ITALIAN PATENT OFFICE

Document No.

102011901953870A1

Publication Date

20121213

Applicant

STMICROELECTRONICS S.R.L.

Title

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 (MB)
Inventori designati: Massimo Orazio SPATA
Depositata il: 13 giugno 2011

****

TESTO DELLA DESCRIZIONE

Campo tecnico

La descrizione si riferisce alle tecniche di allocazione selettiva (schedulazione) delle risorse di un sistema, ad esempio un sistema Multi-Agente, ed è stata messa a punto con particolare attenzione al possibile impiego nell'ambito di architetture Multi-Core.

Sfondo tecnologico

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.

Negli ultimi anni c'è stata una evoluzione dei sistemi di calcolo, con il passaggio dall'approccio mainframe all'approccio client/server distribuito per poi giungere alle reti di computer. Tale tendenza è legata alla crescente disponibilità di microprocessori veloci ed affidabili, prodotti in serie ed a basso costo.

Nei moderni sistemi di elaborazione sta diventando

comune l'impiego di reti di server di calcolo. Anche in sistemi sovra utilizzati si puo` rilevare tuttavia che, una percentuale significativa delle "macchine" o "server" incluse nella rete di calcolo, risulta inattiva o sottoutilizzata per gran parte del tempo.

Un migliore utilizzo dell'ambiente di calcolo distribuito può essere attuato tramite l'implementazione di algoritmi di allocazione selettiva o schedulazione (scheduling). Il nuovo paradigma di potenza di calcolo è chiamato "griglia" (Grid). Il fatto di realizzare un ambiente integrato di griglia di calcolo permette di dare origine a infrastrutture di calcolo con elevate potenzialità.

Dal punto di vista concettuale, una griglia è semplicemente un insieme di risorse di calcolo che svolgono lavori (job/thread) loro assegnati. La griglia è un sistema distribuito di tipo eterogeneo in termini di hardware e software (ad esempio Sistemi Operativi e applicazioni). La griglia 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 o lavori assegnati dagli utenti e li alloca selettivamente (ossia li "schedula") in vista dell'esecuzione su sistemi adatti, compresi nella griglia, sulla base di politiche di gestione delle risorse. Gli utenti possono quindi affidare alla griglia compiti 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:

- riduzione dei costi di hardware,

- bilanciamento del carico di lavoro fra i diversi server di calcolo (tramite un sistema di gestione dei carichi o Load Management System),

- capacità di gestire sistemi eterogenei,

- aumento di produttività, e

- minore esposizione all'obsolescenza dell'hardware.

Nei sistemi distribuiti, i compiti o lavori concorrono tra loro nell'accedere alle risorse condivise e distribuite.

Al riguardo, a livello concettuale, si potrebbe pensare alla possibilità che gli utenti della griglia mettano in atto un approccio di prenotazione manuale delle risorse della griglia. Questo approccio rimetterebbe la scelta e la stima delle risorse necessarie all'utente della griglia e questo implicherebbe inevitabilmente rischi di sopravvalutazione o sottovalutazione delle risorse a seguito dell'errore umano con conseguente spreco delle risorse della griglia.

Inoltre, gli utenti dovrebbero stimare la giusta quantità di risorse hardware da riservare per quella specifica applicazione (in termini di tempo di utilizzo della CPU e della memoria RAM); infine, l'utente dovrebbe sapere stimare il termine temporale del proprio lavoro [3].

La Teoria dei Giochi (Game Theory) è un ramo dell'economia e della matematica applicata che analizza le situazioni di conflitto e ne ricerca soluzioni competitive e cooperative tramite modelli in cui diversi giocatori scelgono le strategie per massimizzare i loro guadagni o profitti. Le decisioni di un soggetto possono influire sui risultati conseguibili da parte di un rivale secondo un meccanismo di retroazione. La mossa, o l'insieme delle mosse, che un soggetto intende fare viene chiamata

strategia.

Le applicazioni e le interazioni della teoria dei giochi sono molteplici: dal campo economico e finanziario a quello strategico-militare, dalla politica alla sociologia, dalla psicologia all'informatica, dalla biologia allo sport.

Ad esempio, nel documento US 2008/263557 A1 è descritto un dispositivo schedulatore per schedulare lo svolgimento di lavori tramite le risorse di una griglia computazionale; tale schedulatore è configurato per:

- identificare una soglia di equilibrio fra risorse e lavori;

- al di sotto della soglia di equilibrio, schedulare lo svolgimento dei lavori tramite le risorse della griglia computazionale secondo strategie ottimali di Pareto; e

- al disopra della soglia di equilibrio, schedulare lo svolgimento dei lavori tramite le risorse di detta griglia computazionale secondo strategie di equilibrio di Nash.

