche, nel cloud, per risparmiare

9

Scalabilità

Luca Cabibbo ASW



## \* Progettare per la scalabilità

- La trattazione che facciamo sulla progettazione per la scalabilità è basata soprattutto su **The Art of Scalability** [TAoS]
  - [SSA] tratta la scalabilità nel contesto più ampio della prospettiva per le prestazioni e la scalabilità – progettare per le prestazioni è un prerequisito importante per la scalabilità
  - la scalabilità è correlata anche alla disponibilità – alcune tattiche per la disponibilità sostengono anche la scalabilità
  - ci concentriamo soprattutto su principi e regole di progettazione più specifiche per la scalabilità

10

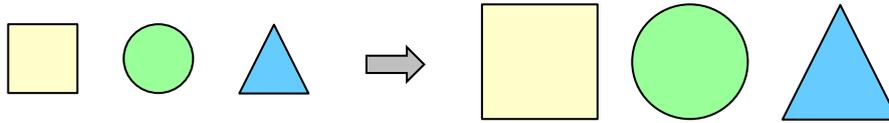
Scalabilità

Luca Cabibbo ASW

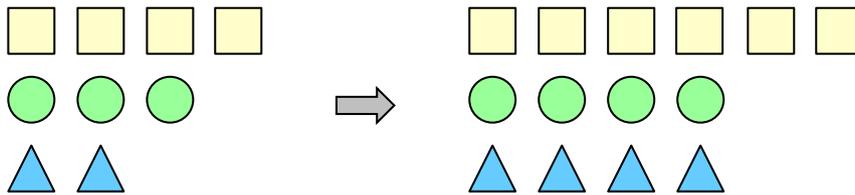


## - Scalabilità orizzontale e verticale

- ▣ Due approcci principali alla scalabilità
  - *scalabilità verticale (scaling up)*



- *scalabilità orizzontale (scaling out)*



## Scalabilità orizzontale e verticale

- ▣ Questi due approcci, da soli, non sono 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 gestire l'incremento nel carico di lavoro
  - nella progettazione per la scalabilità, bisogna descrivere come ciascun tipo di incremento nel carico può essere effettivamente gestito utilizzando questi componenti sostitutivi o aggiuntivi



# Scalabilità orizzontale, verticale ed elastica

- L'approccio della scalabilità orizzontale è spesso considerata più efficace di quello della scalabilità verticale
  - la scalabilità verticale può essere inefficace a fronte di incrementi molti rapidi del carico
  - la scalabilità orizzontale consente invece di perseguire anche una scalabilità “quasi infinita”
  - la scalabilità orizzontale è anche compatibile con la scalabilità elastica
  
- qui consideriamo soprattutto la scalabilità orizzontale



## - 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 atteso, in un ambiente simile a quello di produzione
  - i test per la scalabilità sono analoghi, ma dovrebbero misurare le prestazioni e il livello di utilizzazione delle risorse anche a fronte di carichi di lavoro crescenti e di ambienti di capacità via via crescente – per determinare l'efficacia della soluzione di scalabilità scelta



## - Pianificare per la scalabilità



- La **capacità residua** (*headroom*) è una misura della quantità di capacità del sistema (in una sua certa configurazione) che non viene attualmente utilizzata – e che può essere utilizzata per servire un incremento del carico di lavoro
  - utile per stimare quando potrebbero iniziare dei problemi con quella configurazione – soprattutto in caso di incrementi di carico di lungo termine



## Capacità residua



- Un semplice esempio
  - un sistema ha una capacità massima di 100 t/s (transazioni al secondo) – determinata con un test di carico
  - se il carico attuale è di 30 t/s (nell'utilizzazione di picco), allora la capacità residua è di 70 t/s
  - se è previsto un incremento del carico di lavoro di 10 t/s al trimestre, allora la capacità residua è di 7 trimestri



- Un esempio un pochino più complesso
  - un sistema ha
    - una capacità massima (teorica) di 100 t/s
    - un carico attuale (di picco) di 30 t/s
  - la percentuale di utilizzo ideale potrebbe essere del 50%
    - non è mai una buona idea che il livello di utilizzazione di una risorsa raggiunga il 100%
  - sono previste ottimizzazioni per ulteriori 20 t/s (teoriche)
  - la capacità residua è
$$(100+20)*50\% - 30 = 30 \text{ t/s}$$
  - se è previsto un incremento del carico di 10 t/s al trimestre, allora la capacità residua è di soli 3 trimestri



## - Principi architetturali per la scalabilità

- Secondo **The Art of Scalability**, i principi di progettazione più importanti per la scalabilità sono
  - la **scalabilità orizzontale** – che va di solito preferita alla scalabilità verticale
  - il sistema deve essere preferibilmente basato su
    - **interazioni asincrone**
    - **servizi stateless**
  - anche il **caching** è utile



## - Distribuzione del carico di lavoro

- Intuizioni per sfruttare la scalabilità orizzontale di un sistema software – che deve svolgere dei “compiti di lavoro”
  - il sistema è formato da diversi “lavoratori”
  - il “lavoro” complessivo da svolgere nel sistema viene distribuito su questi “lavoratori”
  - un aumento della quantità di “lavoro” da svolgere viene gestito aumentando il numero di “lavoratori”
  - a tal fine, i “compiti di lavoro” del sistema software devono essere suddivisi in più parti, da distribuire tra più “lavoratori”
  - il “lavoro” da decomporre è costituito da servizi e dati
  - i “lavoratori” sono i nodi (server) – ogni nodo è responsabile di alcuni servizi o dati
  - è importante comprendere come decomporre un sistema ai fini della scalabilità

