MINISTERO DELLO SVILUPPO ECONOMICO
DIREZIONE GENERALE PER LA LOTTA ALLA CONTRAFFAZIONE
UFFICIO ITALIANO BREVETTI E MARCHI

DOMANDA NUMERO	**102007901513894**
Data Deposito	**13/04/2007**
Data Pubblicazione	**13/07/2007**

Sezione	Classe	Sottoclasse	Gruppo	Sottogruppo
G	06	F		

Titolo

"PROCEDIMENTO E SISTEMA DI SCHEDULAZIONE, GRIGLIA COMPUTAZIONALE E PRODOTTO INFORMATICO RELATIVI"

DESCRIZIONE dell'invenzione industriale dal titolo:
"Procedimento e sistema di schedulazione, griglia computazionale e prodotto informatico relativi"
di: STMicroelectronics S.r.l., nazionalità italiana,
Via C. Olivetti, 2 - 20041 Agrate Brianza (Milano)
Inventore designato: Massimo Orazio SPATA
Depositata il: 13 aprile 2007

* * *

TESTO DELLA DESCRIZIONE

Campo dell'invenzione

L'invenzione si riferisce alle tecniche di allocazione selettiva (c.d. schedulazione) delle risorse ed è stata messa a punto con particolare attenzione al possibile impiego nell'ambito di griglie computazionali, ad esempio per fini di simulazione.

Descrizione della tecnica relativa

Nel seguito della presente descrizione si farà ripetuto accenno a riferimenti bibliografici. Per non appesantire la trattazione, tali riferimenti saranno indicati nel corpo della descrizione con un numero fra parentesi quadre (ad es. [x]) che identifica il

riferimento nell'ambito di un "Elenco dei riferimenti" riportato al fondo della descrizione.

La simulazione al computer è diventata un importante strumento per studiare ed interpretare vari processi fisici. Questa tecnica trova però forti limiti in relazione ai suoi obiettivi ed all'accuratezza dei risultati nelle forti esigenze in termini di potenza computazionale che possono essere a malapena soddisfatte anche utilizzando supercomputer dei tipi più avanzati. Simulare un modello molto complesso richiede potenza di calcolo supplementare mano a mano che il numero di parametri del modello aumenta.

Un modo di affrontare questo problema è fornire la potenza di calcolo richiesta sfruttando una rete di computer (server) collegati fra loro. Dal punto di vista dell'utilizzatore, il ricorso a questa soluzione deve risultare trasparente: si desidera, infatti, che la rete appaia come un unico grande supercomputer virtuale. Questo nuovo paradigma di calcolo è chiamato "griglia". Il fatto di realizzare un ambiente integrato di griglia di calcolo permette di dare origine a infrastrutture di calcolo con potenzialità al di là di ogni paragone.

Dal punto di vista concettuale, una griglia è semplicemente un insieme di risorse di calcolo che svolgono compiti o lavori (task o job) loro assegnati. Appare agli utilizzatori come un unico grande sistema, che offre un unico punto di accesso a risorse distribuite e potenti. Gli utenti trattano la griglia come una singola risorsa computazionale. La griglia accetta i compiti assegnati dagli utenti e li alloca selettivamente (ossia li "schedula") in vista dell'esecuzione su sistemi adatti compresi nella griglia sulla base su politiche di gestione delle risorse. Gli utenti possono quindi affidare alla griglia compiti anche piuttosto gravosi (ad esempio molte attività da svolgere in breve tempo) senza doversi preoccupare in merito a dove tali compiti saranno materialmente eseguiti.

In principali vantaggi legati all'impiego di una griglia computazionale (c.d. Grid Computing) sono quindi:

- la riduzione dei costi di hardware,
- il bilanciamento del carico di lavoro fra le diverse "macchine" (tramite un sistema di gestione dei carichi o Load Management System),
- la capacità di gestire sistemi eterogenei,
- l'aumento di produttività,

- la minore esposizione all'obsolescenza dell'hardware.

In realtà però è difficile definire un approccio di sistema alla gestione delle risorse di griglia tale da risultare davvero trasparente per gli utenti: questo è dovuto in via principale alle caratteristiche architetturali eterogenee della griglia.

Al riguardo, è possibile pensare che gli utenti della griglia mettano in atto un approccio di prenotazione manuale delle risorse della griglia (compiuto tramite strumenti di descrizione dei job, ossia tramite script in JDL - Job Description Language). Questo approccio rimette la scelta e la stima delle risorse necessarie all'utente della griglia e questo implica inevitabilmente rischi di sopravvalutazione o sottovalutazione delle risorse a seguito dell'errore umano con conseguente spreco delle risorse della griglia (vedere in proposito il documento [1]).

E' peraltro già stata sperimentata nel settore dell'informatica l'applicazione di principi e criteri desunti dal mondo dell'economia (si vedano ad esempio i documenti [2] ed in [3]).

In [2] è dimostrato che, applicando un algoritmo di schedulazione basato sul cosiddetto Equilibrio di

Nash come modello economico per sistemi distribuiti, il tempo di attesa medio in coda diminuisce quando aumenta l'utilizzazione delle risorse di sistema. Viceversa si è dimostrato che, usando l'algoritmo ottimale, se l'utilizzazione delle risorse aumenta, il tempo di attesa medio in coda aumenta anch'esso. Infatti, partendo da una coda con carico nullo con job omogenei appartenenti alla stessa classe, emerge che la procedura di schedulazione basata sull'equilibrio di Nash conviene solamente se il fattore di utilizzazione per coda (intesa come contenitore di server di calcolo omogenei) è più alto di una soglia temporale convenuta. Infatti, quando la coda è scarica ed i job hanno una durata media inferiore, ad esempio, ad un ora gli algoritmi di scheduling basati su bilanciamento del carico funzionano già molto bene. Il problema si comincia a presentare, quando i job durano mediamente più di un ora allocando pesantemente la CPU (job di tipo CPU-bound).

In [3] sono confrontate tre procedure di schedulazione diverse: equilibrio di Nash, casuale e MinMin (ottimale). Qui, i risultati mostrano che gli approcci basati sulla teoria dei giochi e sull'equilibrio di Nash sono molto simili ad una strategia di pianificazione casuale, mentre

l'algoritmo ottimale di Pareto (MinMin) risulta il migliore algoritmo di schedulazione. In questo documento non sono date informazioni sul fattore di utilizzazione di sistema, ρ. Infatti, il problema principale delle procedure di schedulazione basate su strategie ottimali (ossia, MinMin) è che queste sono applicabili con vantaggio solo quando il valore di questo fattore resta inferiore ad una soglia data x.

Scopo e sintesi dell'invenzione

L'analisi che precede dimostra che è sentita l'esigenza di tecniche di schedulazione (e dunque di schedulatori) in grado di operare in modo del tutto automatico e in condizioni di trasparenza verso gli utenti. In particolare è sentita l'esigenza di disporre di soluzioni che tengano in conto il fatto che l'architettura di griglia è assimilabile ad un'architettura distribuita che si vuole possa presentare le seguenti caratteristiche:

- eterogeneità: in termini di sistema operativo (OS), velocità di clock, rappresentazione dei dati, memoria, architetture hardware;

- apertura: così da garantire scalabilità e re-implementazione di piattaforme;

- sicurezza: per garantire la riservatezza e l'integrità dei dati;

- scalabilità: intesa come capacità di garantire prestazioni adeguate anche se il numero degli utilizzatori e delle risorse aumenta nel tempo;

- resistenza ai guasti (fault tolerance): in particolare per ciò che riguarda la capacità di mascherare e tollerare cadute momentanee (breakdown);

- sincronizzazione: ossia la capacità di ordinare in modo completo gli eventi, con mutua esclusione, integrità delle operazioni, controllo competitivo dei punti morti (deadlock); e

- trasparenza: con la possibilità di garantire accesso a risorse locali e remote con le stesse modalità, senza apprezzabili perdite in termini di prestazioni e senza dover di necessità conoscere lo stato delle risorse.