Documenti quali [2, 7] presentano algoritmi di schedulazione basati sull'Equilibrio di Nash destinati a raggiungere l'obiettivo di massimizzare la capacità produttiva complessiva in sistemi distribuiti. In alcune situazioni, come ad esempio nel caso del "Dilemma del Prigioniero", la soluzione di Equilibrio di Nash non è però la soluzione ottimale di Pareto [8, 9].

Scopo e sintesi

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 per ridurre l'inefficienza delle attuali soluzioni.

La presente invenzione si prefigge lo scopo di dare una risposta 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 un sistema Multi-Core che la comprende 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 varie forme di attuazione, la soluzione qui descritta può fondarsi su una procedura di schedulazione basata sulla teoria dei giochi e sull'Equilibrio di Nash applicati ai sistemi distribuiti come ad esempio una griglia computazionale o un sistema Multi-Core.

In varie forme di attuazione, basandosi sulla Teoria dei Giochi, è possibile attuare una strategia che fonde la soluzione MiniMax con le soluzioni di Equilibrio di Nash.

Uno schedulatore per una griglia può essere modellato sulla base di uno schedulatore da gioco, dove diversi lavori (task o job o thread) concorrono come giocatori per utilizzare il server di elaborazione. In varie forme di attuazione, ogni utente può disporre di un unico lavoro e può scegliere un singolo server di calcolo (tra gli n a disposizione) su cui eseguire il lavoro. In varie forme di attuazione, lo scopo di ciascun utente della griglia può essere quello di ridurre al minimo il suo tempo di completamento del lavoro, massimizzando la capacità produttiva (throughput) totale del sistema.

In varie forme di attuazione, è possibile perseguire l'obiettivo di mascherare l'entropia hardware e software implicita in una griglia attraverso l'impiego di un sistema Multi-Agente.

Così come considerati nell'ambito della presente descrizione, i sistemi Multi-Agente sono un insieme di Agenti (il termine Agente è qui utilizzato nella sua accezione corrente nel mondo delle reti e dei sistemi informatici) situati in uno specifico contesto e interagenti tra loro attraverso una adeguata organizzazione; un Agente è visto come un'entità che si caratterizza per essere almeno parzialmente autonoma, come un programma per computer, un robot, un essere umano, e così via.

Un sistema ad Agenti può essere progettato per realizzare un sistema di infrastrutture trasparente, al fine di automatizzare il processo di presentazione delle richieste e di massimizzare la capacità produttiva totale del sistema.

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.

Ad esempio, un sistema basato sugli Agenti è stato testato dalla Richiedente su un Cluster di Griglia, ed è stato utilizzato per simulare dispositivi microelettronici attraverso strumenti di automazione del disegno elettronico EDA (Electronic Design Automation) [1, 2, 7]; il contesto JADE (Java Agent Development Framework) è stato utilizzato in un middleware conforme alle specifiche FIPA (Foundation for Intelligent Physical Agents) e attraverso un insieme di strumenti grafici che supportano le fasi di sviluppo e di messa a punto (debugging) [4, 5].

Le principali entità previste nell'architettura del sistema sono:

- un database jobInfo, che è un database Mysql con le informazioni sui lavori,
- un Agente Interprete (Interpreter),
- un Agente MiniMax,
- un Agente di Nash,
- un Agente Schedulatore (Scheduler), e
- una procedura come qui considerata.

Breve descrizione delle figure

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

- la figura 1 mostra un sistema con un'architettura Multi-Core,

- la figura 2 illustra un diagramma di sequenza delle interazioni tra i vari Agenti, e

- la figura 3 illustra una specifica implementazione di una forma di attuazione.

Descrizione particolareggiata

Nella seguente descrizione sono illustrati vari dettagli specifici finalizzati ad un'approfondita comprensione delle forme di attuazione. Le forme di attuazione possono essere realizzate senza uno o più dei dettagli specifici, o con altri metodi, componenti materiali, etc. In altri casi, strutture, materiali o operazioni noti non sono mostrati o descritti in dettaglio per evitare di rendere oscuri i vari aspetti delle forme di attuazione.

Il riferimento ad "una forma di attuazione" nell'ambito di questa descrizione sta ad indicare che una particolare configurazione, struttura o caratteristica descritta in relazione alla forma di attuazione è compresa in almeno una forma di attuazione. Quindi, frasi come "in una forma di attuazione", eventualmente presenti in diversi luoghi di questa descrizione non sono necessariamente riferite alla stessa forma di attuazione. Inoltre, particolari conformazioni, strutture o caratteristiche possono essere combinate in ogni modo adeguato in una o più forme di attuazione.

I riferimenti qui utilizzati sono soltanto per comodità e non definiscono dunque l'ambito di tutela o la portata delle forme di attuazione.

