Introduzione al Cloud Computing - Edizione 2013

3 - Cloud for Developers – Quality of Applications

Sommario

Marco Parenzan

Introduzione

Sviluppare una applicazione è un passo di un percorso lungo, che comincia molto prima e finisce molto dopo l'attività di "codifica". Non si comincia una applicazione dallo sviluppo software, ma dalla definizione degli obiettivi e dalla identificazione corretta dei requisiti: quella che sostanzialmente chiamiamo "analisi". Ancora, una applicazione non finisce con il rilascio dell'applicazione.

Uno dei requisiti della nostra applicazione, in generale sempre presente, anche se scontato, è che il servizio che offre sia erogato con qualità. Ma cosa significa "qualità"? Da Wikipedia:

> ***In generale, la misura della qualità indica una misura delle caratteristiche o delle proprietà di una entità (una persona, un prodotto, un processo, un progetto) in confronto a quanto ci si attende da tale entità, per un determinato impiego.***

Quali sono gli attributi che descrivono il requisito di qualità di una applicazione?

1. Performance: la risposta dell'applicazione deve avvenire in un tempo "congruo"
2. Scalabilità: il tempo di risposta deve essere "congruo" indipendentemente dal carico dell'infrastruttura
3. Disponibilità: l'applicazione deve essere sempre disponibile, essendo "resiliente" nei confronti degli eventi che possono sopravvenire nella fase rilascio dell'applicazione. Essendo certi che il 100% di disponibilità non esiste, il tempo di indisponibilità deve essere "congruo" rispetto al costo dell'indisponibilità. Cioè, costa "poco".
4. Manutenibilità: Il tempo di intervento sull'applicazione per le modifiche deve essere "congruo"

congruo /ˈkɔngruo/

Da http://www.wordreference.com/iten/congruo:

> ***(adeguato, proporzionato)***

Se condividiamo tutti quanti cosa significa congruo, nessuno può condividere "quanto" significa "congruo". Nel senso: ognuno ha una percezione diversa della prestazione. Per qualcuno una funzione può essere veloce, per qualcuno può essere lenta. È necessario che questa quantificazione debba essere inserita a livello di analisi e che possa essere testata, sia in fase di sviluppo che in fase di "acceptance test" all'atto del rilascio.

Performance

Da http://it.wikipedia.org/wiki/Computer_performance:

> ***La performance di un computer (in inglese computer performance, prestazioni del computer) è la quantità di lavoro utile prodotto da un computer in relazione al tempo e alle risorse disponibili.***

A seconda del contesto, una buona performance è data da uno o più dei seguenti parametri:

- Tempo impiegato a terminare un lavoro

- Throughput di dati
- Basso utilizzo di risorse computazionali
- Basso consumo energetico
- Disponibilità di elaborazione

Ciò che caratterizza questi parametri è il loro dimensionamento. Possiamo mettere tutti d'accordo che, per esempio, un basso consumo energetico, arbitrariamente "quasi zero", soddisfa tutti. Ciò che non soddisfa tutti, e deve essere quindi oggetto di specifica, è cosa significa essere "congruo". Cioè: se per raggiungere un livello di "eccellenza" nei consumi devo impiegare una quantità di risorse (soldi, tempo) così elevato da rendere economicamente inaccettabile il livello consumo, devo definire cosa è accettabile e cosa non lo è. Bisogna sempre imporsi un limite: "si potrebbe fare meglio, ma costa troppo, per cui mi fermo lì". Tutta la tecnologia evolve in questi termini: ogni evoluzione tecnologica porta ad una ottimizzazione di questi parametri: non è possibile raggiungere l'ottimo in un colpo solo.

Cosa diciamo quindi per un'applicazione? Definiamo un livello di performance minimo accettabile, che sensatamente bilanci la percezione di risposta e il costo per raggiungerlo.

Principi di Scalabilità

Definito il livello di performance accettabile, questo livello deve essere garantito, qualsiasi sia il livello di carico dell'applicazione. Le performance devono essere "costanti". Cosa influenzano le performance? La mole di dati da elaborare, la quantità di calcoli da effettuare. Queste due "dimensioni" sono conseguenti un uso dell'applicazione, da parte degli utenti. Un utente va definito in termini di profilo:

- Dimensioni (range) dei dati da elaborare
- Quantità di calcoli da effettuare

Se mettiamo insieme tanti utenti, allora queste dimensioni e quantità si moltiplicano.

