# Appunti del corso di calcolo ad alte prestazioni

Dave Null

## Indice

## Sommario

In questo documento sono raccolti gli appunti del corso *"Calcolo ad Alte Prestazioni per la Fisica"*, tenuto dal professor Giacinto Donvito al dipartimento interateneo di Fisica dell'Università degli studi di Bari nell'anno accademico 2015/2016.

Per la parte di strumenti per il calcolo scientifico si rimanda al manuale *Amministrare GNU/Linux* di S. Piccardi, distribuito con licenza GNU Free Documentation License (FDL) all'indirizzo:

https://labs.truelite.it/attachments/download/821/corso.pdf

ed in particolare, con riferimento alla revisione 1367 del 9/07/2014, ai capitoli 1, 2, 6 ed i paragrafi 4.1, 4.2, 4.3, 5.1 (ed eventualmente del 8.3).

Una utile introduzione sull'utilizzo di `make` si trova nella appendice "Gli strumenti di ausilio per la programmazione" di *Guida alla Programmazione in Linux* (GaPiL) sempre di S. Piccardi e distribuita anch'essa con licenza FDL all'indirizzo

http://gapil.gnulinux.it/files/2011/12/gapil.pdf

**ATTENZIONE:** Questo documento viene distribuito nella speranza possa essere utile, tuttavia non è garantita la correttezza dei contenuti o della forma. Sono ben gradite segnalazioni di eventuali errori.

## Licenze

# Teoria dei calcolatori e del calcolo scientifico

## Modello di Von Neumann

Praticamente tutti i computer attuali fondano la propria architettura sul *modello di Von Neumann*, quest'ultimo si basa sul principio secondo cui i dati e le istruzioni condividono lo stesso spazio di memoria. Lo schema è il seguente, ci sono 5 unità fondamentali:

1. **CPU (o unità di lavoro)** che si divide a sua volta in

- **Unità di controllo**, che gestisce le operazioni necessarie per eseguire una istruzione o un insieme di istruzioni
- **Unità operativa**, la cui parte più importante è l'unità aritmetico-logica (o ALU) e che effettua appunto operazioni aritmetiche e logiche

2. **RAM (Random Access Memory)**: Unità di memoria, intesa come memoria di lavoro o memoria principale
3. **Unità di input** (tastiere, mouse, ecc.)
4. **Unità di output** (monitor, stampanti, plotter, ecc.)
5. **Bus** ovvero un canale che collega tutti i componenti fra loro

In un calcolatore convenzionale le istruzioni vengono processate **sequenzialmente** nei seguenti passi:

1. un istruzione viene caricata dalla memoria (fetch) e decodificata
2. vengono calcolati gli indirizzi degli operandi
3. vengono prelevati gli operandi dalla memoria
4. viene eseguita l'istruzione
5. il risultato viene scritto in memoria (store)

Esistono pertanto due strategie per migliorare le performance

- Aumentare le prestazioni dei singoli componenti elettronici
- Eseguire più istruzioni contemporaneamente

Si precisa che in generale un piccolo aumento delle performance non è interessante da un punto di vista tecnologico e pertanto si considera un incremento significativo intorno ad un ordine di grandezza (fattore 10).

La prima strategia ha dei limiti fisici:

- Problema della dissipazione del calore
- Limite imposto dalla velocità della luce

Il problema della dissipazione del calore è collegato alla dimensione dei transistor che compongono la CPU, questa dimensione è a sua volta collegata alle dimensioni caratteristiche del processo di litografia utilizzato (*feature dimension*). Il limite attuale per queste dimensioni è intorno ai 10 nanometri e non è possibile scendere molto al disotto di questa dimensione per via di effetti quantistici (oltre naturalmente al limite imposto dalle dimensioni atomiche del supporto di silicio).

L'altro limite è la velocità di I/O che è vincolata al limite fisico della velocità della luce per i segnali elettromagnetici nel vuoto.

Altri problemi sono la dimensione della ram, che pone un limite alle applicazioni in esecuzione (attualmente su un singolo slot 128GB), la larghezza di banda tra memoria e processore, o fra processore/memoria e I/O, la larghezza di banda della cache, ecc.

## Tassonomia di Flynn

La prima forma di parallelismo sperimentata è il **pipelining** e presenta delle analogie con il concetto di catena di montaggio dove, in una linea di flusso (pipe) di stazioni di assemblaggio gli elementi vengono assemblati a flusso continuo.

In questo caso le componenti elettroniche specializzate (unità) che operano su ciascuna delle 5 fasi attraverso cui viene eseguita una istruzione (dal fetch allo store) funzionano simultaneamente. Ciascuna lavora sulla istruzione seguente rispetto alla unità successiva e sulla istruzione precedente rispetto alla unità precedente. In questo modo si evitano i tempi morti sulle singole unità.

Idealmente tutte le stazioni di assemblaggio devono avere la stessa velocità di elaborazione, altrimenti la stazione più lenta diventa il *bottleneck* della intera pipe.

La tassonomia di Flynn è una classificazione delle molteplicità dell'hardware per manipolare stream di istruzioni e di dati. In particolare lo **stream delle istruzioni** (sequenza delle istruzioni eseguite dal calcolatore) può essere singolo, **SI** per Single Instruction stream, o multiplo, **MI** per Multiple Instruction stream, ed anche lo **stream di dati** (sequenza degli operandi su cui vengono eseguite le istruzioni) può essere a sua volta singolo, **SD** per Single Data stream, o multiplo, **MD** per Multiple Data stream. Complessivamente sono possibili 4 combinazioni:

- **SISD**: è la tipica architettura di Von Newman (sistemi scalari mono-processore) e può essere pipelined
- **MISD**: più flussi di istruzioni lavorano contemporaneamente su un unico flusso di dati. Non è stata finora utilizzata praticamente.
- **SIMD**: una singola istruzione opera simultaneamente su più dati, in questo caso il sistema possiede una singola unità di controllo ma diverse unità di elaborazione. Sistemi di questo genere vengono anche chiamati *array processor* o *sistemi vettoriali*. Anche questo genere di sistemi può essere pipelined. Esistono esempi famosi di supercomputer vettoriali (come quelli costruiti dalla Cray negli anni 70) e sviluppati per applicazioni particolari, oggi molti dei moderni processori hanno capacità SIMD ed un set di istruzioni dedicato (SSE, Streaming SIMD Extensions, nel caso Intel).
- **MIMD**: in questo caso si parla anche di parallelismo asincrono ed esistono più CPU che operano su dati diversi, ovvero più unità di controllo ciascuna collegata a più unità operative. In questo senso ci si può riferire a questi sistemi come ad una versione multiprocessore dei sistemi SIMD.

Per realizzare sistemi MIMD è necessario che i vari sottosistemi comunichino tra loro. Questo può avvenire sia in un solo calcolatore con **un'unica memoria condivisa** fra più processori (per questo detti *tightly coupled*), che per questo devono trovarsi nello stesso spazio fisico, sia tramite una rete di calcolatori (con processori in questo caso *loosely coupled*) interconnessi e funzionalmente completi (dotati di CPU, RAM, bus, dischi, etc.), che possono anche trovarsi in posti geograficamente differenti. Inoltre sono anche possibili soluzioni combinate di questi due casi.

I sistemi di tipo MIMD abbracciano una ampia classe di architetture possibili, ricapitolando si ha

- **Architettura MIMD multiprocessore o shared memory system**: i processori condividono i dati e le istruzioni in una memoria centrale comune. La comunicazione avviene dunque mediante condivisione della memoria attraverso un bus.
- **Architettura MIMD multi-computer o distributed memory system**: ogni processore ha una memoria propria. I vari processori comunicano tra loro mediante una rete che consente a ciascun processore di trasmettere e ricevere dati dagli altri. I processori possono anche essere fisicamente lontani.

Inoltre il genere di calcoli che vengono eseguiti su un sistema parallelo può essere di due tipi, distinti per il genere di priorità che comportano e di performance richieste al sistema

- **HPC, High Performance Computing**: il calcolo HPC consiste nell'esecuzione di *task* computazionalmente intensivi nel minor tempo possibile ed è dunque caratterizzato dalla necessità di grandi capacità di calcolo distribuite in brevi periodi di tempo (ore o giorni). Le performance di sistemi per HPC sono spesso misurate in *FLOPS* (FLoating point OPerations per Second).
- **HTC, High Throughput Computing**: il calcolo HTC consiste usualmente nell'esecuzione di molti task debolmente correlati (o di singoli task su grandi moli di dati) che si ha interesse siano completati efficientemente lungo periodo (mesi o anni).

La velocità della condivisione di istruzioni e dati caratteristici dei sistemi a memoria condivisa contro la flessibilità e la scalabilità dei sistemi a memoria distribuita tendono in maniera naturale a far identificare i primi come soluzioni per HPC ed i secondi per HTC.

Anche in questo caso (specialmente nel caso di grandi sistemi) questi paradigmi possono combinarsi (ad esempio i sottosistemi di una infrastruttura per HTC possono essere effettivamente sistemi HPC).

## SETI@home

SETI@home è stato un progetto per la ricerca di segnali radio compatibili con segni di vita intelligente extraterrestre. Questo progetto è stato tra i primi e più famosi ad essersi avvalsi di una infrastruttura di calcolo distribuito *volontario* basata su internet, ovvero ad aver utilizzato le risorse, momentaneamente e gratuitamente messe a disposizione, di personal computer geograficamente distribuiti e connessi ad un nodo centrale tramite una connessione internet.

Oggi parte della tecnologia sviluppata per SETI@home è confluita in una infrastruttura per il calcolo distribuito indipendente dal particolare progetto di ricerca denominata BOINC (Berkeley Open Infrastructure for Network Computing). Tramite quest'ultima gli utenti possono decidere di donare il proprio tempo di CPU ad uno o più progetti. Lo stesso SETI@home prosegue oggi come uno dei progetti ospitati da BOINC.

Nel caso di SETI@home l'obiettivo era analizzare tutto lo spettro delle frequenze di un radio telescopio per l'intero tempo di osservazione. Ciò che ha reso possibile l'impiego della infrastruttura di calcolo descritta

1. I dati possono essere spacchettati in piccole porzioni (dell'ordine di 500 KByte), non è pertanto necessaria una connessione ad alte performance fra i singoli nodi.
2. Il tempo di CPU necessario per analizzare tali porzioni di dati su hardware di consumo è relativamente breve.

3. L'analisi dei segnali in ciascuna porzione è del tutto indipendente dalle altre ed è pertanto possibile eseguire tutti i job contemporaneamente (si dice che il problema è *embarrassingly parallel* o *perfectly parallel*).