A titolo di illustrazione dei principi alla base di varie forme di attuazione conviene ricordare che un gioco

strategico G con due giocatori [8] è definibile nella forma:

(X, Y, E, h)

dove:

- X, Y, E sono insiemi,
- X rappresenta le scelte disponibili per il giocatore I; idem per quanto riguarda l'insieme Y per il giocatore II,
- E rappresenta un insieme di possibili strategie per il gioco,
- $h: X \times Y \rightarrow E$; quindi h è una funzione di uscita che fornisce risultati ottenuti sulle scelte dei giocatori, e se il giocatore I sceglie x e il giocatore II sceglie y, restituisce il risultato $h(x, y)$.

In un gioco G serve conoscere le preferenze dei giocatori per i diversi elementi dell'insieme E.

Un modo veloce e semplice per descrivere queste preferenze è quello di utilizzare le funzioni di utilità $u(x)$ [6, 7, 8]. Ad esempio, per il giocatore I si assume che sia data una funzione u definita in E e con valori in $\mathbb{R}$, interpretando $u(e') \geq u(e'')$ come espressione del fatto che il giocatore I preferisce il risultato e' al risultato e''[7, 8].

Quindi, avendo un gioco (X, Y, E, H) e due funzioni di utilità (una per entrambi i giocatori) (u_1, u_2), questa è la forma di espressione del gioco:

(X, Y, E, H, u_1, u_2)

Attraverso l'operatore di composizione "$\circ$", si definiscono $f1 = u_1 \circ h$ e $f_2 = u_2 \circ h$, ottenendo un gioco strategico (X, Y, f_1, f_2) per due giocatori, come definito in [8], dove:

- X, Y sono insiemi delle scelte disponibili, e

• f_1, f_2: sono funzioni $X \times Y \rightarrow \mathbb{R}$.

Un Equilibrio di Nash per il gioco $G = (X, Y, f_1, f_2)$ è la coppia di valori (x^*, y^*) appartenete a $X \times Y$ dove:

• $f_1 (x^*, y^*) \geq f_1 (x, y^*)$ per ogni x che appartiene a X, e

• $f_2 (x^*, y^*) \geq f_2 (x^*, y)$ per ogni y che appartiene a Y (vedere sempre [8]).

Nella teoria dei giochi, il "Dilemma del Prigioniero" è un tipo di gioco non cooperativo che puo` essere espresso in forma estesa tramite albero delle decisioni, o in forma strategica tramite una rappresentazione matriciale, e dove ogni singolo giocatore (detto "Il prigioniero") ha come obiettivo minimizzare la propria pena o condanna (payoff) [6, 8].

In questo tipo di gioco l'effetto della relazione di preferenza $\geq_i$ (ovvero la scelta della strategia i-esima) è che i giocatori hanno una strategia ovvia da applicare, e questo è conseguente alla presenza di strategie dominanti. L'altro effetto del guadagno è che il risultato dell'interazione strategica non è efficiente per questo gioco [7, 8].

Se si suppone di applicare questo modello ad un esempio specifico e di avere m giocatori equivalenti a m lavori che vanno schedulati su t nodi di lavoro (Worker Node) WN:

• le possibili mosse sono equivalenti a tutte le possibili allocazioni dei lavori j_1, j_2, ..., j_m sui t nodi WN_1, WN_2, ..., WN_t,

• il numero totale di mosse disponibili per ogni Agente è dato da t^m,

• il guadagno in questo modello è equivalente a minimizzare il tempo di completamento del lavoro, ottimizzando il carico [1],

• l'insieme delle matrici del gioco (Game Matrix) $M = \{M_1, \ldots, M_p\}$ con i = 1, ..., p (con p pari al numero totale di matrici di gioco)

• una relazione di preferenza $>_1$ per massimizzare la capacità produttiva: $d>c>b>a$ in cui a, b, c, d sono parametri che rappresentano la lunghezza media esponenziale della coda di esecuzione nella CPU al primo minuto.

In questa forma di attuazione di rete a griglia computazionale, i giocatori possono corrispondere ai lavori o vice-versa. L'insieme di strategie per ogni giocatore è dunque determinato dall'insieme dei carichi iniziali per ogni server di calcolo. Data una strategia per ogni giocatore, il carico complessivo su ogni macchina è dato dalla somma dei tempi di completamento dei lavori che scelgono quella particolare macchina.

In un contesto così come qui considerato a titolo di esempio, ogni giocatore può cercare di minimizzare il tempo totale di completamento del lavoro su una macchina scelta; così, la funzione obiettivo standard può essere quella di ridurre al minimo il tempo di completamento del lavoro su un nodo di lavoro *WN* (e questo obiettivo si chiama anche minimizzazione di tipo makespan).

