Università degli Studi di Bologna
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Scuola di Ingegneria

Corso di **Reti di Calcolatori M**

Modelli di replicazione e di supporto alla tolleranza ai guasti

Antonio Corradi

Anno accademico 2014/2015

Replicazione - Guasti

modelli e definizioni di guasto

failure	qualunque comportamento diverso da quello previsto dai requisiti (**fallimento**)
error	difetto che genera un mal comportamento o *failure*
fault	comportamento nel sistema che può causare *errori* (**guasto**)

Programma che sbaglia e aggiorna un database in modo sbagliato
fault la causa a monte, l'**errore** la causa concreta
che generano l'effetto visibile **failure**

fault ⇨ **transienti, intermittenti, permanenti**

Bohrbug errori ripetibili, sicuri, e spesso facili da correggere

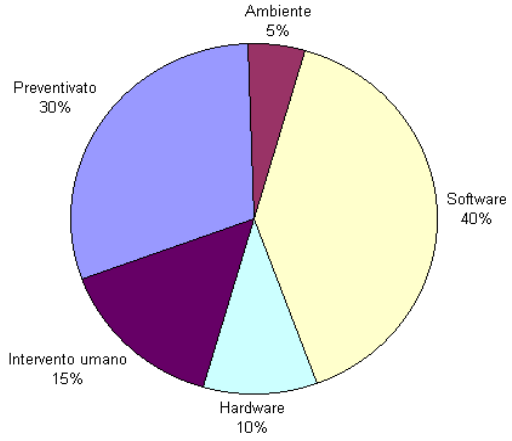
Eisenbug errori poco ripetibili, difficili da riprodurre e difficili da correggere

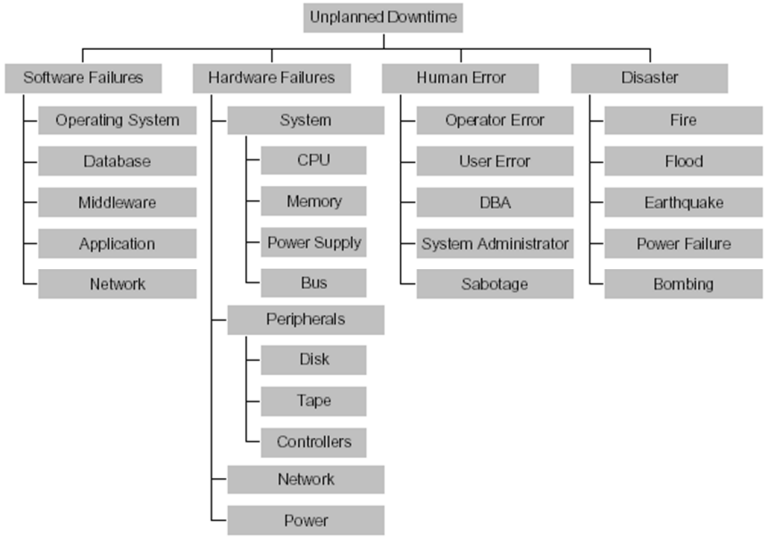
Gli Eisenbug sono spesso legati a problemi transienti e quindi molto difficili da eliminare

NON DISPONIBILITÀ di SERVIZIO

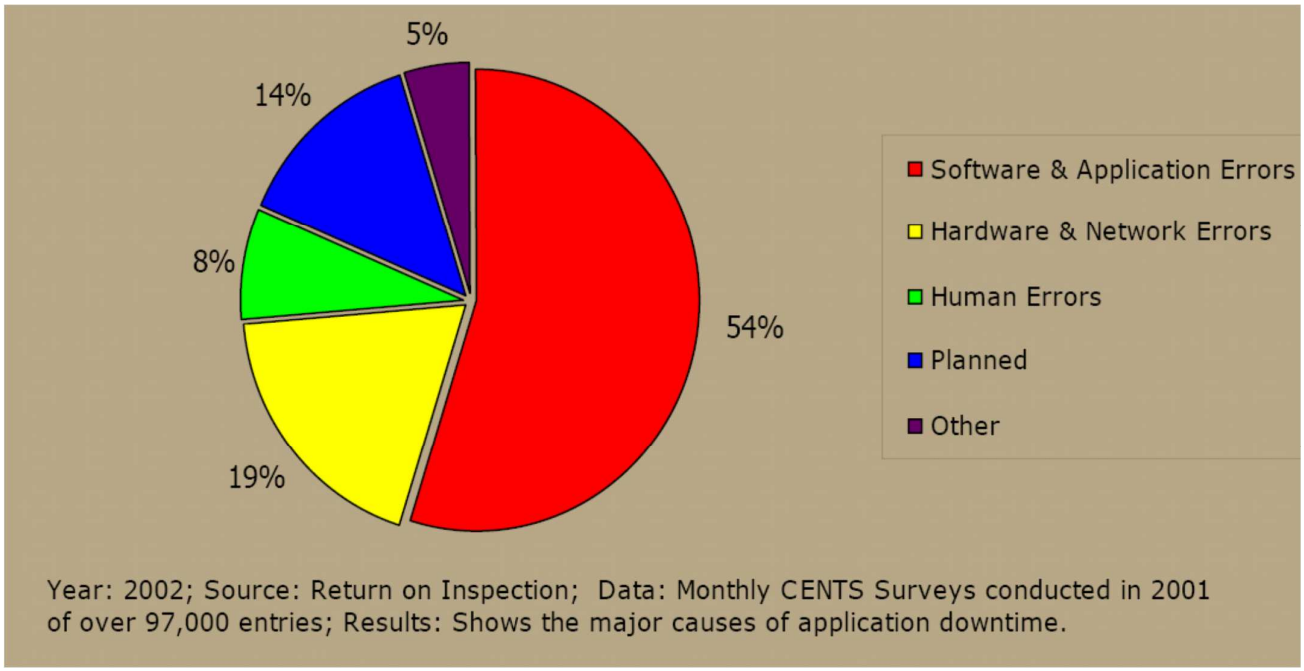
Un sistema può non essere disponibile per molte ragioni diverse, preventivate o meno

Fasi di **IDENTIFICAZIONE** del **guasto** e
di **RECOVERY** per ritornare alla normale operatività





CAUSE di DOWNTIME



NON DISPONIBILITÀ di SERVIZIO

Se un sistema si guasta con una certa probabilità abbiamo dei tempi di non disponibilità (**downtime**) molto diversi

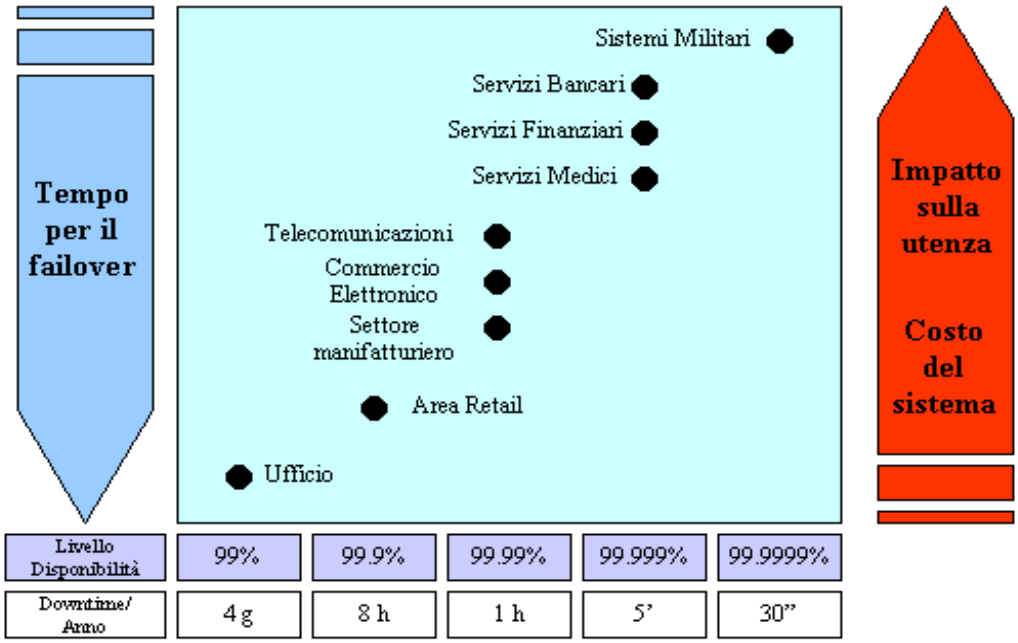
Spesso si indicano il ***numero di 9*** per misurare la disponibilità

Questi qualificano il tempo di servizio continuativo e dipendono non solo dalle occorrenze dei fault ma anche dalla prontezza della identificazione e recovery

Uptime (%)	*Downtime*	*Downtime*
98%	2%	7.3 days
99%	1%	3.65 days
99.8%	0.2%	17h,30’
99.9%	0.1%	8h, 45’
99.99%	0.01%	52,5’
99.999%	0.001%	5.25’
99.9999%	0.0001%	31.5”

TEMPI di RECOVERY

Ogni settore ha tempi diversi accettabili per la riparazione che dipendono dalla criticità del servizio



COSTI del GUASTO

Ogni settore ha **costi di downtime** molto differenziati e impatti diversi a secondo dell'interesse e la importanza del servizio

Anche se è difficile fare delle stime e valutare tutti gli effetti

Settore Industriale	Perdita/h
Finanziario (broker)	$ 6.5M
Finanziario (credito)	$ 2.6M
Manifatturiero	$ 780K
Retail	$ 600K
Aereo	$ 90K
Media	$ 69K

HA High availability – **CA** Continuous availability

Definizioni

DEPENDABILITY ***FAULT TOLERANCE***

possibilità di affidarsi al sistema nel completamento di quanto richiesto

sia in senso hardware, sia software, in generale

confidenza per ogni aspetto progettuale

RELIABILITY ***(affidabilità)***

possibilità del sistema di **fornire risposte corrette**

(accento sulla *correttezza* delle risposte)

es. disco per salvare dati ⇨ tempo di risposta?

AVAILABILITY ***(continuità o disponibilità)***

possibilità del sistema di fornire **comunque risposte in un tempo limitato** (accento sul *tempo di risposta)*

replicazione con copie attive e sempre disponibili

RECOVERABILITY ***(ripristino via persistenza dello stato)…***

Consistenza, Sicurezza, Privacy, ...

