2
COLLANA
FISICA
FISICA FONDAMENTALE

Federico Ruggieri

ACQUISIZIONE DATI E CONTROLLO DI ESPERIMENTI

Roma TrE-Press
2024

ROMA TRE
UNIVERSITÀ DEGLI STUDI

Dipartimento di Matematica e Fisica

Università degli Studi Roma Tre
Dipartimento di Matematica e Fisica

2
COLLANA
FISICA
FISICA FONDAMENTALE

Federico Ruggieri

ACQUISIZIONE DATI E CONTROLLO DI ESPERIMENTI

revisione di

Biagio Di Micco, Paolo Branchini

Roma TrE-Press
2024

Collana FISICA

La Collana FISICA è stata creata su iniziativa dei docenti del Dipartimento di Matematica e Fisica al fine di sostenere scientificamente il progetto editoriale Roma Tre-Press. La collana si propone di incrementare la visibilità dell'ateneo, del dipartimento e delle linee di ricerca scientifica del dipartimento presso la comunità scientifica internazionale e presso il pubblico non specialistico, con contenuti sia originali che di rassegna di alto valore scientifico e scritti di carattere divulgativo. Inoltre, con una sezione riservata alla didattica, propone la pubblicazione di manuali didattici di corsi tenuti nel corso di Laurea in Fisica.

Direttore
Biagio Di Micco

n. 2
Federico Ruggieri
Acquisizione dati e controllo di esperimenti

Revisione
Prof. Biagio Di Micco (Università degli Studi di Roma Tre e Istituto Nazionale di Fisica Nucleare)
Dott. Paolo Branchini (Istituto Nazionale di Fisica Nucleare)

Coordinamento editoriale
Gruppo di lavoro *Roma TrE-Press*

Cura editoriale e impaginazione
teseo editore Roma teseoeditore.it

Elaborazione grafica della copertina
MOSQUITO mosquitoroma.it

Caratteri grafici utilizzati: Avenir Next, Kabel Book (copertina e frontespizio); Garamond, Times New Roman (testo).

Edizioni *Roma TrE-Press* ©
Roma, ottobre 2024
ISBN 979-12-5977-342-5
http://romatrepress.uniroma3.it

FORTE FONDAZIONE ROMA TRE-EDUCATION
L'attività della *Roma TrE-Press* è svolta nell'ambito della
Fondazione Roma Tre-Education, piazza della Repubblica 10, 00185 Roma.

Indice

Indice delle figure

Indice delle tabelle

Introduzione

Acquisizione dati o "Data Acquisition" (DAQ nel seguito) è l'insieme degli strumenti che permettono di acquisire delle informazioni in forma digitale da dei sensori o apparati complessi composti da molti sensori. La selezione dei dati utili e la contestuale eliminazione di quelli non interessanti è generalmente effettuata da un sistema chiamato Trigger. DAQ e Trigger non sono attività semplici da realizzare e non è facile ricavare dei modelli descrittivi generali validi per ogni contesto sperimentale, tuttavia molti aspetti sono ben conosciuti e fanno riferimento a conoscenze largamente applicate anche in altri campi.
In aggiunta l'esperienza e il "buon-senso fisico" dei ricercatori sperimentali che si sono occupati della materia è di fondamentale importanza per la comprensione di questo sistema complesso.
Il DAQ è una materia che coinvolge sia hardware (HW) che software (SW). Il primo (HW) è stato a lungo il principale protagonista dei sistemi DAQ, spesso con soluzioni costruite ad hoc per rispondere alle esigenze specifiche di un esperimento.
Il secondo (SW) ha avuto una crescita notevole negli ultimi 20-30 anni, anche grazie all'avanzamento della tecnologia dei microprocessori che, con potenze di calcolo sempre crescenti, ha di fatto permesso di effettuare con il SW molte attività che prima erano esclusivamente appannaggio dell'HW.
Un altro elemento di difficoltà nell'approccio al DAQ è che non ci sono testi sistematici ed esaustivi sull'argomento. I testi di acquisizione dati standard fanno riferimento a sistemi di controllo o di acquisizione in ambito industriale e/o di basse quantità di dati per i quali sono sufficienti un PC con software tipo LabView. In questo corso non trattiamo questi sistemi anche se sono largamente utilizzati.
I testi di Fisica Nucleare delle Alte Energie o High Energy Physics (HEP nel seguito) riservano generalmente uno o due capitoli al Trigger ed al DAQ (es. R. Leo [R01]). Il DAQ in HEP è stato, nel passato, trattato in scuole dedicate (es. CERN School of Computing) da cui è stato tratto parte del materiale che costituisce questo corso [R03] - [R04] - [R05] - [R06].
Queste lezioni cercano, perciò, di fornire una trattazione didattica semplice ma ampia (DAQ e Trigger sono un campo vasto) con qualche approfondimento (laddove necessario o richiesto).
Alcuni argomenti come, ad esempio, la teoria delle code non fanno parte del bagaglio di conoscenze standard di un laureando in Fisica e richiederebbero una trattazione a sé, essi verranno qui soltanto accennati per non appesantire il filo logico del corso che vuole essere eminentemente pratico.
Nel testo sono presenti, per completezza, alcuni richiami a conoscenze e concetti già affrontati durante i corsi del primo triennio del corso di laurea in Fisica.
Gli Esempi sono il più possibile "generali" e sono presentati solo un numero limitato di sistemi di DAQ realmente utilizzati da esperimenti laddove questo sia necessario per evidenziare soluzioni e/o applicazioni differenti.
Il testo è articolato in capitoli che sono organizzati secondo una logica sequenziale, per cui i capitoli precedenti sono necessari per comprendere i concetti illustrati in quelli successivi.

Capitolo 1

Parallelismo e Pipelining

Al fine di comprendere meglio i concetti che sottendono alla realizzazione di un sistema DAQ, è necessario avere chiari i concetti di Parallelismo e Pipelining che ricorrono in molti sistemi, tecnologici o meno, dai computer alle linee di produzione industriali.

1.1 *Parallelismo*

Il termine Parallelismo indica una tecnica volta a svolgere in parallelo delle funzioni o attività che normalmente sarebbero svolte in serie. L'uso delle tecniche di parallelismo permette di velocizzare i processi complessivi che portano al completamento delle funzioni e/o attività richieste.
Nella figura che segue è mostrato un esempio di parallelismo nel quale una serie di dati (D1, D2, D3, …, Dm) viene elaborata in parallelo da un certo numero m di operatori (Operator1, …., Operator m) fornendo una serie di m risultati (R1, R2, R3, …, Rm).
Se si avesse a disposizione un solo operatore che processasse un singolo dato alla volta in un tempo $\mathbf{t}$, allora per $\mathbf{n}$ dati si dovrebbe aspettare un tempo $\mathbf{nt}$ per ottenere il completamento delle operazioni su tutti i dati.
Utilizzando un numero $\mathbf{m}$ di operatori ($\mathbf{m<n}$) che lavorano in parallelo sui dati possiamo distribuire il carico delle operazioni ed ottenere un tempo di completamento delle operazioni pari a $\mathbf{nt/m}$ quindi ridotto di un fattore $\mathbf{m}$ rispetto al tempo del processamento seriale.

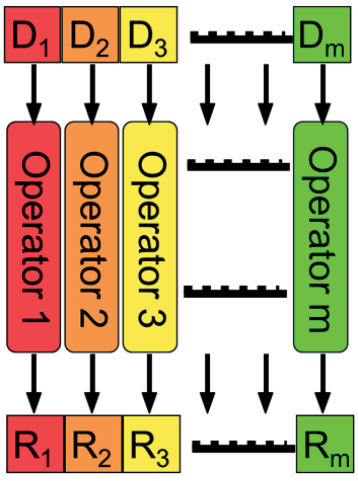


Figura 1.1.1 - Esempio di parallelismo

Le ipotesi che sono fatte implicitamente in questo esempio sono essenzialmente due:
I dati in ingresso sono tutti indipendenti fra loro ed una operazione su un dato non dipende dal risultato di operazioni su altri dati;
Le operazioni effettuate sono tutte uguali e durano lo stesso tempo.
Naturalmente queste ipotesi restrittive possono non verificarsi in casi reali.

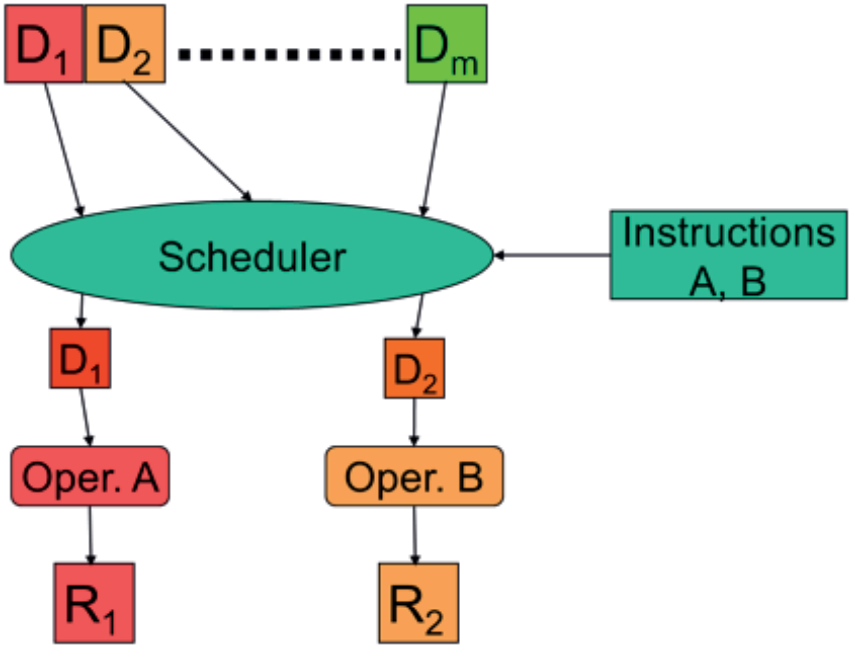


Figura 1.1.2 - Dati diversi su operatori diversi

Nella figura precedente esaminiamo il caso di dati che necessitano di essere processati da operatori diversi. In questo caso è necessario un processo intermedio di schedulazione che, in base a delle istruzioni, possa dirigere ogni dato verso l'operatore destinato a processarlo. In questo caso, nella valutazione del tempo di esecuzione totale delle operazioni su tutti i dati, vanno inclusi i tempi di decisione dello schedulatore ed i tempi differenti di esecuzione dei vari operatori. Questi ultimi, pur riducendo il tempo di esecuzione, rendono più complesso valutare il miglioramento delle prestazioni.
Gli operatori possono anche essere non "elementari" ma composti a loro volta da sotto-funzioni che messe insieme costituiscono un'unità funzionale più complessa che opera sui dati.
In Figura 1.1.3 è rappresentato un esempio semplice di unità funzionale che può essere visto come operatore composto da due funzioni: una moltiplicazione del dato **D** per una costante e un'addizione del risultato con una costante. Il risultato finale **2D+10** sarà dunque il risultato dell'operatore.

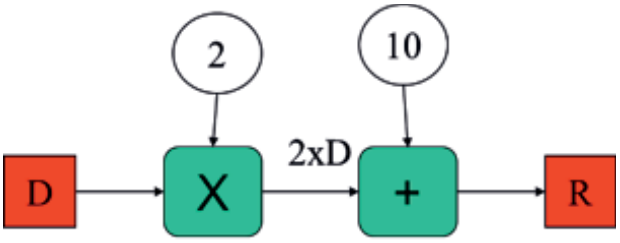


Figura 1.1.3 - Esempio di unità funzionale

Mettendo in parallelo più operatori/unità funzionali è possibile velocizzare le operazioni sui dati. Nell'esempio di Figura 1.1.4 si usano in parallelo **N** unità funzionali e quindi il fattore di velocizzazione, in termini di riduzione del tempo totale di processamento dei dati è pari a **1/N**.

In conclusione, un processamento parallelo banale, come quello esemplificato, richiede:
– buona indipendenza dei dati fra loro;
– azioni ripetitive su uno stesso tipo di dati con possibilità di replica molto elevata.

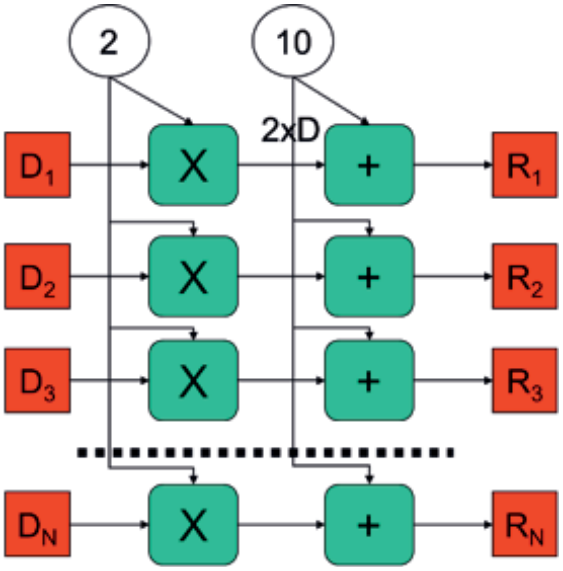


Figura 1.1.4 - Unità funzionali in parallelo

Possono essere consentite più operazioni diverse su diversi tipi di dati con un numero limitato di azioni diverse possibili: tale tipologia è più comunemente definita architettura Super-Scalare. Sono comunque possibili combinazioni di entrambi.

1.1.1 Legge di Amdahl (Gene Amdahl, 1967)

Come già discusso in precedenza, non tutto è parallelizzabile e quindi i meccanismi di parallelizzazione non sono in grado di ridurre i tempi di esecuzione a piacimento semplicemente aumentando il numero delle operazioni svolte in parallelo. Enunciamo quindi la cosiddetta Legge di Amdahl che prende il nome da Gene Amdahl ingegnere informatico e costruttore di computer prima per IBM e poi in proprio con la ditta omonima.
Se **S** è la frazione di calcolo che è seriale e **1-S** quella che può essere parallelizzata, il massimo di incremento di velocità ottenibile con un numero P di processori è:

$$1/(S+(1-S)/P)$$

che per $P \rightarrow \infty$ vale $1/S$.

In sostanza per quanto si possa parallelizzare il sistema, esiste sempre una parte finita di operazioni seriali che condizionano il tempo di esecuzione che non può essere ridotto a zero.
La legge è "ottimistica", infatti nella realtà una parallelizzazione avventurosa può portare a un aumento dei tempi di esecuzione se, per esempio, non si tiene conto dei tempi necessari per trasportare i dati da un processore all'altro (inter-processor communication).

1.2 *Pipelining*

Una pipeline è sostanzialmente la trasposizione del concetto di una "catena di montaggio". Invece di accelerare un intero processo, lo si segmenta in **N** parti e si fa in modo che si possano eseguire gli **N** sotto-processi in parallelo in maniera che il risultato del sotto-processo **n-1** sia presentato all'ingresso del sotto processo **n**, ecc.

Nel caso semplice in cui, in ogni stadio, il sotto-processo impiega un tempo **t**

$$\mathbf{t} = t_1 = t_2 = \ldots = t_N$$

il tempo di attraversamento del sistema è **N t** ed il tempo totale per processare **M** elementi è:

$$\mathbf{T=N\ t\ +(M-1)\ t}$$

Per **M>>N>1** si ottiene: **T ~ Mt**

Nel caso di singolo processo di durata **T=Nt** il tempo totale per processare (non in pipelining) **M** elementi sarebbe:

$$\mathbf{MT=MNt}$$

cioè circa un fattore **N** maggiore.

Il caso più semplice è quindi mostrato nella Figura 1.1.5 accanto, in cui le operazioni da svolgere nello stadio unico sono fattorizzabili in più stadi di durata uguale. In questo caso: **M>>n > 1** sono i dati da processare.**T** (tempo di esecuzione dello stadio unico) $\mathbf{t_1 = t_2 = \ldots = t_n = T/n}$ sono i tempi di esecuzione dei singoli **n** stadi.Il tempo di esecuzione della pipeline è dunque: $\mathbf{T_{pipeline} = T + (M-1)\ T/n}$ Infatti, il primo dato processato uscirà dalla pipeline dopo un tempo **T** (come nel caso dello stadio unico) mentre i successivi **M-1** dopo aver riempito la pipeline seguendo il primo dato, usciranno ogni **T/n**, dunque una pipeline di questo tipo ha un tempo di processamento totale:$\mathbf{T_{pipeline} \sim T_{stadio\ unico}/n}$.

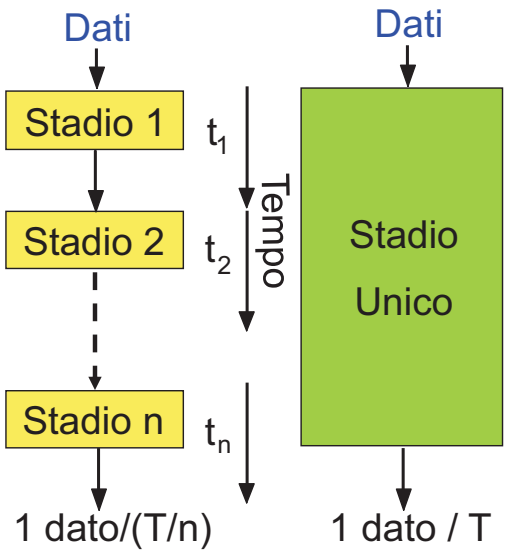


Figura 1.1.5 - Pipeline VS Stadio Unico

Nel caso in cui i tempi di esecuzione degli stadi della pipeline non siano identici, come mostrato in Figura 1.1.6, avremo che:

$$t_1 \neq t_2 \neq \ldots \neq t_n$$

e allora è evidente che lo stadio più lento:

$$t_l = \max(t_1, t_2, .., t_n)$$

sarà quello condizionante per la catena e quindi uscirà un dato processato ogni:

$$t_l > T/n.$$

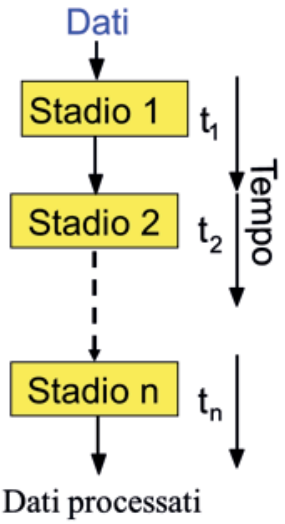


Figura 1.1.6 - Pipeline con stadi a differenti tempi

Possiamo quindi esaminare il caso in cui ciascuno stadio non ha un tempo fisso di esecuzione. In questo caso possiamo definire i tempi di esecuzione come tempi medi:

$$\langle t_1 \rangle \approx \langle t_2 \rangle \approx \ldots \approx \langle t_n \rangle$$

conseguentemente non possiamo ottenere un valore deterministico di uscita di ciascun dato ma, in linea teorica, sarebbe ancora possibile ottenere un dato ogni $\langle t \rangle = T/n$.

In realtà il problema che ci si presenta è complicato dal fatto che lo stadio **i-esimo** potrebbe finire prima dello **i+1** o viceversa. Diventa quindi necessario capire come riuscire ad evitare che questa variabilità dei tempi di esecuzione provochi un rallentamento della pipeline nel suo complesso rendendola inefficiente.

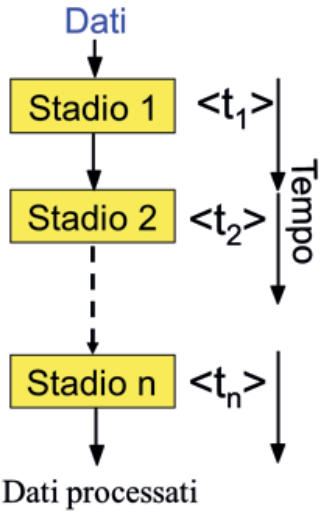


Figura 1.1.7 - Pipeline con stadi a tempo di esecuzione variabile

Quest'ultima situazione è mostrata in Figura 1.1.8 qui sotto in cui l'asse orizzontale è quello del tempo e sono poi rappresentati, per semplicità, solo due stadi di pipeline su due righe con dei rettangoli la cui lunghezza è proporzionale alla durata del processamento (tempo fra ingresso e uscita). I numeri nei cerchietti (**1, 2, 3**) sono i dati in ingresso che entrano nel primo stadio di pipeline (**A**) e poi, al termine del processamento, sono trasferiti al secondo stadio (**B**). Al termine del secondo stadio i dati processati vanno in uscita (**Output**).

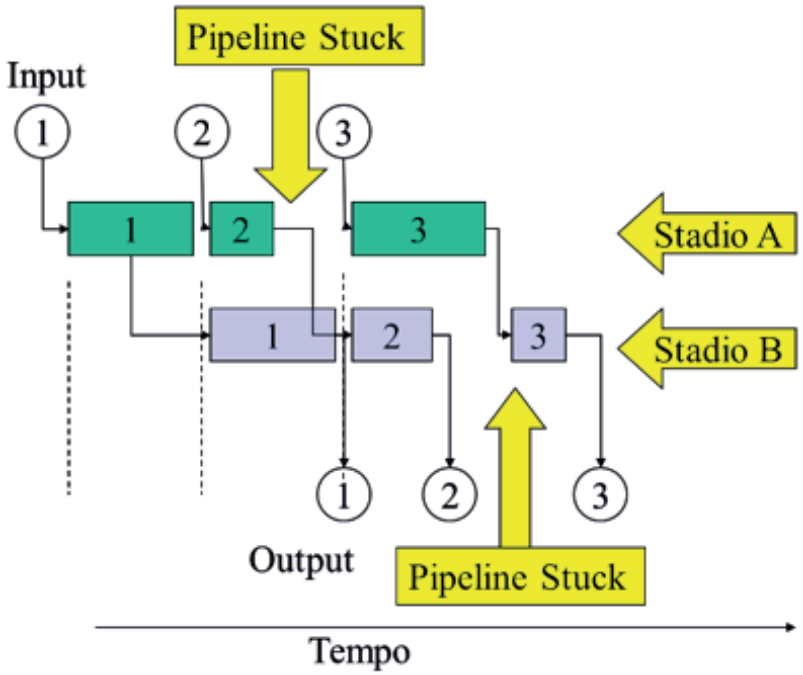


Figura 1.1.8 - Pipeline con blocco (Pipeline Stuck)

Come si può vedere, le differenti durate del primo stadio per i dati **1** e **3** provoca un blocco temporaneo della pipeline (Pipeline Stuck) perché, nel caso del dato **2**, già processato dallo stadio **A**, non è possibile trasferirlo al secondo stadio (**B**) che è ancora occupato a processare il dato precedente (**1**). Successivamente il secondo stadio (**B**) resta inattivo dopo il processamento del dato **2** perché il risultato del primo stadio sul dato **3** non è ancora disponibile.
Per mitigare questi problemi, dovuti alla variabilità e impredicibilità della durata del processamento di ciascun stadio, si possono utilizzare dei registri intermedi che vengono usati per "parcheggiare" i dati in uscita in attesa che siano disponibili gli stadi successivi.

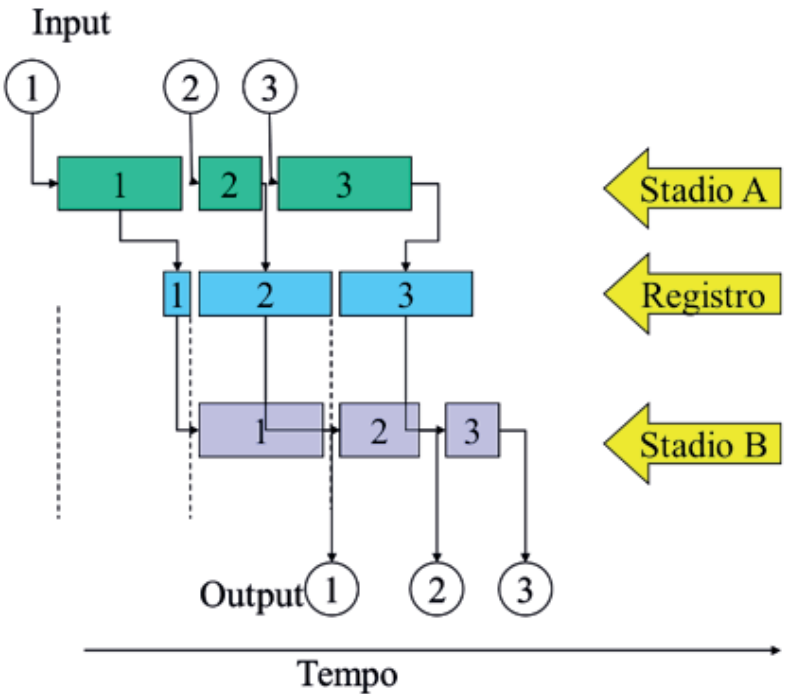


Figura 1.1.9 - Pipeline a 2 stadi con registro intermedio

Nella Figura 1.1.9 è rappresentato un esempio di tale soluzione utilizzando sempre la configurazione semplificata a due stadi ma, questa volta, interfacciati attraverso un **registro** che può ospitare il dato in uscita dal primo stadio (**A**), in attesa che il secondo stadio (**B**) sia libero.

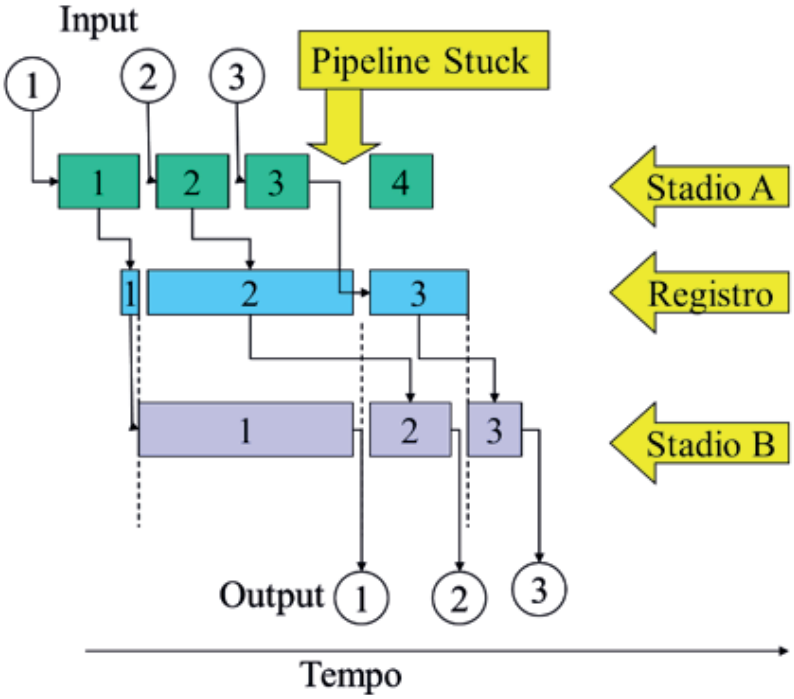


Figura 1.1.10 - Pipeline Stuck anche con registro intermedio

In questa configurazione sembra dunque possibile evitare il blocco della pipeline e far funzionare in maniera efficiente i vari stadi. Esistono, però, ancora dei casi (Figura 1.1.10) in cui un registro, con una sola posizione a disposizione, non riesce ad evitare dei blocchi della pipeline quando le differenze temporali fra i processamenti sono molto ampie.

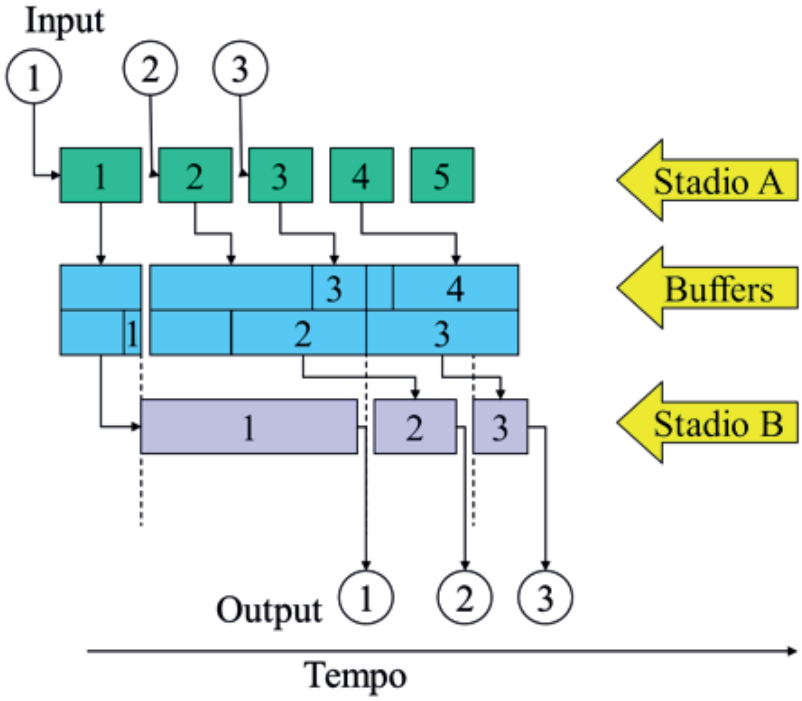


Figura 1.1.11 - Pipeline con stadi interfacciati con un buffer a due livelli

Si può allora pensare di ottenere un miglioramento sostituendo il registro singolo con un buffer di memoria a più posizioni che permetta quindi di tenere in attesa 2 o più dati. Nella Figura 1.1.11 è mostrata una configurazione con un buffer a due posizioni che permette quindi di ospitare due dati in attesa, migliorando le prestazioni di efficienza in questo caso.
Il ragionamento sin qui seguito si può estendere e possiamo pensare che esistano casi in cui anche un buffer a due posizioni non sia sufficiente ad impedire un blocco della pipeline e prevedere di aumentare la profondità del buffer per mitigare anche questi altri casi. Ovviamente esiste un compromesso ragionevole che permetta di avere una ottima efficienza della pipeline senza aumentare a dismisura la profondità del buffer.
La valutazione di tale compromesso, quando il numero di stadi è più numeroso di due, può essere fatta attraverso la simulazione del sistema.

Capitolo 2

DAQ e Trigger

In questo capitolo affronteremo la relazione fra il sistema DAQ ed il Trigger, due sistemi che collaborano nel compito di selezionare gli eventi di interesse e di trasferire i dati di tali eventi verso un sistema di archiviazione permanente o Storage System.

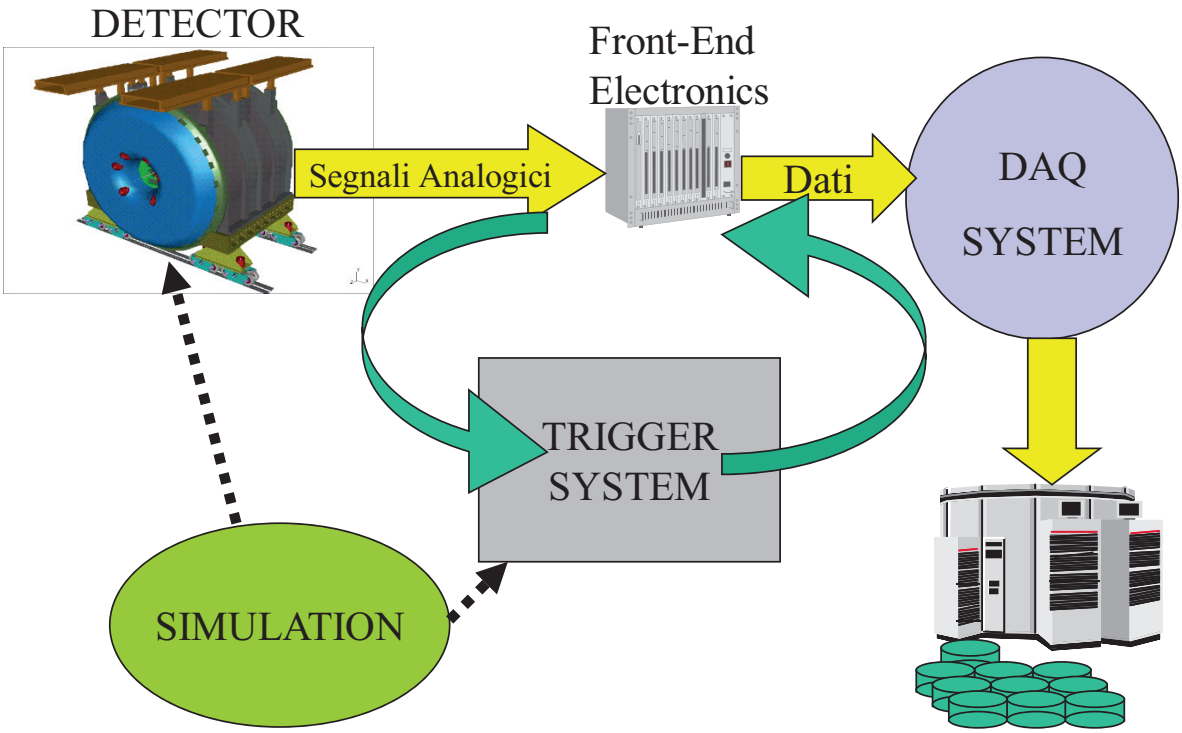


Figura 2.1 - Schema di un sistema sperimentale con le componenti DAQ e Trigger

Nella Figura 2.1 è rappresentato uno schema generale di un esperimento con evidenziate le componenti principali:

- Detector o rivelatore che costituisce lo strumento con cui sono studiati i fenomeni fisici di interesse
- La Front-End Electronics ovverosia l'elettronica che permette di trasformare i segnali raccolti dal rivelatore in dati digitali
- Il sistema DAQ che ha la funzione di acquisire i dati digitali dall'elettronica e formattarli, eventualmente selezionarli, per poi registrarli su un supporto di archiviazione

– Il sistema di Trigger che, analizzando i segnali dell'elettronica proveniente dal rivelatore, sulla base di regole prestabilite decide se acquisire il particolare evento
– Il sistema di simulazione che permette di simulare il comportamento dell'intero sistema per verificare le ipotesi fisiche e quelle architetturali relative all'esperimento.

Importanti sono altresì le relazioni che intercorrono fra le componenti per poter controllare l'intero complesso:

– Acceleratore: fornisce informazioni sul fascio di particelle che è in uso e ottiene risposte da parte del rivelatore
– Rivelatore che invia i dati non elaborati (Raw)
– Database che contiene e registra le condizioni di lavoro del rivelatore
– Il sistema di storage dove vengono archiviati i dati
– Il controllo dell'esperimento che imposta le condizioni di lavoro e monitora lo stato del rivelatore e dei sistemi di Trigger e DAQ
– Il ricercatore che interagisce con il sistema Trigger e DAQ per impostare le condizioni di lavoro dell'esperimento e ricevere informazioni sullo stato del rivelatore e dell'acquisizione dei dati.

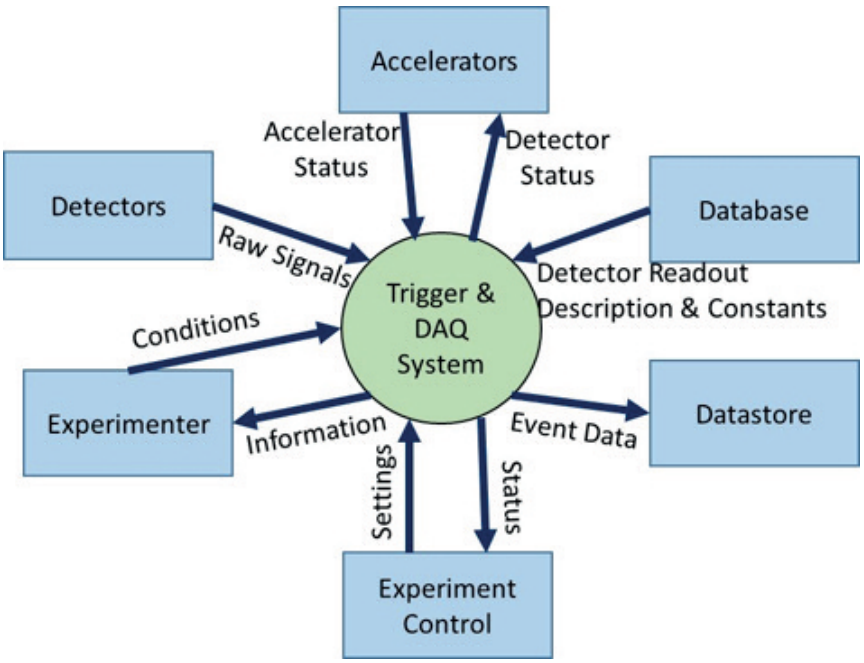


Figura 2.2 - Interazioni del sistema Trigger e DAQ - [R05]

Gli esperimenti più complessi e di grandi dimensioni realizzano un sistema che include le varie componenti già descritte in uno schema di flusso che è mostrato nella figura seguente.

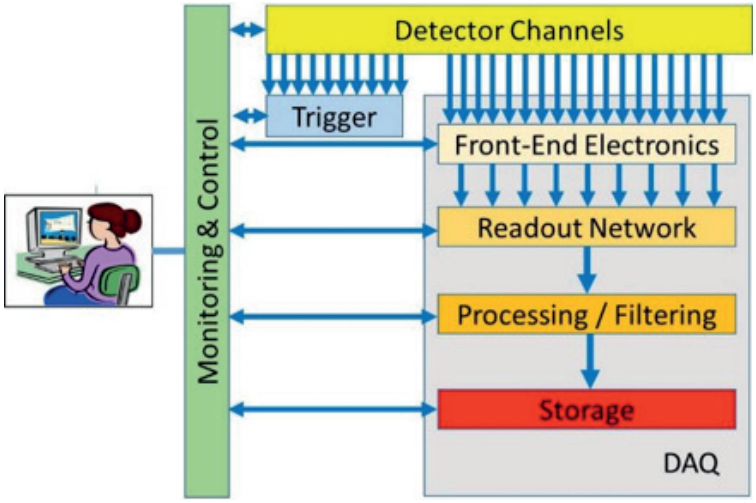


Figura 2.3 - Flusso dati e relazioni fra le componenti DAQ - [R05]

Per comprendere concretamente le funzioni previste all'interno di un sistema DAQ e le sue interazioni con il sistema di Trigger, esaminiamo un sistema semplice di acquisizione dati partendo da un sensore (es. di temperatura) che può essere letto da un Personal Computer (PC) attraverso un'interfaccia. In Figura 2.4 è mostrata la vista esterna di un tale sistema. Il sensore i cui dati devono essere acquisiti è un elemento esterno collegato al PC.

Figura 2.4 - Semplice sistema DAQ – Vista esterna

Nella figura seguente sono invece esplicitate le componenti hardware che costituiscono il sistema ed i loro collegamenti. Il sensore è collegato ad un Convertitore Analogico Digitale (ADC) che ha il compito di convertire il segnale del sensore in un valore digitale composto da un certo numero di bit. Tale dato digitale è poi letto dal Processore del PC (CPU – Central Processing Unit) e quindi registrato su un supporto magnetico di archiviazione (Disco). ADC, CPU e Disco usano un canale di trasmissione dati comune che prende il nome di Bus (si veda Cap. 7 più avanti).
Le tempistiche di acquisizione e trasferimento dati sono scandite da un Clock (orologio) programmabile che fa parte del sistema PC.

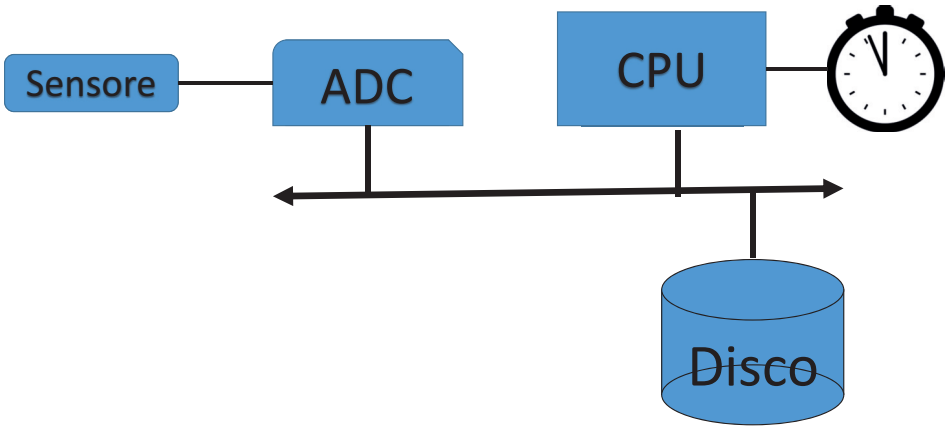


Figura 2.5 - Semplice sistema DAQ - Vista Hardware

Nella Figura 2.6 è invece mostrata la sequenza logica di tali operazioni effettuate da ciascun componente descritto precedentemente.

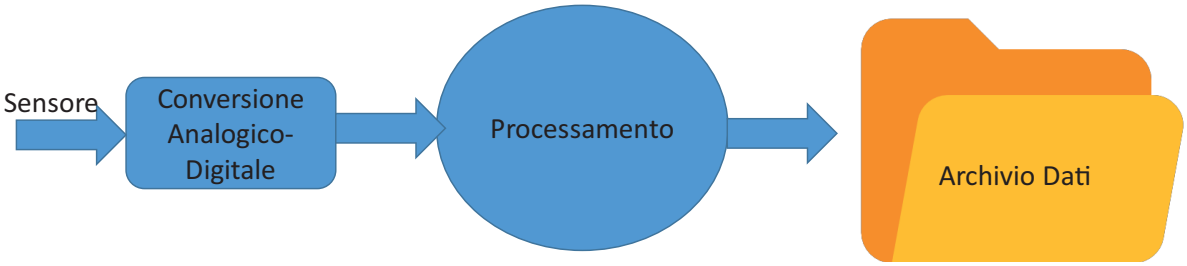


Figura 2.6 - Semplice sistema DAQ - Vista Logica

Naturalmente un tale sistema è estremamente semplificato e richiede che la CPU vada in continuazione ad abilitare l'ADC e ad acquisire i dati da esso prodotti anche in mancanza di valori significativi, esso può essere migliorato con l'integrazione di un Trigger che dia al sistema ADC un segnale (Start) che indichi quando è presente un valore del sensore al di sopra di una certa soglia (Discriminatore) e quindi sia possibile acquisire un dato significativo. Lo stesso Trigger, attraverso un segnale (Interrupt) va anche ad informare la CPU che c'è un dato da leggere nell'ADC. Tale configurazione è mostrata nella figura seguente.

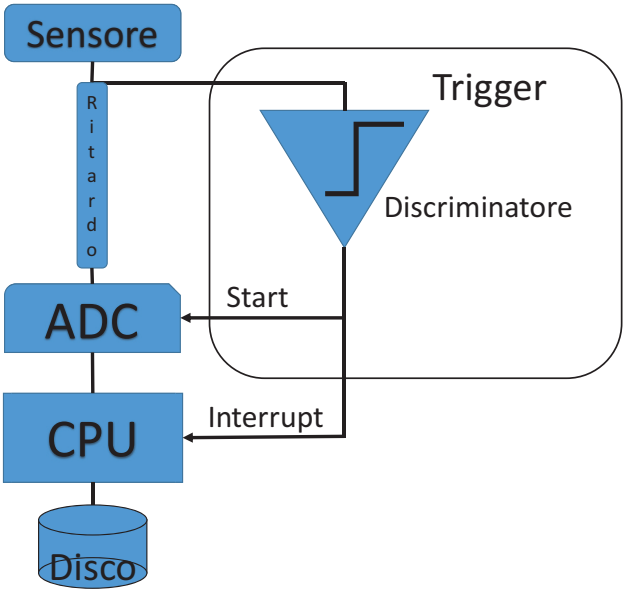


Figura 2.7 - Semplice sistema DAQ con Trigger

Ciascuno degli elementi ovviamente impiegherà un certo tempo a svolgere la sua funzione ed in particolare il ritardo introdotto fra Sensore ed ADC ha proprio il compito di dare al Trigger il tempo di eseguire l'esame del segnale del Sensore per decidere se sia sopra soglia o meno.
A questo punto occorre considerare che anche l'ADC impiegherà un certo tempo per la conversione e che un nuovo segnale di Trigger potrebbe arrivare all'ADC mentre sta già convertendo oppure mentre la CPU non ha ancora finito il processamento.

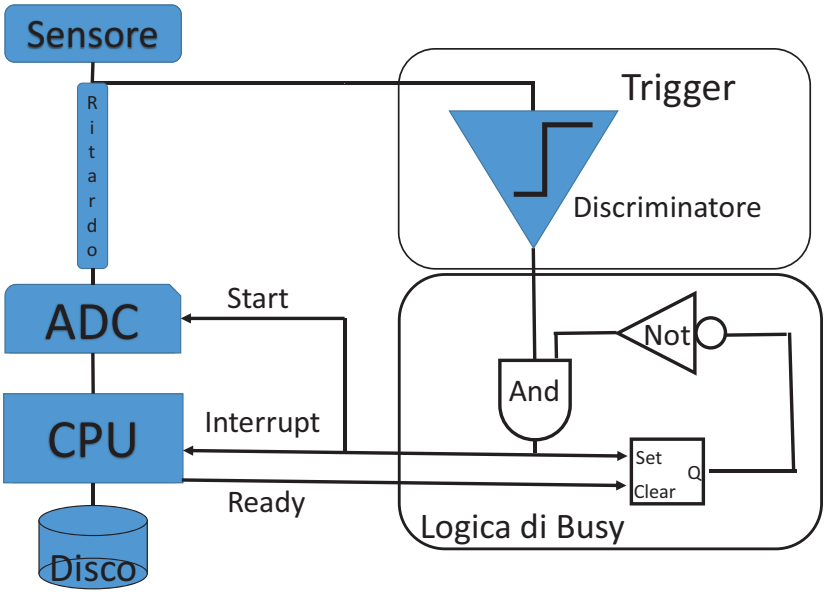


Figura 2.8 - Semplice sistema DAQ con Trigger e Busy

Per evitare che questo accada è necessario inibire la generazione di un nuovo Trigger finché l'ADC non ha esaurito il suo compito. In Figura 2.8 è mostrato il sistema DAQ con l'aggiunta di una Logica di Busy che attraverso l'uso di un AND, un NOT ed un FlipFlop realizza un circuito di Busy. L'uscita del Trigger è inviata ad un AND che trasmette il segnale di Interrupt alla CPU e Start all'ADC e fa anche un Set del FlipFlop la cui uscita Q viene invertita dal NOT e, di conseguenza impedisce alla porta AND di far passare un nuovo segnale di Trigger. Alla fine del processamento, il segnale di Ready della CPU va a produrre un Clear del FlipFlop riazzerando così l'uscita Q e, quindi l'uscita del NOT diventa nuovamente alta e riabilita la porta AND.
Questo sistema, pur eliminando il problema della sovrapposizione dei Trigger, produce una riduzione del tempo in cui il sistema è in grado di accettare un Trigger. Nel prossimo capitolo esamineremo il concetto di Tempo Morto (TM), ma nel frattempo cerchiamo di sfruttare un meccanismo di Pipeline per ridurre il TM modificando il sistema di DAQ come illustrato nella figura seguente.

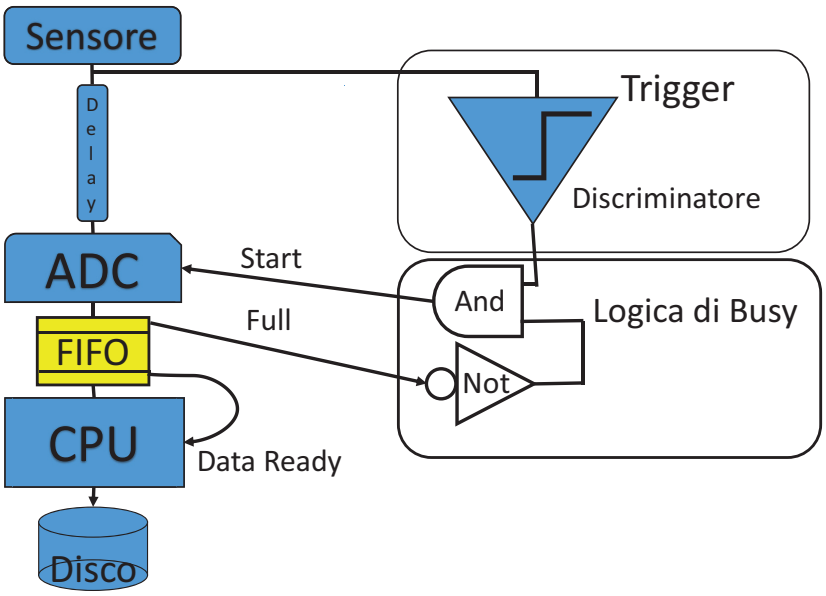


Figura 2.9 - Semplice sistema DAQ con Trigger, Busy e FiFo

Nella Figura 2.9 è dunque mostrato un sistema in cui si è interposta fra l'ADC e la CPU una FiFo (First In First Out) e cioè una tipo di memoria che viene scritta dal lato dell'ingresso e letta dall'uscita dove i dati sono presentanti nella stessa sequenza con cui sono stati introdotti. In tale configurazione il risultato dell'ADC viene inviato alla FiFo appena l'ADC ha finito e senza aspettare che il processamento del dato precedente sia terminato. Se la FiFo si riempie (Full) invia un segnale che inibisce il Trigger attraverso la logica di Busy, così come abbiamo visto precedentemente.
La FiFo informa la CPU sulla presenza di dati da processare attraverso un segnale di Data Ready.

Capitolo 3

Tempo morto e Derandomizzazione

Il tempo morto (TM) è un intervallo di tempo in cui il rivelatore o il sistema di acquisizione non sono in grado di acquisire nuove informazioni.
Il tempo morto percentuale è il rapporto del tempo morto sul tempo totale.
Nel caso di un sistema di acquisizione il TM è in relazione con il numero di eventi che si acquisiscono e condiziona il parametro di efficienza definito come:

N. Eventi Acquisiti / N. Eventi Totali

Per comprendere come il TM sia in relazione con l'efficienza ed il numero di eventi acquisiti è necessario ricordare che i fenomeni di rivelazione di eventi in Fisica delle Particelle sono rari e la probabilità di osservare il fenomeno è piccola. Aumentando però il numero di campionamenti o di osservazioni, si può fare in modo che, comunque, il numero di eventi osservati nell'unità di tempo sia ragionevole.
La distribuzione di probabilità che rappresenta queste condizioni è quella di Poisson:

$$P(r) = m^r e^{-m} / r!$$

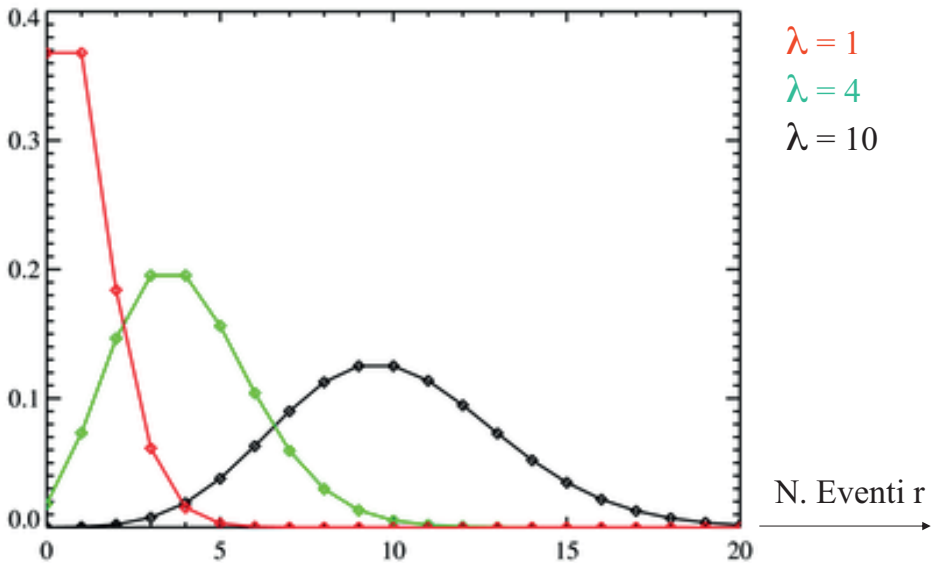


Figura 3.1 - Distribuzioni di Probabilità di Poisson per varie medie

Essa rappresenta la probabilità di osservare **r** eventi nel limite di $\mathbf{p \to 0}$ e $\mathbf{N \to \infty}$ mantenendo la media $\mathbf{m = N\,p}$ finita.
Ai fini pratici viene utilizzata la media per unità di tempo $\boldsymbol{\lambda}$ che quindi porta ad una media: $\boldsymbol{\mu = \lambda\, t}$
La probabilità in funzione di $\boldsymbol{\lambda}$ è quindi:

$$\mathbf{P(r) = \lambda^{r}\, e^{-\lambda} / r!}$$

Nella figura precedente sono mostrate le distribuzioni di Poisson nei casi in cui la media per unità di tempo $\boldsymbol{\lambda}$ è, rispettivamente, uguale a 1, 4, 10. In ascissa è riportato il numero di eventi ed in ordinata la probabilità di osservare un tale numero di eventi nell'unità di tempo. Il caso di $\boldsymbol{\lambda} = 1$ mostra una probabilità quasi uguale di rivelare zero eventi che è ovvia sia per motivi probabilistici, sia perché l'area fra zero e uno, non essendo possibili valori negativi del numero di eventi rivelati, deve bilanciare la parte di area sottesa dalla curva a destra del valore 1 perché la media risulti effettivamente uguale ad un evento per unità di tempo. Per $\boldsymbol{\lambda}$ più alti la distribuzione diventa sempre più simile ad una Gaussiana.
Il Tempo Morto del sistema che acquisisce gli eventi va quindi confrontato con la frequenza di eventi prevista dalla statistica. In aggiunta a ciò il TM stesso del sistema può variare in relazione alla tipologia dell'evento da acquisire e processare.
Ai fini pratici quello che normalmente viene utilizzato è il TM percentuale espresso come: TM totale / Tempo totale di Acquisizione.