Ad esempio, dato un gioco con due nodi WN_1 e WN_2 e due lavori j_1 e j_2 si può definire una matrice M in cui le righe rappresentano le strategie che può scegliere il lavoro j_1 e le colonne rappresentano le strategie che può scegliere il lavoro j_2. La tabella che segue è una possibile rappresentazione di una matrice M di gioco in forma strategica per il problema "Dilemma del Prigioniero"

con due Agenti (lavori) j_1 e j_2 e due nodi (WN_1 e WN_2) e dove nel caso del problema del "Dilemma del Prigioniero" si ha $d > c > b > a$.

M:

		j_2	
		WN_1	WN_2
j_1	WN_1	(c, c)	(a, d)
	WN_2	(d, a)	(b, b)

Nel caso della precedente matrice di gioco, l'Equilibrio di Nash può essere calcolato come segue:

- si fissa un payoff per l'Agente j_1, poi il secondo Agente j_2 si sposta sulle righe per verificare se esiste una strategia migliore, viceversa si fissa un payoff per l'Agente j_2, poi il primo Agente j_1 si sposta sulle colonne [2, 7].

In varie forme di attuazione, la procedura per il calcolo dell'Equilibrio di Nash può essere schematizzata come segue.

Per ogni matrice M,

- per ogni vettore di payoff,

- confronta, la seconda componente di ogni vettore con tutte le altre componenti per linea,

- se non è il valore minimo, allora non è un Equilibrio di Nash,

- altrimenti se è il minimo, prendere la prima componente del vettore e confrontare a tutte le altre per colonna,

- se anche questa è il minimo, restituire questa soluzione come soluzione dell'Equilibrio di Nash per il problema del Dilemma del Prigioniero.

In altre parole, nel caso testé considerato a titolo di esempio, ogni Agente "Prigioniero" farà supposizioni circa le strategie di altri Agenti e sceglierà la strategia migliore per se stesso (ovvero quella con il valore più alto di guadagno), assicurandosi che ogni altro Agente non abbia altre strategie con un guadagno più alto, si sposterà su tutte le matrici e seguirà le soluzioni di Equilibrio di Nash per il problema del Dilemma del Prigioniero [8, 9].

Nella Teoria dei Giochi il MiniMax [10], è una strategia per minimizzare la massima perdita possibile, o, in alternativa, per massimizzare il guadagno minimo (MaxiMin). Nei giochi a somma zero, la soluzione MiniMax è la stessa dell'Equilibrio di Nash.

Nel caso considerato a titolo di esempio, ovvero il Dilemma del Prigioniero, si è in presenza di un gioco non a somma zero, quindi la soluzione MiniMax non equivale necessariamente alla soluzione Equilibrio di Nash.

Di seguito viene descritta una possibile forma di implementazione di una procedura MiniMax.

Per ogni matrice *M*:

- per ogni riga della matrice selezionare i valori massimi di ogni prima componente,
- tra tutti i valori massimi selezionare il valore minimo e restituire il numero di riga per quel valore,
- per ogni colonna della matrice selezionare i valori massimi di ogni seconda componente,
- tra tutti i valori massimi selezionare il valore minimo e restituire il numero di colonna per quel valore, e
- il numero di riga e di colonna definisce il vettore di payoff che è la soluzione MiniMax per quella matrice *M* per il problema del Dilemma del Prigioniero.

Di seguito viene descritta una possibile forma di implementazione di una procedura che integra e fonde la soluzione MiniMax con le soluzioni di Equilibrio di Nash.

Per ogni matrice M:

- calcolare ogni Equilibrio di Nash e selezionare le soluzioni attraverso un Agente Nash,
    - se la soluzione di Equilibrio di Nash non esiste allora restituire la soluzione MiniMax per la matrice gioco iniziale,
- calcolare ogni soluzione MiniMax e selezionare le soluzioni attraverso un Agente MiniMax,
- giocare un nuovo gioco tra tutti gli Agenti selezionati Nash e MiniMax,
- per ogni Agente Nash confrontare la sua soluzione con una soluzione di un Agente MiniMax,
    - se i parametri del vettore dell'Agente Nash corrispondono ai parametri del vettore dell'Agente MiniMax e non ci sono altre soluzioni, restituire questa soluzione,
        - altrimenti per ogni soluzione Nash, e per ogni soluzione MiniMax iterare il problema del Dilemma del Prigioniero, costruire una nuova matrice 2x2 con la soluzione Nash nella prima linea e la soluzione MiniMax nella seconda riga (entrambe le soluzioni fornite dai rispettivi Agenti),
            - applicare la soluzione di Equilibrio Nash alla nuova Matrice e restituire questa soluzione,
- ottimizzare la soluzione ottenuta, giocando un nuovo gioco di controllo tra la soluzione restituita e la soluzione MaxiMin.

