Luca Cabibbo

Architettura dei Sistemi Software

Scalabilità

L'enfasi del corso è l'architettura a microservizi, che supporta nello specifico: Modificabilità, disponibilità, 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 Efesto, 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

Un sistema software viene progettato per un certo carico (progettazione delle prestazioni, voglio n elaborazioni in m millisecondi) ma si prospetta anche un aumento atteso progressivo di carico. La scalabilità riguarda la capacità del sistema software di gestire tali volumi crescenti, se richiesti, senza andare a scapito di altre qualità come le prestazioni (più tempo) e la disponibilità (elevata latenza all'interno del sistema può manifestarsi all'esterno come discontinuità di servizio)

- **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

Non ad intermediari (esempio:prima per fare biglietti aerei ci si doveva recare in un'agenzia che utilizzava servizi appositi per un numero alto ma finito di agenzie, adesso invece l'accesso è diretto ai 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

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

Ragionamento importante

Immaginiamo che il tipo di carico siano delle RICHIESTE. Ogni risorsa va elaborata e per farlo necessita risorse hardware (memoria, uso del processore, banda, numero di accessi al disco). Il software è dimensionato in modo che il carico previsto (la somma delle richieste) non ecceda le risorse hardware disponibili. Quello che succede in realtà è che per ogni risorsa c'è in ogni momento una percentuale di utilizzazione. Se questo livello aumenta e supera una certa soglia, le prestazioni dell'intero sistema iniziano a peggiorare. Quando supera una seconda soglia più alta il peggioramento non è più lineare ma esponenziale. Queste soglie non sono del 100%, ma del 25/30%.
Quando l'utilizzo di risorse aumenta possono verificarsi dei time-out nell'accesso ai dati, quindi delle eccezioni non gestite, e a quel punto un interruzione di servizio. Quindi all'aumento di richieste possono insorgere problematiche di accesso sia a risorse hardware che a risorse fisiche (es accesso a basi di dati, connessioni in generale..).

Per risolvere questo problema non si possono semplicemente aggiungere risorse hardware, perché anche il software va progettato in modo da utilizzare e gestire una certa configurazione di hardware (es utilizzo dei software, dei thread, dei nodi…)

← 13m50'

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

Gestione delle sessioni di lavoro, allocazione di risorse per tali sessioni

Ci occupiamo di questi aspetti

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 riguarda la capacità di gestire incrementi di carico transienti in modo che quando si verifica l'incremento riesco ad usare risorse aggiuntive, quando poi il carico diminuisce di nuovo smetto di usarle così da mantenere costi ridotti

- 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

* Progettare per la scalabilità

- La trattazione che facciamo sulla progettazione per la scalabilità è basata soprattutto su The Art of Scalability [TAoS]

Non segue la progettazione su pattern ma propone delle tattiche per supportare scalabilità

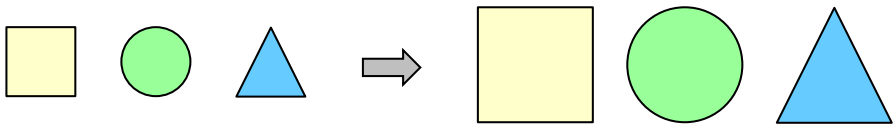
  - [SSA] tratta la scalabilità nel contesto più ampio della prospettiva per le prestazioni e la scalabilità – progettare per le prestazioni è un prerequisito importante per la scalabilità

Basato su prospettive architetturali

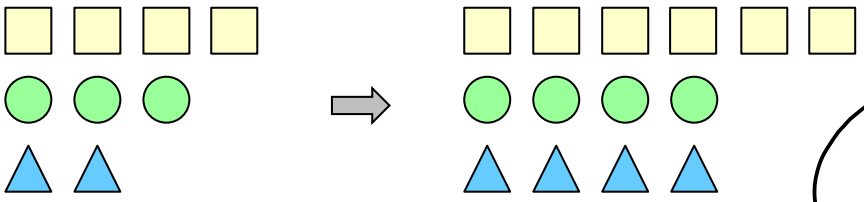
  - la scalabilità è correlata anche alla disponibilità – alcune tattiche per la disponibilità sostengono anche la scalabilità
  - ci concentriamo soprattutto su principi e regole di progettazione più specifiche per la scalabilità

- Scalabilità orizzontale e verticale

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



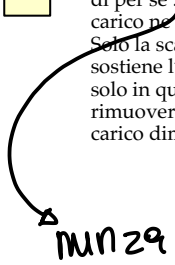
  - *scalabilità orizzontale (scaling out)*



La progettazione per la scalabilità richiede comunque che per un aumento di carico vengano aumentate le risorse e quindi per esempio l'hardware messo a disposizione.