Ci limitiamo qui a fare un esempio con delle semplificazioni che non coinvolgano la distribuzione di Poisson e consideriamo la seguente situazione sperimentale:

– Arrivo di un raggio cosmico su un rivelatore con frequenza media $\mathbf{f=10^{3}\ Hz}$
– Il tempo medio di acquisizione $\mathbf{T=10^{-4}\ s}$ impedisce l'acquisizione di un nuovo evento (per esempio disabilitando il trigger per il tempo **T**).

In tali condizioni durante l'osservazione del fenomeno, dopo l'arrivo di ogni evento (Trigger) si dovrà aspettare un tempo **T** prima di acquisire un evento successivo. Dunque consideriamo un tempo morto $\mathbf{TM = 10^{-4}s/Evento}$ ed il totale degli eventi da acquisire nell'unità di tempo $\mathbf{f = 10^{3}\ Eventi/s}$. Il tempo morto percentuale è quindi pari a:

$$\mathbf{T\,f = 10^{-1} = 10\%}$$

Nel caso semplice proposto, al variare del TM e del suo valore percentuale, varierà anche il numero degli eventi persi. Al raggiungimento del 100% di TM si perderanno il 50% degli eventi, così come mostrato nel grafico che segue.

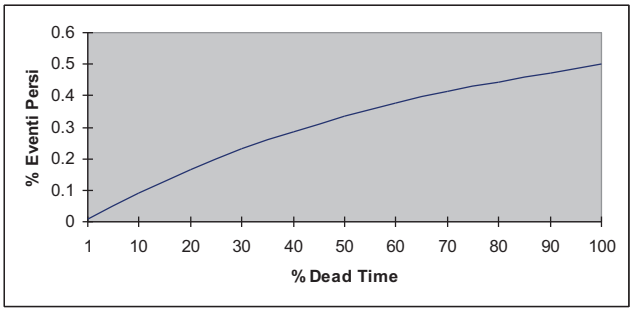


Figura 3.2 - Percentuale di Eventi persi in funzione del Tempo Morto Percentuale

In generale il tempo di arrivo degli eventi è distribuito in maniera randomica e segue le statistiche illustrate mentre il TM del sistema di acquisizione può essere più deterministico e regolare e, pertanto, spesso si cerca di mettere in un buffer gli eventi per permettere di rilasciare il busy e "derandomizzare" la frequenza di arrivo degli eventi.
Il TM non è presente solo nel sistema di DAQ, ma anche nel Trigger e nell'elettronica di Front-End.
Nel Par. 6.2 viene discussa l'efficienza di trigger e di acquisizione prendendo in considerazione il tempo morto.

Capitolo 4

Trasmissione Dati, BUS e Serializzazione

Un aspetto importante del DAQ è quello della trasmissione dei dati che sono acquisiti dall'elettronica. La comunicazione fra le varie componenti del sistema di DAQ richiede un uso di protocolli elettrici e logici che permettano la trasmissione delle informazioni in maniera digitale. In questo capitolo esaminiamo in maniera succinta e non esaustiva alcuni concetti relativi alla trasmissione dei dati ed alle tipologie di protocolli e standard utilizzati.

4.1 *Tipi di trasmissione*

La trasmissione dei dati richiede che ci sia un Trasmettitore ed un Ricevitore, il primo invia i dati al secondo attraverso un mezzo di comunicazione che può essere di vario tipo: un cavo elettrico od ottico, oppure un canale radio.
La comunicazione fra due elementi del sistema di comunicazione può avere alcune caratteristiche diverse a seconda che la trasmissione avvenga in una sola direzione o meno.

Figura 4.1 - Comunicazione unidirezionale o Simplex

Nella figura precedente è schematizzata la comunicazione di tipo unidirezionale o anche detta Simplex in cui un Trasmettitore invia segnali più o meno complessi ad un Ricevitore che si limita a ricevere tali segnali senza rispondere.
Nella figura seguente, invece, esaminiamo il caso in cui lo scambio di informazioni è nei due versi ed i due sistemi comunicanti svolgono entrambi funzione di Trasmettitore e Ricevitore. Questo tipo di comunicazione bidirezionale va sotto il nome di Duplex e richiede che ci siano due canali separati per la comunicazione nelle due direzioni.

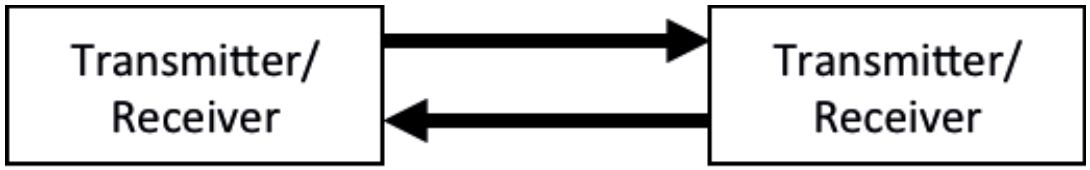


Figura 4.2 - Comunicazione bidirezionale o Duplex

Il sistema può anche essere realizzato in modo che il canale bidirezionale sia unico ed usato in maniera alternata facendo prendere il ruolo di Trasmettitore ad uno dei due e di Ricevitore all'altro per poi invertire tali ruoli per la comunicazione nell'altra direzione.

Figura 4.3 - Trasmissione alternata o Half Duplex

In questo caso c'è un risparmio in termini del numero di canali di comunicazione ma al prezzo di rallentare la comunicazione per gestire un unico canale in alternanza, inoltre c'è la possibilità di sdoppiare completamente la parte trasmettitrice e quella ricevitrice in entrambi i sistemi comunicanti e quindi facendo uso simultaneo dei due canali di comunicazione nelle due direzioni in maniera indipendente.Il sistema mostrato nella Figura 4.4 va sotto il nome di Full Duplex.

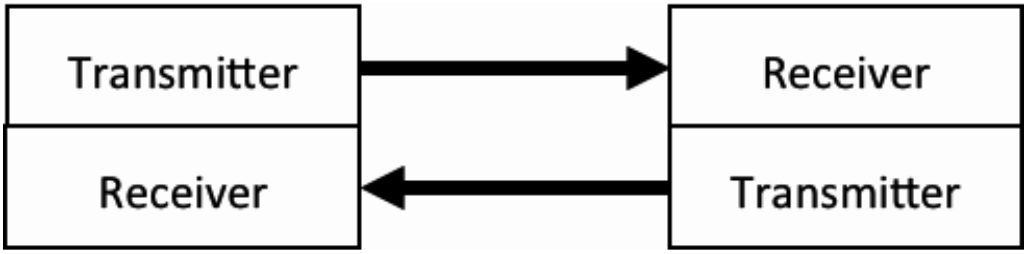


Figura 4.4 - Trasmissione completamente simultanea o Full Duplex

Naturalmente la tipologia di trasmissione è anche influenzata pesantemente dal tipo di canale che viene utilizzato per effettuarla e possiamo distinguere due principali schemi: lo schema di trasmissione parallela e quello di trasmissione seriale.

4.2 *Trasmissione Parallela*

Nell'esaminare la trasmissione Parallela, come mostrato nella figura seguente, notiamo che è necessario avere più mezzi di comunicazione in parallelo (es. più conduttori per trasmissione elettrica) per poter veicolare una diversa parte dell'informazione complessiva. Nel caso in figura si utilizzano 8 canali singoli per trasmettere 8 bit di informazione in contemporanea.

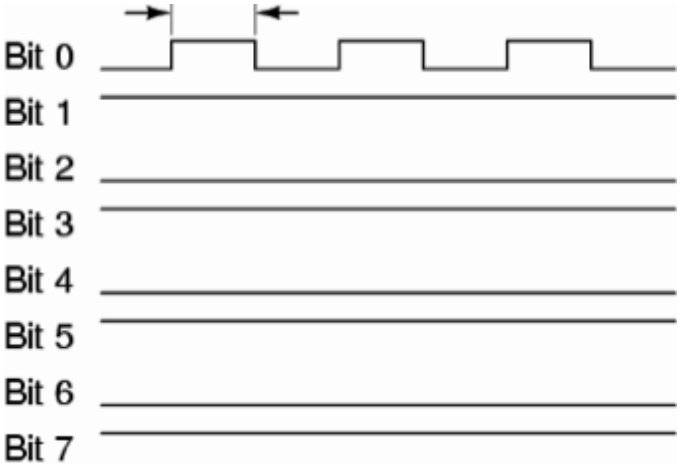


Figura 4.5 - Trasmissione Parallela su 8 bit

I vantaggi di un tale tipo di trasmissione sono ovviamente quelli legati alla velocità con cui si possono inviare i dati: più bit in parallelo si trasmettono e maggiore sarà la cosiddetta Banda Passante, ovvero il numero di Bit/s che vengono trasmessi fra sistemi collegati fra loro tramite questo tipo di trasmissione.

4.3 *Trasmissione Seriale*

La trasmissione seriale è caratterizzata dall'uso di un solo canale elementare trasmissivo che viene utilizzato per inviare in sequenza i bit che compongono l'informazione. In questo caso, però, per individuare la partenza e la fine delle parole composte da, ad esempio, 8 bit, si aggiungono un Bit di Start come indicatore dell'inizio di una nuova parola ed 1 o 2 Bit di Stop per indicarne la fine, in maniera da sincronizzare la trasmissione fra Trasmettitore e Ricevitore.

Figura 4.6 - Trasmissione Seriale di Bit

Infine a questi bit viene aggiunto almeno un Bit di Parità utilizzato per effettuare un controllo sulla presenza di eventuali errori trasmissivi.

4.4 *Topologie trasmissive*

I tipi di trasmissione descritti precedentemente costituiscono il mezzo di comunicazione fra sistemi, il quale viene genericamente indicato come rete di comunicazione o Network.
Le topologie trasmissive riguardano soprattutto il tipo di configurazione di tale Network nei vari casi in cui i sistemi che devono comunicare fra di loro siano 2 o più di 2. Distinguiamo pertanto almeno le seguenti tipologie principali: Point-to-Point, Bus, Star e Ring.

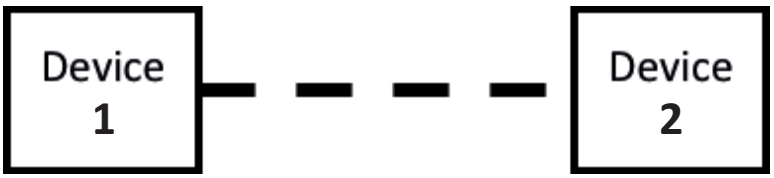


Figura 4.7 - Topologia Point-to Point

La topologia Point-to-Point è quella più semplice nel caso in cui si debbano far comunicare due dispositivi: la rete di comunicazione fra i due può essere dei tipi che abbiamo già esaminato.

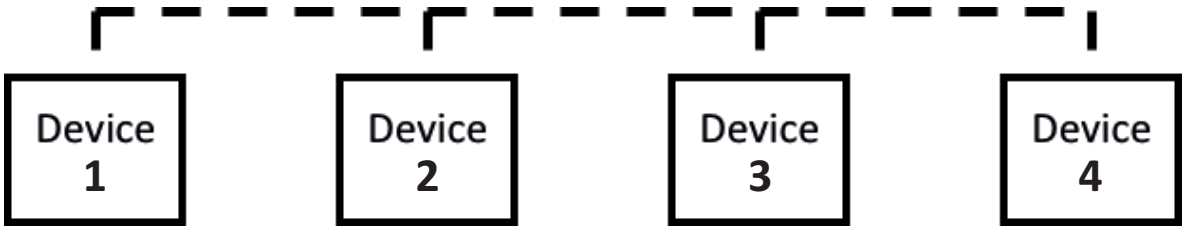


Figura 4.8 - Topologia Bus

La topologia mostrata nella figura precedente è invece orientata alla comunicazione fra vari dispositivi e la rete di comunicazione è realizzata con un mezzo di comunicazione comune (Bus) al quale si interfacciano i dispositivi per comunicare fra di loro. In questo caso sarà necessario un protocollo di accesso al mezzo che permetta ai vari dispositivi di poter comunicare in maniera ordinata e controllata senza far sovrapporre le comunicazioni fra di loro.
In alcune tipologie di reti trasmissive il Bus può essere collassato verso una struttura a stella in cui tutti i dispositivi che devono comunicare sono collegati ad un punto centrale (Hub) che ha la funzione di mettere in comunicazione tutti i dispositivi fra di loro. Nel caso più semplice la inter-

connessione è continua o stabile ed il ruolo dell'Hub è principalmente passivo. Per migliorare l'uso pratico di tale topologia a stella e per gestire meglio la comunicazione fra i dispositivi, l'Hub può essere sostituito da uno Switch che permette la comunicazione contemporanea a due a due di coppie diverse di dispositivi. Il punto debole della topologia Star è proprio l'Hub o Switch che costituisce un singolo punto di fallimento: nel caso di un guasto di tali elementi, la comunicazione è interrotta fra tutti i dispositivi collegati ad essi.

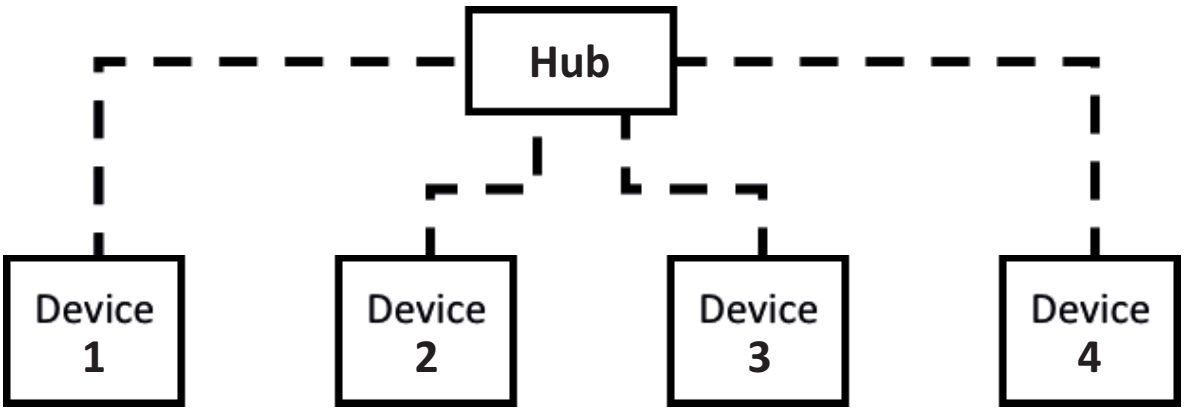


Figura 4.9 - Topologia a stella o Star

Infine esaminiamo per completezza anche la topologia ad anello o Ring che prevede che la rete trasmissiva passi attraverso tutti i dispositivi per poi richiudersi su se stessa.
Questa topologia ha il vantaggio di essere più resiliente in caso di guasto di uno dei dispositivi, garantendo una seconda strada per raggiungere gli altri dispositivi.

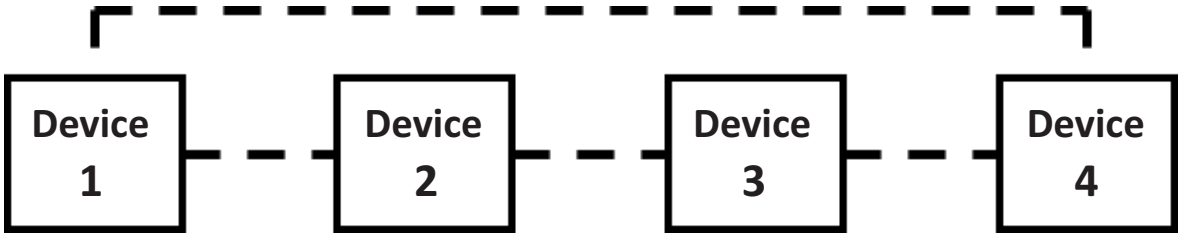


Figura 4.10 - Topologia ad anello o Ring

4.5 *Bus per brevi distanze*

Fra le topologie trasmissive sin qui esaminate, quella di tipo BUS è la più utilizzata per distanze brevi di interconnessione fra dispositivi. Distanze brevi sono quelle che vanno da pochi cm ad un massimo di pochi metri. Naturalmente il vantaggio dei Bus a breve distanza sta nella rapidità del trasferimento dell'informazione fra un dispositivo e l'altro. Tale velocità di trasmissione è sostanzialmente limitata dalla velocità della luce nel mezzo e, in generale, il tempo di attraversamento del mezzo fisico di trasporto dei segnali, si attesta su un ordine di grandezza di circa 5 ns/m.
Nell'ampio spettro di applicazioni che tali distanze e velocità trasmissive comportano, esaminiamo alcuni casi esemplificativi.
Una prima categoria è quella di Bus che possano essere utilizzati all'interno di Computer o Sistemi di calcolo e che spesso sono deputati a collegare parti importanti di esso come ad esempio: CPU, Memoria, Sistemi di Input/Output (I/O).

– PCI bus è all'origine un BUS parallelo usato in personal computers con Banda Passante tipica delle specifiche originali di 100 Mbytes/second (32 bit) e che è poi passata a 200 Mbytes/second (64 bit) con una frequenza di clock a 33 MHz; ha avuto successivi incrementi fino a 64bit x 133 MHz.

– PCI Express ha sostituito il vecchio PCI conservandone il nome ma basandosi su una trasmissione base seriale con canale a 2.5 Gbit/s; per aumentare la Banda Passante si utilizzano canali multipli in parallelo (es. 16X equivalente a 40 Gbit/s totali).
– VME Un bus parallelo ad alte prestazioni (co-disegnato da Motorola e basato sul precedente standard Motorola Versa-Bus) ideato per la costruzione versatile di computer industriali e miltari in cui si potessero comporre sistemi dotati di vari tipi di dispositivi: memorie, periferiche, microprocessori. La realizzazione avviene attraverso l'inserimento di Schede elettroniche in cabinet (rack) standard "passivo" che fornisce il bus di comunicazione e le linee per l'alimentazione. Tipica banda passante di data transfer è 50 Mbytes/second (con larghezza del bus a 64 bit).
– SCSI sta per Small Computer System Interface. È usato per collegare hard drives, ma è stato anche usato per CD-ROM, floppy disk, stampanti, scanner, e altre periferiche ad un computer. Lo SCSI è stato il primo bus a permettere un collegamento a computer di dispositivi esterni in maniera standard e non proprietaria. Il suo protocollo di comunicazione è sopravvissuto alla sua interfaccia elettrica ed è ancora utilizzato all'interno di altre soluzioni come ad esempio Fibre Channel o iSCSI.
– GPIB (IEEE 488) General Purpose Interface Bus, anche conosciuto come HPIB nella sua implementazione fatta da Hewlett Packard e che aveva lo scopo di interfacciare dispositivi elettronici di test come oscilloscopi e multimetri a dei personal computer. Il bus parallelo è a 8 bit per indirizzi e dati con 8 linee addizionali per il controllo della comunicazione.
– USB Universal Serial Bus, come dice il nome esteso, è un BUS Seriale che è stato studiato per interconnettere un numero ampio di dispositivi periferici (tastiere, mouse, modem, stampanti) ai personal computer. All'origine è un bus pensato per dispositivi con basse esigenze di trasferimento dati. Successivamente si è evoluto (USB2, USB3) con velocità di trasferimento sensibilmente più elevate e permettendo quindi il collegamento di dispositivi più esigenti in termini di Banda Passante come Monitor, Dischi e dispositivi di memorizzazione.
– FireWire (IEEE 1394) è un BUS Standard seriale ad alta velocità operante a 100, 200, o 400 Mbps con caratteristiche di versatilità come, ad esempio lo "hot swapping" (possibilità di aggiungere e togliere dispositivi senza dover spegnere l'alimentazione). Ha per molto tempo costituito una alternativa preferenziale sia allo SCSI che all'USB che aveva meno prestazioni ma un costo più basso.

A questa lista di bus possiamo aggiungere altri bus che sono stati usati per sistemi di acquisizione dati:

– Myrinet è un American National Standard: ANSI/VITA 26-1998, "Myrinet-on-VME Protocol Specification" e usa una trasmissione seriale switched a 1.28 Gbps, 2Gbps o a 10Gbps con lo stesso protocollo fisico di 10Gb Ethernet con cui è interoperabile.
– Infiniband è un tipo di trasmissione seriale switched che opera su un canale base a 2.5 Gbps (lo stesso adottato da PCI Express) o a 10 Gbps in entrambe le direzioni (Full-Duplex).
– SCI (Scalable Coherent Interface) è un ANSI/IEEE standard che definisce un protocollo di trasmissione ad alta velocità a pacchetto di tipo punto-punto. Può essere implementato come un link parallelo a 16 bit di tipo ring o switched a stella e raggiunge i 5.33 Gbps.

Altri tipi di connessioni che possiamo usare come esempi sono:

– Ethernet è uno standard (IEEE 802.3) che inizialmente a 10Mbps è poi passato a 100Mbps, 1Gbps in rame (UTP Unshielded Twisted Pair) o in Fibra Ottica (fino a 100 Gbit/s) ed è il sistema di rete largamente più usato, anche in campo DAQ specie per

Event Building e Trigger di Livello 3. Inizialmente pensato per costruire reti locali (LAN- Local Area Network), Ethernet è utilizzato come protocollo di comunicazione anche su lunghe distanze.

– Bluetooth può essere citato nel campo delle reti di comunicazione radio a brevi distanze e permette di collegare fra loro vari dispositivi (Computer, Smartphone, ecc.) per applicazioni che non richiedano un'ampia Banda Passante trasmissiva.

Più avanti nei prossimi capitoli ci soffermeremo su alcuni dei bus e sistemi di comunicazione citati ed in particolare: VME, InfiniBand ed Ethernet.

Capitolo 5

L'Elettronica di Front-End

L'elettronica di Front-End è quella che permette di registrare e rendere disponibili i dati relativi ai diversi rivelatori di cui è composto un esperimento. Essa può comprendere una varietà di tipologie di elettronica che dipendono dal tipo di esperimento e dalla tipologia di rivelatori e dal modo in cui sono utilizzati. Possiamo fare un elenco non esaustivo delle elettroniche più usate:

- Elettronica sul rivelatore: è quella che riceve i segnali dai rivelatori e li pre-amplifica, trasforma, conserva, discrimina e mixa su un numero di canali limitato in vista di una successiva acquisizione;
- Moduli in Counting Room: sono moduli elettronici che ricevono i segnali dal rivelatore e li rendono disponibili, dopo una eventuale discriminazione o selezione, al sistema di acquisizione;
- Analog to Digital Converter (ADC): sono tipici componenti quasi onnipresenti nelle elettroniche dei rivelatori e che permettono la trasformazione di segnali analogici in quantità digitali che possono essere trattate dal Sistema di acquisizione;
- Time to Digital Converter (TDC): anch'essi spesso presenti in molta parte dell'elettronica di rivelatori svolgono il compito di misurare dei tempi e trasformarli in quantità digitali acquisibili e manipolabili dai computer;
- Scale di conteggio (Scaler): sono elettroniche in grado di contare il numero degli impulsi elettrici che ricevono e poi presentare tali conteggi per il Sistema di acquisizione;
- Bit Pattern: sono generalmente dei registri in cui sono rappresentate delle parole (Word) a più bit (16, 32, 64, ecc.) in cui ogni bit rappresenta una condizione logica verificata o meno;
- Control Status Register (CSR): sono registri che permettono di controllare un dispositivo e di leggerne lo stato;
- First-In First-Out Memory (FIFO); sono un tipo di memorie a più stadi in cui le informazioni possono essere prelevate solo nello stesso ordine con cui sono state inserite;
- Buffer Memory: sono un tipo di memorie ad accesso più versatile perché permettono generalmente l'accesso in scrittura e lettura da più parti ed in maniera non necessariamente sequenziale.

L'Elettronica di Front-End, come detto, dipende dal tipo di rivelatore e di esperimento e può essere composta sia da moduli elettronici standard acquistabili sul mercato da ditte specializzate, sia può essere costituita da elettronica costruita ad hoc (custom) che viene realizzata appositamente per un certo rivelatore ed esperimento.

Il collegamento fra l'elettronica sul rivelatore e quella in Counting Room avviene attraverso una serie di cavi che trasportano l'informazione anche per molte centinaia di metri. Tali cavi sono oggi quasi esclusivamente basati su fibre ottiche per una serie di ottime motivazioni:

– Dimensioni del cavo che con fibre ottiche sottilissime è in grado di sostituire i cavi in rame che hanno bisogno di spessori più consistenti per ridurre l'impedenza su lunghe distanze e a causa dello schermaggio necessario per proteggere i cavi da interferenze elettro-magnetiche e fenomeni di cross-talk fra canali diversi;
– Velocità trasmissive elevate della fibra ottica che può trasportare, con ottiche opportune, diversi Tbps a distanze di molte centinaia di metri e oltre;
– Eliminazione di possibili Loop di Massa fra la Counting Room ed il rivelatore a causa delle connessioni di potenza elettrica e di trasmissione dati.

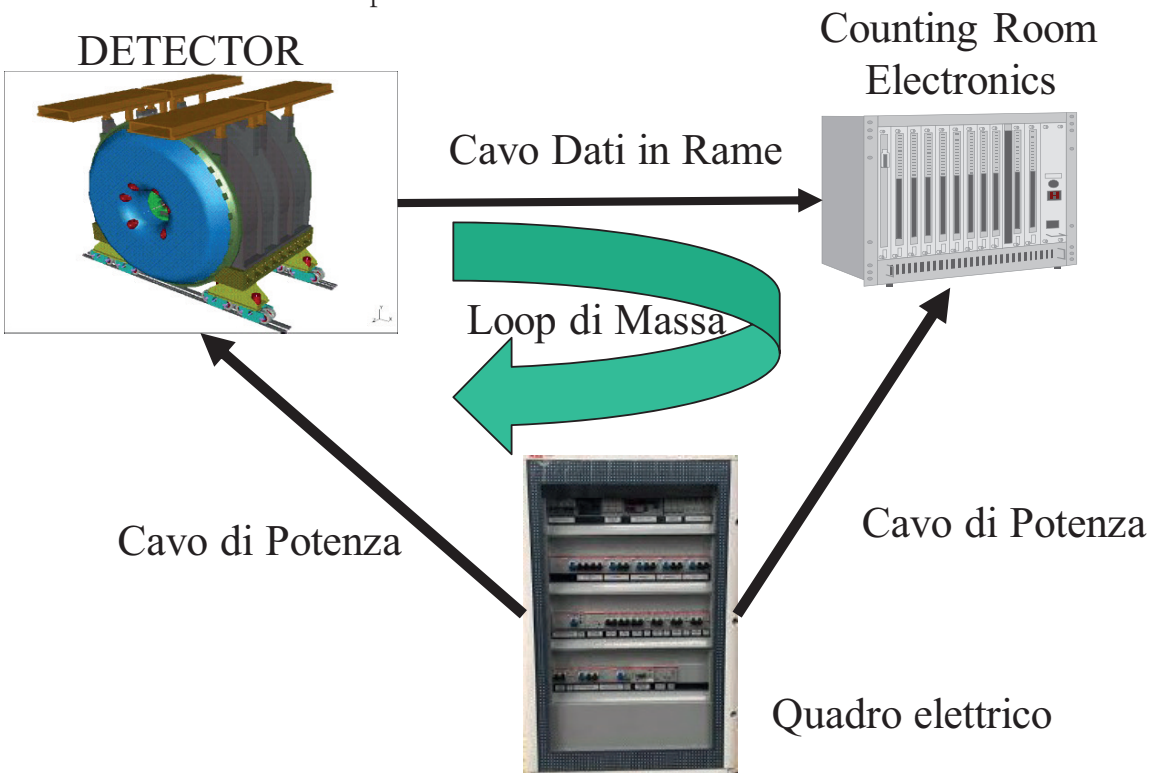


Figura 5.1 - Possibile Loop di Massa fra Detector e Counting Room

Capitolo 6

Acquisizione Dati e Trigger

Il Trigger (in italiano "grilletto") indica comunemente un segnale che avvisa che è avvenuto l'evento ricercato.
Dopo l'integrazione e la discriminazione dei segnali da utilizzare è possibile esprimerlo in termini di logica binaria come una combinazione di AND e OR.
Può avere più livelli, non necessariamente tutti hardware.
Il trigger finale può essere composto da varie configurazioni possibili in OR o in AND fra di loro.

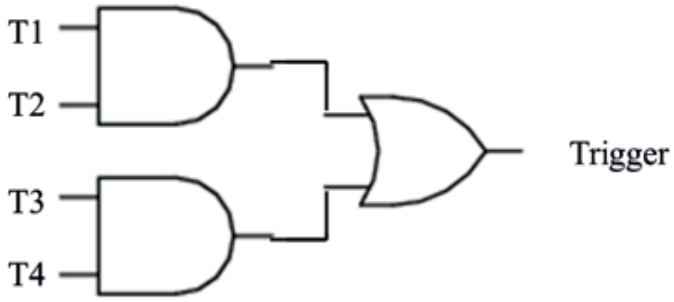


Figura 6.1 - Combinazione logica di segnali di trigger

Generalmente una delle configurazioni è un campionamento delle situazioni che darebbero luogo ad un rigetto. In questo modo si può analizzare un campione di eventi scartati per verificare l'efficienza del Trigger.
Altri tipi di trigger sono definiti per scopi speciali (Calibrazione, Test, ecc.) e vengono attivati di volta in volta a seconda delle necessità.
Le caratteristiche di interesse di un Trigger sono:

- Tempi di risposta: il trigger deve operare nel minor tempo possibile per evitare di introdurre Tempo Morto e quindi possibile perdita di eventi.
- Frequenza: la frequenza con cui scatta il Trigger è rilevante per valutare quanto frequentemente sarà necessario andare ad acquisire i dati sull'elettronica del rivelatore.
- Efficienza: si misura su quanti eventi vengono selezionati dal Trigger rispetto al numero totale degli eventi da selezionare.
- Ridondanza: le configurazioni di Trigger sono generalmente parzialmente sovrapposte per selezionare eventi con più criteri.
- Selettività è la capacità di selezionare gli eventi interessanti senza falsi positivi (vedi successivo Paragrafo 6.1).

6.1 *Selettività*

Il Trigger permette di selezionare solo gli eventi "interessanti" scartando i dati inutili.
In generale è fondamentale effettuare una reiezione del fondo (Background Rejection), cioè degli eventi non interessanti per la fisica che si sta studiando.
Esempi: Cosmici, Beam-gas, Off-momentum particles, Rumore del Rivelatore, ecc.
È necessario eliminare questi eventi perché fanno perdere tempo all'acquisizione e riempiono inutilmente lo storage di dati che non servono per le analisi.
La selettività di un Trigger è dunque la sua capacità di selezionare solo gli eventi di interesse eliminando gli altri (falsi positivi).
Ma attenzione: non sempre bisogna buttare ciò che non riguarda il fenomeno fisico da studiare; a volte è necessario prendere anche altri dati (vedi seguito).

6.2 *Efficienza*

L'efficienza del trigger è dominata da due componenti:

– L'efficienza dell'algoritmo di trigger. In presenza di un gran numero di possibili topologie, i metodi veloci per definire dei possibili candidati generalmente non portano a risultati univoci. D'altra parte non si dovrebbero perdere eventi utili, almeno non in maniera biased. Dato un tempo di processamento, si sceglie un compromesso fra la selettività (alta riduzione del numero di eventi) ed il rischio di inefficienza (perdita di eventi interessanti).
– Perdita per Tempo Morto. La frequenza di accettazione di un trigger è data da: $\mathbf{f=f_0/(1+f_0\tau_r)}$ che esamineremo nel prossimo paragrafo 6.3.

La prima componente dipende dal modo in cui è stato disegnato il trigger in relazione al raggiungimento di obiettivi fisici precisi che riguardano lo studio di fenomeni di interesse. Dunque l'architettura del sistema di trigger può cambiare in maniera significativa se si stanno studiando fenomeni ad una macchina acceleratrice di tipo collider oppure a bersaglio fisso o se invece si sta utilizzando un rivelatore di raggi cosmici. Differenze sostanziali provengono anche dal tipo di fenomeno che si intende studiare e rivelare. Nella fase di progettazione dell'esperimento è quindi necessario introdurre nella simulazione anche il trigger simulando gli algoritmi in rapporto ai fenomeni da rivelare ma anche, ad esempio, di soppressione del rumore nel rivelatore o di altri fenomeni fisici non di interesse e che devono essere rigettati.
A posteriori, avendo a disposizione dati reali, è necessario verificare l'efficienza del trigger e per farlo serve monitorare il suo comportamento sfruttando la ridondanza. Si costruisce pertanto il sistema di trigger in maniera tale che ogni tipologia di evento possa far scattare almeno due sotto-trigger indipendenti ed il trigger globale è l'OR logico di tutti i sotto-trigger. Una valutazione ex-ante dell'efficienza dei sotto-trigger viene quindi fatta attraverso una simulazione che da indicazione sulla loro ridondanza. L'efficienza viene quindi valutata sui dati dopo la ricostruzione.
Nell'esempio seguente, per rivelare elettroni all'interno dell'evento, si usano due sotto-rivelatori:
A – TPC (Track Projection Chamber) un rivelatore di tracce;
B – Un Calorimetro Elettromagnetico.
Ciascuno dei due sotto-rivelatori esprime un sotto-trigger indipendente per rivelare elettroni. Entrambi i sotto-rivelatori sono interessati dagli stessi eventi e dovrebbero rivelare gli stessi elettroni.

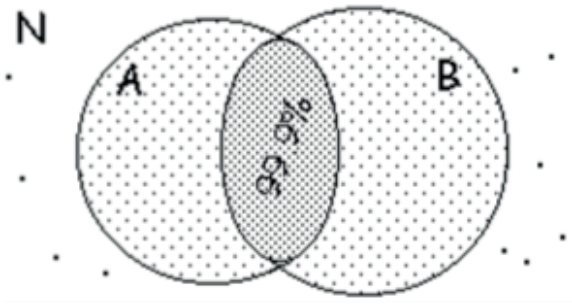


Figura 6.2 - Efficienza di rivelazione

Definiamo dunque N il numero totale degli eventi da rivelare e con NA ed NB il numero di eventi rispettivamente rivelati da A e B. La efficienza di A può dunque essere calcolata come la frazione di eventi che A ha rivelato in coincidenza con B rispetto al totale degli eventi rivelati da B:

$$\varepsilon_A = N(A \cap B) / N_B$$

dove **N(A∩B)** è il numero di eventi di intersezione di A e B.
È rilevante notare che, in questa trattazione semplificata, tale efficienza è relativa alla rivelazione nel suo complesso e non riguarda soltanto il sub-trigger interessato ma anche l'efficienza del rivelatore stesso. Una valutazione accurata dovrà tener conto, oltre a possibili correlazioni fra rivelatori, dell'efficienza del rivelatore sotto i vari aspetti: elettronica, geometrica, ecc.

6.3 *Frequenza di Trigger*

La frequenza di registrazione di eventi nel caso di un singolo livello di trigger è data da:

$$f_1 = f_0/(1+f_0\tau_r)$$

dove:

– f_1 è la frequenza di registrazione,
– f_0 la frequenza di trigger libera (vedere Figura 6.3)
– τ_r il tempo di registrazione dei dati o tempo morto.

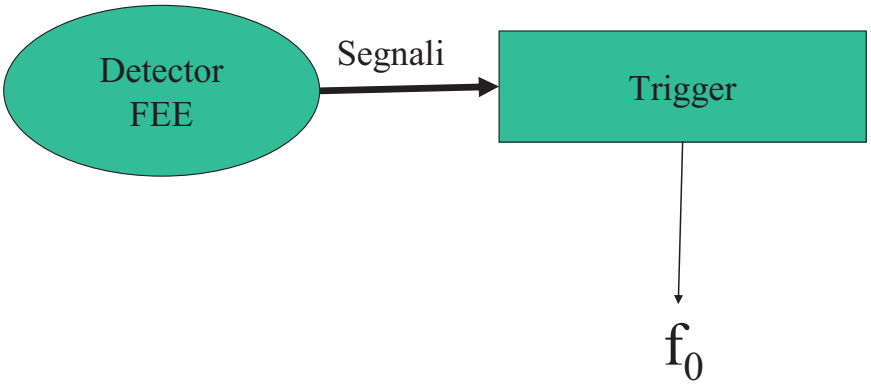


Figura 6.3 - Trigger con Frequenza libera

Lo schema funzionale è quello nella figura che segue e che prevede che il segnale di BUSY che impedisce l'acquisizione di nuovi eventi sia mantenuto fino al completamento della fase di registrazione e dia quindi luogo al tempo morto (DT – Dead Time).
Il DT, in questa ipotesi semplificata, è quindi dovuto esclusivamente alla fase DAQ e si suppone che il DT della logica di trigger sia trascurabile o integrabile nel valore del DT del DAQ. In questo caso la Frequenza che viene vista dalla fase di registrazione è esattamente f_1.

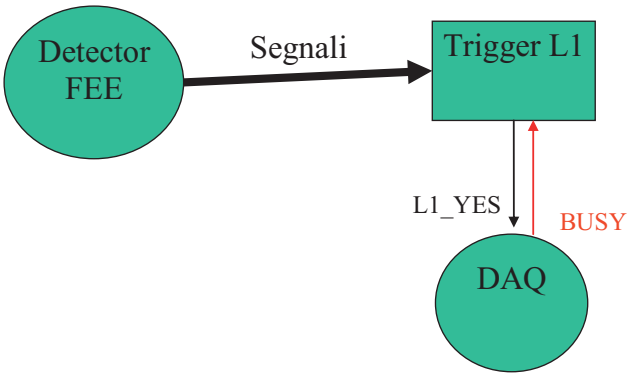


Figura 6.4 - Sistema di Trigger ad un livello con fase di registrazione

Per convincerci che questa sia la formula giusta, partiamo dall'acquisizione del primo evento e poniamo quindi:

$\mathbf{N_a = 1}$ Numero degli eventi acquisiti in tempo $\boldsymbol{\tau}$ che è il tempo morto per acquisirlo

$\mathbf{N_p = f_0\,\tau}$ Numero degli eventi persi in tempo $\boldsymbol{\tau}$ pari alla frequenza di trigger libera per il tempo morto

Dunque $\mathbf{N_T = 1 + f_0\,\tau = N_a + N_p}$ sarà il Numero Totale di eventi nel tempo $\boldsymbol{\tau}$

Il rapporto fra numero di eventi acquisiti e numero totale di eventi è dunque:

$\mathbf{N_a/N_T = 1/(1 + f_0\,\tau)}$ che ci dà la percentuale di eventi acquisiti sul totale.

Dunque su un tempo $\boldsymbol{\tau}$ generico, il numero degli eventi acquisiti sarà proporzionale alla frequenza di registrazione $\mathbf{f_1}$, mentre il Numero Totale di Eventi sarà proporzionale alla frequenza libera di Trigger $\mathbf{f_0}$:

$$\mathbf{N_a = f_1\,\tau} \quad \text{e} \quad \mathbf{N_T = f_0\,\tau}$$

e quindi:

$$\mathbf{N_a/N_T = (f_1\,\tau)\,/\,(f_0\,\tau) = 1\,/\,(1 + f_0\,\tau)}$$

Da cui si ricava che:

$$\mathbf{f_1 = f_0\,/\,(1 + f_0\,\tau)}$$

Nel caso si voglia aggiungere un secondo livello di trigger che magari comporta anche una lettura parziale dei dati, allora lo schema della precedente figura si modifica come segue:

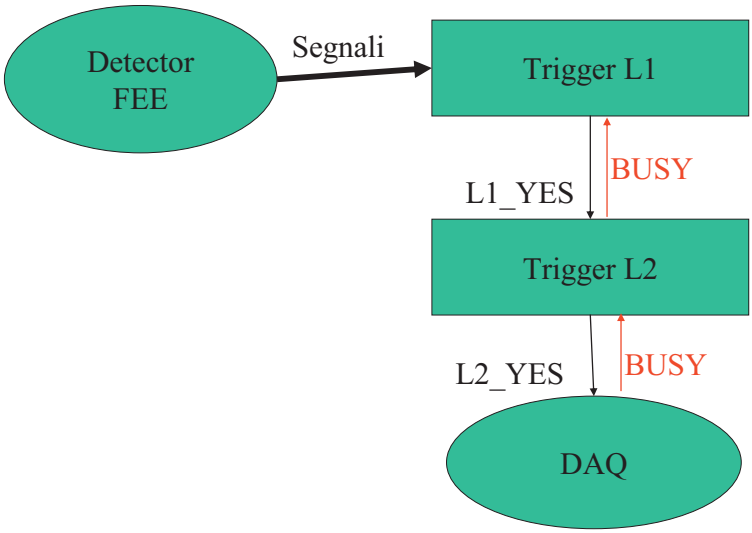


Figura 6.5 - Trigger a due livelli con registrazione dei dati

In questo secondo caso la frequenza di processamento dei trigger sarà data da:

$$f_2 = f_0/[1+ f_0 (\tau_f + \varepsilon \tau_r)]$$

dove:

– f_2 è la frequenza di eventi processati con il trigger di secondo livello;
– ε è la frazione di eventi validati nel secondo livello di trigger;
– τ_f è il tempo morto di decisione del secondo livello che può includere anche una lettura parziale dei dati;
– τ_r è il tempo morto di registrazione dei dati.

Dal rapporto di f_2 / f_1 si può ricavare l'eventuale guadagno nella frequenza di accettazione degli eventi che è misura della migliorata efficienza del sistema.
Perché ci sia un guadagno (riduzione DT):

$$f_2/f_1 > 1 \text{ oppure } \tau_f/\tau_r + \varepsilon < 1$$

Si noti che non è possibile applicare la formula di f_1 ricorsivamente ai due stadi perché essi non sono in realtà indipendenti, così come in Figura 6.4 non è possibile pensare che la frequenza in uscita dalla fase di trigger sia ancora f_0 che è invece la frequenza libera della configurazione in Figura 6.3 che non contempla una fase di registrazione e quindi un segnale di interdizione (BUSY) e un tempo morto conseguente.
Infatti per ogni evento che passa il filtro di secondo livello bisognerà attendere ancora il processo di registrazione per poter rilasciare il BUSY.
Il tempo morto che quindi è applicabile è τ_f (per tutti gli eventi) sommato a una frazione ε di τ_r che riguarda solo gli eventi che passano positivamente la fase di secondo livello.
In sostanza la frequenza che "vede" il secondo livello è esattamente f_2, così come in uscita dal secondo livello c'è una frequenza f_2 complessiva costituita da eventi buoni (εf_2) in ingresso allo stadio di registrazione e di eventi scartati $([1-\varepsilon]\ f_2)$, infatti nel secondo stadio e nella fase di registrazione nessun evento viene perso, tranne quelli scartati dal filtro di livello 2.
Si noti che f_2/f_1 riguarda anche l'efficienza di processamento di trigger, mentre il numero degli eventi buoni registrati nel caso di I livello sarà presumibilmente εf_1. Il secondo livello quindi introduce anche un effetto di "selettività" del Trigger che evita di registrare eventi che dovranno presumibilmente essere scartati in una fase off-line successiva.

6.4 *Trigger number*

È necessario associare un numero univoco al trigger che è scattato per poterlo gestire correttamente in un sistema complesso.
Il trigger number, per ragioni di velocità, viene assegnato a livello HW nel sistema e poi viene collegato ad un Event Number acquisito.
In sistemi complessi e con necessità di decisioni rapide non è possibile o consigliabile distribuire il trigger number ai vari apparati di DAQ (ci sono delle difficoltà anche a distribuire il segnale di Trigger come vedremo).
Uno dei sistemi utilizzabili è quello di avere dei contatori indipendenti e di sincronizzarli con intervalli più o meno lunghi.
Nella Figura 6.6 è mostrato un sistema di contatori distribuiti che sono incrementati dal segnale di Trigger distribuito a tutti i moduli.

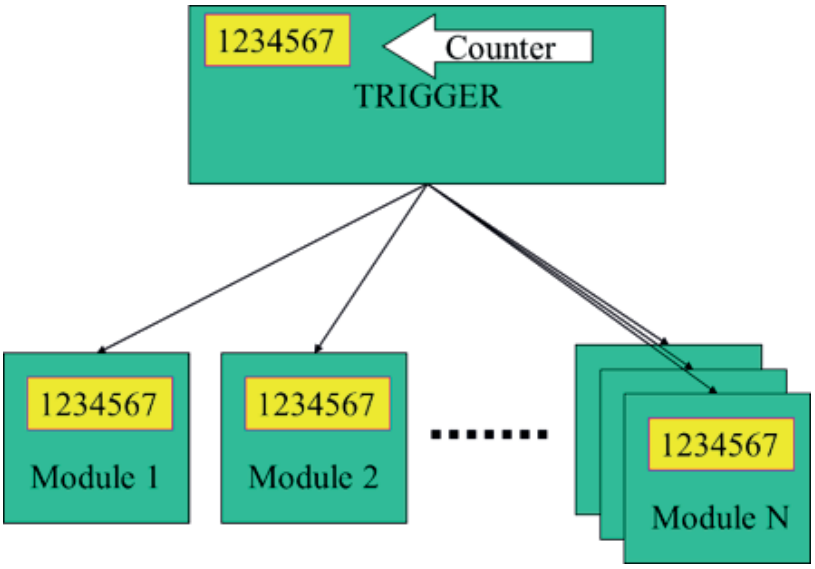


Figura 6.6 - Contatori distribuiti

Nella figura successiva è schematizzato un possibile meccanismo per verificare la sincronizzazione dei contatori. Il meccanismo prevede che allo scadere di un contatore programmato venga effettuato un ciclo di controllo fra tutti i contatori per verificare la corrispondenza del contenuto di essi con il valore di riferimento precaricato.

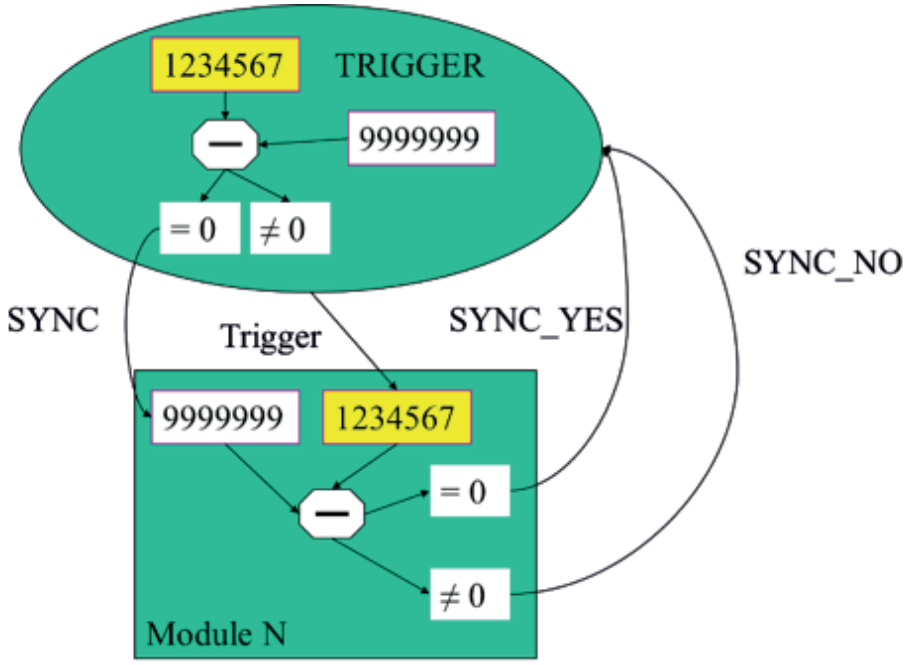


Figura 6.7 - Sincronizzazione dei contatori

In particolare nella figura è mostrato un sistema di Trigger che ha un contatore che si incrementa ad ogni evento (1234567) e un registro con un numero di trigger di riferimento (9999999) che viene confrontato con il valore del trigger attuale attraverso una semplice operazione di sottrazione. Nel caso l'operazione dia un valore diverso da zero non avviene nulla, ma se il risultato è zero significa che è stato raggiunto il momento di controllare la sincronizzazione attivando un segnale (SYNC) verso i moduli. All'interno di ciascun modulo avviene quindi il medesimo confronto fra il valore del trigger attuale ed il registro precaricato. Un risultato zero attiva una risposta

positiva (SYNC_YES) verso il Trigger, mentre un risultato diverso da zero segnala un errore di sincronizzazione (SYNC_NO).
Se tutti i contatori sono sincronizzati il sistema DAQ prosegue, altrimenti va stabilita una politica di re-inizializzazione e di possibile scarto o recupero degli eventi presi dopo l'ultimo periodo di sincronizzazione con esito positivo.

6.5 *Maschera di Trigger*

Il sistema di Trigger ha, in genere, memorie che registrano la configurazione del Trigger che è scattata.
Tali informazioni devono essere lette dal DAQ insieme all'evento interessato per poter permettere un'analisi a posteriori della efficienza e selettività del trigger stesso.
In fase di inizializzazione è possibile caricare nel sistema di trigger la configurazione dei trigger abilitati e delle relative soglie di intervento.

Capitolo 7

Struttura dei sistemi di calcolo

L'uso di computer nei sistemi di acquisizione dati risale agli inizi di questa attività legata agli esperimenti. Naturalmente nel corso degli anni e con l'evoluzione che si è verificata nel campo di quello che viene comunemente definito come Information Technology (IT) molto è cambiato nel modo di utilizzare i computer, nella loro potenza di calcolo ed archiviazione e nella facilità d'uso e di programmazione. Tuttavia lo schema di base di un computer odierno non si discosta in maniera significativa dallo schema previsto da von Neumann.

7.1 *Architettura generale*

L'architettura di un computer è schematizzabile come in Figura 7.1 e prevede un insieme di pochi componenti che svolgono delle funzioni specifiche all'interno del sistema.

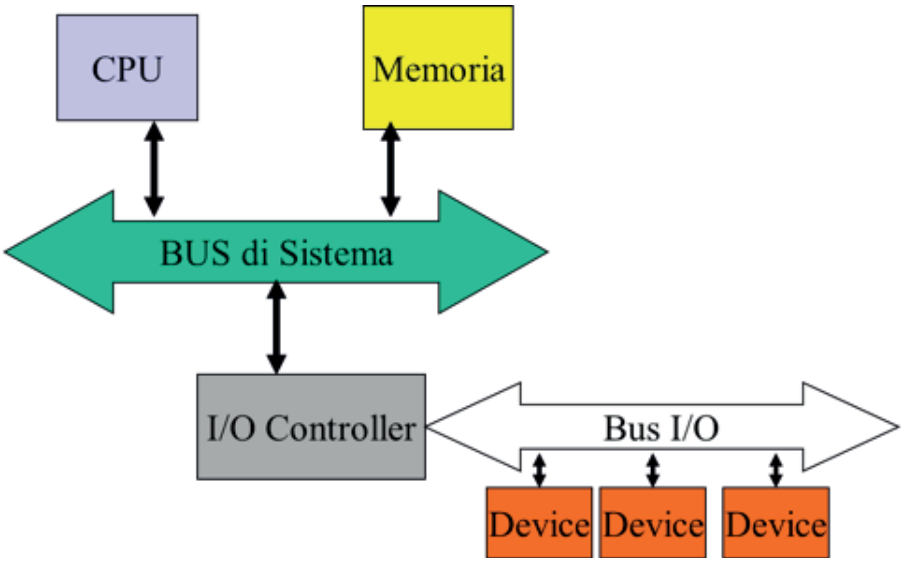


Figura 7.1 - Architettura generale dei sistemi di calcolo

In particolare esaminiamo nei paragrafi successivi le varie parti: CPU, Memoria, Input/Output e Bus di connessione di queste.