In tali sistemi, la nozione di tempo è dunque vitale per dare un ordine preciso agli eventi che derivano da processi parallelizzabili.

La presente invenzione si prefigge lo scopo di dare una risposta completa alle esigenze sopra delineate. Secondo la presente invenzione, tale scopo è raggiunto grazie ad un procedimento avente le caratteristiche richiamate nelle rivendicazioni che

seguono. L'invenzione inoltre si riferisce ad un corrispondente sistema, ad una griglia computazionale che comprende tale sistema nonché ad un prodotto informatico, caricabile nella memoria di almeno un elaboratore e comprendente parti di codice software suscettibili di realizzare le fasi del procedimento quando il prodotto è eseguito su almeno un elaboratore. Così come qui utilizzato, il riferimento ad un tal prodotto informatico è inteso essere equivalente al riferimento ad un mezzo leggibile da elaboratore contenente istruzioni per il controllo del sistema di elaborazione per coordinare l'attuazione del procedimento secondo l'invenzione. Il riferimento ad almeno ad un elaboratore è evidentemente inteso a mettere in luce la possibilità che la presente invenzione sia attuata in forma modulare e/o distribuita.

Le rivendicazioni formano parte integrante dell'insegnamento tecnico qui somministrato in relazione all'invenzione.

In particolare, la soluzione qui descritta si fonda sull'uso di un paradigma di microeconomia per gestire le risorse di griglia. Esiste, infatti, una relazione metaforica tra una griglia ed un modello microeconomico, in cui uno degli obiettivi più

importanti è analizzare i meccanismi di mercato che stabiliscono i prezzi relativi fra i beni e servizi e l'allocazione di risorse limitate fra molti usi alternativi. Tipici settori di studio in microeconomia sono appunto la teoria dei giochi e l'equilibrio di Nash.

Nella forma d'attuazione al momento preferita, la soluzione qui descritta si fonda appunto su un sistema di adattamento di griglia di tipo intelligente (Intelligent Grid Matchmaking System) che attua una procedura di schedulazione basata sulla teoria dei giochi e sull'equilibrio di Nash applicati ai sistemi distribuiti come una griglia computazionale.

Una possibile forma di attuazione della soluzione qui descritta coinvolge l'impiego di un middleware automatico di griglia che sincronizza le azioni di prenotazione delle risorse computazionali in modo da automatizzare l'accesso concorrente a risorse condivise. Questo processo di automazione può essere ottenuto tramite procedure che assicurano un tempo di completamento di lavoro e schedulano lo svolgimento dei lavori in una griglia.

A titolo di sintesi, essendo una griglia computazionale un'architettura distribuita, dove l'eterogeneità è un attributo standard, sia dal lato

simulatori (software e applicazioni) sia dal lato hardware, la soluzione qui descritta permette di automatizzare il processo di sottomissione dei job lato utente, svincolando l'utente stesso da scelte e stime relative alla durata del job da sottomettere, al tipo di hardware compatibile per l'applicazione (ad esempio un'applicazione a 32 bit non verrà eseguita su un server di calcolo a 64 bit), ed alla quantità di risorsa allocata (in termini di tempo di CPU e RAM). L'utente, infatti, spesso non è in grado di stimare in modo corretto questi parametri. Oltre a ciò, questa richiesta di inferenza sul lato utente implica una mancanza di trasparenza; trasparenza che, così come si è visto, costituisce invece una caratteristica preferenziale che ci si aspetta in dovrebbe essere presente in un sistema distribuito come una griglia computazionale.

Nella forma d'attuazione al momento preferita, la soluzione qui descritta persegue l'obiettivo di mascherare l'entropia hardware e software implicita in una griglia attraverso:

- la classificazione dei job e delle risorse in classi omogenee tramite una tecnica basata sull'impiego agenti "profeti" (Prophet Agents: vedere il documento [1]), e

- la creazione di un sistema di adattamento ("matchmaking") fra una specifica classe di job ed una specifica classe di risorse e schedulazione del job sulla coda massimizzando il throughput (High Throughput Computing) con l'impiego di agenti "prigionieri" (Prisoner Agents).

Breve descrizione delle viste annesse.

L'invenzione sarà ora descritta, a puro titolo di esempio non limitativo, con riferimento alle figure annesse, in cui:

- la figura 1 rappresenta, sotto forma di schema a blocchi, un tipico contesto di applicazione della soluzione qui descritta,

- la figura 2 è un diagramma rappresentativo dell'organizzazione funzionale della soluzione qui descritta,

- la figura 3 riporta il diagramma di sequenza ed interazione tra gli agenti operanti nella soluzione qui descritta, e

- la figura 4 riporta cosiddetto goal model dei suddetti agenti.

Descrizione dettagliata di esempi di attuazione

Nella figura 1 dei disegni annessi, i riferimenti R_1, R_2,... R_{n-1}, R_n indicano nel complesso risorse computazionali (nodi di lavoro o Worker Nodes WN), omogenei appartenenti alla stessa coda di tipo noto, collegate fra loro, anche qui secondo criteri di per sé noti, in modo da formare una griglia computazionale.

Un modulo schedulatore S riceve dagli utenti della griglia richieste di svolgimento di "lavori" (job) JR. Tali richieste sono tipicamente riconducibili a richieste di svolgimento di job computazionali, quali job computazionali che corrispondono alla simulazione di un sistema complesso.

Così come meglio rappresentato nello schema funzionale della figura 2, i job JR sono alimentati ad un modulo JC che funge da classificatore e coopera:

- da un lato, con le applicazioni A destinate ad essere eseguite sul sistema (nel seguito della presente descrizione si farà costante implicito riferimento ad una griglia computazionale, anche se tale riferimento non deve essere interpretato in senso limitativo della portata dell'invenzione), e

- dall'altro lato, con un modulo di bilanciamento dei carichi (Load Balancing) LB che a sua volta

interagisce con un modulo HRC che attua la classificazione/assegnazione in rispettive code delle risorse hardware del sistema.

Il modulo HRC si interfaccia con le risorse hardware vere e proprie (CPU e RAM, ad esempio), indicate collettivamente con HR, tramite una funzione AA di analisi architetturale e di amministrazione del sistema, ossia della griglia.

Nell'assenza di più specifiche indicazioni date nel seguito, i moduli/funzioni testé descritti sono da considerarsi nel complesso noti nella tecnica e dunque tali da non richiedere un'ulteriore descrizione particolareggiata in questa sede.

Lo schema della figura 2 evidenzia la presenza, nell'ambito dello schedulatore S, di due classi di agenti (agent: il termine è qui utilizzato nella sua accezione corrente nel mondo delle reti e dei sistemi informatici), ossia:

- agenti "profeti" (Prophet Agents) PHA, del tipo descritto in [1],

- agenti "prigionieri" (Prisoner Agents) PRA, che interagiscono con le code di server omogenei R1, R2, …, Rn-1, Rn e sono in grado di conoscere il tempo di completamento (medio) dei lavori o job richiesti alla

griglia grazie ad informazioni fornite, in modo noto, dagli agenti profeti PHA.

In generale, in un gioco si realizzano le strategie ottimali di Pareto quando l'allocazione di risorse non può migliorare la condizione di un giocatore senza peggiorare la condizione di un altro giocatore. Nel contesto che qui interessa, queste strategie funzionano fino a quando non si eccede un equilibrio di soglia tra le risorse (CPU e RAM, ad esempio) e lavori (job), contrariamente a quanto vale per le strategie di equilibrio di Nash.

Si identifica sperimentalmente pertanto una soglia sotto la quale si applicano le strategie ottimali di Pareto e sopra la quale si applicano le strategie di equilibrio di Nash.

Per spiegare nel dettaglio che cosa si intende esattamente per equilibrio di Nash, può essere utile chiarire alcuni semplici aspetti matematici della teoria dei giochi e definire alcuni concetti basilari.