La scalabilità verticale prevede che all'aumento di carico di abbiano nuove e diverse, più grandi risorse hardware che vanno a sostituire le precedenti. Nella scala i verticale quindi io acquisisco hardware più potente e lo sostituisco al precedente. In questa scalabilità sono vincolato dalla legge di Moore, quindi se il mio incremento del carico è sotto quel raddoppio entro 18mesi allora ok, se è maggiore non mi aiuta.
Questo tipo di scalabilità è utile per sistemi che non scalano in maniera lineare

Nella scalabilità orizzontale uso delle risorse che sono già di per se .. e all' aumentare del carico ne aggiungo di altre. Solo la scalabilità orizzontale sostiene l'elasticità, in quando solo in questo caso posso rimuovere risorse quando il carico diminuisce.



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

Cioè non basta avere le risorse ma ho bisogno di definirne l'architettura, e quindi progettare il sistema, soprattutto i vari scenari che mostrino come vengono utilizzate le risorse aggiuntive

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

Ordine di componenti, allocazione di spazio, ecc

La scalabilità verticale è comunque usata in sistemi in cui la comunicazione sincrona è fondamentale, come ad esempio nelle basi di dati relazionali che infatti scalano al più su una manciata di nodi

- 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

Capacità residua

- 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 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

Il software deve essere progettato per poter sfruttare l'aumento di disponibilità a seguito dell'aumento di carico

- 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"

    Il lavoro da svolgere non deve essere concepito come un lavoro monolitico, altrimenti può riguardare un solo lavoratore. Vogliamo invece decomporre il lavoro in compiti che possono essere distribuiti e assegnati
  - 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"

    Servizi che il sistema deve erogare — Dati persistenti e dati riguardanti le sessioni
  - 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

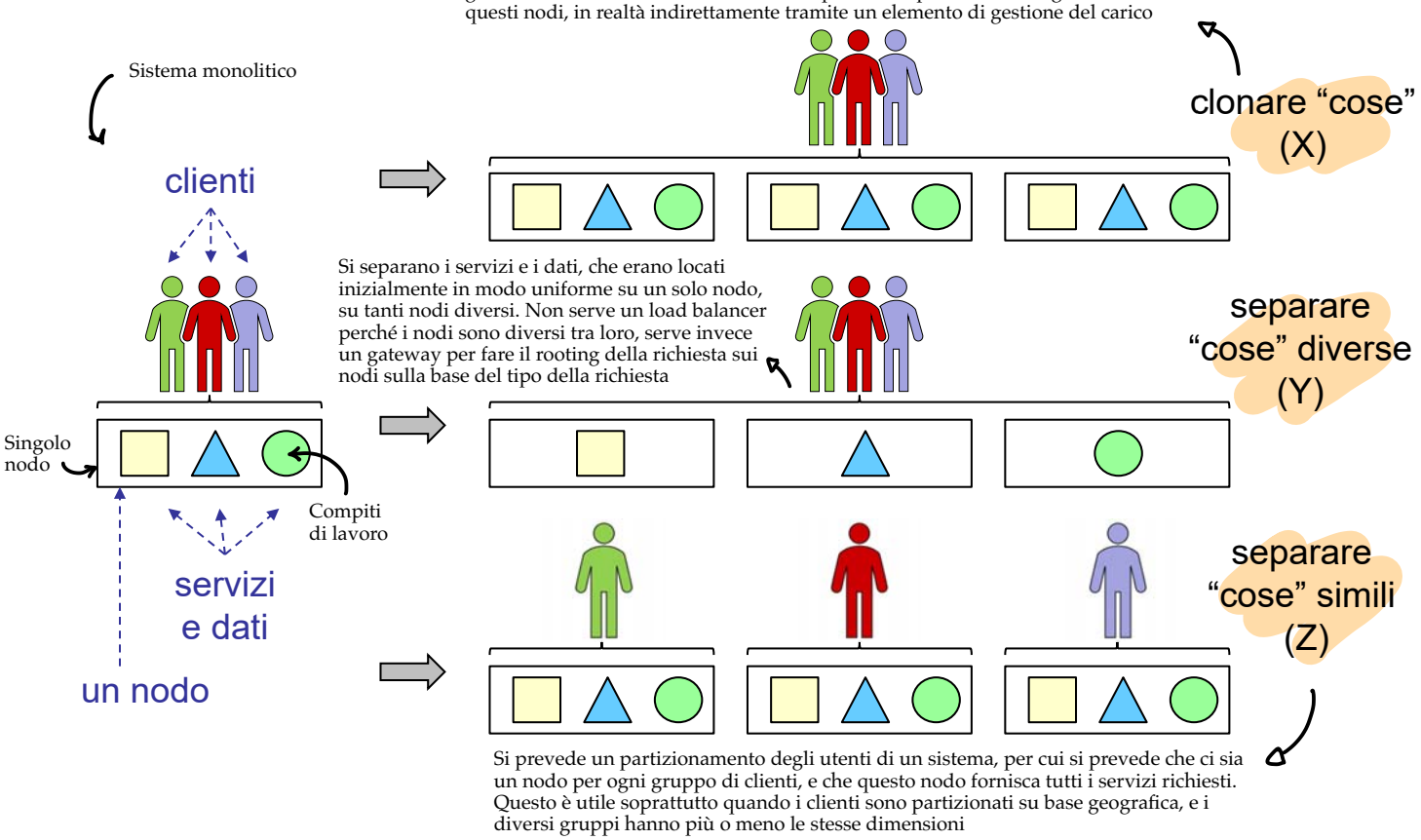
- Cubo della scalabilità

- 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

    ↳ Ortogonali tra loro
  - 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])