7.2 *Central Processing Unit* (*CPU*)

La Central Processing Unit (CPU) è il "cervello" di un sistema di calcolo e sovrintende, tramite una serie di istruzioni di programma, al comportamento generale del sistema.
Le CPU sono generalmente divise storicamente in alcune categorie, le principali sono:

– CISC (Complex Instructions Set Computer) oppure
– RISC (Reduced Instruction Set Computer) oppure
– VLIW (Very Long Instruction Word)

L'indirizzamento, cioè la lunghezza in bit degli indirizzi oggi è principalmente a 64 o 32 Bit che è anche la lunghezza in bit della parola (Word) per rappresentare i dati.
Tale parola è suddivisa in Bytes che è un insieme di 8 bit e che costituisce l'elemento più piccolo da poter trattare all'interno di un programma.

7.2.1 Processori CISC

Un Processore CISC ha un'architettura di istruzioni complesse che possono effettuare varie operazioni di basso livello allo stesso tempo (Es: caricare dalla memoria [Load], effettuare un'operazione aritmetica e registrare in memoria il risultato [Store]).
Nell'architettura CISC, precedente all'avvento di quella RISC (vedi paragrafo seguente), il tentativo era quello di avvicinare il set di istruzioni del computer a quello "naturale" per una programmazione ad alto livello.
Questo comporta generalmente un largo set di istruzioni che devono essere supportate dalla CPU e quindi una maggiore quantità di logica all'interno di esso.
Esempio di architettura CISC è il set di istruzioni X86 della Intel che l'ha ampliata nel tempo per includere istruzioni per il trattamento dei dati multimedia, ecc.

7.2.2 Processori RISC

L'architettura RISC ha invece il goal di costruire un set limitato di istruzioni semplici che possano essere eseguite con grande rapidità.
Nell'architettura RISC il compito di trasformare le istruzioni di alto livello date dal programmatore in istruzioni semplici per la CPU è assegnato al Compilatore, un programma che trasforma le istruzioni del linguaggio di programmazione usato dal programmatore in istruzioni di macchina per il computer.
Inoltre per aumentare la velocità vengono utilizzate più unità logiche (ALU = Arithmetic Logical Unit) che possono lavorare in parallelo e una logica HW stabilisce quali istruzioni mandare in esecuzione allo stesso tempo.
Questa architettura viene chiamata Super Scalare.
Esempi di architetture RISC sono i Processori:

– PowerPC della IBM e Motorola
– Alpha della Digital Equipment, acquisita da CompaQ, acquisita da HP
– PA-RISC della HP
– SPARC della SUN Microsystems ora acquisita da Oracle.

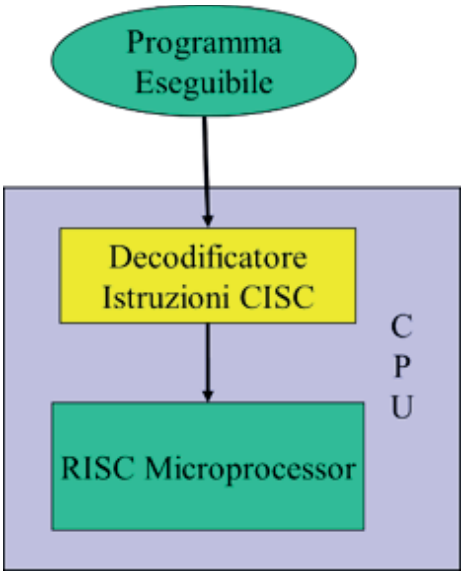


Figura 7.2 - Schema di programmazione di una CPU RISC

In realtà oramai anche le CPU originalmente pensate come CISC utilizzano molta tecnologia RISC nella loro realizzazione pratica e le istruzioni CISC vengono tradotte in istruzioni RISC a volte all'interno del processore stesso da una sezione di decodifica.

7.2.3 Processori VLIW (Very Long Instruction Word)

I processori VLIW utilizzano architetture in cui ci sono più unità logiche che svolgono il medesimo compito (Es: più ALU) e presentano quindi più operazioni in parallelo alla CPU.
Rispetto alla architettura Super Scalare però, il compito di decidere quali istruzioni devono essere processate in parallelo dalle unità logiche viene lasciato al Compilatore (non c'è HW decisionale specializzato), per cui si dice anche che in questi processori il parallelismo è esplicito (Explicitly Parallel Instruction Computing).
Esempio di architettura EPIC è la IA-64 della Intel che è stata realizzata alcuni anni fa nel processore Itanium e più recentemente in GPU AMD.

7.3 *Memoria*

La memoria è un sistema di archiviazione di dati ed istruzioni da eseguire (il Programma o Task).
La memoria è una risorsa critica all'interno del sistema perchè dalla sua capacità e velocità di accesso dipendono buona parte delle prestazioni del sistema stesso.
Le architetture di Memoria sono in generale divise nelle categorie:

– Memoria ad accesso Uniforme
– Memoria ad accesso non uniforme o Non Uniform Access Memory (NUMA).
– Memoria Cache.

Dal punto di vista HW le memorie hanno (o non hanno) dei sistemi di rilevazione di errore:

– Parity Check Memory: un semplice bit di parità permette di controllare se gli altri bit in quella locazione di memoria sono corretti.
– Error Correction Code (ECC) Memory: usa un byte in più per poter, non solo rivelare un errore ma anche poterlo correggere.

7.3.1 Memoria ad accesso uniforme

Nel caso di un accesso uniforme, la CPU accede a tutti i banchi di memoria in maniera identica. Nel caso di due o più processori, l'accesso ai banchi di Memoria è uniforme per entrambi i processori e per entrambi i banchi come mostrato nella figura seguente.

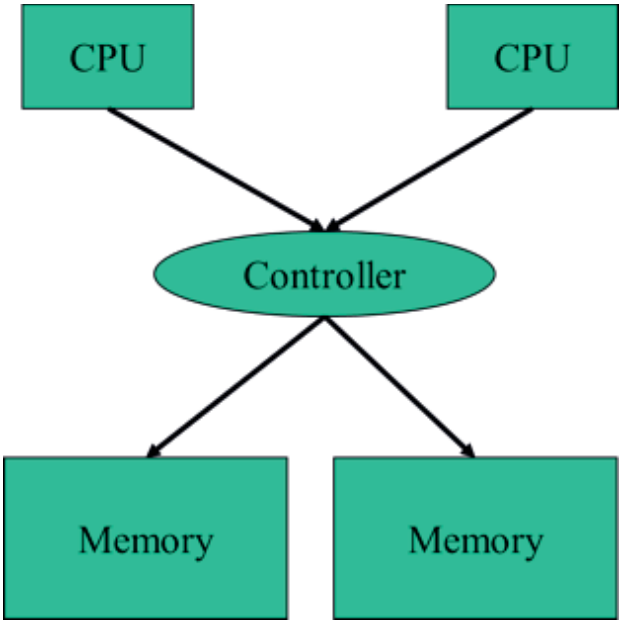


Figura 7.3 - Memoria ad accesso uniforme

Nello schema mostrato in Figura 7.3 i processori accedono alla memoria attraverso un controller comune e i due banchi di memoria sono quindi equivalenti per entrambi i processori.

7.3.2 Memoria ad accesso non uniforme

Nel caso di accesso non uniforme alla memoria, i banchi di memoria possono non essere equivalenti fra loro nel senso che le tecnologie e/o le metodologie di accesso possono essere differenti. Nelle figure sottostanti sono mostrate due possibili configurazioni di Memoria ad Accesso non uniforme (NUMA).

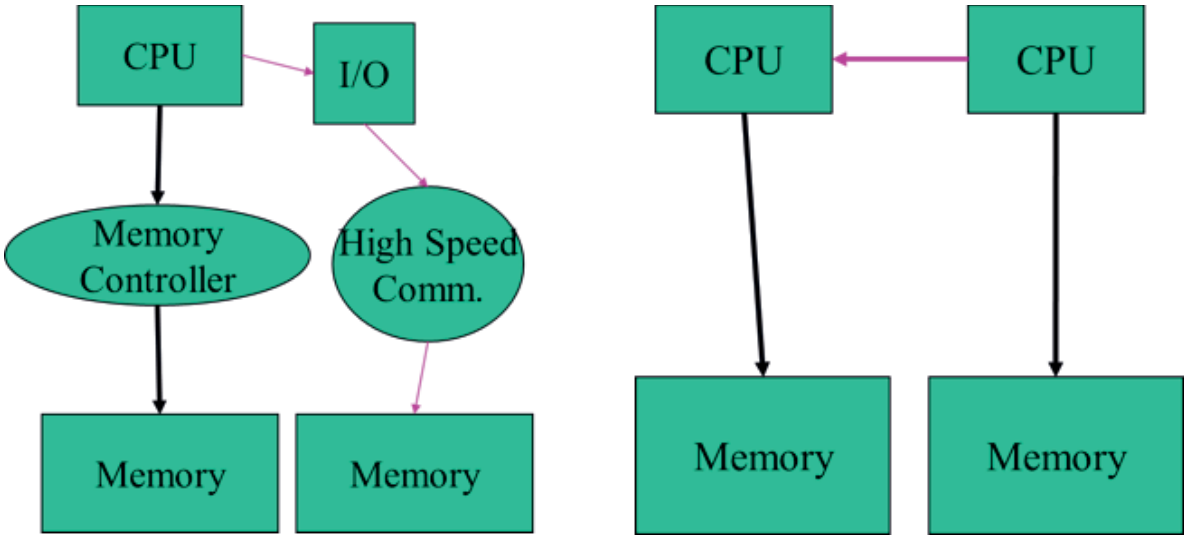


Figura 7.4 - Esempi di Memoria ad Accesso Non Uniforme (NUMA)

Nell'esempio di sinistra i due banchi di memoria sono interfacciati al sistema con due differenti tipologie di collegamento: il primo banco è infatti collegato alla CPU attraverso un Memory Controller interno, mentre il secondo banco è esterno ed accessibile attraverso un canale ad alta velocità di Input/Output. Questo crea la necessità per la CPU di usare meccanismi diversi (istruzioni diverse) per accedere all'uno o all'altro e rende necessaria la conoscenza di dove sono (quale banco di memoria) i dati da utilizzare.
Nell'esempio di destra è invece rappresentato il caso in cui i banchi di memoria siano direttamente collegati a due processori e l'accesso da parte di un processore alla memoria dell'altro processore richiede il passaggio attraverso questa seconda CPU.
In entrambe le situazioni l'accesso alla memoria con accesso più "indiretto" sarà naturalmente più lento e rallenterà i programmi che dovranno accedere a quei dati.

7.3.3 Memoria cache

La memoria di sistema, anche quando sia di tipo uniforme può non essere sufficientemente veloce per garantire il flusso di istruzioni e dati richiesto dai processi in elaborazione sulla CPU. Le memorie molto veloci sono generalmente anche molto costose soprattutto quando vengono richiesti oramai molte decine di GBytes per le Memorie di sistema e, quindi, quelle utilizzate nei computer sono un compromesso fra capacità, velocità e costo. Per velocizzare il flusso dei dati verso il processore si ricorre ad un sistema, spesso a più livelli, di memorie più piccole e rapide che possano contenere i dati e le istruzioni di maggiore utilizzo da parte del processore. Tali memorie sono chiamate Memorie Cache.
La cache funziona sostanzialmente come una raccolta di istruzioni o dati che sono la copia di altri dati situati in una memoria principale con accesso più lento.
Per accelerare le operazioni, la CPU utilizza le copie dei dati nella cache che ha un accesso più veloce (generalmente è sullo stesso chip del processore).
Ogni dato in cache ha anche una etichetta che identifica il dato nella memoria principale di cui esso è una copia.
Quando la CPU vuole accedere quel dato, prima viene controllata la Cache e, se la copia del dato è presente in essa, viene usata quella al posto del dato originale nella memoria più lenta.
Questa situazione favorevole si chiama **Cache hit**; il caso contrario **Cache miss**. La percentuale di successi nell'accedere i dati in cache prende il nome di **hit rate** o **hit ratio**. La cache può essere usata anche in scrittura per velocizzarla nei confronti della memoria standard. La cache può avere due politiche di scrittura:

– **write-through** in cui il dato viene aggiornato anche nella memoria principale all'atto della scrittura da parte della CPU;
– **write-back** in cui il dato viene scritto solo nella cache e non viene aggiornato anche nella memoria principale ma questa operazione viene rimandata ed effettuata nel caso si verifichino particolari condizioni per ottimizzare il trasferimento dati dalla cache alla memoria principale.

Il procedimento di write-through è sostanzialmente più lento perché richiede l'accesso alla memoria principale che è più lenta della cache, ma permette di avere la sicurezza che il dato sia stato aggiornato nella memoria e che, pertanto, ogni nuovo accesso a qual dato restituirà un valore aggiornato. Il procedimento write-back è più veloce perché richiede il solo accesso alla cache ma il dato nella memoria principale non viene aggiornato in tempo reale, ma solo successivamente.

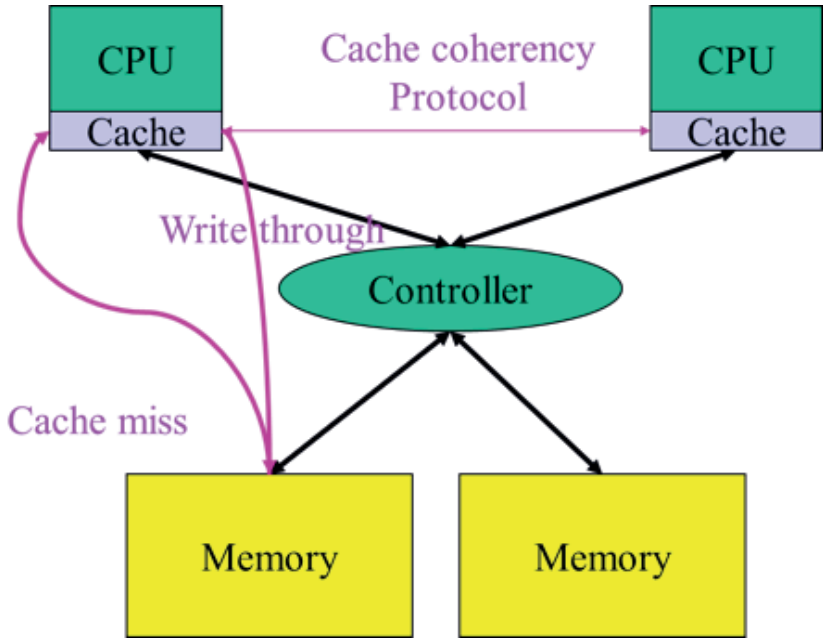


Figura 7.5 - Meccanismo di coerenza delle cache

Nel caso in cui un dato venga aggiornato in cache questo dovrebbe invalidare le altre copie nella Memoria permanente e nelle altre cache. Per regolare questo meccanismo esiste fra le cache un protocollo di coerenza chiamato Cache Coherency Protocol.

7.4 *Il BUS*

Abbiamo già citato nel precedente Capitolo 4 il Bus come una delle topologie trasmissive. In pratica un BUS è un insieme di conduttori elettrici che connettono varie unità funzionali all'interno di un sistema. Come già visto, un BUS può essere:

– Parallelo: consente di trasferire più bits in parallelo; la grande maggioranza dei bus lo è, come ad esempio quelli all'interno di un Computer.
– Seriale: i bit sono impacchettati in sequenza e spediti lungo il cavo.

Normalmente all'interno di un computer ci sono uno o più Bus i più importanti dei quali (sistema e I/O) sono paralleli. Un bus parallelo può ospitare sulle proprie linee:

– Linee Dati.
– Linee di Indirizzi.
– Linee di Controllo.

La larghezza del bus dati è il numero di conduttori che in parallelo trasferiscono l'informazione (Es: 8, 16, 32 bits).
La frequenza di clock del sistema, insieme con la larghezza del bus dati, determina la quantità teorica di dati trasferibile per unità di tempo (Es: 16 Bits x 10 MHz = 160 Mbits/s = 20 MBytes/s).
Un esempio di Bus parallelo interno al computer è il Peripheral Component Interconnect (PCI) già citato alla fine del Capitolo 4.

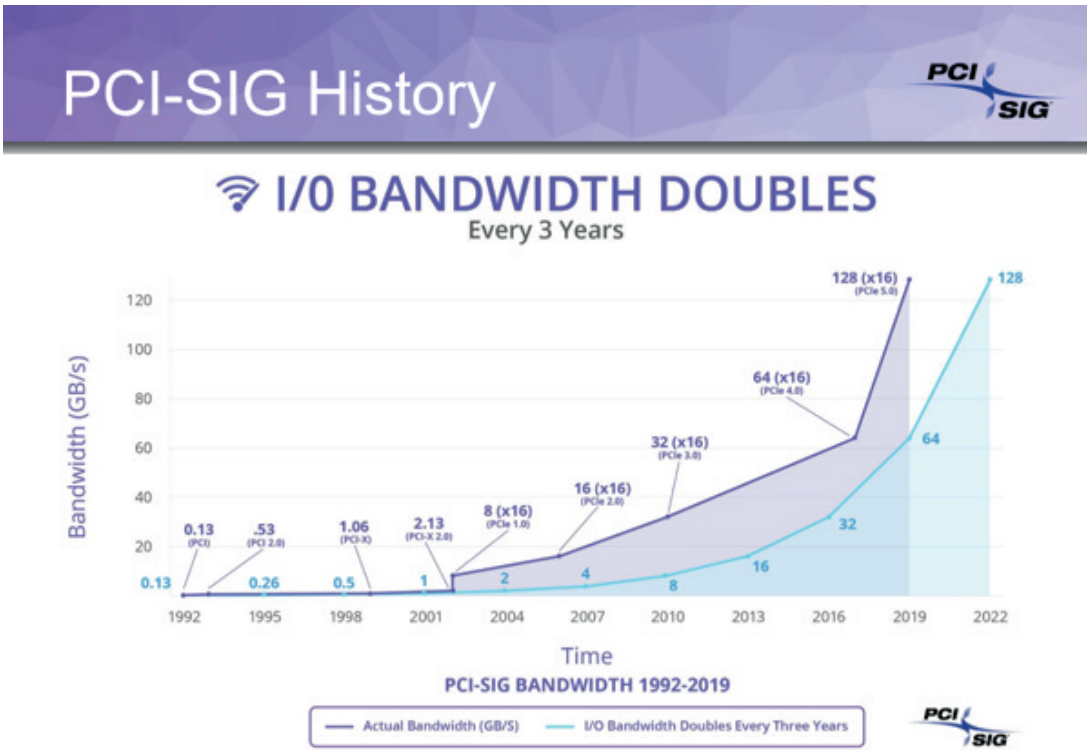


Figura 7.6 - Evoluzione PCI e PCI-Express dal PCI Special Interest Group

Il PCI è uno standard iniziato da Intel nel 1990 e diffusosi in tutti i computer (non solo PC) e sul quale sono stati collegati processori e memorie, nonché dispositivi di I/O. Lo standard originale prevedeva un clock di 33,33 MHz e 32 bit di larghezza (banda passante di circa 133 MB/s) ed alimentazione a 5V.
Gli standard successivi principali sono:

– PCI 2.2 – 66 MHz x 64 bits (533 MB/s) ed alimentazione a 3.3V.
– PCI-X – 133 MHz x 64 bits (1066 MB/s)
– PCI-X 2.0 – 266 o 533 MHz

Come già indicato nel Capitolo 4 il PCI originale è stato soppiantato dal PCI-Express che ha caratteristiche diverse che provengono dalla implementazione di InfiniBand che esamineremo in un capitolo successivo.

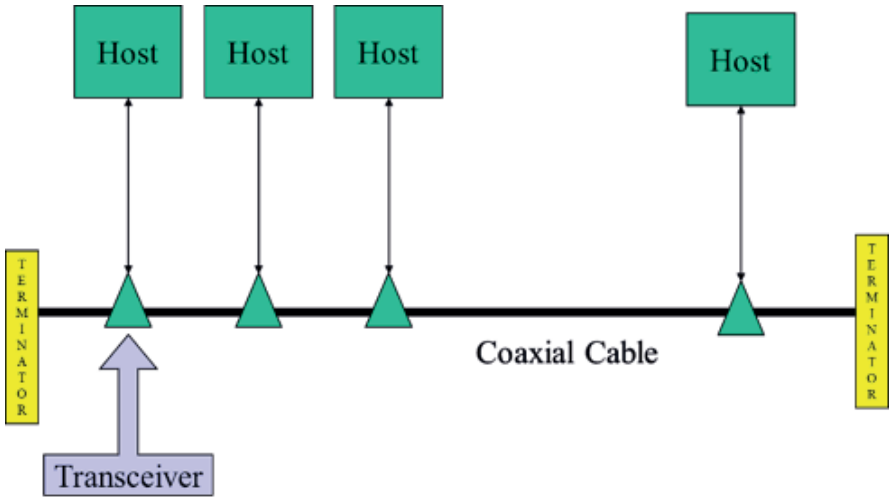


Figura 7.7 - Originale configurazione a bus di Ethernet

Per completezza, citiamo come esempio di BUS Seriale Ethernet su cavo coassiale (10 Base5 o 10 Base2) nella sua struttura originaria. Il mezzo (cavo) è condiviso fra più trasmettitori/ ricevitori ed è quindi usabile a turno con un protocollo CSMA/CD (Carrier Sense Multiple Access/ Collision Detection) in cui:

a) Il trasmettitore ascolta il cavo per vedere se c'è un segnale in trasmissione (Carrier Sense).

b) In caso il cavo sia libero trasmette e continua ad ascoltare per controllare che non ci sia collisione con un altro trasmettitore.

c) In caso venga rivelata una collisione (Collision Detection) viene inviato un segnale di Jam che invalida la trasmissione avvenuta ed i trasmettitori aspettano un tempo casuale per riprovare a trasmettere.

7.5 *Control Status Register (CSR)*

È un Registro di memoria che all'interno di un dispositivo permette di leggere delle informazioni sullo stato di quel dispositivo o di scrivere delle configurazioni di uso.
Esempio: il CSR può contenere il tipo di Modulo ed il codice del Produttore definito dalla IEEE.

7.6 *Direct Memory Access*

Con Direct Memory Access (DMA) si individua la caratteristica di alcune architetture di computer che permette di leggere e scrivere in memoria in maniera indipendente dalla CPU.
È una forma limitata di Bus Mastering in cui i dispositivi che devono trasferire dati li ricevono o li trasmettono in Memoria senza passare attraverso il processore. La CPU si limita ad impostare l'indirizzo di partenza ed il numero di parole da trasferire e poi viene notificata dell'avvenuto trasferimento solo alla fine dello stesso. Questa metodologia di trasferimento dati ha il vantaggio di essere estremamente veloce e di non pesare sul processore che può svolgere altri compiti mentre è in corso il trasferimento DMA. Naturalmente tali vantaggi sono apprezzabili solo nel caso di trasferimenti di significative quantità di dati e non portano benefici nel caso di trasferimento di singole Word o di un numero limitatissimo di esse.

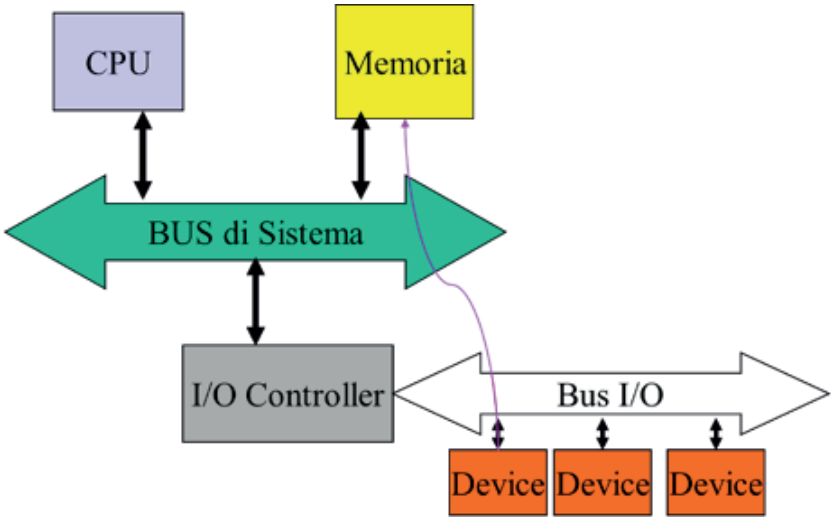


Figura 7.8 - Meccanismo di DMA

7.7 *Programmed I/O (PIO)*

Per quei sistemi che non possiedono un meccanismo DMA oppure per trasferimenti con un numero limitatissimo di dati, è sempre possibile usare un altro meccanismo di trasferimento dati all'interno del computer che invece prevede l'intervento del processore. Il metodo Programmed I/O (PIO) prevede che il transito dei dati avvenga sotto il completo controllo ed attraverso la CPU che si occupa di spostarli nella Memoria.

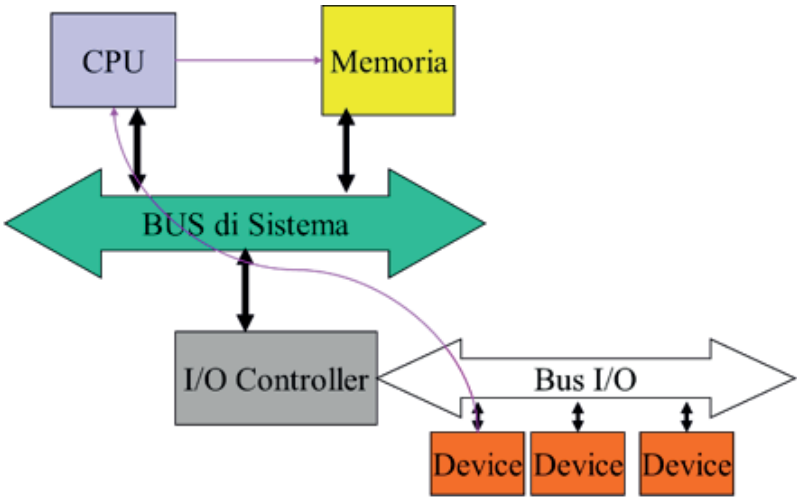


Figura 7.9 - Schema di trasferimento dati in PIO

Il meccanismo di PIO è evidentemente più lento rispetto al DMA perché prevede più passaggi per ogni singolo dato trasferito, prima verso la CPU e poi da questa verso la Memoria. Il PIO è però un meccanismo semplice da usare ed è l'unico nel caso che il sistema non preveda il supporto al DMA.

7.8 *Architettura dei Sistemi di Input/Output (I/O)*

Avendo esaminato le componenti di un sistema di calcolo ed i meccanismi di trasferimento dati, esaminiamo ora come funzionalmente è possibile effettuare delle acquisizioni da dispositivi che sono collegati al sistema.
Sono individuabili due categorie: Sistemi ad Interrupt e Sistemi di Polling.

7.8.1 Sistema ad Interrupt

I sistemi ad Interrupt prevedono la possibilità di segnalare la presenza di dati da trasferire tramite uno o più segnali verso il Processore che viene così interrotto (Interrupt) nel suo processamento per poter servire il dispositivo in questione.
Da un punto di vista HW si identificano in generale i seguenti componenti:

– Sensore che serve per trasferire una informazione di stato o una misura (tensione, corrente, temperatura, ecc.)
– Attuatore che trasforma i comandi ricevuti in segnali di controllo del sistema esterno (tensione, corrente, ecc.)
– Interrupt generator che provvede a generare un Interrupt in base allo stato del sensore o ai segnali da esso ricevuti.
– Buffer di memoria per eventuali dati da leggere dal sistema esterno.

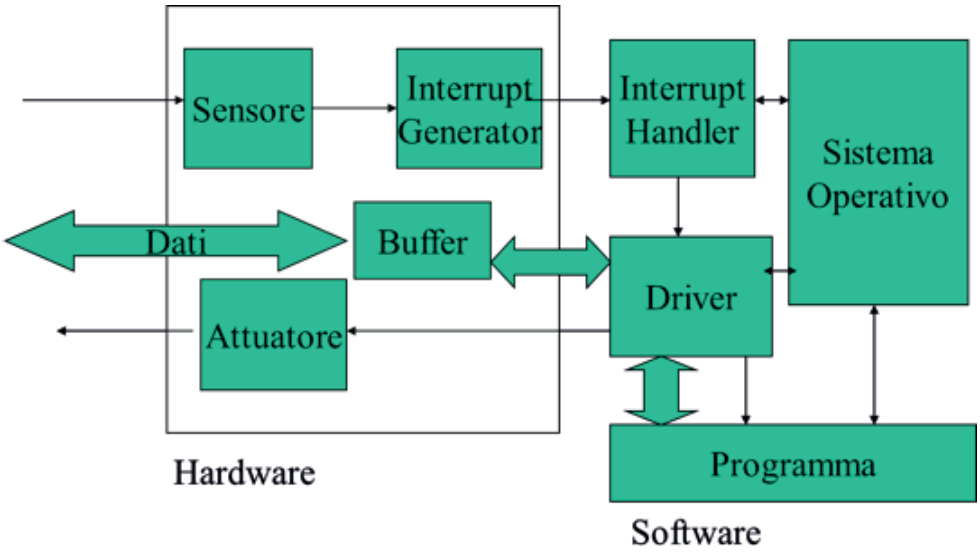


Figura 7.10 - Schema di sistema ad Interrupt

Nella parte SW si identificano in generale i componenti:

– Sistema Operativo: è il sw che controlla l'HW del computer e l'esecuzione dei programmi; è composto da varie parti: Kernel, File System, ecc.
– Interrupt Handler è un modulo che viene attivato dal Sistema Operativo a seguito di un Interrupt per gestirne gli effetti.
– Driver è un modulo SW che interagisce con il SO per controllare una specifica parte di HW (device, sistema di I/O, ecc.)
– Programma è l'applicazione utente che contiene la logica di utilizzo, ad alto livello del sistema HW e del computer.

Il sistema ad Interrupt, pur essendo un po' complesso da implementare e gestire, offre però il vantaggio di non impegnare il processore se non quando c'è effettivamente un dato da leggere e su cui operare. Di conseguenza il sistema ad Interrupt è più efficiente soprattutto quando le letture non sono troppo frequenti o addirittura continue.

7.8.2 Sistema con Polling

Se non c'è un sistema di Interrupt, allora il meccanismo non può che basarsi sul Polling (sondaggio). In questo modo di operare, non c'è il processo asincrono (interrupt) che "sveglia" o attiva gli altri processi, ma si legge, con una periodicità prefissata, lo status del sensore per verificare se è mutato e se è il caso di effettuare delle operazioni.
Il Polling è più semplice da implementare a livello SW e quando non è disponibile un meccanismo di Interrupt hardware.

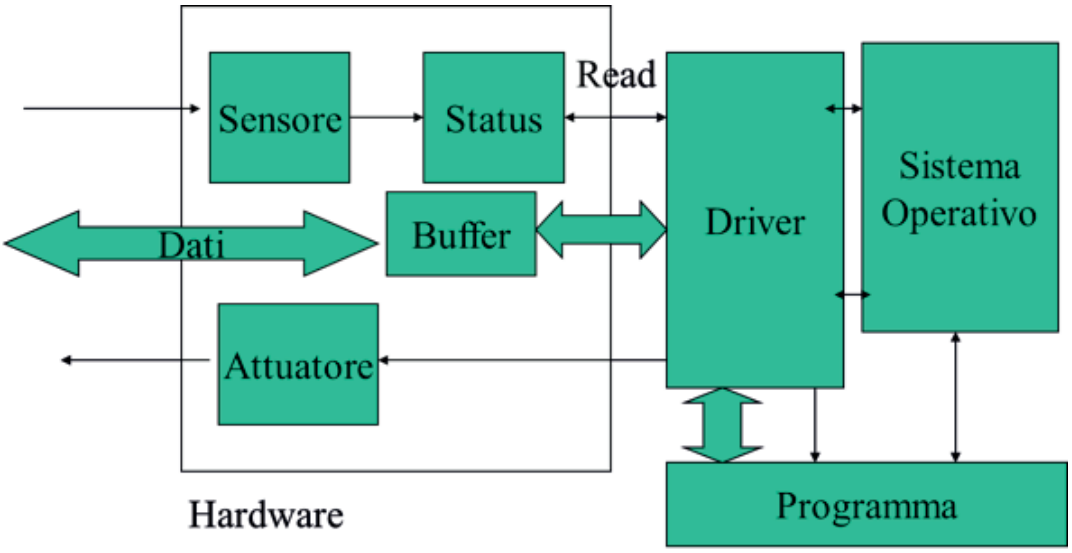


Figura 7.11 - Schema di Sistema a Polling

Capitolo 8

Controllo di Processo e Real Time

I sistemi che lavorano in Tempo Reale (Real Time o RT) sono quelli che richiedono un rispetto delle tempistiche con cui certe azioni o risultati sono effettuati. Tali sistemi sono dunque legati, non solo alla correttezza delle operazioni svolte, ma anche ai tempi in cui tali operazioni sono effettuate. Si potrebbe pensare dunque che un sistema RT debba essere un sistema particolarmente veloce ma, come si vedrà nel seguito questo non è necessariamente vero in tutti i casi. Iniziamo pertanto ad esaminare i sistemi di Controllo di Processo per poi approcciare i concetti base del RT le tipologie di RT, i meccanismi di Task Scheduling ed infine i Real Time Operating System (RTOS).

8.1 *Controlli di Processo*

Nei sistemi di controllo, si possono individuare tre componenti basilari:

– Il Sistema che deve essere controllato che è generalmente composto da sensori, attuatori, cavi, ecc.;
– Il controllore o Controller che invia i comandi al sistema in funzione degli obiettivi previsti per il controllo di esso;
– L'Ambiente o Environment dove il sistema di controllo opera.

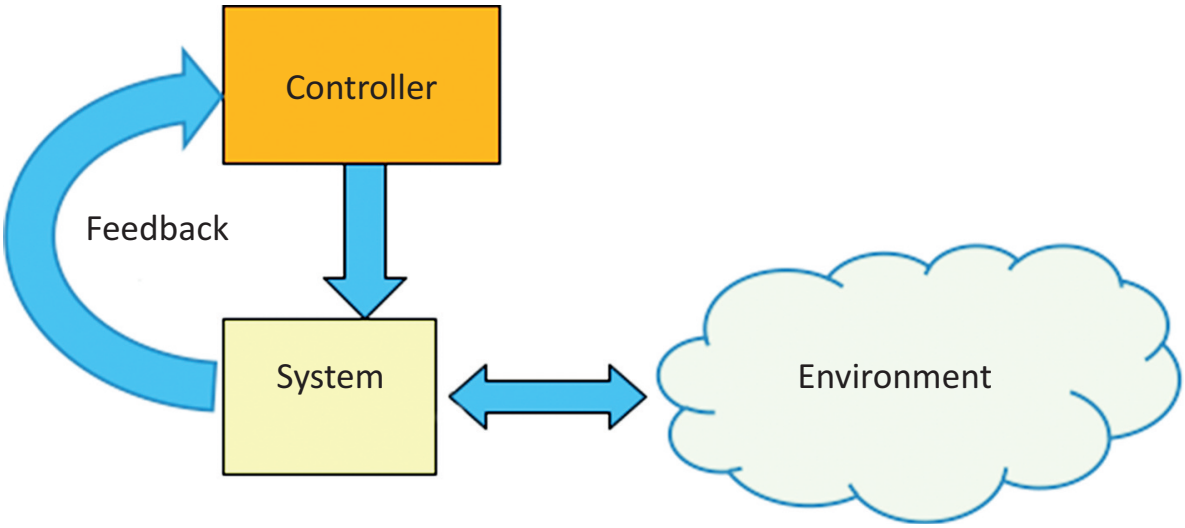


Figura 8.1 - Schema Generale dei Sistemi di Controllo

Nella figura precedente è mostrato lo schema generale ed i suoi tre componenti principali. Il Controller gestisce il System impartendo delle direttive ed ottenendo dal sistema delle informazioni di ritorno (Feedback) che possono o meno produrre nuovi comandi impartiti dal Controller al sistema. Il Sistema interagisce con l'Ambiente ricevendo informazioni e apportando modifiche secondo quanto richiesto dai comandi ricevuti dal Controller.
Nella figura seguente è mostrato un sistema di controllo con maggiori dettagli ed evidenziando alcune componenti e funzioni specifiche. In particolare, il controller può ricevere come Input, sia istruzioni dall'esterno che informazioni provenienti dal System e/o a seguito dell'elaborazione di dati relativi alle letture di sensori che fanno monitoraggio dell'Environment. Il System, da un lato riceve informazioni dai Sensori e, dall'altro produce un feedback verso il Controller, nonché comanda degli Attuatori che operano sull'ambiente da controllare. I dati provenienti dai Sensori sono filtrati e poi elaborati per poter, sia informare il Controller, che fornire un flusso di informazioni verso l'esterno sullo stato del sistema sotto controllo.

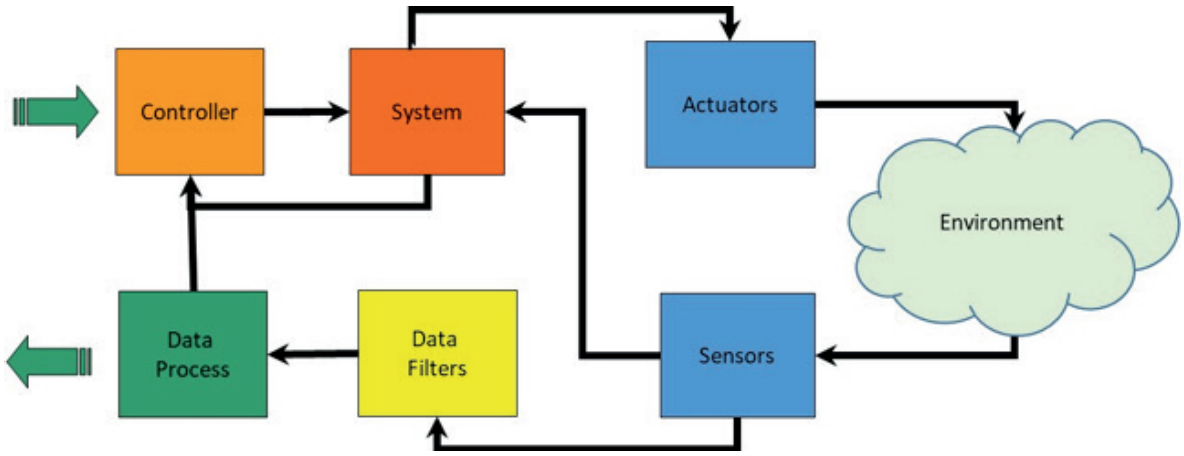


Figura 8.2 - Dettaglio di un Sistema di Controllo

A seconda del tipo di interazione con l'Ambiente, si possono distinguere 3 tipi di sistemi di controllo:

– Sistemi di Monitoraggio (Monitoring Systems) che non modificano l'ambiente;
– Sistemi di Controllo ad anello aperto (Open-loop control systems) che effettuano solo leggere modifiche sull'ambiente;
– Sistemi di Controllo ad anello chiuso (Closed-loop control systems) che hanno una interazione forte fra la percezione dello stato dell'Ambiente e l'attivazione di azioni conseguenti.

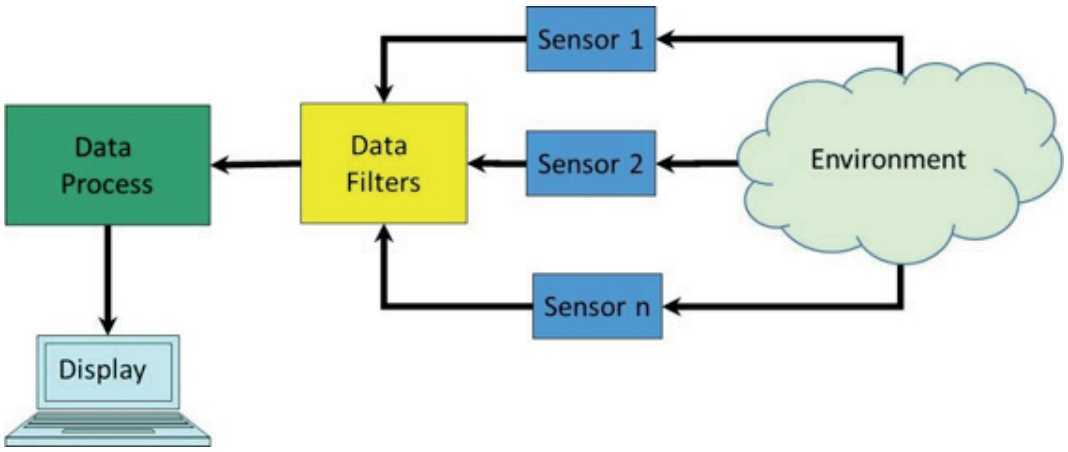


Figura 8.3 - Sistema di Monitoring

La Figura 8.3 mostra uno schema di un Sistema di Monitoring che non modifica l'Environment. Un esempio pratico sono i Sistemi di Sorveglianza o quelli di Controllo del Traffico Aereo.
Nella Figura 8.4 è mostrato lo schema di un sistema Open-Loop in cui la reazione sugli attuatori è frutto di un percorso complesso che dalla lettura dei sensori passa attraverso un processamento dei dati, ad una pianificazione dei possibili interventi, alla decisione di un controller che agisce sul sistema che finalmente comanda gli attuatori.
Un tale sistema non è dunque pensato per reazioni immediate e per produrre decisioni semplici. Esempi pratici di tali sistemi sono robot di montaggio, catene di montaggio industriali e sistemi di controllo di accessi.

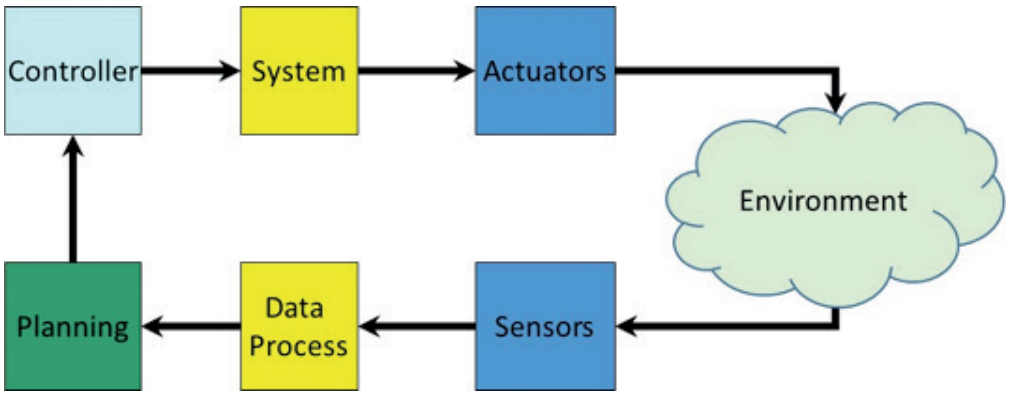


Figura 8.4 - Open-Loop Control System

L'ultimo schema mostrato in Figura 8.5 mostra un Sistema di tipo Closed Loop in cui la parte dei sensori ha un accoppiamento più stretto con la parte degli attuatori, sia in maniera diretta che attraverso dei percorsi brevi (shortcut) che permettono una reazione immediata anche senza il coinvolgimento del controller o di elaborazioni dei dati letti dai sensori.

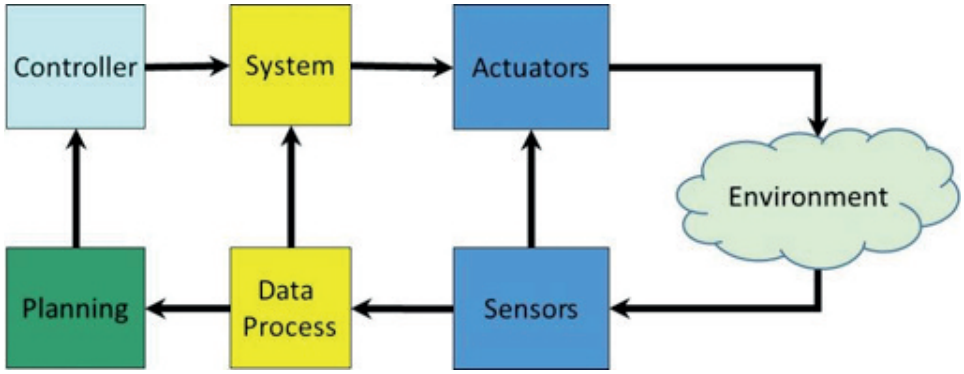


Figura 8.5 - Closed Loop System

In questo ambito rientrano sistemi di controllo di volo o sistemi di sicurezza in questi ultimi, infatti, rientrano i sistemi antincendio. Anche molte delle reazioni umane possono essere catalogate sotto questa categoria.
In conclusione, se c'è una interazione stretta con l'ambiente, il Sistema deve reagire agli eventi all'interno di specifici limiti temporali. Tali limiti temporali sono imposti dall'ambiente stesso ed il Sistema di Controllo deve essere in grado di eseguire i propri compiti all'interno di tali limiti temporali.

8.2 *Sistemi Real Time*

A questo punto introduciamo i sistemi real time: un Sistema Real Time (RTS) è un sistema in cui la correttezza del comportamento non dipende soltanto dalla correttezza logica dei calcoli ma anche dal tempo impiegato per ottenere il risultato. Se i limiti temporali previsti non sono rispettati il sistema fallisce il suo compito.

Possiamo anche definire un RTS come un Sistema capace di garantire la soddisfazione delle necessità di temporizzazione dei processi che vengono controllati ed in particolare:

– È essenziale che i limiti temporali siano garantiti;
– Garantire il comportamento temporale richiede che il sistema sia predicibile;
– È anche auspicabile che il sistema supporti un altro grado di utilizzazione pur soddisfacendo i limiti temporali del sistema;
– Il sistema RT non deve essere il più veloce possibile: essere veloce è una caratteristica positiva ma non garantisce un comportamento sempre corretto se non in relazione all'ambiente su cui si opera;
– In sostanza una risposta precoce o tardiva è una risposta sbagliata;
– Si tratta in generale di garantire il caso peggiore rispetto ad un approccio best-effort.

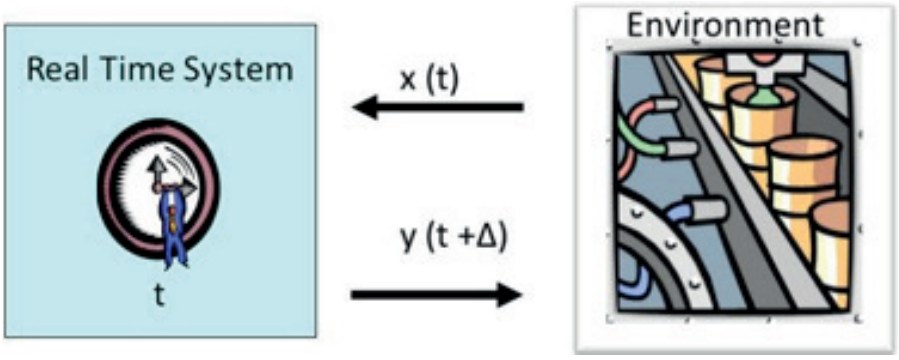


Figura 8.6 - Esempio di sistema Real Time

Alcuni esempi di sistemi di questo tipo sono:

– Un impianto di imbottigliamento in cui il sistema di controllo deve rilasciare la giusta quantità di liquido nelle bottiglie ma solo al momento in cui esse sono poste sotto il beccuccio del dispensatore e non prima o dopo;
– I servo meccanismi nel pilota automatico di un aeroplano che deve apportare le giuste correzioni alla rotta con tempi di reazione adeguati e non troppo improvvisi.

Un sistema di tipo Real Time deve quindi avere le seguenti caratteristiche:

– Veloce in relazione alla risposta ad eventi asincroni: il tempo che intercorre fra lo stimolo esterno e l'inizio della reazione del sistema è chiamata Latenza (Latency) e deve essere quanto più piccola possibile;
– Predicibile nel senso di essere in grado di determinare con certezza il tempo di esecuzione delle proprie attività (Task) e quindi il comportamento temporale del sistema è sempre all'interno di una finestra temporale accettabile ed il sistema nel suo complesso può assumere che tutti i Task rispetteranno tutte le scadenze;
– Deterministico è un caso speciale di sistema predicibile in cui non solo il comportamento temporale del sistema è all'interno di un certo intervallo, ma tale comportamento temporale può essere predeterminato, per esempio, pre-allocando delle finestre temporali per ciascuno dei Task. L'esecuzione di ogni Task, in questo caso, avviene solo

durante la propria finestra temporale. Il Determinismo in un Sistema Real Time, non è sempre essenziale per costruire un sistema RT predicibile.

I sistemi Real Time possono essere classificati in due categorie (Figura 8.7) Hard RT (HRT) e Soft RT (SRT). I primi sono quei sistemi capaci di garantire in maniera stringente i tempi e di non mancare le scadenza temporali in nessuna occasione e devono quindi avere anche latenze molto limitate. I secondi sono sistemi che rispettano le schedule temporali in media o in una buona maggioranza dei casi.

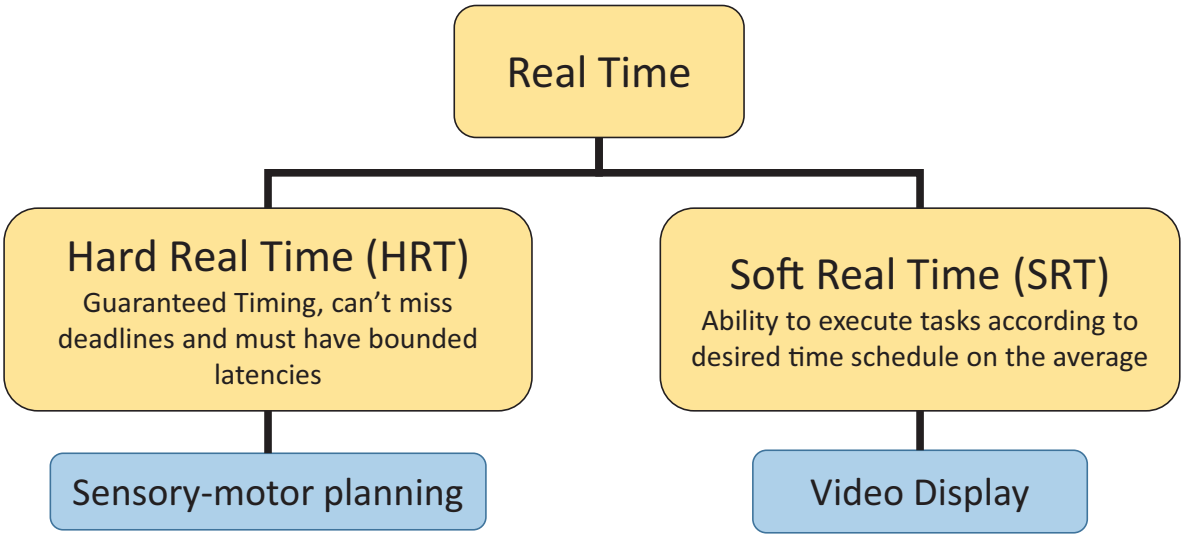


Figura 8.7 - Tipi di sistemi Real Time

8.3 *I Task*

Definiamo un Task come una sequenza di istruzioni che vengono eseguite con continuità, in assenza di altre attività, dal sistema fino al completamento.

Figura 8.8 - Schema di esecuzione di un Task Real Time

Nella Figura 8.8 lungo l'asse orizzontale è rappresentato il tempo e sono indicati:

– ai il segnale di richiesta di attivazione del task;
– si la partenza effettiva del task;
– fi la fine dell'esecuzione del task.