Una applicazione è scalabile se indipendentemente dal numero degli utenti, le performance dell'applicazione risultano essere costanti. Vale anche al contrario: non è strettamente corretto che una infrastruttura "scarica" di utenti offra comunque delle prestazioni migliori ai "pochi" utenti nel sistema. L'applicazione non sarebbe automaticamente più scalabile, in quanto l'utente tende ad abituarsi facilmente a prestazioni migliori, e meno a prestazioni più scadenti.

La percezione della scalabilità avviene nel momento in cui uno dei parametri (dati o calcolo) è variato e nel il sistema è in sofferenza. Dopo si potranno fare ragionamenti di benchmarking costante (per vedere se il sistema sta volgendo alla sofferenza) oppure di elasticità (per avere una reazione automatica/autonoma del sistema). Un sistema è comunque scalabile se, essendo in sofferenza, permette di gestire la situazione di disagio in maniera efficiente, a costi prima di tutto previsti.

Ci sono due modi o due "dimensioni" per gestire la scalabilità di un sistema:

- scalabilità verticale (o scaling-up)
- scalabilità orizzontale (o scaling-out)

Storicamente, la scalabilità verticale è il concetto di scalabilità noto e arbitrariamente condiviso. È bene conoscerlo perché non è un concetto "obsoleto", anzi! Obiettivo è condividere il fatto che non è il solo modo di scalare una applicazione.

Scalabilità verticale

È naturale sviluppare una applicazione (non) pensando ad una singola macchina che esegue il codice. Il "non" è provocatorio: ci si abitua solo a scrivere codice, non a pensare a come quel codice verrà eseguito. È un modello naturale, che non ha bisogno di esplicite specifiche, almeno per partire.

Quando la macchina tende ad affaticarsi, è naturale, ancora, pensare che le risorse possano essere aumentate: CPU, memoria, spazio su disco, banda della rete. Questo è vero fino ad un certo punto: l'hardware tipicamente evolve, le memorie decrescono di prezzo, le nuove CPU migliorano le prestazioni. In questa fascia, il costo tende a crescere linearmente, se non essere asintotico: aggiungere non serve più. Sostituisco o aggiungo componenti e la capacità della macchina cresce, fino alla saturazione. Oltre la macchina non è in grado di accettare risorse aggiuntive.

È tempo di cambiare classe dell'hardware: se siamo partiti con dei server di classe "PC" (solo per semplificare l'esempio) a questo punto bisogna scegliere soluzioni più importanti che permettano di accedere a listini diversi: processori di classe server, più processori, architetture dei dischi nuove che permettano di utilizzare diverse meccaniche. Il costo della soluzione ora non è più lineare, perché l'hardware non accetta più, se mai lo avesse fatto, una manutenzione o delle competenze di una qualità diversa: multiprocessore/multicore/multithread, memoria ECC, dischi RAID, controller in Fibra Ottica, SAN, alimentazione stabilizzata. Server gemelli e server di test per le verifiche in caso di aggiornamento. Il costo non è più strettamente di hardware o di una singola, ma di software di gestione e di contratti di manutenzione, di aggiornamento costante e testato.

Disponibilità

La scalabilità verticale parte da un presupposto mai affrontato esplicitamente: il programmatore non ne deve sapere niente. Le prestazioni, la configurazione dell'hardware e del software non sono competenza di un programmatore. Se è vero che un programmatore, in generale elevato anche al rango di analista, non ha particolari competenze di infrastruttura, esserne totalmente escluso non è allo stesso modo corretto.

Non è comunque una posizione oggi sostenibile, perché non tiene conto di altre necessità che vanno oltre la pura prestazione velocistica. Ad esempio, se la singola macchina "fallisce", cioè non funziona più, il servizio non può essere erogato. Per questo motivo, entrano in gioco due definizioni:

Resilienza (da http://it.wikipedia.org/wiki/Resilienza_(informatica))

> ***In informatica con il termine indice di fragilità o resilienza si indica la capacità di un sistema di adattarsi alle condizioni d'uso e di resistere all'usura in modo da garantire la disponibilità dei servizi erogati. Tali obiettivi si possono raggiungere mediante tecniche di ridondanza…***