Cubo della scalabilità

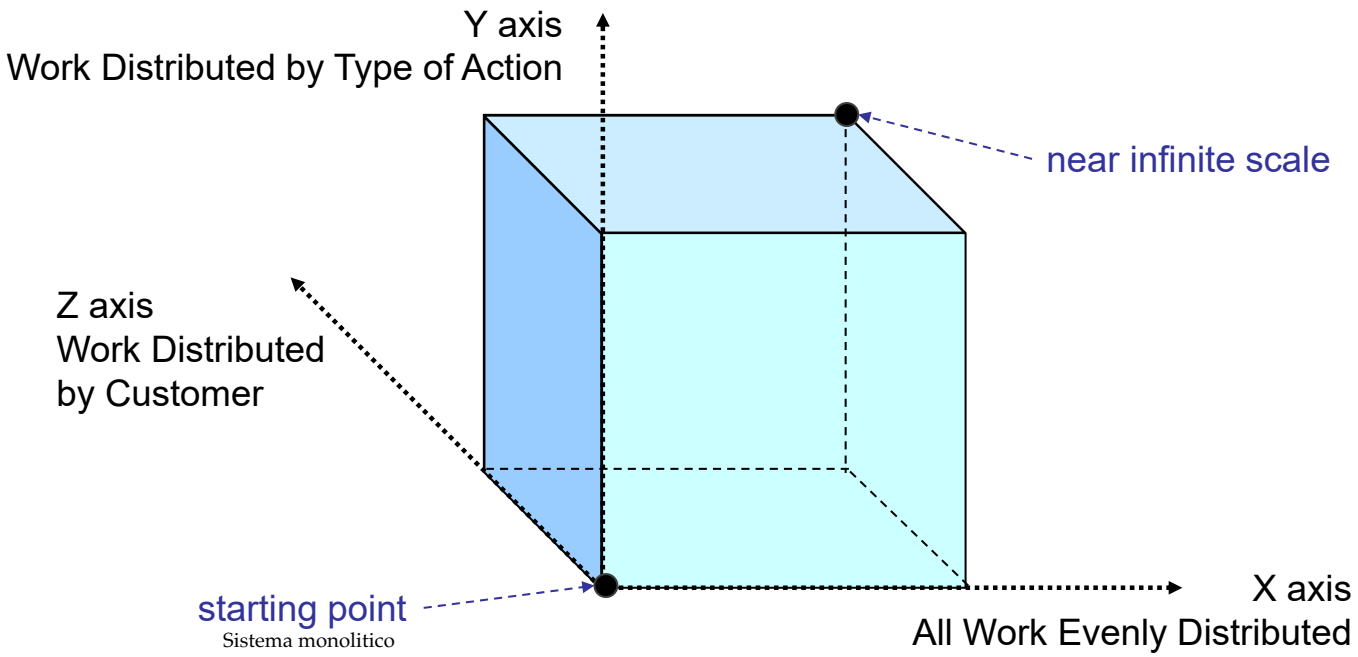
Anziché un singolo nodo ne vengono utilizzati tanti, ognuno dei quali offre tutti i servizi e gestisce tutti i dati con una clonazione replicazione perfetta, e tutti gli utenti accedono a tutti questi nodi, in realtà indirettamente tramite un elemento di gestione del carico



Si prevede un partizionamento degli utenti di un sistema, per cui si prevede che ci sia un nodo per ogni gruppo di clienti, e che questo nodo fornisca tutti i servizi richiesti. Questo è utile soprattutto quando i clienti sono partizionati su base geografica, e i diversi gruppi hanno più o meno le stesse dimensioni

Cubo della scalabilità

L'aspetto fondamentale è che dopo aver decomposto lungo un asse si può decomporre lungo un altro asse, aumentando quindi la scalabilità



Asse X del cubo – clonare "cose"

Abbiamo già visto: Maintain multiple copies of computation funziona quindi per disponibilità, scalabilità, prestazioni

- 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

    Non c'è decomposizione dei dati
    - è necessario anche un meccanismo di replicazione dei dati

La scrittura quindi deve essere fatta su tutti gli n nodi, le scritture non scalano

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

  Vengono preferite decomposizioni basate sui servizi e meno quelle basate sui dati. A volte una decomposizione del primo tipico implica una partizione dei dati, ma questa partizione non è perfetta in quanto alcuni servizi potrebbero avere necessità di accedere a più dati ti tipo diverso. I servizi sono PARTIZIONATI, i dati sono DECOMPOSTI
  - 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