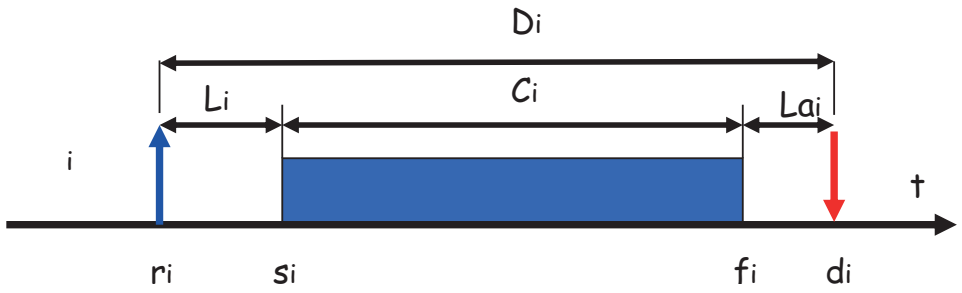


Figura 8.9 - Schema di esecuzione di Task Real Time

Nella descrizione di un task rappresentato in Figura 8.9 si possono definire le seguenti caratteristiche:

– ri Tempo di arrivo della richiesta di attivazione
– si Tempo di partenza del Task
– fi Tempo di Fine del Task
– di Tempo di Scadenza Assoluto
– Di Tempo di Scadenza Relativo
– Ci Caso peggiore di tempo di esecuzione
– Li Latenza
– Lai Lateness (fi - di)
– Ti Tardiness (max (0, Lai))

Ulteriori definizioni sono:
– Lateness è la qualità di esecuzione tradiva o che eccede la scadenza
– Tardiness è la qualità di non aderenza alla temporizzazione corretta o usuale
– Latenza è il tempo che intercorre fra la richiesta e la risposta
– Jitter è una variazione casuale nella temporizzazione di un segnale, per esempio specificatamente di un clock (orologio)

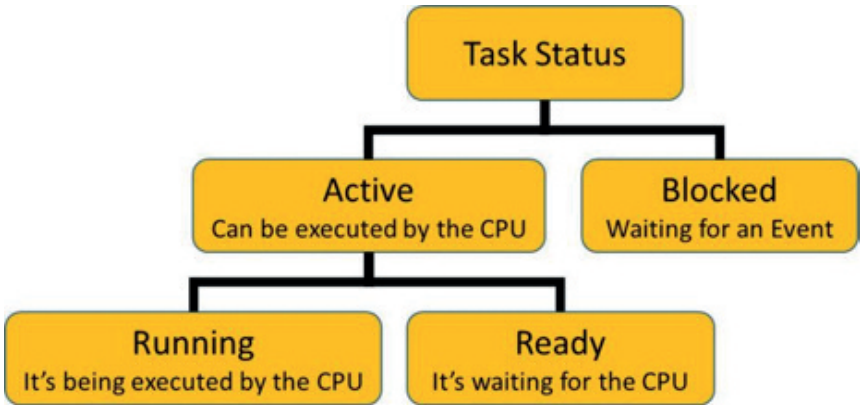


Figura 8.10 - Possibili Stati di un Task

Un Task può avere vari stati all'interno di un sistema e in particolare può essere:
– Active – se è pronto per poter girare su un processore (CPU)
– Blocked – se non è in grado di procedere perché magari è in attesa di un evento

A sua volta un Task che sia Active può trovarsi in una delle due condizioni:
– Running – se sta eseguendo sul processore
– Ready – se sta aspettando di accedere al Processore per andare in esecuzione

I Task che sono nella condizione di Ready sono normalmente gestiti dal sistema attraverso una coda (Ready Queue). Tale Ready Queue non è una FIFO (First In First Out) ma generalmente opera la selezione del Task da mandare in esecuzione mediante specifici criteri che vanno sotto il nome di Algoritmi di Schedulazione (Schedule Algorithms).

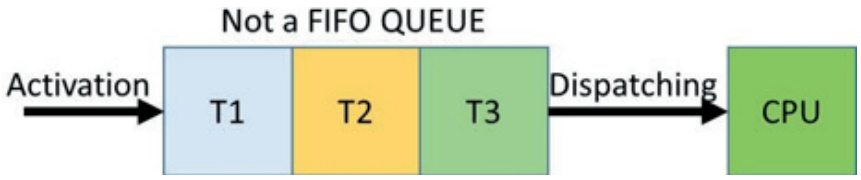


Figura 8.11 - Ready Queue per i Task

I passaggi da uno stato all'altro del Task avvengono secondo precise modalità ed a fronte di specifiche situazioni. Come mostrato in Figura 8.12, un Task viene attivato (Activation) ed è già pronto in stato di Ready e può essere schedulato per eseguire sulla CPU facendo dunque una transizione (Dispatching) in Running dove al termine dell'esecuzione viene terminato (Finished). Mentre è in Running, può essere in attesa di un evento e transire (Wait) verso lo stato di Blocked da cui uscirà solo in presenza del segnale atteso (Signal) per ritornare Ready.

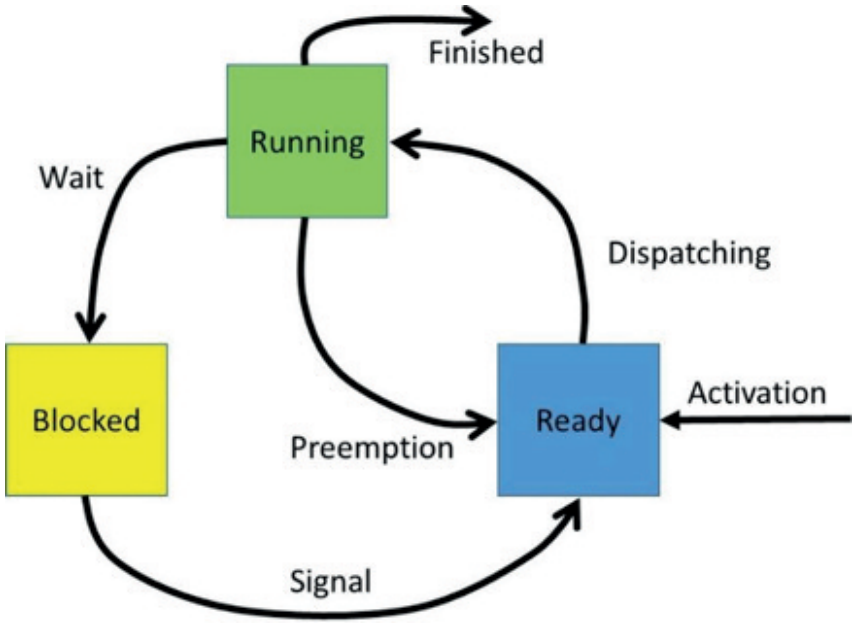


Figura 8.12 - Transizioni di Stato del Task

Molti sistemi prevedono, per ottimizzare l'uso della CPU un meccanismo di svuotamento (Preemption) del processore dal task che sta eseguendo in condizioni particolari (es. un altro Task più prioritario). Nel caso che questo avvenga il Task effettua tramite la Preemption una transizione verso lo stato di Ready e viene rimesso nella Ready Queue per essere poi rischedulato sulla CPU in seguito.
Da un punto di vista del comportamento temporale i Task possono anche essere suddivisi in Periodici e Aperiodici.

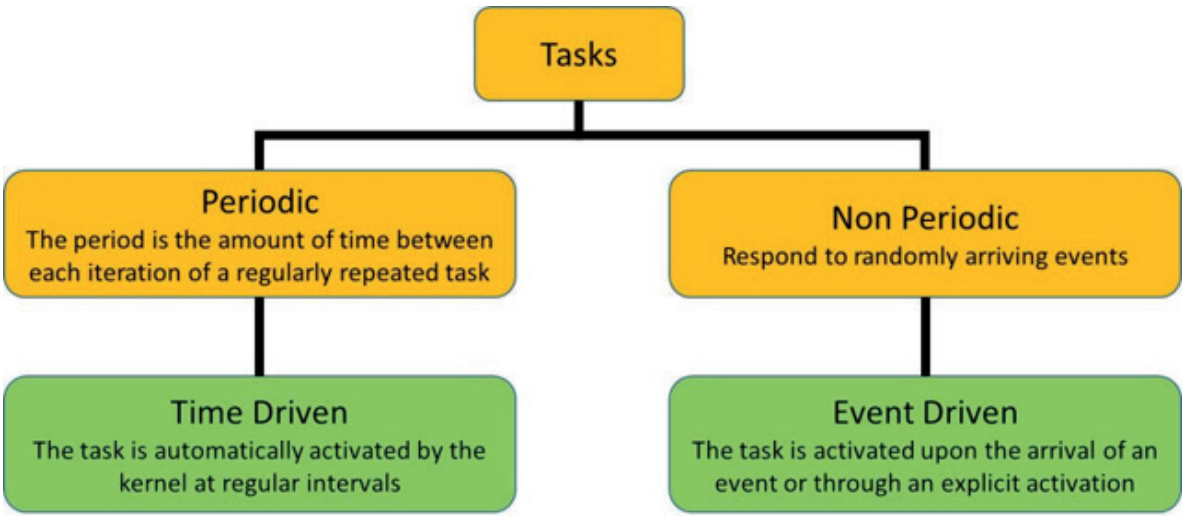


Figura 8.13 - Task periodici e aperiodici

I primi si ripresentano a distanza di intervalli predefiniti ed il periodo è il tempo che intercorre fra le iterazioni dello stesso Task che si ripete regolarmente. Tali Task sono dunque attivati automaticamente dal Kernel del Sistema ad intervalli regolari.
I Task Aperiodici si attivano in conseguenza di eventi che arrivano in maniera casuale oppure attraverso richieste esplicite di attivazione.
I Task Periodici sono tipici di una Acquisizione basata su meccanismi di Polling (vedi Par. 7.8.2), mentre i Task Aperiodici sono tipici di Acquisizione basata su Interrupt (vedi Par. 7.8.1). I Task Aperiodici possono essere trasformati in Periodici convertendo un Interrupt Handler in un Task di Polling con frequenza periodica opportuna. In tal caso il Task, invece di reagire ad uno stimolo esterno nel momento in cui questo avviene, va ad interrogare il dispositivo di Input regolarmente e nel caso che l'evento atteso sia presente, si procede di conseguenza.
Sotto il profilo del Real Time i Task possono essere catalogati come Hard Real Time o Soft Real Time. I Task HRT devono rispettare strettamente le scadenze temporali ed il mancato rispetto può avere effetti catastrofici sul sistema che viene controllato.

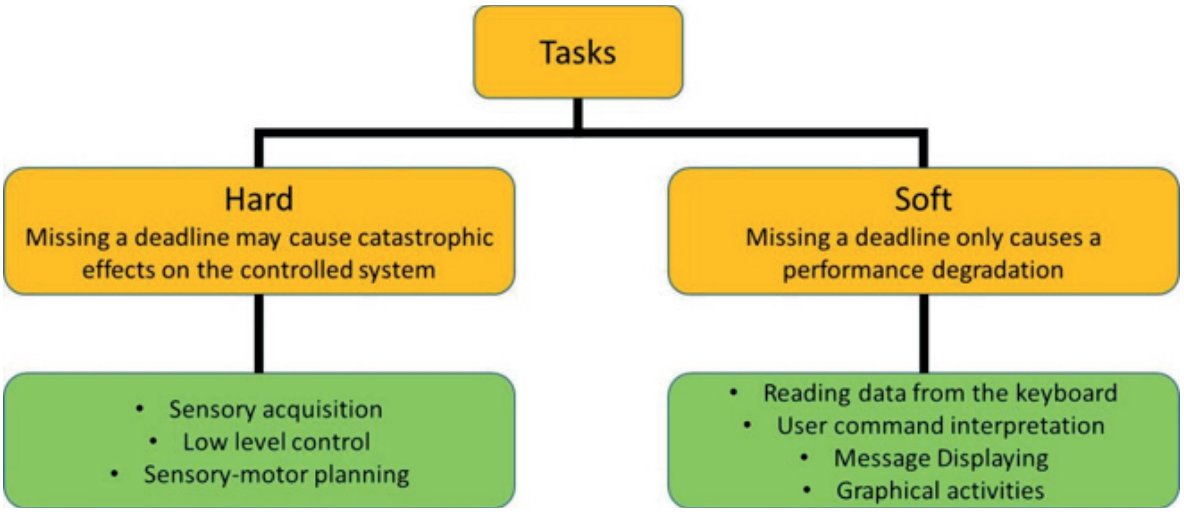


Figura 8.14 - Task Hard e Soft Real Time

Nel caso dei Task SRT le scadenze sono rispettate nella maggioranza dei casi ma può accadere che vengano superate ma tali situazioni non hanno effetti catastrofici sul sistema controllato che percepisce un degrado delle prestazioni.

8.4 *Schedulazione ed Algoritmi di Schedula*

Nel campo dei sistemi informatici, una schedula è un modo particolare di assegnazione dei Task al processore. Come abbiamo visto i Task pronti ad eseguire sono gestiti in una coda (Ready Queue) e il loro prelievo da tale coda per l'invio in esecuzione sulla CPU comporta una decisione che può essere presa secondo criteri diversi che vanno sotto il nome di Algoritmi di Schedula (Schedule Algorithm).

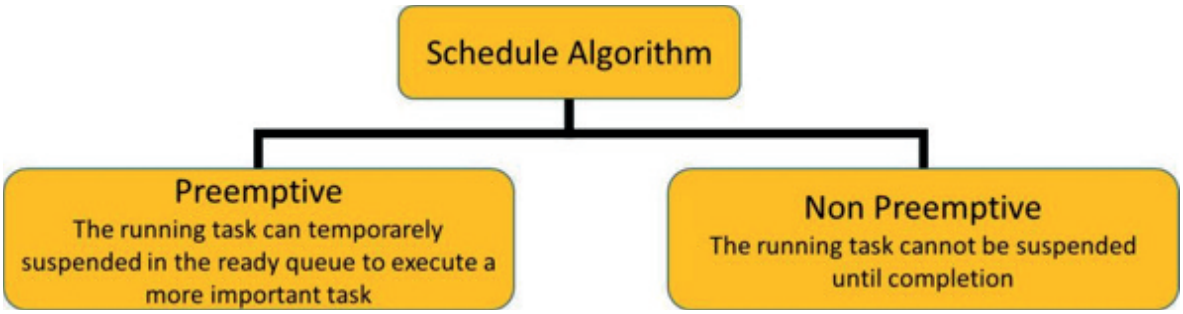


Figura 8.15 - Algoritmi di Schedulazione

Le strategie di tali algoritmi dipendono anche dal tipo di sistema che è a disposizione ed in particolare se è previsto un meccanismo di Preemption oppure no. Nel caso in cui questo sia presente, infatti, l'algoritmo può prendere in considerazione la possibilità di sospendere dei Task e rimetterli nella Ready Queue per mandare in esecuzione sul processore un Task più importante o prioritario. In mancanza di un meccanismo di Preemption, i Task non possono essere interrotti finché non hanno terminato.

Nella figura seguente è mostrato un esempio di schedula Preemptive. L'asse orizzontale è quello dei tempi ed in verticale ci sono tre differenti Task che hanno evidentemente priorità differenti, per cui il Task T1 inizia al tempo t1 finché all'istante t2 non viene attivato il secondo Task T2 che è più importante o prioritario e pertanto il primo Task T1 viene sospeso e T2 esegue fino a che all'istante t3 non diventa attivo il terzo Task T3 che ha una importanza o priorità ancora maggiore e questo provoca l'interruzione del secondo Task T2 per mandare in esecuzione T3.

Quando all'istante t4 il terzo Task T3 ha terminato, viene riesumato il Task T2 che ha la priorità più alta e questo esegue il suo compito fino a terminare all'istante t5, quando è finalmente rischedulato sul processore il Task T1 che è in grado di terminare il proprio compito.

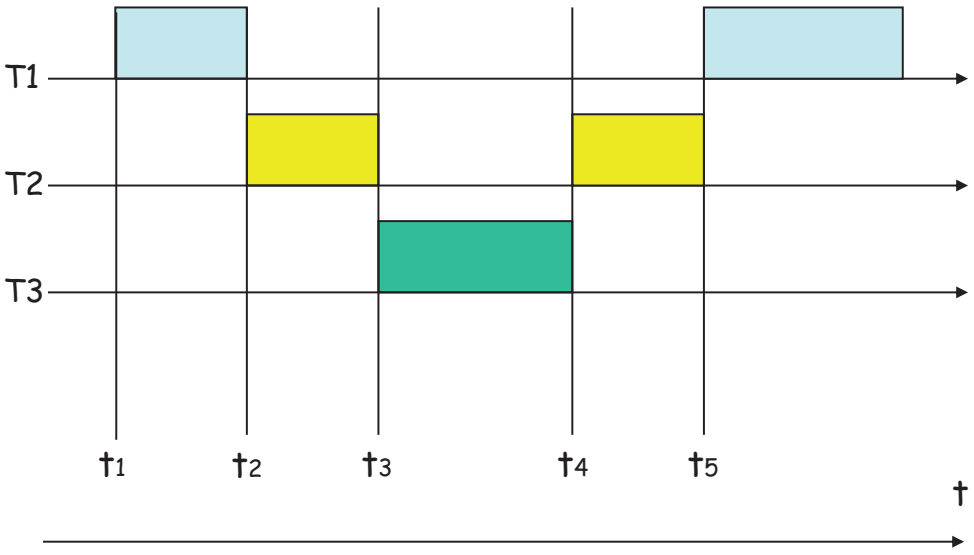


Figura 8.16 - Esempio di schedula Preemptive

Gli Algoritmi di Schedula devono anche tener conto di condizioni e limitazioni che limitano le possibilità. In particolare vi sono:

– Limitazioni Temporali come i tempi di attivazione, quelli di completamento ed il Jitter relativo alle tempistiche;
– Condizioni di precedenza che si possono instaurare e che impongono un preciso ordine di esecuzione dei Task;
– Limitazioni delle risorse che possono essere accedute od utilizzate da un solo Task per volta e richiedono un accesso mutuamente esclusivo.

8.4.1 Algoritmi di Schedula Non Real Time

Alcuni algoritmi non hanno caratteristiche adeguate per essere Real Time ed alcuni esempi sono:

– First Come First Served (FCFS)
– Shortest Job First (SJF)
– Priority Scheduling
– Round Robin

Li esaminiamo rapidamente a scopo didattico.

8.4.1.1 First Come First Served (FCFS)

L'algoritmo FCFS assegna la CPU ai Task basandosi sul tempo di arrivo della richiesta di attivazione, i Task che arrivano prima sono quelli che per primi accedono al processore.
Come mostrato nellaFigura 8.17, i due esempi di tempi di arrivo (r1, r2, r3) delle richieste di 3 Task (rappresentati sull'asse orizzontale dei tempi), in un caso sono tali da far partire nell'ordine T1, T2 e T3, nel secondo caso con tempi di arrivo delle richieste ribaltati, anche l'ordine di esecuzione è invertito. L'algoritmo FCFS ha dunque come difetto quello di avere una schedula altamente impredicibile perché dipendente dai tempi di arrivo casuale dei Task che rende anche i tempi di risposta fortemente dipendenti da essi.

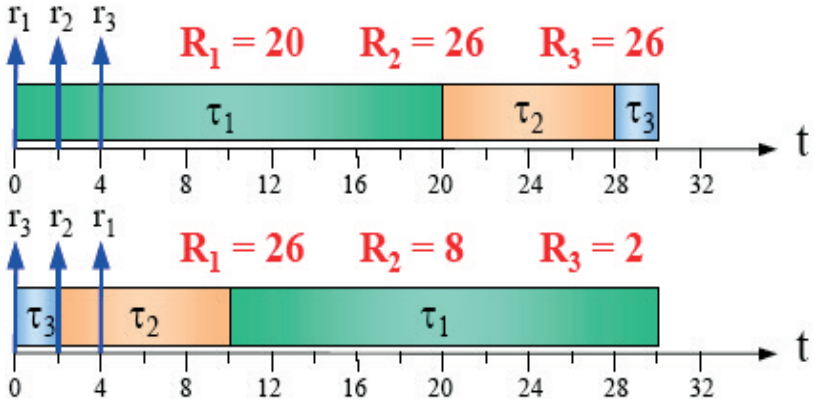


Figura 8.17 - Esempi di Scheduling FCFS

8.4.1.2 Shortest Job First (SJF)

Nel caso di scheduling Shortest Job First, viene selezionato per l'esecuzione sulla CPU il Task che ha la minore durata rispetto agli altri.

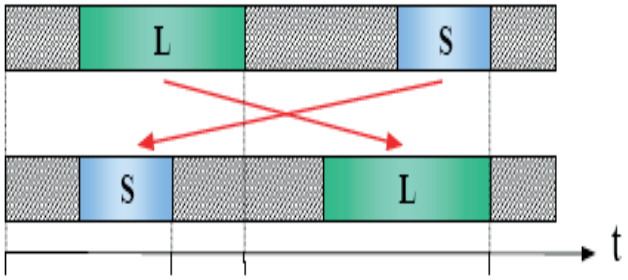


Figura 8.18 - Esempio di Scheduling SJF

Nell'esempio sopra, il Task L più lungo come esecuzione ma arrivato prima viene scambiato di ordine con il Task S, più corto come esecuzione. Anche questo algoritmo non è in grado di garantire tempi di esecuzione certi per un ambiente Real Time in quanto, non è scontato a priori che il Task più corto sia quello da eseguire per primo per rispettare specifiche scadenze temporali.

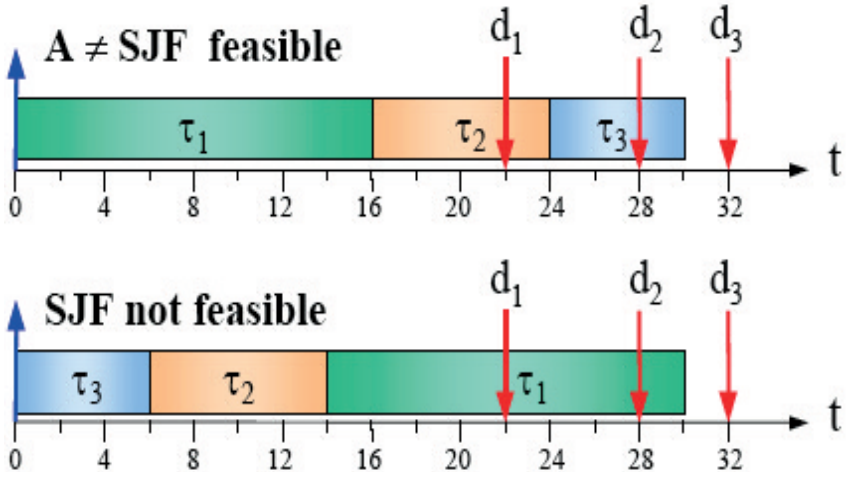


Figura 8.19 - Fattibilità dell'algoritmo SJF

Come si può vedere in Figura 8.19, il rispetto delle scadenze temporali (deadline d1, d2, d3) dei tre Task mostrati, possono essere soddisfatte nel primo caso in cui non viene applicato un algoritmo SJF e viceversa non esserlo nel secondo caso in cui viene usato uno scheduling SJF.

8.4.1.3 Priority Scheduling

Nel Priority Scheduling si opera con meccanismi prioritari basti su un valore di priorità (es. [0, 255]) assegnato a ciascun Task ed il Task con valore più alto di priorità viene selezionato per l'esecuzione sul processore. I Task con identica priorità sono gestiti con un meccanismo FCFS (First Come First Served).

Il problema principale di questo meccanismo di scheduling è che, in sistemi che prevedono il meccanismo di preemption, i Task a più bassa priorità possono sperimentare attese enormemente lunghe per il loro completamento perché vengono sistematicamente messi da parte a favore di Task a più alta priorità. Questo problema che va sotto il nome di "Starvation" a volte viene mitigato da una correzione all'algoritmo che preveda una crescita dinamica della Priorità del Task in relazione al suo tempo di permanenza nella Ready Queue.

8.4.1.4 Round Robin

Nell'Algoritmo Round Robin (giostra) i Task nella coda di esecuzione sono gestiti con un meccanismo FCFS (First Come First Served).

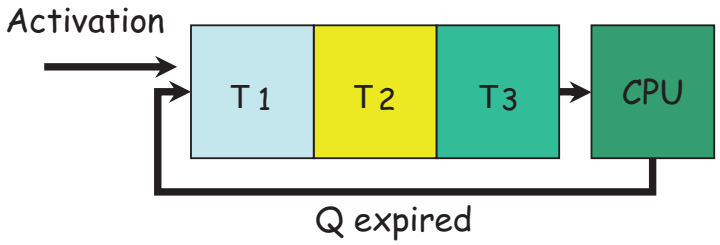


Figura 8.20 - Esempio di coda Round Robin

Tuttavia si aggiunge ad esso una condizione per cui ciascun Task non può occupare il processore per più di un certo intervallo di tempo (Quantum) e quando tale Quantum finisce, il Task viene spostato (preemptive) di nuovo alla fine della coda di esecuzione per essere schedulato più tardi quando anche gli altri Task nella coda avranno ottenuto il loro Quantum sulla CPU. In questa maniera ciascun Task ottiene una frazione N del tempo del processore ed è come se eseguisse su una CPU che è un fattore N più lenta di quella effettivamente disponibile. L'algoritmo Round Robin è quello più usato su sistemi operativi multiutenti che lavorano secondo il principio di dare un po' di tempo di processore a tutti (Time Sharing).

8.4.2 Algoritmi Real Time

Negli algoritmi di schedula di tipo Real Time sono generalmente usate due tipologie di scadenze temporali (Deadlines): la deadline Assoluta che è il valore del tempo assoluto che individua la deadline e quella Relativa che è sostanzialmente l'intervallo che intercorre fra l'Attivazione e la deadline Assoluta. La prima è dunque tipicamente dinamica e la seconda è invece un parametro statico se il comportamento del task è sempre lo stesso.

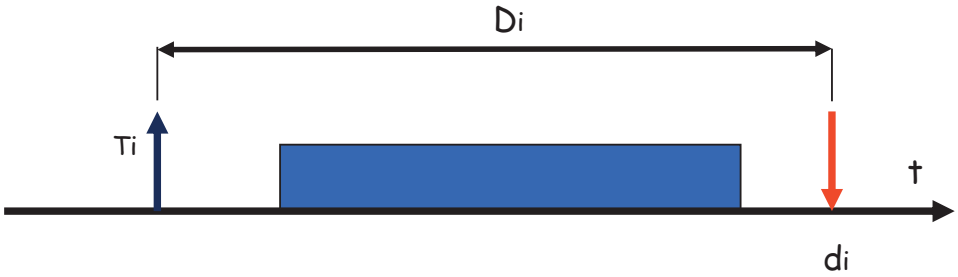


Figura 8.21 - Deadline Assoluta e Relativa

Esamineremo ora alcuni algoritmi che hanno caratteristiche per essere considerati Real Time.

8.4.2.1 Earliest Due Date (EDD)

L'algoritmo EDD seleziona per l'esecuzione il Task che ha la Deadline Relativa più vicina in termini temporali. È possibile utilizzarlo quando i Task arrivano tutti simultaneamente e viene implementato utilizzando per ogni Task una priorità fissata, dato che le Deadline Relative dei Task sono note a priori. In questo algoritmo la modalità di preemption del processore non è richiesta. Lo EDD minimizza la maximum lateness (Lmax vedi Par. 1.3).

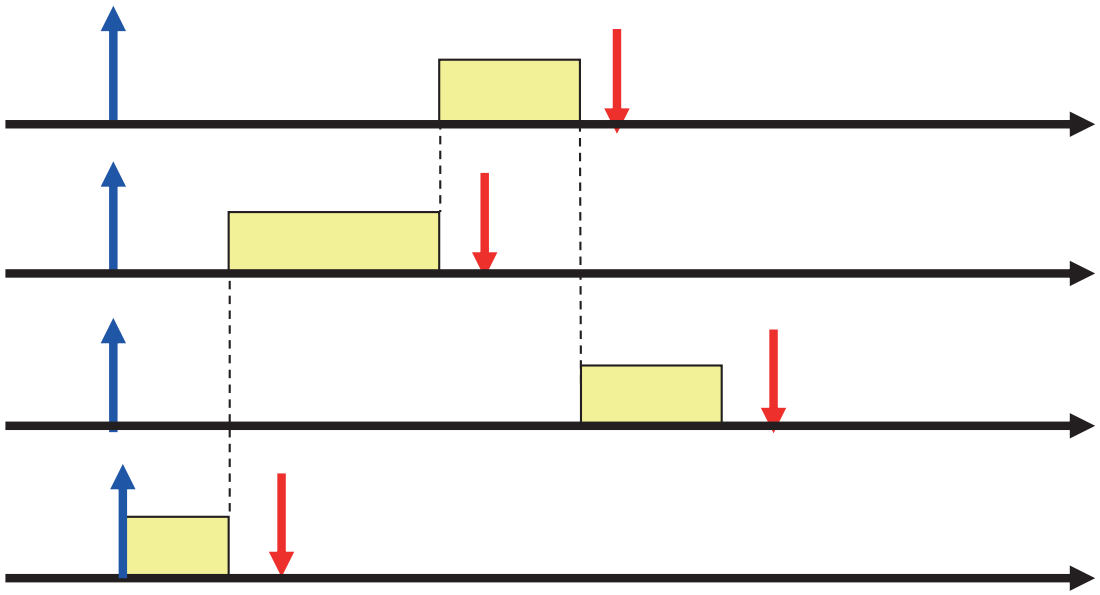


Figura 8.22 – Esempio di schedula Earliest Due Date (EDD)

Nella Figura 8.22 è mostrato un esempio di schedula con algoritmo EDD nel quale le frecce blu rappresentano i tempi (coincidenti) di richiesta di attivazione dei Task e si può notare come i Task siano portati in esecuzione con un ordine che segue quello delle scadenze temporali rappresentate dalle frecce rosse.

8.4.2.2 Earliest Deadline First (EDF)

Questo meccanismo di scheduling seleziona per l'esecuzione i Task che hanno la Deadline Assoluta più vicina in termini temporali. In questo caso le richieste di attivazione dei Task possono arrivare in maniera casuale in un qualunque momento.
La priorità dei Task deve necessariamente essere assegnata in maniera dinamica perché legata al tempo di arrivo delle richieste. Questo comporta la necessità di avere a disposizione un meccanismo di preemption dei Task.
L'algoritmo EDF minimizza la maximum lateness (Lmax vedi Par. 8.3).

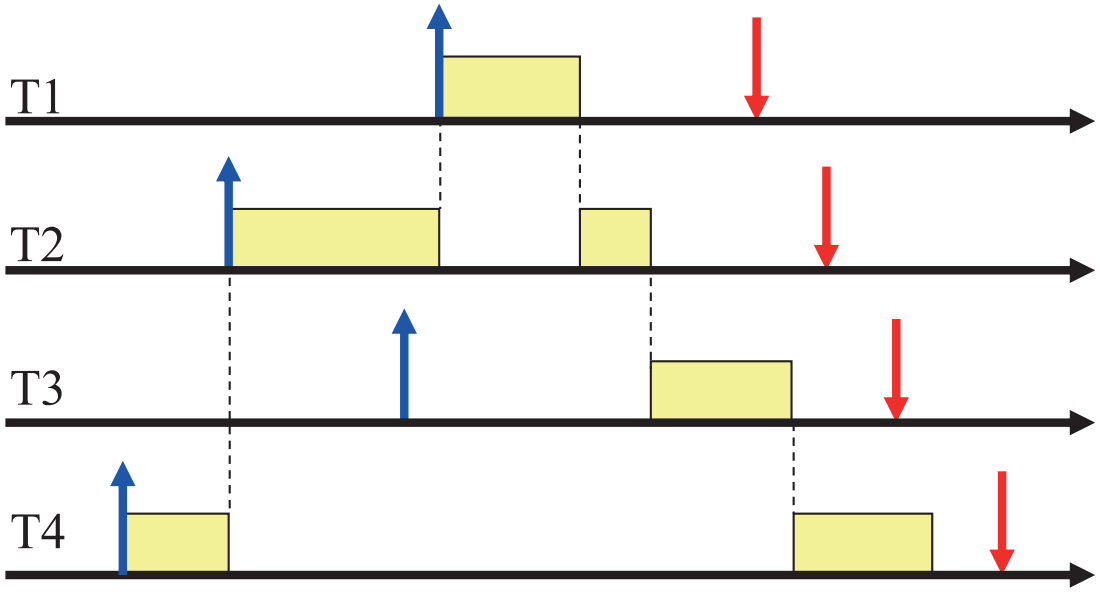


Figura 8.23 - Esempio di Algoritmo Earliest Deadline First (EDF)

Nella figura sopra viene mostrato un esempio di algoritmo EDF in cui i tempi di arrivo delle richieste di attivazione dei Task sono mostrate dalle frecce blu e si può vedere come il primo Task a partire sia quello più in basso (T4) perché la sua attivazione è antecedente alle altre. Successivamente parte il Task nella seconda riga dall'alto (T2) e, poiché la sua Deadline Assoluta rappresentata dalla freccia rossa è precedente in ordine temporale a quella di T4, tale Task viene sospeso per far eseguire il nuovo Task. Per lo stesso motivo, anche il Task T2 viene successivamente interrotto all'arrivo del Task in prima riga (T1) che ha la Deadline Assoluta più prossima. T2 sarà ripreso al termine di T1 e quindi sarà la volta del Task sulla terza riga (T3) dato che la sua Deadline Assoluta è la terza in tempo dopo quella di T1 e T2. Al termine di T3 finalmente T4 potrà completare la sua esecuzione.

8.4.2.3 Periodic Task e Cyclic Scheduling

I Task periodici sono una tipologia che risponde ad una schedulazione ricorsiva basata su periodi definiti di tempo. Nei Task periodici, ogni gruppo di task deve partire dopo un certo periodo predefinito e deve terminare prima che il prossimo periodo sopraggiunga. Per questi Task si usa quindi un tipo di schedula ciclica in cui l'asse del tempo è suddiviso in intervalli di uguale durata (Time Slot) ed ogni Task è allocato in maniera statica in una delle slot per fare in modo che venga rispettata la frequenza di ripetitività desiderata. L'esecuzione dei Task in ogni slot è attivata per mezzo di timer ed è quindi programmata in anticipo. Nella Figura 8.24 è mostrato un esempio di schedulazione ciclica in cui:

– Il Task A ha una frequenza di ripetizione f = 40 Hz e, conseguentemente si ripete ogni periodo t = 25ms;
– Il Task B ha una frequenza di ripetizione f=20 Hz e si ripete con un periodo t = 50 ms;
– Il Task C ha una frequenza di ripetizione f=10 Hz ed un periodo t =100 ms.

Perché tutto funzioni ovviamente la durata di ciascun Task non deve eccedere un certo tempo. Si possono individuare un ciclo minore indicato con Λ e uno maggiore indicato con Γ e si può notare come il funzionamento della schedulazione si basi sulla garanzia che $C_A + C_B \leq \Lambda$ e anche $C_A + C_C \leq \Lambda$ dove C_A è la durata del Task A, C_B la durata del Task B e C_C la durata del Task C.

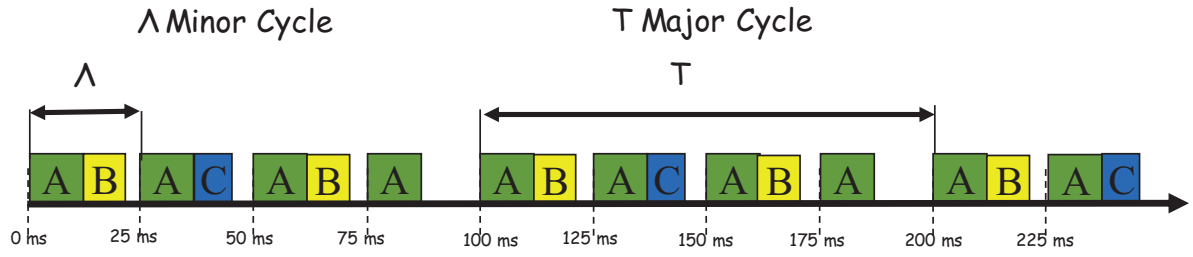


Figura 8.24 - Esempio di schedulazione ciclica

I vantaggi di una schedulazione ciclica possono essere sintetizzati in:

– Semplicità di realizzazione non essendo necessario un Sistema Operativo Real Time;
– Basso aggravio nell'esecuzione dei Task che possono partire senza particolari algoritmi decisionali;
– Controllo del jitter grazie all'esecuzione regolata da un timer

Naturalmente ci sono anche degli svantaggi come, ad esempio:
– Non è un sistema robusto durante i sovraccarichi e non è possibile o auspicabile che i Task durino più del previsto o che si sovrappongano temporalmente;
– È difficile espandere la schedula se non per casi in cui le finestre rimaste libere sono adeguate per uno o più nuovi Tasks;
– Non è di facile realizzazione la gestione di attività aperiodiche in maniera adeguata.

8.4.3 Altri algoritmi di schedula

Vi sono altri algoritmi di schedula possibili ed in particolare possiamo citarne due:
– Schedula basata su Priorità (Priority Scheduling) ma in cui ad ogni Task è assegnata una priorità basata sulle sue scadenze e limitazioni temporali e la fattibilità della schedula è verificata con tecniche analitiche. I Task sono eseguiti su un kernel che gestisca le priorità.
– Rate Monotonic (RM) nel quale ad ogni Task è assegnata una priorità fissata in maniera proporzionale alla sua frequenza di ripetitività.

8.5 *Real Time Operating System (RTOS)*

I Sistemi Operativi dei normali computer non hanno generalmente la possibilità di operare in Real Time. Perché questo avvenga, in generale, i Sistemi Operativi devono essere disegnati per svolgere compiti di Real Time in maniera adeguata. Sono stati creati vari tipi di tali sistemi ed alcuni di essi sono anche di tipo commerciale e sono usati in applicazioni di carattere industriale. Esempi dei più diffusi di tali sistemi sono:
- OS9 (Microware LP)
- Deos (DDC-I)
- embOS (SEGGER)
- FreeRTOS (Amazon)
- Integrity (Green Hills Software)
- Keil RTX (ARM)
- LynxOS (Lynx Software Technologies)
- MQX (Philips NXP / Freescale)
- Nucleus (Mentor Graphics)
- Neutrino (BlackBerry)
- PikeOS (Sysgo)
- SafeRTOS (Wittenstein)
- ThreadX (Microsoft Express Logic)
- μC/OS (Micrium)
- VxWorks (Wind River)
- Zephyr (Linux Foundation)

Nei Sistemi Operativi Real Time (RTOS) il kernel del sistema fornisce uno strato intermedio di astrazione che nasconde i dettagli dell'hardware del processore (o dei processori) al software applicativo che verrà eseguito.

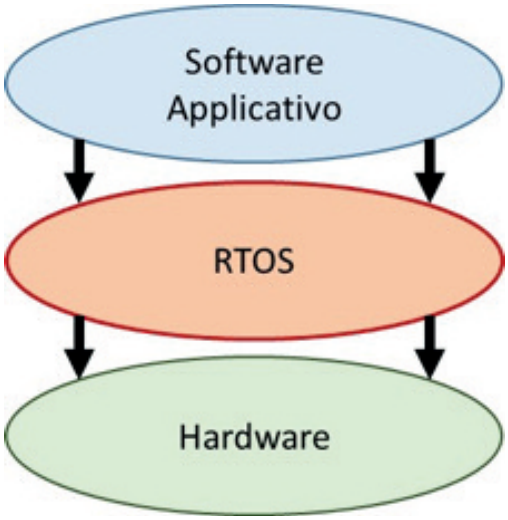


Figura 8.25 - RTOS

Nel fornire questo strato di astrazione il kernel RTOS provvede a fornire al software applicativo cinque categorie principali di servizi di base, così come mostrato in Figura 8.26.

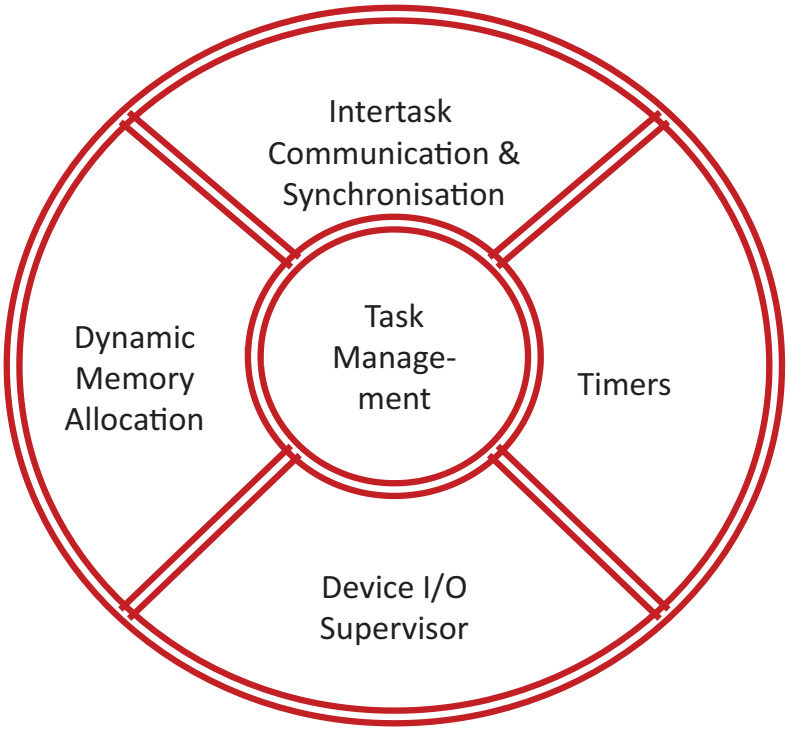


Figura 8.26 - Servizi in un sistema operativo Real Time

Tali servizi sono descritti nei paragrafi seguenti.

8.5.1 Task Management

La categoria più basilare dei servizi del kernel è il Task Management. Questo insieme di servizi permette agli sviluppatori del software applicativo di disegnare il loro software come un insieme di parti di software che gestiscano una specifica tipologia di attività o obiettivo e, magari, una propria scadenza di tipo Real Time.
Ogni parte di software viene chiamato Task e i servizi relativi a questa categoria includono la ca-

pacità di lanciare tali Task e di assegnare delle priorità a ciascuno di essi. Tuttavia il servizio principale in questa categoria riguarda la schedulazione dei Task come una parte inclusa all'interno del Sistema Operativo. Il Task Scheduler, quindi controlla l'esecuzione dei Task del software applicativo in maniera tale che eseguano il loro compito con celerità e responsività.
In un sistema operativo che permetta la Preemption e la possibilità che il processore sia in grado di passare da un Task all'altro, diventa fondamentale avere sotto controllo tali tempi di switch fra un Task e l'altro. Il cosiddetto Task Switching Time è il tempo che viene speso per passare dall'esecuzione di un Task a quella di un altro Task.

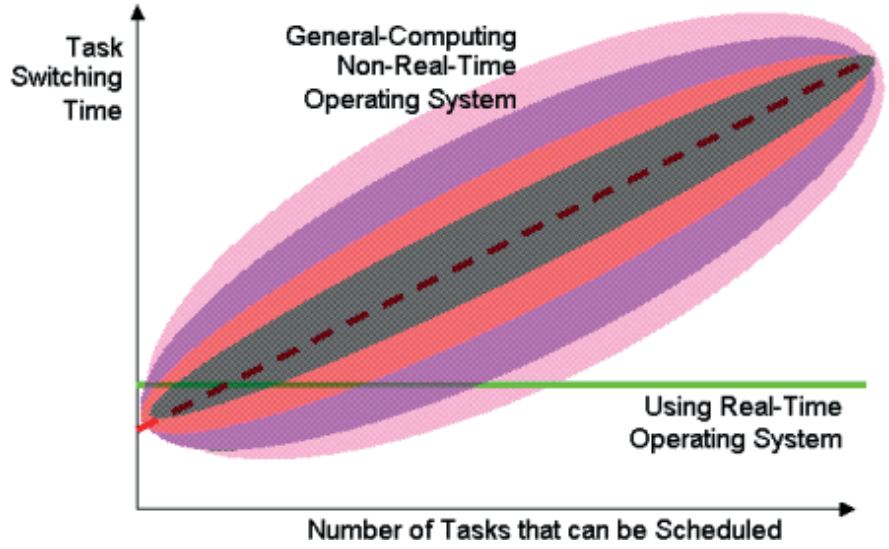


Figura 8.27 - Tempi di Switch fra Task in un Sistema Operativo

Il Task Switching Time, nei sistemi operativi generici (non Real Time) può crescere all'aumentare del numero di Task da schedulare nel sistema, mentre un sistema operativo Real Time deve essere in grado di sopportare un sovraccarico di Task mantenendo costante lo switching time.

8.5.2 Intertask Communication

La seconda categoria di servizi del kernel riguarda la comunicazione e sincronizzazione fra i Task (Intertask Communication and Synchronization).
Questa categoria di servizi permette ai Tasks di comunicare fra loro passandosi informazioni evitando il pericolo che tali informazioni possano essere danneggiate.
Attraverso questi servizi è anche possibile per i Task di coordinarsi fra loro così che sia possibile una produttiva cooperazione. In mancanza di questa classe di servizi, i Tasks potrebbero comunicare fra loro con informazioni corrotte oppure interferire fra loro.

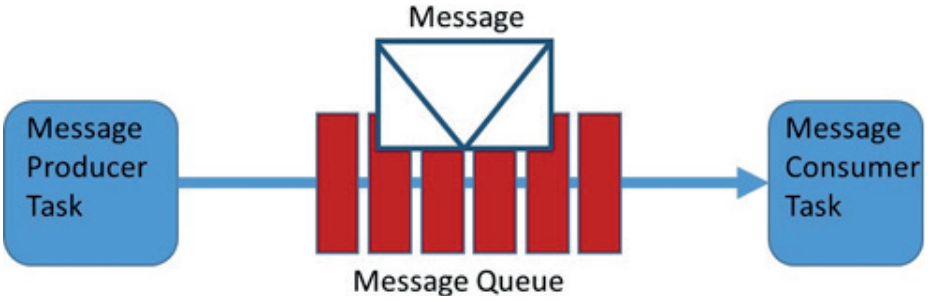


Figura 8.28 - Message System

Nella Figura 8.28 è mostrato un esempio di sistema di messaggi (Message System) fra Task nel quale vengono scambiati messaggi utilizzando i servizi di messaggistica del Sistema Operativo. Lo scambio di messaggi fra Task è una forma di comunicazione che costituisce un'area di sevizi specifici che dipendono dal sistema operativo e che possono mostrare caratteristiche temporali che sono differenti fra sistemi operativi differenti. La maggior parte dei sistemi, infatti, effettua due copie dello stesso messaggio perché lo trasferiscono fra un Task e l'altro attraverso una coda di messaggi (Message Queue). La prima operazione di copia avviene fra la memoria di un Task che manda il messaggio (Sender) a una particolare area di memoria del sistema operativo che è usata come coda di messaggi (Message Queue), mentre la seconda operazione di copia avviene fra la memoria di sistema della Message Queue nell'aerea di memoria del Task che deve ricevere il messaggio (Receiver).
Questo procedimento è dunque non deterministico nella sua tempistica poiché queste attività di copia possono avere tempi lunghi e variabili che dipendono dalla lunghezza del messaggio. Un approccio che permette di eliminare questo comportamento non-deterministico e che consente di accelerare le prestazioni del servizio di messaggi è quello in cui il Sistema Operativo, invece di copiare l'intero messaggio, copia solo il puntatore (indirizzo di memoria) al messaggio e consegna al Task ricevente tale puntatore senza muovere per nulla il contenuto del messaggio.

8.5.3 Timer Service

Molti sistemi integrati (embedded systems) hanno delle necessità stringenti sulle temporizzazioni e la maggior parte dei kernel RTOS forniscono alcuni Timer Service basilari come ritardi (delays), e time-out (tempo scaduto). La ragione di queste necessità è che spesso sia i Task di Sistema che quelli utente necessitano di schedulare ed effettuare attività secondo certi intervalli temporali. Alcuni esempi possono essere:

– uno scheduler che può aver bisogno di operare lo switch fra un Task e l'altro dopo un certo intervallo di tempo;
– un protocollo di comunicazione che deve stabilire quando effettuare ritrasmissioni se non riceve una risposta entro un certo tempo;
– Polling di dispositivi secondo specifiche tempistiche, ecc.

In molti casi, dunque applicazioni richiedono di schedulare eventi futuri e per farlo hanno bisogno di servizi di timer ed i Timer sono una parte integrante di molti sistemi Real Time. Un Timer schedula un evento nel futuro secondo un valore di tempo predefinito come, ad esempio, attivare una sveglia.
Ci possono essere due tipi di Timer: Hard Timer e Soft Timer. Gli Hard Timer usano dei Timer fisici realizzati in hardware attraverso dei chips o circuiti che a seguito di Time-out vanno a generare degli Interrupt direttamente sul Processore e sono indicati per quelle applicazioni in cui viene richiesta una precisione sul Timer o sulla Latenza in maniera tale da avere delle prestazioni del Timer altamente predicibili.
I Soft Timer sono eventi Software che sono schedulati attraverso un dispositivo software apposito e permettono una schedulazione efficiente per eventi che non richiedono alta precisione.
Alcune applicazioni possono richiedere dei Timer di alta precisione, ma molte applicazioni, anche importanti, possono tranquillamente lavorare con Timer con risoluzioni di millisecondi. Un'altra ragione per usare dei Soft Timer è quella di ridurre un sovraccarico di Interrupt di sistema sul Processore se non strettamente necessario.

8.5.4 Dynamic Memory

Molti, ma non tutti i kernel dei sitemi RTOS forniscono dei servizi per l'allocazione dinamica della memoria (Dynamic Memory Allocation) e questi sevizi permettono ai Task di ottenere dal

Sistema Operativo dei banchi di memoria da usare temporaneamente. Spesso tali banchi o Buffer di memoria vengono passati da un Task ad un altro come strumento rapido di comunicazione per grandi quantità di dati.
Tuttavia alcuni kernel RTOS molto compatti che sono destinati a sistemi con memoria limitata, non offrono servizi di allocazione dinamica della memoria.
Il determinismo dei tempi dei servizi è una problematica anche nell'area dell'allocazione dinamica della memoria RAM (Random Access Memory). Molti computer generici con sistemi operativi non Real Time, offrono servizi di allocazione dinamica della memoria prelevandola da quello che viene chiamato "Heap".
Le funzioni "malloc" e "free" che sono utilizzabili all'interno di programmi scritti in linguaggio C lavorano sul Heap per allocare e rispettivamente liberare i banchi di memoria dinamica. I Tasks possono richiedere ed ottenere dei banchi di memoria dallo Heap usando la funzione "malloc" e specificando la grandezza del buffer di memoria richiesta. Quando il Task ha terminato di utilizzare il buffer di memoria la può restituire al sistema operativo usando la funzione "free" ed il sistema operativo a sua volta restituirà il buffer di memoria allo Heap dove il banco di memoria potrà essere riutilizzato per altri Task, magari come parte di un buffer più grande, oppure essere suddiviso in buffer più piccoli.

8.5.4.1 Memory Fragmentation

La gestione della memoria tramite Heap soffre del fenomeno della frammentazione della memoria (Memory Fragmentation) che può causare un degrado del servizio. La frammentazione è causata dal fatto che quando il banco di memoria viene restituito allo Heap, può successivamente essere suddiviso in buffer più piccoli quando arrivino richieste tramite "malloc" di banchi di tipo più ridotto. Dopo che si siano avvicendati molti cicli di "malloc" e "free", può accadere che si formino delle piccole aree o fettine di memoria inutilizzate fra i banchi di memoria che sono utilizzati dai Task. Queste aree sono così minuscole che diventano inutilizzabili per i Task e costituiscono uno spreco di memoria che rimane intrappolata fra Buffer che sono usati dai Task e che non possono essere consolidate in un Buffer più grande che possa essere usato. Con il passar del tempo lo Heap è sempre più frammentato con un numero crescente di queste fettine di memoria inutilizzabili e tutto può sfociare in una situazione in cui i Task richiedono Buffer di memoria di una certa grandezza e, sebbene nel complesso la memoria dello Heap abbia le quantità richieste, queste però risultano frammentate e non contigue per cui il sistema operativo è costretto a rifiutare le richieste dei Task.