19

Scalabilità

Luca Cabibbo ASW



## - Cubo della scalabilità

- Il *cubo della scalabilità* è un modello 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 (X, Y e Z) rappresentano tre modi (o criteri) diversi per decomporre servizi e dati su più nodi
  - è chiamato *AKF Scale Cube* – AKF è il nome della società fondata da Abbott, Keeven e Fisher (Abbott e Fisher sono gli autori di [TAoS])

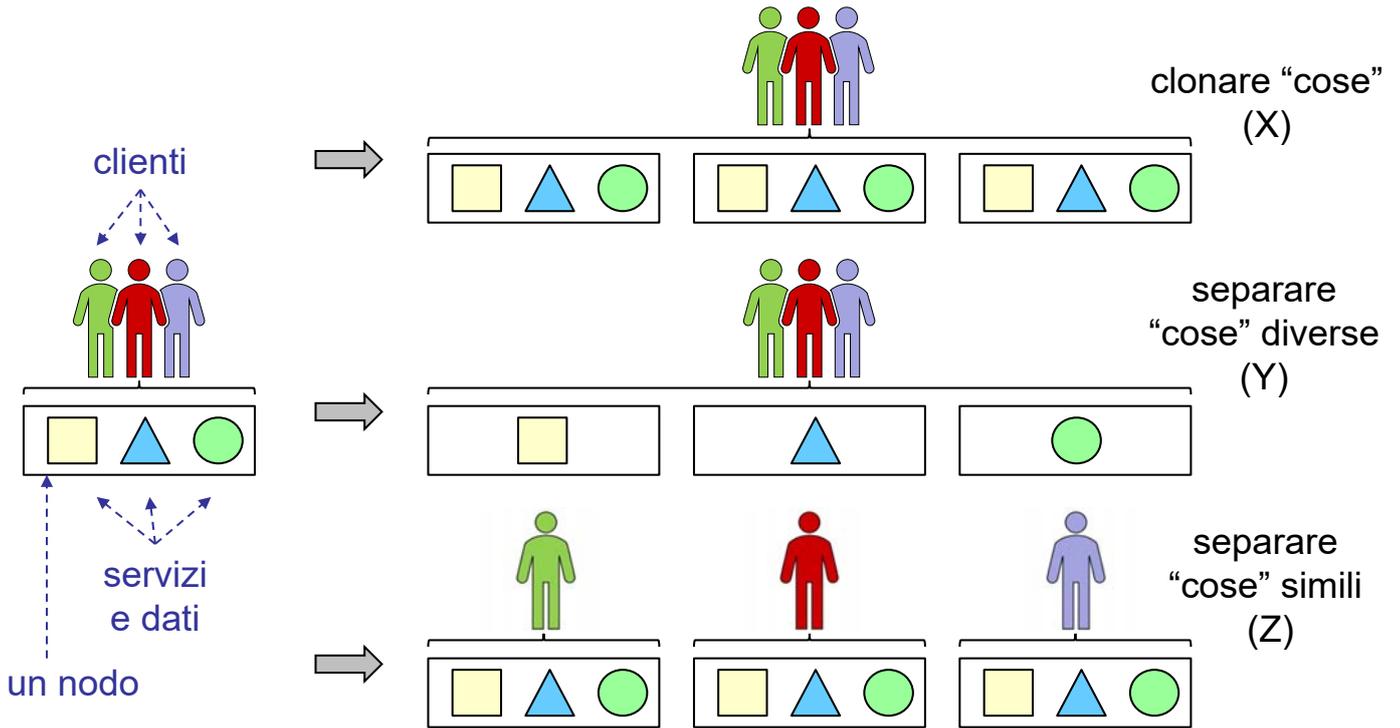
20

Scalabilità

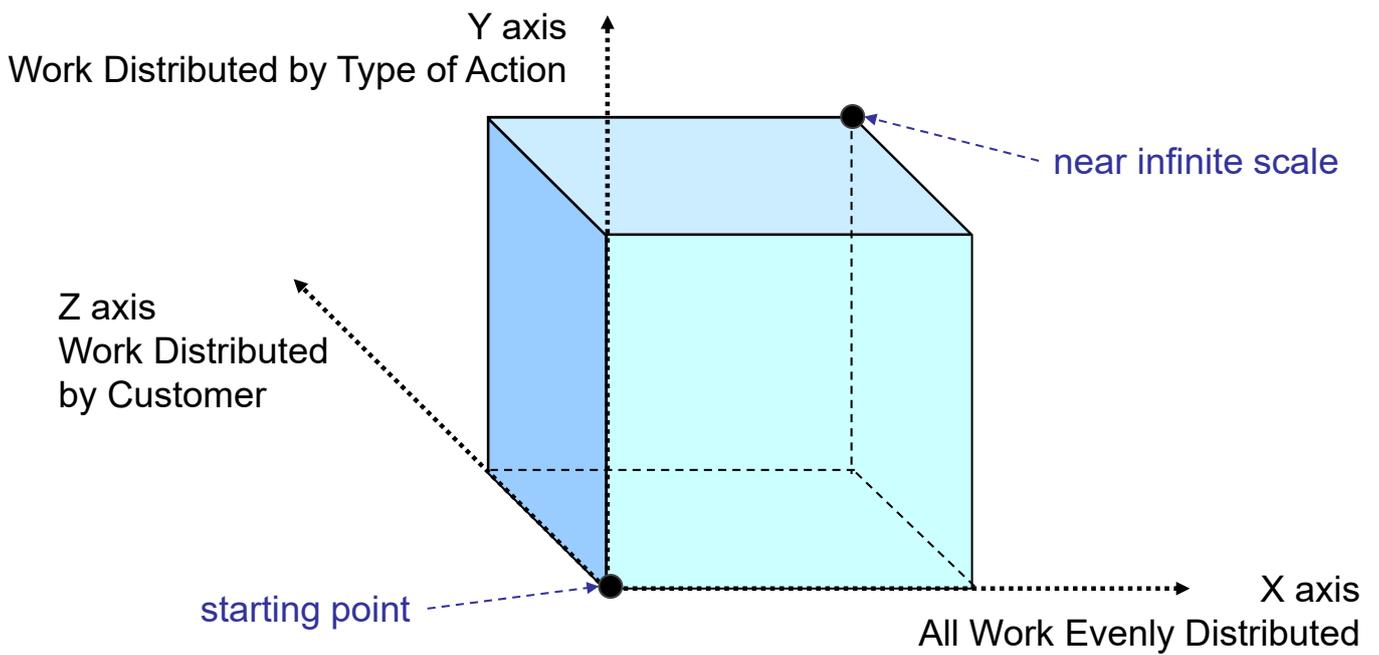
Luca Cabibbo ASW



# Cubo della scalabilità



# Cubo della scalabilità





## Asse X del cubo – clonare “cose”

- L'*asse X* del cubo (*clonare “cose”*) rappresenta la clonazione di tutti i servizi e tutti i dati su più nodi
  - una forma di replicazione orizzontale – ogni nodo eroga tutti i servizi (come ogni altro nodo)
    - è necessario anche un load balancer
  - anche i dati (o la base di dati) sono replicati – ogni nodo contiene una copia di tutti i dati
    - è 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 in più nodi diversi – una decomposizione per tipo di servizio, per tipo di dati o per una loro combinazione
  - 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 altri servizi
  - questa decomposizione porta a separare sia i servizi che i dati su cui essi operano – ma la separazione dei dati non è sempre completa
  - è necessario anche un gateway



## Asse Z del cubo – separare “cose” simili

- L'**asse Z** del cubo (*separare “cose” simili*) rappresenta una decomposizione delle responsabilità in più nodi simili ma distinti
  - una decomposizione basata su una proprietà, che definisce dei gruppi – ciascun gruppo (shard) viene poi assegnato a un nodo diverso
    - un esempio comune è una decomposizione dei clienti del sistema in base alla loro posizione geografica
  - ciascun nodo ospita tutti i servizi del sistema – e gestisce tutte le operazioni richieste dal proprio gruppo di clienti
  - anche i dati del sistema vengono suddivisi sulla base dei gruppi di clienti – ma spesso alcuni dati devono essere replicati su tutti i nodi