Un gioco è caratterizzato da:

- un insieme G di giocatori, o agenti, che può essere indicato con i=1,...,N;

- un insieme S di strategie, costituito da un insieme di N vettori $S_i = (s_{i,1}, s_{i,2}, ..., s_{i,j}, ..., s_{i,M_i})$ ciascuno dei quali contiene l'insieme delle strategie che il

giocatore i-esimo ha a disposizione, cioè l'insieme delle azioni che esso può compiere; per brevità si indicherà nel seguito con s_i la strategia scelta dal giocatore i;

- un insieme U di funzioni del tipo: $u_i = U_i(s_1, s_2, ..., s_N)$ che associano ad ogni giocatore i il guadagno (detto anche payoff) u_i derivante da una data combinazione di strategie (il guadagno di un giocatore in generale dipende infatti non solo dalla sua strategia ma anche dalle strategie scelte dagli avversari).

Un equilibrio di Nash per un dato gioco è una combinazione di strategie (indicabile con l'apice *) $s_1^*, s_2^*, ..., s_N^*$ tale che $U_i(s_1^*, s_2^*, ..., s_i^*, ..., s_N^*) \geq U_i(s_1^*, s_2^*, ..., s_i, ..., s_N^*)$ per ogni i e per ogni strategia s_i scelta dal giocatore i-esimo.

Il significato di quest'ultima disuguaglianza è molto semplice: se un gioco ammette almeno un equilibrio di Nash, ogni agente ha a disposizione almeno una strategia dalla quale non ha alcun interesse ad allontanarsi se tutti gli altri giocatori hanno giocato la propria strategia s^*_i. Infatti, come si può desumere direttamente dalla disuguaglianza, se il giocatore i gioca una qualunque strategia a sua disposizione diversa da s^*_i, mentre tutti gli altri

hanno giocato la propria strategia s^*_j, può solo peggiorare il proprio guadagno o, al più, lasciarlo invariato.

Se ne deduce quindi che se i giocatori raggiungono un equilibrio di Nash, nessuno può più migliorare il proprio risultato modificando solo la propria strategia, ed è quindi vincolato alle scelte degli altri. Poiché questo vale per tutti i giocatori, è evidente che se esiste un equilibrio di Nash ed è unico, esso rappresenta la soluzione del gioco, in quanto nessuno dei giocatori ha interesse a cambiare strategia.

L'equilibrio di Nash cosi definito [5, 6] può essere visto come un record delle strategie di equilibrio s* costituite dalle risposte ottimali di tutti gli agenti, ottenute attraverso l'intersezione di set di strategie ottimali per ogni agente.

Verrà ora discusso il modello di payoff di agente: u (x)

Si supponga che M* sia un insieme di lavori (job) appartenenti alla stessa classe, denotato con: j_1, j_2,..., j_m dove $J = \{j_1, j_2, \ldots, j_m\}$ rappresenta l'insieme di alternative disponibili.

I componenti di queste classi J hanno un numero di attributi comuni che possono essere classificati,

secondo i criteri esposti in [1, 7], come attributi funzionali ed attributi non funzionali (NF).

Dati N attributi NF X_1, X_2.,… , X_N, a_{mn} è un valore di attributo X_n di una componente j_m, dove m=1,2,…,M e n=1,2,…, N.

Perciò, secondo il metodo di Von Neumann e Morgenstern si ha:

$$a_n^{(0)} = \min_{1\le m\le M} \{a_{mn}\} \quad a_n^* = \max_{1\le m\le M} \{a_{mn}\}$$

$RX_n = [a_n^{(0)}, a_n^*]$ è poi il campo di valori che un attributo X_n può assumere tra tutte le alternative disponibili.

Per ogni attributo NF X_n si costruisce una funzione di utilità $u_n: RX_n \to [0,1]$.

Dato $x_n \in RX_n$, $u_n(x_n)$ rappresenta la funzione di utilità ottenuta, quando un componente riceve un numero di attributo x_n di X_n.

Per l'evento successivo u_n è indicato semplicemente con u, eliminando n.

Per l'attributo X si abbia $x^{(W)}, x^{(B)} \in RX$, rispettivamente il peggiore ed il migliore tra tutti i possibili valori. Si suppone che la scelta migliore (peggiore) è una di valore alto (basso) e che $x^{(W)}, x^{(B)} \in \{a^{(0)}, a^*\}$ e così che sai $u(x^{(B)}) = 0$ e $u(x^W) = 1$.

La funzione di payoff è data [7] da:

$$u(x) = \frac{x - x^{(W)}}{x^{(B)} - x^{(W)}}$$

Questa formula esprime la matrice di profitto di agente M'.

Per identificare l'algoritmo di schedulazione pianificazione si procede nel modo seguente.

Dati m agenti prigionieri g_1, g_2, …, g_m e dati m lavori j_1, j_2, …., j_m assegnati alla griglia, supponendo che WN_1, WN_2., WN_c siano c nodi di lavoro (Worker Node) inerenti alle stesse classi di server di calcolo, supponendo che il numero di lavori sia più grande del numero di WN è necessario co-allocare più lavori sullo stesso nodo WN. I giocatori-agenti di tipo prigioniero (Prisoner Agent PRA) sceglieranno quali job devono essere allocati su un server secondo la soluzione al Dilemma del Prigioniero offerto dall'Equilibrio di Nash e considerando le possibili strategie adottabili per massimizzare il proprio profitto, sulla base di congetture degli agenti.

Si formula quindi un modello in termini di:

- giocatori=agenti=jobs: j_1, j_2,…,j_m
- mosse disponibili: scelta di WN_1, scelta di WN_2,…, scelta di WN_c

- profitto: massimizzazione della probabilità di ottimizzare la probabilità del tempo di completamento di un lavoro per un agente j_i
- matrice di payoff $M' = \{M'_1, M'_2., M'_p\}$

La Tabella 1 riprodotta qui sotto illustra la notazione nella formulazione secondo la teoria dei giochi.

Tabella 1

Simbolo	**Definizione**
T	Tempo di completamento stimato per lavoro j-esimo
WN_j	Nodo di lavoro j-esimo
μ	Tempo di servizio di un nodo WN per una specifica classe di lavoro
C	Numero di WN appartenenti ad una Coda
J	Classe omogenea di lavori $[j_1, \ldots, j_m]$
S*	Strategia di vettore di equilibrio di Nash
M*	Insieme di lavori appartenenti alla stessa classe
N	Numero di attributi NF appartenenti alla stessa classe
X_N	Classe di attributi NF

a_{mn}	Valore di attributo di Xn per la Componente j_m con m=1,..,M * e n=1,…,N
u(x)	Funzione di utilità per una classe di attributi NF
$X^{(W)}n$	Valore peggiore della classe X_n di attributi NF
$X^{(B)}n$	Valore migliore della classe X_n di attributi NF
RX_n	Campo di valori per la classe X_n di attributi NF
g_i	Agente-giocatore i-esimo con i=1,…,m
P	Numero di matrice di payoff
N	Numero attributi NF X
M	Numero di lavori-agenti
M′	Matrice di payoff del dilemma d prigioniero

Volendo applicare questo modello ad un specifico esempio si supponga di avere tre agenti per tre lavori che devono essere schedulati su due WN.

Le possibili mosse sono equivalenti a tutte le possibili allocazioni di lavori j_1, j_2, …, j_m sui nodi

WN_1, WN_2,..., WN_c. Il numero totale di mosse disponibili per ogni agente è dato dal numero di WN selezionabili elevato al numero dei lavori-agenti c^m

Nel modello qui proposto il payoff è ottimizzare la probabilità di tempo di completamento del lavoro [1]. La matrice di payoff $M'=\{M'_i\}$ con $i = 1,\ldots,p$ è costituita da un numero di matrici uguale a p ed un numero di righe e colonne pari al numero c dei nodi WN. Le righe di matrice sono record riferiti al numero di agenti/lavori.

