
POLITECNICO DI MILANO
Dipartimento di Elettronica e Informazione
LAUREA SPECIALISTICA IN INGEGNERIA INFORMATICA

---

# Gestione Dinamica delle Risorse per Sistemi Embedded Multi-Core e Workload Eterogenei

Autore:
**Davide ZONI**
Matr. 724961

Relatore
**Prof. William FORNACIARI**

Correlatore
**Ing. Patrick BELLASI**

Anno accademico 2010-2011

# Ringraziamenti

La conclusione di un lavoro di tesi è sempre un'esperienza unica. Tutto si condensa in poche pagine, per riassumere gli sforzi, le delusioni e si spera, anche i risultati ottenuti.

Anzitutto voglio ringraziare il Prof. William Fornaciari che, in qualità di relatore, ha guidato e supervisionato il lavoro di tesi, 'aggiustando il tiro' quando necessario.

Un ringraziamento particolare va all'Ing. Patrick Bellasi, che mi ha seguito durante tutte le fasi del lavoro, fornendo critiche, suggerimenti e opportunità di dialogo, per arrivare la soluzione finale.

Ovviamente un grazie particolare lo devo ai miei genitori, senza dei quali tutto questo non sarebbe mai stato possibile. In loro ho sempre trovato un appoggio e una spinta nei momenti difficili.

Un pensiero va anche a Elisa, una persona speciale, che ho avuto l'onore di incontrare e che penso sia giusto ringraziare, per la pazienza ed il supporto con cui mi ha seguito nei momenti più difficili.

Un ultimo pensiero va a due persone speciali che mi ricorderanno per sempre l'esperienza al Politecnico di Milano: Marco e Giulia. In loro ho trovato due validi compagni di viaggio che spero di non perdere in futuro.

Oggi concludo un percorso durato sei lunghi anni, e mi sento di ringraziare tutti coloro che mi sono stati accanto e che, indirettamente o meno, hanno contribuito a tutto questo.

Davide

# Indice

# Elenco delle figure

# Elenco delle tabelle

Capitolo 1

# Introduzione

*" Il problema è che se non rischi nulla,*
*rischi ancora di più. "*
Erica Jong

Questo capitolo fornisce una breve introduzione alla problematica della gestione dinamica delle risorse anche in ambito dei sistemi embedded. Inoltre viene dettagliata la struttura del documento e gli obiettivi che si è cercato di raggiungere con questo lavoro di tesi.

## 1.1 La gestione dinamica delle risorse nei sistemi embedded

Lo studio dei componenti finalizzati alla gestione delle risorse non costituisce un problema nuovo, tuttavia rappresenta un attivo settore di ricerca nel campo dei sistemi embedded.
Tale tendenza è giustificata da tre fattori distinti.
Un primo fattore è legato ai recenti progressi nei processi produttivi, che consentono oggi di ottenere dispositivi contenenti decine, ma anche centinaia di unità di calcolo sullo stesso *chip*. In tali architetture diventa

quindi fondamentale gestire in modo efficace ed efficiente questa elevata quantità di risorse.

La necessità di software altamente parallelo costituisce la seconda motivazione della ricerca nel campo dei sistemi per la gestione dinamica delle risorse, i.e. Run-Time Resource Management (RTRM).
La progettazione di software in grado di sfruttare architetture fortemente parallele, è oggi supportata da diverse librerie software [1]. Tali componenti permettono di descrivere un algoritmo e fornire alcune meta-informazioni che il compilatore utilizza, al fine di rendere il codice parallelo.
Tuttavia nessun framework a supporto della programmazione parallela, è in grado di valutare l'insieme di applicazioni attive contemporaneamente nel sistema. E' quindi vero che persino un'applicazione ottimizzata per una specifica architettura, può venir penalizzata durante l'esecuzione quando entra in competizione con altre applicazioni per l'uso delle stesse risorse. Tale problema è dovuto all'isolamento nel quale ogni applicazione viene sviluppata. Si rende dunque necessario un componente in grado di amministrare le risorse in modo dinamico, tenendo conto di una visione d'insieme del sistema.

Il terzo ed ultimo fattore, determinante per lo studio nel campo dei RTRM, è l'approccio di esecuzione orientato alla qualità del servizio.
La rivoluzione tecnologica che ha portato alla nascita dei sistemi *multi-core* e *many-core*, trova propulsione nella necessità di eseguire molteplici applicazioni multimediali, contemporaneamente e sullo stesso dispositivo. Per ognuna di queste applicazioni, risulta sempre più interessante poter specificare i requisiti con i quali deve essere eseguita. Purtroppo sia l'insieme delle applicazioni in esecuzione contemporaneamente in un dato sistema, che l'insieme dei requisiti richiesti da ognuna di esse, nella specifica istanza di esecuzione, risultano non facilmente controllabili a *design-time*.

Per essere più precisi, un sistema di gestione dinamica delle risorse (RTRM) deve amministrare le risorse disponibili in maniera flessibile, in relazione al numero di applicazioni presenti nel sistema stesso ed alle lo-

[1] ad esempio OpenMP *www.openmp.org/*

ro richieste in termini di qualità del servizio.
Inoltre è opportuno tenere in considerazione la necessità di offrire un buon rapporto tra prestazioni e consumi, poichè l'area di lavoro dei sistemi embedded, porta ad operare in sistemi mobili con limitata autonomia energetica.

Sebbene il campo di ricerca sia molto vasto, è possibile riassumere alcune delle caratteristiche che un sistema per la gestione dinamica delle risorse per dispositivi embedded deve garantire [1]:

- riduzione dei costi di progettazione software: dal punto di vista applicativo, le piattaforme MPSoC sono progettate per l'esecuzione concorrente di applicazioni che richiedono una notevole potenza computazionale e disponibilità di memoria. L'astrazione delle funzionalità hardware fornite da un sistema di gestione dinamica delle risorse permette di offrire un'interfaccia di programmazione più facile, che garantisce uno sviluppo più rapido delle applicazioni;

- scalabilità: rispetto alla problematica in esame, tale caratteristica riguarda la possibilità di aggiungere o comunque manipolare la struttura hardware della piattaforma senza la necessità di riprogettare completamente il sistema. E' infatti il gestore delle risorse che dinamicamente si adatterà in modo da ammortizzare le variazioni architetturali;

- flessibilità: le decisioni circa la miglior allocazione devono essere prese dinamicamente considerando un ambiente in evoluzione. L'astrazione offerta dalle risorse hardware della piattaforma permette di implementare algoritmi di allocazione dinamica il più possibile indipendenti dal livello hardware sottostante;

- adattatività run-time: l'utilizzo di informazioni aggiuntive fornite al sistema da parte dell'applicazione consente di aggiustare il comportamento del sistema a run-time;

- affidabilità: l'utilizzo di questo componente permette la gestione del degrado prestazionale dovuto a diversi fattori dinamici, quali sovraffollamento di applicazioni, problemi di calore etc;

- gestione della potenza: la possibilità offerta dal gestore dinamico delle risorse permette una gestione più astratta. Sfruttando le informazioni di sistema è possibile definire politiche mirate a specifici aspetti di interesse che spaziano dalla gestione della potenza ai problemi di gestione del calore.

La richiesta di tali requisiti, garantisce proprietà molto interessanti per un sistema hardware in grado di cooperare con un RTRM. Tuttavia questi requisiti implicano uno studio approfondito riguardo ad esempio a problematiche di comunicazione tra sistema hardware e gestore delle risorse, individuazione dei parametri d'interesse e struttura dello stesso sistema di gestione delle risorse.

## 1.2 Organizzazione del documento

Il documento è organizzato come segue.

Il secondo capitolo fornisce una descrizione delle soluzioni proposte nell'ambito dei sistemi di gestione dinamica delle risorse, destinati al settore dei sistemi embedded. In particolare tratta: i modelli per la decisione circa l'allocazione delle risorse e la struttura del componente di gestione dinamica delle risorse.

Il terzo capitolo introduce la componente innovativa del lavoro svolto. Viene dunque presentato un nuovo modello funzionale per il componente di gestione dinamica delle risorse (RTRM), un nuovo modello di ottimizzazione per l'allocazione delle risorse ed un'euristica per la soluzione del problema di allocazione delle risorse.
Il quarto capitolo presenta i risultati sperimentali dei test condotti, relativi alle soluzioni proposto nel capitolo 3.

Il quinto ed ultimo capitolo illustra le problematiche non ancora analizzate ed i possibili sviluppi futuri.

## 1.3 Obiettivi del lavoro

Il lavoro presentato in questo documento ha l'obiettivo di fornire un modello funzionale per la realizzazione di un componente per la gestione dinamica delle risorse (RTRM Run-Time Resource Manager). In particolare tale modello fa della flessibilità, della componibilità e della effettiva realizzabilità i suoi punti di forza.

La flessibilità viene raggiunta con la presentazione di un modello in grado di catturare il più possibile gli aspetti d'interesse per tale area di ricerca. Flessibilità intesa dunque anche come estendibilità derivante da lavori futuri, i quali potranno evidenziare nuove grandezze di interesse nell'ambito della gestione dinamica delle risorse.

La proprietà di componibilità riguarda la possibilità di costruire un sistema altamente modulare. In tal modo ogni singolo componente svolge una funzionalità specifica e dialoga con il resto del sistema tramite interfacce standard. E' dunque possibile intervenire su ciascun blocco componente il sistema di gestione dinamica delle risorse in maniera autocontenuta. Inoltre un design modulare aiuta l'integrazione di nuovi componenti.

Un ultimo aspetto, non meno importante, consiste nello studio ed implementazione di un sistema effettivamente utilizzabile all'interno di piattaforme reali. Questo implica, nella fase di progettazione, la considerazione di aspetti pratici accanto ad aspetti puramente teorici. Per essere più precisi con aspetti pratici si fa riferimento ai tempi di computazione, alla quantità di memoria effettivamente utilizzata ed alle proprietà della soluzione individuata dal sistema. Inoltre vengono prese in considerazione nuove grandezze, utili a caratterizzare scenari reali d'utilizzo. Tali grandezze vengono discusse dettagliatamente nella sezione 3.1, poichè costituiscono un particolare ulteriore rispetto ai contributi innovativi proposti.

Capitolo 2

# Stato dell'arte

*"Se ho visto più lontano è perché sono salito sulle spalle dei giganti che mi hanno preceduto "*

Isaac Newton

Questo capitolo fornisce una descrizione circa lo stato dell'arte, disponibile al momento della stesura del documento, riguardante i sistemi di gestione dinamica delle risorse. Data la complessità di tali sistemi, è necessario articolare la presentazione in tre sezioni distinte, al fine di fornire il quadro generale del problema.

La prima sezione si sofferma sulla struttura logica di tale componente, caratterizzandone i macro blocchi costitutivi, le rispettive funzionalità ed i meccanismi di comunicazione.

La seconda parte del capitolo presenta il modello matematico di ottimizzazione utilizzato per l'allocazione delle risorse di sistema tra le applicazioni che ne fanno richiesta.

La terza e ultima parte del capitolo si concentra sulle problematiche reali d'implementazione di un sistema di allocazione delle risorse. Viene quindi dato spazio alla descrizione di alcune delle tecniche implementative proposte da studi precedenti.

Con la lettura del capitolo, l'obiettivo è duplice:

- una visione d'insieme delle problematiche riguardanti l'allocazione dinamica delle risorse nel campo dei sistemi embedded;
- uno sguardo alle soluzioni proposte ed utilizzate relative alla gestione dinamica delle risorse, da un punto di vista teorico ed implementativo.

## 2.1 Struttura di un sistema di gestione dinamica delle risorse

Lo studio dei sistemi di gestione dinamica delle risorse per sistemi embedded non risulta del tutto recente. I primi studi in merito a questo argomento risalgono alla fine del decennio scorso [2], ma solo negli ultimi anni si è andata definendo una struttura formale per la modellazione di tali componenti [3] [1].

La funzione del RTRM, in relazione con quanto accennato nella sezione 1.1, è quella di accordare le esigenze tra le richieste di risorse delle applicazioni e le risorse disponibili nel sistema. Questa considerazione pone tale componente, ad un livello logico individuato tra il livello applicativo ed il livello hardware (vedi figura 2.1).

Inoltre, con riferimento alla figura 2.1, è possibile identificare due macro blocchi che compongono il sistema di gestione dinamica delle risorse. Il primo, più a destra nell'immagine, denominato *Run-Time Library*, ha due funzionalità:

- fornisce le primitive per l'astrazione dei servizi di sistema verso le applicazioni. Tali primitive costituiscono un API (Application Programming Interface) utilizzata a *compile-time* dal progettista dell'applicazione. In tal modo è garantito, per l'applicazione, il requisito d'astrazione dalla piattaforma fisica sottostante. Tale requisito risulta essere di grande importanza nella progettazione di un sistema di gestione dinamica delle risorse, come discusso nella sezione 1.1;
- a *run-time* costituisce l'interfaccia di comunicazione tra le applicazioni ed il sistema. In tal modo è permesso al sistema di rendere

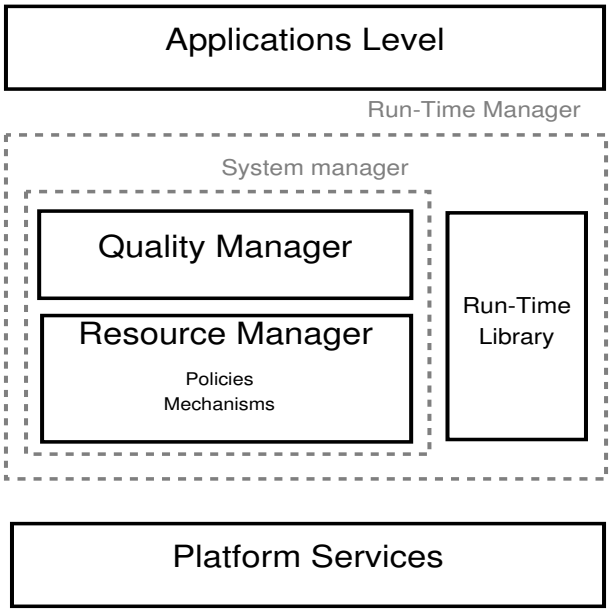



Figura 2.1: Schema funzionale per un generico sistema di gestine dinamica delle risorse [3]. La collocazione del componente, dal punto di vista logico, risulta essere tra il livello applicativo ed i servizi offerti dalla piattaforma.

operative le decisioni prese, mentre le applicazioni possono specificare al sistema una variazione nella richiesta di risorse necessarie al proprio funzionamento.

Il secondo macro blocco, denominato *System Manager* (vedi figura 2.1), si occupa degli aspetti decisionali del problema. In altre parole questo componente fa capo alle tecniche per la selezione delle risorse da assegnare alle applicazioni e dell'allocazione fisica delle applicazioni ai servizi offerti dalla piattaforma.
Tali funzionalità sono affidate a due blocchi di cui il *System Manger* è composto:

- *Quality Manager*: questo componente è deputato alla gestione efficiente dei vincoli di piattaforma ed al soddisfacimento delle richieste delle applicazioni. Tali obiettivi sono ottenuti sfruttando le

informazioni disponibili a *compile-time* ed a *run-time* fornite da applicazioni e piattaforma.
Il fine ultimo di questo componente è definire un'allocazione delle risorse della piattaforma alle applicazioni che ne fanno richiesta. Tale allocazione è tuttavia valutata secondo una visione astratta dalla specifica piattaforma;

- *Resource Manager*: una volta che *Quality Manager* ha selezionato un assegnamento di risorse per ogni applicazione, è necessario allocare fisicamente le risorse alle applicazioni. In altre parole, è necessario trasformare la soluzione astratta fornita dal *Quality Manager* in un'allocazione reale.
  Il componente in esame si occupa di questo aspetto di allocazione, utilizzando informazioni aggiuntive fornite dalle applicazioni. Ad esempio, ad ogni applicazione può essere richiesto di fornire un *task-graph*, che rappresenta la struttura algoritmica dell'applicazione stessa.

Per ognuno dei componenti introdotti sopra, è possibile definire un ulteriore livello di dettaglio andando a specificarne meglio funzionalità e caratteristiche.
Tuttavia lo scopo del lavoro impone un'analisi più dettagliata rispetto al solo componente di *Quality Manager*, del quale è fornito un dettaglio nella figura 2.2.
Per quanto riguarda i componenti di *Run-Time Library* e *Resource Manager*, è disponibile una descrizione dettagliata in diversi lavori [3] [1].

Ad un livello ancora generale, si può affermare che il *Quality Manager* contiene due specifiche funzionalità, indicate come *Quality of Experience* (QoE) e *Operating Point Selection Manager* (OPSM). Tali funzionalità dialogano rispettivamente con il livello applicativo ed il componente di *Resource Manager* (vedi figura 2.2).

Al fine di descrivere le funzionalità offerte da questi componenti, QoE e OPSM, conviene introdurre un minimo di notazione definendo:

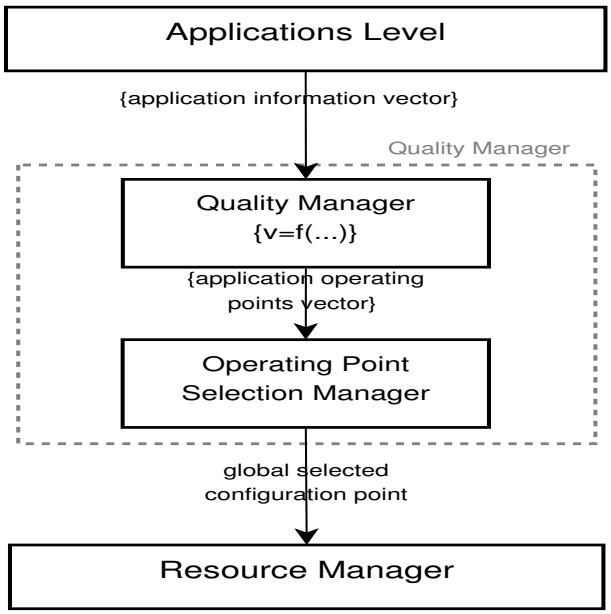



Figura 2.2: Schema dettagliato per il *Quality Manager* [3]. La funzionalità è svolta in due passi consecutivi da due blocchi distinti: QoE Manager e Operating Point Selection Manager.

- $q_{i,j}$ l'insieme dei livelli di qualità $j$ richiesti dall'applicazione $i$. La discussione completa relativamente alla formulazione ed al significato dei livelli di qualità risulta alquanto articolata ed è quindi rimandata alla sezione 2.2.1;

- $r_{i,j}$ indica un vettore contenente l'insieme delle risorse richieste dall'applicazione $i$ per operare nel rispetto dei requisiti $j$. Per ogni elemento di $r_{i,j}$ è indicato il valore di risorsa di una specifica tipologia. Si presuppone dunque che il sistema disponga di risorse classificate secondo differenti tipologie; ad esempio l'insieme delle unità di computazione o la disponibilità di memoria. Tali risorse sono amministrate dal RTRM ;

- $v_{i,j}$ rappresenta, per ogni configurazione $j$ di ogni applicazione $i$, una valutazione del vantaggio per il sistema ad eseguire l'applicazione $i$ nella configurazione $j$ piuttosto che in un altra configurazio-

ne. Tale valore è valutato sulla base delle informazioni contenute in $\boldsymbol{q}_{i,j}$ e in $\boldsymbol{r}_{i,j}$. L'analisi dettagliata di questo elemento viene lasciata alla sezione 2.2.2;

- $d_{i,j}$ l'applicazione $i$ specifica al sistema le informazioni di dettaglio circa la propria struttura algoritmica reale. Tali informazioni possono essere legate ad un'astrazione strutturale di basso livello, come ad esempio il *task-graph*, necessario al *Resource Manager* per effettuare il *mapping* tra i *task* dell'applicazione e le risorse fisiche della piattaforma [1].

E' ora possibile definire la quantità $\left\langle \boldsymbol{q}_{i,1}, \boldsymbol{r}_{i,1}, d_{i,1} \right\rangle, ..., \left\langle \boldsymbol{q}_{i,n}, \boldsymbol{r}_{i,n}, d_{i,n} \right\rangle$ che costituisce l'insieme delle informazioni fornite da ogni applicazione al sistema.
Con la notazione $\langle v_{i,1}, \boldsymbol{r}_{i,1}, d_{i,1} \rangle, ..., \langle v_{i,n}, \boldsymbol{r}_{i,n}, d_{i,n} \rangle$ è definito l'insieme delle informazioni scambiate tra QoE ed il OPSM.
Infine si denota con $v_{i,*}, \vec{\boldsymbol{r}_{i,*}}, d_{i,1}$ l'informazione computata dal *Quality Manager*.

Il processo funzionale del *Quality Manger* si occupa della valutazione di un'allocazione di risorse per ogni applicazione che ne fa richiesta, partendo da informazioni provenienti dalle applicazioni stesse e dal sistema sottostante.
Nella prima fase l'applicazione fornisce le proprie informazioni al sistema, specificando un vettore $\left\langle \boldsymbol{q}_{i,1}, \boldsymbol{r}_{i,1}, d_{i,1} \right\rangle, ..., \left\langle \boldsymbol{q}_{i,n}, \boldsymbol{r}_{i,n}, d_{i,n} \right\rangle$ contenente le differenti modalità operative supportate. Attraverso le funzionalità offerte dal componente di *QoE Manager*, ad ogni modalità viene associato un valore numerico ($v_{i,j}$), che specifica la valutazione che il sistema dà ad ogni modalità $j$ per l'applicazione $i$.
Il nuovo insieme di informazioni $\langle v_{i,1}, \boldsymbol{r}_{i,1}, d_{i,1} \rangle, ..., \langle v_{i,n}, \boldsymbol{r}_{i,n}, d_{i,n} \rangle$, uno per ogni applicazione presente nel sistema, arriva al componente di OPSM. Tale componente ha il compito di valutare l'assegnamento di risorse ad ogni applicazione, rispettando i vincoli imposti dal sistema e cercando di massimizzare la somma dei valori di profitto $v_{i,j}$ assegnati ad ogni applicazione dal *QoE Manager* [2].
L'operazione si conclude con l'output fornito dal blocco di *Quality Ma-*

*nager* di un'insieme di informazioni ( $\overrightarrow{v_{i,*}, \boldsymbol{r}_{i,*}, d_{i,*}}$ ), che corrispondono all'insieme di risorse da assegnare ad ogni applicazione. Tali informazioni costituiscono la soluzione al problema di allocazione dinamica delle risorse, secondo una visione astratta dalla piattaforma.

Le informazioni di allocazione ottima delle risorse arrivano al blocco di *Resource Manager*, che si occupa dell'effettivo *mapping* tra le risorse della piattaforma e le applicazioni.
L'insieme dei modelli teorici che consentono al QoE di produrre un valore di profitto per ogni applicazione e all'OPSM di determinare la soluzione ottima al problema di allocazione delle risorse, sono descritti nella sezione 2.2.

## 2.2 Modello di soluzione

In questa sezione vengono descritti i modelli teorici utilizzati per la realizzazione delle funzionalità di *QoE Manager* e *OPS Manager* introdotti nella sezione 2.1.
La presenza di applicazioni sempre più complesse, con la possibilità di specificare requisiti di prestazione, ha dato vita a nuove sfide per le tecniche di allocazione delle risorse e di scheduling dei processi.

Tuttavia la possibilità di specificare requisiti funzionali, aggiunge alle applicazioni una caratteristica di flessibilità e adattabilità [1] [4].

In primo luogo, è dunque necessario individuare un insieme di tecniche, che permettano al RTRM di sfruttare questa intrinseca flessibilità. La prima parte di questa sezione descrive questa problematica, valutando i lavori proposti fino ad ora.

Una volta determinato un metodo per valutare e sfruttare la flessibilità delle applicazioni, è necessario predisporre un sistema in grado di prendere decisioni rispetto a quali applicazioni allocare e quali requisiti di queste ultime assecondare.
La seconda parte di questa sezione introduce la formulazione matematica al problema, che nel seguito viene denominato problema di *ottimizzazione dell'allocazione*.

### 2.2.1 QoS e Requisiti per le applicazioni

Il concetto di *Quality of Service* (QoS) può risultare alquanto complesso da classificare, ma per essere sintetici può essere espresso come la soddisfazione di un utente o di un'applicazione rispetto ad un servizio fornito [5]. Questo aspetto non risulta strettamente legato al mondo dell'informatica, ma bensì permea ogni ambiente nel quale esite un attore fornitore di servizi ed un attore cliente, in grado di esprimere un giudizio sul servizio offertogli. E' dunque importante restringere fin da subito il campo di valutazione della QoS come concetto relativo al mondo dell'informatica.

Con l'avvento dei *MSoC*, la possibilità di sviluppare applicazioni sempre più complesse e computazionalmente onerose, ha spinto i ricercatori ad introdurre questo termine anche nel ramo dei sistemi operativi e specificatamente nel mondo dei sistemi di gestione dinamica delle risorse. Tale studio prende forza dalla possibilità offerta alle applicazioni di adattarsi rispetto ai cambiamenti del sistema, in funzione del carico, dei servizi offerti e di tutte quelle fluttuazioni a *run-time*, difficilmente catturate in un modello statico realizzato a *compile-time*.

E' dunque necessario per un'applicazione, specificare *a)* le proprie preferenze rispetto all'esecuzione, *b)* la possibilità di rinegoziare tali parametri, *c)* un modo per monitorarne l'andamento durante l'esecuzione e *d)* la possibilità di specificare regole di comportamento.
Tuttavia formulare un modello in grado di fornire un supporto a tutti questi aspetti in modo semplice risulta alquanto complesso, tanto che queste problematiche costituiscono ancora oggi un animato terreno di ricerca [6].
Il problema della gestione della qualità del servizio, può essere suddiviso in tre sottoproblemi interconnessi [6]:

- parametri utili alla specifica delle QoS;
- meccanismi di monitoraggio delle QoS contrattate;
- meccanismi per il *mapping* tra parametri di alto livello verso parametri dipendenti dalla piattaforma.