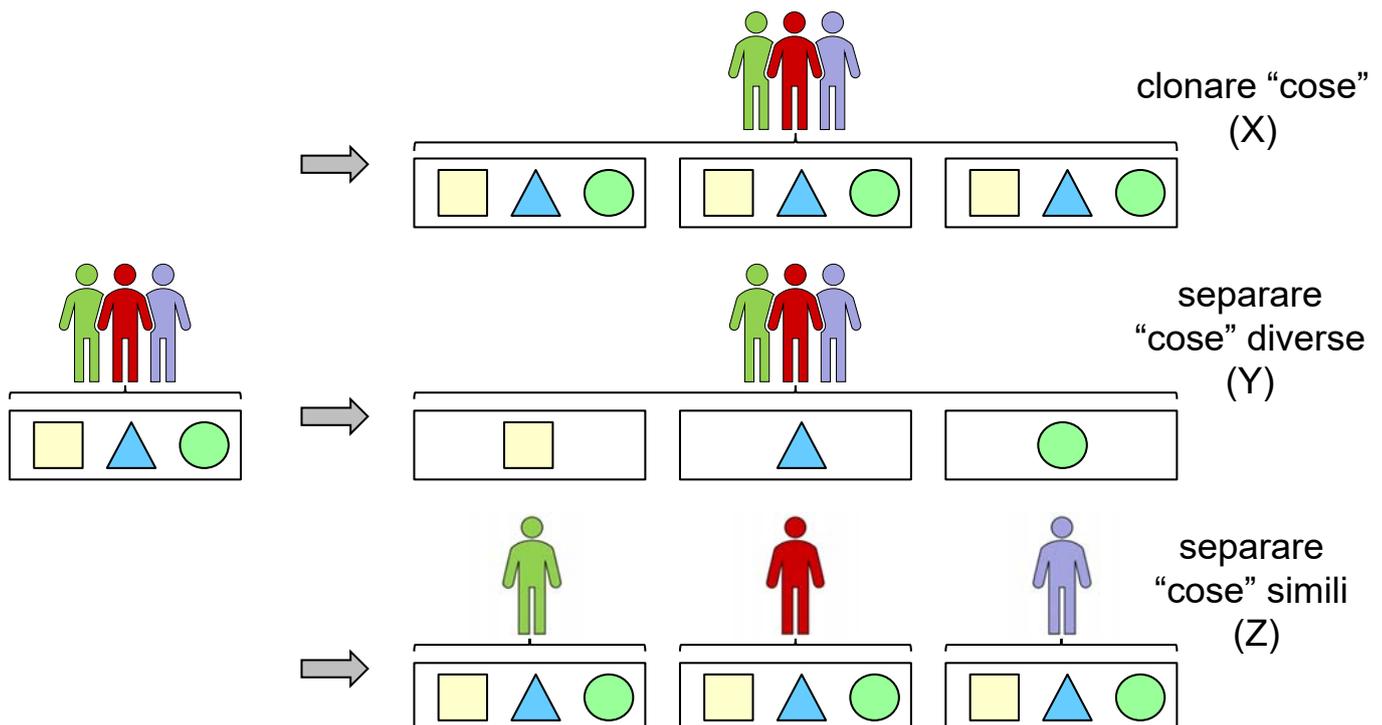
25

Scalabilità

Luca Cabibbo ASW



## Cubo della scalabilità



26

Scalabilità

Luca Cabibbo ASW



## - Scalabilità dei servizi e dei dati

- Il cubo della scalabilità definisce un modello generale per decomporre i servizi e i dati di un sistema software
  - è utile però fare anche delle considerazioni separate riguardo alla decomposizione dei servizi e delle applicazioni e a quella dei dati e delle basi di dati
    - l'implementazione della decomposizione relativa ai servizi e ai dati può variare anche in modo significativo
  - è anche utile fare delle considerazioni sulla capacità (o meno) di sostenere i diversi tipi specifici di incrementi di carico
    - aumento del numero di utenti, aumento del numero di richieste, aumento del volume dei dati



## Asse X per la scalabilità dei servizi

- L'asse X rappresenta la clonazione di tutti i servizi applicativi negli N nodi in cui viene decomposto il sistema
  - ogni nodo contiene tutti i servizi
  - ogni richiesta per un servizio può essere assegnata a uno qualunque dei nodi – mediante un *load balancer* – e la gestione della richiesta coinvolge solo il nodo selezionato
  - è una decomposizione semplice da implementare – soprattutto se i servizi sono stateless
    - altrimenti, bisogna gestire in qualche modo lo stato (delle sessioni) dei servizi



## Asse X per la scalabilità dei dati

- L'asse X rappresenta anche la clonazione di tutti i dati in tutti gli N nodi, ciascuno con la sua base di dati
  - in ogni nodo, la base di dati contiene una copia di tutti i dati
  - è necessaria la *replicazione* dei dati tra i diversi nodi
    - viene di solito adottata una replicazione asincrona
  - tuttavia, la replicazione asincrona ha degli inconvenienti – da comprendere e da saper gestire se necessario



## Asse X per la scalabilità

- Benefici
  - è la decomposizione più semplice da progettare e da implementare
  - sostiene la scalabilità a fronte dell'aumento del volume delle transazioni – ogni nodo riceve  $1/N$  del carico di lavoro
  - è facile aggiungere nuovi nodi – anche perché sono tutti uguali
  - sostiene la tolleranza ai guasti
  - sostiene la scalabilità elastica



## Asse X per la scalabilità

### ❑ Inconvenienti

- la scalabilità è limitata dalla dimensione dei servizi e soprattutto dalla dimensione dei dati
- replicare grandi quantità di dati è costoso
- l'accesso ai dati non scala se il rapporto scritture-letture è alto
- ci possono essere problemi di consistenza e di attualità dei dati
- non sostiene l'isolamento dei guasti



## Asse Y per la scalabilità dei servizi

- ❑ L'asse Y rappresenta un partizionamento dei servizi applicativi negli N nodi in cui viene decomposto il sistema
  - la decomposizione dei servizi lungo l'asse Y si basa su una separazione delle funzionalità del sistema
  - ogni nodo è responsabile solo di uno o pochi servizi
  - ogni richiesta per un servizio viene assegnata al nodo responsabile per quel servizio – mediante un *gateway*
    - la gestione di una richiesta effettuata da un cliente finale può coinvolgere uno o più servizi e dunque uno o più nodi



## Asse Y per la scalabilità dei dati

- L'asse Y rappresenta una separazione dei dati per significato, funzione o utilizzo
  - i dati possono essere suddivisi in modo corrispondente alla suddivisione dei servizi o delle funzioni lungo l'asse Y
    - ci possono essere delle sovrapposizioni tra i dati di interesse per servizi o funzioni differenti
  - è anche possibile (ma meno consigliata) una decomposizione lungo l'asse Y con riferimento soprattutto ai dati (risorse) – e poi suddividere i servizi di conseguenza
  - una doppia decomposizione effettuata separatamente sui servizi e sui dati (risorse) è invece sconsigliata



## Asse Y per la scalabilità

- Benefici
  - è una decomposizione semplice da concettualizzare – ma complessa da implementare
  - il partizionamento dei servizi di solito induce una decomposizione dei dati su cui essi operano
  - le prestazioni possono beneficiare dalla decomposizione dei servizi e dei dati
  - può sostenere la scalabilità sia rispetto all'aumento del volume delle transazioni che all'aumento del volume dei dati
  - sostiene l'isolamento dei guasti
  - può favorire la produttività dei team di sviluppo
  - può consentire di rilasciare i diversi servizi separatamente tra loro