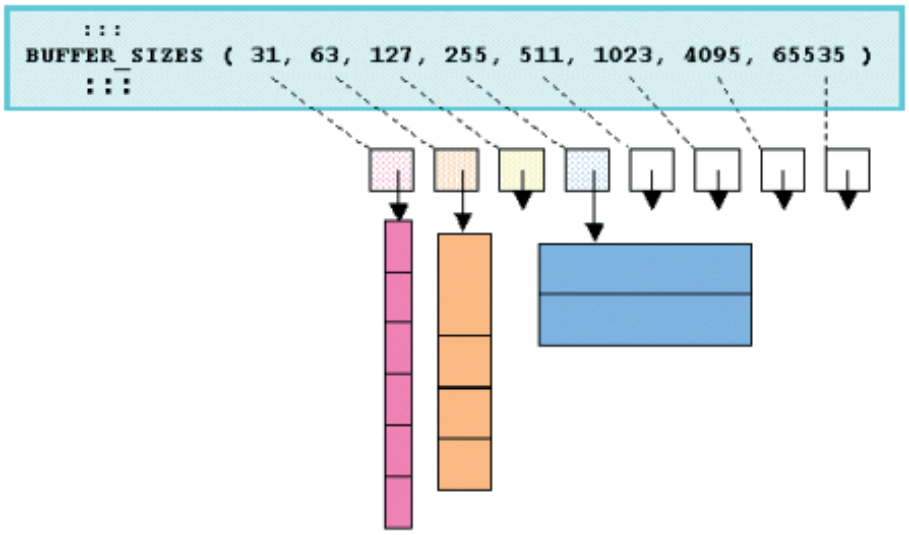


Figura 8.29 - Gestione dei Buffer di Memoria

Questo fenomeno è ciò che va sotto il nome di External Memory Fragmentation e la soluzione generalmente adottata è quella di effettuare la cosiddetta "Garbage Collection" e cioè un procedimento software di deframmentazione. Sfortunatamente gli algoritmi utilizzati per effettuare tale Garbage Collection sono spesso ampiamente non deterministici ed introducono ritardi con partenza e durata casuali nei servizi di Heap.
Tali problematiche sono spesso presenti nei servizi di allocazione dinamica della memoria di sistemi di calcolo con sistemi operativi non Real Time. I sistemi operativi Real Time, d'altra parte, risolvono questa problematica evitando sia la frammentazione della memoria che la Garbage Collection con le loro conseguenze. Gli RTOS offrono quindi delle tecniche di allocazione della memoria che evitano la frammentazione, invece di usare il meccanismo dello Heap. Per fare ciò essi limitano la varietà di grandezze dei banchi di memoria disponibili alle applicazioni. Nonostante questo approccio risulti meno flessibile rispetto a quello dello Heap, esso però permette di evitare il fenomeno della frammentazione della memoria ed evita la necessità di meccanismi di deframmentazione.
Nella Figura 8.29 è mostrato un esempio di un numero limitato di insiemi di buffer di memoria (Pool) e con tale meccanismo il software applicativo può chiedere l'allocazione di banchi di memoria di un numero limitato di possibili grandezze (es. 4 oppure 8 tipi). I Pool evitano totalmente la possibilità di una frammentazione della memoria facendo in modo che quando un Buffer viene ritornato esso rientri nel Pool di Buffer della stessa dimensione da cui è stato prelevato ed evitando che invece possa essere diviso in banchi di memoria più piccoli.

8.5.4.2 Shared Memory

In alcuni tipi di applicazioni è necessario condividere una grande quantità di dati in tempo reale e lo strumento più adatto per applicazioni all'interno dello stesso computer è la shared memory. In questa modalità due o più programmi fanno riferimento ad una stessa area di memoria che viene condivisa e, pertanto, qualcosa scritto in memoria da un programma, è immediatamente disponibile per un altro.
Nei sistemi Unix o Linux esistono delle funzioni che permettono di definire ed allocare tali aree di memoria da utilizzare in comune fra più task o programmi.

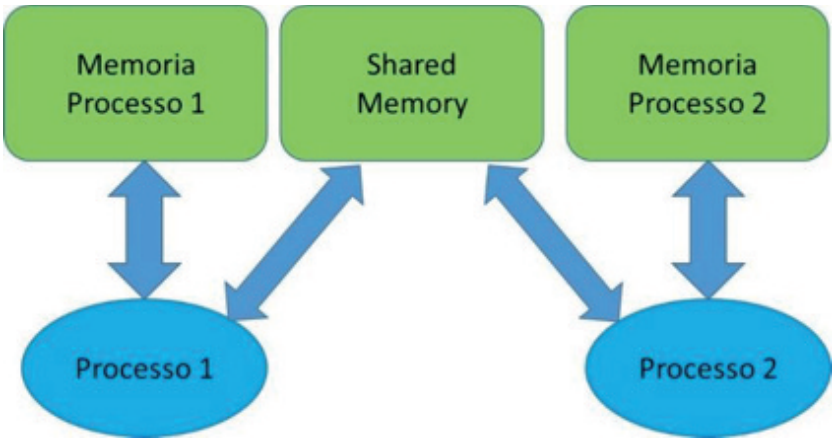


Figura 8.30 - Shared Memory

8.5.5 Device I/O Supervisor

Anche nel caso della categoria dei servizi di Supervisione dei Dispositivi di Input/Output (Device I/O Supervisor), molti ma non tutti i kernel RTOS ne sono provvisti. Questi servizi, quando disponibili, forniscono un meccanismo uniforme per organizzare ed accedere i vari dispositivi hardware che sono tipici di sistemi embedded o comunque di tipo RT.

8.5.6 Servizi addizionali

In aggiunta ai servizi di kernel finora descritti, molti RTOS offrono anche una varietà di altri servizi opzionali per funzionalità di alto livello come l'organizzazione di File System, la comunicazione via rete (Network) e la sua gestione, gestione di DataBase, interfacce grafiche per l'utente, ecc.. Molte di queste componenti software aggiuntive possono essere più complesse e corpose dello stesso kernel RTOS, ma generalmente si appoggiano su di esso e ne usano i servizi base. Le componenti aggiuntive in questione vengono aggiunte ai sistemi RTOS solo quando i loro servizi sono necessari per le applicazioni per contenere il consumo di memoria al minimo.

8.5.7 Hard RTOS

Un sistema Hard RTOS garantisce l'esecuzione di un task in un tempo prefissato e per raggiungere questo scopo, tutti i ritardi interni al sistema devono avere un limite garantito.
In un sistema Hard RTOS se un processo si chiude correttamente ma con un tempo più lungo di quello prefissato, il processo fallisce.
Esempio: Un robot che deve prelevare un pezzo da un nastro trasportatore. C'è una precisa finestra temporale entro cui il pezzo passa davanti a lui, se tarda a prenderlo il pezzo passa oltre ed il processo è fallito.

8.5.8 Soft RTOS

Un sistema Soft RTOS è meno restrittivo e garantisce soltanto una maggiore priorità ai task critici rispetto a quelli non critici.
I ritardi interni al sistema non sono limitati in maniera rigida e pertanto un task critico non è garantito nell'esecuzione in un lasso di tempo prefissato.
Nei sistemi Soft RTOS se un processo completa il suo lavoro correttamente, ma impiega più tempo del previsto, il risultato può ancora essere utile.
Esempio: In un sistema DAQ se il tempo di processamento è più lungo, aumenta il tempo morto, ma gli eventi sono ancora utilizzabili.

8.5.9 Real Time e Linux

Linux è un Sistema Operativo di tipo time-sharing standard così come molti altri SO per uso generico e quindi mostra delle performance buone in media e servizi altamente sofisticati come:

– Gestione dell'Hardware con meccanismi di Polling o di Interrupt;
– Classi di schedulazione che si occupano dell'attivazione dei processi, delle priorità e delle finestre temporali di esecuzione;
– Servizi di comunicazione fra applicazioni.

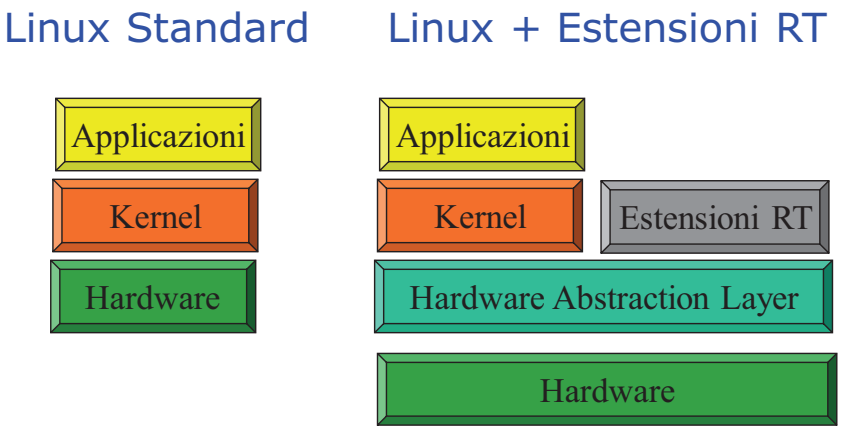


Figura 8.31 - Linux con estensioni Real Time

Pur essendo quindi un Sistema Operativo completo e sofisticato, Linux non ha nativamente un vero supporto per applicazioni Real Time. Esso infatti utilizza dei meccanismi di scheduling non RT:

– Scheduling generico per multitasking (a priorità 0)
– Scheduling prioritario (1-99), per RT, con meccanismi di scheduling di tipo FIFO (non-preemptive) e Round Robin (preemptive)

L'ambiente Linux standard non è Hard Real Time con molti dei suoi servizi come l'accesso al TCP/IP, il display grafico e l'uso di window system, file and data base system, ecc.. Una piattaforma Hard Real time ha, infatti, bisogno di bassa latenza e richiede un'alta predicibilità di comportamento.
È però possibile aggiungere un supporto Hard RT al sistema Linux ma a patto di modificare il kernel standard così come mostrato in Figura 8.31.

Capitolo 9

Reti di Comunicazione e TCP/IP

In un sistema complesso come alcuni DAQ, le componenti hanno necessità di comunicare in vario modo, ma principalmente per:

– Trasferire Dati
– Scambiare Messaggi

Sebbene queste due esigenze portino quasi sempre a sistemi di comunicazione separati, spesso però le tecnologie utilizzate sono le stesse o molto simili.
In aggiunta a ciò, un sistema DAQ deve occasionalmente comunicare con l'esterno per attività varie come, ad esempio:

– Scaricare del Software: Aggiornamenti ai Sistemi Operativi, Patch di Sicurezza, Nuove Versioni dei Task, ecc.
– Inviare informazioni relative al Monitoraggio del Sistema e del suo stato
– Trasferire Dati campionati o collezioni di eventi
– Ricevere comandi e/o richieste di dati da postazioni esterne per necessità di Debugging

In tutte queste attività è dunque necessario utilizzare un Protocollo di Rete che permetta tali tipi di comunicazione. In questo Capitolo ci occuperemo in maniera specifica del protocollo di rete più diffuso a livello mondiale dalla rete Internet, ossia il TCP/IP, ed i protocolli e tecnologie sottostanti, come l'Ethernet, che permettono l'utilizzo della rete fisica.

9.1 *Reti e Protocolli*

Iniziamo con definire i concetti di Network e Internet: una Network è un insieme di sistemi che sono in grado di comunicare fra di loro. Per far comunicare due sistemi che appartengono a Network diversi, è necessario un dispositivo specifico, che va sotto il nome di Router, e che ha la caratteristica di avere connessioni ad entrambe le Network e permette quindi il trasferimento delle informazioni dall'una all'altra.

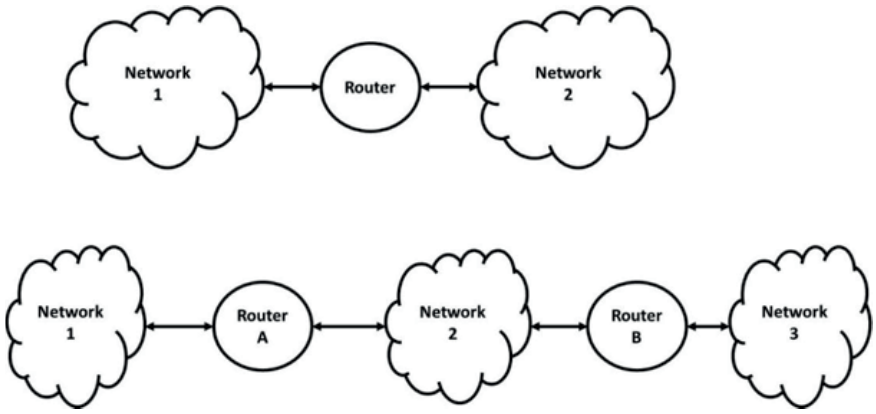


Figura 9.1 - Interconnessioni fra Reti - Internet

È possibile estendere questo modello a più Network Interconnesse fra loro da Router e questo va sotto il nome di Inter-Networking o brevemente Internet.

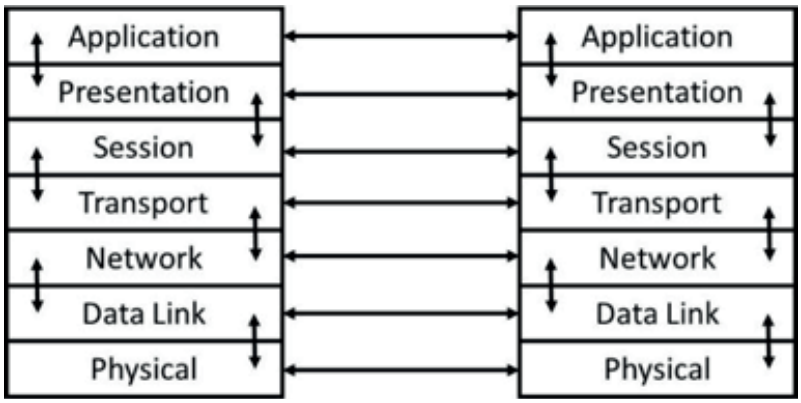


Figura 9.2 - Modello OSI

Per poter comunicare all'interno della Network e attraverso Internet, i sistemi devono uniformarsi a protocolli di rete che garantiscano la compatibilità delle modalità di trasferimento dell'informazione. Lo International Standard Organisation ha dunque definito un modello a 7 livelli chiamato OSI (Open Systems Interconnection):

– Application – A questo livello appartengono le Applicazioni che usano la Rete come il trasferimento di file o un emulatore di terminale;
– Presentation – In questo livello viene effettuata la formattazione dei dati e la loro codifica o criptazione;
– Session – Questo livello si occupa di stabilire e mantenere una sessione di comunicazione;
– Transport – In questo livello viene fornito un meccanismo di trasporto dell'informazione che può essere sia affidabile, che non affidabile;
– Network – Si occupa di consegnare l'informazione strutturata in Pacchetti e include anche i meccanismi di routing;
– Data Link – In questo livello avviene la composizione di organizzazione delle unità di informazione ed il controllo degli errori di comunicazione;
– Physical – Si tratta del livello fisico a cui avviene la trasmissione dei bit sul mezzo di comunicazione.

Questo modello è stato usato come riferimento per molti protocolli di rete, ma il Protocollo IP ed il TCP alle origini hanno avuto una corrispondenza architetturale solo parziale con il modello OSI, come mostrato in Figura 9.3.

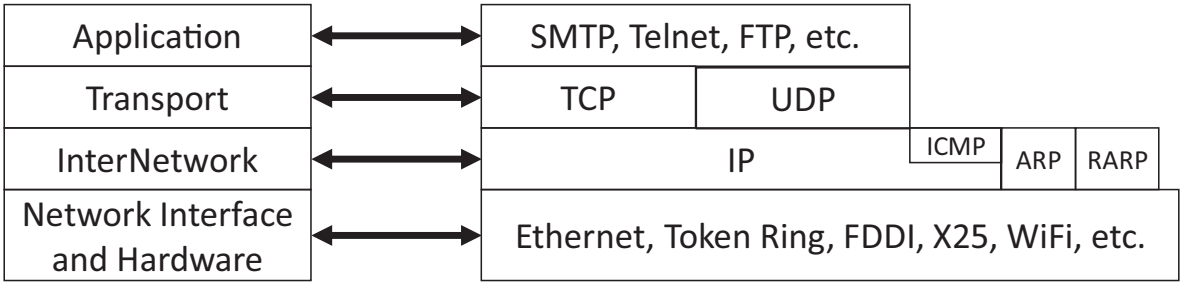


Figura 9.3 - Livelli TCP/IP

Esamineremo nei prossimi paragrafi l'intero stack dei protocolli di Rete più comunemente utilizzati, partendo dal TCP/IP ed arrivando all'Ethernet.

9.2 *Il TCP/IP*

Quando si parla del TCP/IP ci si riferisce all'insieme del protocollo di Trasporto TCP (Tansport Control Protocol) ed al protocollo di Internetwork IP (Internetwork Protocol). Il primo (TCP) è gerarchicamente sopra al secondo e lo usa per comunicare con le varie reti di computer e dispositivi. Partiremo però dall'IP e poi esamineremo, sia il TCP che l'altro protocollo originario di Trasporto UDP.

9.2.1 Internet Protocol (IP)

L'IP o Internet Protocol è un protocollo, che fa parte della suite di protocolli genericamente indicati come TCP/IP, che permette l'interconnessione di reti (Inter-Networking), e che è stato creato per interconnettere reti eterogenee per tecnologia, prestazioni, gestione, pertanto implementato sopra altri protocolli di livello collegamento, come, ad esempio, Ethernet, situandosi così al Livello 3 della pila ISO/OSI sopra descritta.

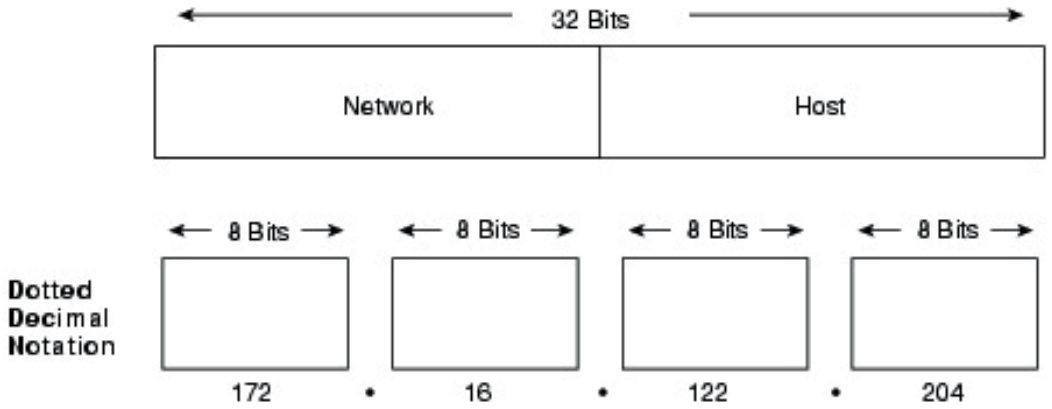


Figura 9.4 - Indirizzamento IPv4

IP è un protocollo a pacchetti (Packet), senza connessione (Connection-less) e di tipo best-effort, che non garantisce quindi alcuna forma di affidabilità della comunicazione in termini di controllo di errore, controllo di flusso e controllo di congestione, che possono essere invece realizzati dai protocolli di trasporto di livello superiore (livello 4), come TCP.

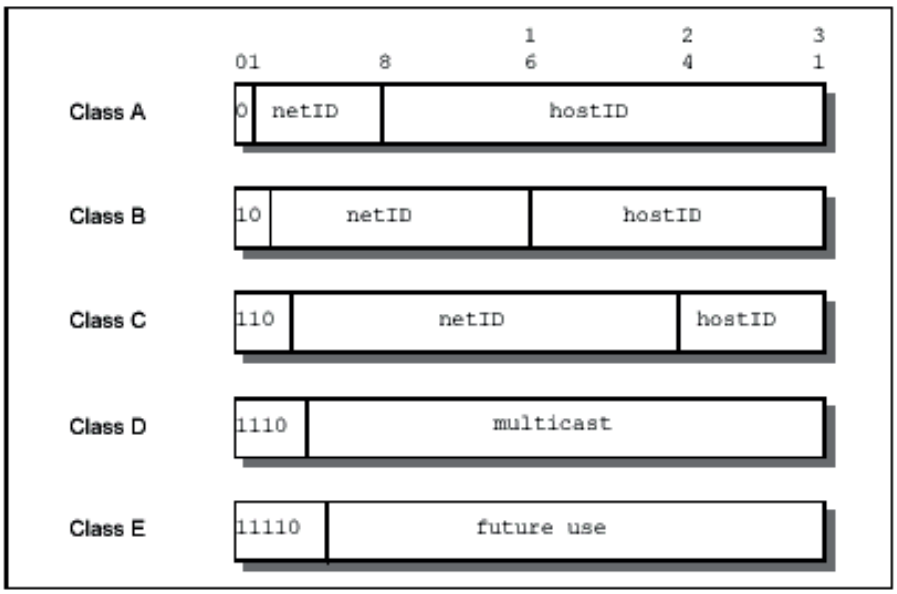


Figura 9.5 - Classi di indirizzamento IPv4

Esistono 2 versioni di IP, quella originaria chiamata versione 4 (IPv4) e una versione relativamente più recente chiamata versione 6 (IPv6) che usa un altro tipo di indirizzamento più ampio per affrontare la problematica dell'esplosione del numero di dispositivi (host) connessi ad Internet. Nel prosieguo ci occuperemo solo di IPv4.
Va inoltre chiarito che nella rete Internet in cui ci sono un numero enorme di reti e Router che sono variamente interconnessi, ci possono essere diverse strade per trasferire un pacchetto da un Host Sender ad un Host Receiver ed essendo IP un protocollo Connection-less come già visto, i pacchetti di una comunicazione fra due Host possono fare potenzialmente percorsi diversi in funzione, per esempio, del carico sulle connessioni o di interruzioni di alcune di esse. Questo fa sì che tali pacchetti, facendo strade diverse, possano arrivare in un ordine diverso da come sono stati spediti.
L'indirizzamento IPv4 è basato su un campo di 32 bit che sono suddivisi in 4 ottetti che vengono rappresentati come 4 campi decimali separati da un punto. Ciascun campo di 8 bit può dunque essere scritto come un numero decimale che va da 0 a 255.
Una parte dei 32 bit (quella più alta) è utilizzata per individuare il Network (Rete) e la parte bassa dei 32 bit è usata per individuare l'Host (Dispositivo). Il numero di bit utilizzati per l'uno e per l'altro può variare. Secondo lo schema originale sono individuate una serie di Classi di indirizzamento (A, B, C, D, E) mostrate in Figura 9.5 dove i 32 bit sono utilizzati per gestire un campo variabile per Network e Host. Questa scelta è stata effettuata per soddisfare esigenze diverse: da una parte reti piccole con pochi Host rischiavano di sprecare indirizzi e, dall'altra, reti molto grandi con molti Host rischiavano di essere suddivise in molte Network che poi necessitavano di essere interconnesse.

.

IP Address Class	Format	Purpose	High-Order Bit(s)	Address Range	No. Bits Network/Host	Max. Hosts
A	N.H.H.H[1]	Few large organizations	0	1.0.0.0 to 126.0.0.0	7/24	16,777, 214[2] (2^{24} - 2)
B	N.N.H.H	Medium-size organizations	1, 0	128.1.0.0 to 191.254.0.0	14/16	65, 543 (2^{16} - 2)
C	N.N.N.H	Relatively small organizations	1, 1, 0	192.0.1.0 to 223.255.254.0	21/8	254 (2^{8} - 2)
D	N/A	Multicast groups (RFC 1112)	1, 1, 1, 0	224.0.0.0 to 239.255.255.255	N/A (not for commercial use)	N/A
E	N/A	Experimental	1, 1, 1, 1	240.0.0.0 to 254.255.255.255	N/A	N/A

Tabella 9.1 - Classi di indirizzi IP e caratteristiche

Non tutte le reti possibili con l'indirizzamento IPv4 sono però utilizzabili in ambito Internet cioè possono essere esposte verso le altre reti. L'allocazione degli indirizzi IP è fatta da organismi internazionali (in Europa da RIPE – Réseaux IP Européens) con procedure per l'allocazione di tali indirizzi. Per facilitare l'uso di IP anche all'interno delle organizzazioni, in realtà si è deciso che alcune reti possono essere utilizzate in maniera libera entro la propria rete locale (LAN – Local Area Network) ma non sono utilizzabili per comunicare con altre reti esterne. Esempi di queste reti sono: la 127.0.0.0 (loopback), la 10.0.0.0 ecc.
Un'altra caratteristica del protocollo IP è la possibilità di consegnare pacchetti secondo meccanismi diversi così come mostrato in Figura 9.6:

– Unicast – il pacchetto è trasmesso da un dispositivo sorgente (S – Source) ad un altro singolo dispositivo di destinazione (D – Destination);
– Broadcast – il pacchetto è trasmesso dalla sorgente a tutti i dispositivi sulla stessa rete che sono tutti destinatari;
– Multicast – il pacchetto è trasmesso dalla sorgente solo verso una categoria di destinatari;
– Anycast – il pacchetto è trasmesso dalla sorgente verso uno qualunque di un insieme di destinatari.

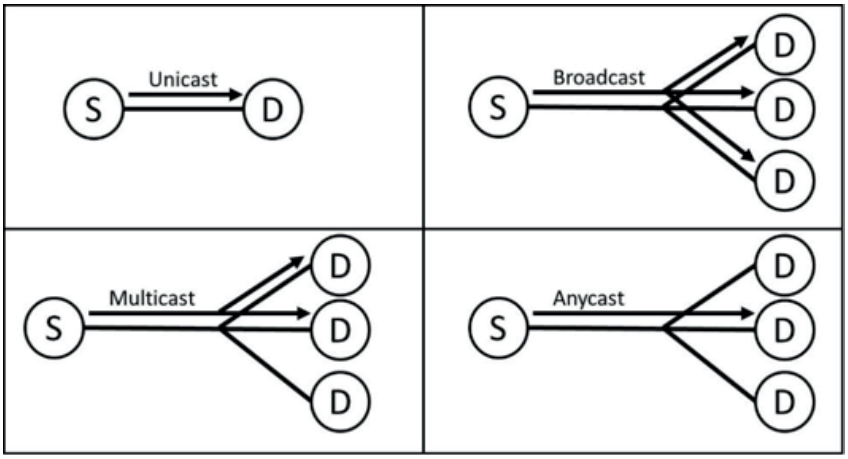


Figura 9.6 - Meccanismi di consegna dei pacchetti IP

Un'altra caratteristica del protocollo IP è la possibilità di suddividere una rete di una certa classe di indirizzi in reti più piccole utilizzando un meccanismo che si chiama di sub-netting.

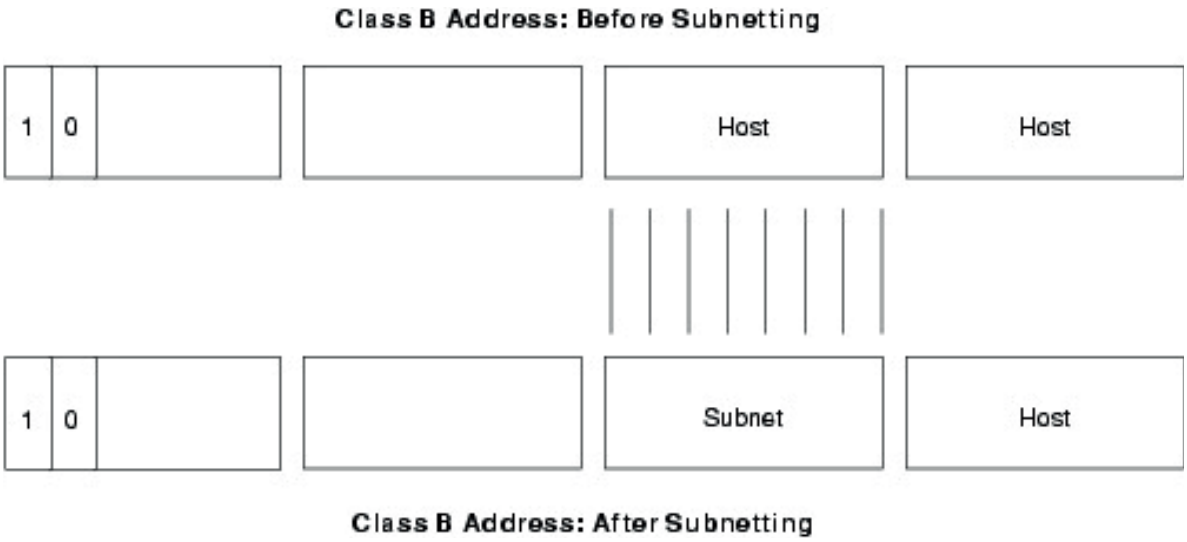


Figura 9.7 - Meccanismo di sub-netting

In tale meccanismo il numero dei bit destinato ad individuare la Network viene incrementato a scapito del numero dei bit destinati agli indirizzi degli Host. In Figura 9.7 è mostrato un esempio in cui una Classe B che ha a disposizione 16 bit per gli indirizzi Host può essere suddivisa in più sottoreti utilizzando come subnet i primi 8 bit più alti dei due campi Host. In questa maniera si otterranno 256 sottoreti di 256 Host invece di una unica rete di 65.536 Host. L'uso del subnetting ha alcuni vantaggi perché permette di suddividere una rete fornendo maggiore flessibilità all'amministratore di rete migliorando o razionalizzando l'uso degli indirizzi disponibili e contenendo il traffico di Broadcast. Per fare in modo che tutti gli Host ed i Router della rete siano al corrente del subnetting, si usa aggiungere nelle tabelle di descrizione della rete una maschera all'indirizzo di rete. Nella Figura 9.8 è mostrato un esempio in cui i primi tre ottetti sono utilizzati come indirizzo di Network (sono con tutti i Bit a 1 e quindi ogni ottetto ha un valore decimale di 255) e solo l'ultimo ottetto è usato per indirizzo degli Host con un valore decimale di 0.

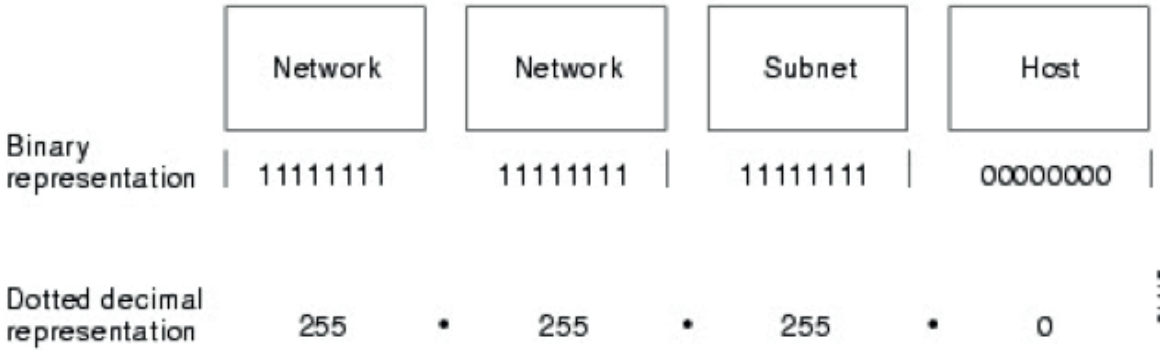


Figura 9.8 - Maschera di subnetting

Naturalmente è possibile utilizzare delle configurazioni di bit di maschera diverse a seconda del numero di bit che si vogliono dedicare all'indirizzo di Network.
Nella Figura 9.9 sono mostrate le possibili configurazioni di bit ed i rispettivi valori in decimale di un ottetto.

128	64	32	16	8	4	2	1		
1	0	0	0	0	0	0	0	=	128
1	1	0	0	0	0	0	0	=	192
1	1	1	0	0	0	0	0	=	224
1	1	1	1	0	0	0	0	=	240
1	1	1	1	1	0	0	0	=	248
1	1	1	1	1	1	0	0	=	252
1	1	1	1	1	1	1	0	=	254
1	1	1	1	1	1	1	1	=	255

Figura 9.9 - Configurazione dei bit per maschere di subnet

Nella figura seguente viene mostrato un esempio di una Classe B piatta cioè senza subnetting in cui il Network 131.154.0.0 ha a disposizione 2 ottetti (16 bit) per gli indirizzi degli Host. La maschera di rete in questo caso è 255.255.0.0 e cioè è quella standard che è prevista per una rete di Classe B.

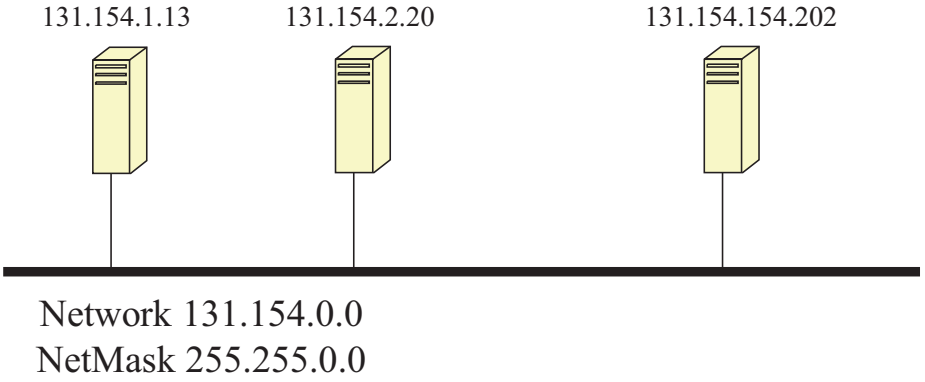


Figura 9.10 - Esempio di Rete di Classe B piatta

In Figura 9.11 è mostrato un esempio di rete di Classe B con subnet in cui la rete 131.154.0.0 utilizzando il terzo ottetto per suddividerla in subnet come quelle mostrate 131.154.1.0 e 131.154.20.0 in cui le maschere 255.255.255.0 hanno i bit a 1 anche nel terzo ottetto.

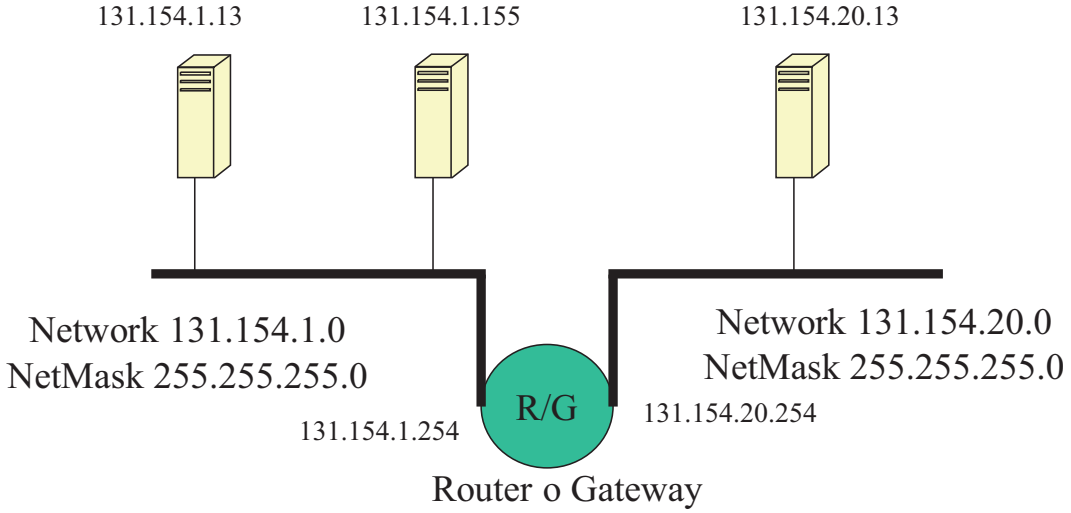


Figura 9.11 - Esempio di Rete di Classe B con subnet

Come si può vedere in questo secondo esempio per mantenere la connettività fra le due sottoreti è necessario interporre un Router o un Gateway che trasferisce i pacchetti da una subnet all'altra.
Come abbiamo visto il protocollo IP trasmettere informazioni sotto forma di pacchetti di dati e in Figura 9.12 è mostrata la struttura di tali pacchetti che andiamo ad analizzare.

– Version — Indica la versione di IP che viene usata (la v4);
– IP Header Length (IHL) — Indica la lunghezza della intestazione (Header) del pacchetto di dati (Datagram) in parole di 32 bit;
– Type-of-Service — Specifica come un protocollo di più alto livello vorrebbe che questo Datagram fosse gestito assegnando vari livelli di importanza;
– Total Length — Specifica la lunghezza in bytes dell'intero pacchetto IP inclusi I dati e lo Header;
– Identification—Contiene un intero (integer) che identifica il corrente Datagram e viene usato per poter rimettere insieme dei Datagram che sono stati suddivisi o frammentati in più pacchetti IP;
– Flags — è un campo di 3 bit nel quale i due bit meno significativi (low-order o least-significant) controllano la frammentazione del Datagram e dove il bit meno significativo specifica se il pacchetto può essere frammentato, mentre il bit di mezzo specifica se il pacchetto contiene l'ultimo frammento di una serie di frammenti del Datagram frammentato; il bit più significativo non è usato;
– Fragment Offset — Indica la posizione del frammento dei dati rispetto all'inizio dei dati nel Datagram originale e permette al processo di destinazione dell'IP di ricostruire in maniera corretta il Datagram originale;
– Time-to-Live — è un contatore che viene gradualmente decrementato ogni volta che un pacchetto attraversa un Router ed arriva fino a 0 ed a questo punto il Datagram viene scartato; questo meccanismo evita che i pacchetti possano girare in loop senza fine;

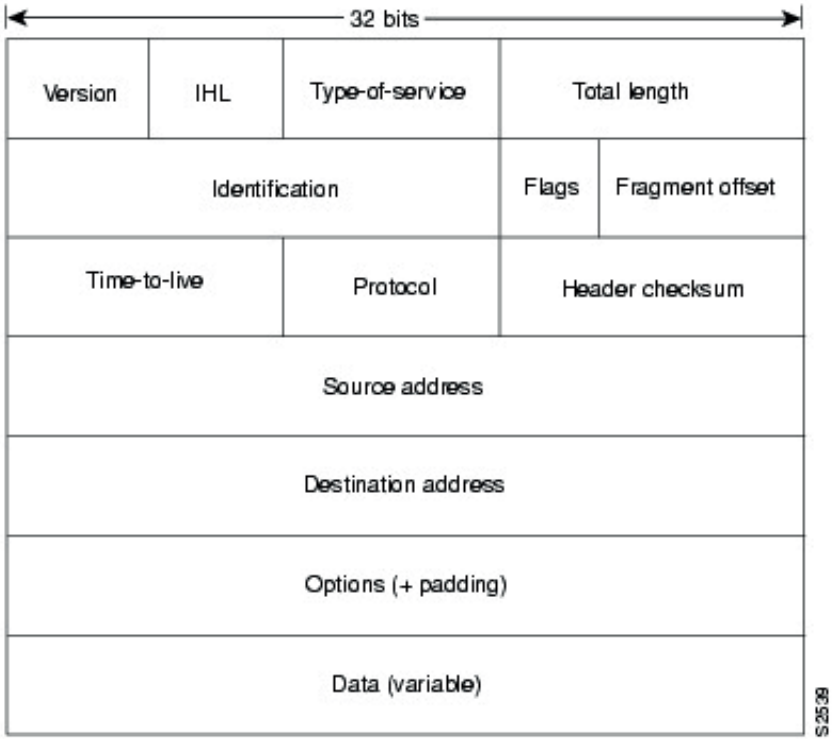


Figura 9.12 - Struttura del pacchetto IP

– Protocol — Indica quale protocollo di più alto livello deve ricevere il pacchetto dopo che è stato processato dall'IP;
– Header Checksum — Contiene un codice Checksum di controllo per verificare l'integrità dell'IP Header;
– Source Address — In questo campo è specificato l'indirizzo IP del nodo (Host) che ha spedito il pacchetto;
– Destination Address — Contiene l'indirizzo IP del nodo (Host) che deve ricevere il pacchetto;
– Options — è un campo che permette all'IP di supportare varie opzioni come per esempio la sicurezza;
– Data — Contiene l'informazione che deve essere passata al protocollo di livello superiore.

9.2.2 Protocollo UDP

Lo User Datagram Protocol (UDP) è un protocollo di trasporto (Livello 4 della pila ISO/OSI) di tipo connectionless che fa parte della famiglia dei protocolli di Internet.
UDP è concepito come una semplice interfaccia fra IP ed i processi di livello superiore e permette, attraverso l'uso di porti (Port) di distinguere le comunicazioni in corso da parte di applicazioni multiple che eseguono su uno stesso dispositivo.
Esso non aggiunge all'IP nessun meccanismo di affidabilità o di controllo di flusso o di correzione di errori di trasmissione così come invece, vedremo, fa il TCP. UDP punta sulla sua semplicità e leggerezza aggiungendo con il suo header solo pochi bytes al messaggio e producendo quindi un sovraccarico minimo alle risorse di rete.
UDP è dunque utilizzabile in tutte quelle situazioni in cui i meccanismi supplementari di affidabilità del TCP non sono necessari come quelli in cui un protocollo di più alto livello provvede a realizzare tali funzionalità. È anche utile notare come le sue caratteristiche siano usate da molti protocolli di livello applicativo molto utilizzati sulla rete come ad esempio il Network File System (NFS), il Simple Network Management Protocol (SNMP), il Domain Name System (DNS), ed anche il Trivial File Transfer Protocol (TFTP).

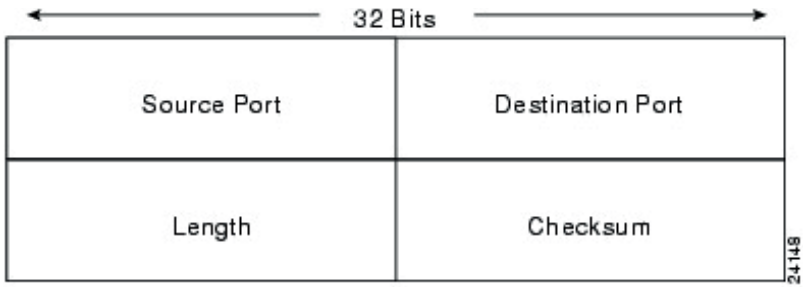


Figura 9.13 - Header del pacchetto UDP

In Figura 9.13 è mostrata la struttura dello Header del pacchetto UDP che contiene solo quattro campi di 16 bit ciascuno:
– Source Port contiene il numero a 16 bit che identifica il porto della sorgente della comunicazione per distinguerla da altre in ambiente applicativo multiutente;
– Destination Port è analogamente il numero a 16 bit del porto della destinazione della comunicazione;
– Length specifica la lunghezza del pacchetto UDP con Header e dati;
– Checksum fornisce un controllo opzionale di integrità del pacchetto UDP con Header e dati.

L'uso di identificativi Port è, come già accennato, legato alla necessità di distinguere le varie comunicazioni di ciascun processo da quelle di altri processi. Esso è un numero intero a 16 bit utilizzato nelle comunicazioni fra Host per identificare il protocollo di più alto livello o il programma applicativo a cui vanno consegnati i pacchetti ricevuti e relativi a tale comunicazione.
Ci sono due tipi di Port:

– Well-known: Sono le porte conosciute per essere usate dai servizi standard della suite TCP/IP, come per esempio Telnet usa il port 23. Questi port hanno valori compresi fra 1 and 1023 e sono controllate ed assegnate da Internet Assigned Number Authority (IANA);
– Ephemeral: Sono Port volatile che non sono assegnati a specifiche applicazioni o servizi hanno un valore nell'intervallo da 1024 a 65525 e sono quindi utilizzabili dalle applicazioni utente alle quali vengono allocati dallo Host su cui sono in esecuzione per il tempo necessario alle loro necessità di comunicazione in maniera che sia univoca la combinazione <transport protocol, IP address, port number>.

UDP e TCP usano entrambi lo stesso principio relativo ai Port e, per quanto possibile, usano uno schema di Port number analogo per i servizi di più alto livello.

9.2.3 TCP

Il Transport Control Protocol (TCP) è un protocollo di trasporto (Livello 4 nella pila ISO/OSI) che lavora al di sopra dell'IP per fornire una serie di servizi alle applicazioni. Come mostrato in Figura 9.14 anche TCP usa, come UDP, i porti per distinguere le trasmissioni relative a diversi processi.

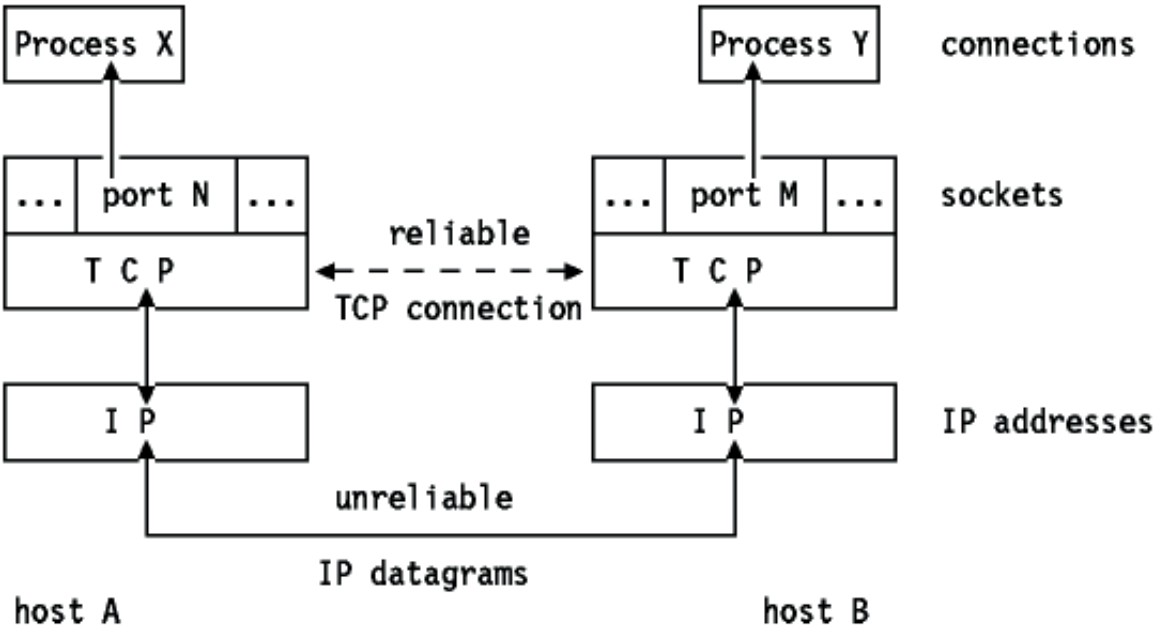


Figura 9.14 - Funzionamento Porti TCP

Il TCP offre una serie ampia di servizi rispetto all'UDP:

– Stream Data Transfer – dal punto di vista dell'applicazione il TCP appare trasferire un flusso (Stream) continuo di byte, anche se in realtà esso raggruppa tali byte in segmenti TCP che sono poi passati all'IP per la trasmissione verso la destinazione. Il TCP decide autonomamente come dividere il flusso dei byte in tali segmenti e quando inviarli.
– Reliability – Il TCP assegna un numero sequenziale a ogni byte trasmesso e aspetta una risposta positiva (Acknowledgment o ACK) dal TCP in ricezione. Se l'ACK non viene

ricevuto entro un intervallo di tempo (timeout) i dati sono ritrasmessi. Il TCP ricevente usa il numero sequenziale per riordinare i segmenti dei dati quando questi arrivano fuori ordine oppure per eliminare i segmenti duplicati.

– Flow Control – Il TCP ricevente quando invia l'ACK al sender indica anche il numero di byte che può ricevere, al di là del segmento già ricevuto, senza che questo causi uno sconfinamento (overrun and overflow) dei suoi buffer interni. Ciò avviene inviando nel pacchetto di ACK il numero di sequenza più grande che può ricevere senza avere problemi.
– Multiplexing – Permette a molti processi all'interno di un singolo Host di usare i servizi di comunicazione di TCP simultaneamente e senza interferenze. Il TCP mette a disposizione un insieme di Port all'interno di ciascun Host e questi concatenati agli indirizzi della rete e dell'Host formano quello che viene chiamato socket ed una coppia di socket definisce in maniera univoca ogni connessione
– Logical Connections – L'affidabilità ed il meccanismo di controllo di flusso descritti sopra, richiedono che il TCP registri e mantenga delle informazioni sullo stato di ogni flusso di dati. L'insieme di queste informazioni che includono anche i socket, i numeri delle sequenze e la grandezza delle finestre è chiamato connessione logica (Logical Connection) e ogni connessione è definita univocamente dalla coppia di socket usati dai processi sorgente e destinazione
– Full Duplex – Il TCP permette la gestione di flussi di dati coesistenti in entrambe le direzioni.

9.2.3.1 La struttura del segmento TCP

Nella Figura 9.15 è mostrata la struttura dello Header e del pacchetto del protocollo TCP il cui pacchetto viene poi inserito all'interno del pacchetto IP.

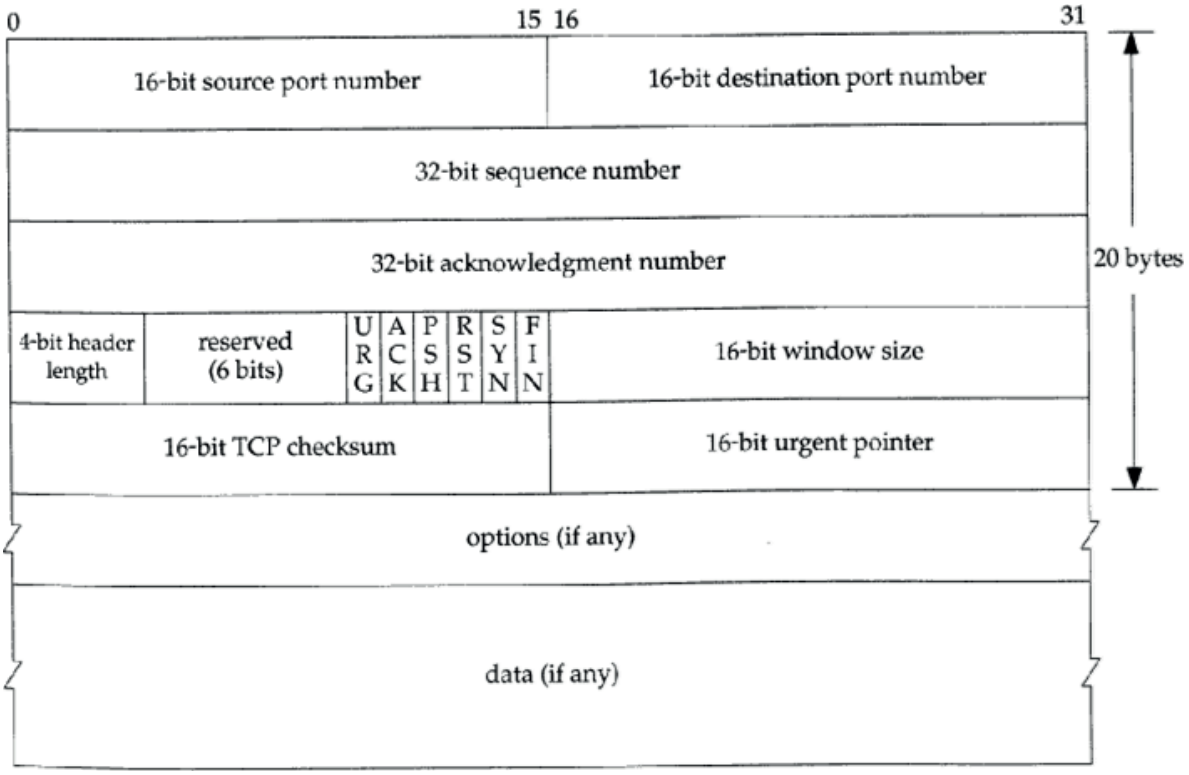


Figura 9.15 - Header del protocollo TCP

I vari campi del pacchetto TCP sono dunque descritti come i seguenti.

