

Sistemi Distribuiti Real-Time



Approfondimento del corso di Sistemi Distribuiti

Sebastiano Vascon

788442

prof. Simonetta Balsamo

Indice generale

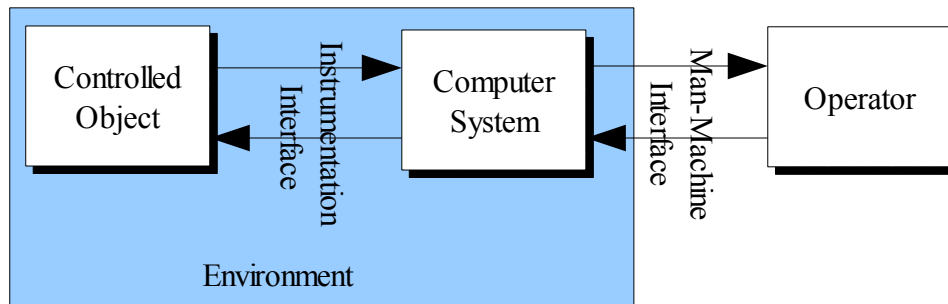
Introduzione.....	3
Classificazione dei sistemi real-time.....	4
Soft real-time e Hard real-time.....	4
Alta disponibilità e alta integrità dei dati.....	4
Hard real-time fail-safe e fail-operational.....	5
Reactive o Embedded real-time.....	5
Sistemi guaranteed-response e best-effort.....	5
Esempi.....	5
Automazione negli impianti.....	6
Telefonia.....	6
Reti di sensori wireless (WSN).....	6
Concetti di affidabilità.....	7
Ridondanza.....	7
Replicazione deterministica.....	8
Modelli di sistemi distribuiti real-time.....	9
Sistemi Event-Triggered.....	10
Sistemi Time-Triggered.....	11
Comunicazione real-time.....	12
Ritardo nella consegna di un messaggio.....	12
Ampiezza di banda e costo.....	12
Urgenza dei messaggi.....	12
Connettività.....	12
Caratteristiche di micronetwork.....	12
Scheduling real time.....	13
Classificazione degli algoritmi di scheduling.....	13
Hard real time e soft real-time scheduling.....	14
Scheduler dinamici e statici.....	14
Scheduler con/senza prelazione.....	14
Scheduling statico.....	14
Scheduling dinamici.....	14
Scheduling di task indipendenti.....	14
Rate monotonic algorithm.....	14
Earliest-deadline-first algorithm.....	15
Least-laxity algorithm.....	15
Scheduling di task dipendenti.....	15
Kernelized monitor.....	15
Priority Ceiling Protocol.....	15
Licenza & Info.....	18
Creative Common License:.....	18

Introduzione

In molti modelli di fenomeni naturali (ad esempio nella meccanica di Newton) il tempo viene considerato come una variabile indipendente che determina una sequenza di stati del nostro sistema.

Definiamo un *sistema real-time* come un sistema che cambia il suo stato in funzione del tempo (reale) [1].

Possiamo generalizzare un sistema real-time (RT) che riceve in ingresso stimoli dal mondo reale scomponendolo in un insieme di cluster come descritto nel seguente diagramma:



Disegno 1: Generalizzazione di un sistema real-time

Chiamiamo l'insieme composto da *controlled object* e *operator* l'environment (ambiente) del sistema. Il sistema deve reagire agli stimoli del controlled object o dell'operatore in un intervallo di tempo fissato a priori.

Ad oggi la maggior parte dei sistemi real-time è distribuita e consiste in un insieme di nodi interconnessi da un sistema di *comunicazione real-time* controllato da *protocolli real-time* (vedi pg. 12).

Basandoci sul precedente diagramma possiamo quindi dire che il tempo tra l'invio dello stimolo e della risposta è vincolato (*time constrained*) e l'insieme di operazioni svolte tra lo stimolo e la risposta lo chiameremo *real-time transaction*.

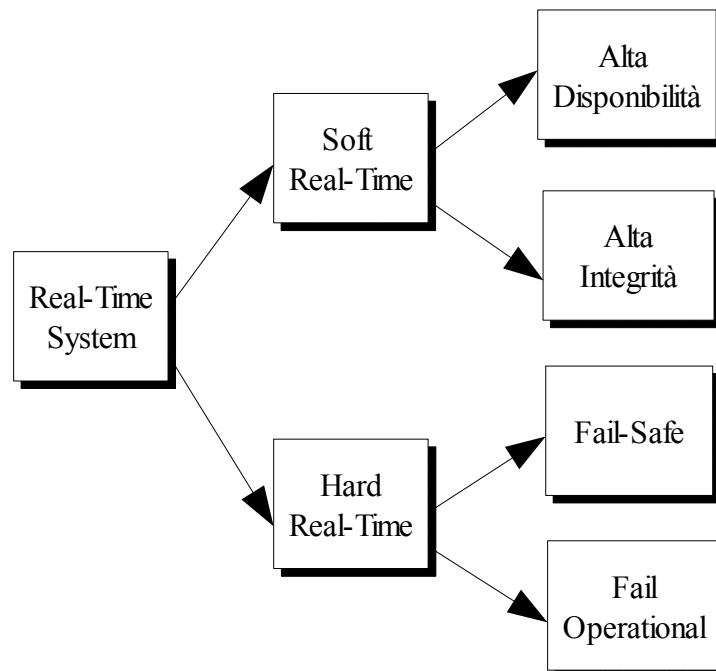
Quindi una transazione real time deve fornire una risposta corretta in un periodo di tempo prefissato, altrimenti il sistema fallirà.