**A. Parametri da considerare nella specifica delle QoS**

Risulta possibile identificare almeno tre livelli per la specifica delle QoS. Il livello utente, rappresenta la possibilità fornita all'utilizzatore dell'applicazione, di specificare la qualità che egli si aspetta di ricevere dal sistema, intesa come esperienza d'uso.
Il problema fondamentale a questo livello è duplice: da un lato è necessario individuare i parametri sui quale l'utente deve fornire un giudizio, dall'altro l'utente potrebbe non essere interessato o non conoscere il problema, fornendo delle valutazioni non allineate con il suo reale giudizio.

Queste problematiche hanno portato alla formulazione di indici numerici, come espressione del gradimento dell'utente [7] [8]. Tale indice di gradimento può essere specificato al termine dell'esecuzione dell'applicazione, oppure come parametro di valutazione a priori. L'aspetto sostanzialmente più interessante, che emerge dalla ricerca in questa direzione, riguarda l'identificazione di due parametri che possono essere considerati come indici unici per l'identificazione della qualità utente:

- *livello di interesse*: questa grandezza riassume l'interesse che l'utente ha verso l'esecuzione di una specifica applicazione. Tale interesse misura quanto l'utente è disposto a sacrificare, in termini di prestazioni su altre applicazioni, per garantire il funzionamento ad una determinata applicazione. Senza la dimensione di costo, per l'utente non avrebbe senso specificare nulla se non il massimo rispetto a tutti i parametri di QoS;

- *livello di qualità*: rappresenta la qualità che l'utente si attende di ricevere per quell'applicazione. Tale grandezza misura un parametro secondo il quale, al di sotto di certe prestazioni, l'utente non è interessato all'esecuzione dell'applicazione stessa.

Il livello applicativo costituisce il secondo livello utile alla specifica di parametri di QoS. In particolare è possibile esprimere due varietà di parametri:

- *parametri di performance*: questi parametri, di tipo quantitativo, indicano grandezze numerabili, quali il *frame-rate* o la *latenza massima*;

- *parametri di comportamento*: questi parametri servono invece a specificare il comportamento espresso in modo qualitativo, rispetto a come bisognerà comportarsi al variare di alcuni *parametri di performance*.

A questo livello di specifica, esiste un'ampia varietà di alternative che spazia da API che estendono i costrutti del linguaggio, a specifici linguaggi di script o di annotazione del codice sorgente stesso [6] [9].

Il livello risorse, rappresenta il più basso livello per la specifica delle QoS. A tale livello, deve essere possibile specificare le risorse necessarie alle applicazioni, per garantire un funzionamento secondo certi QoS. Tale specifica di risorse può essere *coarse grain* (grossolana) o *fine grain* (dettagliata).
Questo significa che all'applicazione può essere richiesto di specificare i requisiti, circa le risorse necessarie all'esecuzione, in maniera più o meno dettagliata. Un modello che utilizza una specifica delle risorse *coarse grain*, può richiedere di specificare la quantità di risorse per tipologia necessarie all'applicazione. Tuttavia, non viene richiesta la specifica del momento di effettivo utilizzo delle risorse, che vengono tutte allocate all'avvio dell'applicazione stessa. Viceversa, in un modello *fine grain* all'applicazione sono chiesti maggiori dettagli circa l'utilizzo delle risorse [6].

**B. Monitoring dei parametri di QoS**

L'individuazione dei parametri necessari alla specifica dei requisiti di QoS costituisce solamente un elemento del problema. In realtà, ogni applicazione ha la necessità di valutare come effettivamente tali parametri variano a *run-time*. Questa funzionalità, definita *monitoring* [6], lega profondamente il sistema e l'applicazione.
In altre parole, il sistema di gestione della QoS deve mettere a disposizione un meccanismo, che permetta l'effettivo controllo dei parametri negoziati da parte dell'applicazione [10] [11].
Questo aspetto riassume uno dei compiti in carico al componente di *Run-Time Library*, presentato nella sezione 2.1. Inoltre tale componente si lega con la specifica dei *parametri di comportamento* definiti sopra. Infatti, so-

lamente grazie ad una visione dinamica circa l'andamento dei parametri di QoS negoziati con il sistema, l'applicazione è in grado di prendere decisioni adattative.

**C. Mapping dei parametri di QoS**

L'aspetto che tuttavia risulta ancora più difficile da modellare, riguarda il meccanismo di *mapping* della QoS rispetto alla piattaforma fisica, sulla quale vengono eseguiti il sistema di gestione della QoS e le applicazioni. La prima difficoltà riguarda la formulazione dei parametri di QoS, che spesso risultano espressi in linguaggio naturale. Tale proprietà rende difficile valutare, dato un insieme di requisiti, l'insieme delle risorse da utilizzare per vederli garantiti.
Un secondo problema è rappresentato dalla numerosità di tali parametri. Per ogni tipologia di applicazione, è possibile definire specifici parametri che risultano incommensurabili tra classi applicative differenti.

Una delle proposte che costituisce lo standard di fatto per il problema di *mapping* è definito *Modello Utilità* [4].
Il meccanismo di base, per lo scambio delle informazioni di QoS e quindi per riservare risorse tramite il RTRM, può essere suddiviso in tre fasi [4]:

- verifica di ammissibilità: il sistema controlla se può soddisfare i requisiti richiesti dall'applicazione e quindi l'insieme delle risorse necessarie;
- allocazione: il RTRM alloca le risorse all'applicazione, secondo un meccanismo decisionale introdotto nella sezione 2.2.2;
- adattamento *run-time*: il RTRM può cambiare l'insieme delle risorse allocate ad un'applicazione, in relazione alla disponibilità delle risorse stesse o a causa di una nuova richiesta da parte dell'applicazione.

Tale meccanismo rispecchia quanto già illustrato in figura 2.2 nella sezione 2.1, tuttavia è necessario considerare come, da requisiti generici, non immediatamente interpretabili dal RTRM, si possa passare a informazioni numeriche, adatte ad un analisi da parte di un calcolatore.

Specificatamente a questo problema, il *Modello Utilità*, ripreso e completato in altri studi [12] [13], definisce un approccio al problema di tipo funzionale.

Per ogni classe applicativa sono definiti un'insieme di requisiti o livelli di QoS dal sistema. Questi livelli risultano fissi ed immutabili, ossia non è possibile definirne di nuovi. Le applicazioni devono specificare i propri vincoli di QoS in conformità alle opzioni rese disponibili dal sistema di gestione della QoS.

Inoltre sono definiti i seguenti tre concetti:

- un meccanismo di *quality profile*, ossia la possibilità di specificare per ogni applicazione insiemi di preferenze circa differenti tipologie di parametri d'interesse. Ad esempio un' applicazione video può specificare un profilo di bassa risoluzione audio e video oppure un profilo di bassa risoluzione audio ma video di qualità media etc. Queste informazioni vengono definite formalmente come

  $$\boldsymbol{q_j} = (q_{(j,a)}, q_{(j,v)}, q_{(j,i)})$$

  dove il pedice $j$ indica lo specifico insieme di vincoli per l'applicazione $i$, mentre con $a$,$v$,$i$ si indicano i vincoli *audio*, *video* e *immagine*. Ogni applicazione $a$ può specificare un vettore $\boldsymbol{q_a}$ di qualità per ogni configurazione operativa per l'applicazione;

- meccanismo di *quality-resource mapping*, ossia una funzione in grado di determinare, definito un elemento $q_{a,j}$, l'insieme delle risorse di sistema necessario. Supposto di considerare $r_t$ tipologie di risorsa con $t \in [1..T]$, dove $r_t$ indica la quantità numerica della risorsa di tipologia $t$, viene definita la funzione

  $$r_t(q_{a,j}) := q_{a,j} \rightarrow \mathbb{Z}^+$$

  Tale funzione associa per ogni applicazione $a$ e per ogni configurazione $j$ l'insieme di risorse di tipologia $t$ necessario;

- meccanismo di *quality-profit mapping*, ossia una funzione definita come

$$u_{a,j}(q_{a,j}) := q_{a,j} \rightarrow \mathbb{Z}^+$$

che associa ad ogni applicazione $a$ e per ogni configurazione $j$ un valore che indica il profitto generato per il sistema. Il profitto, pur essendo una misura di sistema, è valutato considerando, oltre al dispendio di risorse, anche le prestazioni dell'applicazione e quindi la percezione qualitativa dell'utente finale.

Definite le quantità fondamentali, il *Modello Utilità* costruisce l'obiettivo da perseguire definito come

$$U := \sum_{a=1}^{N} \sum_{j=1}^{M} u_{a,j}$$

ossia massimizzare la somma delle utilità delle singole applicazioni, dove $a \in [1..N]$ indica un'applicazione e $j \in [1..M]$ indica una configurazione o modalità operativa per l'applicazione.

### 2.2.2 Modello di ottimizzazione dell'allocazione

Lo schema di base introdotto tramite il *Modello Utilità*, definisce:

- una procedura per trasformare le specifiche dei parametri di QoS in quantità numeriche. In altre parole viene definito un metodo univoco per valutare ogni insieme di vincoli proposto dalle applicazioni, considerando l'onere sull'uso delle risorse, le prestazioni fornite e quindi la soddisfazione dell'utente finale;

- una funzione che determina l'obiettivo del sistema di allocazione delle risorse. Tale funzione

$$U := \sum_{a=1}^{N} \sum_{j=1}^{M} u_{a,j}$$

si propone di massimizzare l'utilità del sistema;

Sebbene queste definizioni costituiscano un valido punto di partenza per il problema di allocazione delle risorse, è necessario ricordare che tale funzione utilità $U$ è limitata dalle risorse disponibili nel sistema.
Per completare la presentazione è opportuno ricordare che, in ogni istante, l'insieme delle risorse utilizzate nel sistema per ogni tipologia, deve essere inferiore o uguale alla massima disponibilità, ossia:

$$\sum_{act=1}^{Active} \boldsymbol{r}(\boldsymbol{q}_{act}) \leq \boldsymbol{R}$$

dove $\boldsymbol{R}$ indica il vettore contenente la totalità delle risorse per ogni tipologia, mentre $act \in [1..Active]$ indica l'insieme delle configurazioni attive.

Il Modello Utilità definisce un punto fermo utilizzato in molti studi relativi alla gestione dinamica delle risorse non solo in ambito embedded [14] [1] [15] [16].

La definizione proposta costituisce un modello per l'allocazione delle risorse alle applicazioni che ne fanno richiesta. Tuttavia è necessario individuare uno schema di algoritmo che, applicato al *Modello Utilità*, sia in grado di fornire una soluzione. Tale soluzione esprime l'allocazione delle risorse ad ogni applicazione.
Data la natura del problema, per il quale:

- è definito un obiettivo limitato da un insieme di vincoli;
- i vincoli e la funzione obiettivo sono funzioni lineari;

viene definita una formulazione che fa uso della programmazione lineare.

Una gran varietà di problemi decisionali possono essere formulati come problemi di programmazione lineare e nello specifico programmazione lineare binaria. Questa è la forma più immediata, dal punto di vista concettuale, di problema decisionale, dove per ogni oggetto vi sono solamente due alternative.
Il problema viene modellato con variabili binarie $x \in \{0, 1\}$, dove i due

valori rappresentano le scelte possibili. Per essere più formali un problema di programmazione lineare binario può essere modellato con $N$ variabili binarie $x_j \in \{0,1\}$, dove $j \in [1,...,N]$, che corrispondono alla scelta o meno del $j-esimo$ oggetto.
Inoltre viene definito un profitto $u_j$ per ogni oggetto, che rappresenta la differenza di valore ottenuta scegliendo $x = 1$ piuttosto che $x = 0$. Poichè tra tutti i problemi di programmazione lineare binaria, ci si concentra sui problemi di zaino binario, ad ogni oggetto risulta associato anche un costo $c_j$, che viene considerato se l'elemento viene scelto.
Questo ultimo parametro modella l'ambiente reale, dove a fronte di un incremento di profitto vi sono anche dei costi. Normalmente $\sum c_j \leq C$, dove C indica la capacità totale del sistema (o dello zaino).

Ogni oggetto può essere scelto o meno. La scelta dell'oggetto incrementa il valore della funzione obiettivo, ma limita le risorse disponibili. La scelta ottima è quella che massimizza il valore della funzione obiettivo, rispettando comunque tutti i vincoli imposti dalle risorse a disposizione. Una formalizzazione matematica è riportata sotto:

$$maximize \sum u_j x_j \tag{2.1}$$
$$subject\ to \sum c_j x_j \leq C \tag{2.2}$$
$$x_j \in \{0,1\}, \qquad j \in [1,...,N] \tag{2.3}$$

Il problema di zaino binario sopra descritto, costituisce l'elemento fondamentale per i meccanismi di ottimizzazione per l'allocazione delle risorse.
Tuttavia, il problema di allocazione delle risorse possiede una complessità non gestibile dal problema di zaino binario. Nella pratica, il modello di *zaino multi-dimensione multi-unità* risulta il modello ottimizzazione per il RTRM, utilizzato dalla totalità dei lavori analizzati.

Il modello di MMKP (multi-dimension multi-resource knapsack problem) rappresenta un'estensione per il problema dello zaino binario. Tale modello presenta due varianti:

- considera la presenza di più tipologie di risorsa, espresse come vettore $(R_1, ..., R_T)$ dove T indica la numerosità delle tipologie di risorsa;
- per ogni elemento sono possibili più configurazioni $(u_i, r_i)$. Ogni elemento è formato dal vettore di configurazioni $\overrightarrow{(u_{i,j}, (r_{i,j,1}, ..., r_{i,j,k}))}$, dove $i \in [1..N]$ indica il numero di oggetti tra cui scegliere, $j \in [1..M]$ indica per ogni oggetto il numero di configurazioni disponibili e $k \in [1..K]$ indica la tipologia di risorsa. Inoltre per ogni oggetto può essere scelta una sola configurazione.

Importante è notare come all'interno di ogni oggetto e ogni configurazione non vi sia un elemento di costo unico, ma un vettore ($\boldsymbol{R}$), che rappresenta il costo della configurazione rispetto alla tipologie di risorse. Assumendo che vi siano $N$ applicazioni o oggetti, ognuna con $M$ configurazioni, $T$ tipologie di risorsa, ognuna in quantità limitata $R_t$, la formulazione del MMKP diviene

$$maximize \sum_{i=1}^{N} \sum_{j=1}^{M} u_{i,j} x_{i,j} \tag{2.4}$$

$$subject\ to \sum_{i=1}^{N} \sum_{j=1}^{M} r_{i,j,t} x_{i,j} \leq R_t, \qquad \forall t \in [1..T] \tag{2.5}$$

$$\sum_{j=1}^{M} x_{i,j} = 1, \qquad \forall i \in [1..N] \tag{2.6}$$

$$x_{i,j} \in \{0, 1\} \qquad \forall i \in [1..N],\ \forall j \in [1..M] \tag{2.7}$$

dove la disequazione (2.5) indica che per ogni tipologia di risorsa esiste un vincolo di disponibilità che non deve essere ecceduto.

Un'ultima considerazione in merito al MMKP, riguarda il vincolo (2.6), che impone sempre la scelta di una configurazione per ogni applicazione attiva.
Tale ipotesi, presente in tutti i lavori analizzati, vincola notevolmente il problema ed è oggetto di discussione nella sezione 3.1.

## 2.3 Schemi pratici di soluzione

Il *Modello Utilità*, definito nella sezione 2.2, ha portato alla definizione di un problema di zaino multi-dimensione multi-unità (MMKP Multi-dimension Multi-unit Knapsack Problem), che costituisce una variante del problema di zaino binario. Tale problema di ottimizzazione combinatoria, indicato nel seguito solamente come MMKP, costituisce la formulazione utilizzata per la modellazione del problema di allocazione delle risorse.
Tuttavia, dato che il problema di zaino binario è di classe NP-difficile, anche il MMKP costituisce un problema NP-difficile [17].
Questo aspetto implica un tempo di computazione che, nel caso pessimo, cresce esponenzialmente con la dimensione del problema.

Lo scopo di questa sezione è offrire una panoramica su una famiglia di tecniche utilizzate per la soluzione del problema di gestione dinamica delle risorse, che utilizza la formulazione del MMKP come modello matematico di ottimizzazione.

La natura del problema in esame, richiede che la soluzione al problema di allocazione sia individuata in tempi contenuti e con un buon grado di approssimazione a quella ottima, valutata tramite la soluzione del MMKP.
Questa necessità giustifica la grande quantità di lavori per la valutazione di algoritmi approssimati, chiamati anche euristici, in grado di soddisfare i requisiti di velocità di computazione e buona approssimazione per la soluzione trovata.

Un procedimento euristico sfrutta la specifica struttura del problema in esame e delle proprietà garantite dalle soluzioni ottime per avvicinarsi il più possibile all'*ottimo teorico*, ovvero alla soluzione valutata utilizzando un algoritmo esatto per lo stesso problema.
Quindi, la soluzione individuata tramite un procedimento euristico, definita *soluzione approssimata*, può divergere completamente dall'*ottimo teorico* anche per piccole variazioni alla struttura del problema in esame.

Esistono numerose famiglie di algoritmi euristici, che si differenziano

in base al modo utilizzato per comporre la soluzione o alle tecniche per la decisione di quali elementi includere nella soluzione. In questo documento viene trattata solamente la famiglia delle euristiche *greedy* per due motivi principali:

- risultano concettualmente abbastanza facili da implementare e altrettanto facilmente ottimizzabili rispetto al problema considerato. La soluzione viene generata in modo incrementale valutando per ogni elemento se deve essere inserito o meno nella soluzione stessa. Una volta valutato, l'elemento non viene considerato ulteriormente;
- costituiscono la famiglia di euristiche più utilizzata nell'ambito dei RTRM in campo embedded. Inoltre sembrano offrire i migliori risultati [1] [16].

Nel seguito sono definiti gli elementi principali per una formalizzazione generale di algoritmo *greedy* proposta nel listato 2.1:

- *Item*: rappresenta l'insieme delle scelte possibili per il problema considerato;
- *Sol*: l'insieme che parte vuoto e si arricchisce di elementi di Item fino a rappresentare la soluzione completa;
- *getBest()*: la funzione che tra tutte le scelte trova la migliore. Questa funzione racchiude il significato dell'algoritmo, poichè una volta nota la struttura del problema, in essa sono definite le proprietà utili ad individuare la scelta migliore. Dipende fortemente dal problema in esame;
- *feasible()*: una funzione che controlla che la soluzione parziale generata rispetti i vincoli del problema.

Per introdurre l'approccio tramite euristiche *greedy* alla soluzione del MMKP, viene presentata un'euristica per la soluzione del problema di zaino binario. Tale presentazione è utile in quanto il MMKP è un'estensione del problema di zaino binario e l'approccio euristico *greedy* utilizzato sfrutta questa analogia.

```
1  Solution Greedy(Item)
2  {
3    Solution Sol= ∅;
4    Temp=Item;
5    do
6    {
7      e = getBest(Temp);
8      Temp = Temp \ {e};
9      if(feasible(Sol∪e))
10       Sol = Sol∪e
11   }
12   while(Temp ≠ ∅)
13   return Sol;
```

Listing 2.1: Schema generale di algoritmo greedy

Il problema di zaino binario, introdotto nella sezione 2.2.2, presenta una singola risorsa $R$ che determina il vincolo di capacità del problema. Ogni oggetto d'interesse per il problema ($i$) determina un profitto ($u_i$), ossia un valore che incrementa la funzione obiettivo. Inoltre ogni oggetto $i$ determina un utilizzo di alcune risorse di tipo $R$ disponibili ($r_i$). Per ogni elemento $i$ la scelta è se inserirlo nella soluzione o meno. La soluzione ottima inserisce gli oggetti $i$ la cui somma dei profitti determina un massimo per la funzione obiettivo, senza eccedere la disponibilità della risorsa $R$.
L'euristica proposta è riassunta in tre passi:

1. costruisce per ogni oggetto d'interesse un rapporto tra il profitto generato e l'uso delle risorse, definito come [17]:

$$\tau_i := u(i)/r(i)$$

2. costruisce un ordinamento totale decrescente dei valori $\tau_i$ calcolati;

3. Per comporre la soluzione finale, sono presi gli oggetti $i$ seguendo l'ordinamento costruito al passo 2 fino all'esaurimento delle risorse $R$ disponibili.

Tale algoritmo è *greedy* nel senso che sceglie gli elementi migliori secondo la *cifra di merito* $\tau_i$ ed una volta presa una decisione non torna mai indietro.

Non è detto che un tale approccio individui la soluzione esatta, però garantisce tre vantaggi:

- esiste un unico punto critico, ossia la scelta della cifra di merito per ordinare gli elementi. In tal modo ogni sforzo può essere concentrato in questo punto;
- vi è la sicurezza di individuare una soluzione, se esiste, anche se non ottima;
- il tempo computazionale cresce linearmente con il numero di elementi presenti nel problema.

Queste tre caratteristiche risultano di grande interesse per il RTRM, dove le decisioni devono essere prese in un tempo contenuto, il tempo di calcolo deve essere il più possibile predicibile e deve essere garantita una soluzione anche se non ottima dal punto di vista del problema formale.

All'interno della famiglia degli algoritmi *greedy*, le proposte analizzate per la soluzione del MMKP, implementano una variante dell'euristica appena presentata, per il problema di zaino binario.
Il primo problema che deve essere superato riguarda la molteplicità delle risorse da considerare in un MMKP. Se per il problema di zaino binario può essere costruito $\tau_i := u_i / r_i$, nel problema di MMKP per ogni oggetto $i$ la quantità $r_i$ diventa il vettore

$$\boldsymbol{r_i} := [r_{(i,1)}, r_{(i,2)} ... r_{(i,t)} ... r_{(i,T)}]$$

dove supposto $t \in [1..T]$ rappresenti le tipologie di risorsa considerate dal problema, ogni elemento di tale vettore indica l'utilizzo di una differente tipologia di risorsa.
Questo significa che il denominatore della cifra di merito $\tau_i$, in quanto vettore, richiede una manipolazione prima di poter essere utilizzato, poichè al numeratore è presente uno scalare.

Una soluzione al problema consiste nell'aggregazione delle risorse al fine di costruire un unico scalare, da usare come denominatore nella costruzione della cifra di merito $\tau_i$ [18]. Per ogni tipologia di risorsa $t \in [1..T]$, è nota $Busy_t$, ossia il numero di risorse di tipologia $t$ utilizate. Inoltre è definito $R_t$ che rappresenta la disponibilità massima della risorsa di tipologia $t$. Ricordando che in un MMKP ogni oggetto $i \in [1..N]$ presenta differenti configurazioni $j \in [1..M]$, viene costruito

$$\tau_{i,j}{}_{MMKP} := \frac{u_{i,j}}{\sum_{t=1}^{T} Busy_t * r_{i,j,t}/R_t}$$

Questa cifra di merito costituisce il punto di partenza per molti altri lavori, ma presenta un difetto evidente: se le risorse risultano tutte libere, la quantità $Busy_t \to 0$ e quindi $\tau_{i,j}{}_{MMKP} \to \infty$ [18].

Sono state introdotte differenti soluzioni a questo problema. Tra tutte, viene riportata quella considerata più interessante [14] [16]. Per ogni oggetto $i$ la configurazione $j$ a minimo utilizzo di risorse, è definita come [19]:

$$minConf(i) := argmin_{j \in [1..M]}(u_{(i,j)}), \quad \forall i \in [1..N]$$

Inoltre viene costruito un elemento

$$\boldsymbol{minConf}_{sys} := \sum_{i=1}^{N} \boldsymbol{r}_{i,minConf(i)}$$

che rappresenta un vettore. Ogni elemento di tale vettore contiene la somma dell'uso di risorse di tutte le configurazioni minime, per ogni tipologia di risorsa .
In tal modo la definizione della cifra di merito $\tau_{i,j}{}_{MMKP}$ diviene:

$$\tau_{i,j}{}_{MMKP} := \frac{u_{i,j}}{\sum_{t=1}^{T} minConf(t)_{sys} * r_{i,j,t}/R_t}$$

dove al posto dell'elemeto $Busy_t$ è utilizzato $minConf(t)_{sys}$.

Una volta individuata una cifra di merito per il MMKP, le differenti implementazioni si differenziano in base al tipo di ordinamento degli

elementi. E' possibile utilizzare un ordinamento globale determinando un'unica lista contenente le configurazioni di ogni applicazione ordinata [19]. In altri lavori viene definito un ordinamento delle configurazioni di ogni applicazione producendo così una lista di configurazioni per ogni applicazione da schedulare [1] [2] [16] [20].

Alcuni degli algoritmi euristici *greedy* proposti, riescono ad essere particolarmente veloci nella computazione di una soluzione approssimata, che comunque rimane molto prossima al valore dell'ottimo teorico [19].
Questa caratteristica può essere sfruttata per introdurre un approccio mirato a discriminare ulteriormente le varie configurazioni proposte al sistema dalle applicazioni. Preso un sistema di esempio con $n$ applicazioni $n \in [1..N]$ ed $m$ configurazioni $m \in [1..M]$, l'algoritmo euristico assegna un valore per ogni $\tau_{n,m}$. Successivamente ordina in base al valore di $\tau_{n,m}$ le configurazioni di ogni applicazione e procede alla selezione in maniera *greedy*.
Questo significa che l'algoritmo non è in grado di scartare alcuna configurazione, sulla base dei requisiti richiesti. Supposto che le $m$ configurazioni di ogni applicazione siano ordinate decrescentemente per QoS, le prime risultano le più interessanti per le applicazioni.

L'approccio considerato in [21], ipotizza di fornire all'ottimizzatore solo una parte delle configurazioni per ogni applicazione. In tal modo viene costruita una soluzione senza considerare le configurazioni che non sono state specificate. Se tali configurazioni rappresentano le configurazioni meno interessanti per le applicazioni ed esiste una soluzione l'algoritmo termina. Nel caso non sia trovata una soluzione, l'ottimizzazione viene ricalcolata aggiungendo alcune configurazioni prima non considerate. Tale procedimento corrisponde ad un rilassamento progressivo dei vincoli imposti dalle applicazioni. In altre parole il sistema prova ad allocare tutte le applicazioni utilizzando le migliori configurazioni, verosimilmente quelle più vincolanti in termini di risorse richieste. Se tale soluzione non può essere individuata, il sistema prova a trovare una soluzione meno performante. Questa procedura prosegue fino a utilizzare, nel caso pessimo, tutte le configurazioni fornite da ogni applicazione per l'ottimizzazione.