Quindi $p=\begin{cases}2*(m-2)....if\,(m>2)\\1.....if\,(m=2)\end{cases}$

Nell'esempio qui considerato, la matrice di payoff per il dilemma del prigioniero può essere rappresentata come una matrice di dimensioni c*c*p. Nell'ipotesi che ogni agente trovi un lavoro, il payoff è rappresentato con record del tipo: (x, y, z) (vedi documento [8]).

Per costruire la matrice di profitto si costruisce una matrice di attributi per ogni lavoro:

- (X_1): tempo di servizio stimato di un lavoro (μ_{WNj})
- (X_2): lunghezza della coda (L_q)
- (X_3): frequenza stimata di interarrivo dei lavori (λ_{WNj}).

Quindi, seguendo il modello descritto nel documento [7], le colonne di attributo sono (X_1, X_2, X_3) e sono rappresentate dalla seguente matrice:

Attributi	μ	L_q	Λ
Lavori			
j_1	a_{11}	a_{12}	a_{13}
j_2	a_{21}	a_{22}	a_{23}
...	...	...	...
j_m	a_{m1}	a_{m2}	a_{mn}

Si calcola il valore di tre componenti di attributi NF rispetto ad n lavori j_1, j_2, …,j_n per ogni vettore di attributi NF X_1, X_2 X_3 e per ogni possibile scelta di nodi (WN1, WN2).

Poi si calcola $u(x)_{mn}$ su WN_1 se $x_1 = \mu$, $x_2 = Lq$, $x_3 = \lambda$. e poi su WN2:

$$u(x_1)_{11} = \frac{x_1 - x_1^{(W)}}{x_1^{(B)} - x_1^{(W)}}$$

$$u(x_2)_{12} = \frac{x_2 - x_2^{(W)}}{x_2^{(B)} - x_2^{(W)}}$$

$$u(x_3)_{13} = \frac{x_3 - x_3^{(W)}}{x_3^{(B)} - x_3^{(W)}}$$

Si calcolano in modo analogo $u(x_1)_{21}$, $u(x_2)_{22}$ ed $u(x_3)_{23}$ per WN_2 e così via sino a WN_c.

In questo modo si hanno tre "pesi" di utilità per lavoro su ogni WN. Queste tre componenti rappresentano pesi associati a rispettivi attributi; il valore medio offre una funzione di utilità unica (ponderata) per ogni lavoro.

Sostituendo i valori x_1, x_2 e x_3 nella funzione u(x) per WN_1 si ha:

$$\left.\begin{array}{l} g_1(u(x_1)_{11}, u(x_2)_{12}, u(x_3)_{13}) \\ g_2(u(x_1)_{21}, u(x_2)_{22}, u(x_3)_{23}) \\ \ldots\ldots\ldots \\ g_m(u(x_1)_{m1}, u(x_2)_{m2}, u(x_3)_{m3}) \end{array}\right\} u(x)_{WN_1} \Rightarrow \begin{cases} u(x)_{WN_1 j_1} \\ u(x)_{WN_1 j_2} \\ \ldots\ldots\ldots \\ u(x)_{WN_1 j_m} \end{cases}$$

Analogamente per $WN_2, \ldots, WN_c$.

Poi, il dilemma del prigioniero della teoria dei giochi, applicato al modello, dà la matrice p M' = $\{M'_1, M'_2, \ldots, M'_p\}$ dove i valori vettoriali sono stati calcolati tramite u(x).

In altre parole, ogni agente prigioniero G fa congetture sulle altre strategie di agente e fa la migliore scelta (con valore di profitto più alto) per lui, assicurandosi che ciascun altro agente non abbia altre strategie con profitto u(x) più lato, spostandosi su tutta la matrice e seguendo soluzioni

di equilibrio di Nash per il problema del dilemma del prigioniero.

Quindi, per esempio, dati tre lavori e due nodi WN, la matrice del dilemma del prigioniero M' è:

M'_1:

		j_3 sceglie WN_1	
		J_2	
j_1		WN_1	WN_2
	WN_1	$(u(x)_{WN1j1}, u(x)_{WN1j2}, u(x)_{WN1j3})$	$(u(x)_{WN1j1}, u(x)_{WN2j2}, u(x)_{WN1j3})$
	WN2	$(u(x)_{WN2j1}, u(x)_{WN1j2}, u(x)_{WN1j3})$	$(u(x)_{WN2j1}, u(x)_{WN2j2}, u(x)_{WN1j3})$

M'2:

		j_3 sceglie WN_2	
		j_2	
j_1		WN_1	WN_2
	WN_1	$(u(x)_{WN1j1}, u(x)_{WN1j2}, u(x)_{WN2j3})$	$(u(x)_{WN1j1}, u(x)_{WN2j2}, u(x)_{WN2j3})$
	WN2	$(u(x)_{WN2j1}, u(x)_{WN1j2}, u(x)_{WN2j3})$	$(u(x)_{WN2j1}, u(x)_{WN2j2}, u(x)_{WN2j3})$

Dove M'1 è la matrice d'ingresso numero 1 e $u(x)_{WNiji}$ rappresenta la funzione di ponderazione ottenuta da singole componenti di attributo per il lavoro i-esimo sul j-esimo nodo WN.

L'Equilibrio di Nash è calcolato sulla matrice di profitto nel modo seguente:

- payoff fisso per gli agenti g_1 e g_3 e poi il secondo agente g_2 si muove sulla riga (asse x della matrice) per verificare se esiste una strategia migliore (in termini di payoff u(x), prima componente);

- payoff fisso per gli agenti g_2 e g_3 e poi il primo agente g_1 si muove sulle colonne (asse y della matrice) per verificare se esiste una strategia migliore (in termini di payoff u(x), seconda componente);

- payoff fisso per gli agenti g_1 e g_2 e poi il terzo agente g_3 si muove sull'asse z della matrice per verificare se esiste una strategia migliore (in termini di payoff u(x), terza componente) [8].

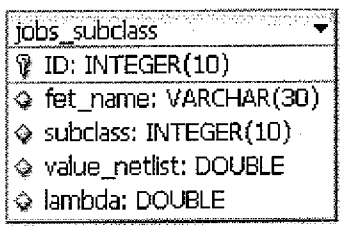
Nel seguito sarà descritta in ancora maggior dettaglio, anche con riferimento alle figure 3 e 4, una possibile forma d'attuazione di un sistema ad agenti che mira a realizzare, su una griglia

computazionale, un'infrastruttura, capace di analizzare l'insieme dei job eseguiti sulla griglia al fine di ottenere un algoritmo di schedulazione (ad esempio di tipo HTC).

In tale possibile forma d'attuazione, le principali entità facenti parte del sistema sono:

- una base dati (jobInfo) che rappresenta ed identifica i lavori o job da eseguire;
- AgentNash (o agente di Nash NA);
- AgentProphet (o agente profeta PHA);
- AgentInterpreter (o agente interprete IA);
- FindNashEquil (che rappresenta la funzione di ricerca dell'equilibrio di Nash.

Il database jobInfo presenta una struttura iniziale che permette di classificare ogni job in una serie di sottoclassi, ad esempio facendo ricorso ad una tabella del tipo:



dove

- fet_name: individua il tipo di job;

- subclass: è un intero che specifica la sottoclasse del job in questione;
- value_netlist: è un parametro di classificazione, di cui sono dati maggiori dettagli in seguito; e
- lambda: tempo medio di interarrivo dei job JR.

L'agente di Nash NA è l'agente principale del sistema. Esiste come unica istanza, che analizza, ad esempio, 6 job alla volta da schedulare su, ad esempio 4 worker node. Il suo comportamento è definito dalla classe NashBehaviour.