Ogni sistema ha una capacità computazionale finita quindi per garantire i requisiti temporali avremo la necessità di fare alcune ipotesi sul nostro sistema:

- **Load hypothesis** (ipotesi di carico):
Definisce il traffico di picco che si assume sia generato dall'ambiente
- **Fault hypothesis** (ipotesi di fallimento)
Definisce il tipo e la frequenza dei fallimenti che il sistema deve essere in grado di gestire (fault-tolerance). Se l'ambiente genererà più fallimenti di quelli sopportabili dal nostro sistema di tollerance allora l'intero sistema avrà una fault che vedremo potrà portare a conseguenze più o meno gravi a seconda della tipologia di sistema real-time.

Classificazione dei sistemi real-time

Di seguito riportiamo alcuni schemi che rappresenta la suddivisione dei sistemi real-time in base ai loro requisiti temporali e di affidabilità:



Disegno 2: Clasificazione dei sistemi real-time

Soft real-time e Hard real-time

Chiameremo sistema *soft real-time* un sistema nel quale le conseguenze di un fallimento nella temporizzazione sono nello stesso ordine di grandezza dei fallimenti subiti dall'ambiente circostante.

Per fare un esempio consideriamo un sistema banale per l'ordinamento di lettere. Se una lettera viene inserita in un'ordine errato a causa di problemi le conseguenze non sono serie semplicemente la lettera dovrà essere sistemata nel posto corretto (la conseguenza del fallimento è nello stesso ordine di grandezza del fallimento stesso).

Se le conseguenze nel caso di fallimento non saranno nello stesso ordine di grandezza ma ben più critiche allora parleremo di sistema *hard real-time*. Ad esempio il controllore di una centrale nucleare se commette un'errore nella temporizzazione degli elementi di controllo potrebbe causare un disastro.

Alta disponibilità e alta integrità dei dati

Possiamo ulteriormente suddividere i sistemi soft real-time in base ai requisiti di disponibilità e integrità.

Con il termine *disponibilità* (availability) vogliamo indicare il grado di operatività del nostro sistema, dire che un sistema ha alta disponibilità (high availability) significa che è pronto e attivo (teoricamente) in ogni istante. Ad esempio il sistema di switching telefonico ha tra i requisiti quello di essere ad *alta disponibilità* per ovvi motivi di comunicazione.

Per *alta integrità* (high integrity) richiediamo che i dati siano integri e consistenti in tutto il

processo di comunicazione e anche in caso di failure del sistema. Per esempio un sistema bancario che effettua transazioni on-line, nel quale una failure di temporizzazione potrebbe portare a conseguenze sgradite nei conti bancari dei propri clienti, non è accettabile quindi richiediamo alta integrità nei dati.

Hard real-time fail-safe e fail-operational

I sistemi hard real-time si possono suddividere in due ulteriori categorie.

Se nel nostro sistema sono presenti degli stati sicuri allora chiameremo il nostro sistema *fail-safe system*. Ad esempio il sistema di gestione dei segnali in una rete ferroviaria potrebbe decidere in caso di errori di inviare un segnale di “stop” a tutti i treni e questi fermandosi si troveranno in uno stato sicuro nel quale, dal quel punto in poi, non potrà succedere nulla di insicuro (le persone che viaggiano sono salve).

Quindi nei sistemi fail-safe è necessario che la nostra applicazione abbia una copertura più ampia possibile dei possibili errori generabili, il che non è banale.

Nei casi nei quali non sia possibile identificare uno o più stati sicuri (ad esempio un sistema di controllo del volo in un aereo non può bloccare tutto, avremo conseguenze disastrose) dovremo provvedere a garantire un livello minimo di servizio anche in caso di failure proprio per evitare risultati catastrofici.

Reactive o Embedded real-time

Se il nostro sistema è in costante interazione con il mondo esterno (ad esempio la consolle di comando di un aereo) parleremo di *reactive real-time* se il nostro sistema è utilizzato per controllare un particolare hardware installato in un sistema più grande parleremo di *embedded real-time*.

Sistemi guaranteed-response e best-effort

Una ulteriore classificazione dei sistemi RT è si basa sul approccio scelto in fase di implementazione del sistema stesso.

Se il sistema viene creato basandosi su specifiche ipotesi di fallimento e sul carico di lavoro e viene progettato in modo da fornire sempre una risposta con questi vincoli allora parleremo di *guaranteed-response system*. I sistemi guaranteed-reponse sono basati sul principio per il quale le risorse utilizzate nell'implementazione sono adeguate a supportare le ipotesi.

Nella realtà molti sistemi real-time sono basati sull'inadeguatezza delle risorse, infatti se volessimo coprire tutti i possibili casi nei quali il nostro sistema possa fallire la spesa sarebbe troppo elevata pertanto si preferisce allocare dinamicamente o condividere le risorse del sistema in modo da abbassare i costi. Pertanto si fanno delle ipotesi probabilistiche ad esempio sul carico lavorativo.

I sistemi progettati in questo modo vengono chiamati *best-effort system*.

La politica best-effort trova largo utilizzo nei sistemi soft real-time mentre quando trattiamo sistemi hard real-time l'approccio guaranteed-response è quantomai doveroso.

Esempi

Di seguito sono riportati alcuni esempi reali di sistemi distribuiti real-time.