## 2.4 Conclusioni

In questo capitolo è stata presentata la struttura di un sistema per la gestione dinamica delle risorse. Particolare attenzione è stata dedicata al modello per l'allocazione ottima delle risorse ed alle tecniche utilizzate per la computazione efficiente di tale soluzione.

Questa analisi attingendo a lavori, che dalla fine degli anni novanta proseguono fino ad oggi, dimostra come il problema della gestione dinamica delle risorse sia un tema di grande interesse scientifico.

Nel seguito dell'esposizione, questo capitolo viene referenziato per confrontare i cambiamenti introdotti dal lavoro svolto e le componenti interessanti riutilizzate per produrre una nuova soluzione.

Capitolo 3

# Un nuovo modello per la gestione dinamica delle risorse

*“ Per cambiare qualcosa, costruisci un modello nuovo che renda la realtà obsoleta. ”*

Richard Buckminster Fuller

Questo capitolo presenta gli elementi innovativi introdotti durante il lavoro di tesi. Tali contributi, di natura teorica e pratica, mirano a raffinare gli studi condotti sino ad ora, inerenti la gestione dinamica delle risorse in ambito embedded.

Se da un lato molto lavoro è stato fatto in questo settore, dall'altro è pur vero che, la continua evoluzione delle architetture, legata al progresso tecnologico, introduce continuamente nuovi elementi da tenere in considerazione nello studio di tali sistemi.

In questo capitolo viene affrontato il problema da un punto di vista teorico, nella prima parte, ed implementativo, nella seconda parte. Vengono introdotti alcuni elementi che permettono, a nostro parere, di ottenere un modello che meglio rappresenta e gestisce problematiche reali, cui un sistema di gestione dinamica delle risorse (RTRM) deve far fronte.

La trattazione viene divisa in cinque parti.
Nella prima parte sono elencati in dettaglio gli aspetti innovati di questo lavoro.
Nella seconda parte è introdotta una rappresentazione formale, che viene utilizzata per il resto del documento, utile a rendere più chiara la comprensione e più immediata l'esposizione dei contenuti.
Nella terza parte è presentato un nuovo modello funzionale astratto per il sistema di gestione dinamica delle risorse ed un confronto rispetto alla struttura proposta sino ad ora.
Nella quarta sezione viene presentato un nuovo modello di ottimizzazione per l'allocazione delle risorse alle applicazioni presenti nel sistema.
La quinta ed ultima parte introduce un'euristica per il modello teorico di ottimizzazione proposto.

## 3.1 Elementi innovativi del lavoro proposto

Il lavoro di ricerca, svolto nell'ambito dei sistemi per la gestione dinamica delle risorse (RTRM), ha prodotto una soluzione di carattere generale, sulla quale è possibile tuttavia intervenire, sotto certi aspetti.
In questa sezione vengono descritti gli aspetti innovativi del lavoro svolto sotto forma di miglioramenti allo stato attuale della ricerca.

**La clusterizzazione delle risorse**

Il miglioramento dei processi produttivi permette di ottenere oggi decine o addirittura centinaia di unità di calcolo all'interno di un singolo chip [1].
Risulta altresì vero che una delle modalità di organizzazione di queste unità di calcolo è il cluster, dove per cluster si intende un insieme di risorse omogenee per capacità di calcolo e processo produttivo che sono accomunate da alcune proprietà. Tali proprietà possono essere rappresentate dall'avere una stessa sorgente di clock, oppure la stessa sorgente di alimentazione. Un punto fermo rimane dunque, che l'analisi di un sistema che presenta unità di calcolo organizzate in cluster [22] [23], deve considerare questa caratteristica nell'eventuale formulazione di un mo-

dello. Questa caratteristica risulta importante, poichè ad una effettiva disponibilità di risorse non è detto che corrisponda un'effettiva allocazione possibile [1].
A titolo di esempio, si suppone di avere due cluster da quattro PE (Processing Element) ciascuno ed un applicazione che utilizza in totale sei PE. Inoltre vale l'ipotesi che un'applicazione può essere allocata su un singolo cluster (vedi [23]). Sebbene l'applicazione richieda una quantità di risorse inferiore a quella disponibile nel sistema, l'allocazione pratica risulta impossibile.
Rilassando l'ipotesi di allocazione dell'applicazione su singolo cluster l'allocazione diviene possibile al prezzo di alti costi di comunicazione (vedi [22]).
Ovviamente un modello teorico di ottimizzazione delle risorse basato su un problema di MMKP (vedi 2.3), non è in grado di catturare la problematica della clusterizzazione delle risorse e quindi può produrre allocazioni di risorse non utilizzabili praticamente.
Fra tutti i lavori analizzati, tale problematica è stata presa in considerazione solamente in [1]. Tuttavia la soluzione proposta, utilizza ancora un modello di ottimizzazione dell'allocazione basato su un problema MMK. Questo determina un impatto, in alcuni casi imbarazzante, sulla bontà della soluzione individuata.

**Convivenza di applicazioni critiche e best effort**

In ogni lavoro precedente, l'utilizzo di un RTRM è stato confinato agli scenari d'uso caratterizzati dalla sola presenza di applicazioni definite *critiche*. Supponendo che per ogni applicazione siano presenti differenti modalità operative, un'applicazione è considerata *critica* se, al termine del processo di ottimizzazione, risulta eseguita in una delle sue modalità.
Questo significa che, il sistema di gestione delle risorse ha libertà nella scelta della modalità da eseguire per ogni applicazione, ma l'applicazione deve comunque essere eseguita.
Tale requisito si nota nel modello di ottimizzazione MMKP presentato nel capitolo 2. Esiste infatti un vincolo che impone, per ogni applicazione, la scelta di una configurazione da parte del RTRM.

Se da un lato, l'utilizzo di applicazioni *critiche* e la garanzia che queste vengano eseguite, costituisce una proprietà interessante per determinati scenari, tale ipotesi non esaurisce tutti gli scenari d'utilizzo possibile.
Gli *smartphone* di nuova generazione costituiscono un valido esempio nel quale, a fianco di applicazioni definite *critiche*, convivono applicazioni di tipologia *best-effort*, per le quali il servizio viene garantito nel limite del possibile dal sistema.
Ad esempio si può supporre che, per un sistema *smartphone*, le applicazioni di chiamata vocale e messaggistica sms, possano essere etichettate come *critiche*, poichè racchiudono le funzionalità di base del dispositivo stesso. Spesso tali applicazioni sono fornite dal *system integrator* e risultano altamente ottimizzate per la piattaforma su cui vanno ad operare.

Tutte le altre tipologie di applicazioni installate sul dispositivo, costituiscono la classe di applicazioni *best-effort*. Questa classe rappresenta la possibilità, per ogni utente finale, di personalizzazione del dispositivo, offerta dal *system integrator* e risulta quindi un valido incentivo per l'acquisto del dispositivo stesso.
Le applicazioni di classe *best-effort* vengono eseguite dal sistema fino a che non vi è la necessità di interromperle, per dedicare attenzione ad applicazioni di più alto livello, o perchè le risorse di sistema sono state saturate [24].

**I costi legati all'utilizzo delle risorse**

La modellazione di un sistema per la gestione dinamica delle risorse, deve offrire la possibilità di gestire le risorse disponibili in modo accurato. Questo significa offrire la possibilità di limitare l'utilizzo delle risorse, secondo criteri indipendenti dalle applicazioni. Tali criteri possono rappresentare un livello di ottimizzazione, ad esempio delle prestazioni o dei consumi legati alla tipologia di piattaforma sottostante. E' dunque necessario che il sistema di gestione dinamica delle risorse sia in grado di valutare le diverse implementazioni dell'hardware che amministra, per sfruttarne al meglio le caratteristiche.

Tutti i lavori proposti in questo ambito non considerano tale livello di

gestione e questo risulta evidente anche dal modello di ottimizzazione presentato nella sezione 2.2.
L'allocazione ottima delle risorse viene determinata sulla base della disponibilità delle risorse, ma il sistema non è in grado in alcun modo di limitare l'utilizzo di una data tipologia di risorsa.

**Modellazione di un sistema flessibile**

Il miglioramento nei processi produttivi determina l'avvento di nuove problematiche da considerare con una certa frequenza. La progettazione di un sistema per la gestione dinamica delle risorse abbraccia un certo numero di aree d'interesse, come evidenziato nel capito 1. Questo aspetto suggerisce la ricerca di un modello flessibile che permetta la distribuzione delle funzionalità verso componenti ben definiti. In tal modo risulta più semplice l'individuazione di eventuali *overhead* di computazione, oppure l'aggiornamento di una specifica funzionalità del sistema, per adattarlo a nuove esigenze.
Tale necessità è stata già evidenziata (vedi [1]), tuttavia il modello formale di ottimizzazione presentato nella sezione 2.2 difetta di alcune caratteristiche:

- non sono considerate politiche per la gestione delle singole risorse;
- non è considerato un supporto per adattare il comportamento del RTRM alla specifica piattaforma che amministra.

## 3.2 Un formalismo di presentazione

La presentazione del lavoro svolto, si serve di modelli formali ed algoritmi. L'obiettivo di questa sezione è introdurre un formalismo che consenta una lettura più scorrevole e al contempo una più facile esposizione dei contenuti.
Il formalismo introdotto è suddiviso in due parti. Nella prima viene descritto l'insieme dei simboli utilizzati per la descrizione delle applicazioni che caratterizzano il sistema in esame. Nella seconda parte viene

presentato un formalismo adatto a descrivere in modo astratto il sistema di gestione dinamica delle risorse.

**Un formalismo per il modello di applicazione**

In questo paragrafo vengono descritte le grandezze d'interesse che caratterizzano l'oggetto applicazione all'interno del modello sviluppato. Come già accennato, l'applicazione definisce determinate informazioni da fornire al sistema prima dell' esecuzione.

| | Preference | clustered resource | non clustered resource submatrix (typology divided) | | | Implementation Details |
|---|---|---|---|---|---|---|
| AWM_1 | value1 | r_cl(1,k) | r_ncl(1,k) | ... | r_ncl(1,T) | Detail(AWM_1) |
| AWM_2 | value1 | r_cl(1,k) | r_ncl(1,k) | ... | r_ncl(2,T) | Detail(AWM_2) |
| ... | ... | ... | ... | ... | ... | Detail(AWM_3) |
| AWM_M-1 | value_M-1 | r_cl(M-1,k) | r_ncl(M-1,k) | ... | r_ncl(M-1,T)) | Detail(AWM_M-1) |
| AWM_M | value_M | r_cl(M,k) | r_ncl(M,k) | ... | r_ncl(M,T) | Detail(AWM_M) |

Figura 3.1: Informazioni fornite dall'applicazione al sistema di gestione dinamica delle risorse.

Tuttavia questa rappresenta solamente una visione logica del problema, in quanto nel seguito deve essere specificato un protocollo operativo di comunicazione tra l'applicazione (A) ed il sistema. Un oggetto, che rappresenta un'informazione relativa all'applicazione, può essere statico o dinamico. Gli oggetti statici non variano i loro valori per tutta la sessione d'esecuzione, mentre quelli dinamici possono variare i loro valori durante l'esecuzione dell'applicazione.
Ogni applicazione (A) può essere descritta come un insieme di configurazioni operative, ognuna contenente i costi per l'uso delle risorse, i vincoli

e le preferenze. La configurazione operativa rappresenta l'esecuzione dell'applicazione nel rispetto di determinati vincoli di QoS.
Tuttavia questi vincoli risultano trasparenti al RTRM, che vede l'applicazione fornire un insieme di modalità operative.

Nel seguito vengono introdotti in dettaglio questi oggetti, facendo riferimento ad una notazione formale che definisce

$$A := \langle p, \Pi, \delta_A, \mathcal{S}_{cur}, \mathcal{S}_{old}, \mathcal{LB}, \mathcal{UB}, F \rangle$$

una singola applicazione, dove:

- $p$ indica la priorità dell'applicazione all'interno del sistema. Questo valore è un oggetto dinamico, poichè è assegnato all'applicazione al suo avvio, ma può cambiare durante la fase di esecuzione nel sistema. Viene usata nel seguito la notazione $p_A$ per riferirsi alla priorità dell'applicazione A;

- $\Pi := (AWM_1, ..., AWM_M)$ è un oggetto statico che indica l'insieme delle possibili configurazioni ($AWM_i$) che l'applicazione può utilizzare durante l'esecuzione, dove con $M$ si indica la molteplicità di tali configurazioni;

- $\delta_A$ è un oggetto statico che contiene le informazioni relative ai costi di cambio configurazione. Indicando con

  $$\delta^A_{(old,new)} := AWM_{old} \rightarrow AWM_{new}$$

  il passaggio da una configurazione operativa ad un'altra causato da una decisione del sistema, è necessario considerare un costo. Tale costo misura lo sforzo che l'applicazione deve compiere per riorganizzare le proprie strutture dati al fine di operare nella nuova configurazione;

- $S_{cur}$ è un oggetto dinamico, che rappresenta lo stato attuale dell'applicazione. In particolare risulta così definito:

  $$S_{cur} := \langle p_{cur}, AWM_{cur}, k_{cur} \rangle$$

dove è contenuta la priorità d'esecuzione ($p_{cur}$) dell'applicazione, la modalità operativa $AWM_{cur}$ utilizzata ed il cluster $k_{cur}$ sul quale è allocata;

- $S_{old}$ è un oggetto dinamico, che rappresenta lo stato attuale dell'applicazione. In particolare risulta così definito:

$$S_{old} := \langle p_{old}, AWM_{old}, k_{old} \rangle$$

Il significato degli elementi di $S_{old}$ è analogo a quelli presenti in $S_{cur}$, con la differenza che $S_{old}$ rappresenta le informazioni relative all'ottimizzazione precedente. In altre parole all'atto di un processo di ottimizzazione, se l'applicazione è già in esecuzione, i valori di $S_{cur}$ vengono copiati in $S_{old}$ ed in $S_{cur}$ vengono memorizzate le informazioni relative al nuovo stato;

- $\mathcal{LB}$ *e* $\mathcal{UB}$ sono oggetti modellano una caratteristica dinamica dell'applicazione, infatti costituiscono i limiti di *lower bound* $\mathcal{LB}$ ed *upper bound* $\mathcal{UB}$ rispetto alle configurazioni proposte dall'applicazione. In altre parole, l'applicazione può specificare quali tra le configurazioni sono effettivamente richieste per la specifica esecuzione. Supponendo che le configurazioni siano ordinate in modo decrescente secondo il valore di profitto, l'applicazione può specificare se non è interessata ad essere eseguita se la modalità selezionata è inferiore a $\mathcal{LB}$. Viceversa, per evitare un eccessivo consumo di risorse quando non effettivamente necessario, l'applicazione può specificare alcune configurazioni che garantiscono una QoS maggiore di quella effettivamente richiesta. Tali configurazioni sono quelle per cui è superato $\mathcal{UB}$. $\mathcal{LB}$ *e* $\mathcal{UB}$ sono definiti come indici rispetto alle configurazioni disponibili riordinate secondo il profitto generato;

- $F$ è un oggetto statico che contiene le informazioni utili a reperire le funzioni callback. Tali funzioni, opzionali, costituiscono un'ulteriore possibilità di personalizzazione per l'applicazione offerta dal sistema. Il sistema ha un comportamento predefinito nel trattare le

informazioni delle applicazioni, tuttavia alcuni di questi comportamenti possono essere cambiati tramite una funzione di callback da definire secondo un'interfaccia offerta dal sistema.

Sebbene $A$ caratterizzi in maniera totale un'applicazione per il sistema, ogni $AWM$ rappresenta un oggetto complesso che è possibile scomporre in differenti elementi informativi

$$AWM := \langle \nu, \boldsymbol{r_{NCL}}, \boldsymbol{r_{CL}}, \Omega \rangle$$

dove:

- $\nu$ indica la preferenza dell'applicazione ad essere eseguita in quella specifica AWM. Tale preferenza è assegnata a *compile-time* dal progettista dell'applicazione, ma può subire variazioni durante l'esecuzione a causa di processi adattativi. Tali processi costituiscono un modello di come ad esempio le preferenze utente possano modificare le preferenze sulle configurazioni dell'applicazione. Questo è un oggetto dinamico;

- $\boldsymbol{r_{NCL}}$ rappresenta il vettore delle risorse necessarie alla configurazione. Con $\boldsymbol{r}_{NCL}[t]$ con $t \in [1, ..., T_{NCL}]$ viene indicato il numero di risorse di tipologia $i$ necessarie per la configurazione, supposto che il numero di tipologie di risorse, con struttura non a cluster, gestite dal sistema siano $T_{NCL}$. E' un oggetto statico;

- $\boldsymbol{r_{CL}}$ rappresenta il vettore delle risorse necessarie alla configurazione. Con $\boldsymbol{r}_{CL}[t]$ con $t \in [1, ..., T_{NCL}]$ viene indicato il numero di risorse di tipologia $i$ necessarie per la configurazione, supposto che il numero di tipologie di risorse, con struttura a cluster, gestite dal sistema siano $T_{NCL}$. E' un oggetto statico;

- $\Omega$ contiene i dettagli implementativi legati alla specifica configurazione che permettono di classificare l'oggetto come statico. Tali dettagli possono rappresentare il *task graph* dell'applicazione e costituiscono informazioni utili al Resource Manager al momento del mapping tra i core fisici della piattaforma e i thread dell'applicazione.

**Un formalismo per il modello di RTRM**

Il livello di Abstract Run-Time Manager rappresenta il livello deputato alle ottimizzazioni tra le risorse del sistema e le applicazioni in esecuzione. Tramite una serie di algoritmi, questo livello si occupa di trovare un punto di equilibrio che rispetti i vincoli delle applicazioni e i vincoli imposti dalla piattaforma hardware. Tale punto di equilibrio non è altro che uno *stato del sistema* particolarmente desiderabile. Si definisce dunque lo *stato del sistema* o, indifferentemente, *configurazione del sistema* ($\rho$) come

$$\rho := \langle \mathcal{P}, \mathcal{L}_A, \Phi, \boldsymbol{R}_{tot}, \boldsymbol{R}_{free}, \boldsymbol{K}_{tot}, \boldsymbol{K}_{free}, \kappa \rangle$$

dove

- $\mathcal{P}$ rappresenta il profitto generato dal sistema considerando le applicazioni attive nelle rispettive AWM;
- $\mathcal{L}_A$ rappresenta l'insieme delle applicazioni *critiche* e *best-effort* sul quale è stata eseguita l'ottimizzazione dell'allocazione. Questo insieme comprende, senza distinzione, le applicazioni *critiche* e *best-effort* già attive o nuove all'atto dell'ottimizzazione;
- $\Phi$ contiene le informazioni per ogni singolo livello di priorità. Data la complessità di tale oggetto, la descrizione viene dettagliata separatamente;
- $\boldsymbol{R}_{tot}$ è un vettore che contiene le informazioni relative alle risorse totali offerte dal sistema. Tale oggetto memorizza informazioni relative alle risorse che non siano organizzate in cluster;
- $\boldsymbol{R}_{free}$ è un vettore contenente le informazioni relative alle risorse disponibili nel sistema che non presentano struttura a cluster;
- $\boldsymbol{K}_{tot}$ contiene le informazioni relative alle risorse totali offerte dal sistema, organizzate in cluster. La struttura logica può essere assimilata ad una matrice dove ogni colonna rappresenta una tipologia di risorsa organizzata in cluster. L'indice di riga rappresenta, per ogni elemento cluster della risorsa, le risorse totali disponibili;

- $\boldsymbol{K}_{free}$ contiene le informazioni relative alle risorse disponibili nel sistema, organizzate in cluster. L'organizzazione logica è identica a quella dell'oggetto $\boldsymbol{K}_{tot}$, tuttavia per ogni elemento sono contenute le risorse disponibili e non totali;

- $\kappa$ rappresenta un vettore che indica, per ogni risorsa clusterizzata, la numerosità dei cluster di tale risorsa.

Definito inoltre $\boldsymbol{R}_{busy}$ come l'insieme delle risorse utilizzate, si può affermare che per ogni $\rho$ ammissibile devono sempre valere $\forall i \in [1..T_{NCL}]$:

$$\boldsymbol{R}_{free}[i] \leq \boldsymbol{R}_{tot}[i] \tag{3.1}$$

$$\boldsymbol{R}_{busy}[i] \leq \boldsymbol{R}_{tot}[i] \tag{3.2}$$

$$\boldsymbol{R}_{free}[i] + \boldsymbol{R}_{busy}[i] = \boldsymbol{R}_{tot}[i] \tag{3.3}$$

Si ipotizza, ai fini della trattazione, che il sistema gestisca $T_{NCL}$ differenti tipologie di risorsa non clusterizzate e $\boldsymbol{R}_x[i]$, con $i \in [1..T_{NCL}]$ e $x \in \{free, busy, tot\}$, indichi la quantità di una specifica tipologia di risorsa.

La stessa formalizzazione vale per le risorse organizzate in cluster, dove $T_{CL}$ ne indica la numerosità delle differenti tipologie.
Quindi per ogni $\rho$ ammissibile devono valere sempre $\forall i \in [1..T_{CL}]$ *e* $\forall j \in [1..\kappa]$:

$$\boldsymbol{K}_{free}[j,i] \leq \boldsymbol{K}_{tot}[j,i] \tag{3.4}$$

$$\boldsymbol{K}_{busy}[j,i] \leq \boldsymbol{K}_{tot}[j,i] \tag{3.5}$$

$$\boldsymbol{K}_{free}[j,i] + \boldsymbol{K}_{busy}[j,i] = \boldsymbol{K}_{tot}[j,i] \tag{3.6}$$

Ogni elemento $\phi \in \Phi$ è strutturato come

$$\phi_p := \langle p, \Lambda, \lambda, \boldsymbol{R}^p_{busy}, \boldsymbol{K}^p_{busy} \rangle$$

dove:

- $p$ indica il livello di priorità del livello cui fa riferimento. Nel seguito si fa riferimento al concetto di priorità come un numero intero dove a valori decrescenti corrispondono priorità crescenti. La priorità

massima è rappresentata dallo 0 ed è associata al livello contenente le applicazioni *critiche*, mentre le applicazioni *best-effort* hanno assegnate priorità che partono dal valore 1. Al crescere del valore numerico decresce la priorità. Con $\min(P_{be})$ è indicato il livello di più bassa priorità *best-effort* disponibile;

- $\Lambda$ corrisponde al vettore delle applicazioni di priorità $p$ in esecuzione nel sistema, $\Lambda := \langle A_1, ..., A_N \rangle$ dove $N$ rappresenta il numero di applicazioni presenti;

- $\lambda$ contiene le informazioni, relative alla configurazione corrente per il livello, estratte da $\Lambda$. Definiamo dunque $\lambda := \{AWM_{(i,j)} | i \in [1..N], j \in [1..M]\}$, dove ogni applicazione $i$ ha una sola AWM $j$ in esecuzione;

- $\boldsymbol{R}^p_{busy}$ indica le risorse in uso dalle applicazioni del livello $p$ con struttura non a cluster;

- $\boldsymbol{K}^p_{busy}$ indica le risorse, con struttura a cluster, in uso dalle applicazioni del livello $p$.

Ad ogni livello di priorità è dunque associato un elemento $\phi$ che contiene l'insieme minimo di informazioni utili a determinare ogni grandezza d'interesse per l'ottimizzazione.
Dalla definizione dei concetti sopra descritti, si assume che il sistema si trova sempre in uno stato $\rho$, per il quale risultano soddisfatti i vincoli imposti dalle applicazioni e i vincoli di capacità sulle risorse disponibili.
Tale stato può essere turbato in presenza di un *evento* che indicheremo come $\chi$, ossia un'eccezione sollevata dalle risorse o dalle applicazioni, che richiede attenzione da parte del sistema. Vengono distinti due tipi di eventi:

- $\chi_{hw}$ modella un evento scatenato dalle risorse. Rappresenta un cambio nella visione logica delle risorse, definita dal componente di *Resource Account Manager* (vedi sezione 3.3);

- $\chi_{sw}$ modella un evento scatenato dalle applicazioni. Può significare l'arrivo o la terminazione di un'applicazione oppure la richiesta di un cambio di configurazione.

Dopo aver definito quest'astrazione per il modello del sistema è possibile formulare la seguente funzione

$$F : \rho_{old} \xrightarrow{\chi} \rho_{new}$$

che indica come il sistema si muova da uno stato all'altro sotto lo stimolo di eventi.

## 3.3 Modello di un sistema di gestione dinamica delle risorse

L'obiettivo di questa sezione è la proposta di un modello astratto in grado di catturare i differenti aspetti innovativi evidenziati nella sezione 3.1. Il modello mira all'indipendenza dalla piattaforma sottostante, alla flessibilità e ad una formulazione distribuita.
L'indipendenza dalla piattaforma, ove possibile, rende più semplici le operazioni di *porting* verso nuove architetture.
La flessibilità garantisce la possibilità di estendere il modello proposto secondo nuove esigenze, derivanti da nuovi studi.
La formulazione distribuita consente invece di intervenire con politiche locali sui singoli componenti, permettendo di adattare il sistema allo specifico scenario d'uso.

Inoltre sono descritte le ipotesi reali di lavoro utilizzate in questo documento. Tali ipotesi partono dalla descrizione del modello astratto e definiscono concretamente un'area di lavoro. Questa necessità si presenta data la vastità di argomenti che possono emergere dalla presentazione di un modello astratto di gestione dinamica delle risorse. E' dunque utile definire la visione generale per un tale sistema, per poi circoscrivere su alcuni componenti un lavoro di dettaglio.