Identificazione e Recovery in C/S

Identificazione e **controllo** reciproci

Il cliente e il servitore si controllano a vicenda

- **il cliente aspetta la risposta dal servitore**
- **il servitore aspetta e verifica la consegna della risposta**
- **ripetizioni del messaggio e timeout**

Strategie di Identificazione e **recovery** dell'errore

Numero di errori che si possono tollerare

(insieme o durante il protocollo di recovery)

Numero di ripetizioni ⇨ numero di errori possibili

Ipotesi sui fault semplificano il trattamento

Ipotesi di guasto singolo

Ipotesi sui fault semplificano il trattamento

Ipotesi di Guasto Singolo (Un guasto alla volta)

il tempo di **identificazione** e **recovery** deve essere inferiore
(**TTR T**ime **T**o **R**epair)
al **tempo tra due guasti**
(**TBF** o **T**ime **B**etween **F**ailure o **MTBF Mean TBF**)

In sostanza, durante il recovery non si manifesta un altro guasto

Con **2** copie si **identifica un guasto**, si continua con la copia corretta
se il fault vuole dire bloccarsi (identificazione *via invariante*)
Con **3** copie si **tollera un guasto**, **due** sono **identificati**

Generalizzazione
con **3t** entità **copie**, si tollerano **t** fault per una risorsa
(**senza ipotesi sui fault**)

IPOTESI DI GUASTO PER PROCESSORI

Consideriamo processori e comunicazione

FAIL-STOP

un processore fallisce fermandosi (halt) e gli altri possono verificarne lo stato

FAIL-SAFE (CRASH o HALT)

un processore fallisce fermandosi (halt) e gli altri possono non conoscerne lo stato

FAILURE BIZANTINE

un processore può fallire esibendo un qualunque comportamento (vedi *generali bizantini e guasti maliziosi*)

IPOTESI DI GUASTO NEL DISTRIBUITO

SEND & RECEIVE OMISSION

un processore fallisce ricevendo/trasmettendo solo una parte dei messaggi che dovrebbe trattare

GENERAL OMISSION

un processore fallisce ricevendo/trasmettendo solo una parte dei messaggi che dovrebbe trattare o fermandosi

NETWORK FAILURE

l'interconnessione non garantisce comportamenti corretti

NETWORK PARTITION

la rete di interconnessione può partizionarsi, perdendo ogni collegamento tra le due parti

Replicazione come soluzione per realizzare componenti

OBIETTIVI

Reliability

MTBF **M**ean **T**ime **B**etween **F**ailures — disponibilità della risorsa

MTTR **M**ean **T**ime **T**o **R**epair — indisponibilità

Availability **A = MTBF / (MTBF + MTTR)**

intesa come percentuale di tempo di **lavoro corretto** (numero di 9)

Anche diversa per letture e scritture

se si hanno copie, la lettura può essere fornita anche se una o più copie sono non disponibili

Reliability probabilità di servizio disponibile in Δt

R (Δt) = reliable sul tempo Δt

R(0) = A, in modo generale come limite

Proprietà correlate

Proprietà formali

Correttezza Safety garanzia che non si verifichino **problemi** invarianti sempre rispettati

Vitalità Liveness raggiungimento obiettivi con **successo** vedi terminazione

Un sistema senza **safety** e **liveness** non garantisce niente per quel fault specifico (**nessuna tolleranza**)

Un sistema con **safety** e **liveness** maschera eventuali fault

Un sistema con **safety** e senza **liveness** opera **sempre correttamente dando risultati anche non accettabili in tempo di risposta**

Un sistema senza **safety** e con **liveness** opera fornendo un **risultato**, anche **non corretto** (di eccezione, ad esempio)

Soluzioni per entrambe Ridondanza in **spazio** o in **tempo**

ARCHITETTURE PER FAULT-TOLERANCE

Componenti replicati uguali

Si introducono costi aggiunti e possibili modelli di **esecuzione**

replicazione Hw

Esecuzione differenziata: copie o **tutte attive** o **meno**, sulla stessa o su più operazioni

Un solo componente esegue e produce risultato, altri backup

Tutti i componenti uguali eseguono e producono risultati diversi (max throughput)

Tutti i componenti eseguono e producono un unico risultato (max garanzia di correttezza: algoritmi diversi)

In queste architetture sono necessarie delle **parti** che controllino il **sistema** e che siano in grado di **riconfigurare** quando necessario (**metalivello**)

Memoria stabile

memoria stabile

uso di replicazione per ottenere certezza di non perdere informazioni (***uso di disco***)

Ipotesi di guasto limitative: *si considera di avere un sistema con una probabilità trascurabile di guasti multipli su banchi di memoria scorrelati*

In ogni caso, la probabilità di guasto durante un eventuale protocollo di recovery deve essere **minima**

Memoria con blocchi sicuri

ogni errore viene trasformato in **omissione** (*codice di controllo associato al blocco e blocco guasto*)

I blocchi sono associati in **due** copie distinte su dischi a *bassa probabilità di* ***errore congiunto***: le due copie hanno lo stesso contenuto di informazioni

Memoria stabile - SUPPORTO

Ogni **lettura** da un blocco **errato** viene trasformata in failure di omissione

Ogni **scrittura** procede su una delle copie e poi sull'altra

La operazione procede da una delle copie e poi dall'altra

Se uno dei blocchi fallisce, si tenta il recovery

In caso di necessità di **recovery**, si attiva un protocollo che ripristina uno stato consistente

se le copie sono uguali nessuna azione
se una sola scorretta, si copia il valore dell'altra
se diverse e corrette entrambe, si ripristina la consistenza

(se indicazione di tempo/ versione, si sceglie la più recente)

Costo elevato della realizzazione

Come limitare i tempi di risposta?

ARCHITETTURE FAULT-TOLERANT

Supporto, Risorse, e Protocolli necessari per la tolleranza

I protocolli impegnano le risorse
durata degli algoritmi
realizzazione degli algoritmi (affidabilità)

Il supporto (hw e sw) per la garanzia di dependability ed il protocollo di recovery devono essere più affidabile del resto *(come si garantisce?)*

Sistemi special-purpose ⇨ risorse ad hoc
Sistemi general-purpose ⇨ utilizzo delle risorse utente
contesa sulla stesse risorse

Principio di minima intrusione
limitare **l'impegno di risorse** (*overhead*) introdotto dai **livelli di supporto di sistema**

Questo principio deve essere rispettato da ogni parte del supporto che deve fornire funzionalità efficienti alle applicazioni ed ai servizi

*(*a parte sistemi ad altissimo costo e QoS*)*

COSTI di REPLICAZIONE elevati

Costi della replicazione di RISORSE molto elevati
e implementazioni in due direzioni e dimensioni

spazio sia in **risorse** a disposizione

tempo sia in **tempo** di risposta

Spesso le ipotesi di guasto più ampie rendono

ancora più difficile il progetto dei protocolli e

inaccettabile il costo delle soluzioni

Costi dipendenti da molti fattori diversi

costi di memoria

overhead di comunicazione

complessità di implementazione

cosa replicare, quante repliche, dove allocarle?, etc.

TREND per ottimizzare protocolli, supporti, infrastrutture

TANDEM

Sistema special-purpose

TANDEM (inglobato da Compaq e HP)

replicazione in doppio di ogni unità di sistema

(due processori, due bus, due dischi)

Obiettivo: sistema **fail-safe**

Un errore viene identificato attraverso la replicazione di CPU (doppio caldo)

Memoria su disco stabile con accesso attraverso uno di due bus a blocchi di memoria replicati

costo del sistema molto elevato

La presenza di copie **replicate** può prevedere

di fare le azioni in **doppio** o

di farle solo **una volta** e riportare le variazioni sulla copia (o sulle copie)

Costo elevato del sistema, sia per risorse hw sia sw

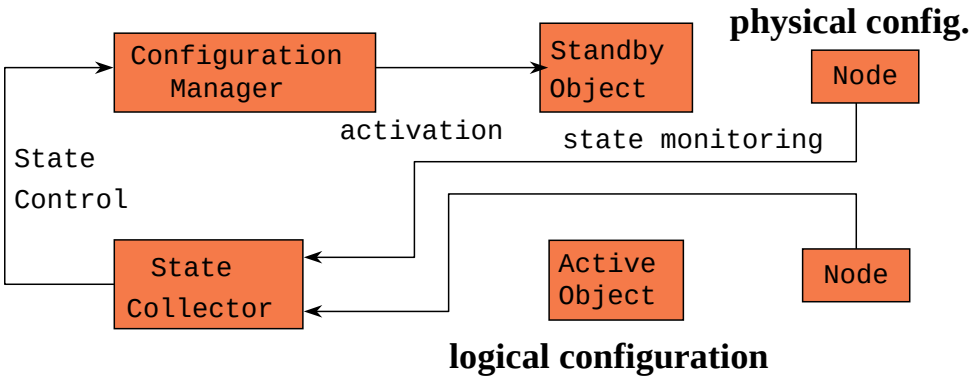
Sistemi stand-by: COLD STAND-BY

Copie fredde

Ogni oggetto implementato da una **sola istanza**

solo al fallimento si rigenera una **nuova istanza**

- Nessuna informazione di stato viene mantenuta tra attivazioni diverse di istanze
- Periodo di riconfigurazione elevato



HOT STAND-BY (copie calde)

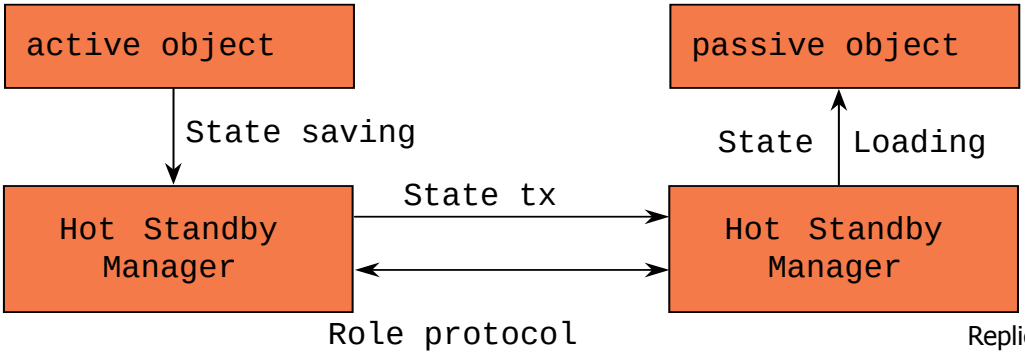
Un oggetto consiste di almeno **due istanze**

Una istanza esegue ed è *attiva*, l'altra è *passiva*

la copia **attiva** (master) esegue e quella **passiva** viene aggiornata

- *Se fallisce la copia attiva, la passiva viene promossa*
- *Se fallisce la copia passiva*

si usano copie passive di stand-by per fare una nuova copia passiva che ottiene lo **stato** dall'oggetto attivo (copia calda con stato)



Redundant Array of Inexpensive Disk

Sistemi general-purpose basati su replicazione su memoria di massa diventata comune

RAID **R**edundant **A**rray of **I**nexpensive **D**isk

insieme di dischi a **basso costo** ma coordinati in **azioni comuni**

per ottenerne diversi standard (di tolleranza ai guasti)

costo limitato

spesso considerati in sistemi commerciali

DISCHI RAID nati per striping, ossia per **velocizzare l'accesso ai file** memorizzati su più di un disco e poi

estesi a considerare la **replicazione**

Alcune classi di lavoro con ipotesi diverse di uso

RAID

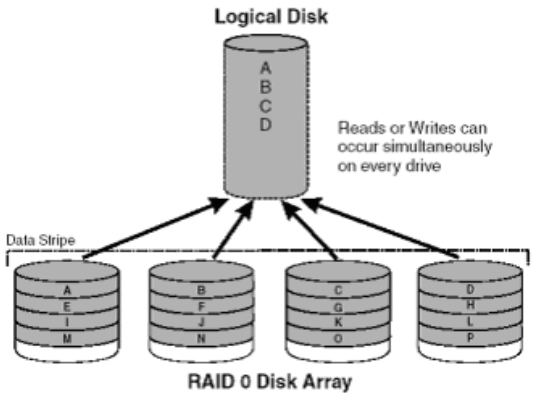
Raid 0: striping solamente

elevata velocità con I/O parallelo ma nessuna ridondanza,

peggior **MTBF**

adatto per applicazioni

con I/O intensivo

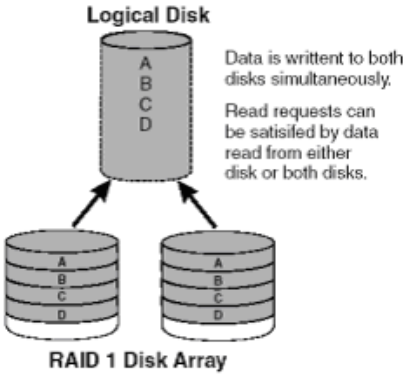


Raid 1: mirroring - ridondanza massima

alta availability

buone performance in lettura meno in scrittura

alto costo



RAID

Raid 3 e 4: striping con disco di parità dedicato

alta velocità per blocchi di grandi dimensioni (immagini)