## Asse Y per la scalabilità

### ❑ Inconvenienti

- l'implementazione può essere complessa e costosa
- richiede di gestire più tipi di nodi
- richiede di gestire più basi di dati – e può richiedere l'uso di più sistemi per la gestione dei dati
- di per sé, non sostiene la tolleranza ai guasti
- sostiene l'isolamento dei guasti solo in modo parziale
- non sostiene la scalabilità elastica



## Asse Z per la scalabilità dei servizi

- ❑ L'asse Z rappresenta un partizionamento basato su una proprietà (un valore o una funzione) che viene valutata all'arrivo di ogni richiesta
  - spesso è una proprietà del cliente che ha effettuato la richiesta
    - ogni nodo contiene tutti i servizi – ma solo i dati relativi a uno specifico gruppo di clienti
    - ogni richiesta viene servita dal nodo associato al gruppo di clienti a cui appartiene il cliente



## Asse Z per la scalabilità dei dati

- L'asse Z rappresenta un partizionamento dei dati in relazione a un certo tipo di dati e a una loro proprietà
  - il partizionamento può essere relativo ai clienti del sistema
    - questo di solito favorisce le prestazioni e l'isolamento dei guasti
  - il partizionamento può essere relativo anche ad altri criteri
    - ad es., i prodotti di un catalogo
  - in genere non tutti i dati vengono partizionati



## Asse Z per la scalabilità

- Benefici
  - è una decomposizione semplice da concettualizzare
  - è una soluzione che può favorire in modo significativo la scalabilità – sia rispetto a un aumento del volume delle transazioni che dei dati
  - sostiene l'isolamento dei guasti
  - sostiene la scalabilità nel volume delle transazioni
  - sostiene la scalabilità nel numero dei clienti



## Asse Z per la scalabilità

### ❑ Inconvenienti

- è una soluzione complessa e costosa da implementare
- i nodi sono simili tra di loro – ma hanno configurazioni differenti
- può essere difficile cambiare il criterio di partizionamento – ad es., per decomporre il sistema su un numero maggiore di nodi
- sostiene l'isolamento dei guasti (e la tolleranza ai guasti) solo in modo parziale
- non sostiene la scalabilità elastica