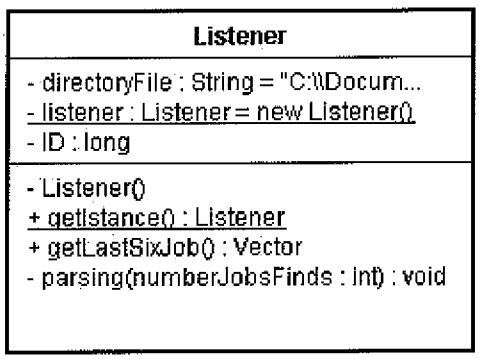
L'obiettivo o Target e realizzare un file (chiamato: 'matrix.txt') contenente le matrici di pay-off, per gli ultimi 6 jobs. Dopo aver definito la matrice su file, si procede a trovare l'equilibrio di Nash.

I passi computazionali prevedono che, in primo luogo, l'agente di Nash istanzi un oggetto Listener L, che conosce le informazioni sui job JR immessi nella griglia, per poi prelevare ciclicamente, tramite il Listener L gli ultimi (ad es.) 6 job sottomessi sulla griglia, e le relative informazioni.

L'agente di Nash istanzia poi per ogni job un agente profeta PHA, per ricevere tramite questo, dall'agente prigioniero PRA, nu*netlistComplexity e lambda.

Partendo dalle 6 coppie di nu* netlistComplexity e lambda, l'agente di Nash realizza la matrice degli NF-attributi e calcola la funzione di utilità su ogni worker node partendo dagli NF-attributi. Calcola poi la funzione U a partire dalla funzione di utilità e scrive la matrice ottenuta dall'analisi dei 6 job sui 4 worker node, sfruttando il dilemma del prigioniero.

In maggior dettaglio, l'Agente Listener L è definibile nel modo seguente



La classe è realizzata sulla base di un design pattern di tipo Singleton, il cui scopo è

permettere la realizzazione di una sola istanza, quest'ultima referenziata dall'agente di Nash NA.

Tale classe è introdotta nel sistema ad agenti per reperire le informazioni sull'insieme dei job eseguiti sulla griglia, in particolare per consentire al chiamante, ossia all'agente di Nash NA il reperimento delle informazioni discusse. A tal fine si prevede all'interno della classe una variabile privata che punta al file di log che contiene le informazioni sui job.

Ciclicamente dall'agente di Nash NA cerca di reperire gli ultimi job (ad esempio gli ultimi 6 job) sottoposti nella griglia tramite l'unica istanza della classe Listener.

Qualora non ci siano ancora nel sistema ad es. 6 job in attesa di schedulazione, l'agente Listener L ritornerà "null", altrimenti sarà restituito all'invocatore un oggetto Vector, il quale contiene in ogni locazione un array di stringhe, riferito ad un unico job, così composto:

array[0]= tabella del Data Base su cui effettuare le query;

array[1]= feat_name, vale a dire il nome della feature, per effettuare le query;

array[2]= tipo di worker node: nelle simulazioni tutti i nodi saranno di tipo SUNSO;

array[3]= path Name del file top.CIR associato al job. Tale valore deve essere necessariamente reperito dinamicamente dall'ascoltatore;

array[4]= id del job, parametro non statico, assegnato tramite l'utilizzo di una variabile incrementale di tipo long;

array[5]=null, sarà l'agentNash a settare tale valore con il proprio nome.

Ogniqualvolta viene lanciata una nuova esecuzione del sistema ad agenti per una simulazione dell'algoritmo di schedulazione, l'agente listener L comincia a leggere partendo di preferenza dalla fine del file, in modo da esser certi che non vengano presi in considerazione informazioni su job "vecchi".

Per ciò che riguarda gli agenti profeti PHA, è previsto che per ogni job venga istanziato un agente di questo tipo, il cui comportamento è definito da una classe ProphetBehaviour, con l'obiettivo (target) di restituire all'agente di Nash NA la coppia (nu*netlistComplexity, lambda).

I passi computazionali di un agente profeta PHA sono tipicamente:

- ricevere in input i parametri del job dall'agente Nash;

- creare una istanza della classe Database;

- avviare 2 query tramite l'istanza del passo 2 ed estrarre rispettivamente nu e lambda;

- creare un agente Interprete IA ed attendere da questo il valore della netlistComplexity (a tal fine indica dove è collocato il file di TOP.CIR); e

- restituire all'agente di Nash NA il parametro di nu*netlistComplexity (tempo medio di completamento) e lambda (tempo medio di interarrivo).

Un'istanza dell'agente interprete IA viene creata per analizzare il file di netList TOP.CIR associato ad ogni job. Il comportamento di tale agente è definito da una classe InterpreterBehaviour con l'obiettivo di:

- calcolare la netlistComplexity, e

- inviare tale valore all'agente profeta che ne ha fatto richiesta.

I passi computazionali di un agente interprete IA sono tipicamente:

- individuare tutti i file dipendenti a partire dal file TOP.CIR;

- contare il numero di righe non nulle del file;

- calcolare la netlistComplexity.

In modo preferito all'interno dell'agente esiste un array del tipo:

```
private final String wordSerch[] = {
    ".tran",  ".pss",  ".hb",  ".dc",  ".ac",
".noise"};
```

che descrive l'insieme delle parole chiavi da ricercare per definire la netlistComplexity. Ogniqualvolta in un file si riscontra una della parole chiavi nell'insieme di cui sopra, si preleva dal sottostante array il peso associato:

```
private final double weightWord[] = {
    0.8, 0.5, 0.5, 0.5, 0.8, 0.8};
```

In ogni file che si analizza, a partire dal file TOP.CIR, se si dovesse individuare più volte la stessa parola chiave, per il sistema equivale a trovarla una volta solamente.

Le tipiche modalità di interazione fra gli agenti descritti (L = Listener; PRA = agente prigioniero; PHA = agente profeta; IA = agente interprete) sono descritte nella figura 3, dove si apprezzerà che i messaggi di ritorno sono inseriti

al solo fine di agevolare la leggibilità dello schema.

La lettura della figura 3 prende spunto dall'agente prigioniero PRA che, in un passo 50, chiede al listener L le informazioni sugli ultimi (ad es.) 6 job sottoposti alla griglia così come contenute nel rispettivo file di analisi LSB.event().

Il passo 51 esprime poi la richiesta, da parte dell'agente prigioniero PRA all'agente profeta PHA, dei valori nu*netlistComplexity, lambda per il singolo job fra quelli considerati. L'agente profeta PHA "gira" in un passo 54 all'agente interprete IA la richiesta di netlistComplexity, mentre il passo 55 esprime la lettura del file NetList relativo al job da parte dell'agente IA interrogato.

I passi 54 e 55 corrispondono poi alla definizione, da parte dell'agente prigioniero PRA, della matrice di payoff ed al reperimento, sempre da parte dell'agente prigioniero PRA, del relativo equilibrio di Nash.

La figura 4 rappresenta infine gli scopi (goal e sub-goal) di alto livello delle varie componenti ossia degli agenti sopra descritti.

In particolare, a livello di (sotto)scopi o subgoal sono previste le seguenti funzioni:

- ricevere informazioni sugli ultimi (ad es. 6) job sottomessi (101);
- calcolare la netlist complexity (102);
- calcolare gli attributi non funzionali (103);
- creare le matrici di payoff (104), con lo scopo qualitativo di inviare tempestivamente all'agente di Nash NA i valori computati (106);;
- cercare l'equilibrio di Nash (105).

Il tutto con lo scopo finale 200 di definire un algoritmo di schedulazione (ad esempio di tipo HTC).

La ricerca dell'equilibrio di Nash da parte degli agenti prigionieri PRA applicato sulla matrice del dilemma del prigioniero può essere esemplificata (sempre assumendo - così come è ragionevole fare - che i lavori della griglia ed i nodi WN possano essere ripartiti in classi omogenee) dallo pseudocodice riprodotto qui di seguito. Tale pseudocodice è riferito alla schedulazione di un lavoro di griglia tramite allocazione ottima delle risorse con la scelta di una classe di nodi WN omogenei.