## Cluster

Un gruppo di calcolatori interconnessi ed in grado di lavorare cooperativamente come un unico sistema è in genericamente indicato come *cluster*.

Perché un cluster possa essere considerato come un sistema a memoria distribuita è necessario:

- hardware di rete ad elevate prestazioni
- uno strato software che implementi le funzionalità richieste (interfacce, protocolli, librerie, etc.)
- applicativi che sfruttino le capacità del sistema

Un esempio di queste librerie è MPI (Message Passing Interface) che permette di implementare in diversi linguaggi (C, C++, Fortran, Python, Julia) applicativi per sistemi a memoria distribuita. L'ovvia premessa è che, indipendentemente dal sistema particolare o dal linguaggio di programmazione, l'algoritmo sia parallelizzabile.

In generale esistono tre tipi di cluster (i primi due sono i più diffusi):

1. **Fail-over Cluster**: il funzionamento delle macchine è continuamente monitorato e quando una di queste ha un malfunzionamento un'altra subentra in attività.
2. **Load Balancing Cluster**: è un sistema nel quale le richieste di lavoro sono inviate alla macchina con meno carico di elaborazione distribuendo/bilanciando così il carico di lavoro sulle singole macchine.
3. **HPC Cluster**: le macchine suddividono l'esecuzione di un *job* in più processi e questi ultimi vengono istanziati simultaneamente su più macchine (calcolo distribuito).

In generale i vantaggi offerti da un cluster rispetto ad un singolo calcolatore sono

- *Incremento delle capacità di calcolo* grazie allo sfruttamento di più unità di calcolo e maggiore disponibilità di memoria
- *Maggiore affidabilità* in quanto il sistema continua a funzionare anche in caso di guasti di parti di esso (ovvero dei singoli calcolatori)
- *Minori costi*, infatti questi sistemi sono fino a 15 volte più economici dei tradizionali supercomputer rispetto ai quali, a parità di prestazioni, permettono un notevole risparmio sui componenti hardware
- *Scalabilità hardware*, possibilità di aggiungere ho rimpiazzare singole componenti a caldo
- *Disponibilità di un gran numero di software Open Source* come MOSIX e openMosix.

Gli svantaggi principali sono:

- Difficoltà di gestione e di organizzazione di un elevato numero di calcolatori
- Possibilità di problemi di connessione e di banda passante (specialmente di calcolatori lontani)
- Assieme allo hardware scala anche la probabilità di failure
- Scarse prestazioni nel caso di applicazioni non parallelizzabili

Fra i primi cluster si ricordano i cluster Beowulf, ovvero cluster di semplici personal computer collegati tramite reti informatiche standard, senza l'utilizzo di hardware esplicitamente sviluppato per il calcolo parallelo. Cluster di questo genere erano in genere cluster di computer IBM compatibili, implementati utilizzando software Open Source con lo scopo di eseguire task computazionalmente intensivi in genere in ambito tecnico scientifico.

I Beowulf cluster sono caratterizzati da

- Accesso al sistema possibile solo dal nodo principale, che spesso è l'unico ad essere dotato di tastiera e monitor.
- Nodi basati su di una architettura standard (x86, AMD64, PowerPC, etc.), usualmente uguale per tutti i nodi.
- Utilizzo di un sistema operativo (in genere GNU/Linux) e software Open Source

## Batch System

Gli atomi di calcolo (in cui viene istanziato un processo, letti o scritti dati, etc.) in un sistema multiutente (come può essere un cluster) vengono detti **job** (utilizzando una terminologia propria dei sistemi UNIX) e la richiesta di esecuzione di un job da parte di un utente è detta sottomissione di un job.

Per poter distribuire i job degli utenti su un cluster è necessario un software chiamato batch system (o PBS, Portable Batch System). Questi software si occupano di gestire l'ordine di esecuzione dei job e, in un sistema distribuito, di scegliere il nodo su cui verranno eseguiti. Dunque i batch system hanno il compito di accomodare le richieste degli utenti (*domanda*) e la disponibilità (limitata) di risorse (*offerta*), operazione nota in questo contesto come **matchmaking**.

Alcuni esempi di batch system sono Condor, Torque/Maui, LSF, SLURM, PBSPro, Oracle Grid Engine, Open Grid Engine.

La maggior parte di questi sistemi organizzano i job in una o più code e per questa ragione sono anche noti come *sistemi di code*. L'operazione di ordinare i job nella coda di esecuzione è nota come **job scheduling**, o scheduling, ed esistono diversi algoritmi che tentano di ottimizzare il problema dello scheduling rispetto a diversi parametri.

La premessa ovvia al problema dello scheduling è la capacità di monitorare le risorse.

Genericamente i parametri rispetto a cui viene ottimizzato un algoritmo di scheduling sono

- **Throughput**: il numero totale di job che vengono completandoti nell'unità di tempo
- **Latenze** ed in particolare
- **Turnaround Time**: tempo totale tra la sottomissione di un processo ed il suo completamento
- **Response Time**: lasso di tempo che intercorre fra la sottomissione di un job ed una risposta (es. stampa a video sul terminale) da parte del job
- **Fairness**: equità del tempo di calcolo assegnato a job della stessa priorità
- **Waiting Time**: tempo che un job spende inattivo nella propria coda

In pratica è non è possibile ottimizzare tutti questi parametri contemporaneamente (si consideri ad esempio il throughput e la latenza) ed è necessario realizzare un compromesso. I criteri con cui si realizza quest'ultimo sono caratteristici del particolare algoritmo di scheduling e rispondono a particolari esigenze, ad esempio, nel caso di un sistema in cui la sottomissione di job è un servizio a pagamento, parametri come la fairness ed il waiting time diventano prioritari.

Alcuni esempi algoritmi di scheduling sono Unix Scheduling, FCFS (First Come, First Served), SJR (Shortest Job first), Small Job first / Big Job first, Priority Scheduling, Round Robin, Multilevel Queue, Multilevel Feedback Queue, Fair-share.

# Principi di sicurezza

Si considerano preliminarmente alcune definizioni generali

- **Credenziali**: producono prova dell'identità dell'utente, in genere sono uno *username* ed una *password* assegnati in combinazione unica
- **Autenticazione**: verifica dell'identità dell'utente
- **Autorizzazioni** o permessi, vengono assegnati all'utente assieme alle credenziali e, a seguito della autenticazione, gli permettono o meno di eseguire operazioni su un sistema (istanziare processi, leggere o scrivere dati, etc.)
- **Confidenzialità**: capacità di inviare un messaggio che solo il destinatario possa comprendere
- **Integrità**: capacità di stabilire che un messaggio ricevuto sia identico a quello inviato dal mittente, ovvero che non vi siano state alterazioni fra l'invio e la consegna
- **autenticità e non ripudio**: rispettivamente capacità di stabilire l'identità del mittente ed impossibilità da parte di quest'ultimo di disconoscere il messaggio

## Algoritmi di crittografia simmetrica e asimmetrica

Nei sistemi informatici la confidenzialità è ottenuta crittograficamente. Il messaggio viene cifrato dal mittente utilizzando un algoritmo (noto anche al destinatario) ed una certa *chiave* mentre il destinatario lo decifra utilizzando una seconda chiave.

Storicamente i primi algoritmi crittografici erano a **chiave simmetrica**, nel senso che una chiave poteva essere ottenuta dall'altra (ad esempio perché erano identiche). Per esempio **AES** (Advanced Encryption Standard, anche conosciuto come *Rijndael*) è un algoritmo a chiave simmetrica selezionato dal NIST (National Institute of Standards and Technology) sia perché (ad oggi) crittograficamente sicuro sia per le sue performance.

Gli algoritmi a chiave simmetrica pongono intrinsecamente dei rischi di sicurezza, indipendentemente dalla bontà degli algoritmi. Infatti perché il destinatario possa ricostruire il messaggio è *necessario che conosca la chiave*. Il mittente è pertanto costretto a comunicargliela e per fare ciò è necessario un canale di trasmissione sicuro, che però potrebbe non esistere (ovviamente non ha senso crittografare una chiave simmetrica con una chiave simmetrica perché si riporrebbe il problema).

Inoltre gli algoritmi a chiave simmetrica soffrono di un ulteriore problema. Infatti il mittente dovrà condividere una chiave differente con ogni destinatario (posto che sia interessato a trasmettere in maniera segreta con ciascun destinatario rispetto ad ogni altro destinatario). Questo comporta che

il numero di chiavi (che devono essere trasmesse e conservate in maniera sicura) cresca in una rete di utenti come il quadrato del numero di questi ultimi.

Questo genere di problemi è risolto tramite l'uso di **chiavi asimmetriche**. Negli algoritmi asimmetrici esistono due chiavi, una detta **pubblica** e l'altra **privata**, e la chiave privata è in grado di decifrare un messaggio cifrato tramite la chiave pubblica. Tramite l'uso di algoritmi asimmetrici il destinatario può inviare (o rendere appunto pubblica) una copia della propria chiave pubblica, ma conservare la propria chiave privata, e per il mittente sarà sufficiente cifrare il messaggio utilizzando la chiave pubblica del destinatario. Una volta consegnato il messaggio il destinatario potrà decifrarlo utilizzando la propria chiave privata.

In generale, essendo questi algoritmi computazionalmente costosi, nel caso di messaggi molto grandi si adotta usualmente la strategia di cifrare questi ultimi usando una chiave simmetrica e poi di condividere la chiave simmetrica cifrandola con un algoritmo asimmetrico.

Uno dei primi algoritmi a chiave asimmetrica ad essere scoperti (1978) ed oggi maggiormente adottati è **RSA** (da nomi dei suoi inventori Ronald Rivest, Adi Shamir e Leonard Adleman). In questo caso le chiavi consistono in genere di una coppia di file ed una *passphrase*, nota solo al proprietario della chiave privata.

L'algoritmo RSA ha inoltre la seguente utile proprietà. Si è già visto che un messaggio cifrato usando la chiave pubblica può essere decifrato utilizzando la chiave privata, tuttavia è vero anche il contrario: ovvero un messaggio cifrato utilizzando la chiave privata può essere decifrato usando la chiave pubblica.