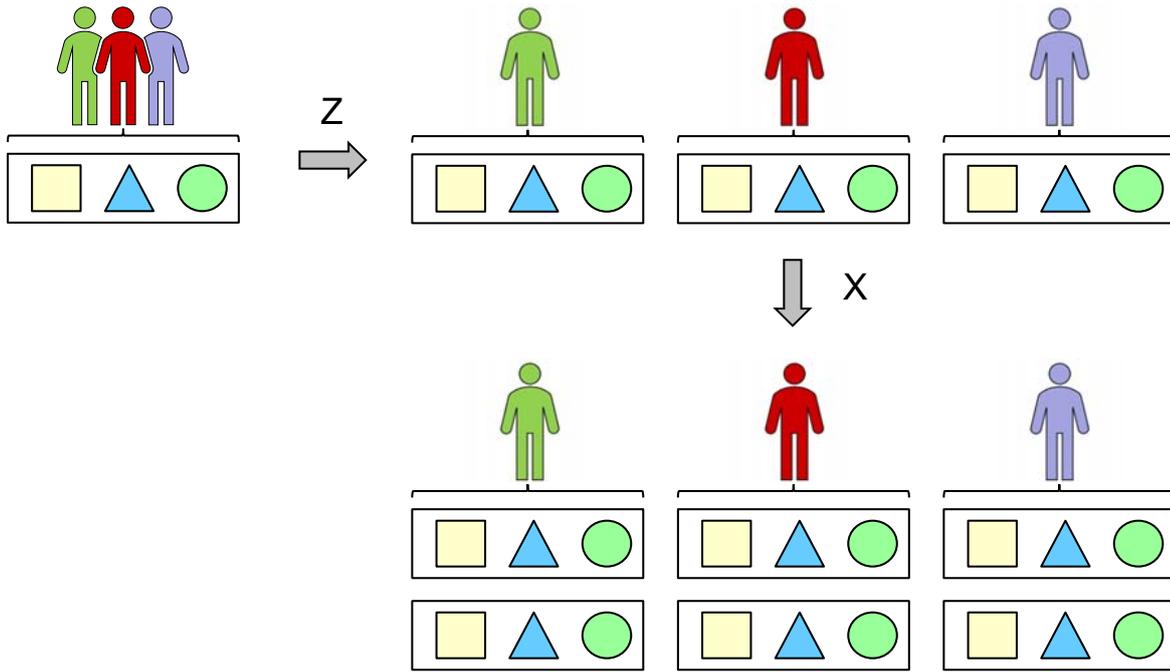


## - Decomposizione lungo più assi

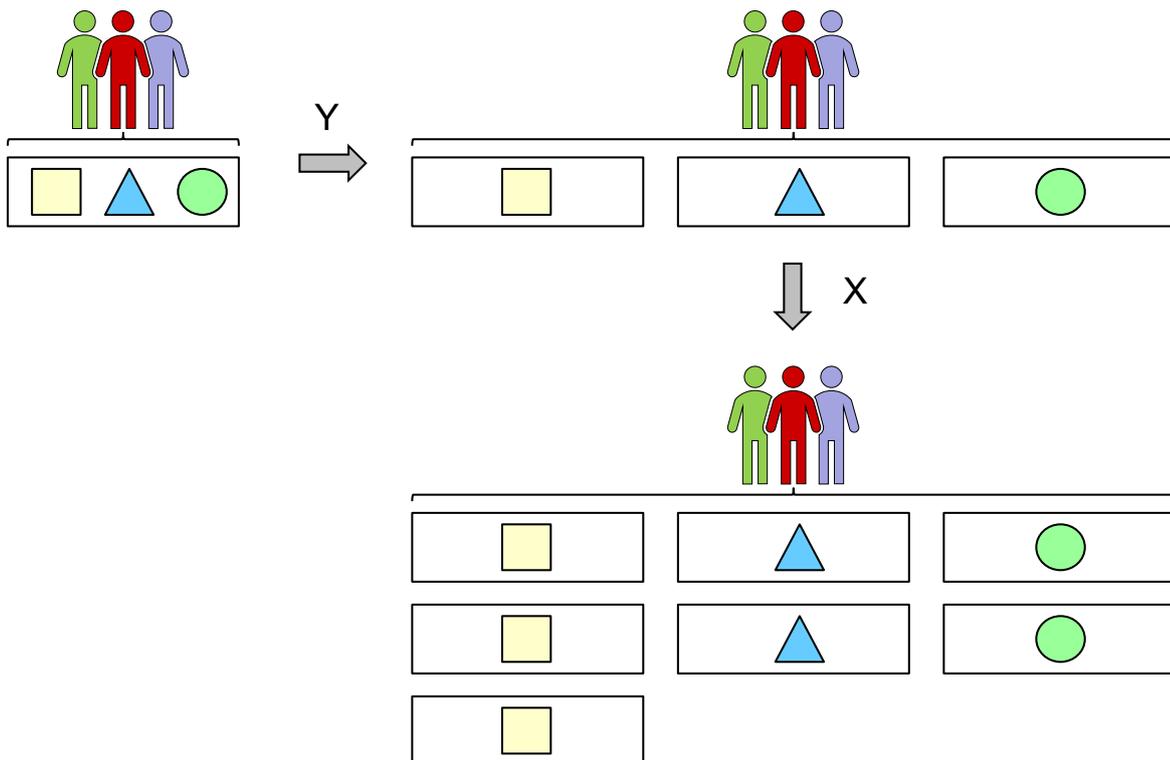
- ❑ La decomposizione lungo un solo asse del cubo potrebbe essere sufficiente nel caso di una crescita limitata e lenta del carico di lavoro
  - per sostenere meglio la scalabilità è anche possibile effettuare una decomposizione del sistema lungo due o più assi
  - questo può avere un impatto positivo anche sulla disponibilità