Verrà ora descritta, con riferimento alla figura 1, una forma di attuazione di architettura generale di tipo

Multi-Core.

Una forma di attuazione di un'architettura Multi-Core può essere vista come una griglia computazionale in cui ogni nodo Core è assimilabile ad una macchina che può svolgere un diverso lavoro.

Dal punto di vista hardware, un acceleratore Multi-Core può presentarsi come una unità di I/O, che comunica con la CPU tramite comandi di I/O ed effettua trasferimenti diretti da/verso la memoria.

Dal punto di vista software, l'acceleratore Multi-Core può essere un altro computer a cui la procedura qui considerata può inviare dati e lavori da eseguire.

L'esempio di architettura Multi-Core di figura 1 ha un proprio modulo di memoria 5, chiamata memoria del dispositivo. Come per le CPU, il tempo di accesso alla memoria può essere piuttosto lento. Le CPU possono utilizzare memorie cache (memorie locali) per cercare di ridurre l'effetto della latenza della memoria, mettendo nelle cache i dati a cui di recente si è fatto accesso, nella speranza che gli accessi futuri cadano nella cache.

L'esempio di architettura Multi-Core qui considerato comprende inoltre un modulo processore Host indicato con il riferimento 1 e un modulo di memoria Host indicato con il riferimento 2. La memoria Host 2 è qui esemplificata come in comunicazione diretta (DMA) con il modulo di memoria 5. Il meccanismo di accesso diretto alla memoria DMA (Direct Memory Access) può permettere ad alcuni sottosistemi hardware di un computer (ad esempio periferiche) di accedere direttamente alla memoria di sistema per scambiarsi dati, oppure leggere o scrivere, senza chiamare in causa la CPU per ogni byte trasferito tramite il meccanismo usuale dell'interrupt e la successiva richiesta

di operazione desiderata, ma generando un singolo interrupt per blocco di dati trasferito.

In una forma di realizzazione, i lavori vengono eseguiti in modo sincrono o asincrono sugli m Core (indicati nel complesso dal riferimento 6) e la procedura qui considerata viene utilizzata per decidere come programmare i lavori su di essi sulla base di un gioco di schedulazione dei lavori. Il relativo software può quindi essere un software ad Agenti, con gli Agenti che sono i giocatori del gioco di schedulazione dei lavori.

Il blocco di schedulazione indicato con il riferimento 3 può essere essenzialmente un blocco su cui viene eseguita la procedura di schedulazione descritta in precedenza, che consente di ottimizzare la capacità produttiva totale del sistema sulla base dei lavori da eseguire. Questo modulo può permettere l'accesso concorrente alle risorse condivise generali, come i singoli blocchi Core 1, Core 2, ..., Core m (indicati in figura con i riferimenti 6a, 6b,.. 6m).

Il blocco schedulatore 3 può interagire con un blocco di controllo dei lavori 4, trasmettendo informazioni sull'allocazione dei lavori e sulla distribuzione di questi ai vari blocchi Core 6a, 6b,.. 6m.

In varie forme di attuazione l'architettura Multi-Core complessiva può comportarsi come un processore parallelo che ha su di esso molti moduli Core, con la procedura che attua un modello di programmazione parallela e il parallelismo dei dati. Questa architettura hardware può permettere di avere migliaia di lavori in esecuzione contemporaneamente, in contrasto con il caso generale di una singola CPU in cui si hanno soltanto pochi lavori in esecuzione allo stesso tempo, e questo aumenta notevolmente la portata di calcolo del sistema.

Prima di definire il modello ad Agenti (la cui definizione è stata già introdotta all'inizio di questo documento) usato per implementare il sistema Multi-Core, descritto con un Diagramma di Sequenza, può essere utile introdurre una funzione obiettivo (Goal) utilizzata nella descrizione che segue.

Funzione Obiettivo:

• ridurre al minimo il tempo totale di completamento lavori, massimizzando la capacità produttiva totale del sistema (throughput) [1, 2, 7],

• inviare e ricevere informazioni riguardanti i lavori richiesti,

• calcolare la complessità del lavoro da eseguire, analizzando i suoi valori di input precedenti,

• calcolare: carico iniziale, tempi di servizio λ dei lavori, e tempi di inter-arrivo μ dei lavori [1],

• creare la matrice dei payoff [1, 2],

• trovare l'Equilibrio di Nash [2, 8, 9],

• trovare la soluzione MiniMax,

• confrontare le soluzioni di Nash e MiniMax, e

• schedulare i lavori sul core C selezionato.

Obiettivo di Qualità:

• inviare valori calcolati ad un Agente di schedulazione.