Automazione negli impianti

L'automazione negli impianti produttivi ha rappresentato il primo utilizzo dei sistemi real-time.

Inizialmente gli impianti presupponevano la presenza di personale che controllasse le singole componenti del processo produttivo, col passare degli anni e con il conseguente progresso tecnologico il numero di persone impiegate è diminuito e tutte le fasi di controllo sono passate a strumenti automatici sempre più affidabili e precisi.

Si è passati dal modello *open loop control*, dove l'operatore decideva se il risultato calcolato dai singoli controllori era buono o meno, al modello *closed loop control* dove l'operatore viene via via rimosso dalla catena di controllo e sostituito con processi automatici.

L'arrivo dei sistemi distribuiti negli anni '70 ha portato ad una svolta nel campo dell'automazione industriale riducendo i costi di cablaggio (introduzione di un unico bus real-time per la comunicazione o meglio sistemi wi-fi) e migliorando la manutenibilità del sistema utilizzando componentistica standard.

Telefonia

Un'applicazione ormai matura dei sistemi real-time è il sistema di controllo dello switch telefonico. Questa applicazione è caratterizzata da un'alta disponibilità e facile manutenibilità.

Ad esempio il sistema AT&T Nr.4 ESS è capace di gestire 700.000 telefonate in un'ora (1994). Consiste di un sistema altamente tollerante ai guasti. Il sistema deve essere operativo 24 ore al giorno e le procedure di manutenzione del sistema stesso non devono intaccare le performance.

Reti di sensori wireless (WSN)

Le reti di sensori sono emerse negli ultimi anni come un nuovo paradigma di progettazione per sistemi distribuiti e rappresentano una naturale evoluzione dell'impiego di sensori connessi attraverso reti cablate (vedi Automazione negli impianti). Questo tipo di reti si configurano come reti wireless di sensori piccoli e a basso costo, in grado di raccogliere e distribuire dati ambientali. Questa topologia di rete rappresenta un miglioramento rispetto alle classiche reti di sensori, nelle quali la gestione degli stessi e la raccolta avveniva attraverso un unico nodo centrale.

Le capacità del singolo sensore sono limitate, ma quando grandi quantità di questi sensori cooperano è possibile osservare un fenomeno fisico con grande dettaglio.

Il problema è sempre nella natura di ciò che vogliamo osservare, se la sorgente è il mondo reale anche gli stimoli che riceveremo saranno cadenzati da temporizzazioni reali e il nostro sistema dovrà essere in grado di rappresentarle in modo appropriato.

Concetti di affidabilità

La nozione di affidabilità è strettamente relazionata con la qualità del servizio (QoS) che si vuole offrire. Di seguito vengono indicati 5 attributi importanti che appartengono al termine affidabilità:

- *Reliability*: Misura la continuità del servizio offerto indicata nei termini della probabilità che il sistema fornisca un servizio corretto anche dopo un tempo di esecuzione t .
- *Safety*: Rappresenta la probabilità che il nostro sistema continui a funzionare correttamente anche dopo un evento catastrofico.
- *Availability*: Misura il rapporto tra il servizio offerto correttamente e il tempo complessivo di vita del sistema. Cioè quanto è disponibile il nostro sistema.
- *Maintainability*: Manutenibilità del sistema cioè quanto tempo è necessario per riportarlo in uno stato corretto dopo un errore. Misura la probabilità che il sistema sia ripristinato al tempo t' se ha avuto una failure al tempo t .
- *Security*: ovvero la capacità del nostro sistema nel prevenire accessi non autorizzati.

Ridondanza

La chiave nella progettazione di sistemi fault-tolerant risiede nella ridondanza. Per ridondanza si intende un insieme di informazioni e risorse (duplicate e non) che non sarebbero necessarie se lavorassimo con sistemi ideali. Vengono distinte 3 tipologie di ridondanza in base all'insieme in esame:

- *Ridondanza delle risorse fisiche*
Utilizzo di più risorse allo scopo di tamponare eventuali problemi con la risorsa principale. Ad esempio un potremo replicare un server in modo che, nel caso il server principale dovesse “cadere”, avremo altri server identici pronti all'uso.
- *Ridondanza temporale*
Ripetizione di una azione computazionale o comunicativa in base al tempo. Ad esempio in una protocollo di comunicazione con ACK il messaggio viene rimandato se non si riceve un'ACK dal destinatario.
- *Ridondanza nell'informazione*
Ci riferiamo a specifiche tecniche di codifica. Ad esempio potremo inviare più bit per rappresentare un certo carattere per poter rilevare e correggere degli errori in fase di ricezione.

Inoltre distinguiamo due tipologie di ridondanza:

- *Passiva*
Se nostro sistema di fault-tolerance attiva le risorse fisiche ridondanti solo quando la risorsa principale “cade”. Nei sistemi real-time l'utilizzo della ridondanza passiva è limitato in quanto i tempi per riattivare il sottosistema sono normalmente troppo dilatati.
- *Attiva (chiamata anche “hot-redundancy”)*
Parleremo di ridondanza attiva se il nostro apparato di ridondanza mantiene attive simultaneamente tutte le risorse fisiche duplicate. Nella ridondanza attiva tutte le risorse replicate attraversano gli stessi stati nello stesso tempo. Questa proprietà, chiamata “*replica determinism*” (replicazione deterministica) è molto importante perché ci garantisce di

lavorare con copie esatte e fedeli sia in termini di integrità dei dati sia in termini temporali. Se il nostro sistema non è in grado di garantire questa proprietà allora perdiamo la capacità di tollerare i guasti. Viene normalmente utilizzata nei sistemi real-time.