## Firma digitale

Per **hash** o **one-way hash function** si intende una funzione $H$ (in pratica un algoritmo) che riceve una stringa di lunghezza qualsiasi $M$ e restituisce una stringa di lunghezza fissata $h$ tale che

1. dato $M$, sia semplice (economico) calcolare $h = H(M)$
2. dato $h$, sia difficile (costoso) calcolare gli elementi di $H^{-1}(h)$
3. dato $M$, sia difficile (costoso) calcolare $M'$ tale che $H(M) = H(M')$ (ci si riferisce a questa eventualità col termine **collisione**)

Si osserva che, a differenza degli algoritmi crittografici, gli algoritmi di hashing non sono per definizione invertibili (non esiste una corrispondenza univoca fra $M$ e $h$). La conseguenza di questo fatto è che esistono certamente collisioni.

Può essere più economico, o interessante per altre ragioni, calcolare le funzioni di hash per due stringe e confrontare i risultati che confrontare direttamente le stringe di partenza. In questo caso il risultato delle funzioni di hash si chiama **checksum** e si preferiscono funzioni di hash che abbiano la ulteriore proprietà che la probabilità di collisione sia *piccola* per stringhe di partenza *simili*.

L'integrità, l'autenticità ed il non ripudio possono essere ottenuti sfruttando le funzioni di hash e l'algoritmo RSA. In questo caso il mittente che vuole apporre la propria **firma digitale** su un messaggio (eventualmente in cifrato) calcola lo hash del messaggio, lo cifra con la sua chiave privata ed allega il risultato al messaggio. In questo modo il destinatario potrà calcolare lo hash del messaggio, decifrare tramite la chiave pubblica quello in allegato ed effettuare un confronto.

Se i due hash coincidono il destinatario potrà dedurre che

1. Il messaggio ricevuto coincide (a meno di una collisione, eventualità con probabilità trascurabili) con quello su cui il mittente ha apportato la firma (integrità)
2. Il mittente può essere solo il proprietario della chiave privata (autenticità) e quest'ultimo non può negare di aver firmato il messaggio (non ripudio)

## Certificati personali secondo lo standard X.509

Gli algoritmi a chiave asimmetrica e la firma digitale costituiscono una soluzione a certi problemi di sicurezza se sono vere alcune premesse

1. Le chiavi private degli utenti non sono compromesse
2. Gli utenti sono in grado di associare con sicurezza l'identità di altri utenti alle rispettive chiavi pubbliche (che in qualche modo devono essere state distribuite in precedenza)

Esistono due modelli per garantire la seconda di queste premesse

1. **X.509**: controllo centrale da parte di un terzo individuo (*hierarchical organization*)
2. **PGP**: sistema distribuito con relazioni fra pari (*web of trust*)

Nel secondo caso si sfrutta l'assunto che se si conosce l'identità di un utente e quest'ultimo *garantisce* sull'identità di un terzo utente, allora si può considerare (ragionevolmente) sicura l'identità di questo terzo utente, anche nel caso non fossimo in grado di stabilirla direttamente.

Questo ragionamento si può applicare più volte ed è eventualmente possibile stabilire in questo modo una o più catene di relazioni di fiducia fra mittente e destinatario.

In pratica in una *web of trust* un utente che sia sicuro della identità associata ad una chiave pubblica può firmarla e queste chiavi pubbliche firmate sono raccolte in database liberamente accessibili online. Software come **GPG** (GNU Privacy Guard), oltre a permettere di cifrare, decifrare e firmare messaggi, ricostruiscono automaticamente queste relazioni di fiducia in base alle chiavi pubbliche fidate nel proprio *portachiavi*.

X.509 è uno standard per PKI (Public Key Infrastructure), che permette di realizzare relazioni di fiducia tra utenti tramite un controllo centrale.

Esistono due concetti fondamentali nello standard X.509

1. **Certification Authority** o CA
2. **Certificati digitali**

La certification authority è una terza parte la cui identità è per ipotesi fidata (*trust anchor*) che può rilasciare certificati digitali per utenti, software o calcolatori. Si occupa di verificare l'identità dei richiedenti (tramite le RA, Registration Authorities) e pubblica periodicamente un elenco dei certificati compromessi che sono ancora in periodo di validità nelle CRL (Certificate Revocation Lists).

I certificati digitali rappresentano una estensione del concetto di chiave pubblica e contengono almeno i seguenti campi

1. la chiave pubblica del proprietario del certificato
2. l'identità del proprietario (es. dati anagrafici)
3. alcune informazioni sulla CA

4. periodo di validità del certificato
5. firma digitale della CA

Il mittente in questo modo può esaminare il certificato (dopo averne ricevuto copia) ed essere sicuro della corrispondenza fra l'identità del destinatario e la sua chiave pubblica.

Per poter applicare lo schema X.509 è tuttavia necessario che un utente possieda la chiave pubblica della CA (che deve essere stata ottenuta tramite un canale sicuro perché è per ipotesi fidata). Queste chiavi vengono solitamente distribuite sotto forma di certificati, detti *root certificate*, in modo da contenere informazioni addizionali come ad esempio il periodo di validità della chiave. Inoltre i root certificate sono **self signed**, ovvero firmati dallo stessa CA, ed è pertanto possibile stabilirne l'integrità (l'autenticità è data per ipotesi).

# Grid Computing

## Principi di calcolo a Grid

Una **Grid computazionale**, o semplicemente una Grid, è una infrastruttura hardware e software che garantisca un accesso affidabile, coerente, capillare ed economico a risorse computazionali ad alte prestazioni.

Dal punto di vista delle istituzioni che forniscono queste risorse il suo scopo è permettere una condivisione flessibile, sicura e coordinata di queste ultime.

Inoltre, dato l'interesse nella condivisione dinamica di risorse fra diverse organizzazioni, le tecnologie Grid non possono rimpiazzare le preesistenti tecnologie per il calcolo distribuito, ma devono piuttosto coesistere ed essere complementari a queste ultime.

Gli utenti di un sistema Grid operano generalmente all'interno di organizzazioni virtuali, a cui sono garantiti certi permessi e risorse.

Ci sono tre proprietà fondamentali che caratterizzano una Grid rispetto ad altri sistemi per il calcolo distribuito

1. Coordinazione su *larga scala* di risorse geograficamente distribuite ed appartenenti a diversi domini amministrativi
2. Protocolli ed interfacce standard, aperte e polivalenti, tali da provvedere ad una ampia gamma di servizi
3. Capacità di fornire vari tipi di qualità del servizio (QoS, Quality of Service), ovvero di coordinare le risorse in modo da fornire un servizio calibrato sulle richieste degli utenti.

Inoltre una Grid presuppone l'assenza di

1. **Central control**: una istituzione/nodo centrale a cui facciano riferimento tutti gli altri.
2. **Omniscience**: la conoscenza da parte di ogni istituzione/nodo dello stato di tutti gli altri.
3. **Existing Trust Relationship**: autenticazione degli individui o delle istituzioni sulla base di relazioni di fiducia preesistenti o comunque esterni alla Grid.

Dal punto di vista dell'utente il genere di problemi che una Grid cerca di risolvere è del tipo

Eseguire il programma X nel sito Y con permessi P, provvedendo accesso ai dati sul sito Z in accordo ai permessi Q

Riassuntivamente una Grid fornisce un livello di astrazione che rende accessibile un ambiente di calcolo distribuito ad alte performance per gli utenti e che permette la condivisione di risorse per le istituzioni, in maniera trasparente ed immune da tre generi di eterogeneità:

1. delle risorse
2. delle policy
3. delle applicazioni

Si osserva che il calcolo su Grid si può definire calcolo distribuito con le seguenti precisazioni

1. Non viene condiviso solo un processo fra i nodi (dunque tempo di CPU e memoria) ma anche dati e software (in breve viene offerto, in accordo ai permessi posseduti dall'utente, l'accesso ad una intera macchina)
2. L'esecuzione di un singolo job avviene su una singola macchina (o su un singolo cluster di macchine), ovvero i job sono tra loro indipendenti.

Un esempio di Grid è **EGI** (European Grid Infrastructure), nata da progetti precedenti come DataGrid e EGEEE (Enabling Grids for E-SciencE).

## Sottomissione di job ad una Grid

Per Grid middleware si intende una serie di componenti software che fungono da intermediari tra l'infrastruttura e le applicazioni utente, rendendo possibile la condivisione di risorse ed infine il calcolo su Grid. Alcuni esempi di Grid middleware sono gLite e EMI, entrambi collegati a EGI.

Le varie componenti di un Grid middleware si possono classificare in

1. **Servizi di Accesso**: CLI, API, etc.
2. **Servizi di Sicurezza**: autenticazione e autorizzazioni
3. **Servizi di informazione e monitoraggio**: monitoraggio dello stato dei servizi, del carico dei sistemi, etc.
4. **Servizi Dati**: catalogo dei file e delle repliche, catalogo dei metadati, archiviazione fisica dei dati
5. **Servizi di Elaborazione**: gestione dei job, interfacciamento con i siti di calcolo, etc.

Considerando l'esempio di EGI è possibile illustrare i diversi componenti di un Grid middleware.

Un utente che vuole sottomettere un job, in questo contesto noto come *gridlet*, deve preparare un **job description file** ovvero un file che contiene una descrizione del job in un linguaggio noto come **JDL** (Job Description Language).

JDL è un linguaggio flessibile, estensibile e di alto livello, derivato da **ClassAd** (CLASSified Advertisement language), il linguaggio utilizzato dal batch system *Condor*. Sia JDL che ClassAd contengono una struttura a record composta da un certo numero di attributi separati da un punto e virgola (`;`).

Un file JDL in genere contiene almeno i seguenti campi

1. `Executable` : deve essere uno shell script (in genere bash, csh o tcsh).
2. `StdOutput` : nome del file che conterrà la l'output del job (i.e. lo standard output)

3. `StdError` : nome del file che conterrà gli errori del job (i.e. lo standard error)

Altri campi importanti sono

1. `Type` : può assumere i valori **Job**, che può essere a sua volta normale (default) o parametrico, **DAG** (Directed Acyclic Graph) o **Collection**
2. `Arguments` : contiene gli argomenti da passare allo script che deve essere eseguito
3. `Environment` : contiene una o più variabili d'ambiente da esportare prima di eseguire lo script (in JDL è possibile costruire un vettore di espressioni con le parentesi graffe `{}`)
4. `InputSandbox` : elenco dei file che devono essere inviati insieme al job
5. `OutputSandbox` : elenco dei file (inclusi lo standard error e output) che devono essere restituiti in caso di esecuzione con successo del job.
6. `Requirements` : richieste sul working node che eseguirà il job (ad esempio sul sistema operativo, supporto per OpenMP o OpenMPI, etc.)