### 3.3.1 Modello Astratto

La struttura logica del sistema per la gestione dinamica delle risorse risulta divisa in quattro livelli, come evidenziato dalla figura 3.2. Il livello applicativo, il primo in alto nella figura, modella il punto di accesso al sistema rispetto alla totalità delle applicazioni. In altre parole ogni applicazione risulta logicamente separata dal resto del sistema e per accedervi deve interfacciarsi con il sistema di gestione dinamica delle risorse.
Il secondo livello, scendendo la figura, è rappresentato dal gestore delle risorse, ossia il componente deputato all'allocazione delle risorse della piattaforma alle applicazioni che ne fanno richiesta.
Proseguendo l'analisi della figura verso il basso, è rappresentato il livello di sistema operativo, che fornisce un'astrazione alle risorse fisiche disponibili sulla piattaforma. L'ultimo livello rappresenta invece le risorse fisiche offerte della piattaforma.

Tale struttura mostra un certo grado di similitudine con la figura 2.1 (vedi capitolo 2) che descrive la visione attuale, precedente a questo lavoro, del sistema di gestione dinamica delle risorse. In particolare è evidente come il gestore delle risorse, costituisca sempre un livello intermedio tra le risorse offerte dalla piattaforma e le applicazioni che ne fanno richiesta. Inoltre in entrambi i modelli esiste una chiara distinzione tra le funzionalità indipendenti e quelle dipendenti rispetto alla specifica piattaforma (vedi tabella 3.1).

Nel seguito viene fornita una descrizione circa le funzionalità di ogni componente schematizzato nel livello di *Run-Time manager* (vedi figura 3.2). In particolare l'analisi è condotta descrivendo da prima il componente di *Abstract Run-Time Manager* e successivamente il *Platform Specific Run-Time Manager*.
Il componente di *Abstract Run-Time Manager* rappresenta la funzionalità macroscopica di gestione delle risorse, indipendente dalla struttura fisica della piattaforma sottostante e dalla specifica implementazione delle applicazioni. Questo aspetto consente, di concentrarsi solamente sulla funzionalità di ottimizzazione dell'allocazione. Questa funzionalità è implementata supponendo che, a tale livello, siano garantite un'insieme di informazioni astratte circa la piattaforma e le applicazioni. Tali informa-

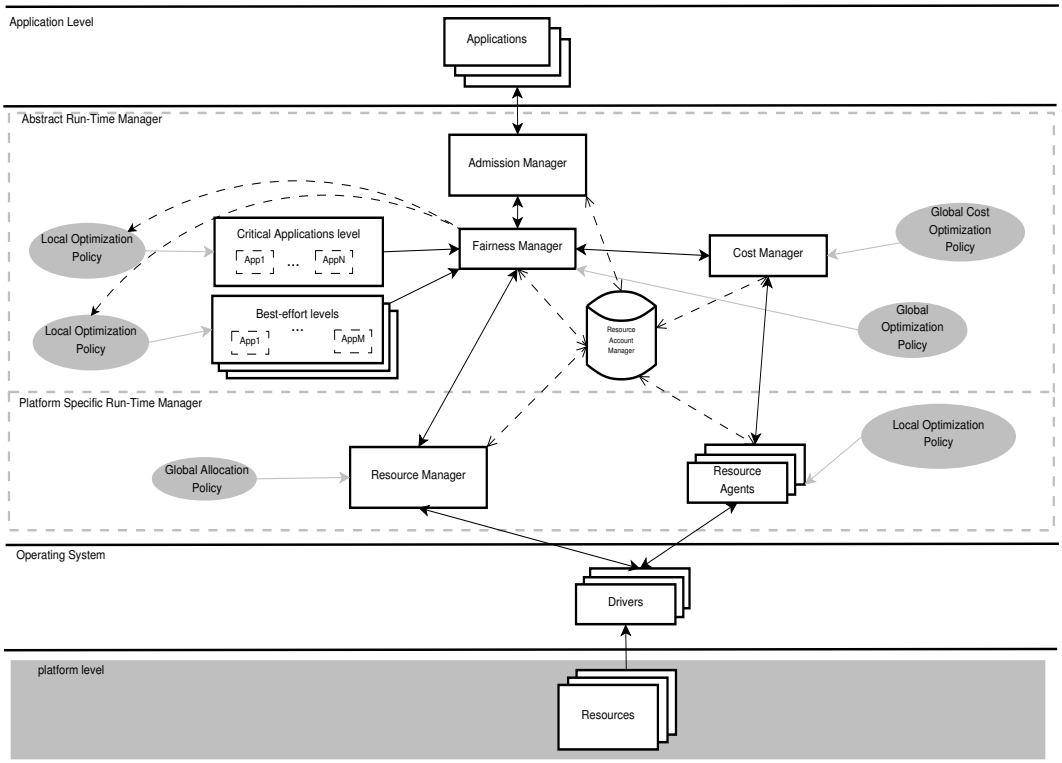



Figura 3.2: Modello astratto di un sistema per la gestione dinamica delle risorse. Viene illustrato lo schema funzionale del sistema inserito nel contesto applicativo, considerando quindi anche i principali attori con i quali deve essere interfacciato.

zioni sono descritte in dettaglio nella sezione 3.2 di questo capitolo. All'interno di questa componente sono definite alcune funzionalità logiche specifiche, che consentono di raggiungere l'obiettivo richiesto:

- *Admission Manager* rappresenta il componente che deve decidere se una nuova applicazione ha diritto di accedere al sistema. Tale decisione è valutata sulla base di informazioni che definiscono lo stato dell'utilizzo delle risorse e le politiche in esecuzione nell'intero livello di *Abstract Run-Time Manager*;

- *Fairness Manager* rappresenta il componente che gestisce l'allocazione ottima delle risorse alle applicazioni che ne fanno richiesta. In

| | **Modello Stato dell'arte** | **Modello proposto** |
|---|---|---|
| **funzionalità astratte dalla piattaforma** | Quality Manager | Abstract Resource Manager |
| **funzionalità dipendenti dalla piattaforma** | Resource Manager | Platform Specific Run-Time Manger |

Tabella 3.1: Mostra i componenti di un generico sistema per la gestione dinamica delle risorse, secondo la formalizzazione proposta nella sezione 2.1 e la nuova proposta presentata in questo documento. Per ogni componente viene evidenziato se la funzionalità che implementa sia astratta o dipendente dall'architettura della piattaforma.

altre parole coordina i livelli *Critical Application Level* e *Best-Effort Application Levels*. La funzione è valutata sulla base di informazioni che definiscono lo stato d'utilizzo delle risorse e le politiche in esecuzione nell'intero livello di *Abstract Run-Time Manager*;

- *Critical Application Level* rappresenta il livello responsabile della gestione delle applicazioni *critiche* in esecuzione nel sistema. Questo componente contiene tutte le informazioni d'interesse per computare l'allocazione ottima delle risorse alle applicazioni *critiche*. L'obiettivo viene raggiunto grazie alle informazioni disponibili al componente ($\phi_0$) e alle informazioni rese disponibili circa lo stato del sistema;

- *Best-Effort Application Levels* rappresenta uno o più livelli deputati alla gestione delle applicazioni di classe *best-effort*, in esecuzione nel sistema. Come il componente di *Critical Application Level*, ogni componente di tipo *Best-Effort Application Levels* contiene le informazioni relative alla sola classe di applicazioni d'interese;

- *Cost Manager* rappresenta un componente che valuta l'impatto sull'uso delle risorse disponibili in termini *economici*. In altre parole utilizzando una visione astratta, rispetto allo stato del sistema $\rho$, quan-

tifica lo sforzo di utilizzare delle risorse sulla base delle politiche in esecuzione;

- *Resource Account Manager* rappresenta il componente che contiene l'insieme di informazioni necessarie ad ogni componente del livello di *Abstract Resource Manager*. In particolare l'elemento si occupa di fornire la visione astratta rispetto alle risorse della piattaforma. La necessità di raccogliere informazioni sulla specifica architettura, per fornire una visione astratta, necessita al componente un meccanismo d'interfaccia verso la piattaforma stessa.
  Nel qual caso sia presente un sistema operativo, tali informazioni possono essere raccolte utilizzando un'interfaccia specifica offerta dal sistema stesso. Ad esempio in ambito *Linux* è disponibile l'interfaccia *cgroups*[1] [25]. Tale interfaccia consente di raggruppare le risorse di sistema definendo una visione astratta della piattaforma. Per quanto riguarda le architetture *NUMA*, sempre in ambito *Linux*, è disponibile una libreria che consente di gestire l'allocazione dei processi utilizzando determinate risorse di sistema [26].

Il *Platform Specific Run-Time Manager* rappresenta il livello di ottimizzazione dell'allocazione dipendente dalla specifica piattaforma. Il suo compito è interfacciare le decisioni astratte dalla piattaforma, valutate dal livello di *Abstract Run-Time Manager*, con l'architettura fisica sottostante. La comunicazione di tale componente, con la piattaforma sottostante, avviene attraverso i driver specifici che controllano i dispositivi fisici. In particolare risulta composto da due elementi:

- *Resource Manager* rappresenta il componente deputato al processo di *mapping* fisico tra i *task* dell'applicazione e le risorse della piattaforma. Tale componente risulta molto complesso, ma nel seguito non viene ulteriormente dettagliato dato l'obiettivo del lavoro in esame (vedi sezione 1.3);

- *Resource Agents* rappresenta l'insieme dei componenti dedicati all'astrazione delle singole tipologie di risorsa presenti nel sistema. Tali

[1] cgroups *http://lxr.free-electrons.com/source/Documentation/cgroups.txt?v=2.6.25*

componenti si occupano di descrivere in modo astratto informazioni d'interesse circa lo stato della specifica risorsa controllata al livello di *Abstract Resource-Manger*.

### 3.3.2 Ipotesi di lavoro

La presentazione nella sezione 3.3 ha messo in luce una nuova struttura logica per un generico sistema di gestione delle risorse. Tale presentazione risulta tuttavia ancora intrattabile, data la vastità delle problematiche che tale modello può introdurre. Questa sezione si occupa di definire alcune ipotesi che delimitano il *modello astratto*, costruendo uno spazio di lavoro opportuno e trattabile.
In particolare:

- ogni applicazione che desidera accedere al sistema, controllato dal RTRM proposto in questo lavoro, è tenuta a fornire un insieme di meta-informazioni al sistema stesso. Tali informazioni sono state formalizzate sopra in questa sezione;

- la tipologia di risorsa organizzata in cluster $t \in [1..T_{CL}]$ è unica. Tale aspetto trasforma alcuni elementi formalizzati precedentemente. Il vettore $\kappa$ diventa uno scalare che indica il numero di cluster dell'unica tipologia di risorsa organizzata a cluster presente nel sistema.
  Gli oggetti $\boldsymbol{K}_*$ con $* \in \{tot, busy, free\}$ rappresentano ognuno un vettore. Ogni elemento dei vettori rappresenta le risorse totali, utilizzate o libere rispetto all'unica tipologia di risorsa strutturata a cluster.
  L'oggetto $r_{i,j}^{CL}[t]$ con $t \in [1..T_{CL}]$, poichè $|[1..T_{CL}]| = 1$, si riduce ad un valore scalare (vedi 3.2).
  Questa ipotesi permette la trattazione di un sistema nel quale siano presenti molteplici tipologie di risorsa, differenti per organizzazione fisica nella piattaforma, preservando il contributo innovativo descritto in 3.3;

- il numero di livelli di priorità è ridotto a due, ossia applicazioni *critiche* e applicazioni *best effort*. Le prime hanno sempre garantito un servizio da parte del sistema, a prescindere dal carico del sistema stesso.
  Le seconde vengono gestite dal sistema all'ottimo delle sue possibilità, compatibilmente con il carico del sistema stesso e le politiche di ottimizzazione da rispettare. Tale restrizione semplifica la trattazione del modello, ma lascia inalterato il contributo innovativo di *mixed workload*;

- il numero massimo di applicazioni contemporaneamente attive nel sistema non è noto a priori. Solamente il numero di applicazioni *critiche* si suppone noto. Inoltre l'esecuzione contemporanea di tutte le applicazioni *critiche* non deve saturare le risorse disponibili.
  Questa ipotesi, rispecchia un vincolo reale per il quale il system integrator, che solitamente fornisce le applicazioni *critiche* per il sistema, deve assicurarsi che queste possano convivere senza problemi sulla piattaforma;

- non esistono risorse allocate a priori per una specifica tipologia di applicazione, ma l'allocazione avviene all'arrivo dell'applicazione stessa;

- la sezione 3.1 descrive la possibilità di modellare un sistema flessibile. Tale considerazione include la problematica di assegnare un costo per le risorse utilizzate. Al fine di concretizzare questo obiettivo, nel seguito sono utilizzate tre grandezze di sistema: *Costo di Migrazione* ($\Psi$), *Costo di Riconfigurazione* ($\Delta$) e *Costo di Accensione Cluster* ($\Theta_k$). Queste grandezze modellano aspetti di grande interesse e di grande impatto sulle prestazioni del sistema [27]. Tuttavia nessuno di questi è accennato in alcun lavoro analizzato relativo alla tematica trattata.

  In particolare, il *Costo di Migrazione* misura lo sforzo che il sistema deve compiere per spostare un applicazione $i$ dal cluster $k1$ al cluster $k2$. In questo lavoro tale grandezza è indipendente dal cluster

di partenza $k_1$, di arrivo $k_2$ e dall'applicazione $i$ da migrare. Un'interessante estensione valutata nel capitolo può considerare il costo ($\Psi_{i,k_1,k_2}$) dipendente da questi fattori.
Si fa presente come, sia il modello proposto nella sezione 3.4, che l'approccio approssimato (vedi 3.5) siano già formulati per integrare tale estensione.

Riconfigurare un'applicazione da una modalità operativa ad un'altra ha un duplice impatto. Da un lato misura lo sforzo o disagio di un'applicazione che deve riorganizzare le proprie strutture dati per adattarsi alla nuova configurazione. Dall'altro lato, misura lo sforzo che il sistema deve compiere per reperire un codice eseguibile per l'applicazione, che ne permetta il funzionamento secondo una modalità differente.
Il *Costo di Riconfigurazione* ($\Delta_{i,AWM_{old},AWM_{new}}$) misura questo secondo aspetto, ossia modella lo sforzo necessario per il sistema per cambiare l'applicazione $i$ da $AWM_{old}$ ad $AWM_{new}$.

Il *Costo di Accensione Cluster* ($\Theta_k$) considera la particolare struttura del sistema. In altre parole, misura lo sforzo del sistema all'attivazione di un cluster ulteriore per allocarvi delle applicazioni.
Nel lavoro proposto tale grandezza risulta indipendente dallo specifico cluster, ma è ipotizzabile che, a cluster differenti, possano corrispondere costi di attivazione differenti.

Oltre a quanto già specificato, solamente per le applicazioni *critiche* valgolo le seguenti ipotesi:

- una volta in esecuzione non possono essere migrate. Questo significa che l'ottimizzatore delle risorse, deve considerarle come applicazioni fisse. Tale ipotesi semplifica in parte il problema dell'allocazione ottima delle risorse, tuttavia sottointende una spiegazione ragionevole.
  Le applicazioni *critiche*, sono considerate il *core business* del dispositivo, ossia gli elementi che forniscono i servizi di base, per i quali la piattaforma è stata progettata.
  A tali servizi di base, possono far riferimento altre applicazioni non

*critiche*. La possibilità di riallocare risorse ad un'applicazione *critica* introduce instabilità nel sistema, poichè il servizio offerto tramite quell'applicazione viene ritardato. Questa ipotesi cessa di valere se sopraggiunge un problema di tipo hardware. Ad esempio, se la risorsa allocata ad un'applicazione *critica* per qualunque ragione dovesse venir spenta, l'applicazione che detiene tali risorse, a prescindere dalla criticità viene migrata;

- per ogni applicazione *critica* è disponibile una singola configurazione operativa. Questa scelta dipende da una considerazione analoga a quella introdotta sopra. L'applicazione *critica* risulta essere altamente ottimizzata per la piattaforma sulla quale opera e non è dunque completamente errato attendersi che sia proposta un'unica configurazione operativa;

- il profitto generato dalle applicazioni *critiche* è sempre $= 0$. In altre parole l'applicazione *critica* costituisce solamente un vincolo per il sistema;

## 3.4 Modello di ottimizzazione

La sezione precedente ha introdotto un modello concettuale astratto che fornisce una visione logica dei meccanismi che il RTRM deve implementare. Tuttavia risulta necessario definire un obiettivo, o concetto di soluzione, da raggiungere in modo formale.
In altre parole è necessario definire un insieme di regole in grado di valutare se una soluzione prodotta dal modello, ossia un nuovo stato $\rho$, è buona o meno per il sistema. Inoltre è necessario definire un altro insieme di regole in grado di individuare una soluzione, ossia il nuovo stato $\rho$, in risposta ad un evento $\chi$.

L'obiettivo di questa sezione è descrivere un modello formale basato sulla programmazione lineare, in grado di formalizzare le regole per individuare e valutare la bontà di una soluzione $\rho$ per il sistema di gestione delle risorse.

La scelta concettuale di utilizzare un approccio matematico basato sulla programmazione lineare è dovuto a due differenti considerazioni:

1. i precedenti lavori nello stesso settore, hanno dimostrato come la programmazione lineare possa descrivere bene un problema di allocazione delle risorse;

2. le proposte migliorative introdotte nella sezione 3.1 possono essere espresse utilizzando questo strumento di modellazione. In altre parole, la programmazione lineare risulta una struttura formale sufficientemente semplice e al contempo espressiva per la trattazione della problematica in esame;

**Il modello formale**

La formulazione di un modello di programmazione lineare necessita di individuare tre tipologie di informazioni:

- un insieme di variabili, che rappresentano i parametri decisionali cui è assegnato un valore durante il processo di ottimizzazione. Al termine del processo di ottimizzazione questi oggetti contengono le informazioni relative alla soluzione individuata;

- un insieme di vincoli, che rappresentano i confini del modello. I vincoli possono essere imposti da scelte progettuali teoriche o da limiti fisici reali;

- una funzione obiettivo in grado di descrivere con un formalismo matematico le proprietà della soluzione desiderata;

**Definizione delle variabili**

Secondo quanto introdotto nella sezione 3.3.2, sono presenti due classi applicative, ossia applicazioni *critiche* e *best-effort*. Vengono dunque introdotti due insiemi di variabili denominati ***variabili di allocazione***:

$$x^{cr}_{n,k} \in \{0,1\} \quad , binarie \qquad \forall n \in [1..N_{cr}], \forall k \in [1..\kappa] \tag{3.7}$$

$$x^{be}_{n,j,k} \in \{0,1\} \quad , binarie \qquad \forall n \in [1..N_{be}], \forall j \in [1..M], \forall k \in [1..\kappa] \tag{3.8}$$

dove gli apici *cr* e *be* identificano la categoria di applicazioni cui fa riferimento la variabile.
Si suppone che siano presenti $N_{be}$ applicazioni *best-effort* e $N_{cr}$ applicazioni *critiche* all'atto dell'ottimizzazione.
La variabile $x^{be}_{n,j,k} = 1$ indica che l'applicazione $i$ di classe *best-effort* è stata configurata sul cluster $k$ nella modalità $j$.
Lo stesso significato è assegnato alla variabile $x^{cr}_{n,k}$ che rappresenta un'applicazione *critica* $i$ configurata sul cluster $k$. Una considerazione da notare, riguarda la mancanza della dimensione $j$ per le variabili $x^{cr}$, in virtù dell'ipotesi che considera le applicazioni *critiche* dotate di una singola modalità operativa (AWM).

La necessità di valutare la migrazione o la riconfigurazione di un'applicazione, richiede l'utilizzo di differenti insiemi di variabili definite ***variabili di migrazione e riconfigurazione***:

$$xMig^{cr}_{n,k1,k2} \in \{0,1\} \ , \quad binarie \quad \forall n \in [1..N_{cr}], \forall k1, k2 \in [1..\kappa] \tag{3.9}$$

$$xMig^{be}_{n,k1,k2} \in \{0,1\} \ , \quad binarie \quad \forall n \in [1..N_{be}], \forall k1, k2 \in [1..\kappa] \tag{3.10}$$

$$xCh^{be}_{n,j1,j2} \in \{0,1\} \ , \quad binarie \quad \forall n \in [1..N_{be}], \forall j1, j2 \in [1..M] \tag{3.11}$$

dove al solito gli apici *cr* e *be* individuano la classe dell'applicazione che la variabile rappresenta, mentre le desinenze *Mig* e *Ch* descrivono rispettivamente una variabile di migrazione o di riconfigurazione per l'applicazione.

La variabile $xMig^{*}_{n,k1,k2} = 1$ con $* \in \{be, cr\}$ indica che l'applicazione $i$, al termine della fase di ottimizzazione, è stata configurata sul cluster $k2$, quando nello stato $\rho_{old}$ si trovava sul cluster $k1$.
La variabile $xCh^{be}_{n,j1,j2} = 1$ indica un'applicazione $i$, che dopo la fase di ottimizzazione, è stata configurata in modalità $j2$, mentre nello stato precedente la sua modalità operativa (AWM) era $j1$. Ovviamente per le applicazioni *critiche* non è prevista una variabile di cambio configurazione, in quanto è presente una sola modalità operativa (singola AWM).

Sebbene non rappresenti una variabile, vengono introdotti in questo contesto i parametri che caratterizzano lo stato precedente per le *applicazioni critiche* e le applicazioni *best-effort* definiti ***parametri di stato*** come:

$$xOld^{cr}_{n,k} \in \{0,1\} \quad , binarie \qquad \forall n \in [1..N_{cr}], \forall k \in [1..\kappa] \tag{3.12}$$

$$xOld^{be}_{n,j,k} \in \{0,1\} \quad , binarie \qquad \forall n \in [1..N_{be}], \forall j \in [1..M], \forall k \in [1..\kappa] \tag{3.13}$$

Questi oggetti non rappresentano delle variabili decisionali, poichè all'atto dell'ottimizzazione, posseggono già dei valori che rimangono fissi. Tali valori possono essere 0 *o* 1 e rappresentano lo stato dell'applicazione $i$ quando il sistema si trova in $\rho_{old}$, cioè lo stato attuale all'inizio di un nuovo processo di ottimizzazione.
Il parametro $xOld^{cr}_{n,k} = 1$ indica che nello stato $\rho_{old}$ l'applicazione $i$ è configurata sul cluster $k$. Lo stesso significato è attribuito al parametro $xOld^{be}_{n,j,k}$, dove però è specificata anche la dimensione $j$, che indica l'AWM di lavoro dell'applicazione *be* nello stato $\rho_{old}$.

Tutte le variabili sopra introdotte e i parametri $xOld^{be}$ e $xOld^{cr}$ hanno valore 0, ad eccezione dei casi specifici descritti per ogni oggetto, per i quali assumono valore 1.

**Definizione dei vincoli**

Anche la definizione dei vincoli del modello richiede un'analisi rispetto alle ipotesi formulate ed al modello astratto (vedi 3.3). La definizione fornita per le applicazioni *critiche* e *best-effort* richiede la formulazione di due insiemi di vincoli definiti ***vincoli di allocazione***:

$$\sum_{m=1}^{M}\sum_{k=1}^{K} x^{be}_{n,m,k} \leq 1, \qquad \forall n \in [1..N_{be}] \tag{3.14}$$

$$\sum_{k=1}^{K} x^{cr}_{n,k} = 1, \qquad \forall n \in [1..N_{cr}] \tag{3.15}$$

Il primo, relativo alle applicazioni *best-effort*, impone che al termine del processo di ottimizzazione, al massimo una configurazione sia selezionata per ogni applicazione. Il secondo vincolo, per applicazioni *critiche*, modella la necessità che tutte le applicazioni *critiche* vengano configurate durante il proceso di ottimizzazione.

Un altro insieme di condizioni necessarie al modello, riguarda i **vincoli di fattibilità**. Tali restrizioni evitano l'individuazione di soluzioni che necessitano di una quantità di risorse superiore a quella disponibile.
La presenza di risorse strutturate in cluster o globali, richiede la specifica di due insiemi di vincoli formalizzati come:

$$\sum_{n=1}^{N_{be}}\sum_{j=1}^{M}\sum_{k=1}^{K} r^{NCL}_{n,j,t} x^{be}_{n,j,k} + \sum_{n=1}^{N_{cr}}\sum_{k=1}^{K} r^{NCL}_{n,j,t} x^{cr}_{n,j,k} \leq R_{tot}[t], \quad \forall t \in [1..T_{NCL}] \tag{3.16}$$

$$\sum_{n=1}^{N_{be}}\sum_{j=1}^{M} r^{CL}_{n,j,t} x^{be}_{n,j,k} + \sum_{n=1}^{N_{cr}} r^{CL}_{n,j,t} x^{cr}_{n,j,k} \leq K_{tot}[k], \quad \forall k \in [1..\kappa] \tag{3.17}$$

La capacità del sistema di tracciare l'accensione di un cluster all'interno di una configurazione $\rho$, richiede un vincolo specifico definito ***vincolo di accensione cluster*** come:

$$\sum_{n=1}^{N_{be}}\sum_{j=1}^{M} x^{be}_{n,j,k} + \sum_{n=1}^{N_{cr}} x^{cr}_{n,k} \leq (N_{cr} + N_{be}) * y_k \quad \forall k \in [1..\kappa] \tag{3.18}$$

dove la presenza di un'applicazione di classe *best-effort* o *critica* sul cluster $k$, impone l'accensione del cluster $k$ stesso.