Replicazione deterministica

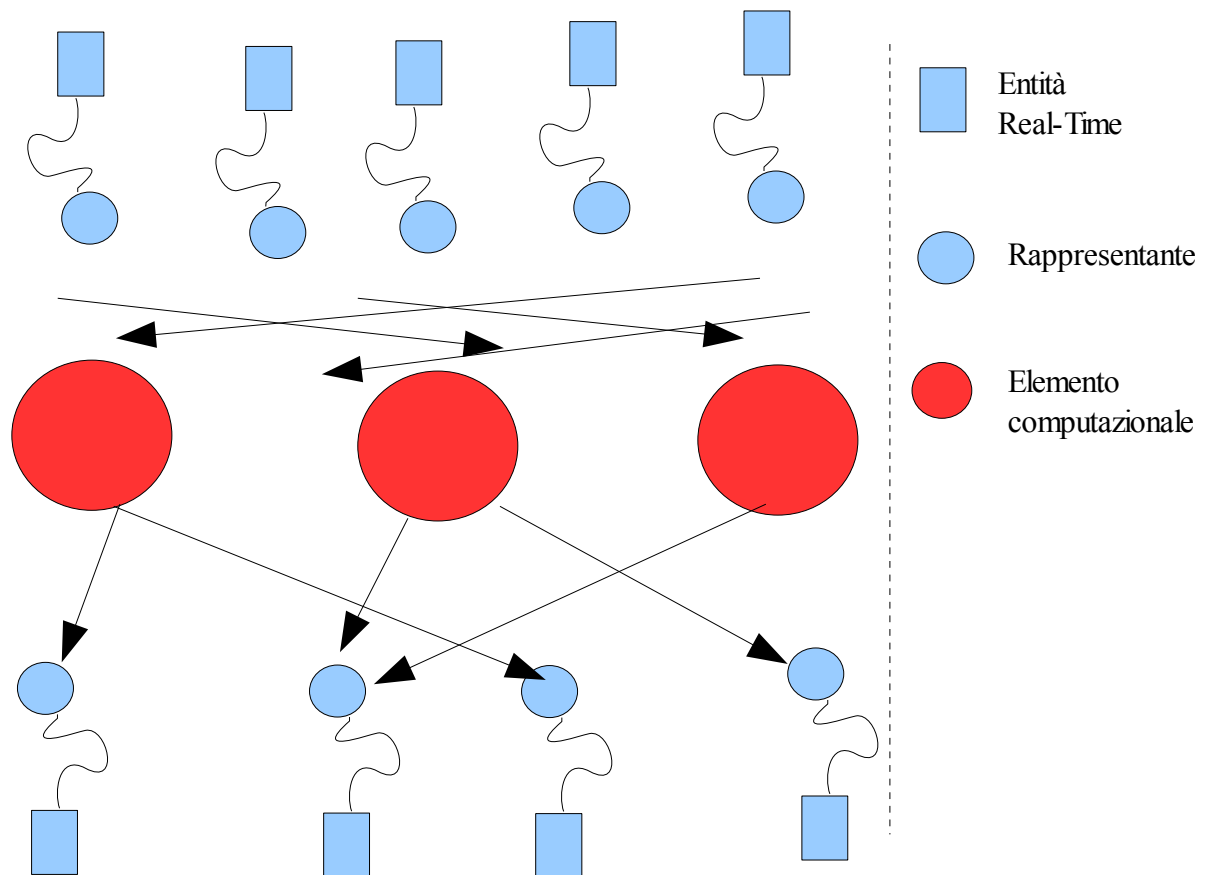
La ridondanza attiva necessita del determinismo nelle repliche. Ovvero l'abilità che deve avere il sistema di replica nel produrre lo stesso output dando in ingresso lo stesso input, richiediamo dunque che ogni replica sia consistente con le altre. In caso contrario non potremo garantire l'affidabilità del nostro sistema.

Modelli di sistemi distribuiti real-time

I sistemi real time sono essenzialmente reattivi e interagiscono con il mondo esterno, reagiscono agli stimoli e producono risultati sotto l'azione di specifici vincoli temporali.

Quindi nel mondo reale ci confrontiamo con variabili diverse da quelle prettamente informatiche, parleremo dunque di *sensori* per rilevare alcune proprietà o parametri dell'ambiente circostante che si evolve in tempo reale.

Prima di iniziare è dunque necessario trovare un modo per mappare quello che vogliamo analizzare del mondo reale nel nostro mondo informatico.



Disegno 3: Generalizzazione di un sistema distribuito real-time

Una entità RT (*real-time entity*) è un elemento dell'ambiente che acquisisce uno o più parametri che ci interessa portare nel nostro sistema. Ad esempio la posizione di una valvola, il livello di fluido in una rete, temperatura, movimento etc.

Un rappresentante (*representative*) di una entità RT ci permette di nascondere la complessità della realtà fisica trasformando l'informazione in un qualcosa di computazionalmente trattabile.

Un elemento computazionale (*computing element*) riceve le informazioni dai representative, processa i dati ricevuti ed eventualmente decide che azioni compiere nel sistema.

La progettazione di un sistema real-time si pone, dunque, due distinti obiettivi:

- la corretta percezione temporale delle entità RT
- la corretta temporizzazione dell'acquisizione delle informazioni e la relativa produzione di risposte.