La figura 2 mostra un esempio di possibili forme di attuazione di un'architettura ottenuto con la tecnica di modellazione ad Agenti con Diagrammi di Sequenza descritti attraverso il linguaggio UML.

L'esempio di architettura qui considerato può comprendere un blocco Agente Schedulatore 10, un blocco Agente Nash 11, un blocco Agente MiniMax 12 e un blocco Agente Interprete 13.

L'Agente Nash 11 è un Agente che può essere istanziato per ogni lavoro proposto ed incaricato di analizzare tutte le possibili soluzioni di Equilibrio di Nash. Il suo comportamento può essere definito dalla classe NashBehaviour. L'obiettivo dell'Agente Nash 11 è quello di analizzare la matrice dei payoff per i lavori proposti. L'Agente Nash 11 può invocare l'inizio del metodo della classe FindNashEquilibrium al fine di trovare la soluzione di Equilibrio di Nash per un gioco in forma strategica come ad esempio il problema del "Dilemma del Prigioniero".

L'Agente MiniMax 12 può essere un Agente che viene istanziato per ogni lavoro proposto. Il comportamento di questo Agente può essere definito dalla classe MiniMaxBehavior. L'obiettivo dell'Agente MiniMax 12 può essere quello di restituire la soluzione MiniMax per la matrice M.

L'obiettivo dell'Agente Interprete 13 può essere quello di analizzare i dati in ingresso per ogni lavoro di simulazione e di inviare in risposta un valore all'Agente Nash che lo ha richiesto come dato in ingresso. Il comportamento dell'Agente può essere definito dalla classe InterpreterBehaviour.

L'obiettivo dell'Agente di Schedulazione 10 può essere quello di schedulare i lavori sui nodi WN più appropriati, ovvero sui Core adeguati, seguendo le corrispondenze segnalate dai valori della soluzione ricavata tramite la procedura qui considerata. Il comportamento dell'Agente può essere definito dalla classe SchedulerBehaviour.

Per ogni lavoro viene istanziato un nuovo Agente, che può essere visto come un giocatore che partecipa attivamente al gioco di schedulazione dei lavori.

Con riferimento all'esempio considerato nella figura 2, in un passo 20 l'Agente di Schedulazione 10 richiede l'assegnazione di un nodo di lavoro WN (o di un Core) all'Agente Nash 11. In un passo 21 l'Agente Nash 11 trova la soluzione Equilibrio di Nash, mentre in un passo 22 l'Agente MiniMax 12 trova la soluzione MiniMax. L'Agente Interprete 13 analizza l'input riferito a quel determinato lavoro e restituisce all'Agente MiniMax 12 i parametri di payoff in un passo 24. L'Agente MiniMax 12 restituisce in un passo 25 la soluzione MiniMax all'Agente Nash 11, e in un passo 26 l'Agente Nash 11 restituisce all'Agente schedulatore 10 la soluzione Nash. L'Agente schedulatore 10 confronta in un passo 27 le soluzioni Nash e MiniMax e in un passo 28 schedula i lavori sui Core selezionati.

In un sistema distribuito come un sistema Multi-Core, uno dei più importanti obiettivi può essere quello di ottimizzare il bilanciamento del carico e la capacità produttiva o throughput totale del sistema, attraverso uno schedulatore.

In generale, trovare una soluzione ai problemi di schedulazione in sistemi multiprocessore è un problema np-completo (tempo polinomiale non deterministico np e np-hard), ossia può non essere possibile trovare una soluzione efficiente in modo deterministico in un tempo limite polinomiale, raggiungibile invece con un modello euristico entro un tempo accettabile.

Per questo motivo, in varie forme di attuazione è possibile modellare lo schedulatore come un gioco di schedulazione dei lavori in cui più lavori concorrono per utilizzare più nodi di lavoro o Core, come giocatori di questo gioco. Così, per ogni lavoro è possibile scegliere un singolo Core di elaborazione per processare il lavoro.

Nella teoria dei giochi la soluzione di Equilibrio di Nash può essere inefficiente o può non esistere, o del resto la soluzione dell'Equilibrio di Nash in un gioco strategico potrebbe essere non unica.

Pertanto, al fine di risolvere questo problema di gioco, in varie forme di attuazione è possibile integrare le soluzioni di Equilibrio di Nash con le soluzioni MiniMax e MaxiMin, tramite un Agente di sistema in una nuova procedura; esso può aumentare l'efficienza dell'Equilibrio di Nash, iterando il gioco strategico. Così, in varie forme di attuazione è possibile utilizzare Agenti Mobili al fine di ottimizzare la distribuzione del carico di lavoro in una architettura Multi-Core. Per Sistemi ad Agenti Mobili intendiamo un insieme di entità software autonome, situate in un certo ambiente ed interagenti tra loro mediante un'opportuna organizzazione [4, 5].