La necessità di considerare la migrazione e la riconfigurazione per le applicazioni *best-effort* e la sola migrazione per le applicazioni *critiche* richiede la formulazione di altri tre insiemi di vincoli, definiti ***vincoli di migrazione e riconfigurazione***:

$$x^{be}_{n,j1,k1} + xOld^{be}_{n,j2,k2} <= 1 + xCh^{be}_{n,j1,j2} \ , \quad \forall n \in [1..N_{be}] \wedge j1, j2 \in [1..M] \wedge k1, k2 \in [1..K] \wedge j1 \neq j2 \tag{3.19}$$

$$x^{be}_{n,j1,k1} + xOld^{be}_{n,j2,k2} <= 1 + xMig^{be}_{n,k1,k2} \ , \quad \forall n \in [1..N_{be}] \wedge j1, j2 \in [1..M] \wedge k1, k2 \in [1..K] \wedge k1 \neq k2 \tag{3.20}$$

$$x^{cr}_{n,k1} + xOld^{cr}_{n,k2} <= 1 + xMig^{cr}_{n,k1,k2} \ , \quad \forall n \in [1..N_{cr}] \wedge k1, k2 \in [1..K] \wedge k1 \neq k2 \tag{3.21}$$

dove ogni insieme di vincoli cattura l'aspetto di migrazione o riconfigurazione per una specifica classe applicativa. In altre parole, il primo vincolo cattura gli aspetti di riconfigurazione per le applicazioni *critiche*, mentre il secondo ed il terzo insieme di vincoli, catturano gli aspetti di migrazione rispettivamente per applicazioni *best-effort* e *critiche*.

**Definizione della funzione obiettivo**

Definito l'insieme dei vincoli e delle variabili, la funzione obiettivo rimane l'ultimo elemento per completare la formulazione del modello. Tale oggetto rappresenta la guida al processo di ottimizzazione e in senso lato può essere inteso come l'insieme delle proprietà formali che caratterizzano una buona soluzione. La formulazione proposta aggrega tutte le

considerazioni descritte nel capitolo definendo

$$profitto := \sum_{n=1}^{N_{be}} \sum_{m=1}^{M} \sum_{k=1}^{K} v_{n,m} x^{be}_{n,m,k} \tag{3.22}$$

$$- \sum_{k=1}^{K} (\Theta_k * y_k) \tag{3.23}$$

$$- \Psi^{be} * \sum_{n=1}^{N_{be}} \sum_{k1=1}^{K} \sum_{k2=1}^{K} xMig^{be}_{n,k1,k2} \tag{3.24}$$

$$- \Delta^{be} * \sum_{n=1}^{N_{be}} \sum_{m1=1}^{M} \sum_{m2=1}^{M} xCh^{be}_{n,m1,m2} * \delta_{n,m1,m2} \tag{3.25}$$

$$- \Psi^{cr} * \sum_{n=1}^{N_{cr}} \sum_{k1=1}^{K} \sum_{k1=1}^{K} xMig^{cr}_{n,k1,k2} \tag{3.26}$$

Nella pagina seguente è mostrato il modello completo, costruito sulla base dell'insieme di vincoli, di variabili, di parametri e dalla funzione obiettivo definiti in questa sezione.

$$
\begin{aligned}
maximize \sum_{n=1}^{N_{be}} \sum_{m=1}^{M} \sum_{k=1}^{K} & v_{n,m} x^{be}_{n,m,k} \\
& - \sum_{k=1}^{K} (\Theta_k * y_k) \\
& - \Psi^{be} * \sum_{n=1}^{N_{be}} \sum_{k1=1}^{K} \sum_{k2=1}^{K} xMig^{be}_{n,k1,k2} \\
& - \Delta^{be} * \sum_{n=1}^{N_{be}} \sum_{m1=1}^{M} \sum_{m2=1}^{M} xCh^{be}_{n,m1,m2} * \delta_{n,m1,m2} \\
& - \Psi^{cr} * \sum_{n=1}^{N_{cr}} \sum_{k1=1}^{K} \sum_{k1=1}^{K} xMig^{cr}_{n,k1,k2}
\end{aligned}
$$

*subject to*

$$\sum_{k=1}^{K} \Big( \sum_{n=1}^{N_{be}} \sum_{j=1}^{M} r^{NCL}_{n,j,t} x^{be}_{n,j,k} + \sum_{n=1}^{N_{cr}} r^{NCL}_{n,j,t} x^{cr}_{n,j,k} \Big) \leq R_{tot}[t] \qquad \forall t \in [1..T_{NCL}]$$

$$\sum_{n=1}^{N_{be}} \sum_{j=1}^{M} r^{CL}_{n,j,t} x^{be}_{n,j,k} + \sum_{n=1}^{N_{cr}} r^{CL}_{n,j,t} x^{cr}_{n,j,k} \leq K_{tot}[k] \qquad \forall k \in [1..\kappa]$$

$$\sum_{m=1}^{M} \sum_{k=1}^{K} x^{be}_{n,m,k} \leq 1 \qquad \forall n \in [1..N_{be}]$$

$$\sum_{k=1}^{K} x^{cr}_{n,k} = 1 \qquad \forall n \in [1..N_{cr}]$$

$$\sum_{n=1}^{N_{be}} \sum_{j=1}^{M} x^{be}_{n,j,k} + \sum_{n=1}^{N_{cr}} x^{cr}_{n,k} \leq (N_{cr} + N_{be}) * y_k \qquad \forall k \in [1..\kappa]$$

$$x^{be}_{n,j1,k1} + xOld^{be}_{n,j2,k2} \leq 1 + xCh^{be}_{n,j1,j2} \qquad \forall n \in [1..N_{be}] \wedge j1, j2 \in [1..M] \wedge k1, k2 \in [1..K] \wedge j1 \neq j2$$

$$x^{be}_{n,j1,k1} + xOld^{be}_{n,j2,k2} \leq 1 + xMig^{be}_{n,k1,k2} \qquad \forall n \in [1..N_{be}] \wedge j1, j2 \in [1..M] \wedge k1, k2 \in [1..K] \wedge k1 \neq k2$$

$$x^{cr}_{n,k1} + xOld^{cr}_{n,k2} \leq 1 + xMig^{cr}_{n,k1,k2} \qquad \forall n \in [1..N_{cr}] \wedge k1, k2 \in [1..K] \wedge k1 \neq k2$$

$$x^{cr}_{n,k} \in \{0,1\} \qquad \forall n \in [1..N_{cr}], \forall k \in [1..\kappa]$$

$$x^{be}_{n,j,k} \in \{0,1\} \qquad \forall n \in [1..N_{be}], \forall j \in [1..M], \forall k \in [1..\kappa]$$

$$xMig^{cr}_{n,k1,k2} \in \{0,1\} \qquad \forall n \in [1..N_{cr}], \forall k1, k2 \in [1..\kappa]$$

$$xMig^{be}_{n,k1,k2} \in \{0,1\} \qquad \forall n \in [1..N_{be}], \forall k1, k2 \in [1..\kappa]$$

$$xCh^{be}_{n,j1,j2} \in \{0,1\} \qquad \forall n \in [1..N_{be}], \forall j1, j2 \in [1..M]$$

$$y_k \in \{0,1\} \qquad \forall n \in [1..\kappa]$$

## 3.5 Metodo euristico

In questa sezione viene descritto un approccio euristico alla soluzione del problema di allocazione delle risorse presentato nella sezione 3.3.
Il modello ottimo di allocazione delle risorse (vedi sezione 3.3), in quanto problema di programmazione lineare intera, risulta essere NP-difficile da risolvere [17]. Questa proprietà implica un tempo di computazione che, nel caso pessimo, cresce in modo esponenziale rispetto alla dimensione del problema.

Uno degli obiettivi del lavoro svolto, descritti nella sezione 1.3, determina la necessità di produrre una soluzione al caso di studio effettivamente utilizzabile su architetture reali. Ovviamente in un sistema di gestione delle risorse, dove sono presenti anche applicazioni *critiche*, il tempo di computazione per ottenere una soluzione deve essere molto contenuto. In aggiunta, la trattazione della problematica nell'ambito dei sistemi embedded impone attenzione anche all'utilizzo delle risorse per effettuare la computazione.
Anche in questo caso, la quantità di memoria utilizzata dal modello di programmazione lineare intera per calcolare la soluzione, cresce in modo esponenziale rispetto alla dimensione del problema (vedi capitolo 4).

```
1 ρ HeuYamca(ρ ρ_old, List A newApp_be, List A newApp_cr)
2 {
3   ρ ρ_new;
4   AllocateCritical(ρ_new, ρ_old, newApp_cr);
5   AllocateBestEffort(ρ_new, ρ_old, newApp_be);
6   return ρ_new;
7 }
```

Listing 3.1: Schema generale dell'euristica

Una delle principali sfide durante la progettazione di un algoritmo approssimato, riguarda l'assenza del modello da approssimare, anche se risulta nota la struttura del problema. E' necessario dunque individuare un metodo alternativo per specificare all'euristica ciò che nel modello viene specificato tramite i vincoli. E' inoltre necessario formulare una cifra

di merito, ad esempio $\tau$, definita nella sezione 2.3, in grado di guidare le scelte dell'algoritmo approssimato.

L'approccio approssimato che viene utilizzato in questo lavoro, si basa sul concetto di algoritmo *greedy*, già introdotto nella sezione 2.3.
La necessità di gestire due differenti classi applicative, *critiche* e *best-effort*, ha determinato la formulazione di una procedura che agisce per passi allocando da prima le applicazioni *critiche* e successivamente quelle *best-effort*.
In tal modo si cerca di garantire che le applicazioni *critiche* vengano allocate all'interno di un sistema libero. In altre parole si cerca di offrire alle applicazioni *critiche* la più alta possibilità di essere allocate.
D'altra parte le applicazioni *best-effort*, in virtù delle molteplici modalità operative disponibili e della minor priorità, si adattano ad essere configurate in un sistema dove già sono allocate le applicazioni *critiche*.

Nel seguito la presentazione è articolata in due parti. La prima presenta in dettaglio lo schema di allocazione per le applicazioni *critiche*. Nella seconda parte, è mostrata in dettaglio la filosofia di allocazione delle applicazioni *best-effort*.

### 3.5.1 Allocazione delle applicazioni *critiche*

Lo scopo dell'algoritmo incaricato dell'allocazione delle applicazioni *critiche* è di allocare tutte le applicazioni. Inoltre, l'allocazione prodotta non deve ostacolare l'algoritmo di allocazione delle applicazioni *best-effort*.
Le applicazioni *critiche* definite in questo lavoro, presentano tre particolarità che devono essere considerate nella stesura dell'algoritmo:

- il profitto generato da un'applicazione *critica* è sempre $= 0$;
- è prevista una singola AWM;
- il sistema di gestione delle risorse deve sempre trovare un'allocazione ad ogni applicazione *critica*.

La loro allocazione è obbligatoria e, sebbene non introducano un incremento di profitto, utilizzano risorse di sistema. Definita un'unica AWM

per le applicazioni *critiche*, non esiste alcun modo di scegliere quante risorse occupare. Tuttavia un'attenzione particolare deve essere dedicata nella scelta del cluster sul quale allocare ogni applicazione *critica*. Infatti sebbene le applicazioni *critiche* siano sempre le prime ad essere allocate, ad ogni istanza del processo di ottimizzazione, devono essere considerati due fattori:

1. allocare un'applicazione *critica* sul cluster $k$, può implicare una migrazione delle applicazioni *best-effort* allocate sullo stesso cluster nello stato precedente $\rho_{old}$, con conseguente costo di migrazione. Potrebbe essere preferibile, ad esempio, allocare l'applicazione *critica* su un cluster non attivo e pagarne il costo di accensione;

2. l'allocazione delle applicazioni *critiche* può risultare ottima, ma produrre frammentazione interna nei cluster. Tale frammentazione al momento dell'allocazione delle applicazioni *best-effort* non è detto possa essere eliminata. Questo problema emerge dalla struttura dell'algoritmo, che considera prima l'allocazione delle applicazioni *critiche* e successivamente, in modo separato, l'allocazione di quelle *best-effort*.

E' dunque corretto affermare che un'allocazione scorretta delle applicazioni *critiche*, può provocare l'individuazione di una configurazione subottima nella successiva allocazione delle applicazione *best-effort*.

```
1  ρ AllocateCritical(ρ ρ_new, ρ ρ_old, List A newApp_cr)
2  {
3    Copy(ρ_new.Φ_cr.Λ, rho_{old}.Φ_cr.Λ);
4    foreach a in newApp_cr
5    {
6      k=EvaluateBestCluster(a,ρ_old,ρ_new);
7      Allocate(ρ_new,a,k);
8    }
9    return ρ_new;
10 }
```

Listing 3.2: Schema di allocazione delle applicazioni critiche

L'algoritmo incaricato di scegliere il miglior cluster dove allocare l'applicazione *critica* è schematizzato nel listato 3.3. L'ipotesi è preferire sempre un cluster già attivo rispetto ad un cluster spento per allocarvi l'applicazione *critica*.

Tale schema procedurale risulta molto semplicistico rispetto al problema e numerosi miglioramenti possono essere apportati.

```
1  #define −2000 INFEASIBLE
2  #define −1000 SWITCH–OFF
3
4  int EvaluateBestCluster(A a,ρ ρ_old,ρ ρ_new)
5  {
6    int evalCl[ρ_new.κ]
7    int indexCl= −1;
8
9    foreach cl in ρ_new.κ
10   {
11     if(ρ_new.K_free[cl]<a.r_CL) || (∃i ∈ [1..T_NCL] | ρ_new.R_free[i] < a.r_NCL[i])
12     {
13       evalCl[cl]=INFEASIBLE;
14       continue;
15     }
16     if(∄a ∈ ρ_new.L_A|a.S_old.k == cl)
17     {
18       evalCl[cl]=SWITCH–OFF;
19       continue;
20     }
21     evalCl[cl]=ρ_new.K_tot[cl]−(ρ_new.K_free[cl]+a.r_CL)−ρ_old.Φ_be.K_busy[cl];
22     if(evalCl[cl] > evalCl[indexCl])
23       indexCl=cl;
24   }
25   return indexCl;
26 }
```

Listing 3.3: Procedura per la selezione del miglior cluster dove allocare l'applicazione critica

**I problemi dell'approccio esposto**

A prescindere dalla caratteristica approssimata dell'algoritmo, l'approccio utilizzato introduce un possibile problema di carattere teorico. Il vincolo di allocare sempre un'applicazione *critica* può non essere rispettato utilizzando l'approccio descritto.
Questo problema sorge dalla struttura a cluster di una delle risorse considerate, sebbene sia presente un'ipotesi, che garantisce la disponibilità di sufficienti risorse per l'esecuzione contemporanea di tutte le applicazioni *critiche*. Tale problema è indotto dalla frammentazione interna che può venire a generarsi all'interno dei singoli cluster.

Si ipotizzi, a titolo di esempio, un sistema a singola risorsa, strutturata in due cluster $K_1$, $K_2$ composti di quattro elementi ciascuno. Si supponga inoltre che siano presenti tre applicazioni *critiche* A, B, C con rispettivamente un consumo di risorse 3, 2, 2. Se all'arrivo dell'applicazione A nel sistema sono già presenti le applicazioni B, C allocate rispettivamente sul cluster $K_1$ *e* $K_2$ come si comporta il sistema? L'unica possibilità è migrare una tra le applicazioni B, C ed allocare A sul cluster lasciato libero. Tuttavia un'altra delle ipotesi formulate in questo lavoro, non prevede la migrazione per le applicazioni *critiche* già allocate.
In realtà, anche garantendo la possibilità di migrazione alle applicazioni *critiche* già allocate, l'individuazione di una soluzione, che allochi tutte le applicazioni *critiche* presenti nel sistema, non è assicurata [1]. Questo problema sorge dalla natura euristica dell'algoritmo di allocazione.
In aggiunta a questo, l'unico lavoro che considera in parte questa problematica [1] prevede che, maggiore è il carico del sistema, rispetto alle applicazioni *critiche* attive, maggiori sono le probabilità che, l'ipotesi di garantire sempre un'allocazione per le applicazioni *critiche* venga violata.

Nel seguito sono delineate due possibilità di intervento, al fine di contenere questa problematica:

1. aggiungere nuove ipotesi alla formulazione del problema. Una ragionevole ipotesi è garantire che la somma delle risorse utilizzate dalle applicazioni *critiche* non ecceda le risorse disponibili del si-

stema e che ogni applicazione *critica* non ecceda l'uso delle risorse disponibili sul singolo cluster.
Inoltre, garantendo che la somma delle risorse di tipo cluster utilizzate dalle applicazioni critiche sia inferiore al 50% delle risorse disponibili nel sistema, si avrebbe garantito il rispetto dell'ipotesi di allocazione per ogni applicazione *critica*.
Questa soluzione interviene solamente sulla formalizzazione del problema e nella realtà impone che le applicazioni *critiche* per il dispositivo non possano occupare più della metà delle risorse disponibili nel sistema;

2. determinare una soluzione di allocazione per le applicazioni *critiche* a *compile-time*. Poichè un algoritmo approssimato non può garantire con assoluta certezza che tutte le applicazioni *critiche* siano sempre allocate, l'unica soluzione è una computazione a *compile-time* della soluzione ottima di allocazione. Le informazioni necessarie per questo approccio riguardano la completa conoscenza di tutte le risorse utilizzate per ogni applicazione *critica* che può accedere al sistema. In tal modo l'aggiunta o l'eliminazione di un'applicazione *critica*, dalla lista delle applicazioni che possono accedere al sistema, determina il calcolo di una nuova soluzione ottima.
Il vantaggio di questo approccio riguarda la certezza di un'allocazione per ogni applicazione *critica*. Tuttavia tale soluzione è individuata in isolamento rispetto alle applicazioni *best-effort*. Questo implica che a *run-time* l'insieme delle applicazioni best-effort attive insieme alle applicazioni *critiche*, potrebbe portare a risultati di allocazione migliori, se viene considerata la specifica situazione e non la preallocazione ottima.

### 3.5.2 Allocazione delle applicazioni *best-effort*

La seconda parte dell'algoritmo euristico proposto, si occupa di individuare la miglior allocazione per le applicazioni *best-effort*, utilizzando sempre un approccio di tipo *greedy*.
L'obiettivo finale è rendere massimo il profitto generato dalle applicazioni, facendo attenzione ai costi generati dalla riconfigurazione e migrazione delle applicazioni stesse e ai costi di accensione dei cluster (vedi modello teorico sezione 3.4).

```
1  struct MinConf_sys
2  {
3     int minConf_sys^NCL [NCL];
4     int minConf_sys^CL [CL];
5  };
6
7  ρ AllocateBestEffort(ρ ρ_new, ρ ρ_old, List A newApp_be)
8  {
9     struct MinConf_sys minConf_sys;
10    minConf_sys=EvalMinConf(old.φ_be.Λ, newApp_be);
11    foreach k in ρ_new.κ
12    {
13       Compute_τ(k, minConf_sys, old.φ_be.Λ, newApp_be);
14       L_k=Order(k, newApp_be, ρ_old.Φ_be.Λ);
15       PickUp(L_k);
16    }
17    return ρ_new;
18 }
```

Listing 3.4: Schema di base per l'allocazione delle applicazioni *best-effort* utilizzando minConf

A differenza dell'algoritmo di allocazione per le applicazioni *critiche*, questo algoritmo risulta molto più complesso, poichè deve considerare diversi fattori tra cui: la molteplicità delle AWM per ogni applicazione, la possibilità di riconfigurare o migrare un'applicazione, la discrezione nell'accensione dei cluster, la possibilità di non configurare tutte le applicazioni e ovviamente l'aspetto strutturale della risorsa clusterizzata.

L'algoritmo conosce inoltre l'allocazione delle applicazioni *critiche*, avvenuta al passo precedente e delle rispettive risorse utilizzate.

L'idea base dell'algoritmo è visitare ogni cluster e allocarvi le applicazioni che garantiscono maggior profitto in quella posizione (vedi listato 3.4).
La visita per cluster permette di considerare implicitamente la struttura particolare del sistema, che costituisce una delle complicazioni al problema presentato nella sezione 2.2.

Al fine di scegliere quali applicazioni allocare in quale cluster, è necessario definire una *cifra di merito* in grado di guidare il processo decisionale. Tale cifra di merito è costruita prendendo in considerazione quanto proposto nella sezione 2.2 e l'obiettivo del modello presentato nella sezione 3.4.

Viene definita per ogni applicazione $a \in [1..N_{be}]$, per ogni AWM $m \in [1..M]$ e per ogni cluster $k \in [1..\kappa]$ una grandezza

$$\tau^{heu}_{(a,m,k)} := \frac{a[m].v - MigrationCost - Reconfiguration Cost}{\frac{\boldsymbol{Conf}^{CL}_{sys} * r^{CL}_{i,j}}{K_{tot}[k]} + \sum_{t=1}^{T_{NCL}} \frac{\boldsymbol{Conf}^{NCL}_{sys}[t] * r^{NCL}_{i,j}[t]}{R_{free}[t]}} \tag{3.27}$$

dove

- $MigrationCost := \Psi * isMig(k, a.\mathcal{S}_{old}.k)$;
- $Reconfiguration Cost :=$

$$\Delta * a[m].isChAwm(m, a.(S)_{old}.AWM) * a.\delta_{(a.(S)_{old}.AWM,m)}$$

- $isMig(k_1, k_2)$ è una funzione che indica, sulla base degli argomenti che rappresentano gli identificativi di due cluster, se vi è migrazione;
- $isChAwm(m_1, m_2)$ è una funzione che indica, sulla base degli argomenti che rappresentano gli identificativi di due cluster, se vi è riconfigurazione;

- $Conf(t)_{sys}$ è un elemento complesso definito come:

$$Conf(t)_{sys} := \langle \boldsymbol{Conf}_{sys}^{CL}, \boldsymbol{Conf}_{sys}^{NCL} \rangle$$

che identifica l'insieme delle risorse minime richieste da tutte le applicazioni attive nel sistema. Poichè sono presenti molteplici tipologie di risorsa, è quasi impossibile identificare per ogni applicazione $a$, la configurazione $m$, tale che:

$$a.m.r_{NCL}[t] < a.m_i.r_{NCL}[t] \quad \forall t \in [1..T_{NCL}] \wedge \forall i \in [1..M]$$

Nel seguito sono proposte due soluzioni per la valorizzazione di $\boldsymbol{Conf}_{sys}^{CL}$ e $\boldsymbol{Conf}_{sys}^{NCL}$.

La prima, ***minConf***, suppone che per ogni applicazione la configurazione a minimo utilizzo di risorse sia quella che fornisce il più basso valore di profittoe risulta definita come segue:

$$\boldsymbol{minConf}_{sys}^{CL} := \sum_{i=1}^{N_{be}} r_{i,minConf(i)}^{CL} + \sum_{i=1}^{N_{cr}} r_i^{CL}$$

$$\boldsymbol{minConf}_{sys}^{NCL} := \sum_{i=1}^{N_{be}} \sum_{t=1}^{T_{NCL}} \boldsymbol{r}_{i,minConf(i),t}^{NCL} + \sum_{i=1}^{N_{cr}} \sum_{t=1}^{T_{NCL}} \boldsymbol{r}_{i,t}^{NCL}$$

Inoltre minConf(i) è definito come:

$$minConf(i) := argmin_{j \in [1..M]}(v_{(i,j)}), \quad \forall i \in [1..N_{be}]$$

La seconda, ***avgConf***, costruisce un oggetto che per ogni tipologia di risorsa valuta la media delle risorse utilizzate da ogni applicazione in ogni configurzione:

$$\boldsymbol{avgConf}_{sys}^{CL} := \frac{\sum_{i=1}^{N_{be}} \sum_{m=1}^{M} r_{i,m}^{CL}}{N_{be} * M} + \frac{\sum_{i=1}^{N_{cr}} r_i^{CL}}{N_{cr}}$$

$$\boldsymbol{avgConf}_{sys}^{NCL} := \frac{\sum_{i=1}^{N_{be}} \sum_{m=1}^{M} \sum_{t=1}^{T_{NCL}} \boldsymbol{r}_{i,m,t}^{NCL}}{N_{be} * M} + \frac{\sum_{i=1}^{N_{cr}} \sum_{t=1}^{T_{NCL}} \boldsymbol{r}_{i,t}^{NCL}}{N_{cr}}$$

La cifra di merito $\tau^{heu}_{a,m,k}$ viene utilizzata per costruire un ordinamento globale delle configurazioni $m$ di ogni applicazione $a$ rispetto al cluster visitato $k$. Nel seguito con $\mathcal{L}_k$ si fa riferimento a tale lista relativa al cluster $k$.

```
1  ρ PickUp(List A L_k ,ρ ρ_new)
2  {
3    bool flagRes, flagOpt, flagPickedUp
4    List A tempList;
5    foreach a in L_k
6    {
7      flagRes=controlResAvailability(ρ_new ,a);
8      flagPickedUp=controlNotAlreadyPickedUp(ρ_new ,a);
9
10     if((flagRes==true)&&(flagPickedUp==true))
11       tempList=preCommit(ρ_new ,a,tempList);
12   }
13   ρ_new=commit(ρ_new ,tempList);
14   return ρ_new;
15 }
```

Listing 3.5: Procedura per la selezione delle configurazioni migliori per ogni applicazione sulla base di una lista ordinata. Tale lista risulta ordinata rispetto ad uno specifico cluster k.

Il listato 3.4 mostra una schematizzazione dell'algoritmo descritto. In particolare la funzione $Compute_{\tau}(minConf_{sys})$ alla linea 13, si occupa di valutare la *cifra di merito* $\tau$, mentre *Order(k,newApp*$_{be}$*,*$\rho_{old}$*.*$\Phi_{be}$*.*$\Lambda$*)* si occupa di generare una lista ordinata sulla base della cifra di merito $\tau$. La funzione *PickUp(*$\mathcal{L}_k$*)*, di cui è fornito uno schema nel listato 3.5, effettua la scelta effettiva degli elementi più profittevoli.

**Complessità temporale** Utilizzando la notazione introdotta nella sezione 3.2 è possibile definire la complessità temporale dell'euristica nel caso pessimo come:

$$O(\underbrace{\kappa * (N_{cr})}_{critiche} + \underbrace{\kappa * (\underbrace{N_{be} * M}_{Compute\_\tau} + \underbrace{(N_{be} * M) * log(N_{be} * M)}_{\text{Order}} + \underbrace{N_{be} * M)}_{\text{PickUp}})}_{best\text{-}effort})$$

dove sono evidenziati i contributi di complessità introdotti dalle funzioni descritte nei listati 3.2 e 3.4.