Begin
Calculate NF-attributes values a_{mn} for matrix (4.a);
Calculate payoff values $u(x)$ for matrix (4.b) using formulas (4.1, 4.2);
Calculate z_max = max($u(x)_{WNij3}$) for $i = 1, \ldots, c$;
for each matrix M'
/*fixed payoff component for agent j_1 and j_3 and playing with agent j_2 moving on x – axis of matrix (4.b) to verify if exists a better strategy for him:*/
for each $WN_i \in \{WN_1, WN_2, \ldots, WN_c\}$
if $(u(x)_{WN1j2} > u(x)_{WNij2})$ **then**
/*fixed payoff component for agent j_2 and j_3 and playing with agent j_1 moving on y – axis of matrix (4.b) to verify if exists a better strategy for him:*/
if $(u(x)_{WN1j1} > u(x)_{WNij1})$ **then**
/*fixed payoff component for agent j_1 and j_2 and playing with agent j_3 moving on z – axis of matrix (4.b) to verify if exists a better strategy for him:*/
if $(u(x)_{WN1j3} >= z\_max$ **then**
$s^*=[u(x)_{WN1j1}, u(x)_{WN1j2}, u(x)_{WN1j3}]$;
end if
else
/* given i' > i
if $(u(x)_{WNij1} > u(x)_{WNi'j1})$ **then**
if $(u(x)_{WNij3} >= z\_max$ **then**
$s^*=[u(x)_{WNij1}, u(x)_{WN1j2}, u(x)_{WNij3}]$;
end if
end if
else
// given i' > i
if $(u(x)_{WNij2} > u(x)_{WNi'j2})$ **then**
if $(u(x)_{WNij1} > u(x)_{WNi'j1})$ **then**
if $(u(x)_{WNij3} >= z\_max$ **then**
$s^*=[u(x)_{WNij1}, u(x)_{WNij2}, u(x)_{WNij3}]$;
end if
end if
end if

```
      end if
    end For
end For
return s*
End
```

La soluzione qui descritta permette di superare le limitazioni intrinseche date dall'infrastruttura intrinsecamente eterogenea e complessa legata alle tecniche di Grid Computing.

La soluzione qui descritta si presta alla realizzazione di un middleware efficiente in grado di eseguire applicazioni distribuite in modo da migliorare le prestazioni, aumentare la velocità di esecuzione, ed automatizzare le procedure di richiesta degli utenti. Gli utenti non devo quindi fare alcuna ipotesi di stima sulle caratteristiche dei lavori affidati alla griglia (ad esempio: esigenze in termini di memoria o CPU). L'accesso concomitante a risorse di calcolo distribuite e condivise si basa su una procedura di "Negoziazione di Risorse" (Resource Negotiation).

Questo meccanismo di affidamento dei lavori può basarsi su un approccio di negoziazione automatica, superando l'approccio (attuato dall'utente) di

prenotazione di manuale; approccio, quest'ultimo, che rimette la scelta e la stima delle risorse necessarie agli utenti della griglia, con conseguente rischio di sopravvalutazione o sottovalutazione di risorse e di spreco di risorse di griglia.

Elenco dei riferimenti

[1] Massimo Orazio Spata, Giuseppe Pappalardo, Salvatore Rinaudo, Tonio Biondi: "Agent-based negotiation techniques for a Grid: the Prophet Agents" - 2nd IEEE International Conference on e-Science and Grid Computing 2006

[2] D. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini: "Economic models for allocating resources in computer systems", in Scott Learwater, Editor, "Market-Based Control: A Paradigm for Distributed Resource Allocation", World Scientific, Hong Kong, 1996.

[3] Y.-K. Kwok, S. Song, and K. Hwang, "Non-Cooperative Grids: Game Theoretic Modeling and Strategy Optimization", submitted to IEEE Trans. Parallel and Distributed Systems, Dec. 2004.

[4] Donald Gross, Carl M. Harris: "Fundamentals of Queueing Theory", Wiley Interscience 1998 - ISBN:

978-0-471-17083-9 Par. 2.3 Queues with parallel channels (M/M/c), page 69

[5] Nash, John F., Jr. - "Equilibrium points in n-person games" *Proc. Nat. Acad. Sci. U. S. A.* **36**, (1950). 48–49

[6] Jose' M. Vidal: "Learning in Multiagent Systems: An Introduction from a Game-Theory", University of Southern Carolina, Computer Science and Engineering, Columbia, 2003, in Eduardo Alonso, Editor, Adaptive Agents: LNAI 2636. Springer Verlag, 2003.

[7] Merad S., de Lemos R., and Anderson T.: "Dynamic Selection of Software Components in the Face of Changing Requirements" - Department of Computing Science, University of Newcastle upon Tyne, UK, Technical Report No. 664, 1999.

[8] Martin J. Osborne, Ariel Rubinstein: "A Course in Game Theory" - The MIT Press (July 12, 1994) - pag. 9 par. "Strategic Games"

Naturalmente, fermo restando il principio dell'invenzione, i particolari di realizzazione e le forme di attuazione potranno essere variati, anche in modo significativo, rispetto a quanto qui descritto ed illustrato, senza per questo uscire dall'ambito

dell'invenzione, così come definito dalle rivendicazioni annesse.

RIVENDICAZIONI

1. Procedimento per schedulare (S) lo svolgimento di lavori (JR) tramite le risorse di una griglia computazionale (R_1, R_2, …, R_{n-1}, R_n), caratterizzato dal fatto che comprende le operazioni di:

- identificare una soglia di equilibrio fra dette risorse e detti lavori (JR);
- al di sotto di detta soglia di equilibrio, schedulare lo svolgimento di detti lavori (JR) tramite le risorse di detta griglia computazionale (R_1, R_2, …, R_{n-1}, R_n) secondo strategie ottimali di Pareto; e
- al disopra di detta soglia di equilibrio, schedulare lo svolgimento di detti lavori (JR) tramite le risorse di detta griglia computazionale (R_1, R_2, …, R_{n-1}, R_n) secondo strategie di equilibrio di Nash.

2. Procedimento secondo la rivendicazione 1, in cui detta soglia di equilibrio è una soglia temporale in termini di durata di svolgimento di detti lavori (JR).

3. Procedimento secondo la rivendicazione 1 o la rivendicazione 2, in cui dette risorse di detta griglia computazionale (R_1, R_2, …, R_{n-1}, R_n)

comprendono risorse scelte fra risorse CPU e risorse di memoria.

4. Procedimento secondo una qualsiasi delle rivendicazione 1 a 3, in cui dette strategie ottimali di Pareto comportano l'allocazione di detti lavori (JR) a dette risorse di detta griglia computazionale (R_1, R_2, ..., R_{n-1}, R_n) raggiungendo la condizione in cui la condizione di lavoro di una di dette risorse di detta griglia computazionale (R_1, R_2, ..., R_{n-1}, R_n) non è migliorabile senza peggiorare la condizione di lavoro di un'altra delle risorse di detta griglia computazionale (R_1, R_2, ..., R_{n-1}, R_n).

5. Procedimento secondo una qualsiasi delle rivendicazione 1 a 4, in cui dette strategie di equilibrio di Nash comportano l'allocazione di detti lavori (JR) a dette risorse di detta griglia computazionale (R_1, R_2, ..., R_{n-1}, R_n) raggiungendo la condizione in cui le risorse di detta griglia computazionale (R_1, R_2, ..., R_{n-1}, R_n) non hanno interesse ad allontanarsi dalla propria strategia di allocazione se tutti le altre risorse di detta

griglia computazionale (R_1, R_2, …, R_{n-1}, R_n) hanno adottato la loro strategia di allocazione.

6. Procedimento secondo una qualsiasi delle precedenti rivendicazioni, in cui detto equilibrio di Nash è valutato come record delle strategie di equilibrio costituite dalle risposte ottimali di dette risorse di detta griglia computazionale (R_1, R_2, …, R_{n-1}, R_n) ottenute tramite l'intersezione di insiemi di strategie ottimali per ciascuna risorsa.