Si osserva che con una singola richiesta è possibile istanziare più job in una volta, infatti

- un job parametrico è un job in cui uno o più argomenti variano in un intervallo
- un DAG è un insieme di job con relazioni di dipendenza reciproca che possono essere descritte da un grafo diretto aciclico (e per il quale esiste una specifica sintassi)
- una collezione è un insieme di job indipendenti

Si osserva che in questo modo si riduce il tempo di sottomissione (singola autenticazione/autorizzazione, etc.).

Considerato lo script di esempio

```
#!/bin/sh
# test.sh
echo $*
```

un semplice job description file può essere (i commenti su una linea usano # quelli su più linee devono essere incapsulati in `/* */`)

```
# test.jdl

Executable    = "test.sh";
Arguments     = "Hello world!";
StdOutput     = "std.out";
StdError      = "std.err";
InputSandbox  = {"test.sh"};
OutputSandbox = {"std.out", "std.err"};
```

Il job description file viene quindi inviato al **WMS** (Workload Management System), che (fra gli altri) ha l'incarico di assegnare l'esecuzione del job ad un particolare sito di calcolo (CE, Computing Element, che in generale può essere una macchina o un cluster) tenendo conto delle preferenze espresse nel job description file (matchmaking). A questo punto il job si trova in una **coda globale**, in attesa di essere assegnato. Successivamente il WMS richiede informazioni allo **Information Index** o **Information System**, che contiene un database dei CE attualmente disponibili, del loro carico di lavoro ed eventualmente delle loro policy. Si osserva che a questo scopo i CE comunicano direttamente con lo information index. Tramite queste informazioni uno dei componenti del WMS detto **Resource Broker** computa il CE più opportuno su cui eseguire il job. Il WMS può quindi

inoltrare il job su una **coda locale** attraverso un suo componente che il cui compito è interfacciarsi con il batch system locale del CE (CREAM, Computing Resource Execution and Management, per gLite e EMI-ES, EMI Execution Service, per EMI). Fra i batch system supportati figurano PBS(Torque/MAUI), LSF, SGE.

Ogni passaggio del ciclo di vita di un job è registrato da un servizio chiamato **Logging and Bookkeeping** (LB), con cui sono in comunicazione sia il WMS che il CE. Questo è utile ad esempio per conoscere lo stato di un job che è stato sottomesso, infatti gli utenti possono tracciare lo stato del proprio job conoscendone lo ID (che viene comunicato durante la sottomissione). Dopo essere stato eseguito successo da un **Worker Node** l'output del job viene inoltrato automaticamente (tramite il CE) al WMS , da cui l'utente può ottenere una copia (in questo schema l'utente non si interfaccia mai direttamente col CE, tuttavia non è l'unica implementazione possibile).

Tra i comandi a disposizione di un utente vi sono

| **Description** | **Command** |
|---|---|
| Submitting a job | `glite-wms-job-submit` |
| Checking job status | `glite-wms-job-status` |
| Retrieving job output | `glite-wms-job-output` |
| Checking job history | `glite-wms-job-logging-info` |
| Listing compatible resources | `glite-wms-job-list-match` |
| Delegate a proxy | `glite-wms-job-delegate-proxy` |

## Gestione della sicurezza in Grid

In genere in una Grid viene utilizzato (ed ampliato) lo standard X.509. Ad esempio in EGI le certification authority riconosciute sono diverse ed hanno in genere base nazionale, tuttavia esiste un progetto per la coordinazione delle CA europee chiamato EUGridPMA (quest'ultimo in realtà include diverse CA extra-europee, oltre a collaborare con analoghe organizzazioni in Asia e negli Stati Uniti).

Sempre in EGI i root certificate sono distribuiti tramite *pacchetti* per diversi sistemi operativi e disponibili in repository pubblici (dopo aver aggiunto il repository è possibile installare tutti i certificati tramite il meta-pacchetto `lcg-CA`).

Uno degli standard di sicurezza adottato su Grid è GSI (Grid Security Infrastructure) basato sullo standard PKI X.509. In questo standard ogni transazione (scambio di dati, invio di istruzioni, etc.) è **mutualmente autenticata**.

Infatti tramite lo standard di sicurezza X.509 è possibile risolvere (oltre ai problemi di confidenzialità, integrità e non ripudio) il problema della autenticazione. Il meccanismo è il seguente.

All'inizio di una transazione tra due utenti A e B (ciascuno dei due può essere un individuo, un software o una macchina), dopo aver stabilito una connessione

1. A invia a B una copia del proprio certificato.
2. B ne verifica la validità tramite la firma della CA (che può verificare a sua volta tramite i root certificates).

3. Se la verifica del certificato di A è andato a buon termine, B invia un messaggio casuale (*challenge string*) che contiene un *timestamp*, ovvero una indicazione che permette di datare univocamente la transazione.
4. A firma la challenge string ed invia la firma a B.
5. B verifica la firma e autentica A

Perché vi sia mutua autenticazione si ripete la procedura a ruoli invertiti. Si osserva che il solo certificato non autentica A, ma garantisce soltanto la corrispondenza fra l'identità di A ed una chiave pubblica. Invece nella procedura descritta (anche nota come CHAP, Challenge-Handshake Authentication Protocol) A viene autenticato nella misura in cui si è certi che solo A sia in possesso della chiave privata di A. L'ultima considerazione è cruciale in tutti i sistemi di sicurezza basati su chiavi asimmetriche e deve essere sempre tenuto presente.

Nella effettiva implementazione della autenticazione nei servizi Grid viene utilizzata una estensione dei certificati X.509 detta **X.509 proxy certificate** o **certificati proxy**. Questi certificati sono contraddistinti dalle seguenti caratteristiche

1. Hanno durata piuttosto breve, dell'ordine di alcune ore (usualmente 12) contro i certificati personali che hanno in genere durata annuale
2. Contengono al loro interno una coppia di chiavi, pubblica e privata

Questa estensione ha la seguente motivazione. Nel contesto di una Grid è spesso necessario che un servizio o l'istanza di un software possano agire per conto dell'utente (ad esempio un job può aver bisogno di recuperare una copia di un file e dunque di dover contattare un server ed autenticarsi). Deve pertanto esistere un meccanismo in grado di trasferire la propria capacità di autenticazione, questo meccanismo è noto come **delegazione**.

Una soluzione ingenua potrebbe essere quella di rilasciare insieme al proprio certificato una copia della propria chiave privata. Tuttavia per quanto già osservato una soluzione di questo genere comporterebbe gravi problemi di sicurezza.

Quando si crea un certificato proxy vengono create una coppia di chiavi ed incluse nel certificato, tuttavia la propria chiave privata personale **non** viene allegata. In questo modo la propria identità è stata delegata. Infatti un individuo o un servizio in possesso un certificato proxy potrà non solo instaurare una relazione di fiducia con altri utenti (essendo in possesso di un certificato valido), ma anche autenticarsi grazie alla propria chiave privata.

Inoltre un proxy può delegare a sua volta la propria identità prolungando la catena della fiducia.

Chiaramente in termini di sicurezza un proxy è un compromesso. Infatti la delegazione implica che esista nel sistema una coppia di chiavi (non cifrate). Pertanto, tramite la chiave privata di un proxy, è possibile impersonare l'identità (delegata) dell'utente che in origine ha creato il proxy. Tuttavia la delegazione è solo temporanea ed ha il grande vantaggio che la chiave privata dell'utente rimane sempre segreta.

Lo standard GSI prevede ulteriori estensioni a X.509 con lo scopo di semplificare la gestione delle autorizzazioni. Infatti il problema delle autorizzazioni è complesso e consiste nell'accomodare i permessi garantiti da due diverse entità

1. Virtual Organization (VO)
2. Resource Provider (RP),

Lo sviluppo dei **VOMS** (Virtual Organization Membership Service) nei sistemi Grid è stato una risposta all'esigenza di centralizzare la gestione delle autorizzazioni nelle organizzazioni virtuali. A questo scopo i certificati proxy sono stati estesi al fine di poter (opzionalmente) contenere informazioni associate ai permessi entro l'organizzazione virtuale di appartenenza. Queste informazioni sono

1. Organizzazione virtuale di appartenenza
2. Una lista dei gruppi di appartenenza. Infatti le organizzazioni virtuali hanno una struttura ad albero (simile ad un filesystem UNIX) i cui nodi sono le organizzazioni e di cui la VO rappresenta la radice (ogni utente della VO deve appartenere ad almeno un gruppo e può appartenere a più gruppi)
3. Ruolo all'interno del gruppo, che può considerarsi come una regolazione fine dei permessi rispetto ai assegnazione ad un gruppo. Si osserva che i ruoli sono relativi al gruppo, nel senso che ruoli con lo stesso identificativo in gruppi diversi hanno diverso significato.
4. Capacità, ovvero autorizzazioni speciali o temporanee (un utente può ad esempio avere permessi di amministrazione solamente durante i suoi *turni*)

Infine i resource provider sono liberi di computare le autorizzazioni del richiedente sulla base di queste informazioni, delle sue credenziali e delle proprie policy.

I comandi a disposizione dell'utente sono `voms-proxy-init`, `voms-proxy-destroy` e `voms-proxy-info`. L'invocazione di `voms-proxy-init` comporta le seguenti operazioni. Viene creato un proxy localmente, successivamente viene inoltrato al server VOMS che aggiunge i campi al proxy e lo firma.

# Cloud Computing & Storage

Le tecnologie di cloud computing hanno avuto negli ultimi anni largo impiego nelle realtà aziendali ed è sempre più probabile che nel futuro diventeranno parte integrante delle infrastrutture per il calcolo scientifico (attualmente già in fase di sperimentazione).

Storicamente Amazon è stata fra le prime aziende a sperimentare queste tecnologie ed a offrire servizi collegati. Ciò nacque dalla esigenza di sfruttare, dunque vendere, le risorse di calcolo in eccesso a propria disposizione, le quali erano state acquistate per affrontare i picchi di domanda sui propri store online in certi periodi dell'anno.

La soluzione ideata fu di mettere a disposizione queste risorse di calcolo ad altre aziende che avessero simili problemi di picchi di carico. Naturalmente una soluzione di questo genere comportò l'ideazione di tecnologie atte a rendere utilmente disponibili queste risorse ed isolare fra loro gli utenti di queste ultime. Oggi, dopo alcuni anni di sviluppo, queste tecnologie vengono complessivamente indicate come **cloud computing**.

Nel caso del cloud computing la *tecnologia abilitante*, ovvero senza la quale questi sviluppi non sarebbero stati possibili, è stata la **virtualizzazione**.