Ridondanza (da http://it.wikipedia.org/wiki/Ridondanza_(ingegneria))

> ***Nell'ingegneria dell'affidabilità, la ridondanza è definita come l'esistenza di più mezzi per svolgere una determinata funzione, disposti in modo tale che un guasto del sistema [1] possa verificarsi solo in conseguenza del guasto contemporaneo di tutti questi mezzi.***

In pratica la ridondanza in ingegneria consiste nella duplicazione dei componenti critici di un sistema con l'intenzione di aumentarne l'affidabilità e la disponibilità, in particolare per le funzioni di vitale importanza per garantire la sicurezza delle persone e degli impianti o la continuità della produzione. D'altra parte, poiché l'introduzione di ridondanze aumenta la complessità del sistema, le sue dimensioni fisiche e i costi, generalmente esse sono utilizzate solo quando i benefici derivanti sono maggiori degli svantaggi sopra citati, il che necessita di uno studio approfondito adattato al particolare caso di interesse.

Costo dell'indisponibilità

Se il sistema si guasta, oppure non riesce a gestire le innumerevoli richieste degli utenti, cosa succede? Il sistema diventa indisponibile, non risponde. Il disservizio diventa tale quando il tempo diventa significativo, quindi diventa superiore ad una soglia che deve essere definita. La non risposta significa "disservizio" e questo ha un costo. Il costo del disservizio è pari a:

> ***C_{tsd} = numero di disservizi x costo del singolo disservizio***

Se il tempo di disservizio (t_d) è minore del tempo di soglia (t_{sd}) allora non c'è un reale disservizio. Il tempo di disservizio si può abbassare, investendo sull'infrastruttura. Quanto investo sull'infrastruttura? Lo stretto indispensabile affinchè il tempo di soglia del disservizio si fissa ad un altro valore, più basso.

Scalabilità verticale e ridondanza

La scalabilità verticale diventa onerosa in conseguenza della ridondanza. Non è possibile supporre che un unico server, su cui si è investito in termini di pure risorse di elaborazione (CPU, RAM e DISCO) non abbiano un analogo investimento in termini di ridondanza.

Le parti "deboli" del sistema vengono ridondate: ne vengono installate due o più che intervengono automaticamente nel momento in cui queste possono fallire. In senso generale, la ridondanza può avvenire in due modi:

Ridondanza per bilanciamento di carico

> ***Le due o più componenti ridondanti si distribuiscono il carico perché la funzionalità riesce a distribuire il lavoro, considerando***

Ridondanza Active/Passive

> ***Active/Passive significa che un solo componente alla volta è attivo, gli altri passivi. Non è sempre possibile tenere tutte le parti in linea. La particolarità dell'Active/Passive è che il "sistema" si accorge che una parte è in fallimento e fa intervenire automaticamente una parte passive che viene elevata a nuova "active". Tipicamente, questo avviene in automatico, senza soluzione di continuità.***

A questo, punto, dove posso intervenire con la ridondanza? Le parti sono:

- Hot Swap
- Alimentazione
- Memoria ECC
- Dischi RAID

- Schede di rete
- Controller dei dischi (in fibra)
- CPU, Memoria e scheda madre
- Cluster

Hot Swap

La ridondanza dell'hardware, per garantire la continuità di servizio, deve essere associata alla possibilità di sostituire la parte in fallimento senza dover spegnere il sistema. Alcune parti lo prevedono (alimentatori, dischi), altre no (schede su bus). La differenza sta sostanzialmente su ciò che è collegato direttamente ai bus delle CPU piuttosto che ai dei connettori su scheda.

Alimentazione

Dove ci sono i computer, o almeno i server, ci sono i gruppi di continuità. L'erogazione di corrente elettrica deve essere garantita anche quando la rete elettrica non fornisce corrente (banalmente per un temporale, per un istante di tempo). Continuità elettrica è diversa da qualità della corrente elettrica: la corrente elettrica è "sporca", non è una sinusoide perfetta. Più che la forma della corrente, è la velocità della corrente: i 50/60Hz della corrente sono gestiti (o ricostruiti) dal gruppo di continuità. Ma questi sono circuiti analogici che cercano di stabilizzare l'erogazione per sistemi digitali. Ci possono essere spunti della corrente elettrica che saltano le protezioni del gruppo di continuità, se già hanno passato le protezioni elettriche dell'impianto. Questi spunti, tipicamente, tendono a far saltare gli alimentatori.