**Complessità spaziale** La cifra di complessità spaziale viene definita come:

$$O(\underbrace{N_{cr} * A}_{\text{app } critiche} + \underbrace{N_{be} * M * A}_{\text{app } best\text{-}effort})$$

dove sono evidenziati i contributi derivanti dalle due classi di applicazioni considerate.
L'algoritmo implementato utilizza un insieme di liste concatenate per la gestione dei dati. Dunque per ogni nuova applicazione $i$ e per ogni sua AWM, viene aggiunto un elemento di tipo $A$ nella lista opportuna. Tale elemento è strutturato secondo quanto descritto nella sezione 3.2.

## 3.6 Conclusioni

In questo capitolo sono stati sfruttati i punti di forza presenti nelle proposte di lavoro precedenti (vedi capitolo 2), al fine di produrre un sistema in grado di trattare alcuni aspetti non ancora investigati.
In particolare sono stati presentati:

1. *un modello funzionale astratto* relativo al componente di ottimizzazione per un sistema di gestione dinamica delle risorse. Attingendo da lavori svolti in precedenza è stato possibile delineare un sistema che fa della flessibilità e distribuzione dei compiti un suo punto di forza;

2. *un modello formale di ottimizzazione delle risorse*. Tale modello cattura aspetti innovativi ancora non trattati dai lavori di settore, disponibili al momento della stesura del documento. In particolare si è cercato di costruire un modello che sia in grado di descrivere meglio, rispetto allo stato attuale, le problematiche pratiche legate alla gestione dinamica delle risorse;

3. *un algoritmo di ottimizzazione* che approssima la soluzione individuata dal modello formale di ottimizzazione. Questo elemento mira a soddisfare uno degli obiettivi proposti nella sezione 1.3, ossia garantire l'implementazione di un sistema utilizzabile praticamente.

Capitolo 4

# Valutazione sperimentale

*" Misura ciò che è misurabile, e rendi misurabile ciò che non lo è. "*

Galileo Galilei

L'obiettivo del processo di valutazione sperimentale è misurare la bontà e l'efficacia dei metodi e delle tecniche proposte a livello teorico. Per quanto proposto nel capitolo 3, ossia il modello formale e l'euristica, è stata dunque fornita un'implementazione utilizzando il linguaggio di programmazione C.
Inoltre la formulazione algoritmica del modello formale, basato sulla programmazione lineare, ha richiesto l'utilizzo di *GLPK*[1] ed in particolare del solutore di programmazione lineare *GLPSOL*, basato sul linguaggio *MathProg* [28].

In questo capitolo sono descritti gli esperimenti svolti ed è discusso il metodo per la generazione degli scenari sperimentali.
Lo scopo primario di questa fase è rivolto alla valutazione dell'efficienza algoritmica, intesa come complessità spaziale e temporale e della bontà della soluzione individuata, intesa come profitto generato dal RTRM.

[1] sito di riferimento del progetto GLPK: *http://www.gnu.org/software/glpk/*

Inoltre, sono stati condotti alcuni esperimenti in un contesto il più possibile reale, il cui scopo è stato quello di valutare l'impatto degli elementi innovativi inseriti nel modello.
Per la simulazione degli scenari descritti in seguito, è stato necessario predisporre un nuovo sistema di generazione di casi di test, in grado di validare gli elementi innovativi proposti in questo lavoro.

La struttura del capitolo è divisa in due parti.
Nella prima parte viene descritto il simulatore implementato. Nelle seconda parte sono descritte le prove sperimentali effettuate.

## 4.1 Sistema per la generazione degli scenari di test

L'implementazione di quanto proposto in un sistema reale, risulta una prima possibilità per la valutazione sperimentale di un RTRM. Questo significa implementare un sistema di gestione delle risorse per una specifica architettura, in grado di comunicare con il sistema operativo, con l'hardware sottostante e con le applicazioni in esecuzione.

Tuttavia l'infrastruttura necessaria per supportare questo genere di sperimentazione risulta fuori dagli scopi di questo lavoro, data la sua eccessiva complessità. Inoltre la fase di valutazione delle soluzioni teoriche proposte, deve poter avvenire il prima possibile, rispetto all'implementazione del sistema finale. In tal modo è possibile correggere eventuali anomalie e facilitare la progettazione della struttura di sistema finale.

A causa delle difficoltà nella predisposizione di un sistema completo, i test vengono eseguiti secondo l'approccio utilizzato da numerosi lavori analizzati, ossia attraverso un insieme di benchmark che modellano scenari ipotetici di lavoro.

I problemi che emergono nell'utilizzo di una simile strategia di test sono tre:

1. *individuazione dei dati di test.* E' necessario utilizzare dei dati che siano rappresentativi per il problema, al fine di ottenere misure

| *AppScen*[*i*].*dsc* | $N_A$ | TNCL | TCL | M | Avg | Var |
|---|---|---|---|---|---|---|
| *AppScen*1.*dsc* | 5 | 4 | 1 | 5 | 22.4 – 34.6 | 59.84 – 245.84 |
| *AppScen*1.*dsc* | 10 | 4 | 1 | 5 | 22.4 – 34.6 | 17.6 – 245.8 |
| *AppScen*3.*dsc* | 15 | 9 | 1 | 10 | 75.3 – 101.2 | 248.1 – 954.1 |
| *AppScen*4.*dsc* | 20 | 9 | 1 | 10 | 134.4 – 182.6 | 352.24 – 2028.84 |
| *AppScen*5.*dsc* | 25 | 9 | 1 | 10 | 40.03 – 80.71 | 890.43 – 4473.9 |
| *AppScen*6.*dsc* | 30 | 9 | 1 | 10 | 40.0 – 80.7 | 890.4 – 4473.9 |

Tabella 4.1: Insieme degli scenari di test considerati. Tali dati rappresentano un sottoinsieme dei dati reperibili in [29], riorganizzati secondo la struttura richiesta dal simulatore implementato. Per ogni scenario viene specificato il numero di applicazioni contenute, la numerosità delle risorse di tipo clusterizzato (*TCL*) e comuni a tutti i cluster (*TNCL*) ed il numero di configurazioni che ogni applicazione specifica (*M*). Per ogni applicazione presente nello scenario viene valutata la media e varianza, rispetto ad i valori di profitto. La colonna AVG riportata il valore medio del profitto più basso e più alto. La colonna Var riporta la varianza del profitto minimo e massimo.

d'interesse reale. Specificatamente a questo aspetto, molti dei lavori analizzati, utilizzano un insieme di scenari di test disponibili in [29]. Tali scenari, *Scen*[*i*].*dsc*, dove *i* identifica il singolo scenario, descrivono ognuno un'insieme di applicazioni secondo una struttura simile a quella introdotta nella sezione 3.2. Inoltre sono descritte le risorse disponibili nel sistema per ogni tipologia.
Tuttavia, a causa degli interventi migliorativi introdotti in questo lavoro, è stato necessario predisporre un ulteriore insieme di dati che meglio specifica il sistema in esame. E' stato quindi necessario ristrutturare l'insieme dei dati di test definendo:

- un nuovo elemento, *Platform.dsc*, che raccoglie tutti i dati relativi alla piattaforma utilizzata, sia quelli introdotti in questo lavoro, che quelli presenti negli scenari di test [29] (vedi tabella 4.2);

| Platform.dsc |
|---|
| Costo accensione cluster |
| Costo migrazione |
| Costo riconfigurazione |
| Numero di Cluster |

Tabella 4.2: Informazioni di piattaforma da specificare per ogni test effettuato. Tali valori rappresentano quantità numeriche astratte rispetto ad una specifica architettura.

| ExtraInfo[i].dsc | | | | |
|---|---|---|---|---|
| Application | {Priority} | {AWM} | $t_{enter}$ | $t_{running}$ |

Tabella 4.3: Dettaglio relativo alla struttura di ExtraInfo[i].dsc.

- un elemento *AppScen*[*i*].*dsc* per ogni scenario *i*, che contiene solamente le informazioni relative alle applicazioni.
  La struttura di tali informazioni risulta logicamente organizzata secondo la formalizzazione proposta nella sezione 3.2.

2. *predisporre scenari complessi e simili alla realtà.* I miglioramenti al modello di RTRM, introdotti durante il lavoro di tesi, rendono tale sistema più flessibile ed in grado di gestire scenari di test più articolati rispetto a quanto proposto fino ad ora. Per ogni lavoro precedente, i dati di test proposti in [29], vengono utilizzati per generare una singola allocazione delle risorse, rispetto a tutte le applicazioni dello scenario.
   Ciò significa che ogni scenario viene utilizzato solamente una volta. Tuttavia un RTRM deve agire in un ambiente altamente dinamico, dove le applicazioni possono entrare ed uscire dal sistema. Viene dunque utilizzato un altro insieme di informazioni che descrivono, per ogni scenario *AppScen*[*i*].*dsc*:

   - l'istante di arrivo e terminazione di ogni applicazione;
   - la classe di ogni applicazione, ossia *critica* o *best-effort*.

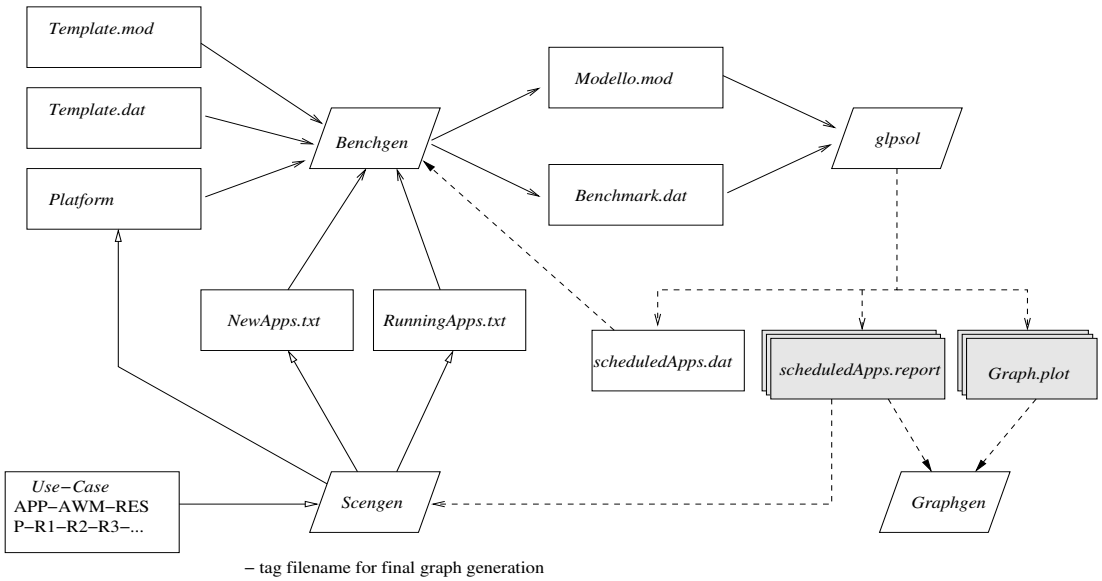



Figura 4.1: Schema di dettaglio relativo al flusso di esecuzione per il modello ottimo.

Tali informazioni sono definite come $ExtraInfo[i].dsc$ (vedi tabella 4.3);

3. *riproducibilità degli scenari di test.* Un'ultima considerazione importante riguarda la possibilità di riprodurre i test condotti. Per far fronte a tale necessità, sono state utilizzate le informazioni descritte nel paragrafo precedente al fine di realizzare un simulatore completo per il RTRM progettato (vedi figura 4.1 e figura 4.1). Tale simulatore, implementato utilizzando il linguaggio di scripting Bash, costruisce uno scenario di test (ScenTest.dsc), partendo dalle informazioni descritte ai punti precedenti. Lo scopo del simulatore introdotto, è dunque quello di simulare l'insieme delle applicazioni in arrivo al sistema. La formulazione flessibile di tale componente, permette di simulare eventi differenti, quali ad esempio l'arrivo e la terminazione di applicazioni o la presenza di applicazioni *critiche* e *best-effort* contemporanea.

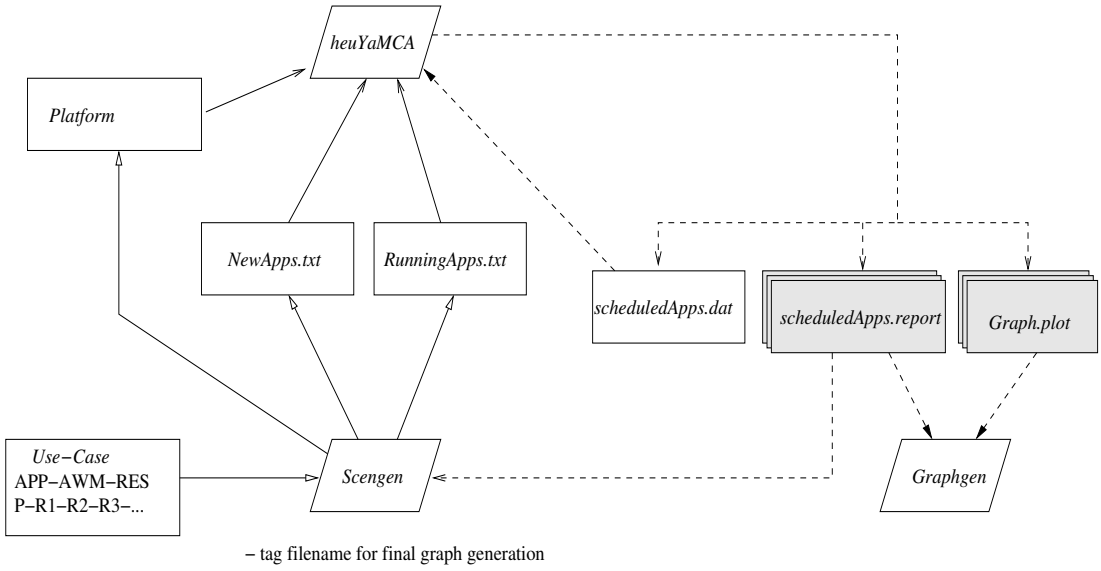



Figura 4.2: Schema di dettaglio relativo al flusso di esecuzione per l'algoritmo approssimato.

## 4.2 Misure sperimentali

L'obiettivo di questa sezione è mostrare i risultati maturati considerando la specifica natura del problema, ossia l'intrinseca dinamicità dell'ambiente di lavoro.

L'insieme dei risultati proposti in tutti i lavori precedenti, non sfrutta appieno l'espressività del modello e dell'euristica proposti. Lo scenario che richiede l'allocazione di decine di applicazioni contemporaneamente, permette di valutare la rapidità di computazione dell'algoritmo, la quantità di memoria utilizzata e la differenza tra il profitto prodotto dal modello ottimo e dall'algoritmo euristico.
Tuttavia questo genere di test non è in grado di raccogliere aspetti di notevole interesse pratico, in un sistema dove le applicazioni arrivano e terminano in istanti differenti.

E' necessario dunque effettuare simulazioni per scenari, dove tutte le applicazioni non risultano avviate contemporaneamente. Questo approccio permette di valutare l'effettivo impatto della migrazione e riconfigurazione delle applicazioni stesse. Come già discusso nella sezione 3.3, la fase di ottimizzazione dell'allocazione delle risorse, rappresenta solamente una delle fasi in carico al RTRM. Infatti una volta determinata la configurazione di sistema astratta ($\rho$), il componente di *Resource Manager* (vedi 3.3) deve effettuare l'effettiva allocazione delle risorse fisiche alle applicazioni. Sicuramente un alto tasso di migrazioni o riconfigurazioni impatta notevolmente su quest'ultima fase [27].

Per riuscire a valutare alcuni aspetti innovativi, si suppone, se non altrimenti specificato che:

- vengono definiti degli istanti logici ($\chi$), per la modellazione del tempo durante l'esecuzione dello scenario. Ogni istante rappresenta il verificarsi di un evento nel sistema. Nello specifico si considera che per ogni scenario considerato ogni evento rappresenti l'arrivo di una nuova applicazione. Viene dunque simulato un sistema nel quale le applicazioni arrivano dinamicamente;

- la piattaforma sia descritta dall'insieme di valori specificati nella tabella 4.4 e dall'insieme di risorse specifiche dello scenario considerato. I valori di costo utilizzati, non rappresentano delle quantità ottenute sperimentalmente. Viceversa rappresentano invece una visione astratta, in linea con i dati di test considerati. Queste quantità modellano un sistema dove si ipotizza che l'operazione più costosa sia l'accensione di un nuovo cluster. Inoltre è supposto che la migrazione e la riconfigurazione di un'applicazione presentino la stessa penalità;

- l'insieme di test proposti confronta valori ottenuti utlizzando tre differenti algoritmi:

    1. *YaMCA* (Yet Another Multi-Cluster Allocator) rappresenta l'implementazione della soluzione euristica (vedi sezione 3.5). Tale

| Platform.dsc | |
|---|---|
| Costo accensione cluster: | 20 |
| Costo migrazione: | 10 |
| Costo riconfigurazione: | 10 |
| Numero di Cluster: | 4 |

Tabella 4.4: Informazioni di piattaforma di base valide per ogni scenario di test considerato se non altrimenti specificato.

algoritmo viene utilizzato ad ogni istante logico aggiungendo alle applicazioni già presenti nel sistema la nuova applicazione in arrivo;

2. *SLPI* (Step-by-Step Integer Linear Programming) che rappresenta l'implementazione del modello basato sulla programmazione lineare intera. In particolare questa versione viene utilizzata come *YaMCA*, simulando di aggiungere, ad ogni istante temporale, un'applicazione al sistema;

3. *OLPI* (One-Shot Integer Linear Programming) rappresenta l'implementazione del modello basato sulla programmazione lineare intera. A differenza di *SLPI*, ad ogni istante logico, *OLPI* produce un'allocazione, che valuta tutte le applicazioni già presenti nel sistema, più la nuova arrivata, considerandole come tutte appena giunte nel sistema. Tale modalità di ottimizzazione si basa su quanto proposto da lavori precedenti (vedi ad esempio [1]), che considerano un'allocazione in blocco delle applicazioni. Ovviamente questo metodo di allocazione, non consente di valutare l'impatto di migrazione e riconfigurazione, poichè considera le applicazioni come appena arrivate nel sistema.

La scelta di utilizzare l'approccio di allocazione incrementale tramite *YaMCA* e *SLPI*, risulta giustificato considerando la visione dinamica del sistema. L'approccio utilizzato dall'algoritmo *OLPI* vuole mostrare come l'allocazione possa variare, rispetto ai risultati pro-

dotto da *YaMCA* e *SLPI*, poichè il sistema, in questo caso, ha a disposizione subito tutti i dati relativi a tutte le applicazioni da allocare. Tuttavia in uno scenario reale questo risulta impossibile.

Definite le ipotesi sotto cui sono condotti i test, vengono introdotte le grandezze considerate d'interesse per tali test. In particolare si valutano le differenze, ad ogni istante $\chi$, tra la soluzione prodotta dagli algoritmi *YaMCA* e *SLPI*, rispetto alle seguenti quantità:

- scostamento percentuale di profitto definito come:

$$\mathcal{P}_{err}^{slpi-yamca} := \frac{\rho_{slpi}.\mathcal{P} - \rho_{YaMCA}.\mathcal{P}}{\rho_{slpi}.\mathcal{P}}$$

  dove $\rho_{slpi}.\mathcal{P}$ e $\rho_{yamca}.\mathcal{P}$ indicano rispettivamente i profitti individuati dal modello ottimo e dall'algoritmo euristico;

- scostamento percentuale del numero di migrazioni tra le due soluzioni ad ogni istante, definito come:

$$Mig_{err}^{slpi-yamca} := \#Mig_{slpi} - \#Mig_{yamca}$$

  dove $\#Mig_{slpi}$ e $\#Mig_{yamca}$ indicano rispettivamente il numero di migrazioni effettuate dal modello ottimo e dall'algoritmo euristico;

- scostamento percentuale del numero di riconfigurazioni tra le due soluzioni ad ogni istante, definito come:

$$ChAwm_{err}^{slpi-yamca} := \#ChAwm_{slpi} - \#ChAwm_{yamca}$$

  dove $\#ChAwm_{slpi}$ e $\#ChAwm_{YaMca}$ indicano rispettivamente il numero di riconfigurazioni effettuate dal modello ottimo e dall'algoritmo euristico;

Le stesse quantità sono definite rispetto alle soluzioni valutate utilizzando *YaMCA* e *OLPI* come:

- scostamento percentuale di profitto definito come:

$$\mathcal{P}_{err}^{olpi-yamca} := \frac{\rho_{olpi}.\mathcal{P} - \rho_{YaMCA}.\mathcal{P}}{\rho_{olpi}.\mathcal{P}}$$

  dove $\rho_{olpi}.\mathcal{P}$ e $\rho_{yamca}.\mathcal{P}$ indicano rispettivamente i profitti individuati dal modello ottimo e dall'algoritmo euristico;

- scostamento percentuale del numero di migrazioni tra le due soluzioni ad ogni istante, definito come:

$$Mig_{err}^{olpi-yamca} := \#Mig_{olpi} - \#Mig_{yamca}$$

  dove $\#Mig_{olpi}$ e $\#Mig_{yamca}$ indicano rispettivamente il numero di migrazioni effettuate dal modello ottimo e dall'algoritmo euristico;

- scostamento percentuale del numero di riconfigurazioni tra le due soluzioni ad ogni istante, definito come:

$$ChAwm_{err}^{olpi-yamca} := \#ChAwm_{olpi} - \#ChAwm_{yamca}$$

  dove $\#ChAwm_{olpi}$ e $\#ChAwm_{yamca}$ indicano rispettivamente il numero di riconfigurazioni effettuate dal modello ottimo e dall'algoritmo euristico;

### 4.2.1 Misure prestazionali: tempo di computazione e memoria utilizzata

I risultati presentati in questa sezione mirano a valutare la reale necessità di un approccio euristico per la soluzione del modello basato sulla programmazione lineare intera (vedi sezione 3.4).
In particolare vengono mostrati i valori di occupazione di memoria e tempo impiegato, per l'individuazione della soluzione utilizzando gli algoritmi *YaMCA OLPI* e *SLPI*.

Le simulazioni utilizzano i dati degli scenari $1-6$ (vedi tabella 4.1), rigenerati tramite l'ambiente descritto nella sezione 4.1.

Le simulazioni sono effettuate limitando il tempo di computazione dell'algoritmo basato sulla programmazione lineare (*OLPI* e *SLPI*) a 120 secondi.
Il sistema di riferimento utilizzato per tali simulazioni ha un processore Intel Core Duo2, con una frequenza di clock fissata a 1.4 GHz. Le implementazioni proposte di *YaMCA OLPI* e *SLPI* fanno uso di un singolo processore essendo a singolo thread. In particolare per gli algoritmi *OLPI* e *SLPI* tale restrizione è dovuta alla versione a singolo thread disponibile per lo strumento *GLPSOL*.

Dai grafici relativi al consumo di memoria (vedi figura 4.3), si può notare come, anche per un numero ristretto di applicazioni, l'algoritmo basato sulla programmazione lineare tenda a consumare una gran quantità di memoria diversamente dall'euristica.

Considerando i grafici che riportano il tempo di computazione necessario alla valutazione della soluzione, si nota come l'euristica riesca sempre a mantenere un valore assoluto inferiore alla decina di millisecondi (vedi figura 4.4). Viceversa l'algoritmo basato sul modello di programmazione lineare, sebbene individui la soluzione ottima, utilizza un tempo per la computazione inaccettabile per un'applicazione in un sistema reale. Infatti in alcuni casi viene superata al decina di secondi.

Una nota importante, riguarda l'impossibilità di reperire un tempo per alcuni scenari per quanto riguarda l'algoritmo basato sulla programmazione lineare. Tale restrizione è rappresentata dal formato di output dello strumento *GLPSOL*, che limita la misura di tempi con approssimazione al decimo di secondo.
Di conseguenza per alcune iterazioni degli algoritmi *SLPI* e *OLPI*, in particolare quelle relative a casi molto semplici, di rapida risoluzione, viene erroneamente riportato nel grafico un tempo nullo. Viceversa l'algoritmo *YaMCA* fornisce misure temporali più accurate, ma nel grafico sono state arrotondate al decimo di millisecondo. Tali risultati confermano quanto ipotizzato teoricamente. Risulta quindi giustificata la scelta di studiare un approccio euristico per la valutazione dell'allocazione ottima di risorse.