## Virtualizzazione

Informalmente, con virtualizzazione si intende la creazione di una versione virtuale di qualcosa, che può essere ad esempio una piattaforma hardware, un dispositivo o un sistema operativo. Questo

comporta la creazione di un livello di astrazione intermedio che riproduce certe funzionalità e nasconde i dettagli di ciò che avviene a più basso livello, interfacciandosi con i livelli di astrazione superiore come farebbe l'oggetto che è stato virtualizzato.

Un esempio di questo genere di tecnologie è la **Java Virtual Machine** (JVM), ovvero un processore virtuale in grado di eseguire un set di istruzioni detto **Java bytecode**. In questo caso se si vuole eseguire codice Java, quest'ultimo deve essere compilato in bytecode ed eseguito su JVM. Questo risolve (in teoria) non solo il problema della portabilità del codice, che in questo modo è indipendente dalla piattaforma, ma anche dello stesso eseguibile, ovvero il java bytecode che può essere eseguito su architetture e sistemi operativi diversi purché sia stata implementata per questi ultimi una JVM.

Nel discorso che segue per virtualizzazione si intenderà *virtualizzazione dell'intera piattaforma hardware*, ovvero dove generalmente il livello di astrazione creato si frappone fra l'hardware fisico ed il sistema operativo.

### Tipi di virtualizzazione

Si possono distinguere diversi tipi di virtualizzazione ed il primo che è possibile evidenziare è la **emulazione**. Si parla di emulazione quando i sistemi virtuali possiedono una architettura diversa dall'ospite ed usano un differente set di istruzioni. In altri termini nella emulazione l'hardware virtuale viene simulato (a spese di maggiori risorse, dunque **overhead**).

Il caso in cui i sistemi virtualizzati (*guest*) abbiano la stessa architettura dei sistemi ospite (*host*) si può distinguere ancora in **virtualizzazione completa** e **paravirtualizzazione**.

Nel caso della virtualizzazione completa il sistema operativo ospite non è a conoscenza di essere eseguito su di una macchina virtuale e vede le stesse interfacce di una macchina fisica e dunque funziona senza alcuna modifica. In questo caso il livello software fra le macchine virtuali (VM) ed la macchina fisica viene detto **hypervisor** ed ha l'onere di tradurre (o inoltrare) le *system call* delle prime verso quest'ultima.

Nel caso della paravirtualizzazione i sistemi operativi delle VM sono a conoscenza della virtualizzazione dell'hardware sottostante e possono pertanto essere messe in atto delle ottimizzazioni. Queste consistono in modifiche della interfaccia, tramite chiamate speciali dette **hypercall**, permettendo ad esempio di eseguire istanze di codice in un ambiente non virtuale, dunque direttamente sull'hardware.

Gli hypervisor si distinguono a loro volta fra

- Tipo 1 o **bare metal**
- Tipo 2 o **hosted**

Gli hypervisor bare metal vengono eseguiti direttamente sull'hardware e le funzionalità di virtualizzazione si fondono insieme ad un kernel specifico in un SO *leggero* (dedicato alla virtualizzazione, non general purpose) che include anche tutti i driver per le periferiche ed il sistema di gestione/realizzazione delle macchine virtuali. Sono esempi di hypervisor bare metal: XEN, VMware vSphere, Parallels Server 4, Bare Metal e Hyper-V.

Gli hypervisor hosted invece sono normali applicativi eseguiti all'interno di un sistema operativo, ed in particolare installati ed eseguiti in user space, in grado di fornire funzionalità di virtualizzazione. Un esempio di questi ultimi è Oracle VirtualBox.

Esiste in realtà un terzo genere di hypervisor, in qualche misura ibrido rispetto ai primi due, detto **KVM** (Linux Kernel-based Virtual Machine), il quale è una infrastruttura di virtualizzazione che trasforma il kernel Linux in un hypervisor. In questo caso le funzioni di virtualizzazione vengono offerte da applicativi in user space che tuttavia fanno uso di interfacce esposte da un modulo del kernel Linux. Inoltre KVM può approfittare di alcune estensioni hardware dedicate alla virtualizzazione, in particolare legate alla gestione della memoria, ed in genere supportate dalle ultime generazioni di processori che permettono un migliore sfruttamento delle risorse (riduzione overhead).

### Vantaggi della virtualizzazione

L'avanzamento tecnologico degli ultimi anni ha prodotto una situazione in cui (generalmente ed in media) gli applicativi non sono più in grado di saturare le risorse hardware disponibili, sia nel caso di personal computing che, in maniera ancora più accentuata, in ambito aziendale.

Infatti si stima che un moderno server venga sfruttato solo al 15-20% e pertanto la virtualizzazione offre, permettendo di ospitare 3 o 4 machine virtuali sullo stesso hardware, il vantaggio di un miglior sfruttamento delle risorse, chiamato in ambito aziendale **consolidamento dei server**, e dunque la **riduzione dei server fisici**. Quest'ultima comporta la riduzione dei consumi energetici (quindi la necessità di impianti di raffreddamento meno potenti), dei **guasti hardware**, dei tempi tecnici per il montaggio ed il cablaggio, del numero di armadi (*rack*) e pertanto l'abbattimento dello spazio dedicato in sala macchine per questi ultimi ed il loro relativo cablaggio.

Inoltre il software, ed in particolar modo il sistema operativo in esecuzione su una macchina, è in genere strettamente legato all'hardware su cui viene eseguito. Pertanto se, ad esempio per un guasto hardware, si deve migrare una installazione da una macchina ad un'altra, si dovrà spendere del tempo nella configurazione del nuovo hardware. A questo proposito la virtualizzazione offre il vantaggio della **indipendenza hardware**, infatti il sistema operativo guest non si interfaccia con l'hardware fisico ma con un livello di astrazione di quest'ultimo e l'amministratore potrebbe spostare o clonare una macchina virtuale su altre macchine fisiche che eseguano lo stesso ambiente di virtualizzazione senza ulteriore lavoro (se si esclude la configurazione di quest'ultimo).

L'indipendenza dallo hardware in alcuni casi non rappresenta semplicemente una semplificazione della amministrazione straordinaria di sistema ma una necessità. Il tipico esempio è il supporto di vecchie applicazioni (**legacy**), ad esempio sviluppate per DOS, non in grado di supportare hardware più recente. In questi casi gli ambienti di virtualizzazione permettono l'esecuzione anche di sistemi *legacy* permettendo ai responsabili IT di liberarsi del vecchio hardware non più supportato e più soggetto a guasti.

Un ulteriore vantaggio è la **standardizzazione del runtime**, ovvero mettere appunto ambienti di sviluppo, postazioni di lavoro, server di posta, etc., omogenei in maniera semplicemente riproducibile, e la **creazione di ambienti di test**, ovvero di ambienti isolati che possano essere facilmente creati, distrutti o ripristinati ad uno stato precedente, per testare software.

### Svantaggi della virtualizzazione

Una delle caratteristiche richieste fin da principio alle tecnologie di virtualizzazione è l'isolamento dei dati e dei processi dei sistemi virtualizzati. In particolare questo aspetto, come si vedrà in seguito, emerge preponderantemente nei casi in cui gli utenti dei sistemi virtualizzati acquistano come servizio

l'infrastruttura di virtualizzazione da terzi e pertanto si richiede che sullo stesso hardware coesistano diversi utenti senza collidere.

Tuttavia l'isolamento è realizzato dal software di virtualizzazione ed i **problemi di sicurezza** che derivano da bachi di quest'ultimo sono fra i principali svantaggi di queste tecnologie.

Inoltre l'introduzione di un livello software fra i sistemi virtualizzati e lo hardware ha come inevitabile contropartita una **diminuzione delle performance** (seppure minima, grazie al perfezionamento di queste tecnologie ed al loro elevato grado di maturità). Concretamente questa riduzione delle performance consiste in un overhead di esecuzione praticamente non rilevabile ed in una **riduzione del throughput di I/O** su disco e di rete misurabili (meno importanti nel caso di paravirtualizzazione e trascurabili per quest'ultimo per traffico di rete).

## Cloud Computing

Le peculiarità delle tecnologie di virtualizzazione hanno introdotto negli ultimi anni una modifica radicale degli approcci al calcolo, con applicazioni al calcolo scientifico.

Storicamente in principio, in ambito aziendale ma anche scientifico, il calcolo era affidato ai *mainframe*, grandi macchine con differenze architetturali sostanziali rispetto ai server comunemente diffusi oggi, le cui risorse, ovvero tempo di calcolo, venivano allocate in maniera molto critica – sistemi di questo genere continuano ad esistere ad esempio in banche o agenzie governative.

Successivamente, grazie agli sviluppi tecnologici ed in particolare la miniaturizzazione dell'elettronica, si affermò un modello di calcolo basato su hardware in pieno controllo dell'utente (*personal computers*), successivamente sull'accesso a risorse di calcolo remote (*client-server computing*) ed in particolare (successivamente) sull'accesso a documenti remoti (*Web*).

Negli ultimi dieci anni, in particolare grazie alle tecnologie di virtualizzazione, si è sviluppato un nuovo paradigma basato sull'accesso a risorse di calcolo remote con caratteristiche innovative detto **cloud computing**.

### Traditional infrastructure model

Sia in ambito aziendale che scientifico la crescita nel tempo di una istituzione in genere comporta una maggiore richiesta di risorse di calcolo, ad esempio per un aumento della base di utenti.

L'approccio tradizionale a questo problema era di acquistare un surplus di risorse, rispetto alla domanda attuale, per tenere il passo delle delle richieste future. Le previsioni sul tasso di crescita e la frequenza degli aggiornamenti determinavano l'entità di questo surplus. Si osserva che, anche nel caso ideale di una crescita monotona con previsioni affidabili sul tasso di crescita, questa è una soluzione non ottima in quanto per certi lassi di tempo vengono allocate (dunque pagate) risorse che rimangono inutilizzate.

Nei casi reali questo genere di soluzioni presenta ulteriori svantaggi. Infatti in questi ultimi la domanda di risorse, pur potendo avere in media semplice (ad esempio una crescita lineare), è in genere soggetta a fasi di crescita e contrazione, con un tasso non prevedibile. Le conseguenze di queste oscillazioni sono dei **surplus con perdite non accettabili** (spese troppo ingenti rispetto al consumo di risorse) e **periodi di deficit**.

Da questa discussione emerge che i problemi di questo approccio sono la lentezza degli interventi di aggiornamento delle infrastrutture e la loro irreversibilità, nel senso che una volta acquistate non è possibile (o non è conveniente) cederle o smantellarle e rappresentano un costo fisso.