Il primo obiettivo concerne la progettazione dei collegamenti tra le entità RT e i rappresentanti, una volta risolte le relative problematiche avremo mappato la realtà in una rappresentazione computabile. Da questo punto il progettista proseguirà il suo lavoro ragionando esclusivamente nei termini di un sistema distribuito.

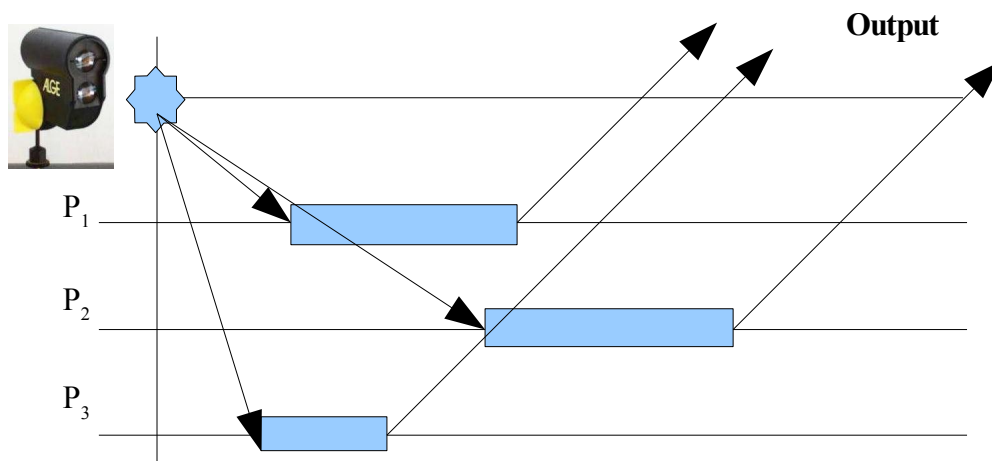
Il secondo obiettivo concerne la computazione dei dati in tempo reale.

Questo modello ci permette una visione semplice del sistema, ma ci nasconde non pochi problemi che renderanno la progettazione complicata proprio per la natura del problema stesso.

Esistono due principali modelli per la progettazione di sistemi distribuiti real-time.

Sistemi Event-Triggered

I sistemi event triggered (ET) sono sistemi che reagiscono direttamente e immediatamente ad eventi esterni connessi al sistema.



Disegno 4: Modello Event-Triggered

Vengono anche chiamati sistemi in attesa (“idle system”) e in effetti sono costantemente in attesa che succeda qualcosa. Quando un evento occorre un messaggio viene inviato all'interno del sistema (ad esempio una persona passa di fronte ad una fotocellula) e le varie unità computazionali produrranno un risultato in base ai dati ricevuti.

Solitamente la progettazione di tali sistemi non è rigida verso certi particolari vincoli in quanto il comportamento viene definito dagli eventi per i quali non è spesso possibile fare assunzioni temporali.

Questo porta ad un modello molto elastico che sarà in grado di supportare picchi di lavoro importanti e momenti di relax usando le risorse via via necessarie in base al carico di lavoro.

Comunque nella progettazione si analizza il caso peggiore e si fanno assunzioni su di esso, pertanto non sono consigliati il loro utilizzo nel caso di sistemi hard real-time ma piuttosto per sistemi misti hard e soft real-time.

Una cosa da tenere in considerazione è che i sistemi ET sono soggetti a quella che viene chiamata

doccia di eventi (*events showers*) che rappresenta solitamente una situazione di eccezione (ad esempio un incendio, una esplosione, un incidente etc.) dove tutti i nodi della rete iniziano a inviare messaggi di errore (moltiplicati in caso di sistemi di ridondanza) e che manderebbero presto in crash l'intero sistema.

Il progettista può dunque cercare di definire la sequenza di messaggi che si possono generare in una catena di messaggi di errore in caso di eccezione nel sistema, facendo ciò può limitare l'effetto *events showers*.

Una soluzione al problema è rappresentata dall'invio ritardato degli eventi in modo da permettere al sistema un'analisi dei dati introducendo un certo lasso di tempo medio tra un evento e il successivo, questo però pone un'ulteriore problematica relativa al tempo medio usato per intervallare gli eventi.

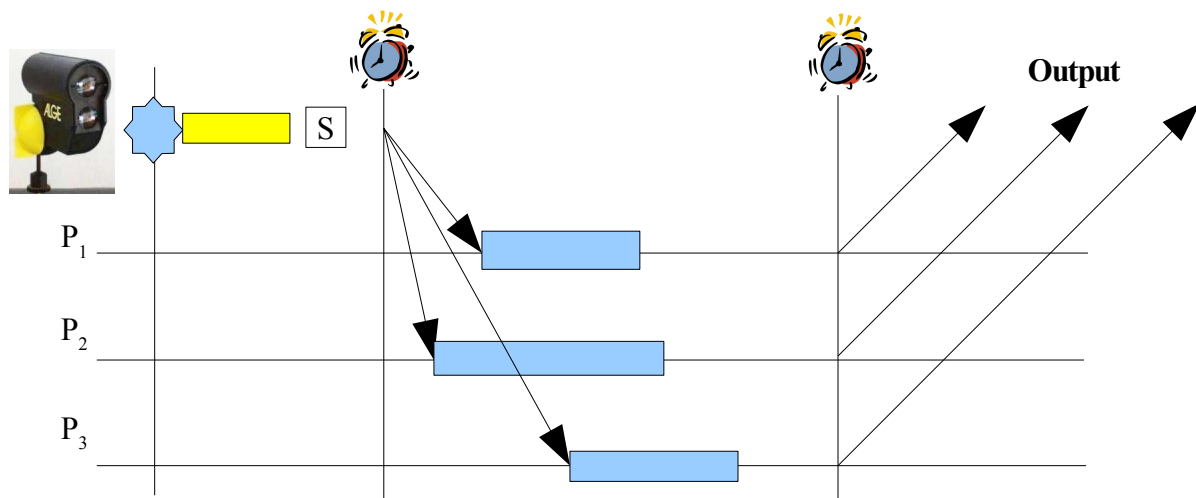
Notiamo dunque il ruolo fondamentale dei *representative* che interfacciano il mondo reale con quello del nostro sistema.