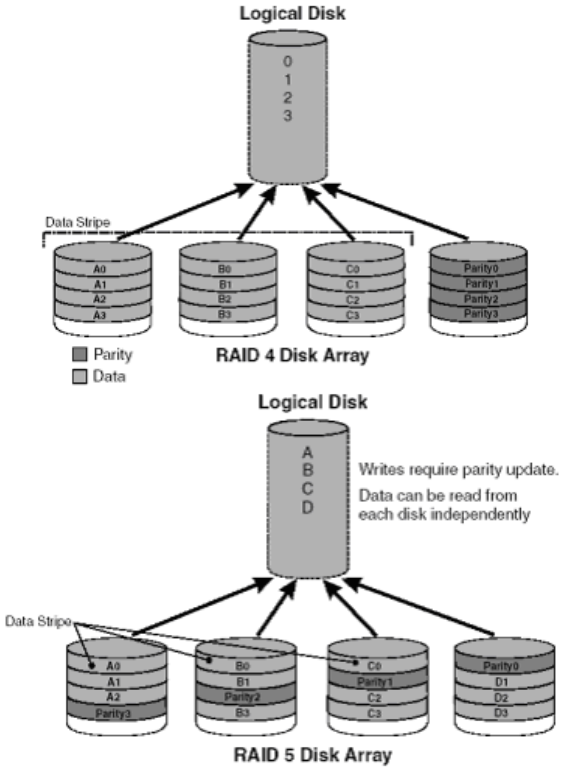
una operazione di I/O alla volta per la contesa sul disco di parità

Raid 5: striping senza disco di parità dedicato

La parità distribuita consente una buona velocità anche per molte letture di piccoli blocchi

Meglio se scritture di grossi blocchi

Disk LAN server



Gestione delle risorse

Possibilità di considerare risorse replicate in ambito distribuito con un coordinamento del tutto generale (software fault-tolerance)

risorse replicate

copie multiple delle risorse su nodi diversi con diversi gradi di replicazione

risorse partizionate

le risorse sono allocate in modo unico su nodi diversi (con qualche grado di replicazione)

Uso della replicazione delle entità rilevanti con QoS

dati e processi ***replicati***

ASTRAZIONE DI RISORSA UNICA

Grado di replicazione

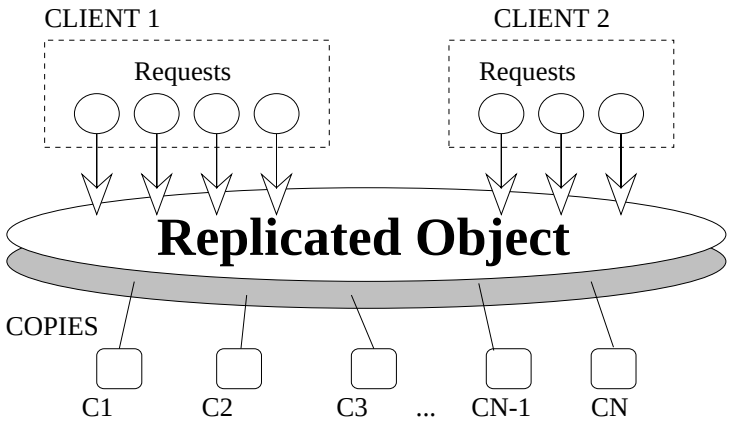
numero di copie della entità da replicare

maggiore il **numero** di **copie,** maggiore la **ridondanza**

migliori le proprietà di **affidabilità** e **disponibilità**

reliability & ***availability***

maggiore il costo e l'overhead



Due modelli estremi di architetture FT

- **Uno solo** esegue (master-slave)
- **Tutti** eseguono (copie attive e pari)

e variazioni

ARCHITETTURE PER LA REPLICAZIONE

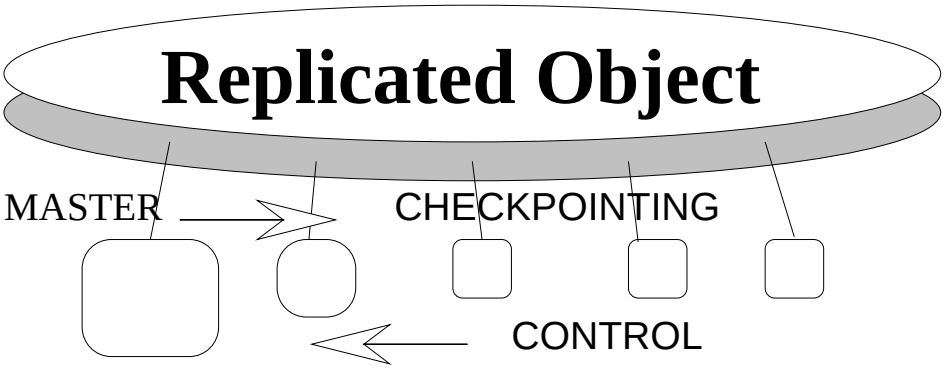
Modello Passivo *(anche Master-Slave)*

Una sola copia esegue, le altre sono solo di riserva

Questo modello è il primo modello di replicazione

*Il **master** è visibile all'esterno e fornisce i servizi e controlla gli slave*

Gli slave controllano il master per identificare guasti ed errori



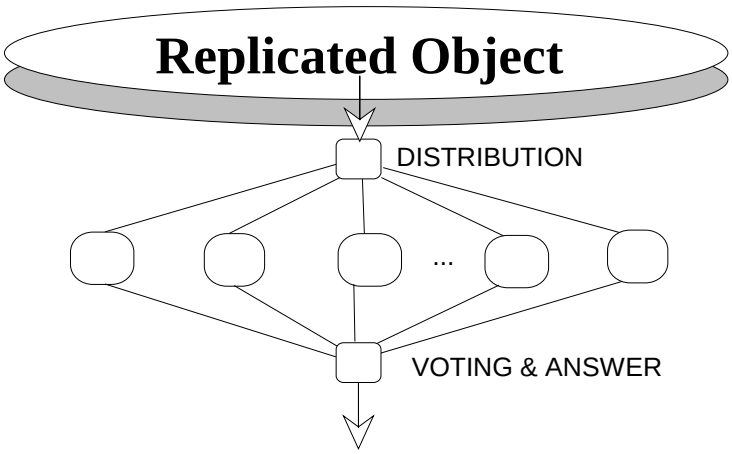
REPLICAZIONE ATTIVA

Modello Attivo

***Tutte le copie* eseguono in modo sincrono e con necessità di forme di coordinamento tra le copie**

Nella **TMR** (Triple Modular Redundancy) **tre** copie
si ovvia ad un errore e se ne identificano fino a due

Le diverse copie possono anche eseguire algoritmi diversi



MODELLO DI REPLICAZIONE PASSIVA

Modelli

Master Slave (modello passivo)

Copie attive (modello attivo)

modello passivo o **master/slave (primary/backup)**

Un **solo processo (attivo) esegue le azioni sui dati**

La altre copie (passive) servono in caso di guasto

*una sola copia corretta, le altre possono anche non essere aggiornate (***copie calde** o **fredde***)*

Possibile scollamento dello stato tra il **master** e gli **slave**

*In caso di **guasto**, si dovrebbe riprendere dall'**inizio***

Il master aggiorna lo stato degli slave (**checkpoint**)

Checkpoint - Aggiornamento degli Slave

checkpointing ⇨ *azione di aggiornamento* (anche a catena)

La **politica** di gestione di ogni sistema separa

Le azioni necessarie per garantire una risposta sicura:

aggiornamento copia **primaria**

dalle azioni successive

aggiornamento copie **secondarie** (***checkpointing***)

In base alla politica otteniamo diverse garanzie e diverso stato di aggiornamento delle copie

Il cliente ottiene una risposta con **tempi inferiori** se il **master fornisce la risposta prima di aggiornare le copie**

Altrimenti **maggiori garanzie**, ma **tempi di risposta aumentati**

Gestione con protocolli all'interno dell'oggetto

Master – Slave: CHECKPOINT

CHECKPOINT

azione di aggiornamento e di instaurazione del nuovo stato sugli slave

Azione periodica	**time-driven**
Azione scatenata da evento	**event-driven**

In caso di risorsa **sequenziale**,

lo stato è evidente e '**facile**' da identificare e instaurare

In caso di **risorsa parallela**, bisogna pensare a tutte le operazioni che possono essere in atto insieme

lo stato dovuto alle diverse **azioni contemporanee** è più difficile da identificare e instaurare

Maggiore complessità e problemi nel quando fare un checkpoint di più attività in parallelo in un oggetto

Master - Slave: recovery del guasto

Rilevazione del guasto: da parte di chi? quando?

Recovery del guasto

La copie secondarie devono identificare il guasto tramite una osservazione del master

uso di messaggi esistenti e assunzioni sugli intervalli di tempo

messaggi ad-hoc per stimolare risposte dal master

Può bastare

uno slave per il protocollo (***se guasto singolo***)

gerarchia di slave e protocolli molto più complessi (***se guasto multiplo***)

anche ***usando protocolli di elezione***

La **risorsa**, vista dall'esterno, tollera, a secondo delle strategie, un numero diverso di errori ed è in grado di fornire operatività in caso di errori (**fault transparency**)

MODELLO DI REPLICAZIONE ATTIVO

Copie attive - tutte le copie attive e corrette devono eseguire le operazioni

Un processo per ogni copia di dato esegue la operazione sui dati privati

Comunicazione del cliente con i servitori in modo

esplicito od **implicito**

Se il controllo del cliente ***esplicito*** ⇨ *manca astrazione*

Questo schema non viene considerato se non in sistemi ad-hoc → richiede la visibilità della replicazione

Se il controllo del cliente ***implicito*** ⇨ *ok trasparenza*

necessità di un supporto che passi le operazioni e le risposte in modo opportuno

Replicazione copie attive

Controllo **implicito interno alla risorsa**

o esiste un **solo gestore** (schema **statico**)

schema a **farm** centralizzato
e poi da lì si dirigono le richieste verso gli altri

o esistono **gestori** (schema **dinamico**)

un gestore diverso per ogni richiesta
il gestore dirige le richieste verso gli altri e controlla

Politica di scelta del gestore: decisione

Statica fissata

Dinamica per **località** / **a rotazione**

Facendo nascere la necessità di evitare interferenza di **gestori** che contemporaneamente vogliono iniziare una gestione

Replicazione copie attive

In genere, nei modelli attivi

perfetta sincronia

tutte le copie devono agire d'accordo, soprattutto in caso di risposta o di azioni verso l'esterno (innestamento)

approssimazioni alla sincronia

quasi tutte le copie devono essere in accordo,
ma le azioni possono procedere prima del completo accordo

Queste strategie **meno sincrone** sono meno costose

in realizzabilità del protocollo

in tempo di risposta per il cliente

Ma anche problematiche per ordine delle operazioni

in proprietà semantiche

I **moderni sistemi Cloud** escludono a priori la **perfetta sincronia** **per una sincronia all'infinito**

Coordinamento tra copie attive

A secondo delle azioni abbiamo politiche diverse

azioni di lettura possono avvenire in modo indipendente sulle diverse copie ed in parallelo

azioni di scrittura richiedono maggiore coordinamento tra copie

azioni di cambiamento di stato

richiedono maggiore coordinamento ma

In caso di stato partizionato, possono procedere contemporaneamente e indipendentemente

Fase di riconciliazione delle copie

azioni con semantica differenziata

Prendiamo le azioni su un direttorio,

aggiunta/eliminazione di file, scrittura e lettura di file, richiesta di listato del direttorio

Le operazioni non sono tutte uguali e possono anche tenere conto della semantica dell'oggetto per ottenere maggiore parallelismo

Aggiornamento delle copie attive

Ogni azione richiede di aggiornare lo stato di tutti

azione di aggiornamento deve avvenire prima di fornire il risultato all'esterno (peso sul tempo di risposta)

Su cui si devono considerare ritardi o guasti

Se si lavora con **gestori diversi** per ogni operazione è compito del gestore comandare le azioni degli altri

Se le **operazioni possono essere parallele** non ci devono essere problemi sulle decisioni dei gestori e su come le operazioni sono portate avanti (in che ordine)

Decisioni sulla durata dell'azione

*In caso di **guasto durante una operazione**, si deve dichiarare il guasto e potere continuare a fornire una qualche risposta*

Coordinamento tra copie attive

ACCORDO prima di dare risposta

Tutte le copie devono avere fatto l'azione (**massimo accordo**)

Voting (non tutte le copie) a ***maggioranza* (quorum) *pesati***

le copie in accordo possono continuare a lavorare correttamente

le copie non aggiornate sono in uno stato non utilizzabile immediatamente, ma solo dopo un reinserimento nel gruppo (recovery)

Rilevazione del guasto: da parte di chi? quando?
Rilevazione del reinserimento: da parte di chi? quando?