7. Procedimento secondo una qualsiasi delle precedenti rivendicazioni, comprendente le operazioni di:

- produrre un modello di detta griglia computazionale (R_1, R_2,…, R_{n-1}, R_n) comprendente agenti rappresentativi di detti lavori (JR) suscettibili di compiere mosse disponibili corrispondenti alla scelta di una determinata risorsa per lo svolgimento di un determinato lavoro (JR), e
- valutare detto equilibrio di Nash in funzione della massimizzazione del profitto di detti agenti.

8. Procedimento secondo la rivendicazione 7, comprendente l'operazione di identificare detto profitto come massimizzazione della probabilità di ottimizzare la probabilità nel tempo di completamento di un determinato lavoro per uno di detti agenti.

9. Procedimento secondo la rivendicazione 7 o la rivendicazione 8, comprendente l'operazione di valutare detto profitto sulla base di una matrice di profitto con associati attributi per ogni lavoro.

10. Procedimento secondo la rivendicazione 9, in cui detti attributi sono scelti fra:

- tempo di servizio stimato per un determinato lavoro da parte di una risorsa di detta griglia computazionale (R_1, R_2, ..., R_{n-1}, R_n),
- lunghezza della coda di svolgimento di detto servizio, e
- frequenza stimata di interarrivo dei lavori.

11. Procedimento secondo una qualsiasi delle precedenti rivendicazioni, comprendente l'operazione di determinare, per ciascuna di dette risorse di detta griglia computazionale (R_1, R_2, ..., R_{n-1}, R_n) una

funzione di utilità unica per ciascuno di detti lavori.

12. Procedimento secondo la rivendicazione 9 e la rivendicazione 11, comprendente l'operazione di determinare detta funzione di utilità unica come media, eventualmente ponderata, di una pluralità di componenti di utilità rappresentative di pesi associati a rispettivi attributi per un determinato lavoro.

13. Procedimento secondo una qualsiasi delle precedenti rivendicazioni, comprendente le operazioni di:

- classificare detti lavori (JR) e dette risorse di detta griglia computazionale (R_1, R_2, ..., R_{n-1}, R_n) in classi omogenee, e

- creare un adattamento fra le classi di detti lavori (JR) e le classi di dette risorse di detta griglia computazionale (R_1, R_2, ..., R_{n-1}, R_n) massimizzando il throughput computazionale della griglia.

14. Procedimento secondo una qualsiasi delle precedenti rivendicazioni, comprendente le operazioni di:

- creare, per detti lavori (JR), rispettivi agenti profeta (PHA), suscettibili di acquisire (IA) informazioni sul tempo medio di completamento e sul tempo medio di interarrivo dei rispettivi lavori (JR),

- creare un agente principale (NA) per schedulare lo svolgimento di detti lavori (JR) tramite dette risorse della griglia computazionale, detto agente principale (NA) essendo configurato per:

  - ricevere da detti rispettivi agenti profeta (PHA) dette informazioni sul tempo medio di completamento e sul tempo medio di interarrivo dei rispettivo lavori (JR),

  - realizzare un file di matrici di pay-off, per un insieme di detti lavori (JR) arrivati per ultimi a detta griglia computazionale, trovando il relativo equilibrio di Nash.

15. Dispositivo schedulatore (S) configurato per attuare il procedimento secondo una qualsiasi delle rivendicazioni 1 a 14.

16. Griglia computazionale (R_1, R_2,....R_{n-1}, R_n) comprendente una pluralità di risorse per lo svolgimento di lavori (JR), detta griglia computazionale comprendendo uno schedulatore secondo la rivendicazione 15.

17. Prodotto informatico, caricabile nella memoria di almeno un elaboratore elettronico e comprendente porzioni di codice software configurate per realizzare il procedimento secondo una qualsiasi delle rivendicazioni 1 a 14.