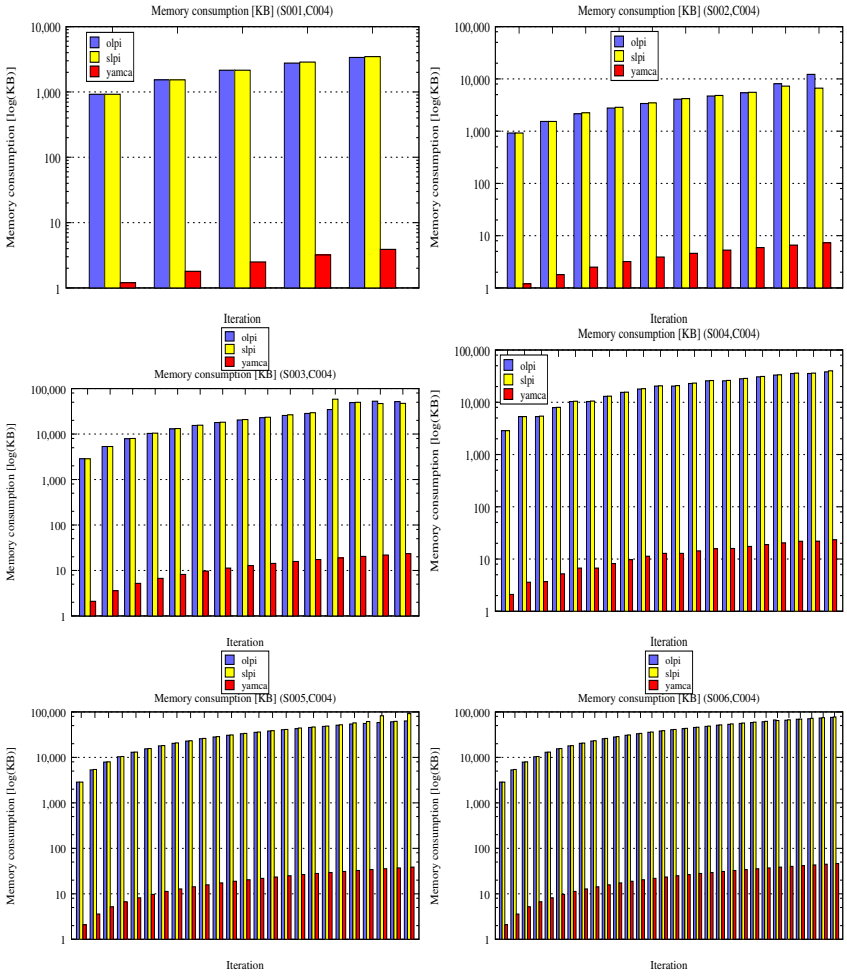



Figura 4.3: Grafici relativi al consumo di memoria per l'individuazione della soluzione da parte degli algoritmi proposti, specificatamente agli scenari di test $1 - 6$

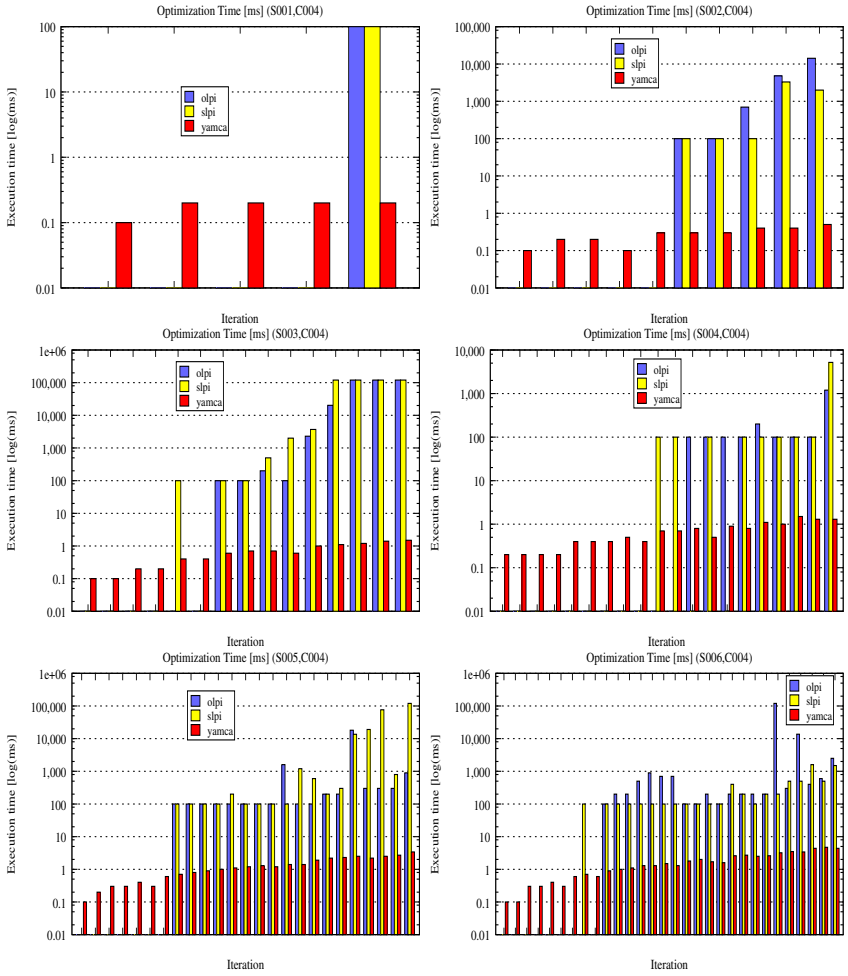



Figura 4.4: Grafici relativi al tempo di computazione per l'individuazione di una soluzione da parte degli algoritmi proposti, specificatamente agli scenari di test $1 - 6$

### 4.2.2 Valutazione del profitto

L'obiettivo di questa sezione è mostrare i risultati maturati considerando la specifica natura del problema, ossia l'intrinseca dinamicità dell'ambiente di lavoro
Vengono utilizzati gli scenari di test riportati in tabella 4.1 per costruire i grafici che mostrano l'andamento delle grandezze definite nella sezione 4.2.

Lo scopo è valutare la bontà dell'algoritmo euristico *YaMCA* nel catturare la dinamica del sistema. Inoltre si vuole valutare se alcune grandezze, nello specifico la migrazione e la riconfigurazione, influenzano l'andamento del profitto ottenuto.

In particolare viene mostrato il comportamento dell'algoritmo *YaMCA* utilizzando i due differenti metodi per la valutazione del minimo utilizzo di risorse descritti nella sezione 3.5.
L'andamento del profitto ottenuto da *YaMCA*, utilizzando *minConf* e *avgConf*, ha sostanzialmente la stessa forma per gli scenari 1 e 2 (confronto figure 4.5a, 4.5d e 4.7a, 4.7a).
Tuttavia la differente natura nel metodo utilizzato per l'aggregazione delle risorse viene evidenziata con scenari con un numero maggiore di applicazioni. Con riferimento all'inizio della simulazioni 3, 4, 5, 6, l'andamento di *YaMCA*, utilizzando *minConf*, evidenzia un valore di $\mathcal{P}_{err}^{slpi-yamca}$, sempre inferiore rispetto all'implementazione *YaMCA*, che utilizza *avgConf*. La differenza è dovuta ad una differente forma di penalizzazione delle configurazioni. In particolare con *minConf* sceglie come configurazioni minime per ogni applicazione quelle a profitto minimo. Tale scelta risulta interessante, se l'applicazione esibisce una correlazione tra basso valore di profitto e basso utilizzo di risorse. Infatti nei primi istanti di simulazione quando il sistema risulta poco carico, non sono penalizzate le configurazioni ad alto profitto che utilizzano molte risorse. Al contrario *avgConf* costruisce una media rispetto alle risorse utilizzate da ogni configurazione penalizzando maggiormente l'utilizzo di risorse nei primi istanti di simulazione. Il comportamento risulta risulta invertito nelle fasi finali della simulazione. Il valore di $\mathcal{P}_{err}^{slpi-yamca}$ tende a crescere utiliz-

zando *minConf*, mentre con l'uso di *avgConf* risulta più stabile e di valore inferiore.

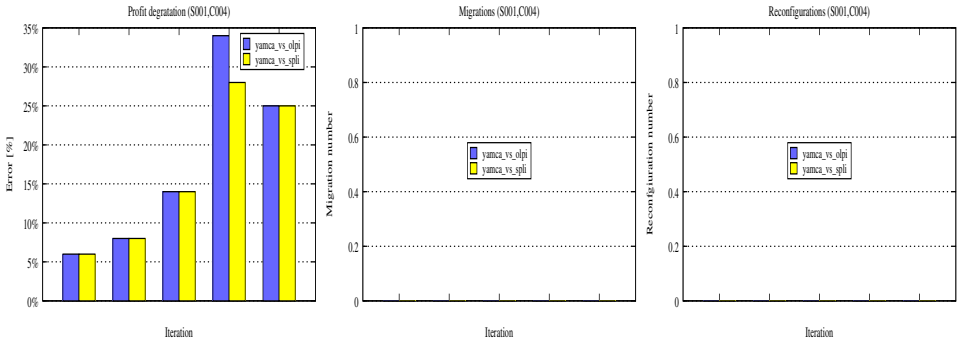
Una seconda considerazione interessante, rigurada l'impatto della riconfigurazione e migrazione delle applicazioni da parte del RTRM. Rispetto a questi fattori, l'utilizzo di *minConf* oppure avgConf, il comportamento di *YaMCA* risulta uguale. In particolare, il degrado del valore di profitto si evidenzia quando *YaMCA* effettua più operazioni di migrazione o riconfigurazione rispetto a *SLPI*. Ad esempio nello scenario 3 relativo all'uso di *minConf*, si nota, come agli istanti 9 e 10 corrisponda un valore di $\mathcal{P}_{err}^{slpi-yamca}$ tra il 25 ed il 30% (vedi figura 4.5g).
Tale scostamento è giustificato dall'alto numero di riconfigurazioni compiute dall'euristica *YaMCA* rispetto all'algoritmo SLPI. Lo stesso comportamento nello scenario 3, si evidenzia anche utilizzando *minConf* agli istanti 12 e 13, dove sono alti valori di migrazioni e riconfigurazioni a determinare un degrado del valore di profitto (vedi figura 4.7g).

Un'ultima considerazione interessante riguarda i grafici relativi allo scenario 2, dove, alla seconda iterazione, l'algoritmo *YaMCA* si comporta meglio dell'algoritmo *SLPI* (vedi figura 4.5d)

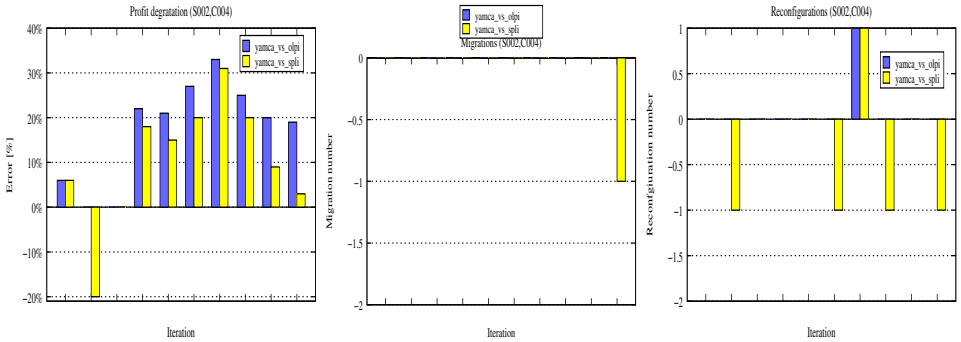
Tuttavia, questa apparente contraddizione non nasconde un errore di modellazione, ma bensì riassume una caratteristica di completa impredicibilità della distribuzione di arrivo delle applicazioni.
In altre parole, sia l'algoritmo *SLPI* che l'algoritmo euristico computano la soluzione, cercando di ottimizzare rispetto alle informazioni a disposizione. Tali informazioni riguardano lo stato del sistema $\rho$ e le applicazioni nuove da allocare, ma non esiste alcun dato che permetta una previsione dell'arrivo di applicazioni in un istante futuro.
Un errore di allocazione nell'istante precedente di *YaMCA*, può determinare una situazione simile a quella descritta all'iterazione successiva.
Tale comportamento risulta evidente dai grafici di allocazione relativi all'algoritmo *YaMCA* ed *SLPI* relativi all'istante 2 (vedi figure 4.5d e 4.7d).
Questo eventualità risulta impossibile tra *YaMCA* ed *OLPI*, per due ragioni:

1. l'algoritmo *OLPI* non presenta mai fenomeni di riconfigurazione e migrazione, poichè considera un arrivo contemporaneo di tutte le applicazioni;

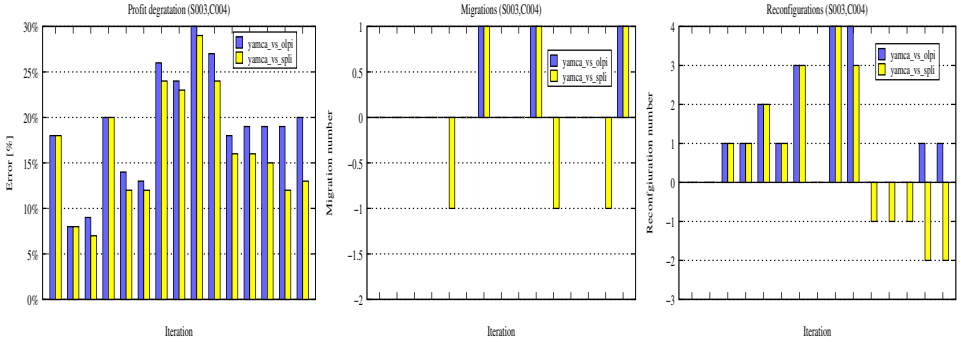
2. l'arrivo contemporaneo di tutte le applicazioni, fornisce a *OLPI* un vantaggio sul modo di organizzare l'allocazione ottima, poichè possiede tutte le informazioni rispetto a cosa e come deve essere allocato. Viceversa *YaMCA* ed *OLPI* allocano le nuove applicazioni basandosi anche sullo stato precedente ($\rho_{old}$);




(a) Profitto Scen 1 (b) Migrazioni Scen 1 (c) Reconfigurazioni Scen 1




(d) Profitto Scen 2 (e) Migrazioni Scen 2 (f) Reconfigurazioni Scen 2




(g) Profitto Scen 3 (h) Migrazioni Scen 3 (i) Reconfigurazioni Scen 3

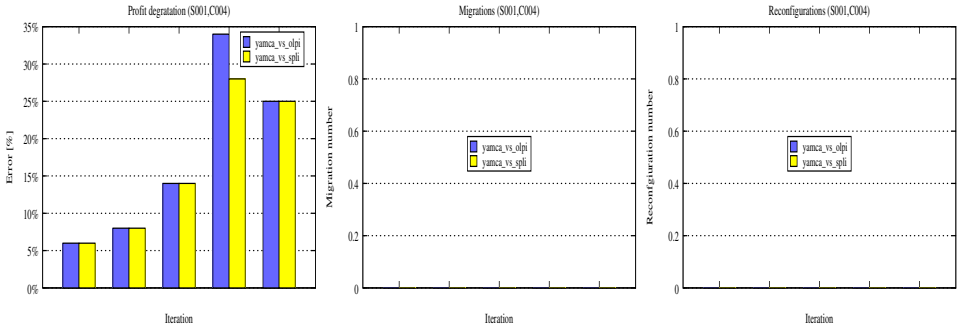
Figura 4.5: Grafici relativi agli scenari $1 - 3$, utilizzando *minConf*.


(a) Profitto Scen 4 (b) Migrazioni Scen 4 (c) Reconfigurazioni Scen 4

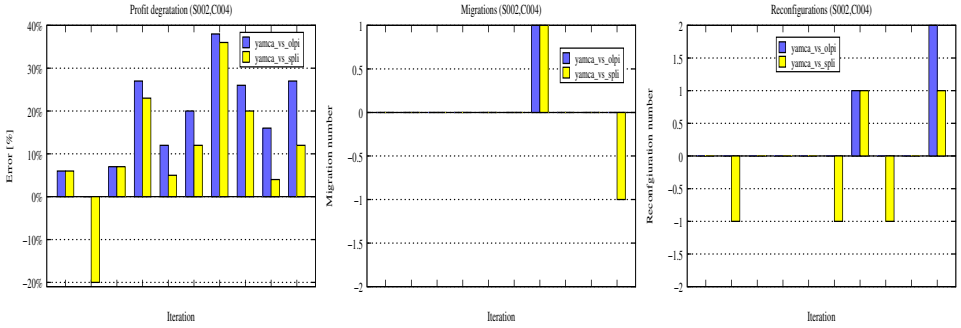
(d) Profitto Scen 5 (e) Migrazioni Scen 5 (f) Reconfigurazioni Scen 5

(g) Profitto Scen 6 (h) Migrazioni Scen 6 (i) Reconfigurazioni Scen 6

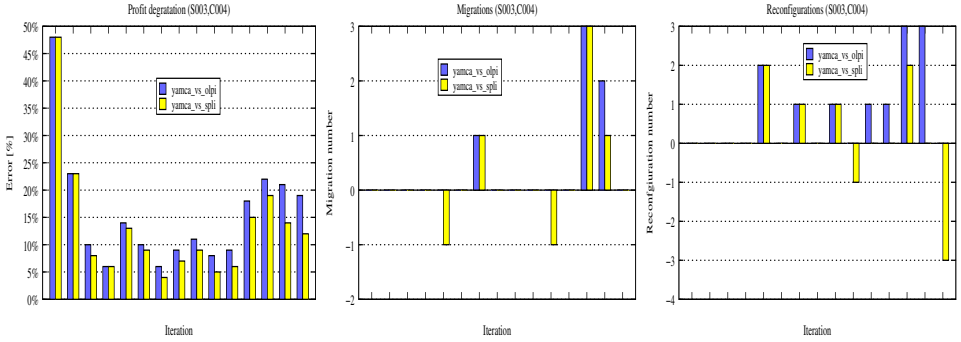
Figura 4.6: Grafici relativi agli scenari 4 – 6, utilizzando *minConf*




(a) Profitto Scen 1 (b) Migrazioni Scen 1 (c) Reconfigurazioni Scen 1

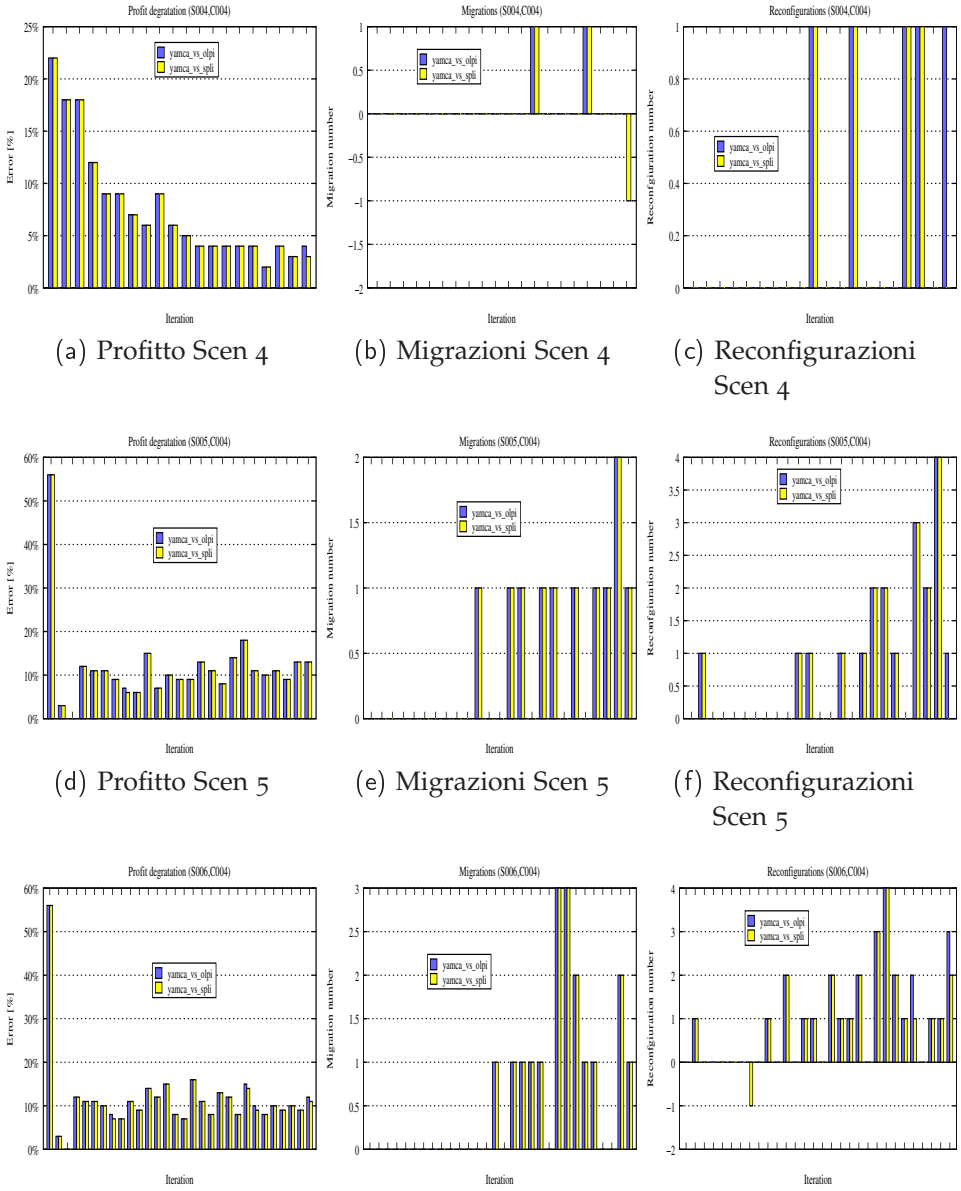



(d) Profitto Scen 2 (e) Migrazioni Scen 2 (f) Reconfigurazioni Scen 2




(g) Profitto Scen 3 (h) Migrazioni Scen 3 (i) Reconfigurazioni Scen 3

Figura 4.7: Grafici relativi agli scenari $1 - 3$, utilizzando *avgConf*.




(a) Profitto Scen 4 (b) Migrazioni Scen 4 (c) Reconfigurazioni Scen 4

(d) Profitto Scen 5 (e) Migrazioni Scen 5 (f) Reconfigurazioni Scen 5

(g) Profitto Scen 6 (h) Migrazioni Scen 6 (i) Reconfigurazioni Scen 6

Figura 4.8: Grafici relativi agli scenari 4 – 6, utilizzando *avgConf*.

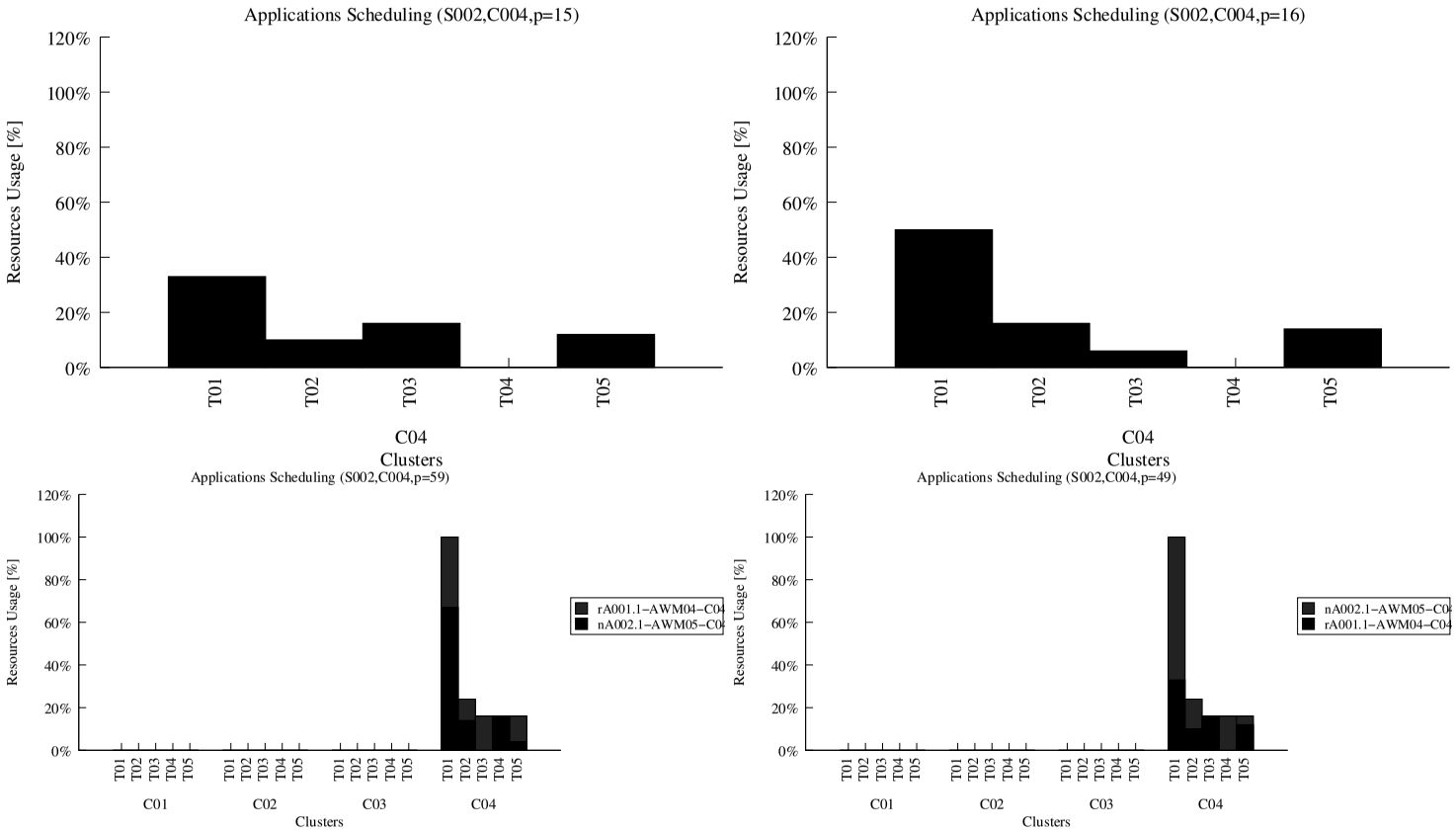



Figura 4.9: Evoluzione di allocazione dell'algoritmo euristico e dell'algoritmo basato sulla programmazione lineare. Lo scenario considerato è il 3, agli istanti 1 e 2. Tale evoluzione mostra come per un errore di allocazione all'istante 1, l'algoritmo *YaMCA* riesca ad ottenere un miglior profitto all'istante 2 rispetto a *SLPI*. In particolare all'istante 1 *YaMCA* alloca A001 in AWM04, mentre *SLPI* alloca A001 in AWM05 ottenengo un profitto maggiore. Tale valore di profitto è riportato nella parte alta di ogni grafico, in relazione alla specifica allocazione. Per il primo istante è riportato solo il cluster acceso, mentre l'istante 2 mostra tutti i cluster ed in particolare si vede come ancora solo il cluster 4 sia acceso. Per ogni cluster la prima barra verticale indica il consumo di PE del cluster stesso. Tale consumo è rapportato alla disponibilità del singolo cluster. Le successive barre verticali mostrano l'utilizzo percentuale delle risorse di tipologia $T_{NCL}$, ossia comuni ad ogni cluster.