A titolo di esempio si può pensare di applicare la procedura qui considerata ad un esempio specifico, supponendo di avere n giocatori equivalenti a n lavori t che devono essere schedulati su m nodi Core C. In questo caso considerato come esempio:

- le possibili mosse sono equivalenti a tutte le possibili assegnazioni dei lavori t_1, t_2, ..., t_n sui nodi Core C_1, C_2, ..., C_m,
- il numero totale di mosse disponibili per ogni Agente è dato da m^n,
- il payoff è equivalente a minimizzare il tempo di completamento del lavoro, ottimizzando il carico totale del sistema,
- l'insieme delle matrici di gioco è dato da $M = \{M_1, ..., M_p\}$ con i = 1, ..., p (con p pari al numero totale di matrici di gioco),

• una relazione di preferenza $>_i$ per ridurre al minimo il tempo di completamento di un lavoro: dove *a*, *b*, *c*, *d* sono parametri che rappresentano al primo minuto la lunghezza media esponenziale della coda di esecuzione dei lavori sul nodo Core.

Nell'esempio qui considerato, i giocatori corrispondono ai lavori e l'insieme di strategie di ogni giocatore è il carico iniziale di ogni nodo Core.

Data una strategia per ogni giocatore, il carico complessivo su ogni nodo Core è la somma dei tempi di elaborazione di ciascun lavoro che sceglie un determinato nodo Core. Ogni giocatore può cercare di minimizzare il tempo totale di completamento del lavoro sul nodo Core scelto. Così, la funzione standard obiettivo può essere quella di ridurre al minimo il tempo di completamento del lavoro su un Core selezionato usando una minimizzazione di tipo makespan.

Dato un gioco con due nodi Core C_1 e C_2 e due lavori t_1 e t_2, in varie forme di attuazione è possibile definire una matrice del gioco *M*, dove le righe rappresentano le strategie che il lavoro t_1 può scegliere e le colonne rappresentano le strategie che il lavoro t_2 può scegliere.

La tabella che segue è un esempio di semplice matrice del gioco *M*, per un gioco espresso in forma strategica come ad esempio il problema Dilemma del Prigioniero, con due Agenti (lavori t_1 e t_2) e due nodi Core (C_1 e C_2):

M:

		t_2	
t_1		C_1	C_2
	C_1	(c, c)	(a, d)
	C_2	(d, a)	(b, b)

dove, nel caso del Dilemma del Prigioniero ad esempio si ha che d>c>b>a.

Nel caso della suddetta matrice del gioco *M*, l'Equilibrio di Nash può essere calcolato come segue: payoff fisso per l'Agente t_1 poi il secondo Agente t_2 si sposta sulle righe per verificare se esiste una strategia migliore, viceversa payoff fisso per l'Agente t_2 quindi il primo Agente t_1 si sposta sulle colonne.

In altre parole, secondo tale esempio, ogni Agente Nash può fare supposizioni sulle strategie degli altri Agenti e fare la scelta migliore (ovvero quella con il valore più alto di guadagno) per lui, affinché ogni altro Agente non abbia altre strategie con guadagno più alto, spostandosi su tutte le matrici e seguendo le soluzioni di Equilibrio di Nash per il problema del Dilemma del Prigioniero.

Nell'esempio qui considerato, per ogni matrice del gioco *M* l'Agente MiniMax calcolerà la soluzione MiniMax, come descritto nel seguito:

- Per ogni matrice *M*

• Per ogni riga della matrice ottenere i valori massimi di ogni prima componente.

• Tra tutti i valori massimi ottenere il valore minimo e restituire il numero di riga per quel valore.

• Per ogni colonna della matrice ottenere i valori massimi di ogni seconda componente.

• Tra tutti i valori massimi ottenere il valore minimo e restituire il numero di colonna per quel valore

Il numero di riga e di colonna definiscono il vettore di payoff che è la soluzione MiniMax del problema del Dilemma del Prigioniero per quella matrice *M*.

L'Agente Nash e l'Agente MiniMax possono essere utilizzati dalla procedura qui considerata per giocare il gioco di schedulazione dei lavori e decidere come distribuire i lavori sui Core come illustrato nella Figura 3.

Nell'esempio qui considerato, il blocco di controllo dei lavori 4 comunica con un blocco 7 destinato a inviare le richieste di lavoro agli Agenti Nash 11, MiniMax 12 e Interprete 13; gli Agenti hanno a disposizione la matrice del gioco *M* e in un passo 30 l'Agente Interprete 13 analizza i lavori e i dati in ingresso per creare e restituire in uscita un vettore di parametri.