Fig.1

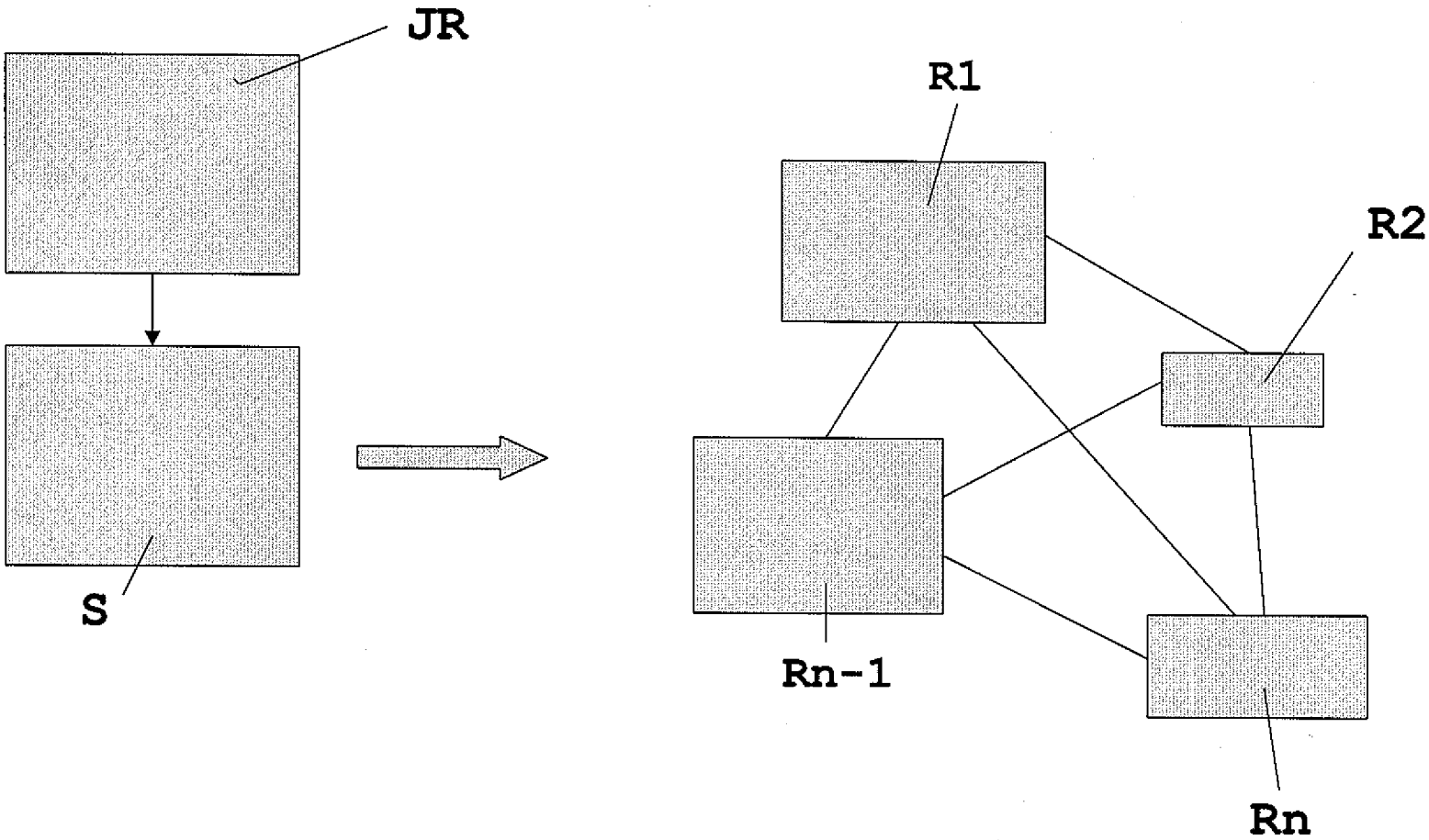


Fig. 2

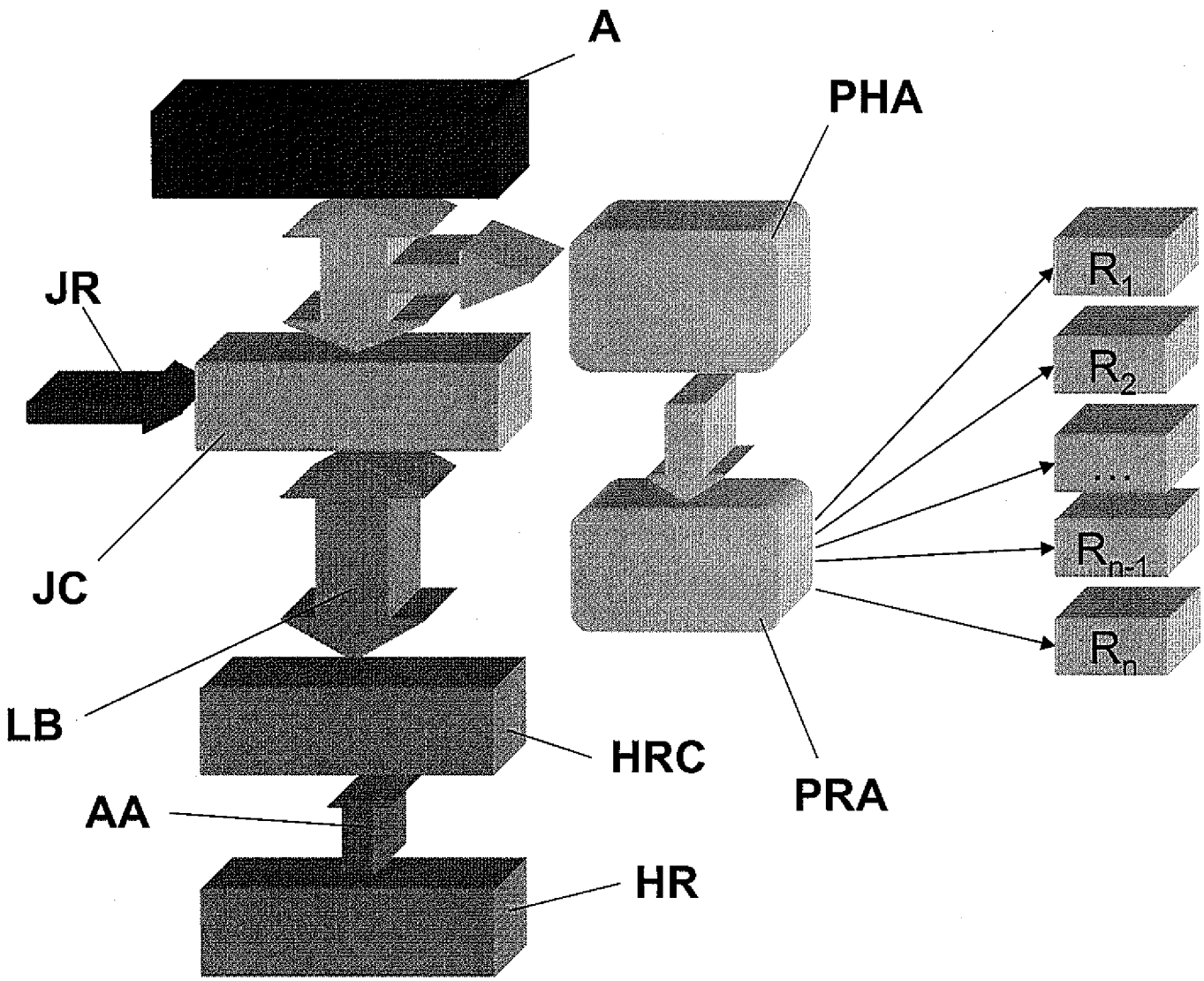


Fig. 3

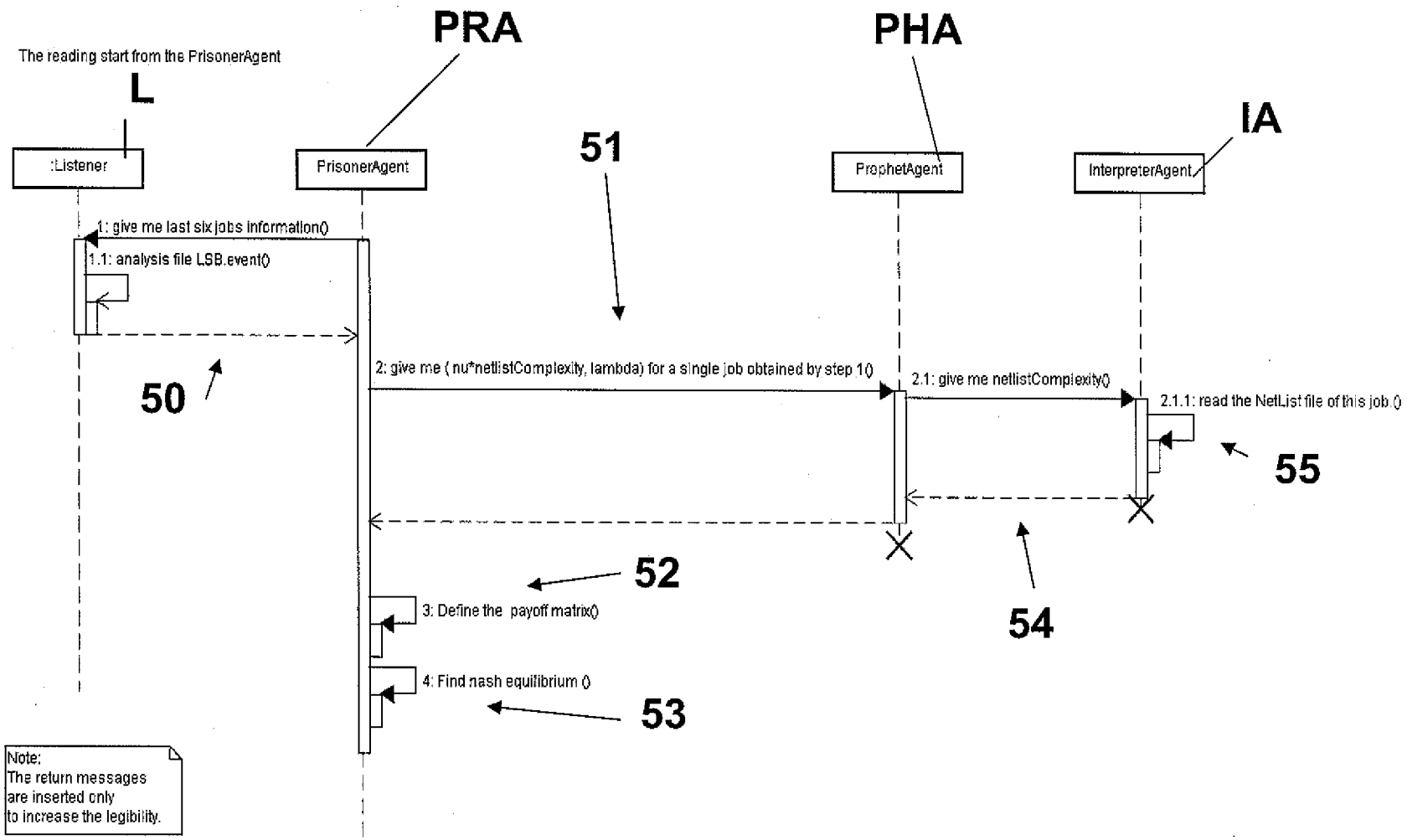

The reading start from the PrisonerAgent
L
PRA
PHA
IA
51
:Listener
PrisonerAgent
ProphetAgent
InterpreterAgent
1: give me last six jobs information()
1.1: analysis file LSB.event()
50
2: give me (nu*netlistComplexity, lambda) for a single job obtained by step 1()
2.1: give me netlistComplexity()
2.1.1: read the NetList file of this job ()
55
52
3: Define the payoff matrix()
54
4: Find nash equilibrium ()
53
Note:
The return messages
are inserted only
to increase the legibility.

Fig. 4