– SrcPort e DstPort individuano due campi da 16 bit che contengono i Port sorgente (Source Port) e destinazione (Destination Port) rispettivamente. La combinazione di questi due campi e quelli degli indirizzi IP di source e destination identificano univocamente ogni connessione TCP.
– SequenceNum – Il numero di sequenza a 32 bit identifica il byte nel flusso dei dati che provengono dal sorgente alla destinazione come il primo byte del segment di dati.
– Il campo Acknowledgement number contiene un intero a 32 bit che è il numero di sequenza prossimo che il sender dell'acknowledgement si aspetta di ricevere e questo è dunque è il numero di sequenza dell'ultimo byte di dati ricevuto aumentato di uno. Questo campo è valido solo se il campo ACK flag (vedi più avanti) è abilitato e quando la connessione è stabilita il flag ACK è sempre ON.
– I campi Acknowledgement, SequenceNum ed AdvertisedWindow sono tutti coinvolti nell'algoritmo di gestione della finestra a scorrimento (Sliding Window) del TCP.
– Il campo Header Lenght indica la lunghezza dell'header in parole di 32 bit ed è necessario perché la lunghezza del campo options è variabile.
– Il campo a 6-bit Flags è usato per riportare delle informazioni di controllo fra i due TCP in comunicazione fra loro. I Flag possibili includono SYN, FIN, RESET, PUSH, URG e, come abbiamo già visto, ACK.
  • I Flag SYN e FIN sono usati per stabilire e terminare, rispettivamente, la connessione TCP.
  • Il Flag ACK, come già esposto, è alzato nel momento che il campo Acknowledgement è valido e richiede l'attenzione del ricevitore.
  • Il Flag URG significa che questo segmento contiene dati urgenti e quando è attivato il campo UrgPtr indica dove cominciano i dati non urgenti nel segmento.
  • Il Flag PUSH significa che il trasmettitore (sender) ha invocato l'operazione di push e che quindi questo deve essere notificato al processo ricevente dal TCP del ricevitore. Questo Flag si riferisce all'opzione che il TCP fornisce all'applicazione sorgente di chiedere l'invio dei dati presenti nel buffer anche se il buffer stesso non è ancora pieno e cioè contiene meno dati dello MTU (Maximum Transmission Unit).
  • Infine il Flag RESET significa che il TCP ricevente è in una situazione di confusione e chiede di abortire la connessione.
– Il Checksum è il codice di controllo che copre l'intero segment TCP: Header e Dati ed è un campo obbligatorio che deve essere calcolato dal sender e poi verificato all'arrivo dal receiver.
– Il campo Option è usato per specificare la grandezza massima del segment (MSS – Maximum Segment Size) che viene normalmente specificata nel primo pacchetto che viene scambiato e specifica la grandezza massima del segmento che il sender vuole ricevere.
– Infine vi è la porzione del segmento TCP in cui possono opzionalmente inseriti i dati da inviare.

9.2.3.2 TCP Sliding Window

La finestra a scorrimento (Sliding Window) di TCP è un meccanismo che è utilizzato per diverse finalità ed in particolare per:

1 garantire la consegna dei dati in maniera affidabile;
2 assicurare che i dati vengano consegnati nell'ordine corretto;
3 assicurare un controllo di flusso fra sender e receiver.

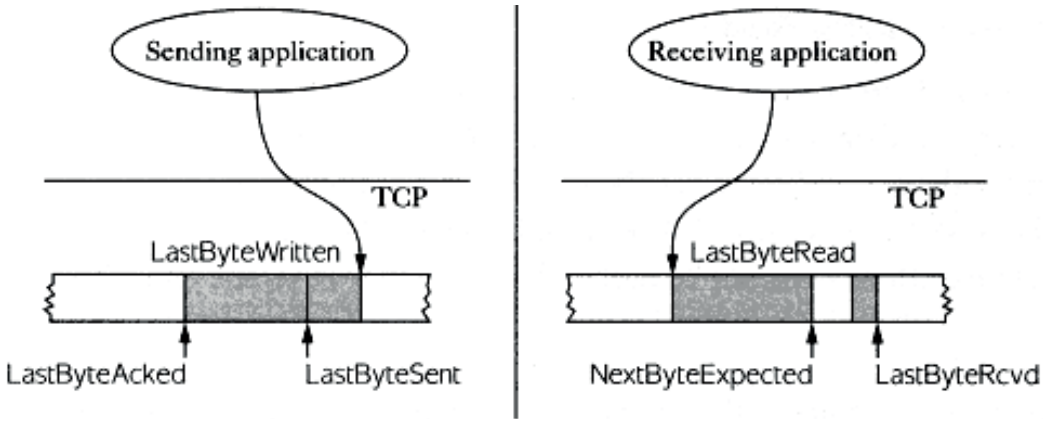


Figura 9.16 - TCP Sliding Window

In Figura 9.16 è mostrata una rappresentazione grafica delle due finestre utilizzate dall'applicazione che invia i dati e di quella che li riceve. I due lati del TCP interagiscono per implementare una consegna di dati affidabile ed ordinata grazie ad essa. Premesso che ogni byte ha un numero di sequenza e che gli ACK sono cumulativi, esaminiamo con un po' di dettaglio l'uso che se ne fa per raggiungere questo scopo.

– Dal lato del trasmettitore (Sending side):
  - LastByteAcked <=LastByteSent – l'ultimo byte che ha ricevuto l'ACK deve essere minore o uguale all'ultimo byte inviato;
  - LastByteSent <= LastByteWritten – l'ultimo byte inviato, a sua volta, deve essere minore o uguale all'ultimo byte scritto dall'applicazione;
  - I byte fra LastByteAcked e LastByteWritten devono essere ospitati in un buffer.
– Dal lato del ricevitore (Receiving side):
  - LastByteRead < NextByteExpected – l'ultimo dato letto dall'applicazione ricevente deve essere minore del prossimo byte atteso;
  - NextByteExpected <= LastByteRcvd + 1 – a sua volta il prossimo byte atteso deve essere minore uguale dell'ultimo byte ricevuto più uno;
  - I byte fra NextByteRead e LastByteRcvd devono essere ospitati in un buffer.

Un altro uso della Sliding Window è relativo al controllo di flusso (Flow Control). Come già visto nei precedenti paragrafi, i campi Acknowledgement ed AdvertisedWindow contengono informazioni che vanno nella direzione opposta, dal ricevitore al trasmettitore, sul flusso dei dati che il primo è in grado di sostenere.
L'algoritmo della sliding window del TCP permette al ricevitore di esporre al trasmettitore la grandezza della finestra in cui ricevere i prossimi dati usando il campo AdvertisedWindow. Il trasmettitore viene pertanto limitato in ogni momento della comunicazione a non eccedere il valore di AdvertisedWindow come numero di bytes che non hanno ancora ricevuto l'ACK. Il valore di AdvertisedWindow viene stabilito dal ricevitore sulla base della quantità di memoria usata dalla connessione come buffer per i dati. In particolare:

– Sender buffer size : MaxSendBuffer è il buffer massimo da trasmettere
– Receive buffer size : MaxRcvBuffer è il buffer massimo per ricevere
– Dal lato ricevitore (Receiver side):
  - LastByteRcvd - NextByteRead <= MaxRcvBuffer – la differenza fra l'ultimo byte ricevuto ed il prossimo byte da leggere è ciò che non è ancora stato letto dall'applicazione e deve rimanere nella memoria di buffer a disposizione dell'applicazione.
  - AdvertisedWindow = MaxRcvBuffer - (LastByteRcvd - NextByteRead) – la finestra

di possibili dati da ricevere è dunque la differenza fra il massimo del buffer di ricezione e quello che è nel buffer in attesa di essere letto dall'applicazione.

– Dal lato trasmettitore (Sending side):
  - LastByteSent - LastByteAcked <= AdvertisedWindow – l'ultimo byte inviato meno l'ultimo byte acknowledged sono i dati che sono nel buffer perchè ancora non hanno ricevuto l'ACK e questo deve essere minore dell'AdvertisedWindow.
  - EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked) – la finestra disponibile è dunque quella fra AdvertisedWindow ed i dati che non hanno ancora ricevuto l'ACK.
  - LastByteWritten - LastByteAcked <= MaxSendBuffer – l'ultimo byte scritto dall'applicazione meno l'ultimo byte che ha ricevuto l'ACK è la quantità di dati ancora da inviare più quella ancora da confermare con ACK e deve essere ovviamente minore della grandezza massima del buffer del trasmettitore.
  - Il trasmettitore dunque si ferma se (LastByteWritten - LastByteAcked) + y > MaxSendBuffer.

È necessario dunque che il ricevitore mandi sempre l'ACK in risposta all'arrivo di segmento di dati, altrimenti la trasmissione si blocca temporaneamente e si innesca il meccanismo della ritrasmissione come vedremo nel prossimo paragrafo.
Da parte sua il trasmettitore insisterà nel caso che AdvertisedWindow = 0.

9.2.3.3 Ritrasmissione adattabile

Il TCP garantisce la consegna affidabile dei dati e, quindi, ritrasmette ogni segmento dati per il quale non abbia ricevuto un ACK all'interno di un certo periodo di tempo (timeout). Questo timeout non è fisso ma viene impostato come funzione del parametro di Round Trip Time (RTT) atteso fra le due parti della connessione; si tratta in sostanza del tempo che ci mette un messaggio ad andare dal trasmettitore al ricevitore e tornare. Purtroppo, questo parametro ha un'alta variabilità per ogni coppia possibile di Host in Internet ed inoltre cambia con il tempo a seconda del traffico e delle possibili strade che i pacchetti possono percorrere. Calcolare un valore appropriato del timeout non è dunque un compito semplice TCP usa dunque un meccanismo di ritrasmissione adattiva. Vari algoritmi si sono avvicendati nel tempo per effettuare tale meccanismo, qui esaminiamo, a scopo didattico, il meccanismo originale:

– Si parte misurando un campione SampleRTT per ogni coppia di segment/ACK;
– Si calcola la media pesata degli RTT;
– Si calcola quindi una stima di RTT come: EstimatedRTT = a*EstimatedRTT + b*SampleRTT, dove a+b = 1; a è compreso fra 0.8 e 0.9 e b è compreso fra 0.1 e 0.2;
– Si imposta il timeout sulla base di EstimatedRTT come TimeOut = 2 * EstimatedRTT

Una modifica di questo algoritmo che va sotto il nome di Karn/Partridge Algorithm prevede di aggiungere due altri elementi: non si campiona lo RTT durante le ritrasmissioni e si raddoppia il timeout dopo ogni ritrasmissione.

9.2.3.4 Controllo della congestione

Il TCP, come già accennato ha anche funzionalità che cercano di evitare o moderare i problemi di congestione di traffico. Un primo meccanismo è quello che va sotto il nome di Slow Start.
Questa funzione opera osservando che la frequenza alla quale i pacchetti dovrebbero essere inviati in rete è quella della ricezione degli acknowledgment inviati dal ricevitore. Slow start dunque aggiunge un altro tipo di finestra al TCP del trasmettitore: la finestra di congestione (congestion window o anche "cwnd"). Quando viene stabilita una nuova connessione con un Host di un'altra

rete, la congestion window è inizializzata ad un solo segmento (tipicamente 536 o 512 o come annunciato dal ricevitore) ed ogni volta che viene ricevuto un ACK la finestra viene incrementata di un segmento. Il trasmettitore può dunque trasmettere un blocco di dati fino al minimo fra la Congestion window e la Advertised window vista nel paragrafo precedente. La finestra di congestione è dunque un meccanismo di controllo di flusso imposto dal trasmettitore, mentre l'Advertised window è un controllo di flusso imposto dal ricevitore. Il primo si basa sulla valutazione della congestione della rete percepita dal trasmettitore, il secondo è invece, come già visto, basato sulla quantità di spazio disponibile nel buffer del ricevitore per la connessione in corso.
Il trasmettitore, dunque inizia trasmettendo un segmento ed aspettando l'ACK. Quando questo viene ricevuto, la finestra di congestione è incrementata a due segmenti e questi possono essere entrambi inviati ed alla ricezione di entrambi gli acknwoledgment, la finestra di congestione viene incrementata a quattro segmenti con una crescita potenzialmente esponenziale. In realtà tale crescita non è esattamente esponenziale perché il ricevitore può ritardare l'invio dei suoi ACK. Ad un certo punto, si raggiunge la capacità limite della comunicazione ed uno dei router intermedi può iniziare a scartare dei pacchetti che non arrivano al ricevitore che fa mancare gli ACK. Il trasmettitore dunque comprende che la sua Congestion window è divenuta troppo grande.

Un'altra funzione del TCP riguarda il cosiddetto Fast retransmit che permette al TCP di generare un acknowledgment immediato (un ACK duplicato) quando riceve un segmento di dati non nell'ordine giusto o atteso. La funzione di questo ACK duplicato è di far sapere al trasmettitore che si è ricevuto un segmento fuori ordine e di poter comunicare anche qual è il numero di sequenza atteso. Il TCP del trasmettitore non sapendo se l'ACK duplicato è stato causato dalla perdita di un segmento o soltanto da un ordine sbagliato di arrivo, aspetta di ricevere altri ACK. Infatti, nel caso si tratti solo di un problema di ordine dei pacchetti, saranno ricevuti solo uno o due ACK duplicati finchè il segmento sarà rimesso in ordine ed un ACK normale verrà inviato. Se invece vengono ricevuti tre o più ACK duplicati di fila questo è una forte indicazione che il segmento è andato perso. In questo caso il TCP effettua una ritrasmissione di quello che sembra essere il segmento perso, senza attendere lo scadere del timer per la ritrasmissione.

9.2.4 Network Address Translation (NAT)

Le reti usate per DAQ sono di tipo "chiuso", cioè non c'è necessità, in generale, di accedere a risorse esterne al sistema DAQ. La rete locale non ha pertanto necessità di avere indirizzi IP registrati ufficialmente e può usare invece indirizzi "nascosti".
In ciascuna classe di indirizzi IP sono stati riservati dei range di indirizzi non registrati dallo IANA (Internet Assigned Numbers Authority) che possono essere usati per reti nascoste.

– Range 1: Class A – Da 10.0.0.0 fino a 10.255.255.255
– Range 2: Class B – Da 172.16.0.0 fino a 172.31.255.255
– Range 3: Class C – Da 192.168.0.0 fino a 192.168.255.255

È però, a volte, necessario accedere a siti esterni da sistemi DAQ (es: per scaricare delle patch di sistema o degli aggiornamenti) e anche dall'esterno accedere ai sistemi DAQ per effettuare controlli e monitoraggi o manutenzioni correttive da parte di esperti remoti.
Per questi casi si usa il NAT che permette a degli host su reti nascoste di accedere ad Internet. Il NAT è uno standard IETF – RFC 1631 e lavora al Livello 3 della pila standard ISO/OSI e, per farlo funzionare è necessario che ci sia un Router che viene interposto fra la rete locale (LAN – Local Area Network) e la rete esterna Internet.

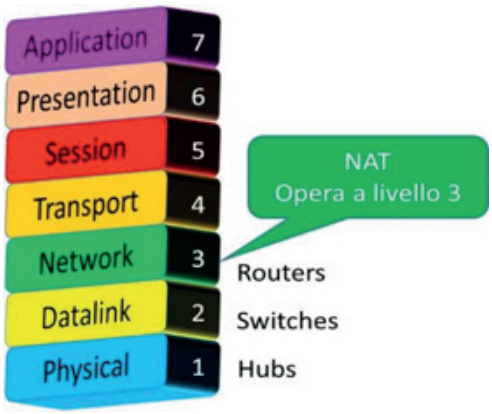


Tabella 9.17 - NAT e pila ISO/OSI

Un router NAT traduce indirizzi nascosti in indirizzi registrati che sono validi per essere usati verso una rete geografica e Internet e viceversa.

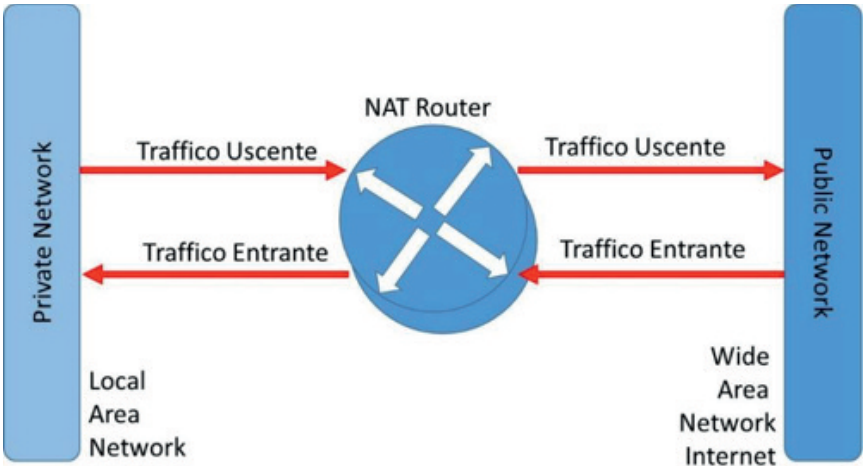


Figura 9.18 - Router NAT

Esistono vari tipi di NAT e li esaminiamo qui di seguito.

9.2.4.1 Static NAT

Il NAT statico effettua una mappatura uno ad uno fra un IP non registrato e uno invece registrato. Questo tipo di NAT è particolarmente utile quando un dispositivo sulla rete interna deve essere accessibile anche dall'esterno.

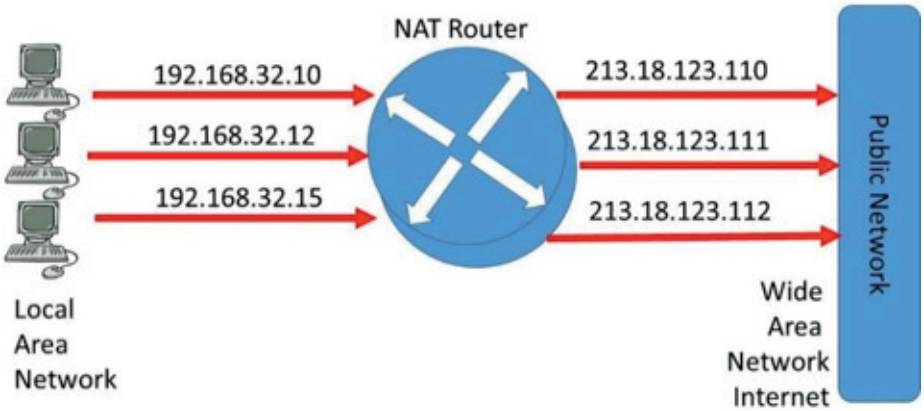


Figura 9.19 - Esempio di Static NAT

Come si può vedere nell'esempio mostrato in Figura 9.19, nello Static NAT il computer con indirizzo IP interno 192.168.32.10 verrà sempre tradotto nell'indirizzo IP esterno 213.18.123.110.

9.2.4.2 Dynamic NAT

Il meccanismo di NAT dinamico mappa un indirizzo IP di tipo non registrato con un indirizzo IP registrato prelevato da un gruppo di tali indirizzi a disposizione. Tale associazione è dunque temporanea e non fissata.

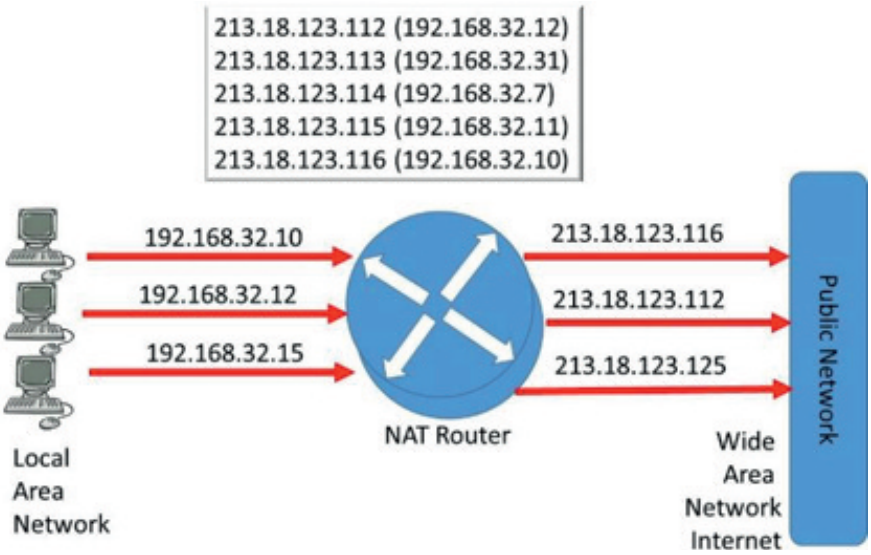


Figura 9.20 - Esempio di Dynamic NAT

Nel NAT dinamico, come si vede nell'esempio di Figura 9.20, il computer con l'indirizzo IP interno 192.168.32.10 viene tradotto con il primo indirizzo disponibile nel gruppo di indirizzi registrati che va da 213.18.123.100 a 213.18.123.150.

9.2.4.3 Overloading

Il NAT di tipo Overloading (sovraccarico) è una forma di dynamic NAT che mappa più indirizzi IP non registrati su un singolo indirizzo IP registrato usando dei Port differenti per gestire i diversi accoppiamenti. Questo modo di fare traduzione di indirizzi è anche conosciuto come PAT (Port Address Translation), single address NAT o port-level multiplexed NAT.

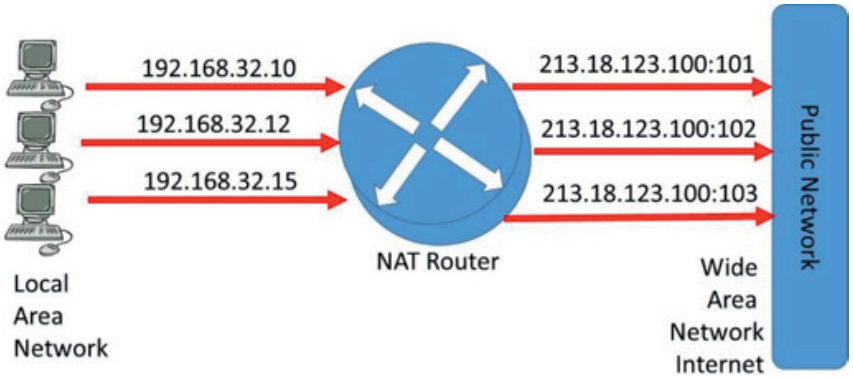


Figura 9.21 - Esempio di Overloading NAT

Nel NAT Overlaoding, come mostrato nell'esempio in Figura 9.21, ogni computer della rete privata viene tradotto nello stesso indirizzo IP registrato (213.18.123.100), ma con un numero differente di Port assegnato. Questo meccanismo è quello generalmente utilizzato dai fornitori

commerciali di collegamenti IP (Internet Service Provider – ISP). La rete interna viene realizzata utilizzando indirizzi IP non registrati e quindi non-routable e dunque viene installato un Router NAT che ha un indirizzo IP registrato unico. Il meccanismo dunque si compone dei seguenti passi:

– Un computer sulla rete locale (LAN) tenta di collegarsi con un altro computer di una rete esterna, per esempio un web server;
– Il Router NAT riceve il pacchetto del computer sulla LAN e salva l'indirizzo IP interno del computer ed un numero di Port in una tabella di traduzione di indirizzi (Address Translation Table - ATT);
– Il Router NAT sostituisce l'indirizzo IP interno del computer con l'indirizzo IP registrato del Router stesso e sostituisce il Port di sorgente del computer con quello che il Router ha assegnato e salvandola nella ATT;
– Nella ATT c'è dunque la mappatura fra l'indirizzo IP interno e il numero di Port del computer con l'indirizzo IP del Router ed il porto assegnato;
– Quando un pacchetto ritorna dal computer di destinazione, il Router controlla il Port di destinazione sul pacchetto e guarda nella ATT per verificare a quale computer della rete interna il pacchetto appartiene, ne cambia l'indirizzo ed il porto di destinazione con quelli ricavati dalla ATT ed invia il pacchetto a quel computer;
– Il computer della rete locale riceve il pacchetto dal router e il processo si ripete finchè il computer ha necessità di comunicare con il sistema esterno;
– Poiché il Router NAT adesso ha già registrati nella ATT l'indirzzo e il port di sorgente del computer, continuerà ad usarli per tutta la durata della connessione;
– Un timer viene attivato ogni qual volta il router accede a un elemento della tabella ATT e se quell'elemento non viene acceduto di nuovo entro un certo tempo limite, esso viene rimosso dalla tabella.

Source Computer	Source Computer's IP Address	Source Computer's Port	NAT Router's IP Address	NAT Router's Assigned Port Number
A	192.168.32.10	400	215.37.32.203	1
B	192.168.32.13	50	215.37.32.203	2
C	192.168.32.15	3750	215.37.32.203	3
D	192.168.32.18	206	215.37.32.203	4

Tabella 9.2 - Esempio di Address Translation Table

9.2.4.4 Overlapping

L'Overlapping NAT è necessario in casi molto particolari quando, per qualche motivo, gli indirizzi IP usati sulla rete interna sono indirizzi registrati ma usati in un'altra rete. In tali casi, il Router NAT deve mantenere una tabella (Lookup Table) di questi indirizzi così da poterli intercettare e sostituire con indirizzi IP registrati ed unici.

È importante notare che il Router NAT deve tradurre gli indirizzi interni in indirizzi registrati unici così come deve tradurre gli indirizzi registrati esterni in indirizzi che siano unici nella rete

privata. Per fare ciò si può usare un meccanismo di NAT statico oppure il Domain Name System (DNS) e implementare un NAT dinamico.

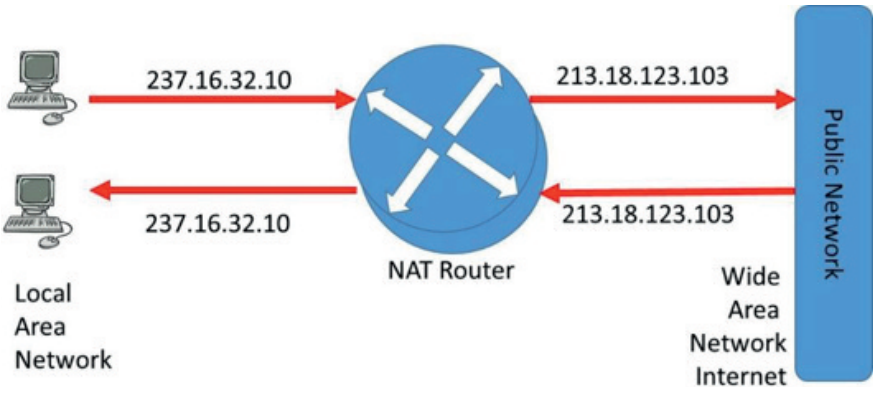


Figura 9.22 - Esempio di Overlapping NAT

Nell'esempio di Overlapping NAT mostrato in Figura 9.22, il range di indirizzi IP interni (237.16.32.xx) è anche un range registrato e utilizzato in un'altra rete, pertanto il Router traduce gli indirizzi per evitare potenziali conflitti con l'altra rete e traduce anche gli indirizzi registrati dell'altra rete in indirizzi non registrati sulla rete locale quando l'informazione viene inviata dall'esterno verso la rete interna.

9.2.5 Domain Name System

Avendolo già citato nel paragrafo precedente e per completezza, brevemente descriviamo il DNS o Domain Name System che è un servizio equivalente a quello dell'elenco telefonico per Internet. Nella rete Internet infatti, per ricordare gli host ed i dispositivi è più comodo usare dei nomi invece dell'indirizzo IP. L'articolazione di tali nomi a dominio è di tipo gerarchico e si parte da quelli che sono definiti come Top Level Domain (TLD) che possono essere di vari tipi: geografico (individuando nazioni) come .it (Italia), .de (Germania), .jp (Giappone), ecc., oppure di categorie come .com (commercial), .net (network), .org (organisation), .gov (government), .edu (education), ecc.. Questi poi hanno dei sottodomini tipicamente quelli delle organizzazioni e soggetti pubblici e privati che hanno ottenuto tale dominio registrato.
Per esempio se si digita www.garr.it nel browser per accedere al server web del GARR la rete delle Università e della Ricerca, la vostra richiesta raggiungerà un server DNS che tradurrà tale nome nell'indirizzo IP corrispondente e che permetterà di raggiungere il sito web richiesto. Questa è la funzione che svolge il sistema DNS.

9.3 *Rete Locale e protocolli di Livello 2*

Vediamo di esaminare ora i meccanismi sottostanti al protocollo IP che, come abbiamo visto, lavora al Livello 3 della pila ISO/OSI. Ci interessa approfondire cosa accade negli strati inferiori dello stack dei protocolli di rete. Come abbiamo accennato nei paragrafi precedenti, per controllare se un indirizzo di destinazione è sulla rete locale lo strato IP utilizza la Network Mask relativa ad ogni rete collegata ad una interfaccia fisica. Se su una macchina Windows diamo il seguente comando su una finestra di terminale: >netstat –r.
Otteniamo una stampa sul terminale del tipo:

Indirizzo rete	Mask	Gateway	Interface	Metric
127.0.0.0	255.0.0.0	127.0.0.1	127.0.0.1	1
131.154.10.0	255.255.255.0	131.154.10.254	131.154.10.10	1

Questo comando ci ha dato quindi informazioni sulle reti a cui è collegato il nostro computer. Se volessimo raggiungere l'indirizzo 131.154.20.21, poiché tale indirizzo appartiene alla rete 131.154.20.0 e questa rete non è fra quelle elencate nelle interfacce, essa non può essere raggiunta direttamente. Ci si rivolgerà, dunque, al router/gateway (131.154.10.254) che provvederà ad instradare i pacchetti a destinazione usando le sue tabelle di routing.
Nel caso si volesse raggiungere lo host 131.154.10.150, si può notare che esso appartiene alla rete 131.154.10.0 che vediamo essere direttamente connessa su una interfaccia. Pertanto lo host in questione dovrebbe essere direttamente raggiungibile.
A questo punto esaminiamo come si invia un pacchetto a destinazione su una rete locale. Per raggiungere lo host di destinazione, il pacchetto IP è, a sua volta, incapsulato all'interno di un protocollo di Livello 2 nella pila OSI (Data Link Layer - DLL) per poter essere trasmesso sul mezzo fisico (Livello 1 della pila ISO/OSI). Il DLL dipende dal mezzo (esempio: Ethernet, Linea Seriale, ecc.) e pertanto può variare a seconda del tipo di rete locale su cui sono collegati i computer che vogliono comunicare fra loro. A Livello 2 si stabilisce anche il limite del pacchetto fisico da inviare (es: 512 Bytes o 1500 Bytes) detto anche MTU (Maximum Transmission Unit).
In applicazioni di trasferimento dati per DAQ la rete più usata è Ethernet, ma è possibile utilizzare anche altre reti con o senza un protocollo di alto livello (TCP/IP).
Esistono implementazioni di trasferimento dati che usano direttamente il protocollo di Livello 2 perché non vi sono necessità di Routing ed invece sono determinanti i problemi di latenza che il protocollo di alto livello (TCP) introduce.

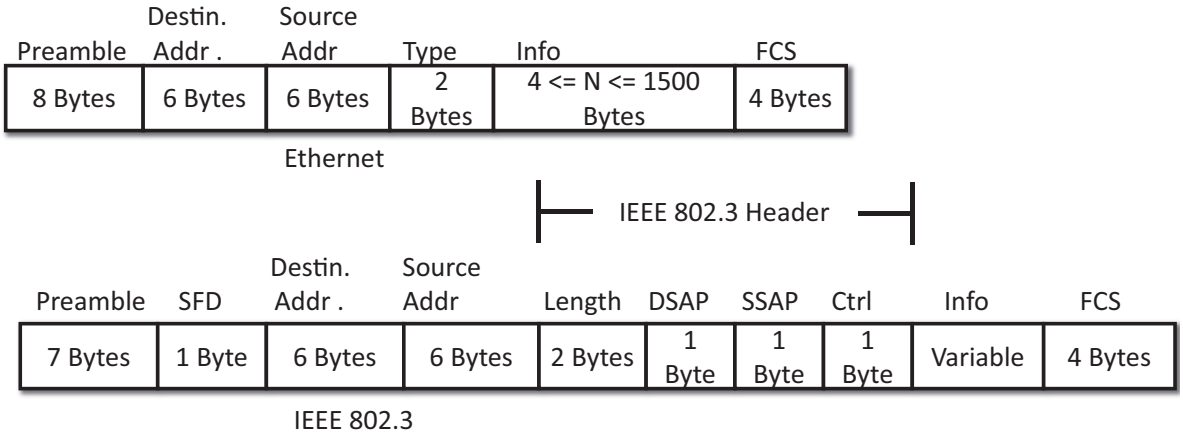


Figura 9.23 - Struttura del pacchetto Ethernet /IEE 802.3

Nella Figura 9.23 sono mostrate, a titolo di esempio, le strutture del pacchetto Ethernet, sia nella sua versione originale che in quella standard IEEE 802.3 e nella prima, in alto, vale la pena di notare come il secondo e terzo campo da sinistra contengano rispettivamente l'indirizzo di destinazione (Destination Address) e quello di sorgente (Source Address) di 6 byte ciascuno, mentre il penultimo campo (Info) è quello che contiene la parte dei dati del pacchetto che può essere, a sua volta, un pacchetto IP. La grandezza di 6 byte dei due campi di indirizzi ci rende evidente che si tratta di una diversa tipologia di indirizzi rispetto a quella IP (4 byte) ed infatti questo tipo di indirizzi va sotto il nome di Medium Access Control (MAC) Address.

9.3.1 MAC Address

Il MAC Address è un indirizzo hardware codificato all'interno dei dispositivi di rete che identifica in maniera univoca ogni nodo della rete. All'interno di un host ogni interfaccia di rete ha un indirizzo MAC unico che la identifica.

Figura 9.24 - Struttura generica del pacchetto MAC

Il protocollo MAC è usato per fornire il Data Link Layer all'Ethernet nelle reti locali di comunicazione (Local Area Network - LAN) e ha una generica struttura di pacchetto rappresentata in Figura 9.24, simile a quella rappresentata nella parte alta della Figura 9.23, dove vediamo che la struttura dati dei protocolli superiori (Payload Data) viene incapsulata aggiungendo un Header di 14 byte prima dei dati e una coda di 4 byte (32 bit) che contiene il Cyclic Redundancy Check (CRC) per il controllo di integrità del contenuto del pacchetto contro possibili errori di trasmissione.
Gli indirizzi di source e destination che hanno una lunghezza di 6 byte (48 bit) hanno comunemente un formato di rappresentazione in esadecimale di 6 campi separati dai doppi punti come nell'esempio mostrato in Figura 9.25 dove è evidenziato che il gruppo dei primi tre campi (3 byte = 24 bit) è usato per identificare il costruttore del dispositivo. Il MAC Address è usato in molti servizi di rete locale che hanno bisogno di conoscere l'indirizzo hardware del dispositivo di rete come: DHCP, ARP, BOOTP che esamineremo nei prossimi paragrafi.

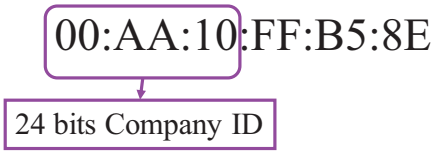


Figura 9.25 - Esempio di MAC Address in formato esadecimale

Nel prosieguo, per semplicità, ci limiteremo a far riferimento alla struttura del pacchetto di tipo Ethernet ed al MAC Address relativo.

9.3.2 Address Resolution Protocol (ARP)

Nel Par. 9.3 abbiamo introdotto l'argomento di come sia necessario in una rete locale sapere come raggiungere un destinatario a partire dal suo indirizzo IP. Per dare una risposta a questa necessità è stato creato lo Address Resolution Protocol (ARP) originariamente specificato nello IETF (Internet Engineering Task Force) Standard RFC 826: "Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48 bit Ethernet address for transmission on Ethernet hardware".
Perché due dispositivi su una stessa rete possano comunicare, è necessario che essi conoscano i rispettivi indirizzi fisici (MAC Addresses) e per ottenerli un host può inviare in rete in Broadcast un pacchetto secondo il protocollo ARP che ha la generica struttura mostrata in Figura 9.26.

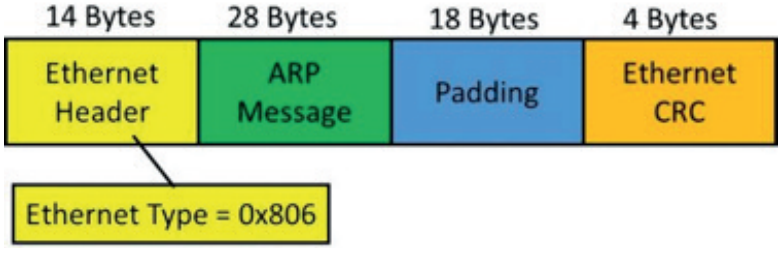


Figura 9.26 - Struttura del pacchetto ARP

Si tratta di un pacchetto Ethernet particolare che contiene, secondo un formato specifico, un messaggio di richiesta (ARP Request) del tipo:

"chi è X.X.X.X ? – rispondete a Y.Y.Y.Y"

dove X.X.X.X e Y.Y.Y.Y sono degli indirizzi IP. Il messaggio viene inviato sulla rete Ethernet con un indirizzo broadcast (FF:FF:FF:FF:FF:FF) e un tipo di protocollo Ethernet specifico con valore 0x806 (in esadecimale) nello header.

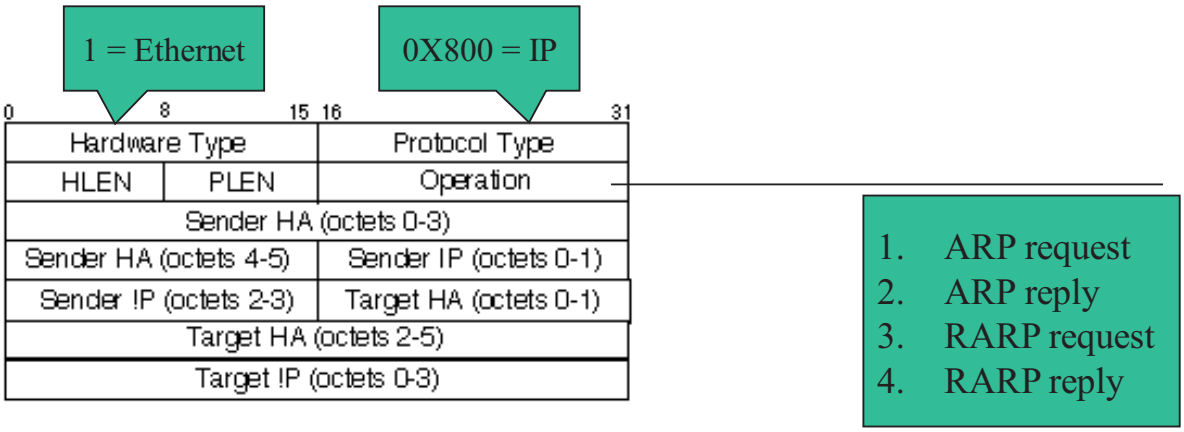


Figura 9.27 - Struttura del Messaggio ARP

La struttura generica dei messaggi di tipo ARP è mostrata in Figura 9.27.
Poiché il messaggio è inviato in broadcast, viene ricevuto da tutti i sistemi nella rete locale (LAN) per assicurarsi che il dispositivo o host cercato, se è connesso in rete, riceva la richiesta. Soltanto il sistema oggetto della ricerca risponderà, mentre gli altri sistemi sulla rete scarteranno il pacchetto senza rispondere perché non è destinato a loro.

```
>arp -a
Interfaccia: 192.168.10.194 --- 0x3
  Indirizzo Internet    Indirizzo fisico      Tipo
  192.168.10.12         00-60-b3-17-f0-97     dinamico
  192.168.10.13         00-00-74-91-2d-01     dinamico

>ping 192.168.10.254
Esecuzione di Ping 192.168.10.254 con 32 byte di dati:
Risposta da 192.168.10.254: byte=32 durata<1ms TTL=64
.....
>arp -a
Interfaccia: 192.168.10.194 --- 0x3
  Indirizzo Internet    Indirizzo fisico      Tipo
  192.168.10.12         00-60-b3-17-f0-97     dinamico
  192.168.10.13         00-00-74-91-2d-01     dinamico
  192.168.10.254        00-02-a5-46-00-44     dinamico
```

Figura 9.28 - Esempio di funzionamento della Tabella ARP

Il sistema cercato, risponderà con un altro messaggio (ARP Response) del tipo:

"X.X.X.X è hh:hh:hh:hh:hh:hh"

dove hh:hh:hh:hh:hh:hh è il proprio indirizzo Ethernet (MAC), cioè quello del computer con indirizzo IP X.X.X.X e questo pacchetto è inviato come unicast solo al computer che ha inviato la richiesta (Y.Y.Y.Y) che ha incluso il suo indirizzo MAC nella richiesta nei 6 byte del campo Sender HA (Figura 9.27).Il computer, dopo aver ottenuto l'indirizzo MAC corrispondente all'indirizzo

IP cercato, lo registra in una tabella ARP per usi futuri evitando di ripetere la richiesta se dovrà nuovamente comunicare con l'altro host. La tabella ARP è dinamica, non solo nell'introduzione di una nuova coppia di indirizzi MAC ed IP, ma anche nella loro cancellazione se, in futuro, non si ottenesse risposta dall'altro Host entro un tempo prefissato.
Nella Figura 9.28 è mostrato un esempio di funzionamento della tabella ARP su un sistema Windows in cui nella finestra di terminale si dà il comando "arp –a" che provvede a stampare l'intero contenuto della tabella ARP in quel momento. Si procede poi con il comando "ping 192.168.10.254" a cercare di contattare lo host con indirizzo IP 192.168.10.254 che dovrebbe essere sulla stessa rete e si ottiene una risposta. A questo punto tramite il meccanismo ARP si è già trovato l'indirizzo fisico (MAC) ed un nuovo comando "arp –a" permette di stampare la tabella che contiene una nuova entry relativa all'indirizzo IP appena trovato.
In aggiunta al meccanismo ARP ne esiste un altro che è il suo inverso Reverse Address Resolution Protocol (RARP) che è usato per mappare l'indirizzo MAC a quello IP e che trova uso, ad esempio, nelle Workstation senza disco che non sanno qual è il loro indirizzo IP quando devono fare il Boot da un altro computer.
In molti casi i processori di acquisizione in un sistema DAQ possono essere diskless e non avere un sistema operativo a bordo perché la mancanza di un disco con le sue parti meccaniche rende la macchina più affidabile.
In altri casi lo stesso sistema operativo deve essere caricato su più macchine allo stesso tempo.
Sono necessari perciò dei servizi di rete per poter assegnare un indirizzo IP alle macchine (DHCP) e per poter fare il BOOT del sistema operativo da un server (BOOTP). Esamineremo questi servizi nei prossimi paragrafi.

9.3.3 Dynamic Host Resolution Protocol (DHCP)

Il Dynamic Host Resolution Protocol (DHCP) è uno standard IETF RFC 2131 e permette ad un server di rete di assegnare un indirizzo IP ad un Host che lo richieda.
L'indirizzo può essere assegnato sia in maniera statica è cioè ad un indirizzo MAC corrisponderà sempre lo stesso indirizzo IP, sia dinamica, in cui l'indirizzo IP viene preso temporaneamente da un pool di indirizzi a disposizione.

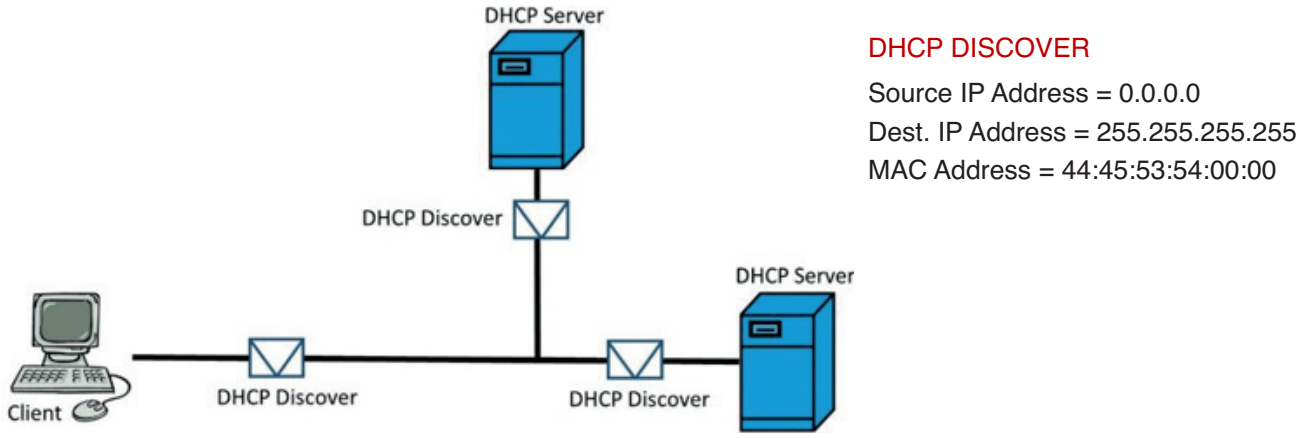


Figura 9.29 - Esempio di DHCP Discover

L'assegnazione può comunque essere condizionata ad una lista di MAC Address abilitati all'uso del servizio che verrà negato a tutti gli altri.
Inizialmente come mostrato in Figura 9.29 il client configura 'sommariamente' il TCP/IP per ricevere automaticamente un IP dal server DHCP.

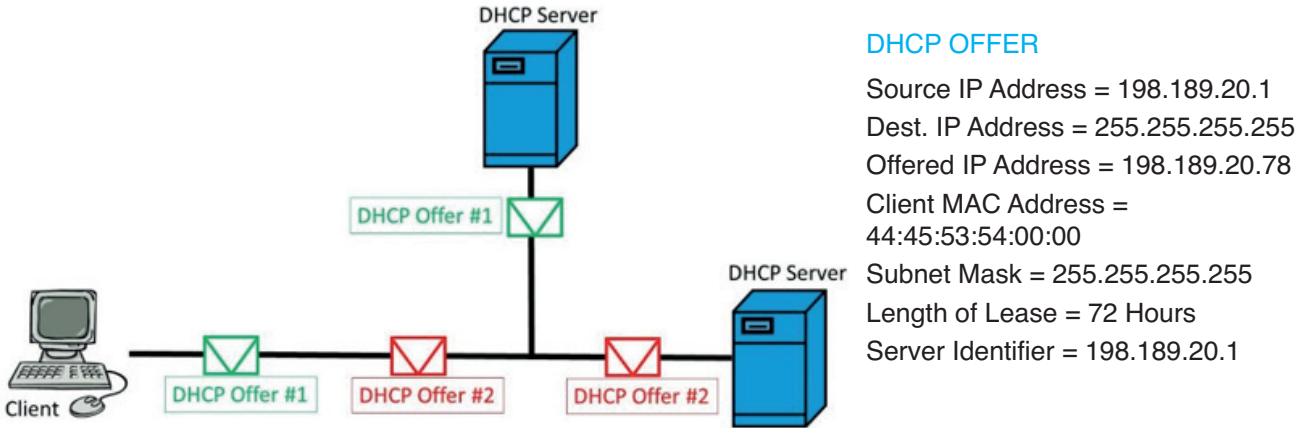


Figura 9.30 - Esempio di DHCP Offer

La richiesta fatta dal client viene fatta all'indirizzo broadcast, poiché l'indirizzo IP del server è sconosciuto, la richiesta avrà quindi come mittente 0.0.0.0 (poiché l'IP non è stato ancora configurato) e 255.255.255.255 (broadcast) come destinazione; la richiesta (DHCPDISCOVER) contiene inoltre l'indirizzo MAC della scheda di rete e il nome del computer, questo per essere identificato.
Il server DHCP, come mostrato in Figura 9.30, manda un messaggio (DHCPOFFER) broadcast contenente l'indirizzo IP. Il client utilizzerà il primo IP che riceverà, nel caso che ci siano più server DHCP sulla rete, gli altri verranno ignorati.

Dopo aver accettato un IP, il client manda un messaggio (simile a DHCPREQUEST) broadcast informando tutti i server DHCP che ha accettato un IP. Il messaggio include l'indirizzo del server DHCP che ha mandato l'IP che è stato accettato; tutti gli altri server ritirano le loro offerte.

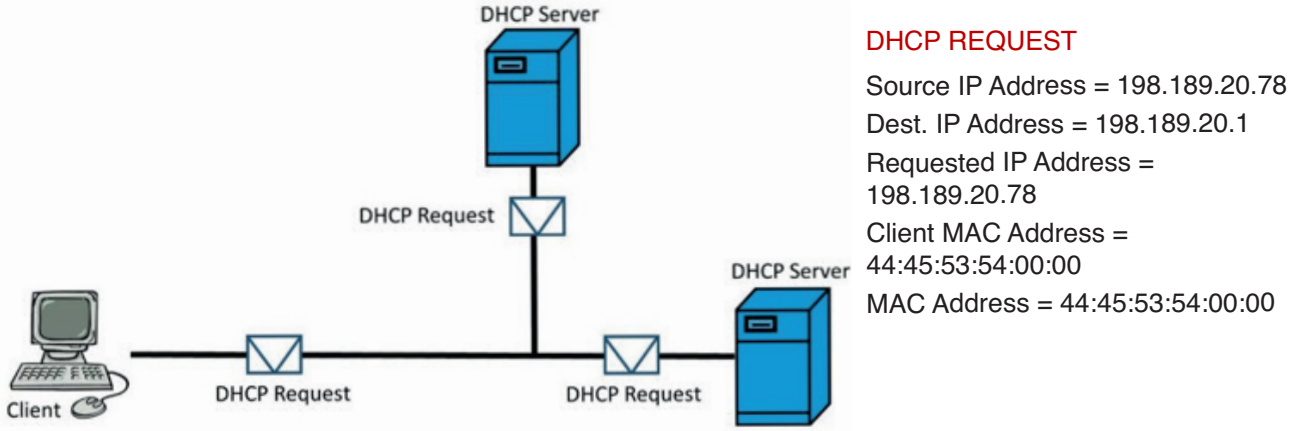


Figura 9.31 - Esempio di accettazione Indirizzo IP

Il server DHCP manda quindi un messaggio di conferma (DHCPACK) al client, contenente il valore della durata di assegnazione (Lease) per l'IP. Quando il client riceve il messaggio di ACK completa la configurazione del TCP/IP.

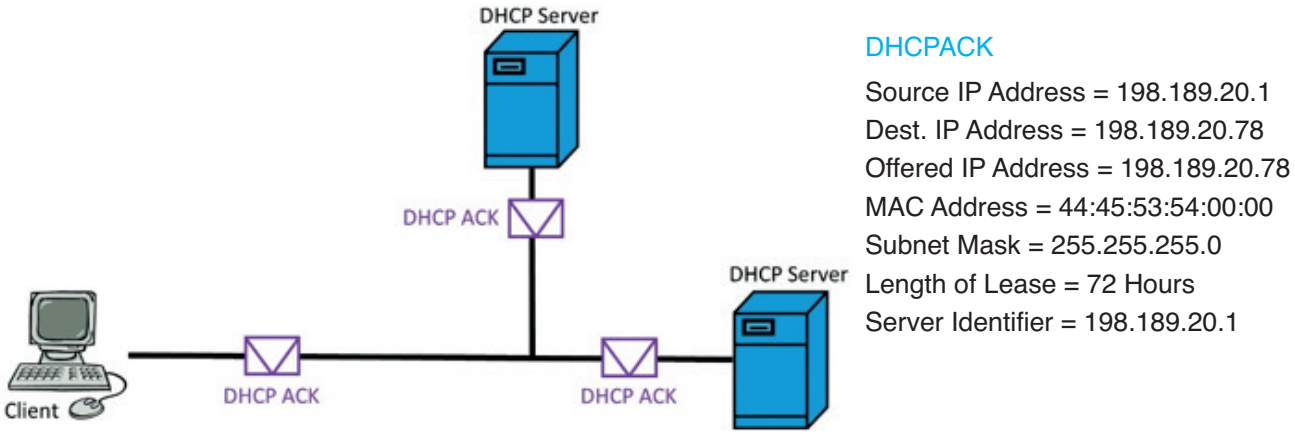


Figura 9.32 - Esempio di DHCP ACK

9.3.4 Bootstrap Protocol (BOOTP)

Il Bootstrap Protocol (BOOTP) è uno standard IETF RFC 951 e RFC 1542 che permette a un client di effettuare il bootstrap del sistema operativo da un server attraverso la rete.
Il Bootstrap Protocol fornisce un metodo per una stazione senza disco fisso di compiere un bootstrap ottenendo le informazioni necessarie al boot da un server di rete. I messaggi BOOTP sono contenuti in datagrammi UDP ed il formato è comune ai due tipi di messaggio, Richiesta e Risposta.

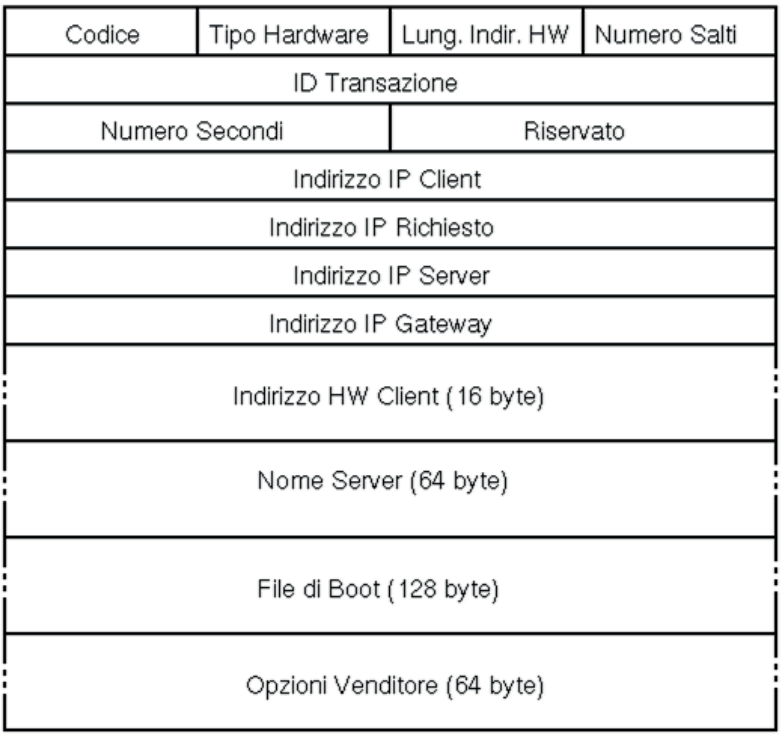


Figura 9.33 - Struttura del pacchetto BOOTP

Nella Figura 9.33 è mostrata la struttura del pacchetto BOOTP. Dove i vari campi sono descritti nel seguito.