Nota che mentre nella decomposizione lungo X aggiungere un nodo è facile, nella decomposizione lungo Y è difficile perché comporta la modifica e decomposizione di un servizio

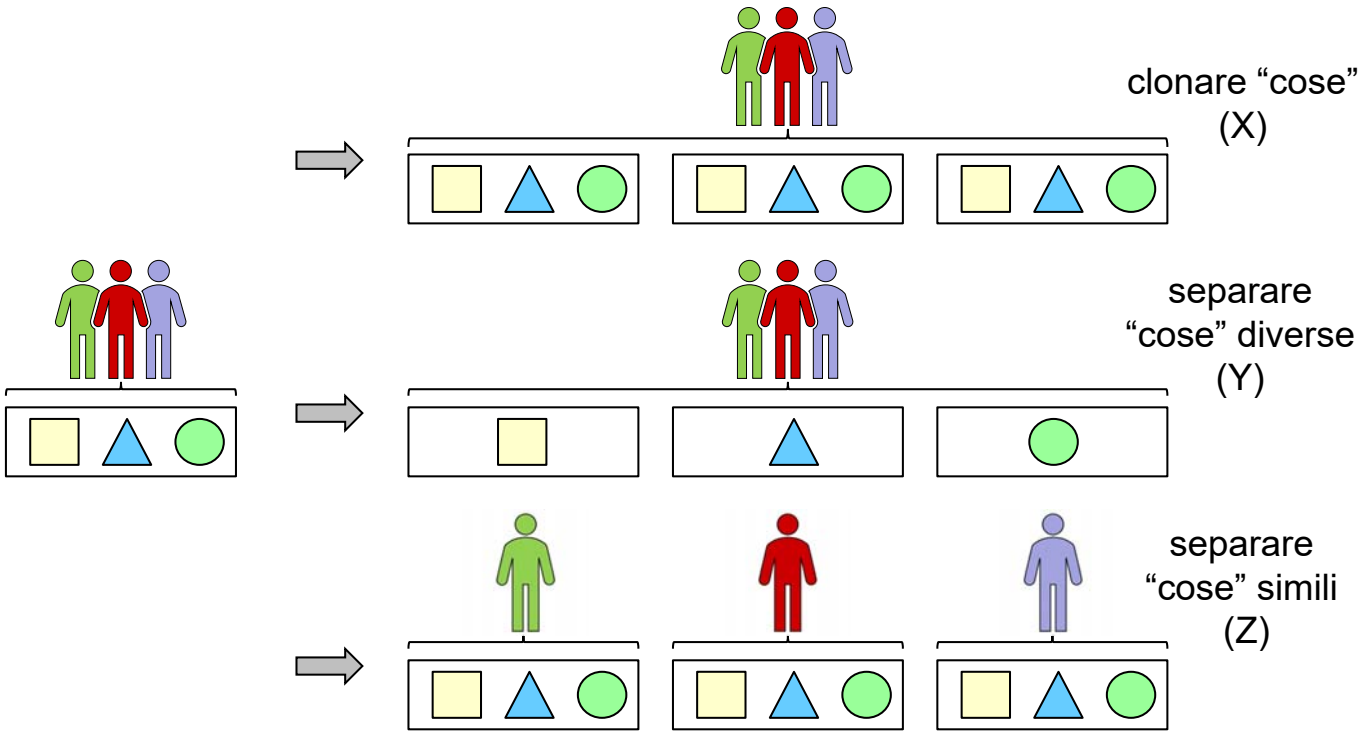
Asse Z del cubo – separare “cose” simili

- 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

    La partizione potrebbe essere anche sulla base di altro, ad esempio decomposizione a seconda che il tipo di vendita sia di libri o di elettronica. I servizi “al di sotto” sono gli stessi, è come decomporre sulla base delle intenzioni di acquisto dei clienti
    - 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

Anche qui non è facilissimo aggiungere nodi

Cubo della scalabilità



- 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

Quindi rivediamo i tre assi sulla base del tipo di incremento, con attenzione al supporto della disponibilità (soprattutto nelle accezioni di tolleranza ai guasti e isolamento dei guasti)

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

La replicazione può essere sincrona o asincrona

1h07m30s → 1h11m30s

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 Non c'è tolleranza dei guasti dei nodi interni

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

La decomposizione dei dati non è una partizione come abbiamo già visto perché alcuni dati potrebbero essere di interesse su più servizi e quindi su più nodi

  - 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

Se si rompe il nodo che gestisce il servizio Y posso comunque utilizzare il servizio X se esso è gestito su un altro nodo

  - 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 Nel caso di microservizi questo potrebbe essere un lato positivo
  - di per sé, non sostiene la tolleranza ai guasti
  - sostiene l'isolamento dei guasti solo in modo parziale
  - non sostiene la scalabilità elastica

Non c'è tolleranza ai guasti perché il nodo per un singolo servizio non è replicato e quindi se è down non posso accedere a tale servizio

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 Tra zone diverse
  - sostiene la scalabilità nel volume delle transazioni
  - sostiene la scalabilità nel numero dei clienti È l'unica forma di partizionamento che si occupa del numero di clienti