Necessità di monitoraggio e di **controllo esecuzione**

Semantica di gruppo

Si possono approssimare semantiche molto diverse anche nei confronti delle azioni che si attuano nel gruppo e degli ordini di esecuzione delle azioni

Modelli di REPLICAZIONE

Quale modello è più diffuso e comune?

Modello Master-Slave più semplice e con un'unica entità che esegue

Modello Copie Attive più complesso e con coordinamento

In termini di costo, è determinante il **numero di copie**, *che lavorino o meno*

Una analisi dell'installato esistente mostra un **grado di replicazione molto limitato** (non oltre qualche unità)

Si cominciano ad affermare anche modelli intermedi con

un insieme di risorse che sono in grado di eseguire gli ***stessi compiti* in modo indipendente**

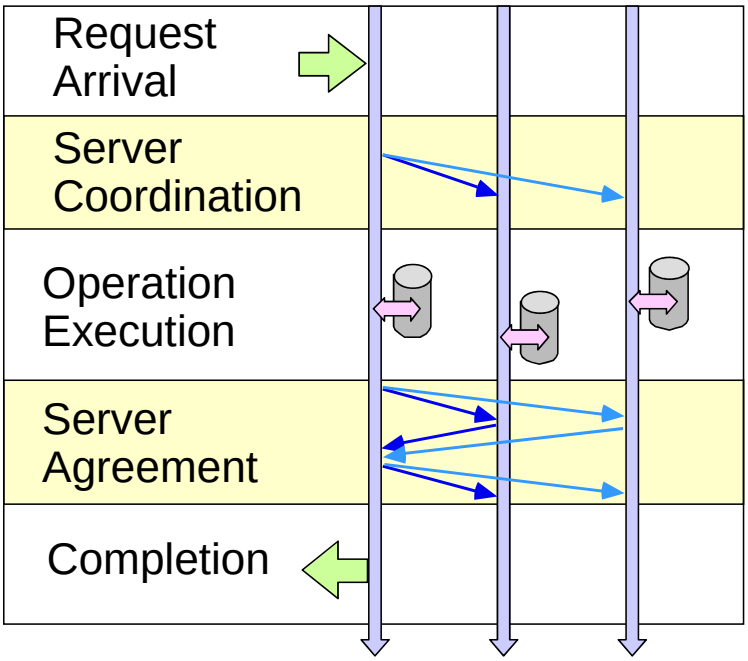
(*magari con divisione dei compiti*) ma possono fare da ***back-up*** una all'altra

Si può quindi ottenere bilanciamento di carico

OPERAZIONI su COPIE ATTIVE

Per un modello della corretta operatività delle copie, possiamo prevedere una sequenza di fasi:

1) arrivo richiesta
2) coordinazione copie
3) esecuzione
4) accordo finale
5) risposta al cliente



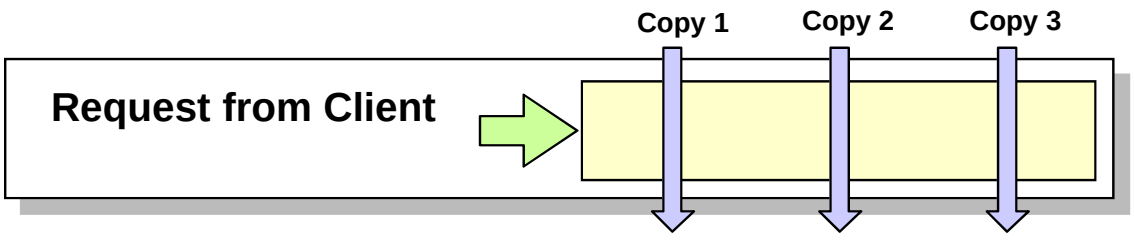
FASE 1: richiesta del cliente

Il cliente può inviare la richiesta

- *solo ad una della copie*
- *a tutte le copie*

In caso invii ad una sola copia, questa si deve fare carico di inviare a tutte le altre e di gestire la prima fase

Ruolo di una copia specifica deciso in modo dinamico/statico

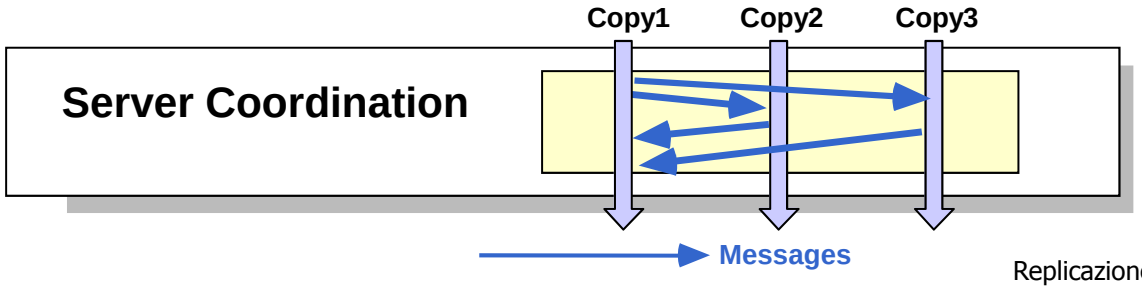


FASE 2: coordinamento server

Le **diverse copie** devono coordinarsi secondo una *politica precisa*

- Una **copia master** che si incarica di fare da gestore
- **Tutte le copie** svolgono la parte di esecuzione delle operazioni (senza ruoli)
- Le **copie possono avere pesi diversi** nel gruppo e contribuiscono in modo diverso

Prima fase di coordinamento



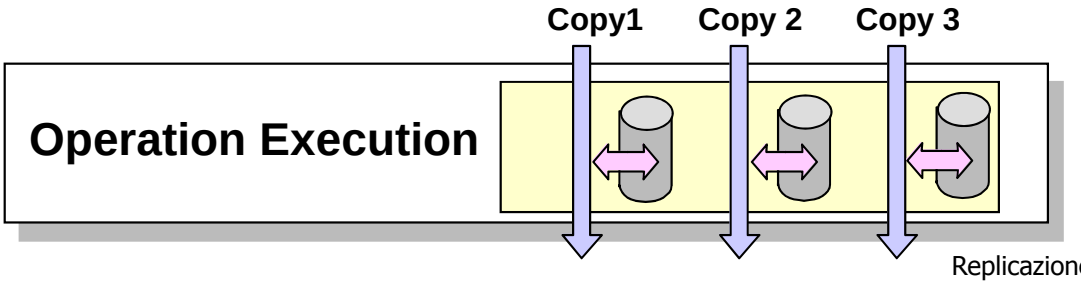
FASE 3: esecuzione nei server

La fase precedente influenza la esecuzione che le diverse copie devono completare

A secondo della politica decisa (anche per questa operazione)

- Una sola esecuzione o di tutte
- Esecuzioni differenziate e in cascata che si devono coordinare

Se uno solo esegue, modello master-slave



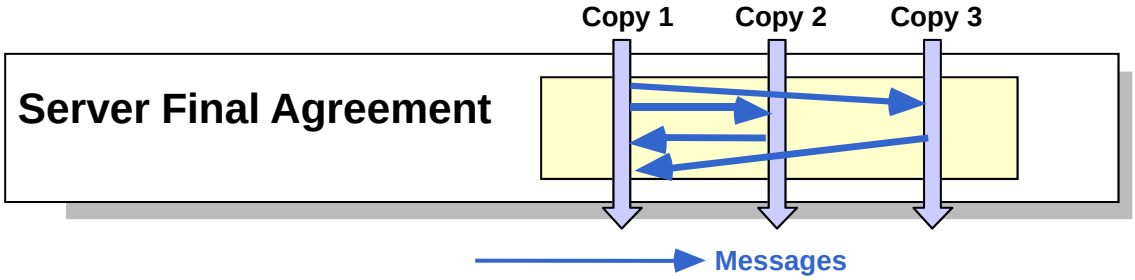
FASE 4: coordinamento server

Le diverse copie devono accordarsi sul risultato che la risorsa deve considerare ad operazione effettuata

Anche fare i commit dei dati se è il caso

Lo stato deciso definisce anche la correttezza delle copie ed esclude copie divergenti

Seconda fase di coordinamento

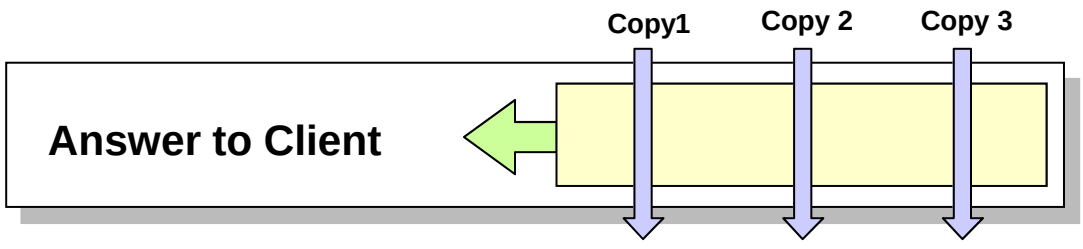


FASE 5: risultato al cliente

Fase di ricezione della risposta

Il cliente riceve **l'esito della operazione**

- Una sola **risposta unificata**
- solo **da una copia** (che ha interrogato)
- **da tutte le copie** (tutte le risposte)



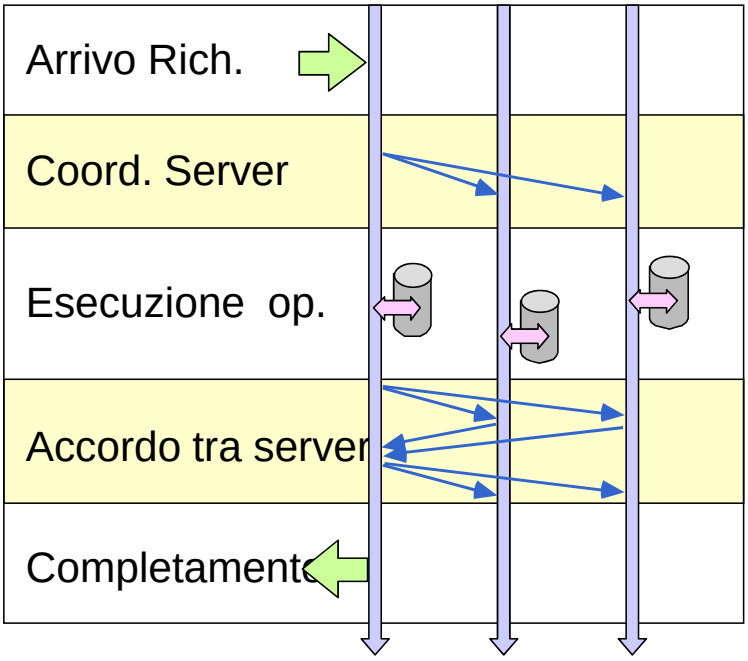
REPLICAZIONE COPIE ATTIVE

Le 5 fasi danno una idea della complessità della politica a copie attive

Il coordinamento delle diverse copie tende ad introdurre costi accettabili solo

per **gradi di replica-zione** limitati

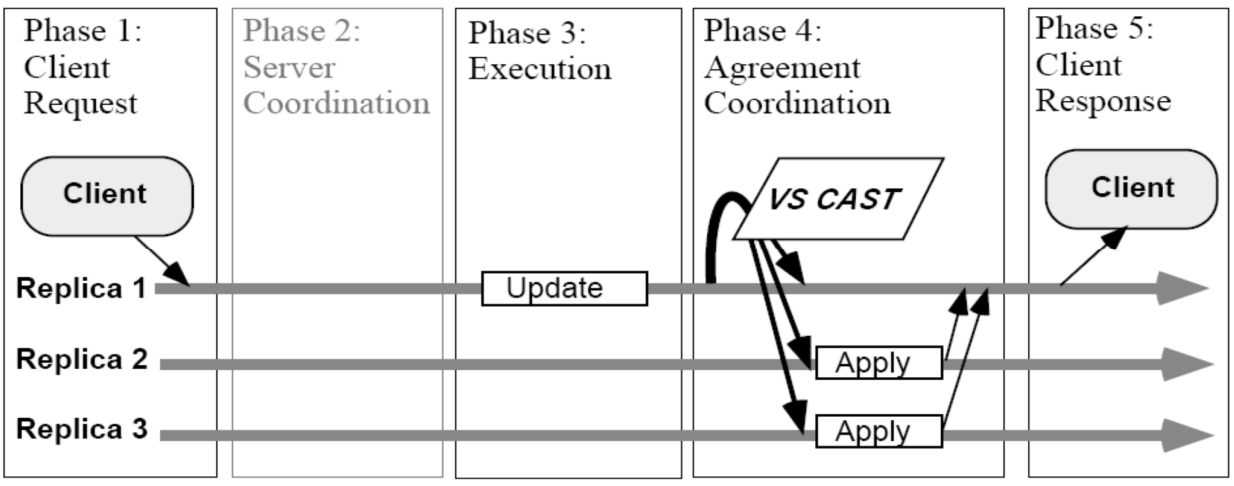
per **politiche di coordinamento semplici**