Il **cloud computing** è un insieme di tecnologie che permette un approccio al problema delle risorse radicalmente differente, rendendo possibile ad esempio approntare una infrastruttura di calcolo in tempi dell'ordine di alcuni minuti, contro le diverse settimane necessarie nel caso di IT tradizionale.

### Definizione di cloud computing

In letteratura la definizione di riferimento per le tecnologie di cloud computing è quella data dal NIST, che in sintesi definisce queste ultime come

> **fornitura** di tecnologia di **informazione e comunicazione** (ICT) come **servizio**

Si osserva che concettualmente il termine di maggior peso in questa definizione è *servizio*, il quale racchiude il contenuto innovativo di queste tecnologie (e dei paradigmi che le rendono possibili).

La definizione del NIST inoltre prevede alcuni punti

- **On demand self service:** l'utente del servizio deve essere in grado di ottenere in maniera semplice, trasparente ed automatica risorse di calcolo, come ad esempio CPU time o dischi, alla bisogna, senza la necessità di una interazione diretta con gli amministratori dei service provider o di un intervento umano.
- **Broad network access:** il servizio deve essere distribuito ed accessibile tramite la rete
- **Resource pooling:** le risorse di calcolo, fisiche o virtuali, disponibili al service provider devono essere distribuite e riassegnate dinamicamente agli utenti in base alla domanda senza che questi ultimi collidano, ovvero realizzando una condizione di **isolamento** dei dati e dei processi di questi ultimi in un ambiente fortemente **multi-tenant**.
- **Rapid elasticity:** il servizio deve essere non solo in grado di fornire risorse di calcolo *elasticamente*, ovvero in base al carico, ma specialmente in maniera **rapida**, ovvero adattandosi a quest'ultimo in tempi brevi ed idealmente tenendo il passo della domanda in tempo reale.
- **Measured service:** i service provider devono essere in grado di misurare l'erogazione del servizio, utilizzando metriche di un livello di astrazione adeguato al genere di servizio, in modo da poter sia addebitare eventuali costi di servizio che, più in generale, ottimizzare le risorse fra gli utenti, ad esempio in base alle priorità assegnate alle organizzazioni a cui appartengono.

Una analogia utile a cogliere alcuni aspetti del cloud computing è quella con il noleggio auto:

- l'utente effettua una prenotazione telefonica o online senza la necessità di un intervento umano (*self service*) e accendendo un contratto temporaneo relativo alla prestazione particolare (*on demand*)
- il service provider (agenzia di autonoleggio) possiede una rete di distribuzione geograficamente distribuita (*broad network access*)
- le risorse (il parco vetture) viene riorganizzato per garantire il servizio in maniera trasparente all'utente (*resource pooling*)
- i dettagli del noleggio sono stabiliti in base alla domanda e possono essere semplicemente modificati (*rapid elasticity*)
- l'addebito agli utenti viene effettuato in base al consumo (*measured service*)

### Service models

Il paradigma reso possibile dalle tecnologie di cloud computing presenta delle analogie con altri sistemi di condivisione di risorse di calcolo geograficamente distribuite, come ad esempio la GRID, tuttavia *nel cloud computing il focus è sul servizio* e questa, che è una caratteristica distintiva di queste tecnologie, ne ha sancito il successo in ambito aziendale.

Infatti nel caso del grid computing la risorsa messa a disposizione dalla infrastruttura è essenzialmente la capacità di eseguire un applicativo, ovvero di sottomettere un job ad un **batch system**. Quest'ultima è una differenza sostanziale rispetto al cloud computing propriamente detto in cui il panorama dei servizi offerti è molto più ricco e variegato, comportando un profondo divario nella implementazione di queste due tecnologie oltre che nell'approccio al problema.

Inoltre il grid computing si è affermato solamente all'interno della comunità scientifica mentre il cloud computing ha suscitato grande interesse nelle realtà aziendali ed è oggi una tecnologia matura ampliamente adottata in questi contesti, con alcune sperimentazioni in ambito scientifico.

Le tecnologie di cloud computing vengono in genere classificate in base alla tipologia di servizi che vengono erogati e si distinguono, in prima istanza, in tre livelli

- Infrastructure as a Service (**IaaS**)
- Platform as a Service (**PaaS**)
- Software as a Service (**SaaS**)

e successivamente il concetto di *Something as a Service* è stato esteso ad altri sottolivelli e servizi specifici.

### Infrastructure as a Service

Nel caso di IaaS il service provider mette a disposizione essenzialmente un ambiente di virtualizzazione e fornisce un certo numero di macchine virtuali, con certe caratteristiche, in base alla domanda dell'utenza. Pertanto in questo livello di servizio tutto ciò che arriva fino al livello del sistema operativo è di competenza del service provider, mentre ciò che viene installato sulla macchina (middleware, software, etc.) diventa responsabilità dell'utente.

Semplificando, la differenza rispetto alla virtualizzazione su hardware in proprio controllo è che è possibile ottenere in pochi passaggi e su richiesta una macchina virtuale, con CPU, memoria e dischi richiesti, con un sistema operativo scelto senza dover conoscere i dettagli, o dover spendere energie nell'implementazione, della infrastruttura sottostante.

In questo caso l'utente può configurare il servizio assemblando virtualmente macchine, dischi, componenti di rete, etc, in maniera automatizzata senza interagire direttamente con gli amministratori del datacenter in cui è fisicamente ospitato lo hardware (ad esempio tramite una interfaccia Web). Tuttavia questo da solo non sarebbe sufficiente a definire questo genere di servizi cloud computing, infatti ciò che permette di definire quest'ultimo cloud computing è la **flessibilità** del servizio, in accordo al concetto più generale di *rapid elasticity* (che diventa **dinamicità** o **scalabilità dinamica** del caso SaaS o PaaS).

Ad esempio un ipotetico servizio in cui sia possibile noleggiare una macchina virtuale con certe caratteristiche accendendo un contratto specifico per quella soluzione (*managed hosting*) non

potrebbe essere definito cloud computing: nel caso di IaaS è importante che l'utente sia in grado di modificare le caratteristiche del servizio, ovvero l'infrastruttura e dunque CPU, dischi, etc., dinamicamente.

Questi servizi sono caratterizzati da:

- Ambienti multitenant virtualizzati e sistemi critici per l'isolamento dei dati e dei processi degli utenti
- Addebito in genere stabilito in base al wall time, non in base all'uso
- Nelle implementazioni reali, possibilità di accoppiare servizi per l'installazione e la manutenzione dei sistemi operativi o del runtime
- Accesso con diritti di amministrazione da parte dell'utente

Lo svantaggio peculiare della IaaS consiste nella lentezza e nella complessità nella creazione o modifica di macchine virtuali. In risposta a queste difficoltà molti utenti hanno preferito sistemi più evoluti, nella direzione di PaaS e SaaS, in cui i sistemi scalino dinamicamente, in maniera più automatica e trasparente all'utente, e IaaS è rimasto un servizio riservato ad utenti con esigenze piuttosto specifiche.

### Platform as a Service

Nel caso Platform as a Service quello che fornisce il service provider è un sistema dove il runtime è già disponibile, ovvero dove l'applicazione oggetto del servizio è pronta per essere usata dallo sviluppatore, che in questo contesto è l'utente del servizio. Ad esempio prima di poter sviluppare codice in C è necessario istallare un compilatore, un debugger ed in genere delle librerie, nel caso di PaaS allo sviluppatore viene fornito un ambiente pronto per la compilazione e l'esecuzione del codice, senza che quest'ultimo sia a conoscenza dei dettagli dell'infrastruttura che rende questo possibile o dell'onere di doverla preparare.

Nel caso PaaS il service provider fornisce all'utente tutti gli strumenti per lo sviluppo remoto delle sue applicazioni in maniera semplice (ad esempio tramite l'uso di interfacce Web) e la possibilità di gestire in maniera semplice l'intero ciclo di vita di queste ultime.

Sinteticamente Platform as a Service è un metodo per l'erogazione di risorse di calcolo attraverso una piattaforma, ovvero tramite una infrastruttura di componenti hardware e software disponibile all'uso e di cui non è necessario conoscere i meccanismi interni. Tutto ciò permette in questo modo di abbattere i tempi ed i costi per lo sviluppo ed il collaudo di applicazioni ed ha segnato il successo di questo approccio. Tuttavia questo da solo non sarebbe sufficiente per parlare di questi servizi come cloud computing, infatti la caratteristica principale di questi servizi è di scalare dinamicamente in base alle richieste degli applicativi sottomessi.

Questo tipo di servizi è caratterizzato da:

- Ambienti multitenant
- Scalabilità dei sistemi

Il principale svantaggio per l'utenza nella sottoscrizione a questi servizi è la dipendenza degli applicativi dalla piattaforma particolare e dunque la difficoltà, o addirittura l'impossibilità, di migrare verso un'altro service provider in un secondo momento.

Come già osservato i service provider di IaaS hanno gradualmente inglobato servizi di PaaS ed oggi è molto difficile trovare un service provider che faccia esclusivamente PaaS.

### Software as a Service

L'ultimo livello nella scala dei servizi offerti dal service provider è il caso **Software as a Service**, o **cloud application**. In questo caso l'utente del servizio è effettivamente l'utente finale ed a quest'ultimo viene fornito direttamente l'applicativo. Alcuni esempi noti sono **Gmail** e **Google Documents**.

Questi applicativi sono probabilmente la forma più popolare di cloud computing e sono in genere di utilizzo molto immediato, infatti sono in genere accessibili direttamente tramite un browser web e rimuovono la necessità di installazione e configurazione del software su computer individuali. Comportano inoltre notevoli semplificazioni anche per i fornitori del servizio, specialmente nell'erogazione del supporto dal momento che tutta l'infrastruttura (applicativi, rutime, sistema operativo e hardware) sono loro direttamente accessibili.

Sinteticamente si può dire che SaaS è un metodo per la distribuzione di applicativi che fornisce un accesso remoto alle funzionalità di questi ultimi, usualmente tramite interfacce Web, a diversi utenti forniti di licenza. Questi servizi sono in genere caratterizzati da

- Ambienti multitenant e sistemi per l'isolamento dei dati degli utenti
- Universalmente accessibili, ovvero entro certi limiti indipendenti dal software o hardware particolare dell'utenza
- Addebiti stabiliti in base all'uso
- Grande scalabilità dei sistemi per l'erogazione del servizio (intrinseca nella gestione del servizio e completamente delegata al fornitore)
- Sistemi di gestione delle licenze quindi, termini più astratti, gestione critica dei permessi
- Isolamento dei dati degli utenti

I principali problemi per l'utenza di questi servizi sono legati all'accesso, la persistenza ed al possesso ai dati, con ovvie implicazioni per la privacy degli utenti. Infatti **il servizio ed i dati sono strettamente collegati**, cosa che comporta notevoli conseguenze come ad esempio la possibile perdita dei dati in caso di interruzione del servizio.