In una doccia di eventi più eventi possono arrivare simultaneamente e dovranno essere eseguiti simultaneamente pertanto diviene necessario l'utilizzo di uno scheduler real-time (vedi p. 13).

Un sistema costruito con l'obiettivo dell'elasticità crea una complessità progettuale non indifferente ma risulta particolarmente vantaggioso quando dobbiamo estendere il sistema stesso (ad esempio vogliamo aggiungere ulteriori entità RT).

Sistemi Time-Triggered

I sistemi time triggered (TT) sono basati su una semplice filosofia: l'ambiente e la sua evoluzione sono sottoposti a delle ben definite assunzioni dove tutto è specificato a priori (carico di lavoro, ritardi, casi peggiori e soprattutto temporizzazione).



Disegno 5: Modello Time-Triggered

Uno degli svantaggi di questo approccio lo troviamo nella gestione di eventi urgenti, in quanto all'occorrenza dell'evento questo dovrebbe aspettare un tempo t prima di essere eseguito perdendo la sua natura di "evento urgente". A favore di questo approccio è la mancanza del "events showers" in quanto ogni evento viene controllato da un timer che gestisce la relativa propagazione nel sistema rendendo tutto deterministico.

A differenza dell'approccio ET il sistema TT lavora sempre a massimo regime sia che si trovi in una situazione di quiete sia di massimo carico lavorativo.

Comunicazione real-time

Un sistema distribuito real-time deve prevedere anche un sistema di comunicazione real-time per abbassare al minimo i ritardi tra le varie componenti.

Un sistema di comunicazione real-time è dunque composto da:

- Protocolli real-time per lo scambio dei messaggi (es. RTP/RTCP)
- Infrastruttura di rete real-time. Una rete che evidenzia certe proprietà temporali (ad esempio ritardo di propagazione) e di affidabilità

Quando si progetta una infrastruttura di rete, diventa necessario essere a conoscenza di alcuni parametri in quanto possono influenzare, pesantemente, sulle prestazioni generali del sistema:

Ritardo nella consegna di un messaggio

La latenza di una rete in un sistema centralizzato non ha grosso peso, diventa invece importante quando si parla di sistemi distribuiti e ancora di più nel caso di sistemi distribuiti real-time. Ad esempio in un sistema telefonico un ritardo eccessivo nell'invio dei pacchetti potrebbe degradare la comunicazione o renderla impraticabile.

Il fatto che la latenza non sia in parte predicibile univocamente impone ai nostri eventi di diventare asincroni in quanto non è prevedibile con precisione quando arriveranno a destinazione.

Ampiezza di banda e costo

Le applicazioni distribuite real-time necessitano di un'ampiezza di banda costante in quanto devono mandare costantemente informazioni provenienti dal mondo reale.

Urgenza dei messaggi

La nostra infrastruttura dovrà riconoscere i messaggi urgenti da quelli non urgenti e quindi instaurare vie preferenziali o ritardare i messaggi a priorità più bassa per garantire il tempo di trasmissione ai messaggi prioritari.

Questo diventa quantomai necessario in caso di sistema event-triggered hard real-time.

Connettività

Il mondo reale non attende che si risolva un blackout per proseguire il suo normale ciclo e allo stesso modo il nostro sistema real-time deve essere in grado di garantire la connettività tra le sue componenti (ridondanza fisica).

Caratteristiche di micronetwork

I sistemi distribuiti sono soggetti alle caratteristiche della microrrete ovvero quell'insieme di proprietà dei singoli componenti dell'infrastruttura.

Ad esempio buffer degli switch, ampiezza delle code, dimensione dei pacchetti, etc. sono tutti fattori che limitano in una qualche misura il nostro sistema e di questo dobbiamo esserne a conoscenza in fase progettuale.

Scheduling real time

Parte fondamentale del processo real-time è lo scheduling delle operazioni da eseguire in quanto, oltre a garantire la correttezza sul risultato, bisogna anche garantire il tempo di esecuzione.