REPLICAZIONE COPIE PASSIVE

In caso di replicazione **master-slave** i costi sono ovviamente più limitati ed accettabili

La centralizzazione ed il ruolo master semplifica la gestione



POLITICHE DI AGGIORNAMENTO

In caso di replicazione di **risorse** (**dati**) possiamo considerare due proprietà fondamentali

- **ruolo di chi decide gli aggiornamenti**
  copia **primaria** o **tutte**
- **prontezza di propagazione degli aggiornamenti**
  eager (immediato e prima della risposta) pessimista o **lazy** (ritardato) e ottimista

possiamo anche vedere in modo opposto per il cliente

Per l'aggiornamento possiamo avere:

*Copia **primaria eager***

*Copia **primaria lazy***

*Aggiornamento **eager di tutte le copie***

*Aggiornamento **lazy di tutte le copie***

AGGIORNAMENTI COPIE ATTIVE

In caso politica **eager** tendiamo a privilegiare **la consistenza e la correttezza della operazione sul client**

Obbiettivo è di **non fornire risposte prec**oci ma anche potenzialmente da disfare, in caso di problema successivo

l'aggiornamento delle copie serve per il **coordinamento** e la **garanzia della consistenza** e si può prevederlo **in due fasi** del protocollo (pensando ad **azioni concorrenti sugli oggetti**)

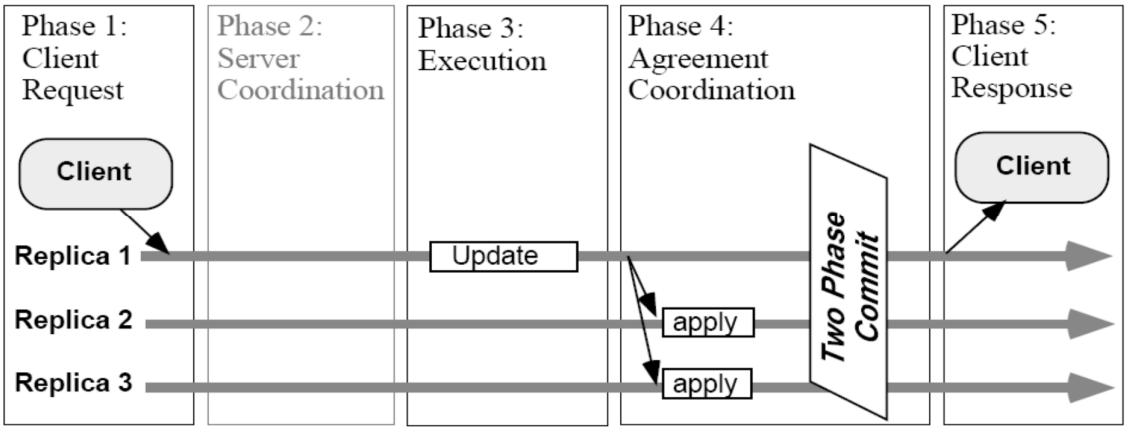
Si noti che non è detto che tutte e due le fasi siano necessarie sempre, ma la presenza è dovuta alla consistenza delle copie e la correttezza

Azioni a posteriori con **verifica della consistenza**, in caso contrario **undo** degli effetti (protocollo a due fasi e roll back)

Azioni a priori che garantiscono che **tutte le copie corrette ricevano solo i messaggi giusti e le operazioni consentite nell'ordine giusto** (multicast atomico)

Copia primaria eager

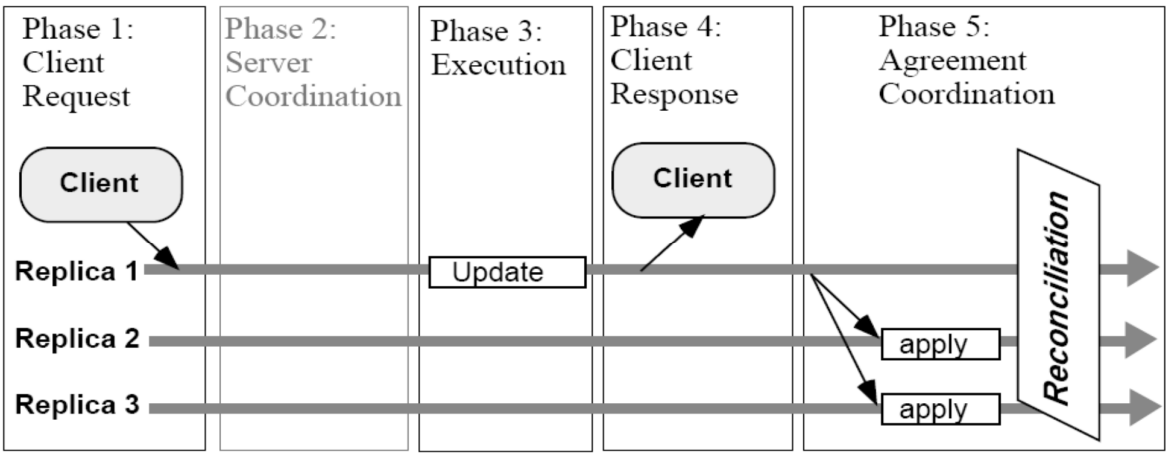
La copia esegue e fornisce la risposta dopo avere coordinato tutte le copie di backup *(una operazione alla volta)*



In particolare si richiede al gestore di **coordinare** tutte le copie e di **controllare** gli aggiornamenti

Politiche lazy

La copia esegue e fornisce la risposta senza coordinare le copie (**politica ottimista**) *(**una o più operazioni per volta**)*



In particolare si richiede al gestore di **coordinare** la riconciliazione finale degli stati delle copie

Politiche lazy

La copia esegue e fornisce la risposta senza coordinare le copie (**politica ottimista**) *(più operazioni per volta)*

In caso di **Amazon S3** (Amazon **S**imple **S**torage **S**ervice), il sistema di memorizzazione e persistenza di Amazon prevede

sia operazioni **consistenti** sia operazioni **eventually consistent**

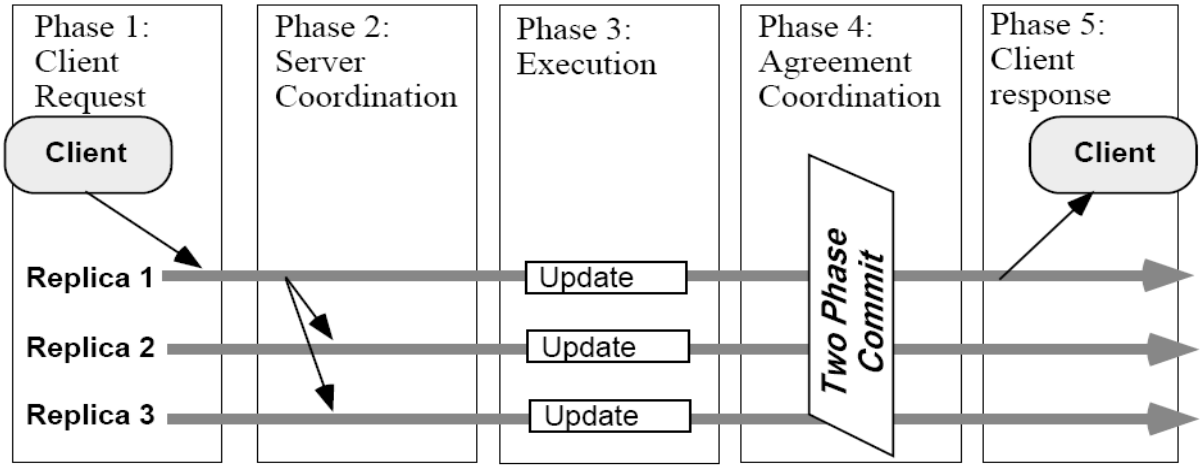
consistenza forte garantisce che gli accessi dopo un update devono restituire il dato aggiornato

consistenza eventuale (o finale, o all'infinito) come forma di **consistenza debole**: update sono eseguiti ma altre operazioni possono anche non fornire l'ultimo valore. Continuando a fare accessi, al limite (all'infinito) si fornisce il valore aggiornato

La **finestra di inconsistenza** dipende da molti fattori: ritardi dovuti alla comunicazione, il carico di lavoro del sistema, il numero di repliche, …

Aggiornamento eager di tutte le copie

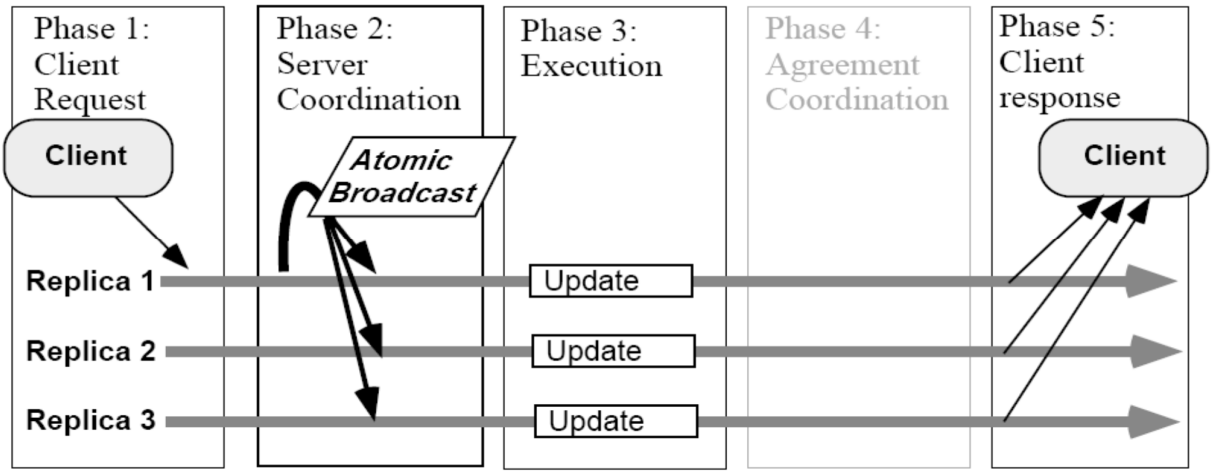
Si aggiornano tutte le copie in modo veloce con politiche diverse (**two-phase commit**) **politica pessimista**



tutte le copie eseguono le operazioni e poi ci si accorda sul risultato con un protocollo di accordo (*eventuale **undo***)