### Alcuni esempi di service models

Queste definizioni sono piuttosto astratte e generali, è possibile vedere con alcuni esempi concreti la loro declinazione nel mondo reale.

### SaaS case study: Salesforce.com e Google Apps

Un importante esempio di Software as a Service è salesforce.com che fornisce software per gestire le relazioni con i clienti per le aziende (*business-to-business*). In questo caso l'utente, ovvero l'azienda, compone le proprie applicazioni assemblando pacchetti relativi a particolari servizi (modulo gestione clienti, modulo gestione fornitori, gestione del magazzino, etc.).

Nella esperienza comune Google Apps rappresenta un esempio classico di SaaS rivolto agli utenti finali, ovvero i consumatori. Tuttavia questi servizi (oltre a molti altri) sono rivolti anche alle aziende, alle quali viene vengono offerti in una forma simile a quella rivolta ai consumatori ma con la possibilità di personalizzazioni.

Non solo, infatti alcuni di questi servizi vengono offerti gratuitamente ad enti pubblici o di rilievo in cambio della pubblicità derivante dalla adesione ai servizi (ed alla raccolta di dati).

Si osserva comunque che Google in realtà fornisce PaaS, infatti le applicazioni offerte all'utenza condividono le stesse tecnologie, che vengono messe a disposizione degli sviluppatori: questi toolkit incarnano il concetto di PaaS.

Si osserva come questo esempio metta bene in luce le limitazioni dell'approccio PaaS per gli sviluppatori, infatti un ipotetico applicativo che faccia uso dei servizi di geolocalizzazione di Google diventa dipendente da quest'ultimo.

#### IaaS e PaaS case study: Amazon AWS e Windows Azure

Il caso di studio di Amazon AWS è tra i più significativi in quanto la gamma dei servizi offerti è così estesa da poter illustrare tutti gli aspetti di IaaS e PaaS.

Storicamente il primo dei servizi offerti da Amazon Web Services è stato *Amazon Simple Storage Service* (**S3**), ovvero un sistema per la scrittura e la lettura di dati diverso dal semplice storage remoto (2006). Infatti questo servizio possedeva delle peculiarità che lo rendevano un vero esempio di PaaS. Ad esempio l'accesso ad i dati avveniva programmaticamente: ovvero all'interno di un linguaggio di programmazione tramite delle apposite librerie.

Un'altra caratteristica innovativa di S3 disponibile fin dalla nascita del servizio era quella di richiedere che i dati venissero conservati in data center geograficamente distribuiti.

In un secondo momento è stato varato un servizio per la creazione di macchine virtuali, ovvero *Amazon Elastic Computing Cloud* (**EC2**), che rappresenta la formulazione classica di IaaS.

**Windows Azure** (in precedenza **Microsoft Azure**) è un'altro esempio di di PaaS e IaaS. Quest'ultimo si differenzia rispetto ad AWS nel fatto di essere fortemente orientato allo sviluppo di nuove applicazioni nativamente in cloud.

#### Deployment and isolation models

Quanto presentato fin'ora potrebbe indurre a credere che i casi d'uso delle tecnologie di cloud computing riguardino solamente alcune grandi aziende (Amazon, Google, etc.), tuttavia non è questo il caso.

Infatti cloud computing è appunto una tecnologia ed i suoi impieghi possono essere classificati in base a 3 caratteristiche indipendenti: *service model*, *deployment model* e *isolation model*.

I servizi degli esempi precedenti (EC2, Windows Azure,...) sono infrastrutture di cloud che secondo lo standard vengono definite pubbliche, nel senso che chiunque può accedere a questi ultimi dietro compenso economico.

La caso opposto è quella di qualcuno che costruisca una infrastruttura di questo tipo per se stesso e quindi permetta l'accesso a certi utenti particolari. Benché contro-intuitivo, questo genere di soluzioni viene scelto dalla gran parte delle istituzioni sia in ambito accademico che aziendale: ovvero queste istituzioni utilizzano tecnologie di cloud computing su risorse che sono completamente sotto il loro controllo.

In generale si distinguono diversi modelli di erogazione dei servizi:

1. **Public cloud:** L'utente ha diritto all'accesso di risorse di calcolo remote che sono allocate dinamicamente su richiesta da quest'ultimo, attraverso applicazioni web o API, via internet a seguito della accensione di un contratto. Il service provider addebita all'utente una somma calcolata in base all'utilizzo delle risorse.
2. **Private cloud:** L'utente accede a risorse di calcolo remote che sono fornite da una istituzione di cui fa parte. Spesso l'adozione di queste tecnologie è il primo passaggio prima di una transizione a modelli di Public Cloud.
3. **Hybrid cloud:** L'utente ha accesso a risorse sia locali che remote e l'accesso a risorse remote è implementato tramite tecnologie di cloud computing. In genere queste soluzioni sfruttano le tecnologie di cloud computing per funzioni specifiche o in situazioni di carico di lavoro straordinario. Un caso ricorrente, sia in ambito aziendale che accademico, è di fare affidamento a risorse di cloud computing pubbliche in determinati periodi dell'anno ed un esempio in fisica delle alte energie è l'uso di AWS da parte di CMS. In questo modo è stato possibile affrontare richieste di calcolo particolarmente intensive e superiori alle capacità dell'esperimento, acquistando all'asta istanze di calcolo EC2 (**Spot Istances** e **Spot Fleet**), in quanto per Amazon i periodi di inattività rappresentano un costo senza profitto.
4. **Community cloud:** L'utente ha accesso a risorse remote messe a disposizione da una organizzazione di diverse istituzioni le quali collaborativamente condividono una infrastruttura comune di cloud computing. Rispetto al Public cloud le differenze consistono nel minor numero di utenti e nel sistema di addebito del servizio, invece rispetto al Private cloud è possibile partizionare i costi di implementazione e manutenzione del servizio fra le diverse istituzioni.

Per isolamento si intendono una serie di aspetti come:

- **Segmentazione delle risorse:** gli utenti devono avere pieno accesso alla quantità di risorse che gli è stata destinata e non altre
- **Protezione dei dati:** impedire l'accesso o la scrittura di dati da parte di utenti senza permessi
- **Sicurezza delle applicazioni:** impedire l'esecuzione o la modifica di applicazioni da parte di utenti senza permessi
- **Auditing:** monitoraggio degli accessi ai propri file o applicativi da parte degli utenti

ed i modelli di isolamento sono essenzialmente due:

1. Infrastrutture dedicate, con singolo utente o diversi utenti appartenenti alla stessa istituzione
2. Infrastrutture multitenant, ovvero con diversi tipi di utenti per cui l'isolamento è critico

### Considerazioni

#### Interesse nei confronti del Cloud Computing

L'interesse nei confronti di una nuova tecnologie in genere segue un andamento che si divide in diverse fasi

1. Nascita della tecnologia e repentina crescita dell'interesse
2. Picco di aspettative e massima attenzione nei confronti di quest'ultima
3. Rapida decrescita e picco negativo dell'interesse
4. Maturazione della tecnologia con una più graduale crescita dell'interesse
5. Plateau di produttività caratterizzato da un interesse stabile nel tempo

La visibilità del cloud computing ha seguito questo andamento generale ed attualmente ci troviamo nel plateau di produttività.

### IaaS Cloud Stack più usati

Allo stato attuale esistono diversi software (toolkit) detti **IaaS Cloud Stack** per realizzare una infrastruttura di cloud computing di tipo IaaS. Alcuni di questi sono proprietari, come **WMware** e **Microsoft Hyper-V**, mentre altri sono *open source*, **OpenStack**, **Apache CloudStack** ed **OpenNebula**, ma in ogni caso si tratta di software che può essere installato e configurato su hardware di proprio controllo per implementare un cloud privato (ma che può eventualmente essere reso successivamente pubblico).

A differenza di quest'ultimo il software impiegato da piattaforme di cloud pubblico, come ad esempio AWS, non può essere impiegato al di fuori della rispettiva piattaforma ed è accessibile solo tramite le IPA pubbliche.

### Avvantaggiarsi delle tecnologie cloud

Deve essere chiaro che l'utilizzo delle tecnologie cloud, ad esempio in campo di calcolo scientifico, non garantisce un vantaggio: infatti **è necessario che gli applicativi utilizzati siano sviluppati in modo da approfittare delle potenzialità offerte da queste tecnologie**. In particolare questo software deve avere le seguenti caratteristiche

- **Distribuito:** capacità di eseguire istanze di codice parallelamente su macchine con memoria non condivisa
- **Immutabile:** caratteristica di non modificare (idealmente) lo stato della macchina durante la propria esecuzione. Questo permette avvantaggiarsi del calcolo distribuito potendo eseguire istanze di porzioni di codice su diverse CPU senza che queste debbano comunicare tra loro, dunque aggirando le latenze di comunicazione tra i processi.
- **Failover in app:** capacità intrinseca ed automatica di gestione di una eventuale interruzione anomala dei sotto-processi, provvedendo meccanismi per farne partire altri
- **Scalabilità in app:** capacità di modificare automaticamente e dinamicamente la quantità di risorse utilizzate in base al carico ed alla disponibilità

### EGI Federated Cloud

La *EGI federated cloud* è una organizzazione che riunisce istituzioni dotate di sistemi di cloud computing privati ed aggrega questi sistemi offrendo alla comunità scientifica, in Europa e nel mondo, servizi di tipo IaaS come unico service provider.

Le idee dietro questo progetto internazionale sono essenzialmente l'adozione di **standard aperti** per le interfacce esposte all'utente, in modo da semplificare sia il *resource pooling* che l'integrazione con

gli applicativi degli utenti, e la **implementazione eterogenea** delle infrastrutture, lasciando libertà agli enti aderenti di utilizzare le tecnologie di loro discrezione dunque semplificando la messa in opera di nuove infrastrutture o l'integrazione di quelle preesistenti.

Un aspetto interessante di questo *federated cloud* è il **catalogo delle macchine virtuali**, ovvero un elenco pubblico di piattaforme o servizi sviluppati dagli utenti e sottomessi da questi ultimi. Le macchine virtuali in questo catalogo vengono validate dalla federazione ed in questo modo è possibile per gli utenti istanziare servizi sviluppati da altri utenti in un ambiente collaborativo aperto.