Asse Z per la scalabilità

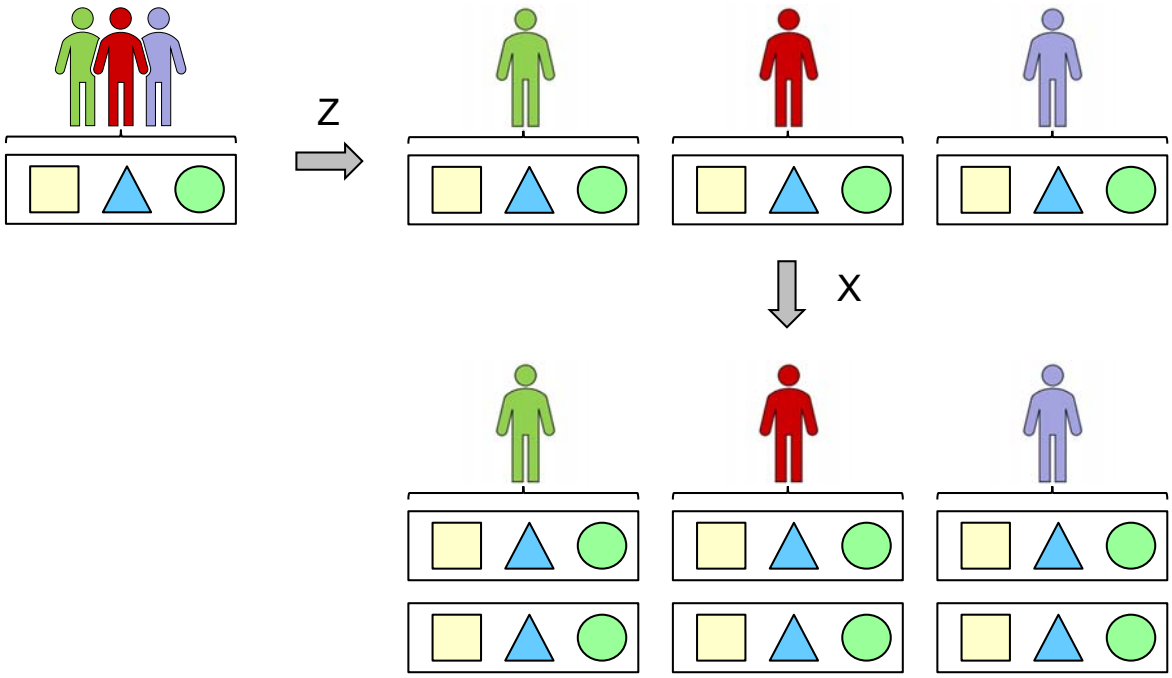
- Inconvenienti
  - è una soluzione complessa e costosa da implementare
  - i nodi sono simili tra di loro – ma hanno configurazioni differenti
  - può essere difficile cambiare il criterio di partizionamento – ad es., per decomporre il sistema su un numero maggiore di nodi
  - sostiene l'isolamento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

- Decomposizione lungo più assi

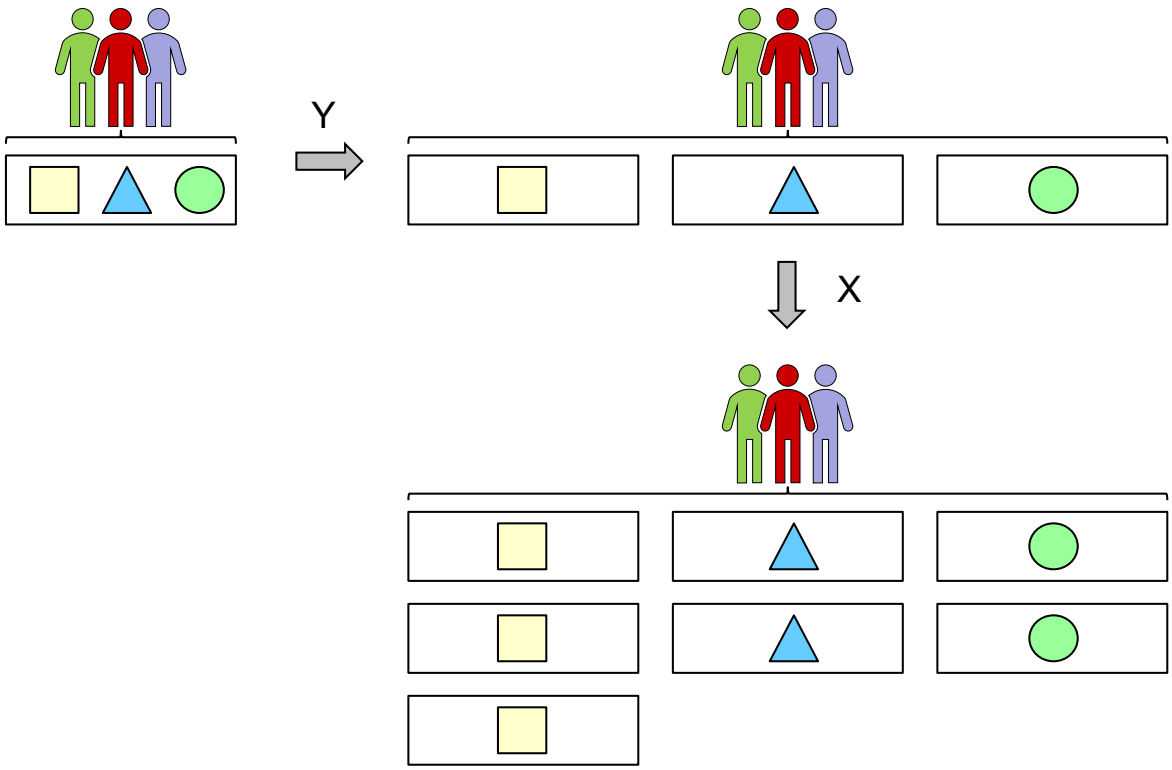
Questo perché le varie decomposizioni hanno benefici e inconvenienti abbastanza complementari