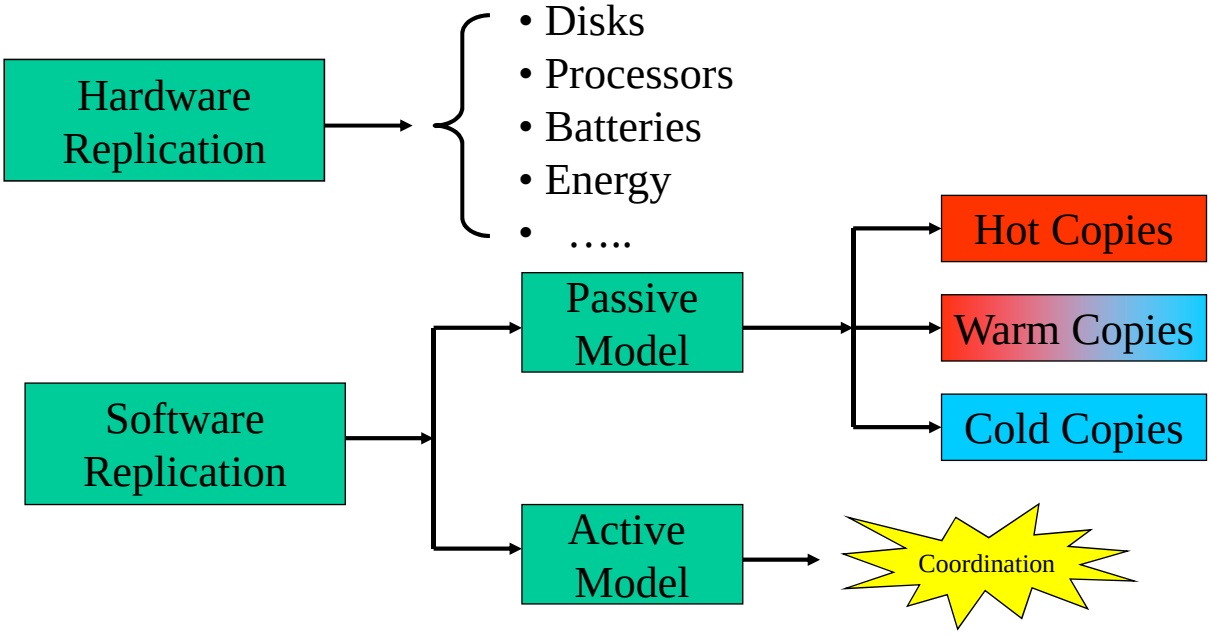
Aggiornamento eager di tutte le copie

Possibilità di avere un supporto per comunicazioni atomiche e per l'accordo dei risultati (**atomic multicast**) **politica pessimista**



Una copia riceve la richiesta e la passa alle altre che eseguono indipendentemente (*richieste in ordine, nessun undo*)

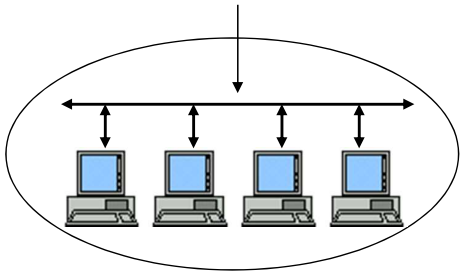
DIVERSE FORME di REPLICAZIONE



Copie **calde**, aggiornamento continuo
fredde nessun aggiornamento
tiepide se aggiornamenti ma non continui

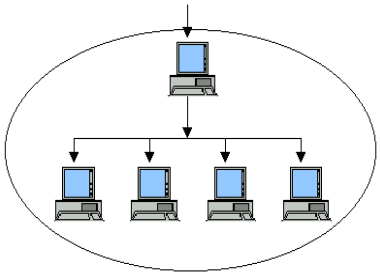
CLUSTER per HIGH AVAILABILITY

Cluster con motivazioni diverse
- **alta disponibilità**
- **alte prestazioni**
- **bilanciamento di carico**



Un cluster per **l'alta disponibilità** è un **insieme di nodi indipendenti** che collaborano allo svolgimento di un **servizio sempre attivo 24/7**

Soluzione molto diffusa per **alta disponibilità** o **high availability**:
- **robusta** ed **affidabile**
- **economica** (parte dall'esistente e utilizza hardware off-the-shelf)



CLUSTER HIGH-AVAILABILITY

Un cluster come insieme di nodi che possono eseguire le stesse operazioni

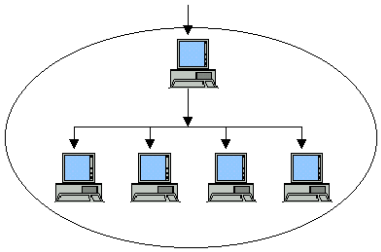
Modello **a copie attive**

tutti i nodi sono in grado di eseguire

Modello a **bilanciamento di carico**

ogni nodo è capace di eseguire operazioni e viene caricato in modo simile (throughput)

Soluzione più diffusa
- ricezione delle **operazioni**
- **smistamento** della singola operazione
- **esecuzione** (e se **guasto?**)
- **invio** della **risposta**



OPERAZIONI nei CLUSTER

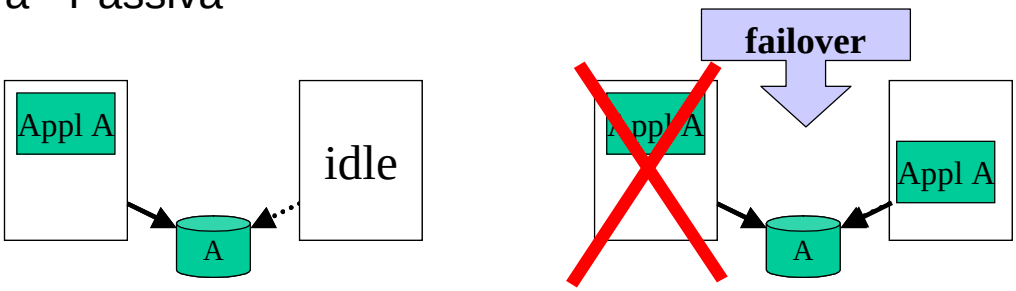
Il supporto al cluster deve gestire:

- **Monitoraggio** dei servizi
  per considerare dinamicamente la qualità del servizio (***finale e percepita***)
- **Failover** (**migrazione del servizio**)
  Il failover rappresenta la migrazione in caso di guasto e tende a rappresentare un momento di non disponibilità del servizio
  Tipicamente dovrebbe essere automatico, trasparente e veloce
- **Heartbeat** (**controllo dello stato dei nodi**)
  con heartbeat si intende tutto l'apparato per il coordinamento tra le diverse copie
  invio di messaggi di are-you-alive? e con quale intervallo?
  In caso di partizionamento, o lavoro indipendente o blocco

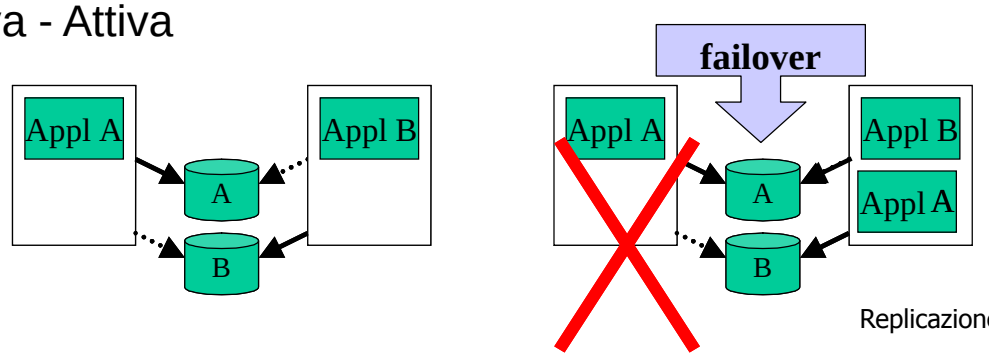
OPERAZIONI CLUSTER: FAILOVER

Nei cluster possiamo avere varie configurazioni per operare

Attiva - Passiva



Attiva - Attiva

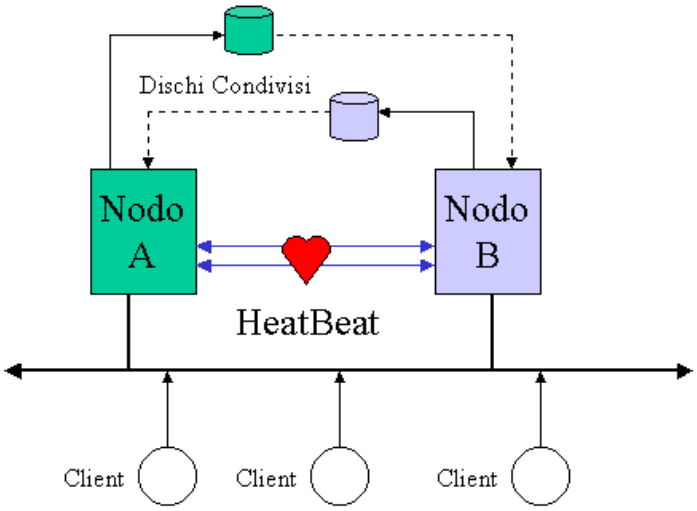


OPERAZIONI in CLUSTER: HEARTBEAT

Heartbeat garantisce un controllo della **disponibilità delle risorse** che devono essere presenti
Può quindi attivare il failover

Un caso problematico se **partizionamento** delle risorse

A volte bisogna **impedire la operatività di una parte** oppure basarsi su algoritmi di **quorum** per decidere **chi procede**

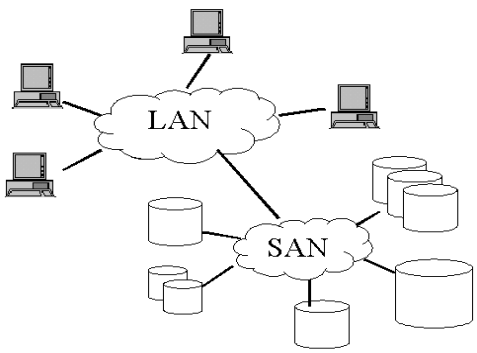


ESEMPI di CLUSTER

I più **comuni cluster** tendono a lavorare con ipotesi simili
- *nessun* ***disco condiviso*** (anziché memoria condivisa)
- *non* assumono di avere a disposizione una **SAN** per i costi **Storage Area Network**
- utilizzano anche **risorse di backup private** ad alcuni nodi

Microsoft Cluster Service e **Sun Cluster 3.0** profondamente integrati nella architettura proprietaria che li propone
Red Hat Cluster suite (open source)

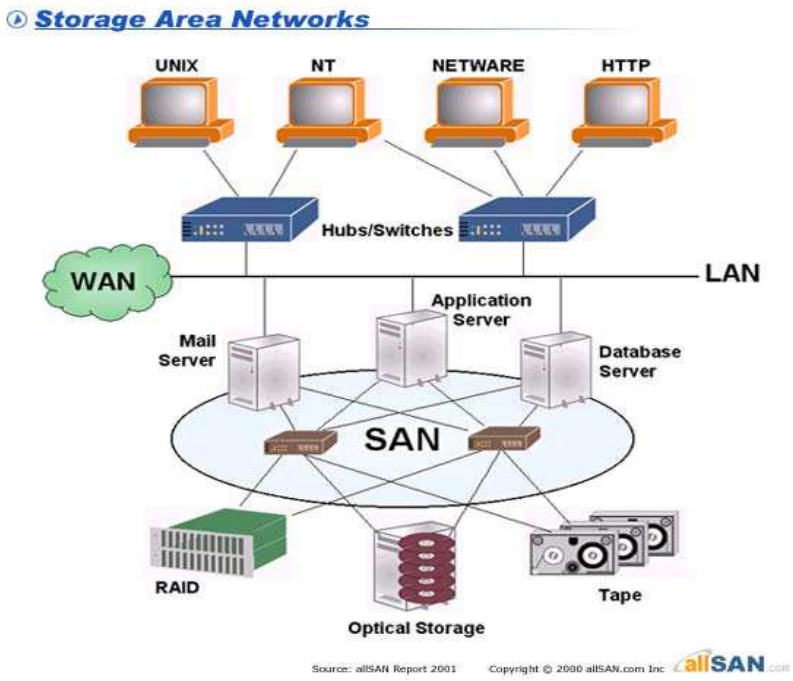
Soluzione standard dovrebbe arrivare dalle *SAN che propongono di creare risorse globali di memoria*