# Decomposizione lungo più assi

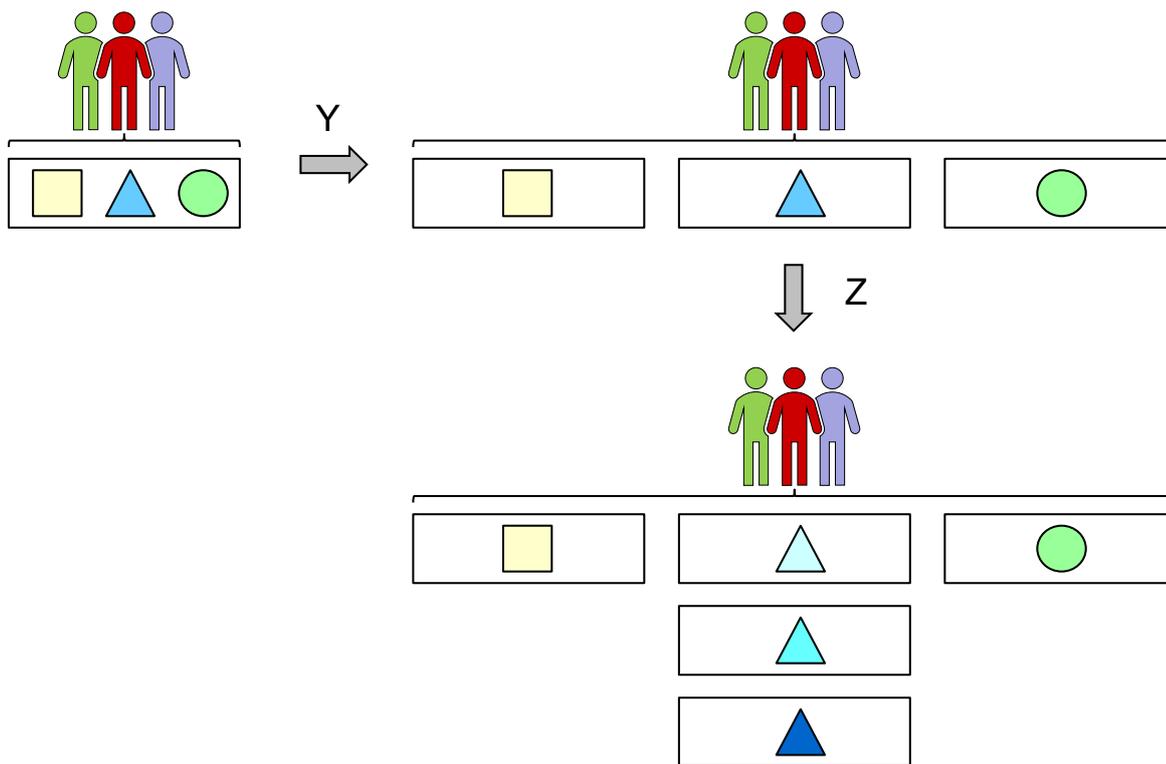


# Decomposizione lungo più assi





## Decomposizione lungo più assi



43

Scalabilità

Luca Cabibbo ASW



## - Cubo della scalabilità: Sommario

- 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
    - sostiene la tolleranza ai guasti 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 e alla crescita dei dati
    - sostiene l'isolamento dei guasti (tra servizi)
  - 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
    - sostiene l'isolamento dei guasti (tra gruppi di clienti)
  - è possibile suddividere un sistema lungo più assi di scalabilità

44

Scalabilità

Luca Cabibbo ASW



## - Comunicazione asincrona e scalabilità

- La **sincronizzazione** si riferisce all'uso e al coordinamento di più unità di esecuzione (thread o processi) concorrenti che fanno parte di uno stesso compito complessivo, quando le diverse unità devono essere eseguite in un ordine opportuno (per evitare anomalie)
  - ad es., il login
  - ma non sempre è necessaria una sincronizzazione stretta



## Comunicazione sincrona e asincrona

- La comunicazione in un sistema software distribuito può essere basata su
  - **chiamate sincrone** (*request-reply*)
  - **comunicazione asincrona** (*send-and-forget*)
- Le chiamate sincrone possono avere un impatto negativo sulla scalabilità
  - infatti possono diffondere rallentamenti e guasti a cascata – e possono indurre un livello di utilizzazione maggiore delle risorse
  - nel caso della comunicazione asincrona il consumo complessivo di risorse è in genere minore



## Comunicazione asincrona e scalabilità

- Quando la scalabilità è importante, la comunicazione asincrona dovrebbe essere preferita a quella sincrona
  - 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
  - alcune opzioni da usare quando un'operazione deve comunque restituire dei risultati
    - accedi ai risultati in modo asincrono
    - utilizza delle callback
    - usa chiamate sincrone con timeout stringenti



## Comunicazione asincrona e consistenza

- La comunicazione asincrona può avere un effetto negativo sulla consistenza dei dati – non può garantire una consistenza forte
  - 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 (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 (e in modo transazionale)
  - in ogni caso, la comunicazione asincrona supporta alcune forme deboli di consistenza – che sono spesso accettabili per molte applicazioni reali
    - ad es., l'**eventual consistency**





## Scalabilità e servizi stateless

- Quando la scalabilità è importante, i servizi dovrebbero essere preferibilmente stateless
  - è un altro principio di progettazione fondamentale per la scalabilità
  - molti servizi scalabili di successo sono stateless – ad es., Google Search
- Tuttavia, non sempre è possibile utilizzare solo servizi stateless – molti servizi richiedono la gestione di uno stato
  - nel caso di servizi stateful
    - prova a rendere il servizio stateless
    - minimizza la dimensione dello stato delle sessioni
    - tra le alternative per la gestione dello stato, scegli la soluzione di compromesso meno peggiore



## Gestione dello stato dei servizi

- Alcune opzioni per la gestione dello stato nei servizi
  - gestire lo stato delle sessioni nell'application server
  - gestire lo stato delle sessioni come cookie nel browser dell'utente
  - gestire lo stato delle sessioni in un repository (distribuito e condiviso dai nodi dell'application server) – una base di dati oppure una cache condivisa



## - Caching per la scalabilità

- Un altro approccio per gestire un grande volume di richieste consiste nel cercare di evitare (almeno in parte) di dover gestire queste richieste
  - è possibile utilizzare una cache
  - in generale, una **cache** è una memoria usata da un dispositivo o da un'applicazione per la memorizzazione temporanea di dati che probabilmente devono essere usati presto di nuovo
  - i tipi di cache più rilevanti per l'architettura del software sono le object cache, le application cache e le content delivery network