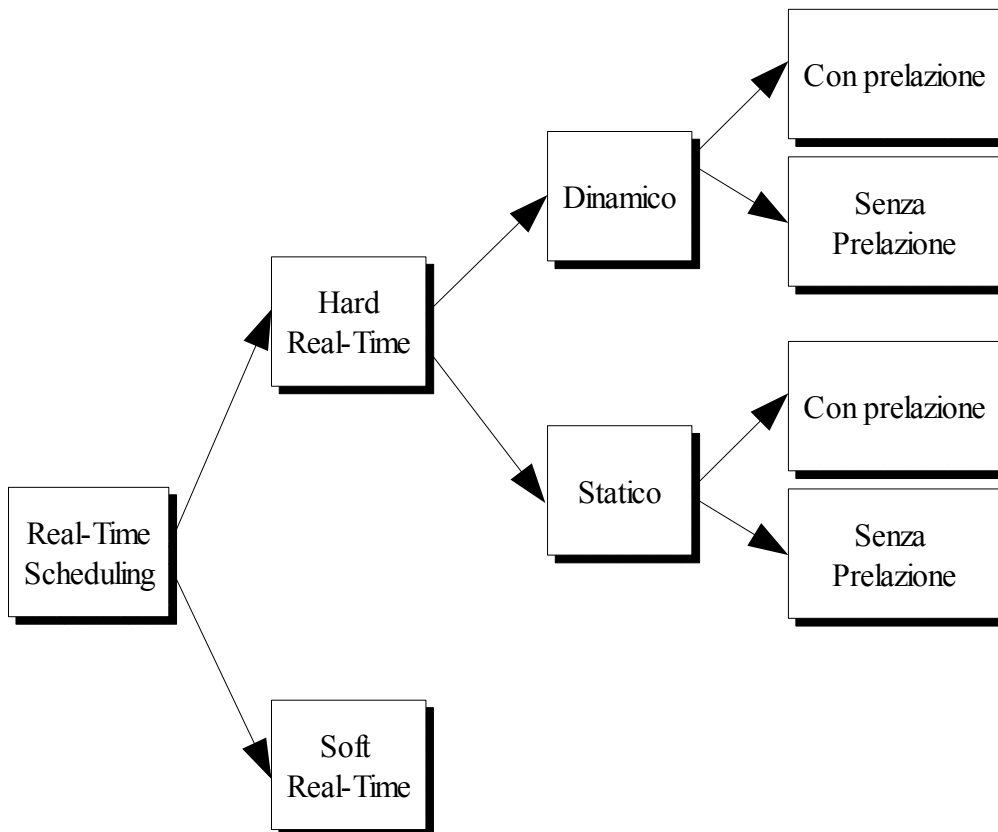
In particolare le decisioni dello scheduling diventano critiche nei sistemi distribuiti hard real-time, dove ogni minimo errore di temporizzazione può avere, come detto, conseguenze disastrose.

Ogni transazione real-time necessita quindi di risorse computazionali, di comunicazione e di memorizzazione pertanto lo scheduling dovrà occuparsi anche dell'allocazione corretta di tali risorse in modo da soddisfare i vincoli temporali.

Una transazione RT può dunque essere scomposta in un set di task e un set di protocolli. I task competono per avere uno dei processori e i controller della rete per avere accesso alla rete di comunicazione.

Quindi il problema dello scheduling lo possiamo decomporre in due problemi fortemente correlati, lo scheduling dei task e lo scheduling delle risorse di rete.

Classificazione degli algoritmi di scheduling



Disegno 6: Tipologie di scheduling

Hard real time e soft real-time scheduling

Nei sistemi hard RT la scadenza dei task critici deve essere garantita a priori in tutti i possibili scenari anticipabili. Questo richiede un design attento delle risorse del sistema in quanto stiamo lavorando a priori. Se stiamo progettando un sistema soft real-time invece, nel caso in cui una scadenza non sia soddisfatta questo non porterà a conseguenze disastrose pertanto potremo accettare un progettazione a basso costo dello scheduler.

Scheduler dinamici e statici

Uno scheduler viene chiamato dinamico (on-line) se fonda le sue decisioni in esecuzione sulla base delle richieste correnti. Gli scheduler dinamici sono quindi flessibili e si adattano con l'evoluzione del sistema.

Uno scheduler viene chiamato statico (pre-run-time) se le decisioni sono prese prima che lo scheduler sia in esecuzione e non potranno essere modificate. Per fare ciò è necessaria una profonda e completa conoscenza pregressa del task che si vuole eseguire (tempo massimo di esecuzione, precedenze, vincoli di mutua esclusione, scadenze, etc.). Tale scheduler ha un basso costo in fase di esecuzione proprio perché non vengono fatti calcoli decisionali.

Scheduler con/senza prelazione

Negli scheduler con prelazione un task può essere interrotto conseguentemente alla richiesta di un task con priorità maggiore. Comunque la prelazione è autorizzata se alcune asserzioni non verranno violate (vedi mutua esclusione). Negli scheduler senza prelazione i task non sono interrotti fino al loro completamento

Scheduling statico

Negli scheduling statici l'ordine di esecuzione dei task in modo che vengano garantite le dipendenze temporali e di risorse viene calcolato off-line. L'implementazione di scheduling statici è tipica in applicazioni time triggered e con task periodici.

Scheduling dinamici

Gli scheduler dinamici determinano quale task deve essere eseguito all'arrivo di un evento e in base allo stato corrente del sistema.

I vari algoritmi di scheduling dinamico si differenziano in base alle assunzioni che vengono poste.

Scheduling di task indipendenti

In questa categoria assumiamo che i task non presentino nessuna dipendenza (ad esempio mutua esclusione).

Rate monotonic algorithm

E' un algoritmo dinamico con prelazione basato su priorità statiche con le seguenti assunzioni:

1. Le richieste per tutti i task appartenenti ad uno specifico task-set per il quale abbiamo vincoli hard real-time sono periodiche.
2. Tutti i task sono indipendenti (assenza di mutua esclusione o dipendenze).