STORAGE AREA NETWORK (SAN)

Una Storage Area Network è un **insieme di risorse interconnesse** per ottenere la massima interconnettività di **memoria** a disposizione di tutti gli utilizzatori

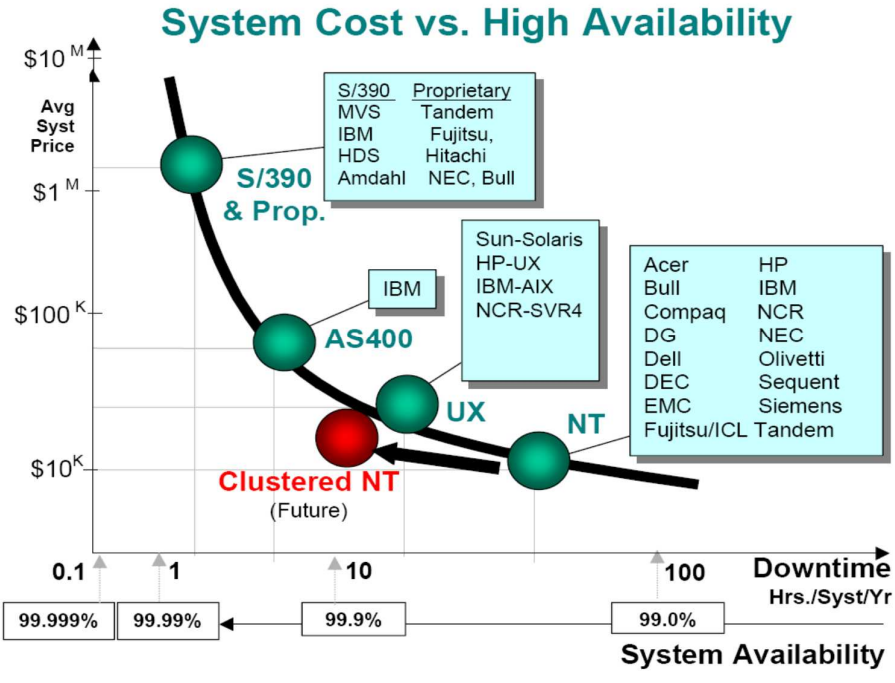
Soluzione per **garantire agli utenti** di accedere alle **risorse libere senza effetti visibili di interferenza** introdotti dalla interconnessione e idealmente **senza limiti di capacità**



HIGH AVAILABILITY (HA)

I costi della alta disponibilità per i servizi tendono a diminuire

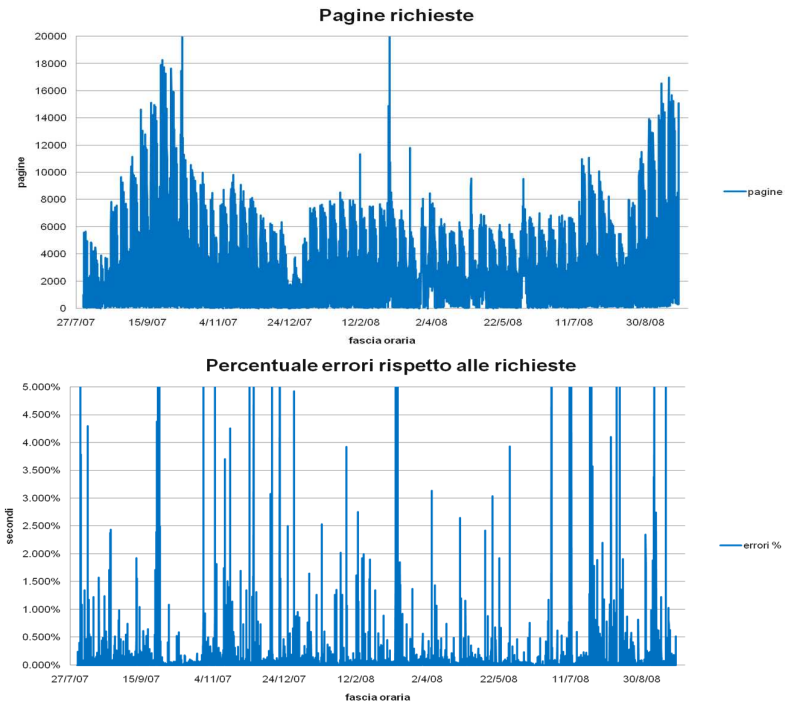
Soluzioni sempre più comuni per ottenere maggiori garanzie di servizio a costi decrescenti, sempre più alla portata anche di piccole installazioni



ESEMPIO di ATENEO

Il sistema di Alma ha necessità di rispondere a molte richieste di servizi per unità di tempo, in particolare a richieste di esecuzione di Web Services

Soluzione per garantire agli utenti di ottenere
risposte in tempi limitati senza errori
e anche con
possibilità di sopravvivere
ai guasti singoli



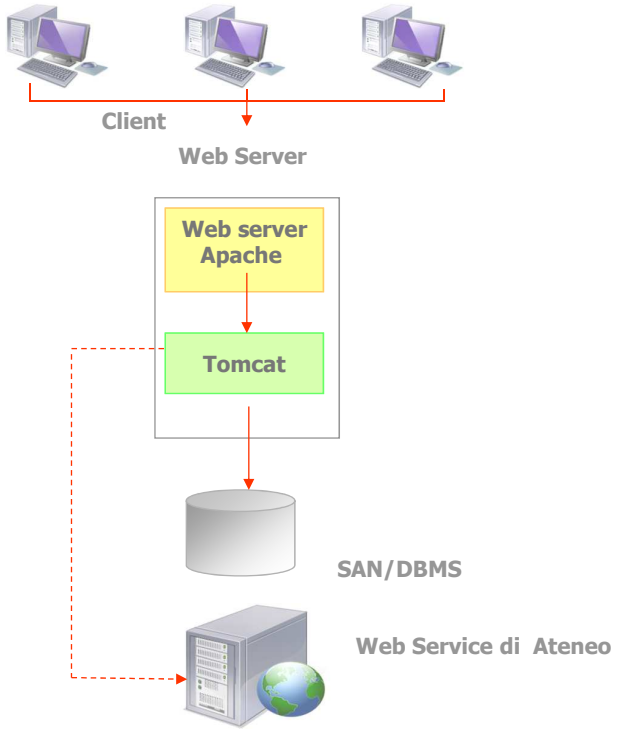
ARCHITETTURA ALMA WEBSERVICE

Soluzione minima

per garantire agli utenti di ottenere le risposte da un Web server

attraverso la invocazione di un Web service che si interfaccia operando su un database di back end

Operazioni richieste dai clienti finali del portale
sia dai **programmi** sia **applicazioni interne all'ateneo**
Le risposte sono fondamentali

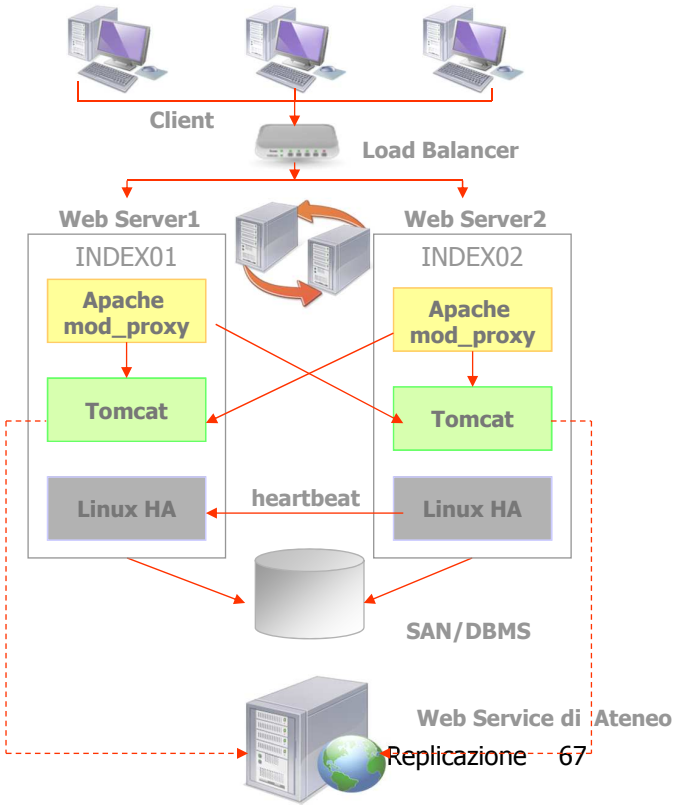


ARCHITETTURA ATTUALE

- Load balancing effettuato da bilanciatore hardware sui due server

- INDEX01 e INDEX02: due Web Server in cluster
- due istanze di Tomcat gestite da Apache proxy

Affidabilità ottenuta attraverso il modulo Linux High Availability
master-slave con heartbeat



Componente di architettura complessa

Progetto COFLIGHT per il controllo del traffico aereo internazionale (2013 - …)

Componente per richieste cliente con replicazione per availability

Decomposizione **per dominare complessità**

Problema nella gestione delle **richieste** e dello **stato**

Uso di processore facade come **front-end** che riceve **tutte le richieste** e le sequenzializza comandando e controllando **processing server** (**servitori senza stato**) in numero variabile

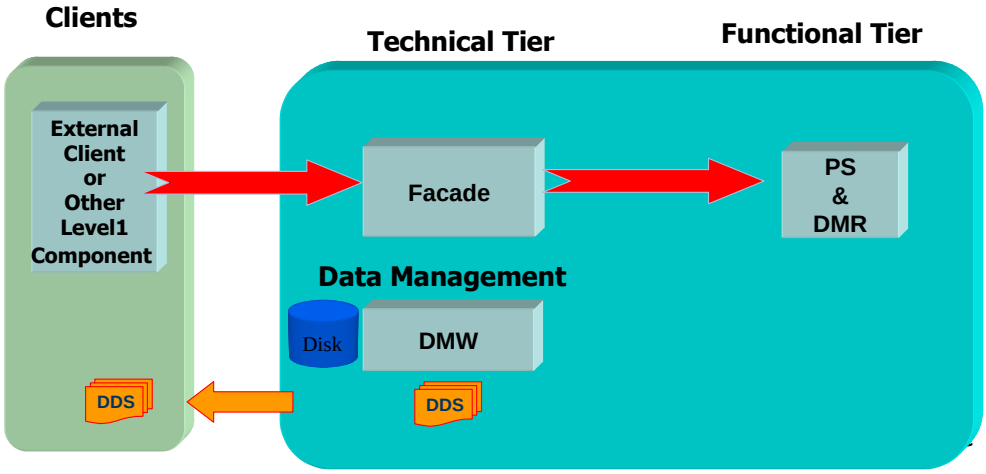
Facade ⇨ front-end per accesso ad operazioni e gestione

Server ⇨ capacità di esecuzione

Uso di middleware esterno con funzioni di controllo e per la identificazione dei guasti (CORBA real-time)

Principi di progetto Componente

Suddivisione funzionale delle responsabilità del servizio
in parti indipendenti, eventualmente replicate e distribuite
- **ricezione richieste** (parte accettazione - technical tier)
- **esecuzione operazioni** (parte esecutiva - functional tier)
- **coordinamento dati** (data management)