- 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

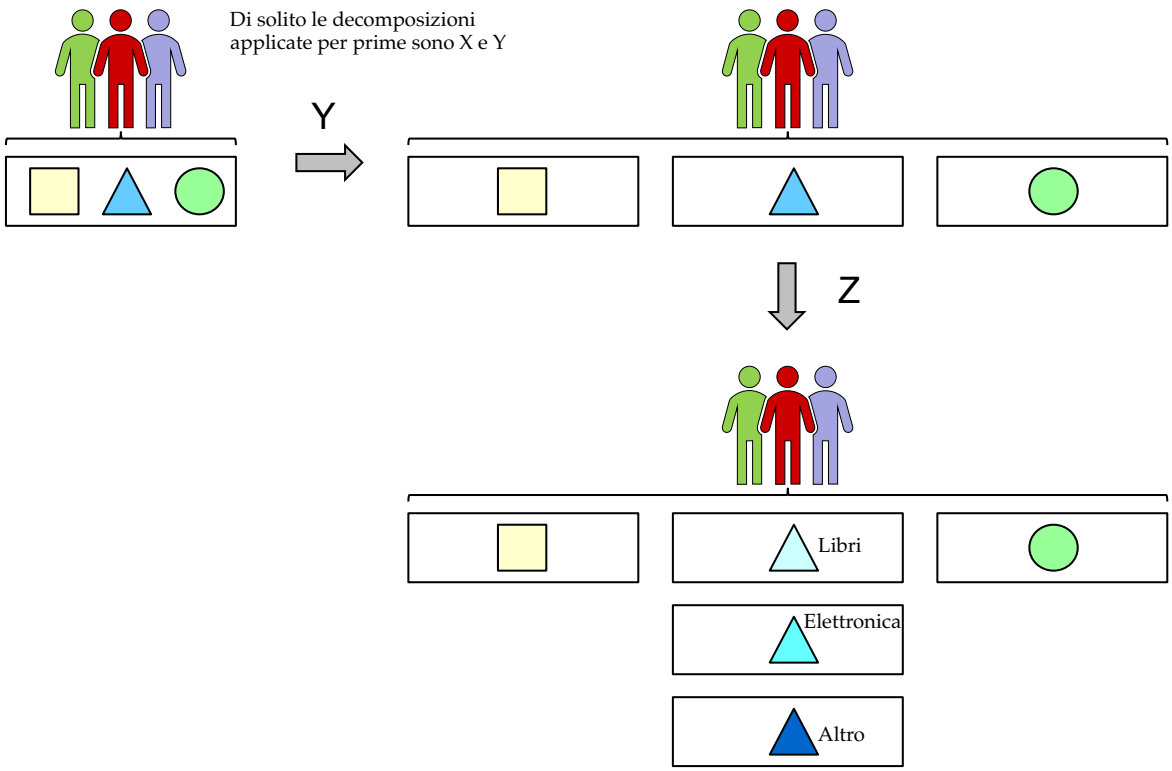
Decomposizione lungo più assi



Decomposizione lungo più assi



Decomposizione lungo più assi



- 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

Nota che una separazione della base di dati dai servizi è una separazione di tipo Y, anche se poi vedremo che una separazione così netta può non essere la migliore

- Comunicazione asincrona e scalabilità

Comunicazione sincrona: quando qualcuno dice qualcosa a qualcun altro, in quel punto quei due elementi sono sincronizzati e quindi entrambi sanno cosa sta succedendo.
Comunicazione asincrona: solo una delle due parti sa cosa sta succedendo, mentre l'altra parte non sa che la prima sa cosa sta succedendo

- 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) Cioè è importante che le cose vengano fatte in un certo ordine (es login, verifica dati, visualizzazione dati)
  - ad es., il login
  - ma non sempre è necessaria una sincronizzazione stretta
    Cioè non deve essere necessariamente applicata a TUTTI i compiti