Server di fascia media installano due alimentatori, addirittura collegati a due linee di alimentazione separate, supponendo addirittura (e non è infrequente) che l'alimentazione stabilizzata (UPS) possa fallire.

Memoria ECC

Algoritmi nati nella trasmissione dei segnali elettrici (e radio) hanno sviluppato tecniche di correzione degli errori per i dati digitali. Se nella trasmissione dei segnali si parla di "errori di trasmissione", nel caso delle memorie si parla di "errori per guasti". Ma il fine è lo stesso.

Bisogna pensare che anche una memoria può fallire, che un bit possa non essere memorizzato correttamente e avere la consapevolezza che anche un bit può cambiare l'esito di una elaborazione. Gli algoritmi di codifica dei dati digitali, implementati nelle memoria a correzione di codice (dall'inglese Error Correcting Code) permettono di gestire queste situazioni di guasto che garantiscono la continuità nel breve.

Le memorie ECC non possono essere sostituite a caldo: se siamo in una condizione di fallimento, è possibile continuare, ma bisogna pianificare un fermo (e quindi gestire in altro modo la continuità di servizio) per la sostituzione della memoria guasta.

Dischi RAID

Schede di rete

Controller dei dischi (in fibra)

CPU e scheda madre

Una CPU, una memoria o una scheda madre non può essere "gemellata". Le CPU non vengono mai accoppiate in termini di disponibilità o indisponibilità per guasto. Se una CPU fallisce, fallisce tutta la macchina. Ancora di più per le schede madri. Per gestire la ridondanza di CPU e scheda madre si interviene in altra maniera.

Cluster

Due macchine, in configurazione Active/Passive, di puro calcolo, gemelle, che condividono un unico storage di dati, e che sono tra loro collegate per rendersi conto (vicendevolmente) se l'altra è indisponibile, è tipicamente una configurazione da "cluster". Le macchine fisiche sono in realtà un unico sistema, e così va considerato. Se devo essere "completamente" disponibile, il cluster diventa l'ultimo step di crescita in una configurazione completamente verticale. Il costo non è prettamente "doppio".

Virtualizzazione

La virtualizzazione è l'abilitatore del Cloud Computing e quindi protagonista della rivoluzione che è in corso. In termini di disponibilità di una soluzione, in termini di business continuity, non è però una soluzione e non va confusa. Due sono i principi alla base della virtualizzazione:

- acquisire un nuovo host con elevate risorse (CPU, MEMORIA, STORAGE), installare un Hypervisor (http://it.wikipedia.org/wiki/Hypervisor) stile VM Ware o Hyper-V (per parlare di architetture x64)
- Trattare i server come appliance.

Da http://it.wikipedia.org/wiki/Appliance

> ***In informatica, un Appliance, o Computer Appliance, è un particolare dispositivo elettronico hardware provvisto di un software integrato (firmware) con funzione di sistema operativo, utilizzato per eseguire particolari complesse e massicce funzioni applicative software, quindi spesso utilizzato nelle grandi reti di calcolatori o Server farm aziendali.***
>
> ***Il termine Appliance significa apparecchio di applicazione, per indicare proprio la funzione di un dispositivo progettato per un applicativo specifico. La differenza sostanziale con i normali server o le normali apparecchiature di rete è che l'appliance non è pressoché progettato per essere flessibile alle modifiche del software o dell'hardware successive alla configurazione e installazione fatta per la sua specifica funzione applicativa.***

Il server diventa completamente software, in quanto l'hardware viene virtualizzato. "Acquisire un nuovo server", compatibilmente con la disponibilità di risorse (reali, hardware) sull'host, significa creare dei files che sono associati ad un profilo macchina e a dei dischi e far partire un nuovo processo di sistema operativo (host, una VM altro non è che un file .exe).

L'economia di scala (ovvero i grandi vettori che spingono il mercato da una parte o dall'altra) sta nel consolidamento.