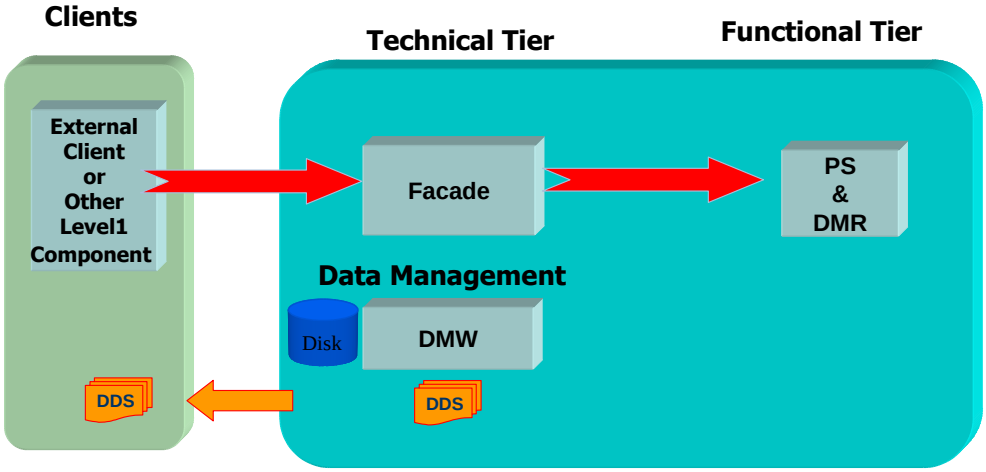
Principi di base per la comunicazione

Necessità di potere disporre di diversi meccanismi di comunicazione tra parti per la flessibilità necessaria
- **Richieste RPC sincrone** (da C a facade, da facade ai processori)
- **PUB-SUB DDS** (per la comunicazione di dati asincroni in modo distribuito e con QoS)

PS processing server

DMR Data Management Read

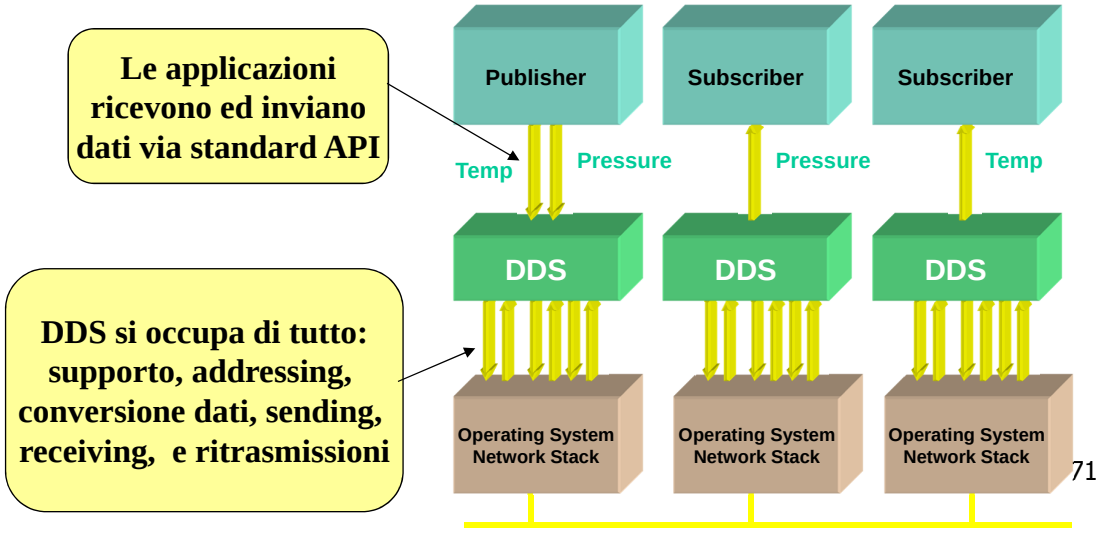
DMW Data Management Write



OMG Data Distribution Service

Proposta di sistema PUB-SUB standard per la distribuzione di valori ai consumatori interessati

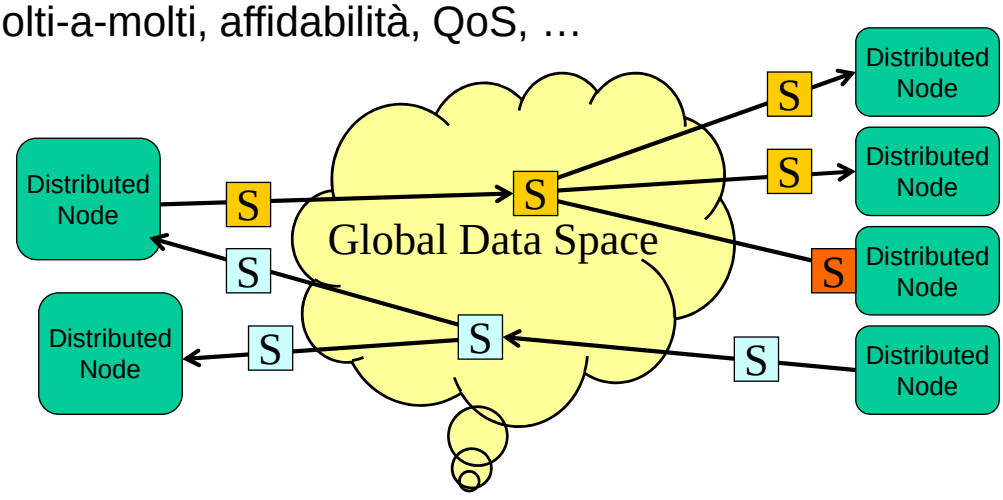
con implementazione completamente distribuita ed altamente affidabile (QoS modulabile ed elevata)



Object Management Group DDS

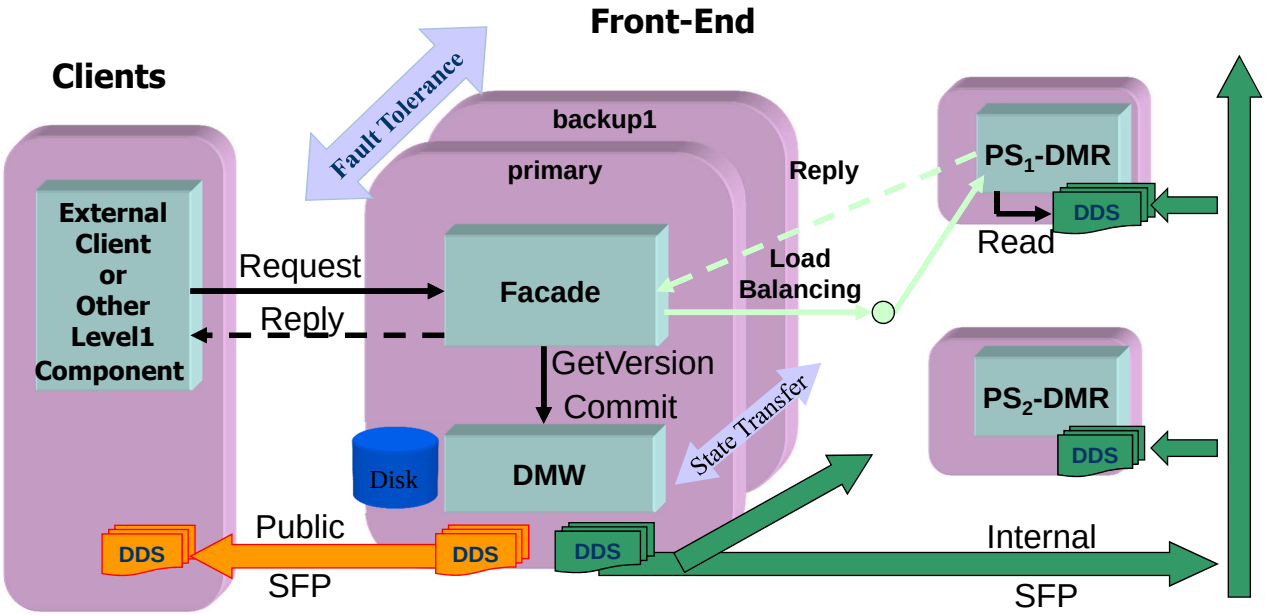
L'uso di PUB-SUB DDS permette di fornire un supporto flessibile di propagazione di dati tra tutti gli interessati

Svincolando l'utente da problemi di accoppiamento e di implementazione del servizio, e permettendo relazioni molti-a-molti, affidabilità, QoS, …



Esistono alcune implementazioni molto diffuse

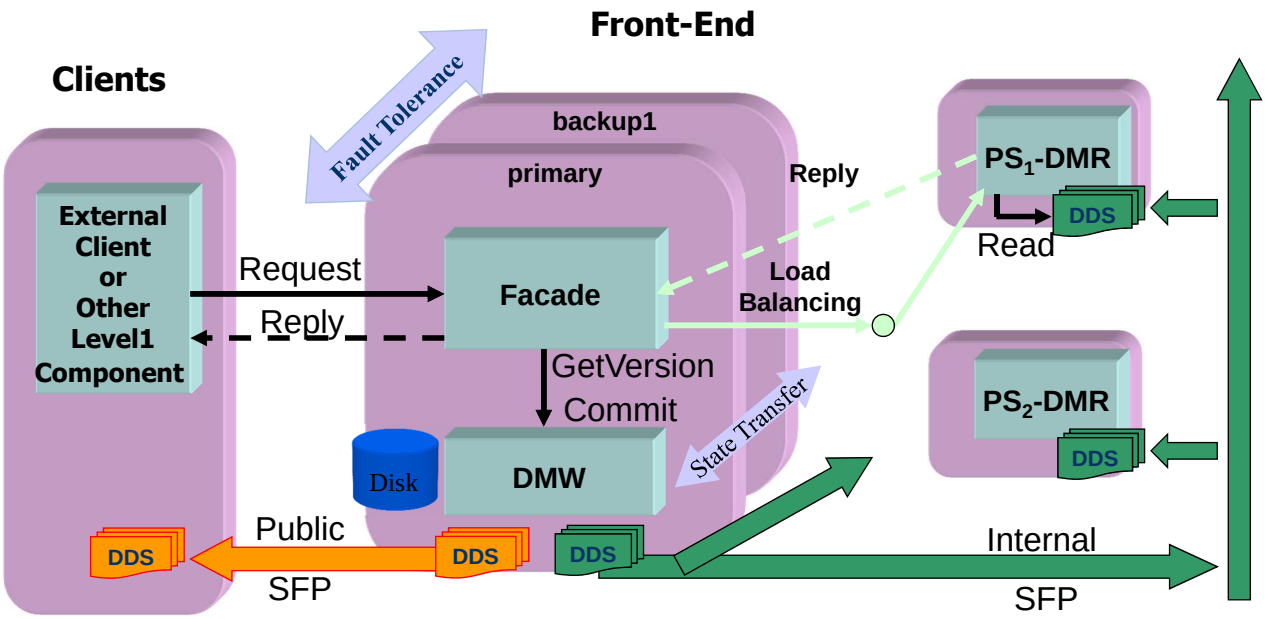
Proposta di componente e interazione



DMR Data Management Read
DMW Data Management Write

SFP Short Flight Plan

Proposta di componente e interazione



DMR Data Management Read
DMW Data Management Write

SFP Short Flight Plan

Progetto di componente Coflight

Replicazione

- Componenti con **replicazione** per **fault tolerance**
  - scelta tra modelli **primary – backup** e a **copie attive**
  - semplicità della architettura
- Componenti per ottenere **separazione** dei compiti e **meccanismi di interazione**
  - Separazione dello **stato**
  - Comunicazione **sincrona cliente/servitore**
- Distribuzione **asincrona** dati via **DDS (OMG standard)**
  - Uso di modello ad eventi per **broadcast / multicast** dati