## Cloud Storage

Per **cloud storage** si intende l'utilizzo di tecnologie analoghe al cloud computing (dunque on demand, self service, network access, resource pooling, rapid elasticity, measured service) per l'archiviazione di dati.

A seconda dei casi, queste tecnologie si rendono necessarie

- in relazione al cloud computing, in quanto il problema del processamento dei dati non può prescindere da quello dell'archiviazione di questi ultimi ed inoltre non è possibile utilizzare soluzioni di archiviazione tradizionali in una infrastruttura di cloud computing
- per via di necessità specifiche degli utenti, come ad esempio la necessità di archiviare grandi moli di dati in maniera semplice e trasparente o di doverle condividere.

Si individuano tre tipologie di cloud storage nel contesto di servizi di cloud computing di tipo IaaS, escludendo lo storage volatile dello spazio disco della istanza di macchine virtuali (al riavvio di una istanza le modifiche vengono perse)

- **Posix Storage:** permette, oltre che l'accesso remoto, la condivisione di file fra più host. Una semplice implementazione, per infrastrutture di piccole dimensioni e non molto complesse, può consistere anche di un solo volume NFS (*Network File System*) montato sulle macchine che devono avere accesso ai file. Tuttavia man mano che le infrastrutture scalano in dimensione diventa necessario usare file system che possa aumentare dinamicamente le proprie dimensioni e migliorare le performance sfruttando parallelamente l'hardware a disposizione.
- **Block Storage:** espone alle macchine virtuali un dispositivo di archiviazione virtuale, con una interfaccia equivalente ad un disco fisico. Quest'ultima nei sistemi UNIX consiste in un file speciale, in particolare in Linux in un *bock device file*, tramite il quale le funzionalità dei dispositivi di archiviazione sono accedute tramite le chiamate di sistema per la lettura e scrittura di file. In questo caso l'utente, proprio come nel caso di un disco fisico, può decidere come formattare ed utilizzare il dispositivo. Inoltre in genere quest'ultimo è disponibile sulla rete e condiviso da più host contemporaneamente e l'utente è in grado di gestire questi dispositivi tramite la stessa interfaccia attraverso cui amministra le macchine virtuali.
- **Object Storage:** non permette l'accesso ai dati di tipo Posix (open, seek, write, close) ma mette a disposizione una serie di IPA per l'accesso e la modifica di questi ultimi. In questo caso non è prevista una struttura a directory ma una organizzazione ad **oggetti**, file binari eventualmente provvisti di metadati, e **bucket**, contenitori in cui scrivere/leggere oggetti (ma non altri bucket, non è infatti possibile costruire alberi di bucket ed esiste un solo livello). Oltre alle IPA è in genere possibile accedere ai file (oggetti) tramite web services sfruttando il protocollo HTTP. Un tipico caso d'uso di questa tipologia di cloud storage è la memorizzazione delle immagini delle macchine virtuali nei Market Place o negli Image Service.

#### Esempi di tecnologie disponibili per il cloud storage

Nella seguente tabella sono elencati alcuni esempi di tecnologie disponibili per le varie tipologie di cloud storage

| | **Open Stack** | **Amazon** | **Others** |
|---|---|---|---|
| **Posix Storage** | NA | NA | GlusterFS, Lustre, GPFS, CEPH |
| **Block Storage** | Cinder | EBS | CEPH, iSCSI storage |
| **Object Storage** | Swift | S3 | CEPH, GlusterFSUFO |

mentre nella seguente tabella sono riassunte alcune caratteristiche delle tipologie di cloud storage per il caso particolare dello stack per il cloud computing open source **Open Stack**

#### Caratteristiche di un cloud storage

Un servizio di cloud storage possiede le stesse proprietà caratterizzanti di un servizio di cloud computing, ovvero *on demand & self service*, *broad network access*, *resource pooling*, *rapid elasticity* e *measured service*. Tuttavia nel caso del cloud storage si aggiungono alcune richieste

- **High availability:** ovvero la ridondanza dei dati e la capacità dei sistemi garantire l'accesso ai dati nonostante la failure di un singolo dispositivo hardware.
- **High Level API:** possibilità di accesso ai dati, oltre che eventualmente tramite standard POSIX, anche tramite una interfaccia da remoto, standard ed universale (in genere un *web service*)

In una tecnologia di cloud storage fra gli aspetti di maggior rilievo ci sono le interfacce, ovvero il modo con cui viene esposto l'accesso ai file ai livelli di astrazione superiori.

# Analisi di Big Data

Per **Big Data** si intendono *dataset* le cui dimensioni trascendono le capacità di hardware e software convenzionali di catturare, gestire o processare dati in tempi ragionevoli. Tuttavia bisogna tenere presente che il concetto di big data è in costante evoluzione e la crescita delle dimensioni effettive dei dataset associati è molto veloce (di alcuni ordini di grandezza in pochi anni).

## Apache Hadoop

**Apache Hadoop** è un framework software scritto in Java per l'analisi di big data su cluster con le seguenti caratteristiche

- **Scalabile** : nuovi nodi possono essere aggiunti o rimossi *a caldo*, senza modifiche all'infrastruttura o alle applicazioni che accoglie
- **Efficiente** : rendendo possibile il calcolo su un sistema ad elevato parallelismo (*massively parallel computing*) senza l'impiego di hardware specifico

- **Flessibile** : possibilità di incorporare e scrivere applicazioni per qualsiasi genere di dato, strutturato o meno.
- **Tollerante ai guasti** : in caso di malfunzionamento di un nodo, perdita di una replica di un file o di un malfunzionamento di un disco, il sistema è in grado di gestire automaticamente il problema

Il framework Hadoop comprende

- **Hadoop Common** : le utility e le librerie comuni necessarie agli altri servizi
- **Hadoop Distributed File System (HDFS)** : un file system in grado di utilizzare risorse distribuite e massimizzare il volume di dati accessibili alle applicazioni
- **Hadoop YARN** : un framework per lo scheduling dei job e per la gestione delle risorse di archiviazione
- **Hadoop MapReduce** : un sistema per il calcolo parallelo basato su YARN ed ottimizzato per grandi dataset

I problemi riscontrati nella analisi di grandi dataset sono in genere legati al temo di CPU e alla lettura dei dati. La soluzione offerta da Hadoop consiste in un unico framework per la gestione integrata dei processi e dei dati, che cerca di porre rimedio a questi problemi e di risolvere le inefficienze di un approccio tradizionale.

L'osservazione principale è che trasferire i dati verso le CPU (come nel caso tradizionale di macchine separate per l'archiviazione) è sia costoso, per via delle infrastrutture di rete, che inefficiente, per via delle latenze. Una possibile soluzione è scalare il numero di CPU *assieme* ai dischi e tenere queste ultime il più vicino possibile ai dati da analizzare. Si osserva tuttavia che uno degli svantaggi di questo approccio è che la gestione delle failure diventa più delicata.

## HDFS

L'utente può interagire con HDFS in diversi modi

- montare dischi in userspace (FUSE) simulando un file system di basso livello
- monitorare il file system utilizzando una interfaccia Web
- utilizzando le API in Java
- utilizzando la shell nativa di HDFS da terminale

L'architettura del sistema di archiviazione in un cluster Hadoop tramite HDFS comprende i seguenti componenti

- Uno o più **client** che possono inoltrare richieste al sistema tramite le interfacce citate
- Un *unico* **namenode** che coordina l'archiviazione e la lettura dei dati
- Diversi (usualmente centinaia o migliaia) di **datanodes**, che contengono fisicamente i dati

### Scrittura di file in HDFS

La scrittura di un file su HDFS avviene in più fasi

1. **Request from user** L'utente chiede al client di scrivere un file e fornisce ridondanza desiderata, **replication factor**, e le dimensioni dei blocchi in cui questo verrà diviso dal sistema, **blocksize**.

2. **Client ask namenode** Il client divide il file in blocchi delle dimensioni stabilite e comunica al namenode ridondanza e numero di blocchi
3. **Namenode assignment** Il namenode determina i datanode su cui archiviare i blocchi. Infine restituisce gli indirizzi dei datanode al client (in ordine di distanza dal client crescente)
4. **Client writes data** Il client comincia a scrivere i dati sul primo datanode.
5. **Replication pipeline** Il primo datanode inoltra i dati al secondo e così via, fino all'ultimo datanode
6. **Completion and Inform namenode** Quando l'ultimo datanode finisce di ricevere i dati lo comunica al namenode.
7. **Namenode stores metadata** Ad operazioni concluse il namenode conserva i metadati dei file archiviati sul proprio disco (indirizzi dei namenode, etc.).

TODO

l'idea di hadoop è quello di risolvere i problemi classici che avvengono in un data center. Immaginiamo che ci siano tre copie del mio file e ho due armadi. Conviene che le mie tre copie siano distribuite almeno una su un altro armadio in modo che se uno si spegne comunque un'altra copia sta sull'altro.

Chiedo la scrittura ad hadoop specificando la block size e il numero di repliche. Il client specifica queste cose al namenode che è un server che controlla tutto. Il namenode controlla dove è disponibile spazio e scrive il file sul primo datanode che copia poi sull'altro e cosi via. Questo riduce la complessità del client appoggiando una parte dei compiti sul datasystem. Una volta che tutti e tre hanno comunicato al namenode che hanno scritto la procedura è finita. Quindi la velocità effettiva è tre volte più lenta rispetto alla copia singola. Il sistema quindi ha completato la scrittura di quel blocco e si passa al blocco al successivo.

Quando il client vuole leggere un file va al namenode e chiede il file. Il namenode comunica con i datanode in parallelo le richieste sui file che comunicano i blocchi presenti di quel file. Il vantaggio è che se un nodo per esempio è occupato o non funziona il sistema può recuperare dagli altri nodi il dato cercato e questo per l'utente è completamente trasparente.

Riguardo la placement strategy lui fa questo quando si richiede di fare 3 copie: una replica in un rack e un altra replica in 2 rack. Questo fa si che si minimizzano quanto più possibile l'uso degli up link di rete utilizzando la rete con la connessione interna del rack che è meno costosa. Quindi comunque ho la ridondanza senza aver stressato troppo l'up link. Inoltre questi sistemi vengono utilizzati per scrivere poco e leggerli molte volte. Quindi avere due copie all'interno dello stesso rack fa si che quando vado a fare analisi potrei leggere al doppio della velocità utilizzando le due copie all'interno dello stesso rack.

## Map Reduce

TODO