Comunicazione sincrona e asincrona

- La comunicazione in un sistema software distribuito può essere basata su
  - *chiamate sincrone* (*request-reply*) Invocazione remota
  - *comunicazione asincrona* (*send-and-forget*) Scambio di messaggi
- Le chiamate sincrone possono avere un impatto negativo sulla scalabilità L'analisi osserva soprattutto sull'UTILIZZAZIONE DELLE RISORSE, che sono la cosa "scarsa" che dobbiamo dosare
  - 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

    Soprattutto nel caso di catene lunghe di componenti, altrimenti (con comunicazione sincrona) si rischiano catene di attese e quindi di avere risorse bloccate (in attesa) e quindi non utilizzabili attivamente

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

    All'interno dello stesso nodo le chiamate sincrone sono chiamate locali e quindi non sono un problema
  - alcune opzioni da usare quando un'operazione deve comunque restituire dei risultati

    Cioè non necessariamente l'unica opzione è quella della messaggistica, l'importante è limitare la sincronizzazione, e ci sono alternative:
    - 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)

    La sincronizzazione supporta la consistenza
  - 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*

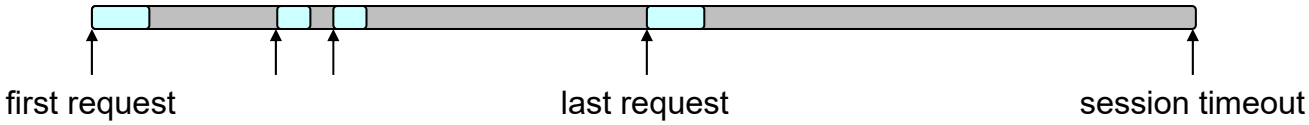
- Stato dei servizi e scalabilità

- Un servizio è *stateful* se l'esecuzione di un'operazione dipende dallo stato della conversazione (o sessione) con il client che ha richiesto l'operazione – altrimenti il servizio è *stateless*
  - se un servizio è stateful, allora il sistema deve gestire (da qualche sua parte) lo stato delle sessioni con i propri client
  - anche l'utilizzo di servizi stateful può avere un impatto negativo sulla scalabilità
  - quando la scalabilità è importante, i servizi dovrebbero essere preferibilmente stateless

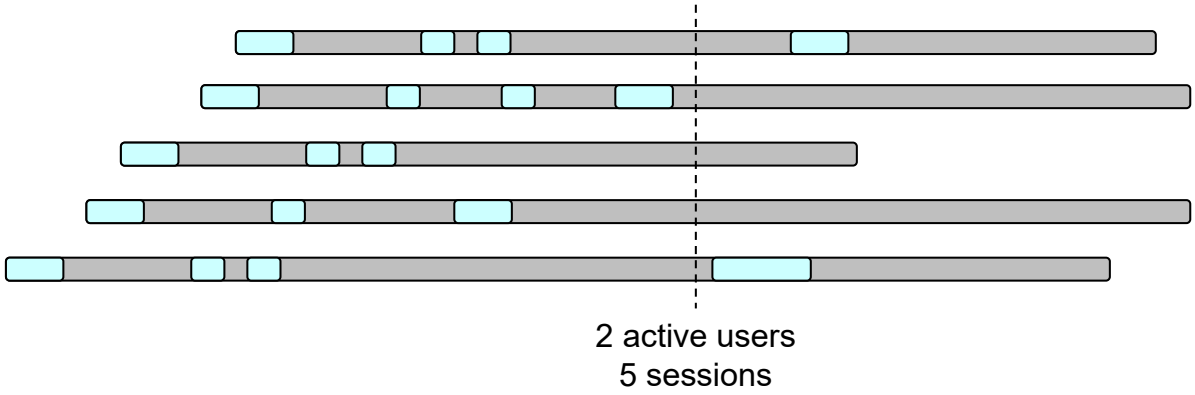
Altro principio di progettazione riguarda il preferire servizi stateless rispetto ai servizi stateful. In questo caso si fa riferimento non alla parte persistente dei dati ma alla parte di gestione delle sessioni. L'intuizione è che il comportamento dei clienti durante una sessione è tendenzialmente questo: l'utente fa alcune operazioni, separate tra loro da qualche tempo di attesa che può variare, dopo di che la sessione scade autonomamente molto tempo dopo di quando l'utente non abbia fatto l'ultima operazione. Il punto è che lo stato della sessione deve essere mantenuto dal sistema per tutta la sessione che è tendenzialmente un tempo molto maggiore del tempo in cui queste informazioni sono effettivamente necessarie e utili. Siccome mantenere lo stato della sessione richiede risorse (memoria), mantenerlo non è ottimale, e questo è vero soprattutto quando ci sono molti clienti diversi