– Il campo Codice vale 1 per una Richiesta e 2 per un Responso
– Il campo Tipo Hardware è lo stesso del corrispondente campo in una trama ARP, e vale 1 per un collegamento Ethernet a 10 Mbit/sec.
– Il campo Lunghezza Indirizzo Hardware vale 6 per Ethernet

– Il campo Numero Hop è messo a zero dal client e può essere usato da un server proxy
– Lo ID Transazione è un intero inizializzato dal client in modo casuale e serve ad identificare il collegamento specifico tra client e server in presenza di più collegamenti simultanei nella rete. Il server pone lo stesso identificativo nel messaggio Responso
– Il campo Numero Secondi viene messo dal client ad un valore determinato dall'implementazione. I server BOOTP in rete esaminano questo campo, e i server secondari intervengono allo scadere di questo tempo se non è stato inviato alcun responso da parte del server primario
– Se il client conosce il proprio Indirizzo IP allora riempie il campo corrispondente, altrimenti lo pone a zero. Se il campo è zero, il server pone l'Indirizzo IP giusto nel Responso nel campo Indirizzo Richiesto.
– Il campo Indirizzo Server è inizializzato dal server che risponde. Se il server è su un'altra rete ed è in uso un server proxy, questi riempie il campo Indirizzo Gateway
– Il client riempie il campo Indirizzo Hardware. Questo campo è allineato a 32 bit, eventualmente ponendo al termine un riempimento di zeri.
– Il server pone il proprio nome nel campo Nome Server come stringa terminata da un NUL e, se applicabile, il Nome File di boot, anch'esso come stringa. Il client successivamente può eseguire il download del file di boot con un altro protocollo, p.es TFTP.
– Il campo Opzioni Venditore può contenere informazioni aggiuntive specifiche dell'implementazione.

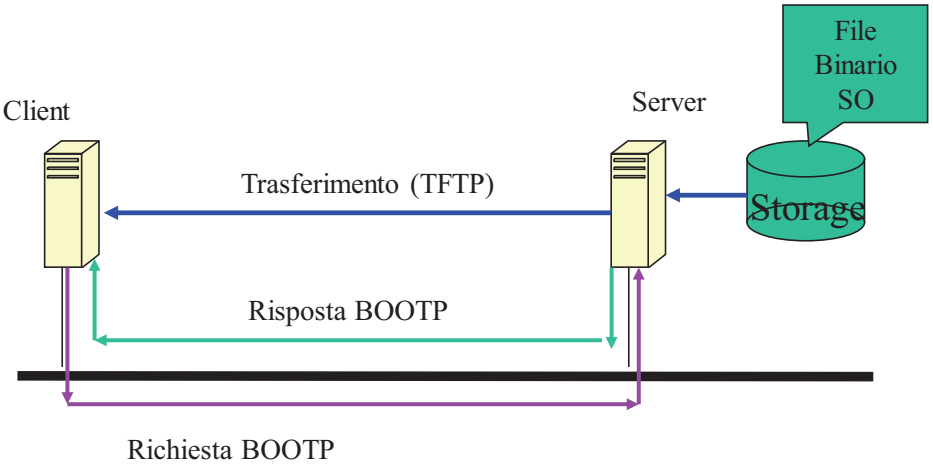


Figura 9.34 - Schema di funzionameto del BOOTP

Come mostrato in Figura 9.34, la Richiesta BOOTP è trasmessa all'indirizzo di broadcast locale 255.255.255.255, con indirizzo mittente posto a 0.0.0.0 poiché il client spesso non conosce il proprio indirizzo. Per quel che riguarda i numeri di porta, BOOTP usa la porta 67 per il server e 68 per il client. Il client, dunque, non usa una porta effimera e la ragione è che i server trasmettono il messaggio Responso in modo broadcast e vi potrebbero essere interferenze con altri servizi di altre stazioni che usassero la stessa porta effimera del client richiedente.
Ogni client riceve tutti i Responsi BOOTP, ma riesce a distinguere il suo tramite il campo ID Transazione del messaggio.

Capitolo 10

Data Acquisition

Possiamo elencare i compiti di un sistema di Acquisizione Dati come un percorso che va dal detector fino all'archiviazione di lungo termine e quindi schematicamente distinguiamo le seguenti fasi:

– Lettura dei dati dalla Elettronica di Front-End e formazione di un sub-evento relativo ad uno specifico rivelatore o sottoinsieme di esso (Sub-Event Building);
– Assemblaggio dell'evento completo raccogliendo i dati dei Sub-Event (Event Building)
– Processamento degli eventi e ulteriori selezioni (L3 o L4);
– Registrazione degli eventi su sistemi di archiviazione Disco o Nastro Magnetico.

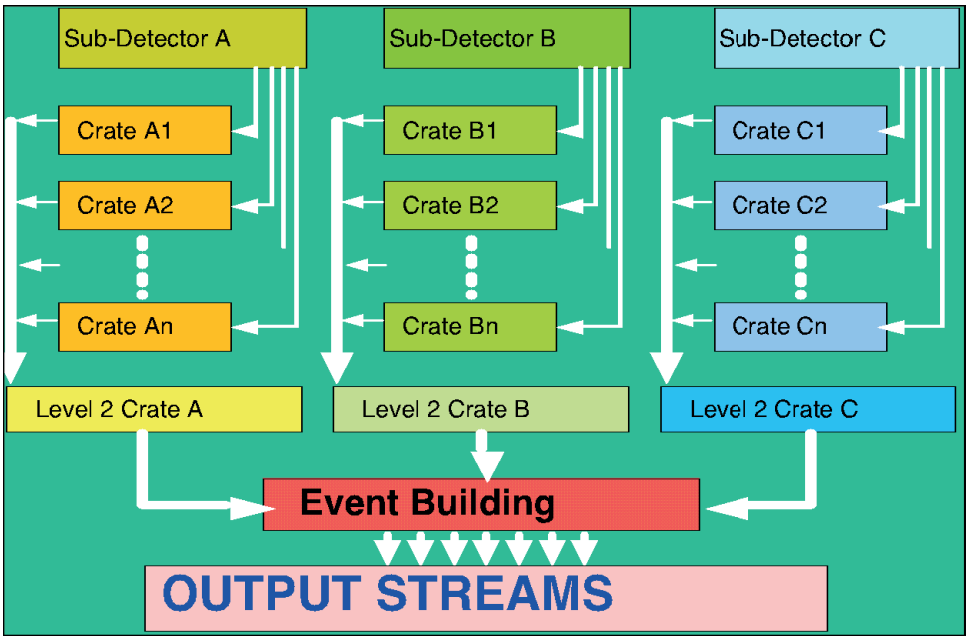


Figura 10.1 - Schema DAQ con sotto-rivelatori ed Event Building

Tali processi avvengono in maniera asincrona rispetto al Beam Crossing nel caso di esperimenti a Collider ma in generale anche in esperimenti a Bersaglio Fisso (Fixed Target).
Le scelte Hardware e Software per tutte le componenti necessarie possono essere comuni a tutti i sotto-rivelatori ma non sempre ciò accade. Quello dove è necessario uniformità è sul formato dei dati ed il modello del loro trasferimento verso il sistema centrale di Event Building.
È anche possibile individuare altre componenti all'interno di un sistema di acquisizione dati che sono mostrati nella Figura 10.2 e che completano l'architettura del sistema. Alcuni (Front-End,

Event Building, ecc.) sono stati già citati e/o verranno approfonditi nei paragrafi successivi. Ci soffermiamo rapidamente a descrivere gli altri elementi della figura che non avranno una trattazione approfondita nel seguito.
Lo Slow Control (SC) è un sistema di controllo che normalmente lavora in parallelo con altre componenti come DAQ e Trigger e può interagire con essi. In particolare lo SC si occupa di gestire e monitorare tutti quei dispositivi hardware e software che permettono al rivelatore ed all'esperimento di funzionare in maniera adeguata od ottimale. Lo SC, infatti, legge lo stato di quei sistemi che cambiano solo lentamente nel tempo come Alimentatori di Alte e Basse Tensioni, Sensori di Temperatura e Pressione, Campo Magnetico, ecc. ed in molti casi è programmato per operare su tali sistemi e non solo per monitorarli.
Il Logbook è un elemento software indispensabile all'interno di un esperimento. Originariamente realizzato con un vero e proprio libro su cui annotare le informazioni relative alla situazione del rivelatore, del DAQ, del Trigger e dell'esperimento (Tipi di Run di acquisizione, Trigger, osservazioni delle persone in turno, ecc.) è poi stato sostituito con una applicazione che registra le informazioni su un opportuno DataBase.

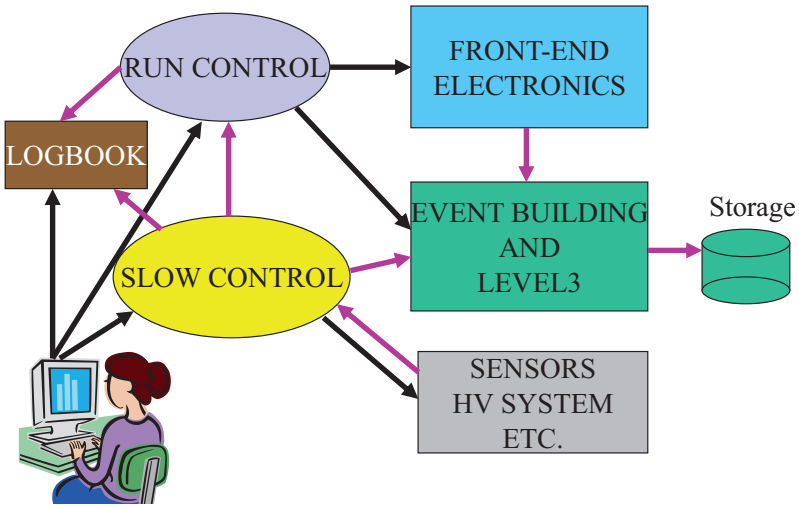


Figura 10.2 - Schema di un DAQ con i vari componenti

10.1 *DAQ di Livello 2*

Nella figura seguente è schematizzato un sotto-sistema di DAQ di Livello 2 in cui i dati sono letti direttamente dall'Elettronica di Front-End (non c'è dunque un Livello 1 specifico) tramite un Bus VME verso un Processore che ha in esecuzione al suo interno una serie di componenti software.

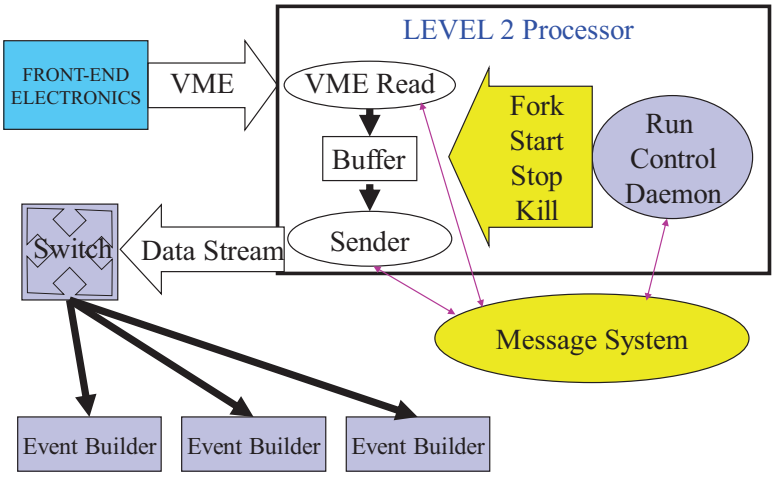


Figura 10.3 - Schema di un sotto-sistema di DAQ di Livello 2

– Un programma di lettura del VME e dell'elettronica del sotto-rivelatore (in gergo informatico un tale task viene chiamato Producer);
– Un Buffer di memoria in cui viene trasferito il sub-evento letto e che sarà letto dal successivo task Sender in maniera da costruire una semplice pipe-line;
– Un task (Sender) che, dopo una opportuna formattazione e/o eventuale selezione del sub-evento, lo invia ad una rete verso il sistema di Event-Building (in gergo informatico il task in questione è chiamato Consumer);
– Un task (Run Control Daemon) che colloquia, attraverso un sistema di messaggi con il sistema centrale di Run Control e che impartisce i comandi per creare gli altri task, farli partire, metterli in pausa, distruggerli, ecc.

10.2 *Event Building*

La raccolta dei sub-eventi forniti dalle varie parti dei sub-detector attraverso il Livello 2, viene effettuata su un sistema di processori che va sotto il nome di Event-Builder.

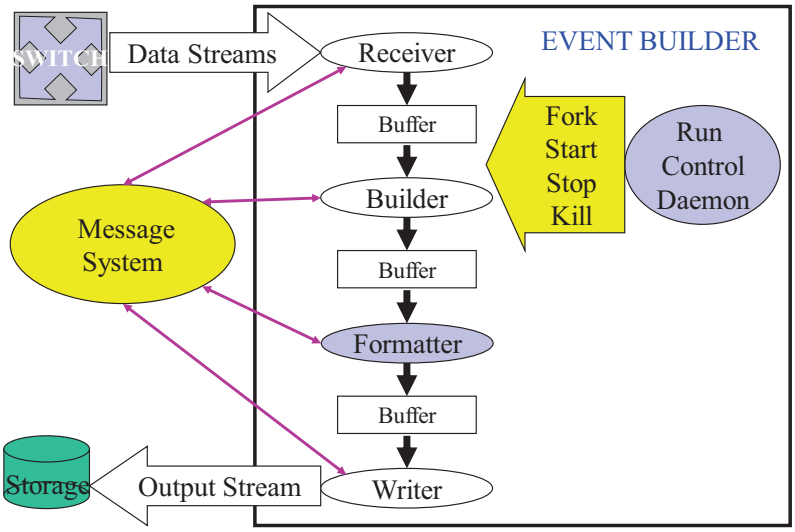


Figura 10.4 - Schema di un Event Builder

Nella Figura 10.4 è rappresentato un possibile schema semplificato di Event Builder in cui, all'interno di un sistema di calcolo dedicato allo scopo, viene costruita una pipe-line che legge i sub-eventi provenienti dai vari sub-detector (Receiver) e li rende disponibili ad un task che li mette insieme (Builder) per formare un evento completo che poi, a sua volta sarà formattato opportunamente (Formatter) e spedito allo stadio successivo (Writer) che potrà essere un Livello 3 oppure un sistema di Storage.
Fra ciascuno di questi stadi e il successivo è interposto un Buffer di Memoria per la derandomizzazione.
Sono altresì presenti, come nel caso del Livello 2 le componenti di Message System e quella di Run Control Daemon.

10.3 *Readout Networks*

Il sistema di rete di lettura dei dati è un elemento fondamentale, sia per il Livello 2 che per quelli successivi (Event Building e/o Livello 3) e nel corso degli anni si sono succeduti numerosi standard utilizzati nell'ambito degli esperimenti di fisica delle alte energie. Lo schema prevalente è quello che vede una catena che parte dalla elettronica di Front-End e poi si concentra verso dei Readout Controller all'interno di Crate (Contenitori di cassetti di elettronica) e da questi ultimi i dati vengono ulteriormente raccolti attraverso dei Crate Interconnect come è mostrato in Figura 10.1. All'interno dei Crate o come interconnessione fra essi sono stati utilizzati nel tempo dei Bus di vario tipo e prestazioni come, ad esempio:

– CAMAC – Uno dei primi Bus standard utilizzati con una robusta costruzione meccanica e una grande varietà di moduli di elettronica che nel tempo sono stati sviluppati e prodotti a livello industriale ma con una velocità del Bus interno al Crate piuttosto limitata (0,5 – 2 Mbytes/s);
– Fastbus – Uno standard sviluppato per aumentare la velocità di trasferimento dati all'interno del Crate (40 Mbytes/s) e dell'interconnessione fra Crate (4 Mbytes/s) con una scheda elettronica più ampia per una maggiore integrazione di numero di canali ma che aveva alcune scelte meccaniche non sufficientemente robuste (flessione delle schede e connettore con i pin dal lato del Backplane) ed una limitata disponibilità di moduli e di diffusione industriale;
– VME – Uno standard industriale molto longevo con una grande diffusione sia di produttori che di tipologie di schede.

Una tipologia di interconnessione che abbiamo già visto nei paragrafi precedenti è quella di tipo Switch che si è sviluppata in ambito di reti di comunicazione e che ha un ampio numero di produttori, di velocità di collegamento e di caratteristiche.
Le tecnologie degli switch, come per esempio quella a Crossbar, permette di avere comunicazioni e trasferimenti simultanei fra più coppie di porte indipendenti.
Il protocollo e la tecnologia più comunemente usata negli switch commerciali è Ethernet, ma ci sono realizzazioni in altri standard di comunicazione come Fibre Channel e Infiniband che esamineremo nel seguito.
Gli switch attuali prevedono il supporto ai molti standard di velocità di Ethernet dal FastEthernet a 100 Mbit/s o Gigabit Ethernet (1 Gb/s) ai più recenti e performanti 25 Gb/s e 100 Gb/s.
L'uso degli switch commerciali nell'ambito di un sistema di acquisizione dati, richiede alcune considerazioni su alcune possibili problematiche come:

– Il livello di uso dei singoli link che afferiscono allo switch;
– Il possibile collo di bottiglia (bottleneck) sulle porte di uscita;

– Il grande numero di porte necessario nel caso di sistemi con un gran numero di dispositivi da acquisire e di processori.

Anche l'elemento di costo gioca un ruolo importante nella scelta dello switch e della sua tecnologia visto l'alto numero di porte richiesto.

10.4 *Data Mover*

Il processo di Event Building può produrre dati che vengono archiviati direttamente su dischi e questo li rende disponibili, sia per una successiva rielaborazione, sia per poterli trasferire su un sistema di storage più affidabile come quello dei Nastri Magnetici. È quindi prevedibile che in questo caso sia necessario un ulteriore sistema di trasferimento dati che possiamo definire come Data Mover. La figura seguente ne illustra lo schema generale che non si discosta da quelli già visti, se non per il fatto che questo sistema lavora in maniera del tutto off-line rispetto la DAQ vero e proprio, ma lo complementa con una funzione di archiviazione dati su Nastri Magnetici che sono supporti più affidabili nel tempo e con un minore costo per Pbyte rispetto ai dischi magnetici tradizionali.

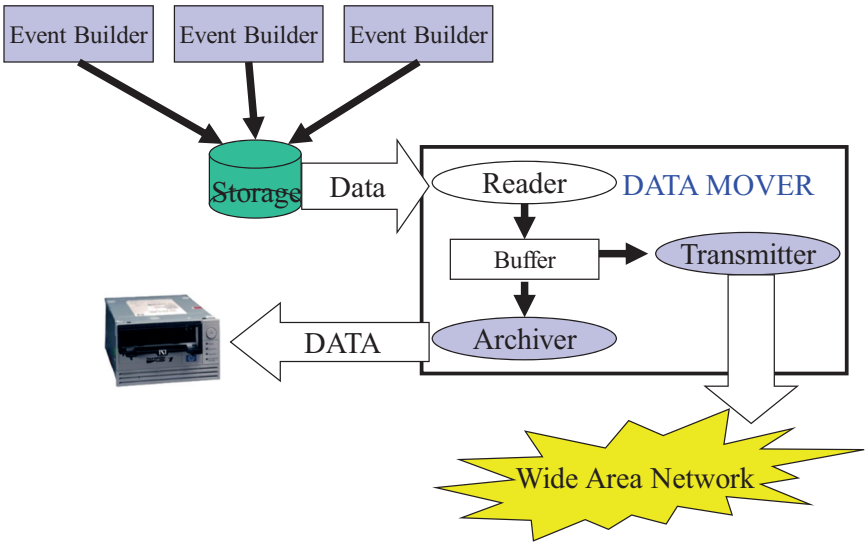


Figura 10.5 - Schema di un Data Mover

A conclusione di questa prima parte del Capitolo possiamo dunque ricavare uno schema generale di trasferimento dati all'interno dei sistemi DAQ ma anche di quelli quasi On-Line che è mostrato in Figura 10.6.

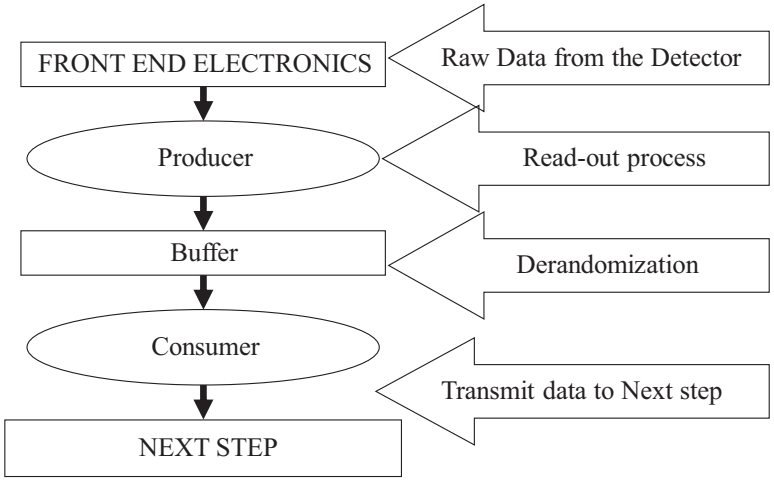


Figura 10.6 - Schema generale di un Trasferimento Dati

10.5 *Run Control*

Una parte importante di un sistema DAQ è legata al controllo del sistema stesso e tale compito viene generalmente assolto da un componente software (un programma) che va sotto il nome generico di Run Control.
Esso è generalmente composto da più task e sotto-programmi che gli permettono di esercitare le sue funzioni che sinteticamente elenchiamo:

– Configurare il sistema permettendo di scegliere l'insieme dei sub-detector da coinvolgere e dell'elettronica ad essi collegata, il tipo o i tipi di Trigger da utilizzare, le condizioni di presa dati, il numero di eventi da acquisire in ciascun Run, dove archiviare i dati, ecc.
– Inizializzare le componenti HW e SW, che sono state precedentemente selezionate nella fase di configurazione, verificandone la presenza e rispondenza alla configurazione richiesta;
– Attivare e Disattivare l'acquisizione facendola partire e terminandola perché sono state raggiunte certe condizioni oppure perché è stato richiesto dall'Operatore o ancora mettendola in pausa se vengono riscontrati errori di tale gravità da compromettere l'acquisizione corretta dei dati;
– Controllare che il flusso dei dati non si sia bloccato o abbia un malfunzionamento;
– Gestire gli Errori gravi e/o fatali
– Chiudere il sistema in maniera "pulita" non lasciando task o hardware attivi;
– Permettere a degli operatori di tenere sotto controllo tutto il sistema;
– Interagire con gli altri sistemi come, ad esempio, quello di Trigger.

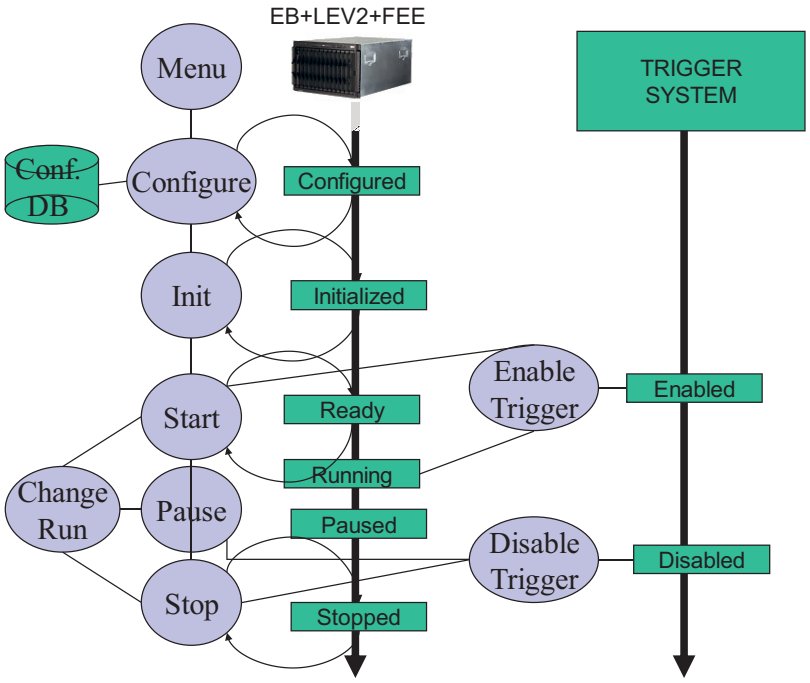


Figura 10.7 - Schema di Macchina a Stati per un Run Control

In generale il sistema DAQ e il suo RUN Control sono strutturati come una macchina a stati così come mostrato in Figura 10.7.
Ogni operazione deve portare il sistema in uno stato ben definito e questo significa che il passaggio fra uno stato e l'altro può avere solo successo o fallire, ma non devono esistere situazioni indefinite in cui una parte del sistema è riuscito a transire al nuovo stato e un'altra parte no.

Il processo di acquisizione viene perciò suddiviso in una serie di passi logici che devono essere compiuti l'uno dopo l'altro ed esistono regole precise che permettono di passare da una condizione ad un'altra, mentre alcuni passaggi fra stati non vengono permessi.
Nella Figura 10.8 è mostrato lo schema semplificato delle operazioni di Configurazione ed Inizializzazione del sistema di DAQ in preparazione di un Run di Acquisizione Dati. Il Task Configure parte con una lettura dal DataBase delle Configurazioni che stabilisce, in base alle configurazioni previste, quali sono i sub-detector coinvolti e l'elettronica necessaria. Questo permette di preparare delle tabelle (DAQ Tables) che specificano i componenti del sistema di acquisizione. Alla fine del processo di configurazione, se non si sono verificati errori, lo stato del sistema viene passato a Configured.

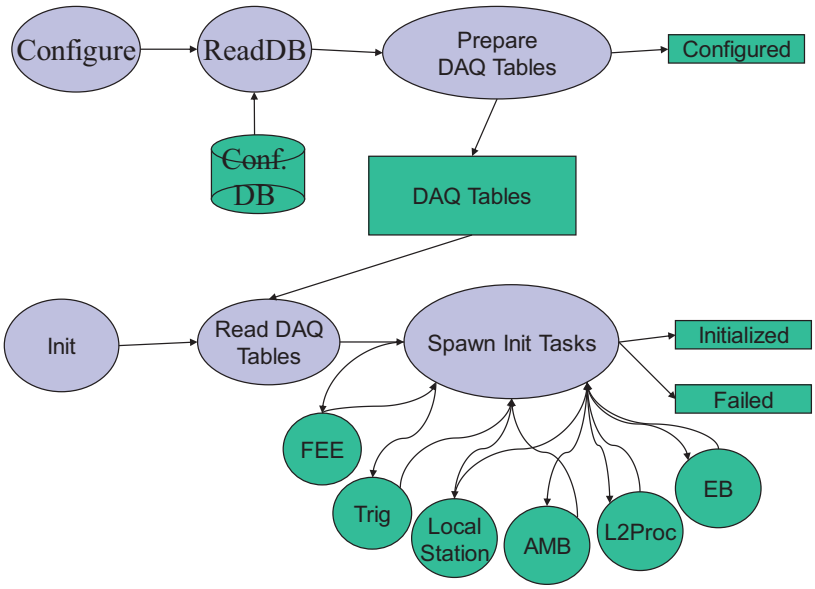


Figura 10.8 - Schema delle operazioni coinvolte nel Run Control

Il passo successivo è quello della Inizializzazione (Init) che parte leggendo dalle DAQ Tables le componenti che sono coinvolte nell'Acquisizione e le inizializza. Questo compito può essere svolto sia in sequenza, componente per componente, oppure, per motivi di velocità, in parallelo facendo partire un insieme di task che si occupano specificatamente di ciascuna tipologia di componenti: l'Elettronica di Front-End, il Trigger, le eletttroniche legate ai sub-detector (Local Station e AMB), i processori di Livello 2, l'Event Builder, ecc. Al termine di tutti i processi che eseguono con successo il proprio compito, lo stato del sistema viene portato ad Initialized. Nel caso che anche uno solo dei task fallisca, la transizione fallisce (Failed) ed a questo punto sono possibili due scelte:

– Si fa un Reset del sistema e si ricomincia il processo di Init tutto da capo con un allungamento sensibile dei tempi di partenza del sistema, oppure
– Si verificano quali sono i Task che hanno fallito e si fanno ripartire solo quelli provando a completare la transizione senza ripartire dall'inizio e, quindi, con minore ritardo.

Quest'ultima soluzione non sempre dà esito positivo se le ragioni di fallimento dei task non vengono rimosse.

Capitolo 11

Farming, Cluster e Grid Computing

Nel precedente Capitolo 1 abbiamo esaminato sia il concetto di Parallelismo che di Pipeline e fatto alcune considerazioni sull'uso di entrambi. Quando si affrontano problematiche che riguardano il computing questi concetti divengono di nuovo rilevanti.

11.1 *High Performance Computing*

Il Calcolo ad alte prestazioni è comunemente associato a parallelismo ed operazioni in Floating Point. Preliminarmente è opportuno individuare due categorie di parallelismo che sono state storicamente alla base di molti computer high performance.

– SIMD (Single Instruction Multiple Data) eseguono una istruzione (anche complessa) in parallelo su più dati, ottenendo in parallelo i risultati; tipico esempio è APE per Lattice QCD
– MIMD (Multiple Instruction Multiple Data) sono macchine parallele più generali, ma generalmente meno efficienti su specifiche problematiche.

Possiamo qui di seguito a fare due esempi di operazioni parallelizzabili nelle due categorie appena viste:

– SIMD: Moltiplicazione di un Vettore M per un numero n: $\mathbf{M'_i = n\ M_i}$ per $\mathbf{i=1,m}$. Permette di replicare fino ad m volte la stessa operazione su più dati diversi.
– MIMD: **A = B + C** ; **D = E x F** ; **G = H / I** ; Sono eseguibili in parallelo da 3 operatori diversi (Somma, Moltiplicazione e Divisione).

Il Calcolo ad alte prestazioni è comunemente associato a parallelismo ed operazioni in Floating Point e nel passato era effettuato su singole macchine (Mainframe) con processori (anche vettoriali) di grandi prestazioni misurate in Flops (Floating Point Operations per Second) vendute da ditte come CRAY, IBM, ecc.
Molte applicazioni sono sensibili alle performance del processore, della banda passante con la memoria e alla banda passante verso altri processori (generalmente i primi vicini).

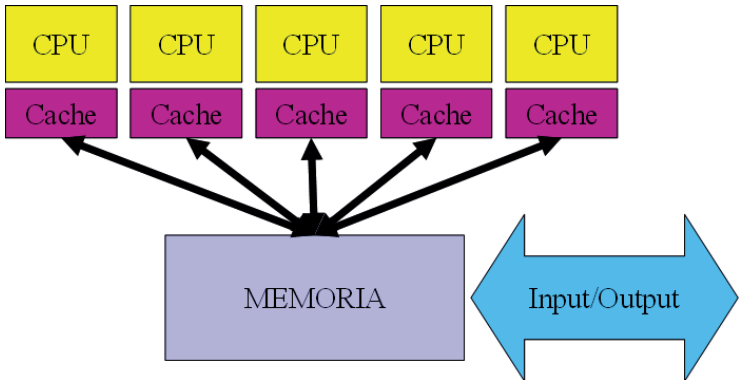


Figura 11.1 - Sistema multiprocessore con memoria condivisa

Gli attuali processori commerciali usano, in generale, architetture di tipo super-scalare, cioè una combinazione di pipelining e moduli funzionali multipli.
Per alcune applicazioni si può disegnare un'architettura più adatta alla soluzione del problema da affrontare.
I super computer attuali hanno un'architettura mista che include, oltre alle CPU anche processori paralleli inizialmente studiati per applicazioni grafiche le GPU (Graphic Processing Unit).

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,299,072	415,530.0	513,854.7	28,335
2	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi, China	10,649,600	93,014.6	125,435.9	15,371
5	Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou, China	4,981,760	61,444.5	100,678.7	18,482
6	HPC5 - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband, Dell EMC, Eni S.p.A., Italy	669,760	35,450.0	51,720.8	2,252
7	Selene - DGX A100 SuperPOD, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia, NVIDIA Corporation, United States	272,800	27,580.0	34,568.6	1,344
8	Frontera - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR, Dell EMC, Texas Advanced Computing Center/Univ. of Texas, United States	448,448	23,516.4	38,745.9	
9	Marconi-100 - IBM Power System AC922, IBM POWER9 16C 3GHz, Nvidia Volta V100, Dual-rail Mellanox EDR Infiniband, IBM, CINECA, Italy	347,776	21,640.0	29,354.0	1,476
10	Piz Daint - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100, Cray/HPE, Swiss National Supercomputing Centre (CSCS), Switzerland	387,872	21,230.0	27,154.3	2,384

Tabella 11.1 - Lista TOP500 di giugno 2020

L'organizzazione TOP500 <www.top500.org> pubblica regolarmente due volte all'anno una lista dei 500 calcolatori più potenti al mondo ed in Tabella n. 11.1 è mostrata la tabella di giugno 2020. La lista viene compilata sulla base di un benchmark effettuato con una applicazione LINPACK per la risoluzione di equazioni lineari. Le performance vengono tipicamente indicate in TFlops (10^{12} Floating Point Operations / Second). Nella lista, ma non nelle prime posizioni, ci sono molti cluster di computer e, in generale, il loro rapporto Prestazioni/Costo è migliore.

11.2 *High Throughput Computing*

Lo High Throughput Computing è, a volte, chiamato "Embarrassingly Parallel Computing" e si riferisce a quei tipi di parallelismo in cui un intero task viene semplicemente replicato su più processori.

Le applicazioni sperimentali di HEP sono basate su strutture dati ad Evento e si processano molti eventi, per cui è più rilevante il numero di eventi processati nell'unità di tempo che il tempo di processamento del singolo evento. La totale indipendenza degli eventi fra di loro permette di processarli separatamente su CPU diverse che elaborano in parallelo. Il livello di efficienza è buono perché non ci sono comunicazioni fra i processori e nessun sincronismo è necessario: ciascuna CPU elabora il proprio evento in maniera totalmente indipendente dalle altre ed il tempo di esecuzione di ciascun evento non è un parametro limitante in prima approssimazione.

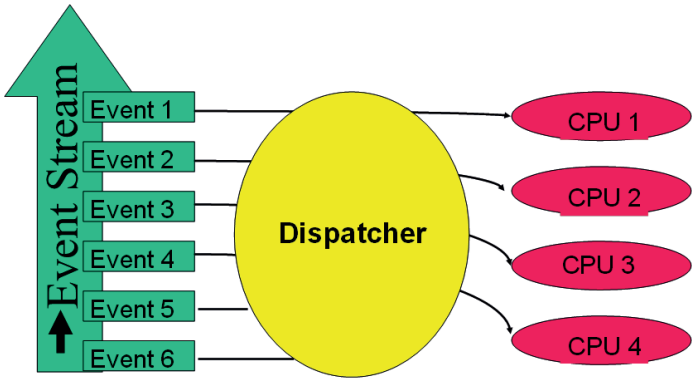


Figura 11.2 - Esempio di HTC nel caso di esperimenti di High Energy Physics

L'insieme di processori che sono usati per questo tipo di parallelismo può essere composto da computer commerciali di costo relativamente modesto con un buon rapporto prezzo/prestazioni. Le architetture sono diverse ma basate su molte similitudini e normalmente si parla di Computer Farm o di Cluster.

11.2.1 Computer Farm

Una Farm di computer è un insieme di calcolatori, a costo limitato o COTS (Commodity Off The Shelf) e cioè un gruppo di macchine più o meno equivalenti, che eseguono compiti simili su tutti i componenti (Nodi).

Una Farm è tipicamente scalabile da poche decine a migliaia di nodi e permette di effettuare un parallelismo ragionevolmente scalabile (di tipo HTC) con bassi costi per un buon numero di applicazioni che non richiedono un vero parallelismo.

Le applicazioni ideali su una Farm sono di tipo High Throughput Computing, mentre applicazioni HPC sono possibili nei casi in cui:

– la comunicazione fra processi è molto scarsa o, equivalentemente, quando: **TP >> Tc**, dove:
  - **TP** = tempo di processamento
  - **Tc** = tempo speso nella comunicazione con altri processi
– Il problema di calcolo è facilmente distribuibile (replica) su più processori in parallelo ed il tempo di calcolo sui processori è equivalente.

Ovviamente i valori sono dipendenti dai processori e dall'evoluzione tecnologica ed architetturale delle macchine di calcolo.

L'efficienza è il rapporto fra il tempo di CPU effettivamente utilizzato e quello teoricamente disponibile.

11.2.2 Cluster

Un cluster è un insieme di computer che condividono risorse comuni per uno o più scopi comuni. In un cluster i nodi sono generalmente interconnessi con una rete molto veloce ed a bassa latenza (spesso non una LAN Ethernet) e in generale un cluster possiede un File System distribuito o permette di rendere disponibile un file-sharing.
Un cluster viene generalmente esposto all'esterno come un sistema unico ed è spesso anche ad alta affidabilità e ridondante.

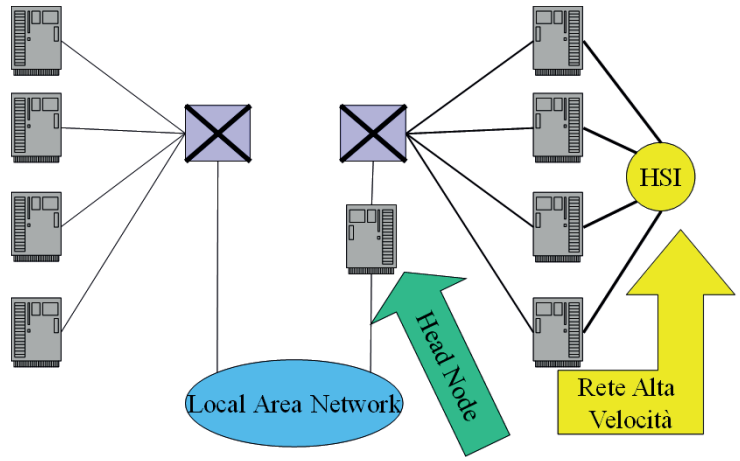


Figura 11.3 -Esempio di architettura Cluster

Come mostrato nella Figura 11.3 una serie di nodi o computer possono essere interconnessi, sia attraverso una rete locale che anche attraverso una rete ad alta velocità a loro dedicata per accesso veloce a dati in memoria o su disco.
Il cluster Beowulf è un antesignano dei cluster di calcolo composti da COTS partito come progetto nei primi anni '90 negli USA.
È composto da una serie di nodi di calcolo interfacciati verso l'esterno da una workstation/server chiamata Head Node che provvede alle attività di controllo del cluster ed a fornire accesso ad esso. Generalmente nella rete interna viene implementata una numerazione IP nascosta con utilizzo di meccanismi NAT (Network Address Translation) verso la rete generale (LAN/WAN/Internet).

11.3 *High Availability Computing*

I cluster sono anche utilizzabili per ottenere alta affidabilità e/o distribuzione del carico. Questo è necessario quando si vogliono avere applicazioni che non possono sopportare malfunzionamenti e dovendo essere sempre in funzione necessitano resilienza (Es. Database, Servizi Web, Server Disco, ecc.).

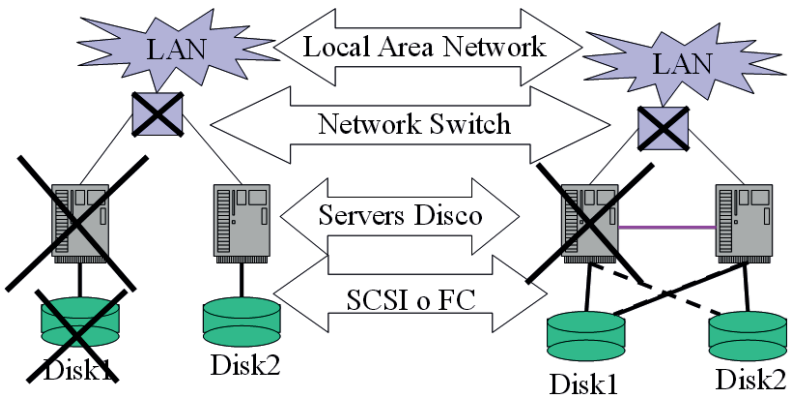


Figura 11.4 - Esempio di Sistema ad Alta Affidabilità

Per ottenere alta affidabilità a livello hardware in generale si punta a realizzare architetture che evitano singoli punti di fallimento replicando tutte le componenti soggette a possibili malfunzionamenti (Dischi, rete, computer). In Figura 11.4 è mostrato un esempio di una tale architettura in cui i dischi serviti da un computer, in caso di fallimento di quest'ultimo non sono più accessibili e quindi è possibile prevedere che ci sia una connessione ulteriore del disco con un altro computer che, non appena rileva la caduta del primo server attraverso un doppio percorso di connessione, è in grado di prendere in consegna tale disco e servirlo verso l'esterno.
A livello applicativo è spesso necessario effettuare delle modifiche per rendere operativa l'alta affidabilità (es. Database replicati, Application Server separati, ecc.).

11.4 *Message Passing*

Per ottenere il trasferimento dei dati attraverso una rete a bassa latenza ed alta banda passante servono protocolli ottimizzati per tale scopo. Quello più usato è il Message Passing Interface (MPI) che è un protocollo standard per la comunicazione fra processi appartenenti a nodi diversi in un cluster di computer. Ne esistono varie implementazioni sia Open Source che free che commerciali e l'adozione di una versione piuttosto che un'altra è legata a vari fattori quali le performance, la disponibilità del codice sorgente da poter, eventualmente, modificare, la disponibilità di una versione per uno specifico sistema operativo e, ovviamente, il costo sia di licenza che di manutenzione ed aggiornamento.
L'uso di MPI libera la programmazione da una conoscenza approfondita dell'Hw ed assicura un'ampia portabilità su un gran numero di architetture e configurazioni.
È de facto lo standard per la comunicazione tra nodi appartenenti a un cluster di computer che eseguono un programma parallelo multi-nodo. MPI rispetto alle precedenti librerie utilizzate per il passaggio di parametri tra nodi, ha il vantaggio di essere molto portabile (MPI è stata implementata per moltissime architetture parallele) e veloce (MPI viene ottimizzato per ogni architettura). Lo standard, alla versione 3.1, definisce la sintassi delle chiamate MPI per i linguaggi C e Fortran.

11.5 *Sistemi di interconnessione*

Molte tipologie e tecnologie di interconnessione sono state usate all'interno di complessi sistemi di Cluster e Farm per collegare fra di loro i nodi (computer o processori o device) abbiamo già visto come in ambito rete la tecnologia Ethernet sia quella più diffusa ed economica anche nelle sue velocità più recenti (da 10 Gb a 100 Gb) ma in alcune situazione in cui è richiesta bassissima latenza nelle comunicazioni per massimizzare l'efficienza del parallelismo di più processori che eseguono applicazioni multi-nodo, sono state e sono ancora utilizzate soluzioni diverse.
Possiamo citare, oltre a Ethernet, ad esempio:

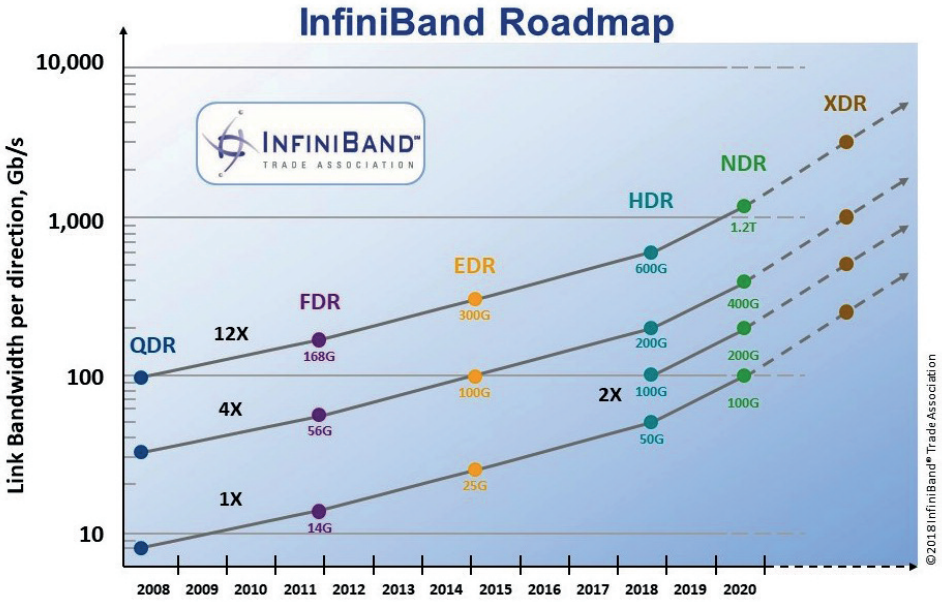
– Infiniband è una tecnologia standard basata su Switch proposta da Intel ed attualmente gestita dalla Infiniband Trade Association.
– Myrinet – Proprietario (Myricom, Inc.) ha costituito per anni il principale sistema di interconnessione per clusters HPC
– Quadrics: Sistema switched Proprietario della Quadrics.
– NumaLink: System interconnect sviluppato da Silicon Graphics

Di queste è Infiniband che esaminiamo a scopo didattico perché ancora molto usata.

	N. Wires	SDR (Gbps)	DDR (Gbps)	QDR (Gbps)
1X	4	2.5	5 G	10 G
2X	16	10	20 G	40 G
3X	48	30	60 G	120 G

Tabella 11.2 - Tipologie di velocità di connessione di Infiniband

Esso inizialmente supporta varie velocità con multipli del singolo link seriale differenziale e full-duplex da 2,5 Gbps in Single Data Rate. Tale link è poi stato utilizzato da Intel come base per il suo PCI-Express.



SDR - Single Data Rate
DDR - Double Data Rate
QDR - Quad Data Rate
FDR - Fourteen Data Rate
EDR - Enhanced Data Rate
HDR - High Data Rate
NDR - Next Data Rate
XDR –Extended Data Rate

Figura 11.5 - Evoluzione di Infiniband

Un elenco delle velocità iniziali di connessione Infiniband è mostrato in Tabella 11.2, mentre in Figura 11.5 è mostrato il percorso evolutivo dello standard con le varie configurazioni e velocità. Lo standard di Infiniband prevede anche il supporto al Direct Memory Access (DMA), al Quality of Service (QoS), al Multicast, al Message Queuing, al Communication Flow Control sia Link Level che End to End.

La comunicazione in Infiniband avviene attraverso una rete switched basata su pacchetti di dati con un'architettura di protocollo a strati: Physical, Link, Network, Transport, Upper Layers che richiama alcuni dei livelli ISO/OSI visti in precedenza.

La Figura 11.6 mostra questi strati evidenziando come alla base ci sia uno strato fisico che permette la comunicazione fra un "End Node", che può essere un Computer, e un altro End Node, sia attraverso Switch che Router, con quest'ultimo che usa un strato di Network in cui gli indirizzi sono basati su IPv6.

Sopra il livello di Network un livello di Transport permette l'uso di messaggi a supporto del livello superiore che è quello delle transazioni fra nodi.

Vale la pena notare che sopra il livello fisico, il livello di Data Link è ottenuto usando un indirizzamento MAC che però è differente da quello Ethernet ed è a 64 bit, di cui, i primi 24, identificano il produttore ed i restanti 40 bit, il dispositivo.

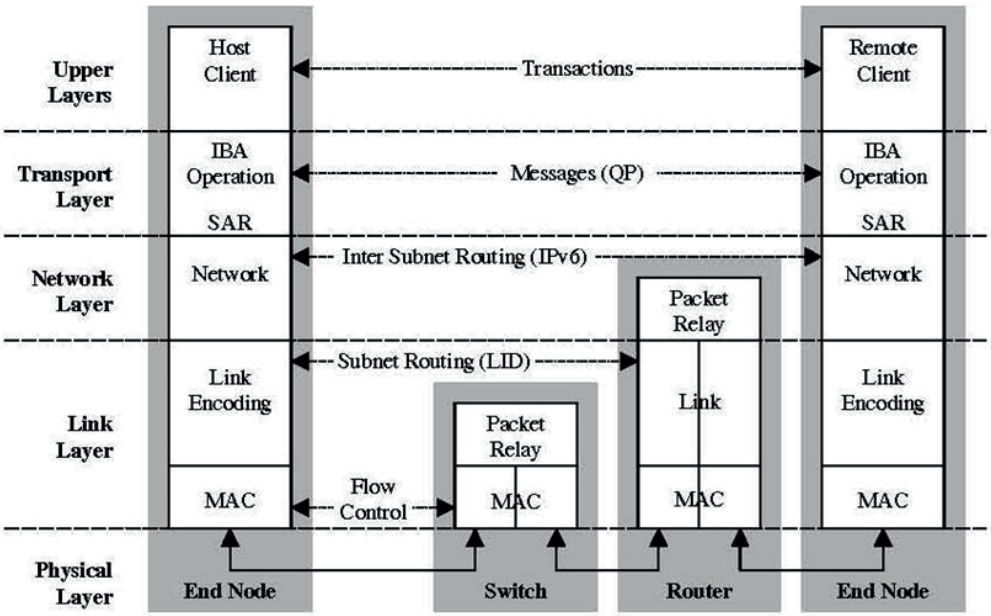


Figura 11.6 - Strati del protocollo Infiniband

Nella Figura 11.7 sono mostrati gli elementi di un sistema con processore e memoria che tramite un Host Channel Adapter usa dispositivi di I/O collegati tramite un Target Channel Adapter (TCA) a uno switch Infiniband.

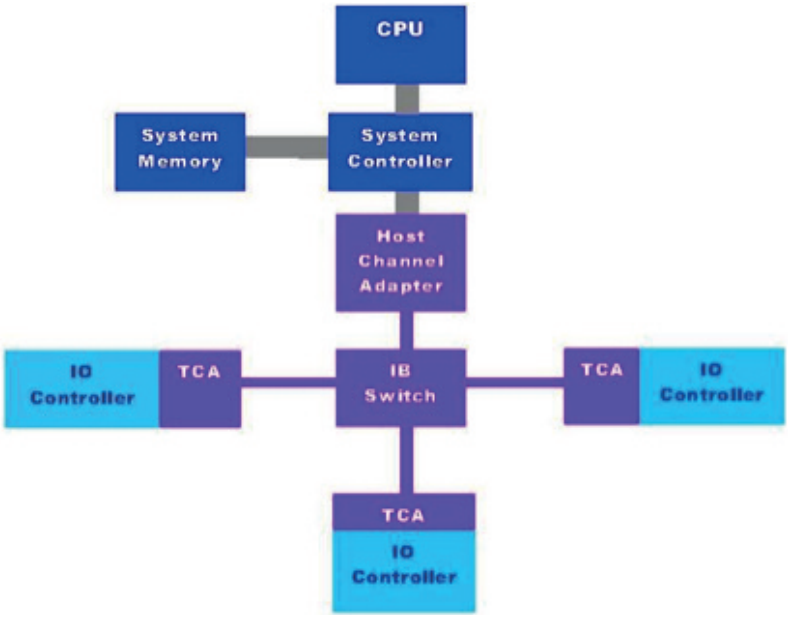


Figura 11.7 - Elementi di un sistema Infiniband

11.6 *Uso delle Farm On-Line*

Le Farm sono state molto usate negli ultimi anni nei sistemi di DAQ con due compiti specifici:

– Event Building per la ricomposizione di eventi completi a partire dai dati provenienti da sotto-rivelatori o da flussi differenti.
– Trigger di Livello 3 per selezionare gli eventi sulla base di una ricostruzione ed analisi quasi-online dei dati.