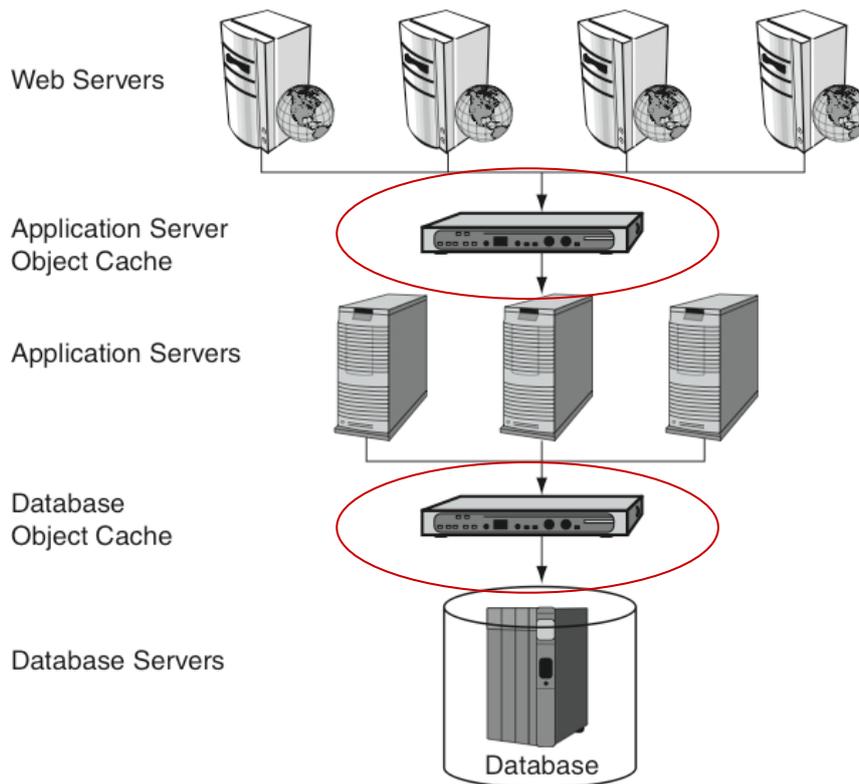


## Object cache

- Una **object cache** consente di memorizzare oggetti dell'applicazione che devono essere riutilizzati
  - ad es., **Memcached**
  - è un insieme di coppie chiave-valore, in cui gli oggetti sono memorizzati in una forma serializzata
  - una object cache può essere posta
    - tra l'application server e la base di dati
    - tra il web server e l'application server
  - un utilizzo comune è la gestione dello stato delle sessioni



## Object cache



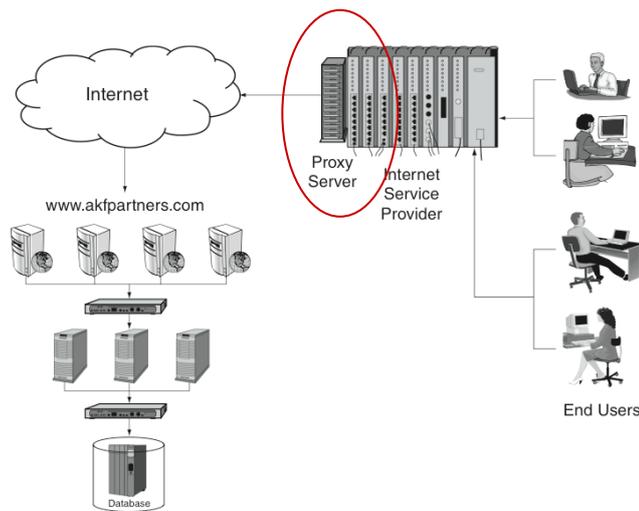
## Application cache

- Una **application cache** è una cache posta davanti all'intera applicazione – tra l'utente e l'applicazione
  - di solito memorizza richieste-risposte HTTP
  - esistono molti tipi di application cache – i due tipi principali sono
    - **proxy cache** (o **forward proxy cache** o **proxy server**)
    - **reverse proxy cache** (o **gateway cache**)

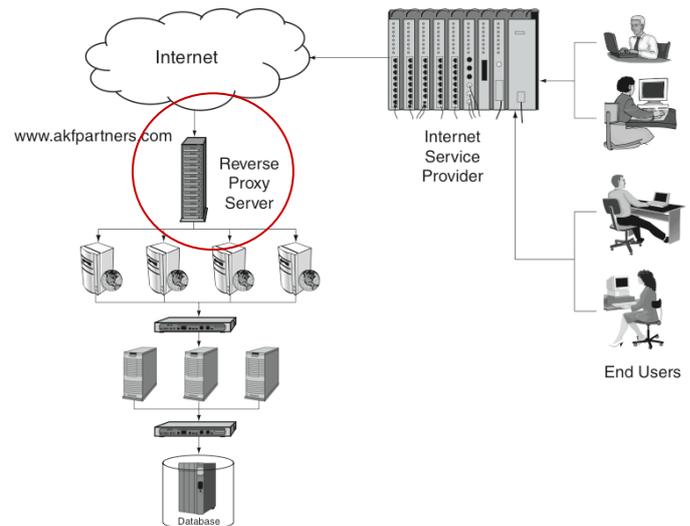


# Application cache

## Proxy cache



## Reverse proxy cache



# Content Delivery Network

- Una **Content Delivery Network (CDN)** consente di collocare i contenuti **statici** di un'applicazione web vicino all'utente finale – per ridurre il tempo di risposta per quei contenuti, ma anche per ridurre il numero di richieste nella parte server del sistema software
  - una CDN viene spesso realizzata come una rete di gateway cache collocate in diverse aree geografiche – posizionate ai “margin” di Internet
  - il dominio di una CDN viene poi indicato nei DNS come alias per il dominio dell'applicazione
  - una CDN può anche aggiornare periodicamente i propri contenuti



## \* Discussione

- La scalabilità è la capacità di un sistema di gestire volumi di elaborazione crescenti
  - i principi fondamentali di progettazione per la scalabilità sono la scalabilità orizzontale (la distribuzione del carico e il cubo della scalabilità), la comunicazione asincrona e i servizi stateless – nonché il caching
  - la scalabilità non va considerata in isolamento – piuttosto bisogna considerarla insieme anche ad altre qualità – come le prestazioni, la disponibilità, la consistenza dei dati, il monitoraggio e la sicurezza – ed identificare e valutare le possibili sinergie e gli eventuali compromessi