3. La scadenza di ogni task (T_i) corrisponde alla durata del suo periodo (p_i)
4. Conosciamo a priori il tempo di computazione massimo per ogni task
5. I tempi per il cambio di contesto vengono ignorati
6. La somma dei fattori di utilizzo degli n task è limitata superiormente da

$$\mu = \sum \frac{c_i}{p_i} \leq n(2^{\frac{1}{n}} - 1)$$

c_i = tempo di computazione del task i -esimo

p_i = durata del periodo del task i -esimo

L'algoritmo assegna le priorità in maniera statica e sulla base della durata del periodo del task da eseguire. I task con periodo più corto avranno priorità più alta, quelli con periodo più lungo priorità più bassa. In fase di esecuzione quindi verrà scelto ogni volta il task con periodo più corto (priorità più alta).

Earliest-deadline-first algorithm

Viene semplicemente selezionato l'algoritmo con scadenza più prossima.

Least-laxity algorithm

La priorità dei task viene calcolata in base alla differenza tra la scadenza e il tempo di computazione necessario al task. Più questo valore è basso più il task ha priorità alta.

$$laxity_i = d_i - c_i$$

Scheduling di task dipendenti

In questa categoria, più vicina alla realtà e pertanto più interessante, assumiamo che i task presentino delle dipendenze (ad esempio mutua esclusione).

Kernelized monitor

L'algoritmo Kernelized Monitor, proposto da Mok [2], alloca il processore per un quanto di tempo q non interrompibile inoltre si assume che una sezione critica sia eseguibile (quindi inizi e si concluda) all'interno del quanto di tempo. Viene usata per il task scheduling la politica **earliest deadline first**.

L'analisi di schedulabilità fatta in questo protocollo richiede l'uso di upper bounds sui tempi di esecuzione di tutte le sezioni critiche che compaiono nei task, infatti il quanto dovrà essere tarato sulla più lunga sezione critica eseguibile.

Dal momento che tali upper bounds possono essere troppo pessimistici, usare il protocollo kernelized monitor può portare ad una bassa utilizzazione del processore.

Priority Ceiling Protocol

Viene utilizzato per schedulare un insieme periodico di task che hanno accesso esclusivo ad una o più risorse comuni protette da semafori.

Tale protocollo è stato realizzato da Sha, Rajkumar e Lehocky nel 1990 per risolvere il problema dell'inversione della priorità (*priority inversion*).

Inversione della priorità:

Nasce quando un job ad alta priorità deve aspettare che venga eseguito un job a priorità più bassa, tipicamente dovuto al fatto che altre risorse sono usate da task in esecuzione.

Siano 3 task T_1, T_2 e T_3 con priorità $p_1 > p_2 > p_3$, tutti e tre utilizzano la risorsa R e l'algoritmo di scheduling utilizzato è di tipo rate monotonic.

Supponiamo che T_1 e T_3 debbano accedere alla risorsa alla quale è associato il mutex S

Se T_3 inizia ad essere processato e fa un lock(S) prima che T_1 parta allora quando T_1 partirà e tenterà di fare un lock(S) verrà bloccato per un tempo non definito, cioè fino a quando T_3 non farà un unlock(S) liberando la risorsa 'S'. T_1 infatti non può continuare l'esecuzione senza la risorsa 'S' detenuta da T_3 . Risulta evidente che in questo caso T_1 venga penalizzato a favore di T_3 a dispetto dell'ordine delle priorità $p_1 > p_3$.

Se, prima che T_3 faccia un unlock(S), parte il T_2 , allora T_3 verrà sospeso per permettere a T_2 di essere processato in virtù delle priorità $p_2 > p_3$. In questo caso T_1 dovrà attendere che anche T_2 finisca di essere processato. Infatti T_1 è bloccato su T_3 che a sua volta è bloccato su T_2 a dispetto dell'ordine delle priorità $p_1 > p_2$.

L'algoritmo proposto si basa sui seguenti punti:

- Un task T ha il permesso di entrare nella sezione critica solo se la sua priorità è più alta del task che attualmente blocca la risorsa.
- Quando ha terminato l'utilizzo della sezione critica la rilascia dandole la precedente priorità.

Bibliografia

- [1] Sape Mullender, Distributed Systems, Addison-Wesley, 1994
- [2] Mok, A.K., Fundamental Design Problems of Distributed System for Hard Real-Time Environment, 1983
- [4] Couloris – Dollimore – Kindberg , Distributed System Concept and Design 4ed. , Addison Wesley , 2009
- [5] L. Galli, Distributed Operating Systems – Concept & Practice , Prentice Hall , 1999
- [6] Distributed Fault-Tolerant Real-Time Systems: The Mars Approach , IEEE 1984
- [7] F. D'Este , Reti di sensori , 2008

Licenza & Info

Il contenuto di questo testo è da ritenersi “as is” e non mi assumo la responsabilità per eventuali errori sia di forma che di contenuto. Se doveste trovare imprecisioni o errori siete pregati di informarmi e provvederò a effettuare le dovute correzioni.

Creative Common License:

Sei libero di:

riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire, recitare e modificare quest'opera.

Alle seguenti condizioni:

Devi attribuire questo lavoro a Sebastiano Vascon (indicando questo link <http://sebastiano.vascon.it>). Non puoi usare quest'opera per fini commerciali. Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.

Ulteriori dettagli, che vi consiglio di leggere, circa la licenza Creative Common li trovate a questo link: <http://creativecommons.org/licenses/by-nc-sa/2.5/it/>