Stato dei servizi e scalabilità

- Comportamento di un utente nell'ambito di una sessione



- Utenti e sessioni concorrenti



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

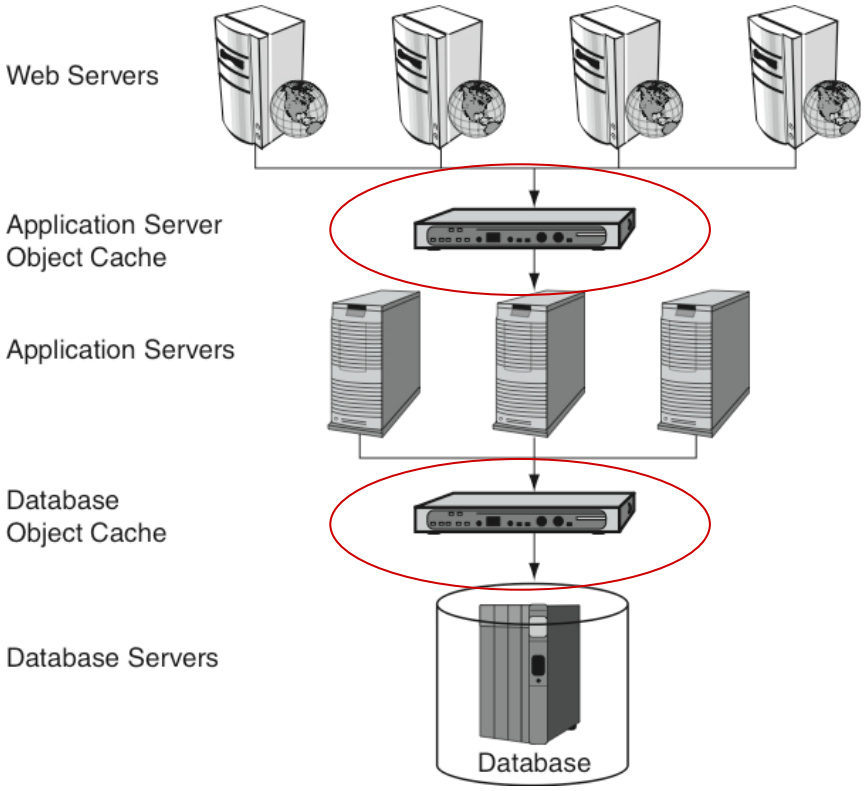
- Caching per la scalabilità

- 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 riusati
  - 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

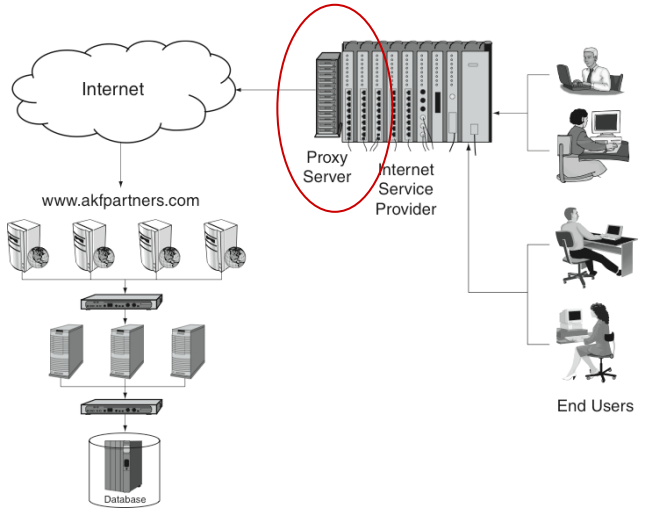


Application cache

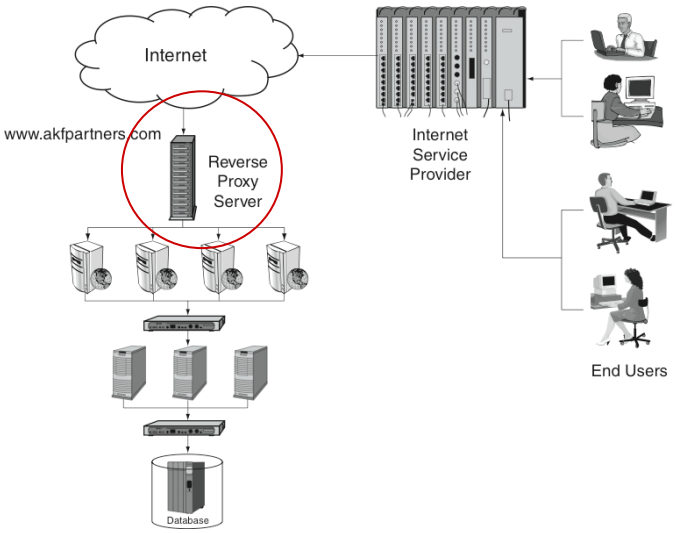
- Una *application cache* è una cache posta davanti all'intera applicazione – tra l'utente e l'applicazione
  - di solito memorizza richieste-risposte HTTP
  - esistono molti tipi di application cache – i due tipi principali sono
    - *proxy cache* (o *forward proxy cache* o *proxy server*)
    - *reverse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