Lo schema dell'Event Building è quello mostrato in Figura 11.8 dove dei processori di Livello 2 acquisisco i dati da un Front-End di parte del rivelatore e inviano tali sotto-parti di eventi a specifici computer di Event Building per comporre gli eventi completi e archiviarli.

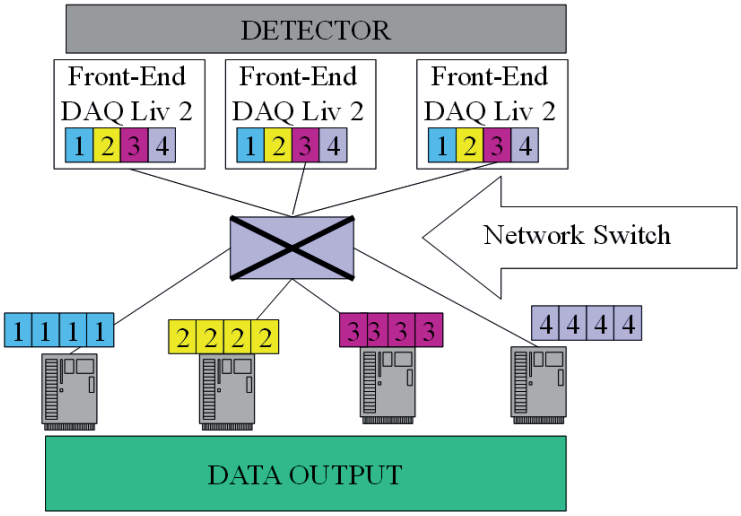


Figura 11.8 - Schema di una Farm per Event Building

Un esempio di uso di Farm per Trigger di Livello 3 è invece mostrato in Figura 11.9 dove, in aggiunta all'attività di Event Building, si realizza un filtro basato sulla ricostruzione veloce degli eventi per una ulteriore selezione.

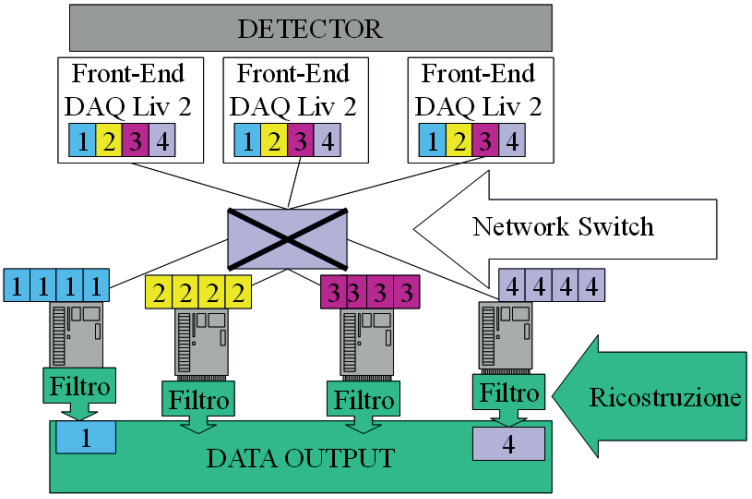


Figura 11.9 - Esempio di Farm per Trigger di Livello 3

11.7 *Uso delle Farm Off-Line*

L'uso di Farm nell'Off-line è quello che coinvolge attività di calcolo non in linea con i sistemi di acquisizione dati ed è ormai comune da molti anni con alcune principali applicazioni che sono:

– Simulazione MonteCarlo
– Processamento dei dati: ricostruzione, selezione, formattamento e riduzione dei dati, ecc.
– Analisi dei dati: selezione, generazione di n-tuple, istogrammi, grafici ecc.
– Applicazioni di Calcolo numerico e Teorico (lattice QCD).
– Applicazioni di Biologia: Genoma, Folding proteico, ecc.
– Applicazioni aerospaziali e di studio meccanico.

– Applicazioni di studio della Terra.
– Modellazione e rendering 3D per cinematografia.

Spesso le applicazioni sono quelle che abbiamo definito nel Par. 11.2 come High Throughput Computing, come mostrato nell'esempio di Figura 11.10 dove il processo di calcolo sui dati è replicato per elaborare contemporaneamente più file separati e fornire in parallelo l'output delle elaborazioni.

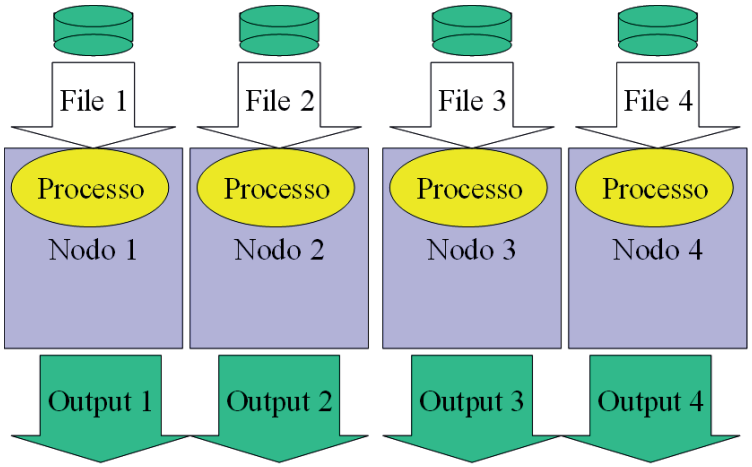


Figura 11.10 - Esempio di uso di Farm in attività di High Throughput Computing

11.8 *Architettura Software*

All'interno di un sistema di calcolo possono essere individuate una serie di componenti software che provvedono alle funzioni principali del sistema stesso e a fornire supporto alle applicazioni degli utenti.

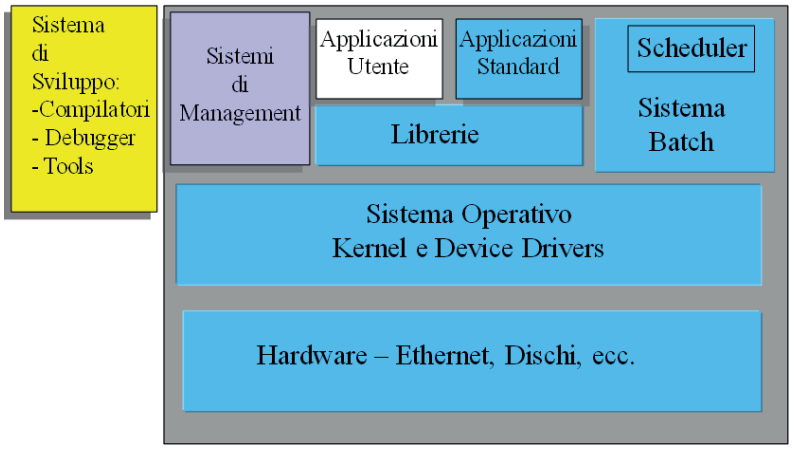


Figura 11.11 - Schema dell'Architettura Software di un Computer

Nella Figura 11.11sono rappresentate schematicamente le componenti principali:
– Un sistema di sviluppo del software, se la macchina deve anche avere questo compito, che comprende Compilatori, Debugger e strumenti per l'editing del codice, ecc.
– Sistemi di Management per gestire la macchina o il cluster se necessario;
– Applicazioni sia sviluppate dall'utente che applicazioni standard presenti nel sistema (es. database, strumenti grafici);
– Un sistema di Batch per schedulare job;

– Librerie per offrire strumenti e funzioni necessarie alle applicazioni (es. Librerie Matematiche, MPI, ecc.);
– Un Sistema Operativo sottostante che gestisca l'hardware, i processi ed i vari dispositivi;
– Lo strato Hardware che comprende le varie componenti: CPU, Memoria, Dischi, Interfacce di I/O, ecc.

Analizzeremo nei paragrafi seguenti alcune di queste componenti.

11.8.1 Sistema Operativo

LINUX è il sistema ormai largamente prevalente per farm e cluster in ambito scientifico grazie alla sua diffusione, completezza di software e attivo sviluppo da parte di una vasta comunità.
Inoltre è un sistema operativo Open e Free e consente l'installazione su migliaia di nodi senza aggravi di costi di licenze.
La disponibilità di molte applicazioni e tools standard (NFS, AFS, MPI, Batch Systems, GRID, Nagios, ecc.) rendono possibile creare un ambiente completo e performante.
LINUX, essendo Open Source, è largamente configurabile e modificabile per scopi specifici se si possiedono le competenze per farlo.
Il Kernel contiene tutte le funzioni principali ed a questo vengono aggiunti/installati i driver dei vari dispositivi fra cui anche quelli di I/O (es. USB, dischi, monitor, tastiere, ecc.).

11.8.2 Batch Systems

Il Batch System è un componente software per schedulare e controllare l'esecuzione dei Jobs su più nodi di calcolo ed un tale sistema provvede a gestirli attraverso delle code di esecuzione. Il sistema batch è usato in molte applicazioni tranne quelle che richiedono interattività con l'utente o che devono essere permanentemente in esecuzione.
Esistono diversi prodotti Free/OpenSource o a pagamento, alcuni esempi sono:
– PBS (+ Scheduler MAUI) sia Open che a pagamento
– LSF (Load Sharing Facility) prodotto dalla Platform
– Condor – free ma non completamente Open
– Codine (Gridware/SUN)
– BQS

Esaminiamo per primo PBS (Portable Batch System) che esiste in due tipologie:
– OpenPBS sviluppato da: NASA Ames Research Center, Lawrence Livermore National Laboratory eVeridian Information Solutions Inc. è una versione OpenSource. <www.openpbs.org>
– Professional PBS della Altair è un prodotto con due versioni di cui una commerciale. <www.altair.com/software/pbspro.htm>

PBS possiede uno scheduler non molto sofisticato e, in generale, soprattutto per la versione Open, si usa affiancargli uno scheduler MAUI che è "quasi" OpenSource distribuito dalla Cluster Resources Inc. <www.clusterresources.com/products/maui/>

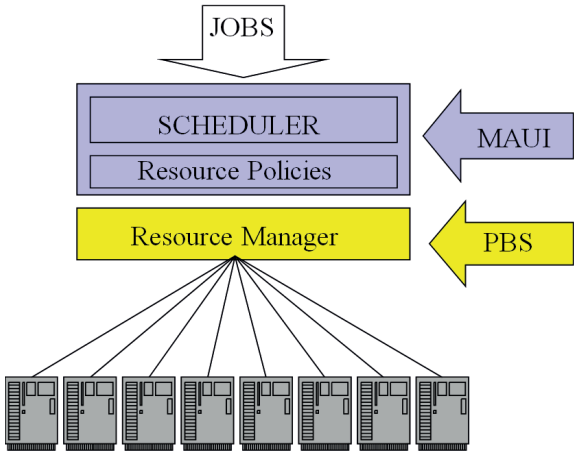


Figura 11.12 - PBS con Scheduler MAUI

Un altro Batch System molto diffuso è **LSF** (**Load Sharing Facility**) che è un prodotto commerciale della Platform ed è usato anche al CERN sulle farm a disposizione degli utenti e degli esperimenti. Ha il vantaggio di essere multipiattaforma, completo nelle funzionalità, supportato a livello commerciale e scalabile anche a livello geografico.
Per contro è a pagamento con un costo per le licenze.
Infine **Condor** è un Batch System sviluppato presso l'Università del Wisconsin e che si è evoluto con numerose funzionalità ed è la base su cui si sono sviluppati i primi tools di workload management in ambito GRID.
I suoi punti a favore sono che il software non è a pagamento, è scalabile anche a livello geografico (Condor Pool e flock) ed è completo di molte funzionalità come: match making, check-pointing, parassitaggio, ecc.
Gli aspetti meno positivi sono che: non è completamente Open Source e che supporto e sviluppo sono concentrati a Madison su base di best effort (gratis) o a pagamento.

11.8.3 Controllo delle Farm

Una Farm ed un Cluster necessitano di strumenti per controllare da postazione remota tutto il sistema e le funzionalità. Il controllo deve permettere di accedere con privilegi da amministratore a tutta la Farm o Cluster. Inoltre è necessario che, a fronte di problemi o fallimenti di nodi o parti di essi, sia possibile da remoto far ripartire il nodo o l'intera Farm. Perché ciò avvenga minimizzando i singoli punti di fallimento, sono generalmente previste più tecnologie per accedere ai nodi ed ai dispositivi di rete che li collegano. Qui di seguito esaminiamo alcune delle tecnologie tradizionali ed attuali.

ILO (**Integrated Lights-Out**) è una tecnologia proprietaria proposta da HP che permette la gestione fuori banda, cioè su rete dedicata, del sistema o computer e vi sono altre tecnologie simili (**lights out management**) offerte da altri produttori. Si tratta in sostanza di una porta Ethernet dedicata all'accesso remoto del computer e dalla quale si possono effettuare tutte le operazioni come se si fosse fisicamente collegati alla console del computer. La maggior parte dei server moderni la offrono come dispositivo standard a bordo.

Consolle remotizzata: **KVM** (**Keyboard Video Mouse**) tramite switch oppure con linea seriale e Terminal Server è una tecnologia piuttosto consolidata che permette un controllo completo spesso anche la modalità grafica. Un KVM può essere analogico o digitale:
– In analogico i segnali analogici provenienti dal video (VGA) vengono ridiretti verso la

porta Terminale tramite una tecnologia switched. Il segnale può anche subire alcune conversioni A/D per il trasporto su distanze > 2 m su cablature UTP o F/O.

– In digitale il video VGA viene interpretato e trasportato in rete su protocollo TCP/IP verso il Terminale remoto.

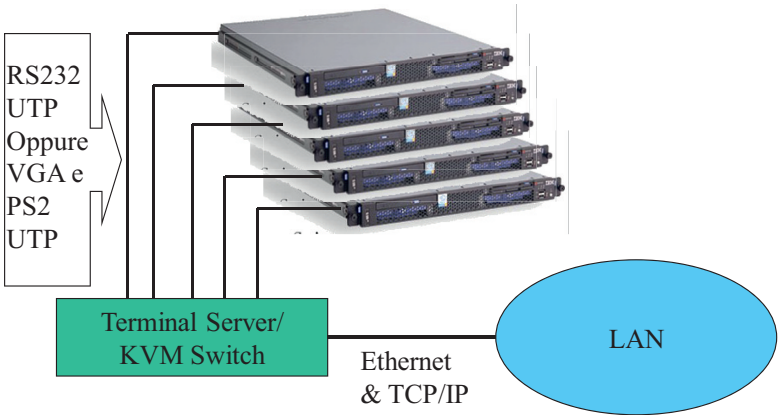


Figura 11.13 - Sistema KVM tradizionale

In alternativa ci si può accontentare di un semplice terminale alfanumerico utilizzando un **Terminal Server** che colleziona le porte seriali su cui sono redirette le consolle dei nodi di calcolo tramite configurazione del BIOS.

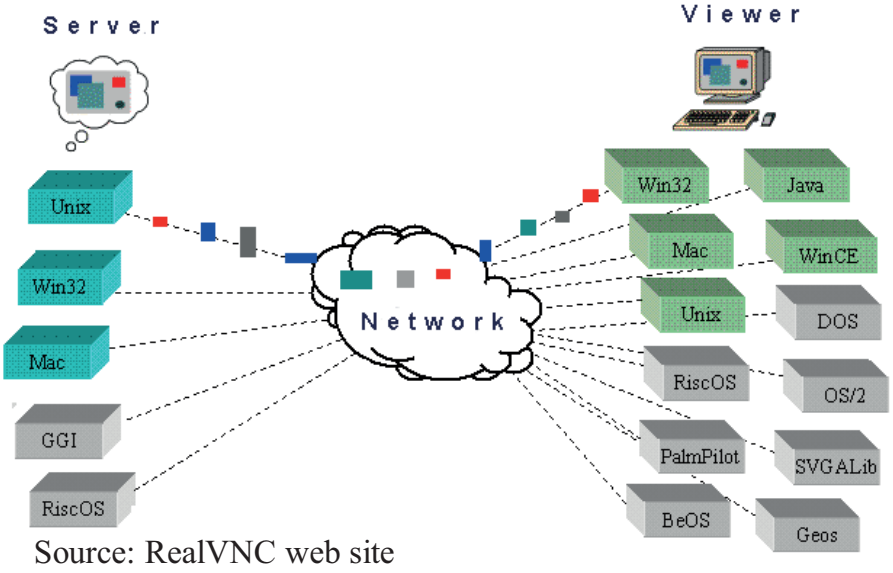


Figura 11.14 - Applicazioni di VNC

Il Controllo remoto può anche essere realizzato via software in-band, cioè sulla rete di comunicazione normale. In questo caso un esempio molto diffuso è **VNC** (**Virtual Network Computing**) è un prodotto Free originariamente sviluppato dalla AT&T (ex Olivetti Research) in Inghilterra. Ne esistono varie versioni, anche a pagamento, con specifici miglioramenti (sessioni crittografate, ecc.) e può essere utilizzato sia attraverso un programma client disponibile per varie piattaforme, sia attraverso un applet Java via browser web. Per farlo funzionare è necessario installare un server sul nodo da controllare e richiede una buona connessione di rete per gestire la grafica.

Altre tecniche e strumenti per completare il controllo remoto possono essere:

– **Webmin** che è uno storico software free (BSD license) per amministrazione remota via web di sistemi Linux.

– Presiere per alimentazione controllate da remoto: **Power Distribution Unit** (**PDU**) con porte Ethernet UTP che permettono via rete di accendere, spegnere e controllare le alimentazioni dei server, oltre ad avere funzioni di accensione programmata per l'operazione di Power-up, in maniera tale da non accendere tutte contemporaneamente le macchine ma rispettare una sequenza di accensione che eviti lo spunto di corrente all'accensione e che alcune macchine partano più velocemente di altre che sono indispensabili per le loro funzioni.
– Anche gli Switch Gigabit Ethernet sono gestibili, sia tramite SNMP (Simple Network Management Protocol) o attraverso strumenti proprietari.
– Per ultimo, ma non meno importante è il Monitoraggio dell'hardware e del software delle Farm e ci sono molti software (es. GANGLIA, Nagios) che consentono sia di monitorare che, eventualmente, inviare allarmi ed effettuare semplici operazioni di recupero già programmate.

11.8.4 File System e Storage

Nei Cluster, ma spesso anche nelle Farm, sono usati dei File System (FS) che permettono di condividere almeno una parte dello storage. Nel corso degli anni sono stati sviluppati molti FS che offrono varie funzionalità e performance. Qui di seguito ne facciamo un piccolo elenco non esaustivo:

– OpenAFS (Andrew File System)
– NFS (Network File System)
– GPFS (Global Parallel File System) proprietario di IBM
– Lustre (File system parallelo e distribuito)
– PVFS (Parallel Virtual File System)
– GFS (Global File System)
– Xrootd (sviluppato a SLAC ed usato per primo dall'esperimento BaBar)
– DCache (sviluppato a Desy e FNAL)
– RFIO (sviluppato al CERN)
– GFAL (sviluppato nel progetto LCG)
– CASTOR (sviluppato al CERN)
– SRM (specifica standard di gestione remota dello Storage)

Le principali caratteristiche ricercate in un File System locale o distribuito riguardano principalmente i seguenti aspetti:

– Gestione di grandi quantità di dati (TB-PB e oltre)
– Gestione di grandi Volumi fisici e logici (> 10TB)
– Access Control List (ACL) per controllare i permessi di accesso ai files da parte degli utenti, invece di limitarsi al GID (Group ID) e UID (User ID) standard di Unix.
– Lock Manager per l'accesso in contesa ai files da parte di più processi.
– Fail-over con copie dei file su dischi diversi per garantire affidabilità.

11.8.5 Potenza dissipata e raffreddamento

Un aspetto rilevante nella costruzione di un Data Centre (DC) sia che esso sia destinato ad ospitare una Farm, un Cluster o un Super Computer, è il consumo di energia elettrica necessaria al suo funzionamento, a cui va aggiunta l'energia necessaria per poterlo raffreddare dalla sua dissipazione. Le potenza elettrica fornita ad un computer, infatti, solo in parte limitata è spesa per la sua funzionalità, molta potenza elettrica viene invece dispersa in calore. Nel corso degli ultimi decenni si è cercato sempre di più di aumentare la potenza di processamento dei processori aumentando il numero di transistor ed innalzando la frequenza di lavoro. Per valutare le prospettive

di incremento delle prestazioni dovute all'evoluzione tecnologica, si è fatto ricorso alla cosiddetta legge di Moore che prevedeva un raddoppio della densità dei transistor sul semiconduttore ogni 24 mesi, producendo così anche un virtuale raddoppio delle prestazioni.

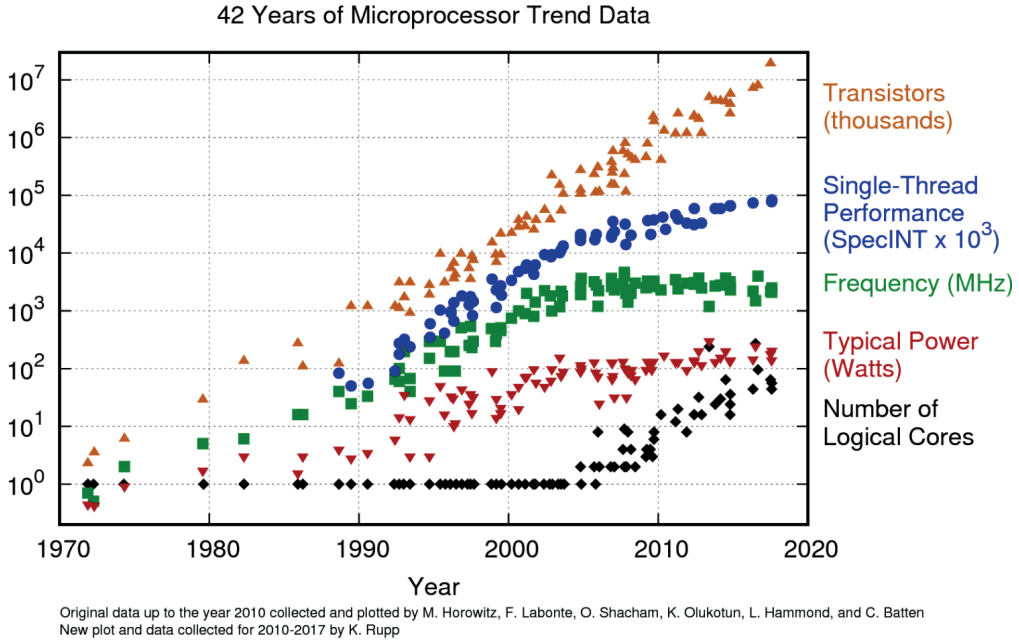


Figura 11.15 - Sviluppo dei Processori nel tempo

Un tale trend esponenziale aveva però alcune problematiche, la prima delle quali era che il raddoppio della densità di transistor non era sufficiente a fronteggiare la richiesta di potenza di calcolo di alcune applicazioni e quindi si è fatto ricorso anche all'aumento di frequenza dei processori per incrementarne le prestazioni di picco.
Questo ha incrementato anche, a sua volta, la potenza elettrica dissipata dal processore portandola ai limiti di temperatura e di quanto poteva essere dissipato senza danneggiare il processore stesso. Si è dunque giunti, nei primi anni 2000 ad una saturazione delle prestazioni e della potenza elettrica dissipata che ha portato a riconsiderare le architetture che fino a quel momento puntavano alla massima potenza di picco del singolo processore. A partire dalla seconda metà degli anni 2000 hanno così cominciato a fare la loro comparsa processori cosiddetti multicore che, all'interno di una CPU implementano più processori logici e di calcolo. Questa soluzione ha avuto un notevole successo grazie al fatto che oramai all'interno di un computer eseguono molti programmi e task con funzioni diverse e quindi, invece di offrire un time-sharing di un singolo processore ai task, si è potuto portare la loro esecuzione su core diversi all'interno della stessa CPU con il risultato di ottenere prestazioni migliori con frequenze di clock dei processori meno spinte.
Un altro aspetto della evoluzione dei processori è legato anche alla riduzione della potenza elettrica utilizzata, sia abbassando la tensione di alimentazione dei transistor, sia, come già accennato limitando la frequenza di clock. Questa evoluzione tecnologica è stata anche favorita dalla diffusione di computer portatili e dispositivi mobili che, funzionando a batteria, hanno necessità di risparmio energetico ed alta efficienza.
Tali sistemi di risparmio energetico, utilizzati nei processori per sistemi portatili, poiché puntano su meccanismi di riduzione del clock o spegnimento di parti del processore quando non sono in uso, non sono completamente efficaci in sistemi che devono garantire il 100% di CPU per settimane.
Il disegno dei parametri termici e di potenza elettrica di una Sala Macchine, sia esso un Data Centre o una Counting Room di un esperimento, sono limitati ed è necessario uno studio accurato e

specialistico. L'efficienza energetica di un Data Centre è espressa generalmente con un valore di **PUE** (**Power Usage Effectiveness**) che è una misura di quanto sia efficiente l'uso dell'energia in un Centro di Calcolo ed è definito come:

PUE= PT / PC

Dove:

– PT è la potenza totale del Centro e
– PC è la potenza usata dalle sole apparecchiature informatiche: Computer, Storage e Rete.

Il PUE misura quindi il contributo delle infrastrutture come condizionatori, UPS ecc.
Un PUE pari a 1.5 significa che le infrastrutture consumano circa 50% aggiuntivo rispetto al consumo delle attrezzature IT
Molti Computing and Data Centre più vecchi sono anche al di sopra di PUE 1.5 mentre quelli più nuovi e costruiti per il risparmio energetico sono intorno ad un PUE di 1.1.

11.8.6 Farm e GRID Computing

All'inizio degli anni 2000 si è sviluppato un modello di calcolo "collaborativo" chiamato GRID computing e che è particolarmente adatto alle FARM di calcolo perché permette di distribuire il carico su più risorse (anche a livello Geografico) e cerca di realizzare uno sfruttamento ottimale delle risorse esistenti. Originalmente proposto da Ian Foster e Ken Kessler il Grid Computing si è sviluppato, principalmente nell'ambito della High Energy Physics e degli esperimenti a LHC.
I suoi principali vantaggi sono:

– non è necessario sapere in dettaglio dove e come sono organizzate le risorse;
– sono disponibili molte più risorse che vengono messe "in comune" in un'ottica di sharing all'interno della comunità.

La sua diffusione al di fuori dell'ambito della Fisica delle Alte Energie, tuttavia, è rimasta molto limitata, principalmente a causa di due fattori:

– il suo utilizzo per l'utente generico è ostico perché è necessario un piccolo step di formazione per l'uso di questi strumenti e non è particolarmente "user friendly";
– richiede un'organizzazione per la gestione efficiente dell'infrastruttura dei servizi e non ha incluso un facile meccanismo di accounting e distribuzione dei costi.

Nella Figura 11.16 è mostrata la raffigurazione esemplificativa della infrastruttura di Farm al CERN con la corrispondenza agli elementi caratteristici dell'architettura di GRID Computing che è invece esplicitata nella Figura 11.17 con le sue componenti principali.

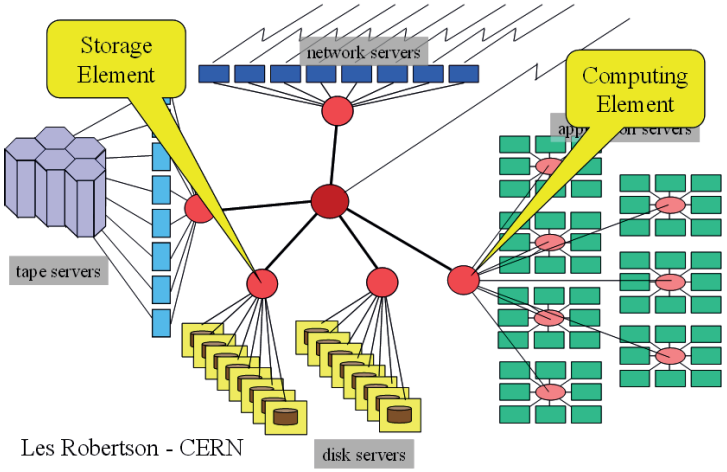


Figura 11.16 - Farm di Calcolo in Architettura GRID Computing

L'architettura si basa su tre componenti principali di cui la prima è il **Computing Element** (**CE**) che è l'elemento di gestione del calcolo nella infrastruttura GRID e soprassiede alla sua gestione. Una intera Farm può essere vista attraverso un CE.
Di fatto c'è una corrispondenza fra CE e le code di esecuzioni in batch definite sulla Farm. In base alle informazioni sul numero dei nodi, sul numero dei job in coda, ecc. e delle risorse richieste vengono effettuate le scelte di workload e i Job sono schedulati, attraverso Batch System sui nodi di calcolo della Farm o **Worker Node**.

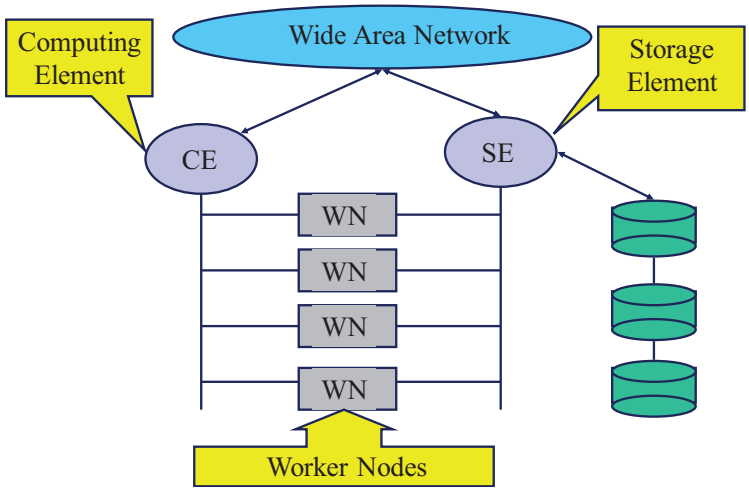


Figura 11.17 - Schema di un Sistema di Grid Computing

L'altro componente principale è lo **Storage Element** (**SE**) che espone verso l'esterno ed il resto della GRID le risorse di storage perché siano utlizzabili. L'accesso è basato su politiche legate ai certificati digitali di tipo X500 che sono usati per effettuare l'autenticazione degli utenti ed ai gruppi sperimentali definiti nelle Virtual Organisation (VO) dove sono definite le autorizzazioni ed i ruoli dei singoli utenti.

Capitolo 12

Archiviazione Dati

L'attività di salvataggio dei dati provenienti dal DAQ è, ovviamente, fondamentale e deve garantire la registrazione affidabile su supporti di storage adeguati. Un sistema di DAQ deve, dunque, poter archiviare dati in maniera veloce su un supporto affidabile e duraturo, anche se, in generale, l'archiviazione passa per una bufferizzazione di file su dischi magnetici e quindi per l'archiviazione su nastri o cassette a nastro magnetico per l'archiviazione a lungo termine. Questo è l'ultimo stadio dell'acquisizione dove i dati selezionati devono essere salvati ed è un punto cruciale del sistema.
Esamineremo dunque, per completezza, i dispositivi di archiviazione (dischi e nastri magnetici) e alcuni tipi di interconnessione per tali dispositivi come SCSI e Fibre Channel.

12.1 *Dischi*

I sistemi a disco hanno molti vantaggi perché, ad esempio, permettono un accesso casuale ai file e, nelle implementazioni attuali (anche RAID) sono in grado di fornire enormi quantità di storage e buone velocità di scrittura (dalle decine alle centinaia di MB/s). I dischi sono comunque una risorsa (per ora) più costosa delle cassette a nastro magnetico per grandi quantità di dati (PB e oltre) e, quelli tradizionali magneto-meccanici, essendo continuamente in rotazione sono soggetti ad usura meccanica.
I nuovi dischi a Stato Solido (SSD) non hanno problemi meccanici ma hanno un costo maggiore per TB e possono, in alcune tipologie, avere problemi legati al numero di riscritture possibili, prima del deterioramento.

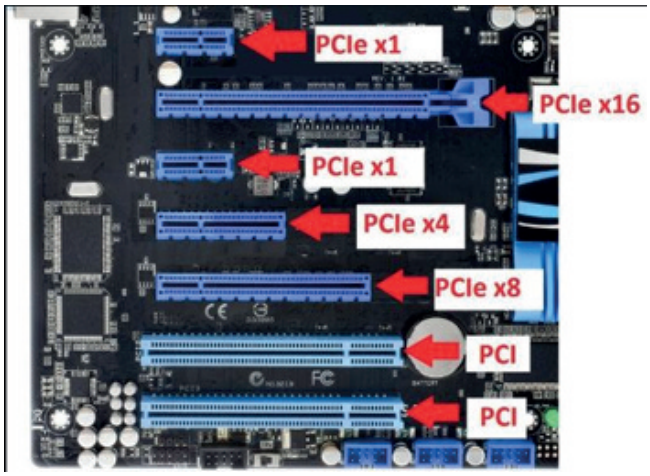


Figura 12.1 - Esempi di connessioni PCIe a diverse molteplicità

Per i dischi tradizionali si possono usare bus di interfacciamento come Serial ATA (SATA), Fibre Channel e SCSI, mentre i più nuovi sistemi a stato solido offrono anche interfacciamenti diretti verso il Bus PCI-Express con maggiori prestazioni, sia in termini di banda passante che di velocità di accesso o latenza, come il Non-Volatile Memory Express, o più comunemente SSD NVMe. **NVMe** è un protocollo/interfaccia di comunicazione sviluppato specificatamente per gli SSD da un consorzio di cui fanno parte aziende come Intel, Samsung, Sandisk, Dell e Seagate. Il suo scopo è quello di favorire il raggiungimento del massimo delle prestazioni dei più moderni SSD, la cui performance è ormai limitata dai più classici protocolli SATA.
Come abbiamo già visto, gli slot PCIe funzionano come un insieme di linee o canali che offrono una banda passante (originariamente 2,5 Gbit/s ognuna) ed alcuni dispositivi, che hanno particolari esigenze di banda, possono utilizzare connettori con più canali in parallelo. Le configurazioni di numero di canali/linee possono quindi essere x1, x2, x4, x8 oppure x16, come ad esempio le GPU usano tipicamente uno slot PCIe x16, ovvero a sedici canali, il massimo disponibile su un singolo slot, per ora. Gli SSD NVMe non raggiungono velocità così alte e usano tipicamente PCIe x4. La più recente versione PCIe 3.0 raggiunge bande passanti approssimative di 1 GByte/s per canale/linea e con le configurazioni SSD NVMe su slot PCIe x4 si raggiunge una banda passante teorica di 4 GByte/s.

12.2 *Nastri Magnetici*

L'archiviazione su nastro magnetico è da sempre considerata l'operazione necessaria per garantire la sicurezza dei propri dati. In generale i nastri magnetici su cassette sono utilizzati per effettuare backup: cioè si effettua una copia su nastro di ciò che c'è su disco e poi si continua a lavorare con la copia su disco.
In HEP il nastro è considerato invece la copia master dei dati originali. Essa viene scaricata su disco quando serve per le analisi. Il disco è quindi un buffer su cui lavorare.

Figura 12.2 - Cassetta a nastro magnetico LTO e Driver

Le tipologie di cassette a nastro magnetico sono diverse sia in formato proprietario, come IBM 3592, che in formato aperto come Linear Tape Open (LTO) che è una tecnologia a nastro magnetico sviluppata originariamente alla fine degli anni '90 del secolo scorso come uno standard aperto in alternative ai formati proprietari. Hewlett Packard Enterprise, IBM, e Quantum controllano il Consorzio LTO che dirige lo sviluppo della tecnologie e gestisce la licenze e le certificazioni dei dispositivi e delle cassette da parte dei produttori.
Il formato standard della tecnologia LTO va sotto il nome di Ultrium e la sua versione originale fu rilasciata nel 2000 con una capacità della cassetta di 100 GB. L'ottava generazione delle cassette LTO Ultrium è stata rilasciata nel 2017 e ha una capacità di 12 TB (30 TB con compressione di 2,5:1) mantenendo le stesse dimensioni della cassetta.

12.3 *Hierarchical Storage System*

Anche nel campo dell'off-line computing viene usato, per grandi quantità di dati, un sistema a più livelli detto di tipo gerarchico.
In sostanza si individuano due o più livelli di archiviazione con diverse caratteristiche di utilizzo; ad esempio:

– Dischi per l'accesso diretto ai file comunemente più usati;
– Cassette a nastro magnetico per i file scarsamente utilizzati.

Il sistema gerarchico prevede la migrazione dei dati da disco a nastro (e viceversa) seguendo delle precise politiche basate, ad esempio, sulla data dell'ultimo accesso al file, sulla sua dimensione, sullo spazio rimasto su disco, ecc..
Alcuni esempi di tali sistemi sono HPSS sviluppato da IBM e Castor sviluppato al CERN.

12.4 *SCSI*

Lo SCSI (**Small Computer System Interface**) è un'interfaccia standard progettata per realizzare il trasferimento di dati fra diversi dispositivi interni di un computer (detti devices) collegati fra di loro tramite un bus.
Per collegare un computer ad un host, il bus di collegamento ha bisogno di un **Host Adapter SCSI** che gestisce il trasferimento dei dati sul bus stesso. La periferica deve disporre di un controller SCSI, che è solitamente incorporato in tutte le periferiche, ad eccezione di quelle di più vecchia concezione. L'interfaccia SCSI viene per lo più usata per la comunicazione con unità hard disk e unità nastro di memorizzazione di massa, ma anche per connettere una vasta gamma di dispositivi, come scanner d'immagini, lettori e scrittori di CD (CD-R e CD-RW), lettori DVD. In effetti lo standard SCSI è stato ideato per favorire l'intercambiabilità e la compatibilità dei dispositivi (tutti, almeno in teoria).
In passato l'interfaccia SCSI era molto diffusa in ogni tipologia di computer, mentre attualmente trova un vasto impiego solamente in alcuni settori di server.
I computer desktop e portatili sono invece di solito equipaggiati con l'interfaccia ATA/IDE (acronimi rispettivamente di Advanced Technology Attachment e Integrated Drive Electronics) e Serial ATA, per gli hard disk, e con l'interfaccia USB (Universal Serial Bus) per altre periferiche di uso comune. Queste ultime interfacce, all'origine, erano più lente della SCSI, ma anche più economiche. Oggi una interfaccia USB3 può agevolmente permettere trasferimenti di 30-40 MB/s sostenuti.
È interessante notare che l'USB utilizza lo stesso set di comandi dello SCSI per implementare alcune delle sue funzionalità

Interfaccia	Velocità del bus (*Transfer rate*) (MByte/s)	Larghezza del bus (bits)	Max lunghezza cavi (metri)	Max numero di dispositivi
SCSI	5	8	6	8
Fast SCSI	10	8	1,5-3	8
Wide SCSI	20	16	1,5-3	16
Ultra SCSI	20	8	1,5-3	8
Ultra Wide SCSI	40	16	1,5-3	16
Ultra2 SCSI	40	8	12	8
Ultra2 Wide SCSI	80	16	12	16
Ultra3 SCSI	160	16	12	16
Ultra-320 SCSI	320	16	12	16
iSCSI	Velocità rete	N/A	N/A	

Tabella 12.1 - Tiplogie di collegamento SCSI

Come vedremo dei prossimi paragrafi, il protocollo SCSI è anche utilizzato al di sopra di altre tecnologie come, ad esempio, Fibre Channel e IP.

12.4.1 iSCSI

iSCSI è un'abbreviazione per ***Internet SCSI***, una implementazione del protocollo SCSI al di sopra del protocollo IP per collegare dispositivi di storage su una rete e trasferire dati tramite l'uso di comandi SCSI incorporati in un protocollo IP.
iSCSI supporta interfacce di tipo Gigabit Ethernet (e superiori) al livello di layer fisico e questo permette ai sistemi che supportano le interfacce iSCSI di collegarsi direttamente a switch Gigabit Ethernet e/o Router IP.
Un sistema operativo che necessita di accedere a un dispositivo genera un comando SCSI che viene incorporato in un pacchetto IP ed inviato su una connessione Ethernet. Il ricevente decodifica il pacchetto IP e separa i comandi SCSI dai dati inviandoli al controller SCSI e da questi al dispositivo di storage.
Il sistema iSCSI provvederà anche a fornire una risposta alla richiesta ricevuta sempre tramite lo stesso protocollo.
La tecnologia iSCSI si è diffusa in ambito SAN (Storage Area Network) perché permette di sviluppare una SAN usando tecnologie di rete LAN (Local Area Network), WAN (Wide Area Network) o MAN (Metropolitan Area Network).
iSCSI è stato sviluppato da IETF ed è diventato uno standard ufficiale nel febbraio del 2003.

12.5 *Fibre Channel*

Fibre Channel è un protocollo standard di comunicazione per computer e dispositivi che è stato disegnato per avere un ampio campo applicativo e soddisfare molti tipi di necessità, in particolare quella di una sempre crescente domanda di alte prestazioni nel trasferimento dei dati. Gli obiettivi di Fibre Channel sono principalmente:

- Permettere a molti protocolli esistenti, sia di trasmissione che di rete, di poter utilizzare lo stesso strato fisico ed il mezzo trasmissivo;
- Avere una alta banda passante (almeno 100 MB/s ed oltre);
- Topologie di connessione flessibili;
- Possibilità di connessione a distanza di diversi km;
- Supporto a molte velocità di trasporto dati, mezzi trasmissivi e connessioni;
- In generale FC prova a combinare i benefici di tecnologie di trasporto a canale e di rete.

Fibre Channel ha avuto un notevole impatto nell'ambito dei sistemi di storage avvalendosi del protocollo SCSI come livello superiore. Rispetto al più tradizionale SCSI, i benefici sono di mappare i comandi SCSI al di sopra di un canale FC per ottenere:

- Maggiori velocità trasmissive;
- Maggior numero di dispositivi che si possono collegare insieme;
- Maggiori distanze fra dispositivi.

Esaminiamo nel seguito le tre topologie di connessione previste dallo standard FC.

12.5.1 FC Point-to-Point

Una topologia Point-to-Point è la più semplice (ed economica) delle tre previste e consiste di soli due dispositivi connessi direttamente fra loro come rappresentato in Figura 12.3. Le due fibre che costituiscono il canale trasmissivo incrociano il Transmit dell'uno con il Receiver dell'altro e vice versa. Non c'è un mezzo condiviso e questo permette di sfruttare appieno la banda trasmissiva di ciascuno dei due link con una connessione full-duplex.
Anche dal punto di vista della rapidità di inizializzazione della connessione non c'è necessità di

procedure particolari e i due dispositivi possono cominciare a comunicare immediatamente.

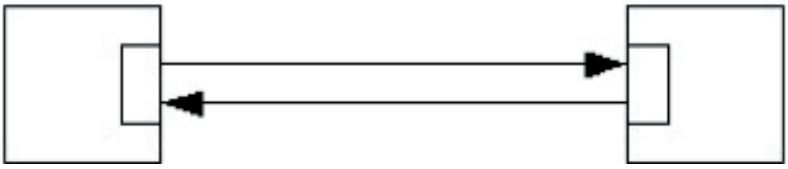


Figura 12.3 - Fiber Channel Point-to-Point

12.5.2 FC Arbitrated Loop

Il Fibre Channel Arbitrated Loop è una topologia che consente di collegare più di due dispositivi in maniera economica ma con una maggiore complessità del cablaggio.
Questa topologia è effettivamente vantaggiosa per il suo costo e perché permette, grazie allo standard FC-AL di collegare fino a 127 porte FC in una singola rete. A differenza delle altre due topologie il mezzo trasmissivo è condiviso fra i dispositivi collegati con conseguenti limitazioni di accesso di ciascun dispositivo nel caso in cui sia già in uso da parte di altri. La caratteristica di AL non è scontata e può anche non essere presente in alcuni dispositivi e quindi perché FC-AL possa operare correttamente, tutti i dispositivi collegati devono supportare questa caratteristica opzionale.

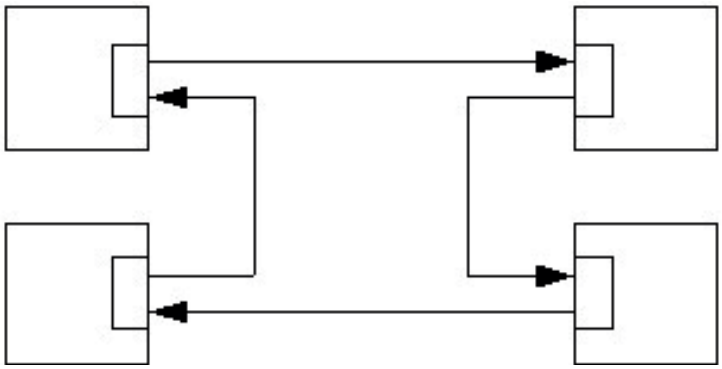


Figura 12.4 - Fibre Channel Arbitrated Loop

12.5.3 FC Fabric

La topologia FC Fabric è particolarmente utile quando si debbano connettere molti (> 127) dispositivi in una configurazione di tipo cross-point switched. Il beneficio di questa topologia è che usando un sistema switched è possibile far comunicare fra di loro contemporaneamente molti dispositivi visto che non c'è un unico mezzo fisico condiviso ma tanti collegamenti point-to-point verso uno switch. Naturalmente questo richiede l'uso di (almeno) uno switch che comporta un costo aggiuntivo.

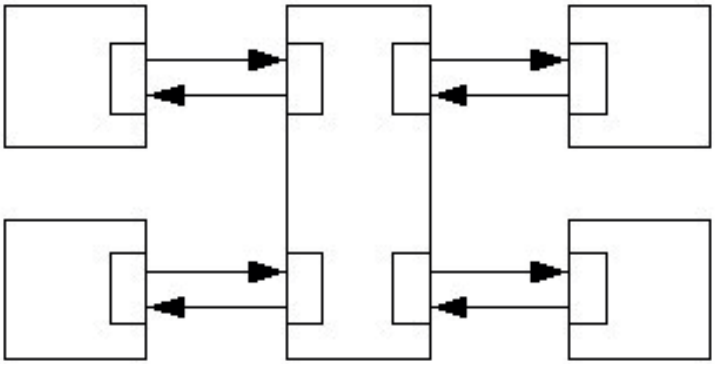


Figura 12.5 - Fibre Channel Arbitrated Loop

Nella tabella seguente, sono elencate le varie caratteristiche del canale FC che può essere utilizzato e le conseguenti velocità (MB/s) trasmissive permesse.

Product Naming	Throughput (Mbytes/s)	Line Rate (Gbaud)	T11 Specification Technically Complete (Year)*	Market Availability (Year)*
1GFC	200	1.0625	1996	1997
2GFC	400	2.125	2000	2001
4GFC	800	4.25	2003	2005
8GFC	1,600	8.5	2006	2008
16GFC	3,200	14.025	2009	2011
32GFC	6,400	28.05	2013	2016
128GFC	25,600	4X28.05	2014	2016
64GFC	12,800	56.1	2017	2019
256GFC	51,200	4X56.1	2017	2019

Tabella 12.2 - Fibre Channel tipologie di connessione

Le tecnologie viste sinora possono essere utilizzate per realizzare quella che va sotto il nome di Storage Area Network (SAN) in cui dispositivi di storage (Dischi, Nastri) sono collegati con server che usano tali dispositivi per le applicazioni che eseguono su di essi. Tali installazioni SAN non sono tuttavia isolate, ma connesse verso una rete esterna di tipo geografico o Wide Area Network (WAN).

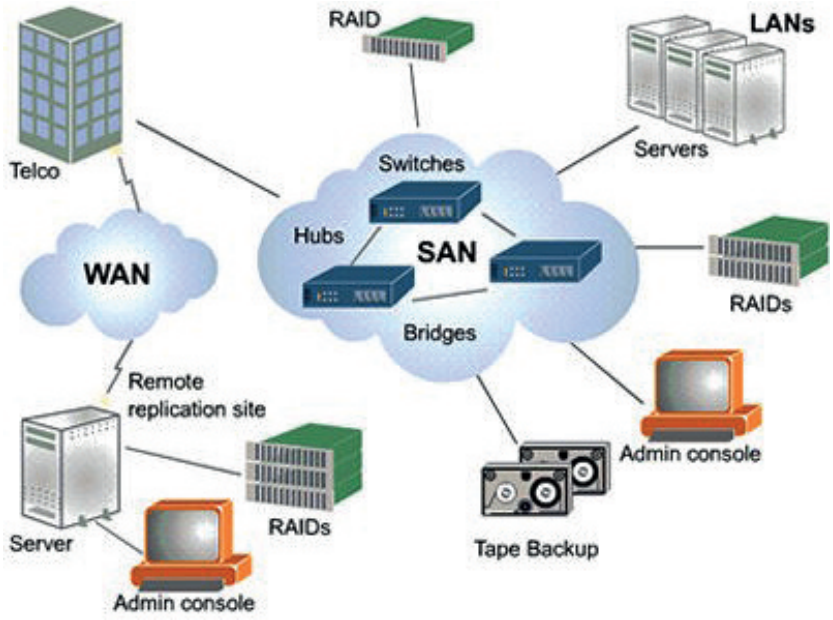


Figura 12.6 - Storage Area Network

La connessione di rete permette l'accesso alla SAN e alle sue risorse anche da parte di server e client remoti, nonché, viceversa, di consentire ai server della SAN di poter accedere a risorse remote.

Bibliografia

[R01] W.R. Leo – Techniques for Nuclear and Particle Physiscs Experiments – Springer – Verlag, ISBN: 3540572805

[R02] Formulae and Methods in Experimental Data Evaluation - CERN web - <http://rd11.web.cern.ch/RD11/rkb/PH14pp/node1.html>

[R03] Real Time Process Control (Introduction) – Tono Riesco / CERN - <http://agenda.cern.ch/fullAgenda.php?ida=a036670>

[R04] Basic concepts of real-time operating systems - by David Kalinsky (Nov. 18, 2003) - <http://linuxdevices.com/articles/AT4627965573.html>

[R05] Clara Gaspar – Trigger and Data Acquisition – Summer Student Course 2002 CERN - <http://agenda.cern.ch/fullAgenda.php?ida=a034444>

[R06] Paris Sphicas – Trigger and Data Acquisition Systems – CERN Summer Student Course 2004 - <http://agenda.cern.ch/fullAgenda.php?ida=a042488>

[R07] TCP/IP Tutorial and Technical Overview -Adolfo Rodriguez, John Gatrell, John Karas, Roland Peschke - 2001 - <ibm.com/redbooks>

[R08] <http://www.erg.abdn.ac.uk/users/gorry/course/syllabus.html>

[R09] VME – The VMEbus Specification Manual - VITA

[R10] VMEbus device drivers for computer systems running DIGITAL UNIX - <http://h30097.www3.hp.com/docs/dev_doc/DOCUMENTATION/HTML/AQ0R7GTE/DOCU_001.HTM>

[R11] VME Bus Specifications ISO std iec15776 <https://www.iso.org/obp/ui/#iso:std:iso-iec:15776:ed-1:v1:en>

[R12] Infiniband: Introduction to InfiniBand – White Paper by Mellanox – <http://www.mellanox.com/pdf/whitepapers/IB_Intro_WP_190.pdf>

[R13] Fibre Channel Tutorial <http://www.recoverdata.com/fc_tutorial.htm>

Acquisizione dati o "Data Acquisition" (DAQ) è l'insieme degli strumenti che permettono di acquisire delle informazioni in forma digitale da sensori o apparati complessi. La selezione dei dati utili e la contestuale eliminazione di quelli non interessanti è generalmente effettuata da un sistema chiamato Trigger. DAQ e Trigger non sono attività semplici da realizzare e non è facile ricavare dei modelli descrittivi generali validi per ogni contesto sperimentale, tuttavia molti aspetti sono ben conosciuti e fanno riferimento a conoscenze largamente applicate anche in altri campi. Un altro elemento di difficoltà nell'approccio al DAQ è che non ci sono testi sistematici ed esaustivi sull'argomento. Questo testo di appunti del Corso di Acquisizione Dati e Controllo di Esperimenti ha il compito non facile di mettere insieme quanto necessario per una comprensione delle problematiche principali senza la pretesa di essere esaustivo.

Federico Ruggieri è stato Dirigente di Ricerca dell'INFN fino al 2019 e Direttore del GARR dal 2015 al 2022, Professore Straordinario a tempo determinato dal 2020 e incaricato del corso di Acquisizione Dati e Controllo di Esperimenti presso il Dip. di Matematica e Fisica dell'Università di Roma TRE a partire dal 2005 fino al 2022. Nel corso e in questo libro, che ne costituisce una sintesi, ha portato la sua esperienza quarantennale nelle attività sperimentali di Fisica delle Particelle Elementari.

ROMA TRE
UNIVERSITÀ DEGLI STUDI