### 4.2.3 Un caso reale

In questa sezione sono utilizzati i dati contenuti nello scenario 4 (vedi tabella 4.1), per la simulazione di un caso reale di allocazione.
Tale scenario risulta dimensionato per 20 applicazioni, ma la simulazione, che prevede ingresso ed uscita delle applicazioni stesse, non consente di avere mai più di 10 applicazioni attive contemporaneamente. Al fine di rendere quanto più possibile reale la simulazione, viene ridotto ad un terzo l'insieme delle risorse del sistema previsto per questo scenario.

Tale simulazione prevede due ipotesi:

- carico misto, ossia applicazioni *critiche* e *best-effort* in esecuzione nello stesso sistema;
- per ogni applicazione viene specificato un istante di arrivo ed un tempo di permanenza nel sistema;

In particolare l'insieme di informazioni *ExtraInfo4.dsc* è descritto dalla tabella 4.5

Per ogni istante logico $\chi$, vengono valutati i valori di profitto, migrazione e riconfigurazione prodotti dagli algoritmi *YaMCA*, *OLPI* e *SLPI*.
Inoltre vengono descritte le allocazioni delle risorse prodotte ad ogni istante $\chi$ dagli algoritmi *YaMCA*, *SLPI* e *OLPI*.

L'analisi dei risultati si concetra sulla presenza delle applicazioni *critiche*, che rispetto ai precedenti test costituiscono una novità. In particolare dai grafici emerge come l'algoritmo *YaMCA* riesca in generale a mantenere un buon valore di profitto anche all'ingresso delle applicazioni critiche. Infatti all'istante 6, con l'arrivo dell'applicazione critica A011, l'euristica riesce ad ottenere un valore di profitto addirittura migliore rispetto a *SLPI*.
Tuttavia, un limite dell'algoritmo YaMCA risulta evidente all'istante 8, con l'arrivo dell'applicazione critica A014 (vedi seconda riga di grafici in figura 4.13). Lo schema delineato per l'allocazione delle applicazioni *critiche* prevede che queste siano allocate sul miglior cluster già attivo. Per il caso in esame, tale implementazione determina un alto valore

| Application | Priority | AWM | $t_{enter}$ | $t_{running}$ |
|---|---|---|---|---|
| A001 | 1 | * | 001 | 1 |
| A002 | 1 | * | 001 | 2 |
| A003 | 0 | AWM1 | 001 | * |
| A004 | 1 | * | 001 | 2 |
| A005 | 1 | * | 002 | 5 |
| A006 | 0 | AWM1 | 002 | 5 |
| A007 | 1 | * | 003 | 2 |
| A008 | 1 | * | 004 | 1 |
| A009 | 1 | * | 004 | 1 |
| A010 | 1 | * | 005 | 5 |
| A011 | 0 | AWM1 | 006 | 2 |
| A012 | 1 | * | 007 | * |
| A013 | 1 | * | 008 | 2 |
| A014 | 0 | AWM1 | 008 | 2 |
| A015 | 1 | * | 008 | 2 |
| A016 | 1 | * | 009 | 4 |
| A017 | 1 | * | 009 | 2 |
| A018 | 1 | * | 010 | 1 |
| A019 | 0 | AWM1 | 010 | 1 |
| A020 | 1 | * | 011 | 5 |

Tabella 4.5: Dettaglio relativo alla struttura di ExtraInfo[i].dsc.

di $\mathcal{P}_{err}^{slpi-yamca}$, proprio a causa dell'inserimento di A014 in un cluster, il quarto, già carico. Viceversa la soluzione SLPI tende a distribuire meglio le applicazioni ottenendo come vantaggio un più alto profitto ed una maggior quantità di applicazioni attive.
Ovviamente la discrepanza tra i risultati ottenuti da *YaMCA* e da *SLPI*, mette in luce come la natura del problema combinatorio lasci spazio a differenti concetti di soluzione.
Nel test discusso in questa sezione, risulta evidente come, l'implementazione proposta di *YaMCA*, tenda ad utilizzare una politica di risparmio sulle risorse, cercando comunque di mantenere alto il profitto globale ot-

tenuto.

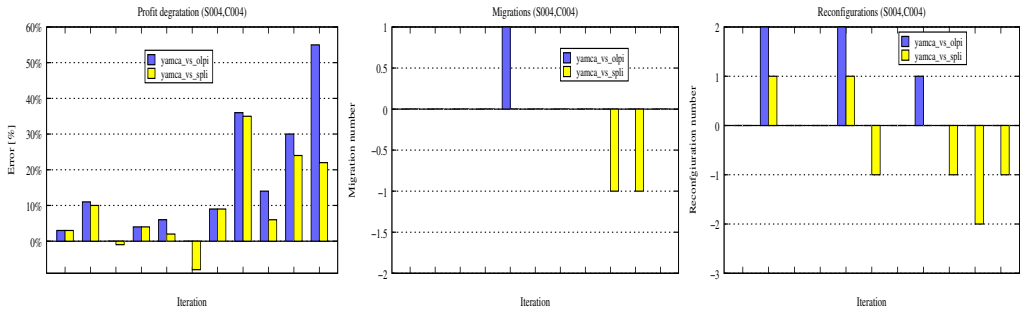



Figura 4.10: Profitto, migrazioni e riconfigurazioni

Un'ultima considerazione, riguarda un comportamento simile a quanto descritto nella sezione 4.2.2. In particolare all'iterazione 3 e 6, $\mathcal{P}_{err}^{slpi-yamca}$ risulta negativa (vedi primo grafico da sinistra della figura 4.10). Si rammenta ancora che questa apparente contraddizione non costituisce un errore di modellazione, ma riassume una caratteristica di completa imprevedibilità della distribuzione di arrivo delle applicazioni.

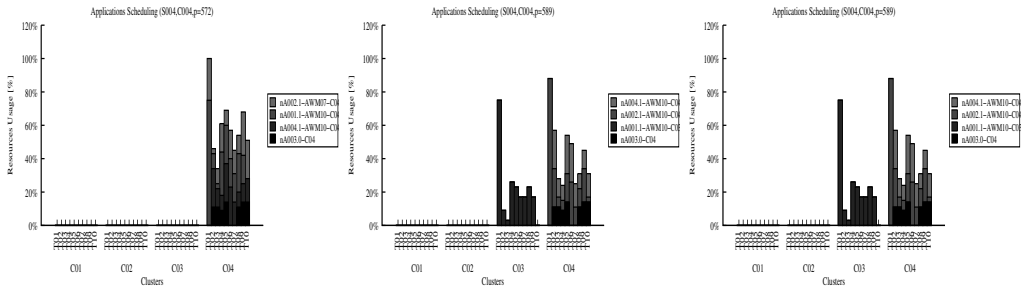



Figura 4.11: Scheduling delle applicazioni e relativi AWM sui cluster disponibili per la prima iterazione. La prima colonna mostra le iterazioni dell'euristica *YaMCA*. Le successive colonne lo scheduling ottenuto rispettivamente con *SLPI* e *OLPI*

.

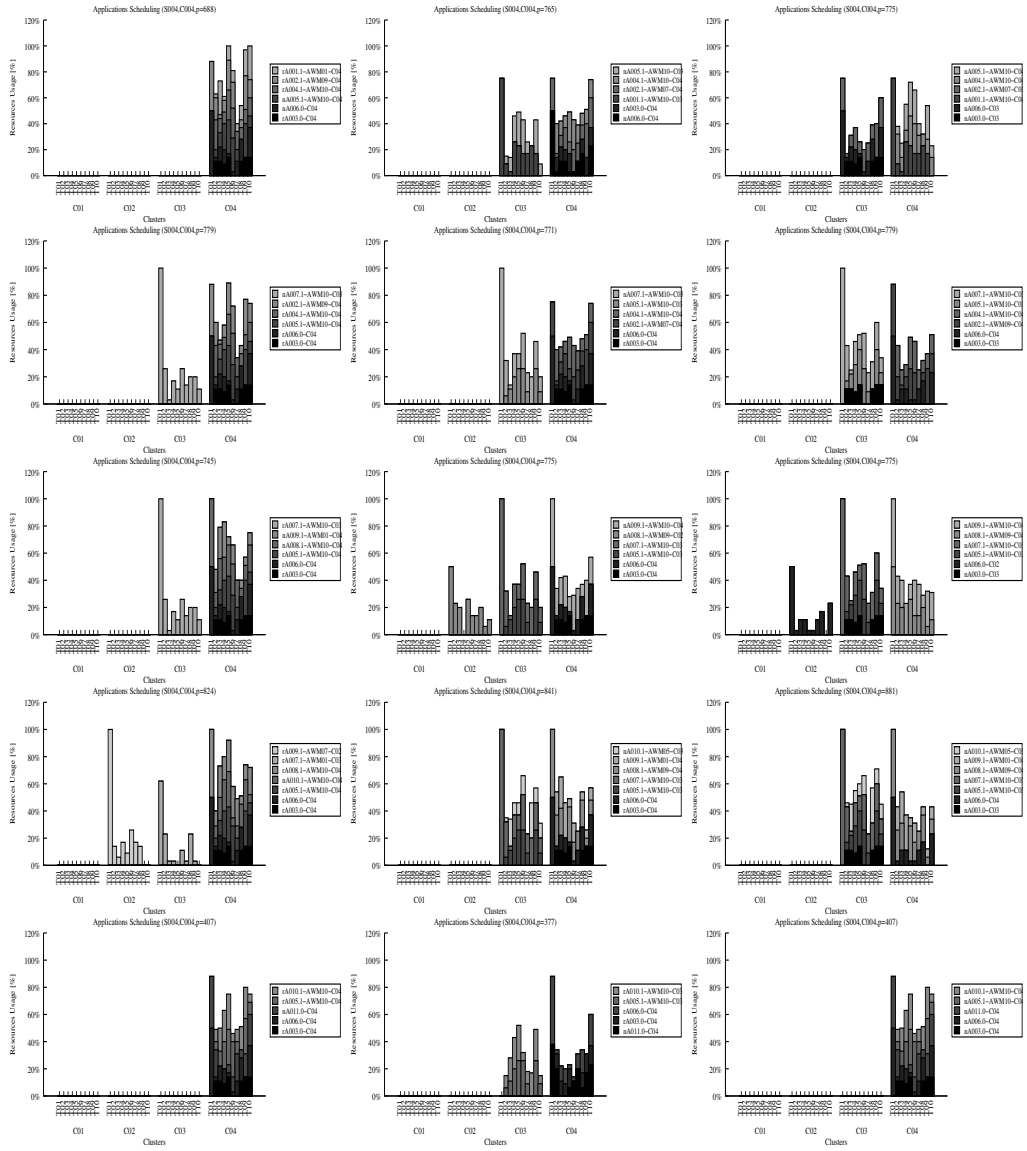



Figura 4.12: Scheduling delle applicazioni e relativi AWM sui cluster disponibili per le iterazioni $2-6$. La prima colonna mostra le iterazioni dell'euristica *YaMCA*. Le successive colonne lo scheduling ottenuto rispettivamente con *SLPI* e *OLPI*

.

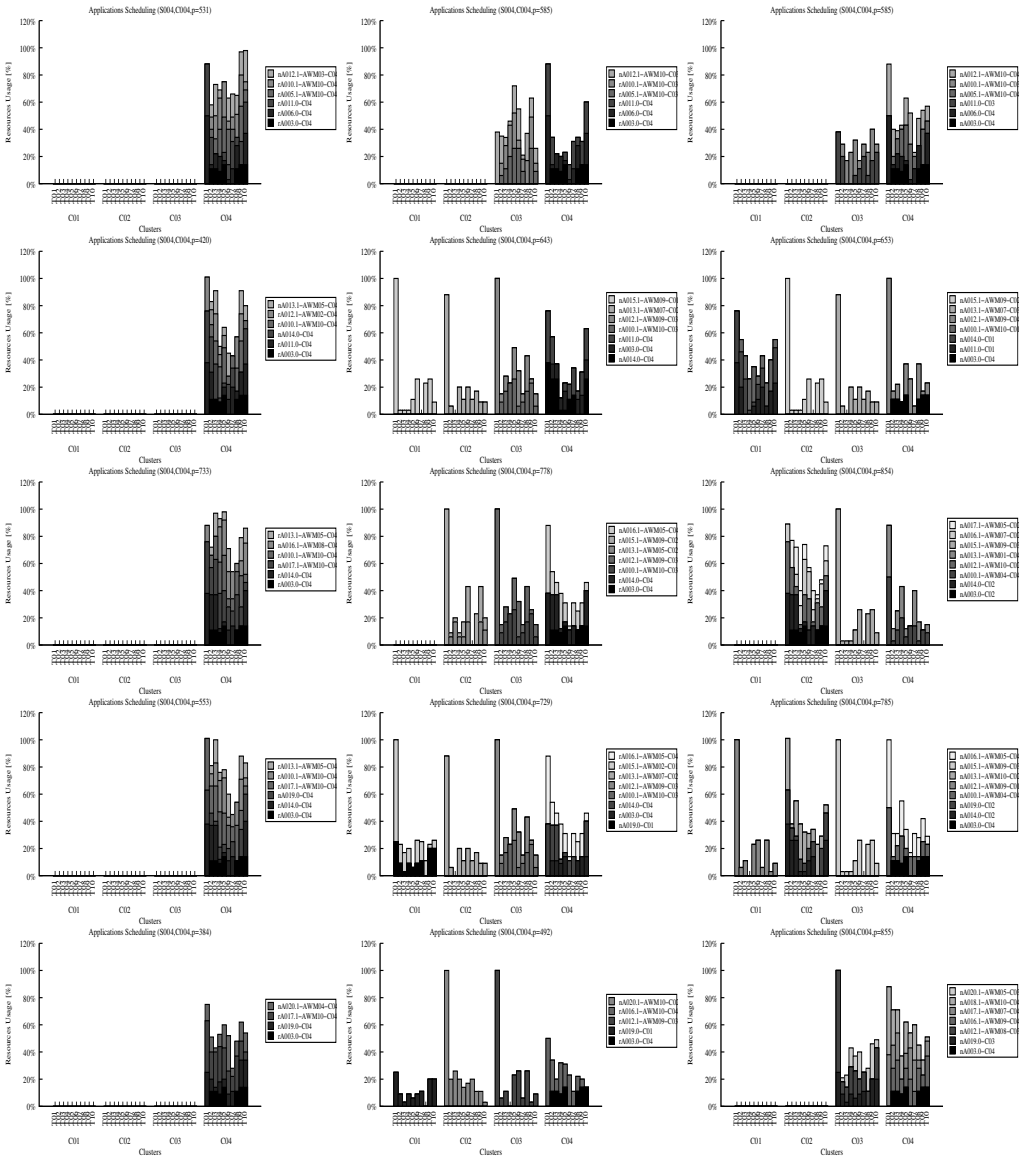



Figura 4.13: Scheduling delle applicazioni e relativi AWM sui cluster disponibili per le iterazioni $7 - 11$. La prima colonna mostra le iterazioni dell'euristica *YaMCA*. Le successive colonne lo scheduling ottenuto rispettivamente con *SLPI* e *OLPI*

.

## 4.3 Conclusioni

In questo capitolo si è cercato di raccogliere alcune evidenze sperimentali relative al comportamento dell'euristica *YaMCA*, il cui schema teorico è stato introdotto nella sezione 3.5.
Tali misure sono state confrontate rispetto a due implementazioni del modello basato sulla programmazione lineare intera (*OLPI* e *SLPI*), proposto nella sezione 3.4.

Dalle misure è emersa la necessità di utilizzare un approccio euristico per la soluzione del problema di allocazione. Infatti gli algoritmi *OLPI* e *SLPI* hanno mostrato complessità temporali ed utilizzo di memoria non indicati per un uso in un sistema embedded reale. L'algoritmo *YaMCA* ha mostrato di poter raccogliere soluzioni qualitativamente accettabili con valori di tempo e consumo di memoria che ne permettono un'implementazione reale.
Un secondo aspetto analizzato riguarda la gestione contemporanea di applicazioni *critiche* e *best-effort* contemporanea. Anche in questo caso l'euristica *YaMCA* ha mostrato di saper gestire tale complessità senza degradare sensibilmente le prestazioni e la qualità della soluzione.

Un'ultima considerazione emersa dalle prove sperimentali ha mostrato un comportamento dell'algoritmo *YaMCA* orientato alla gestione della potenza. Infatti gli scenari di test evidenziano come gli algoritmi *OLPI* e *SLPI* producano soluzioni per le quali le applicazioni risultano distribuite all'interno della piattaforma. Viceversa l'algoritmo euristico tenta di utilizzare al meglio ogni singolo cluster, cercando tuttavia di non degradare il valore di profitto.
Tale comportamento è determinato dalla scelta imposta nell'ordinamento dei cluster da visitare per l'allocazione delle applicazioni. Infatti la politica utilizzata impone all'algoritmo *YaMCA* di visitare prima i cluster già attivi.
Data la natura combinatoria del problema, la soluzione può essere individuata secondo differenti politiche di gestione dell'allocazione. Al fine di catturare anche la possibilità di distribuire le applicazioni in modo

omogeneo rispetto alle risorse, è in fase di realizzazione un'estensione all'euristica proposta.
Dai risultati emersi sembra che la definizione di un differente ordine di visita dei cluster possa determinare un differente metodo di allocazione.

Capitolo 5

# Conclusioni

*" Appena sei diventato maestro in un'arte,*
*affrettati a diventarne discepolo di un'altra"*
Proverbio cinese

In questo lavoro di tesi è stata affrontata la problematica della gestione dinamica delle risorse, nel contesto dei sistemi embedded. In particolare sono state analizzate le proposte emerse da studi precedenti, con lo scopo di migliorarne l'efficacia e l'utilizzabilità.
Più precisamente, sono stati individuati alcuni punti di lavoro sui quali intervenire (vedi sezione 3.1), al fine di ottenere un modello di ottimizzazione usabile in scenari reali.

In particolare è stato considerata la convivenza di applicazioni critiche e *best-effort* nello stesso sistema e la strutturazione in cluster delle risorse disponibili.

Tale studio ha richiesto la formulazione di un nuovo modello per superare i limiti delle soluzioni già presenti in letteratura. Dato lo scopo del lavoro, che richiede un sistema computazionalmente efficiente, è stato proposto un algoritmo euristico, in grado di approssimare i risultati del modello formale con una cifra di complessità spaziale e temporale nettamente inferiore (vedi sezione 4.2.1).

Secondo quanto appreso dall'analisi dei lavori precedenti, non risulta che sia mai stata proposta una soluzione in grado di catturare tutti questi aspetti, relativi alla gestione dinamica delle risorse.

Tali risultati teorici sono stati vagliati attraverso una serie di misure sperimentali. Data la natura sostanzialmente nuova e mai trattata della problematica in esame, così come proposta, è stato necessario predisporre un nuovo sistema per la validazione.
Tale sistema deve essere in grado di misurare aspetti differenti del modello, considerando diversi scenari d'utilizzo (vedi capitolo 4). Da tali misurazioni sono emerse considerazioni interessanti utili ad investigare possibili miglioramenti dell'algoritmo euristico.

La natura, sostanzialmente multidisciplinare, della problematica trattata, ha permesso lo studio e l'implementazione di un prototipo, focalizzandone gli aspetti teorici e di modellazione.
Tuttavia tale prototipo, grazie ai requisiti di modularità imposti nella progettazione, è già oggetto di ulteriori interventi migliorativi e di completamento.
In particolare sono tre i punti chiave sui quali è posta l'attenzione:

1. *Miglioramenti nell'algoritmo euristico.* La soluzione delineata, rappresenta un primo passo in grado di considerare tutti gli aspetti innovativi considerati durante questo lavoro di tesi. E' comunque possibile intervenire al fine di migliorare ulteriormente tale algoritmo, sulla base di alcune considerazioni emerse durante l'analisi. Ad esempio lo scheduling attuale risulta essere troppo consevativo per quanto riguarda l'utilizzo dei cluster. Su questo punto in particolare è focalizzato lo studio attuale;

2. *Necessità di un'implementazione reale.* Una volta individuate le informazioni che le applicazioni ed il RTRM possono scambiarsi, al fine di descrivere le rispettive richieste, è necessario preoccuparsi di fornire un prototipo realmente funzionante del RTRM stesso. Tale lavoro implica un ulteriore studio, ad un livello sperimentale, per individuare come mappare la soluzione proposta su un'architettura

esistente. In altre parole è necessario valutare un protocollo reale di comunicazione tra le applicazioni ed il sistema;

3. *Valori reali per applicazioni e sistema.* Tutta la modellazione e le misure sperimentali sono state svolte utilizzando una visione astratta dell'intero problema. In altre parole è stata considerata la gestione dinamica delle risorse da un punto di vista prettamente teorico, considerando una descrizione del sistema e delle applicazioni dimensionata in modo ragionevole, anche se non direttamente inferita da scenari reali.
   E' quindi ancora necessario uno studio approfondito sulle misure che il sistema e le applicazioni possono fornire in un contesto di lavoro reale, poichè questo è il fine ultimo per il quale il sistema è stato pensato e progettato.

# Bibliografia


[1] Vincent Nollet. *Run-Time management for future MPSoC platforms*. PhD thesis, Technische Universiteit Eindhoven, 2008.

[2] Shahadat Khan, Kin F. Li, Eric G Manning, and M. Mostofa Akbar. Solving the Knapsack Problem for Adaptive Multimedia Systems, pp. 157-178.

[3] Vincent Nollet, Diederik Verkest, and Henk Corporaal. A Safari Through the MPSoC Run-Time Management Jungle. *Journal of Signal Processing Systems*, 2008.

[4] Shahadatullah Khan. *Quality Adaptation in a Multisession Multimedia System: Model, Algorithms and Architecture*. PhD thesis, University of Victoria, 1998.

[5] S. Raina. Introduction to Quality of Service (QoS). *Telecommunications Quality of Service: The Business of Success (QoS 2004)*, pages 48–51, 2004.

[6] K. Nahrstedt and Jingwen Jin. QoS specification languages for distributed multimedia applications: a survey and taxonomy. *IEEE Multimedia*, 11(3):74–87, July 2004.

[7] J. Altmann and P. Varaiya. INDEX project: user support for buying QoS with regard to user's preferences.

[8] Hal R. Varian. The Demand for Bandwidth: Evidence from the INDEX Project, 2001.

[9] Duangdao Wichadakul, Klara Nahrstedt, Xiaohui Gu, and Dongyan Xu. An Integrated Approach of QoS Compilation and Reconfigurable, Component-Based Run-Time Middleware for the Unified QoS Management Framework. pages 373–394, 2001.

[10] Anita Mittal, G. Manimaran, C. Siva, Ram Murthy, Anita Mittal G. Manimaran, and C. Siva Ram Murthy. Integrated Dynamic Scheduling of Hard and QoS Degradable Real-Time Tasks in Multiprocessor Systems, 1998.

[11] Sugawara Tomoyoshi and Tatsukawa Kosuke. Table-based QoS control for embedded real-time systems. *ACM SIGPLAN Notices*, 34(7):65–72, July 1999.

[12] Chen Lee, John Lehoczky, Ragunathan (raj Rajkumar, and Dan Siewiorek. On Quality of Service Optimization with Discrete QoS Options, 1999.

[13] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen. *A scalable solution to the multi-resource QoS problem*. IEEE Comput. Soc.

[14] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal. Run-Time Management of a MPSoC Containing FPGA Fabric Tiles. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(1):24–33, 2008.

[15] S. Alam, M. Hasan, M. Hossain, and A.S.M. Sohail. *Heuristic solution of MMKP in different distributed admission control and QoS adaptation architectures for video on demand service*. IEEE.

[16] Md Mostofa Akbar A, M. Sohel Rahman B, M. Kaykobad B, and E. G. Manning A. Solving the Multidimensional Multiple-choice Knapsack Problem, 2004.

[17] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations (Wiley-Interscience Series in Discrete Mathematics and Optimization)*. John Wiley & Sons, 1990.

[18] Y. Toyoda. A Simplified Algorithm for Obtaining Approximate Solutions to Zero-One Programming Problems. *Management Science*, 21(12):1417–1427, August 1975.

[19] Ch. Ykman-Couvreur, V. Nollet, Fr. Catthoor, and H. Corporaal. *Fast Multi-Dimension Multi-Choice Knapsack Heuristic for MP-SoC Run-Time Management*. IEEE, November 2006.

[20] Abu Zafar M. Shahriar, M. Mostofa Akbar, M. Sohel Rahman, and Muhammad Abdul Hakim Newton. A multiprocessor based heuristic for multi-dimensional multiple-choice knapsack problem. *The Journal of Supercomputing*, 43(3):257–280, August 2007.

[21] Giovanni Mariani, P. Avasare, G. Vanmeerbeeck, Chantal Ykman-Couvreur, Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In *Design, Automation & Test in Europe Conference & Exhibition, 2010. DATE '10.*, 2010.

[22] Local Memory and Christoph Lameter. Local and Remote Memory: Memory in a Linux/NUMA System. *Architecture*, pages 1–25, 2006.

[23] NVIDIA. *Nvidia Cuda C Programming Guide*. Version 3. edition, 2010.

[24] Reto Meier. *Professional Android Application Development*. Wiley, 2009.

[25] Wolfgang Mauerer. *Professional Linux Kernel Architecture (Wrox Programmer to Programmer)*. Wrox, 2008.

[26] By Josh Aas. Understanding the Linux 2.6.8.1 CPU Scheduler, 2005.

[27] Andrea Acquaviva, Andrea Alimonda, Salvatore Carta, and Michele Pittau. Assessing Task Migration Impact on Embedded Soft

Real-Time Streaming Multimedia Applications. *EURASIP Journal on Embedded Systems*, 2008:1–16, 2008.

[28] Andrew Makhorin. Modeling Language GNU MathProg. 2010.

[29] Mhand HIFI. Benchmark Data.

Politecnico di Milano
*Dipartimento di Elettronica e Informazione*
*Piazza Leonardo da Vinci, 32 – 20133 MILANO*