Consolidamento

Se ci si trova in condizioni di un alto numero di server, che implica problemi logistici o infrastrutturali (spazio fisico, condizionamento, alimentazione elettrica, cablaggio di rete, ecc…) allora si tende a "consolidare" i server originali in una nuova infrastruttura tipo:

- Macchine ad alta densità (http://it.wikipedia.org/wiki/Blade_server) per soluzioni tipicamente da Data Center

Una delle percezioni errate in termini di Disaster Recovery è il fatto che fare il backup della macchina virtuale (e nella sua essenza, fare il backup dei file .vhd dei dischi e la configurazione della macchina), è sufficiente a fare il "backup" della soluzione. Le best practice dicono che la macchina virtuale va trattata come una macchina reale, quindi, questo non esime dal fare il backup dei singoli servizi installati nella macchina virtuale (es. tra i più critici, nel mondo Microsoft, SQL Server, Exchange, Active Directory).

Rischi della virtualizzazione

Un rischio, interessante in generale (non in termini di pura disponibilità), è il dimensionamento. Il rischio è sempre quello di sovraccaricare un host (quindi sottodimensionarlo) con troppe macchine virtuali. Più importante è l'aspetto del Disaster Recovery.

La virtualizzazione semplifica il ripristino di una macchina, se i dati della macchina sono integri verso una nuova macchina. Ma la virtualizzazione non risolve il tema della disponibilità e anzi ne amplifica il rischio. Se su una macchina host ho consolidato n-macchine virtuali, allora se la macchina host fallisce, allora n macchine, non una, non saranno disponibili.

Preludio alla Scalabilità orizzontale

Come abbiamo detto pervasivamente, l'informatica prevede la disponibilità implementata con la ridondanza: devo avere una "parte di ricambio". La scalabilità verticale, in generale, contempla il modello active/passive: la parte di ricambio rimane sostanzialmente in attesa (a parte i dischi con il RAID).

L'esempio eclatante è il cluster: date due macchine, una macchina rimane ferma, in attesa, sempre sincronizzata, pronta ad intervenire. Ma appunto, ferma.

Cosa succederebbe se il modello applicativo prevedesse di utilizzare entrambe le macchine, utilizzando come parametro l'utente e il fatto di poter distribuire il carico di utenti tra le due macchine? Succederebbe una cosa interessantissima: due macchine, o tre, o dieci, sarebbe esattamente la stessa cosa. I problemi che sorgerebbero con due macchine sono le stesse avendone dieci.

Si creano quindi delle nuove opportunità:

- Avendo n macchine, il fallimento di una macchina non rende indisponibile il sistema
- Accettando la fallibilità dell'hardware (in un tempo congruo all'investimento) posso spendere molto meno sulla singola macchina (dove il costo complessivo delle singole macchine e la loro infrastruttura è comunque minore della singola macchina che scala in orizzontale)
- Si risolve contemporaneamente il problema della scalabilità e della disponibilità
- È possibile partizionare non solo il numero di utenti ma anche le funzioni
- Il presumibile costo "amministrativo" di una proliferazione di macchine è già gestito dagli amministratori di sistema, che già sanno gestire un parco macchine distribuito

Se si assume un modello di scalabilità orizzontale, la consapevolezza impatta sulle applicazioni. Una applicazione deve essere sempre consapevole dell'infrastruttura in cui va in esecuzione, soprattutto se è orizzontale. Se lo è, la regola che funziona (quasi) certamente è lo scalare verticalmente. Di certo non può non essere consapevole di scalare orizzontalmente. Deve esserlo, perché altrimenti non può funzionare (prendendo come riferimento il modello di sviluppo di applicazioni).

Non tutti i mali vengono per nuocere:

- Disegnare una applicazione che permette di scalare orizzontalmente è più complessa e quindi se non si ha la percezione dei costi, diventa un problema
- Una applicazione che scala orizzontalmente scala anche verticalmente e quindi è un modello applicativo riusabile anche quando la scalabilità orizzontale non è necessaria
- Un'applicazione che scala orizzontalmente, per sua natura (stabiliti i principi), architetturalmente evoluta e di conseguenza, se non si vuole pagare in termini di gestione post-rilascio, manutenibile.

Una applicazione che scala orizzontalmente è un'opportunità. Le risorse spese nella definizione e realizzazione di una applicazione che scala orizzontalmente vanno intese come investimento e non come costo. I vantaggi sorpassano gli svantaggi (se vogliamo chiamarli tali).

La parola d'ordine è manutenibilità.

Manutenibilità