A partire dalla matrice *M* è possibile calcolare una prima soluzione 40 (tramite l'Agente Nash) e una seconda soluzione 42 (tramite l'Agente MiniMax) che vengono inviate ai due blocchi 11 (Agente Nash) e 12 (Agente MiniMax). In un passo 34 è possibile calcolare la soluzione secondo la procedura qui considerata e trovata la soluzione finale ottimizzando la soluzione MiniMax.

Nell'esempio qui considerato, nel passo 34 viene eventualmente giocato un nuovo gioco tra i due Agenti 11 e 12. Infine in un passo 32 i lavori vengono schedulati sui nodi Core suggeriti dalla soluzione calcolata nel passo 34.

Al fine di testare ulteriormente la possibile applicazione di forme di attuazione, la Richiedente ha condotto prove riferite, quale esempio di rete Smart Grid, ad una rete intelligente per la distribuzione di elettricità; una tale rete Smart Grid dispone di strumenti di monitoraggio intelligente per tenere traccia di tutto il flusso elettrico del sistema, così come di strumenti per integrare le energie rinnovabili nella rete.

La visione dello scenario del futuro energetico delle

Smart Grids, ci permette di immaginare che un giorno le nostre case saranno collegate a una rete intelligente che gestisce l'energia elettrica con la massima efficienza e il minimo costo, con fonti di energia pulite e rinnovabili come il vento e il sole. Gli utenti potranno eventualmente contribuire a generare la propria energia attraverso pannelli solari o turbine eoliche di piccola dimensione ed utilizzare l'energia immagazzinata per ricaricare ad esempio le batterie di auto elettriche.

Una tale impostazione può essere soggetta ad alcuni evidenti limiti. Per esempio, mentre in teoria ha senso che un utente possa decidere di caricare la sua auto elettrica al di fuori delle ore di punta, ovvero quando i costi energetici sono più bassi, in pratica, se tutti agissero secondo questa strategia e prendessero la stessa decisione, allora il risultato sarebbe un picco della domanda energetica al di fuori delle ore di punta.

D'altra parte, se tutti gli utenti agissero seguendo la stessa strategia e cominciassero a mettere a disposizione l'energia nella rete Smart Grid durante i periodi di picco della domanda, il risultato potrebbe essere un maggiore guadagno iniziale, che sarebbe trasformato in un surplus di fornitura e in un basso livello della domanda, e di conseguenza un minore guadagno e un aumento dei prezzi dell'elettricità.

È quindi necessario trovare un equilibrio tra domanda e offerta tra tutti gli utenti o giocatori delle reti Smart Grid, utilizzando una visione di giochi non "cooperativi".

La soluzione potrebbe essere quella di trovare una soluzione per bilanciare il gioco applicando la teoria dell'Equilibrio di Nash. Tuttavia, così come già si è visto all'inizio della presente descrizione, l'Equilibrio

di Nash in alcuni casi può essere inefficiente (o sub-ottimale) può non essere unico o addirittura può essere inesistente.

In una possibile forma di attuazione una rete Smart Grid può essere intesa come un sistema di potenza elettrica che può prevedere milioni di sensori collegati tra loro da un sistema avanzato di comunicazione e di acquisizione dati. Questo sistema fornisce una analisi in tempo reale attraverso un sistema di calcolo distribuito che permette una risposta predittiva, piuttosto che reattiva, agli ingressi di tale sistema.

Quindi, per quanto qui interessa, si può immaginare l'impianto elettrico come un mercato energetico in cui tutti i giocatori sono gli utenti, o Agenti del sistema, che giocano per l'acquisto o la vendita di energia.

La soluzione qui esemplificata prevede di applicare su un sistema Multi-Agente la procedura qui considerata per il gioco in forma strategica con il cosiddetto Folk Theorem.

Come noto, secondo questo teorema della teoria dei giochi, se sono soddisfatte le condizioni di MiniMax, nel caso di giochi ripetuti all'infinito, è possibile riscontrare un legame fra ciò che accade in un certo periodo e che cosa può accadere in futuro, dimostrando che un comportamento non ottimale a breve termine può diventare tale a lungo termine, e pertanto si possono ottenere dei payoff di equilibrio che sono efficienti.

In questa prospettiva, considerando la capacità dei concorrenti di "punire" un comportamento deviante esercitato in passato, è estremamente probabile che vi sia un numero infinito di soluzioni.

Le prove condotte dalla Richiedente hanno dimostrato che la soluzione qui proposta è in grado di calcolare

dinamicamente condizioni di carico Pareto ottimali aumentando la capacità totale (throughput) dei "lavori", con carico variabile e con diversi rapporti lavori/nodi CPU. Il conseguente risultato di applicazione di questa procedura su campioni di 250 matrici di parametri ha prodotto un miglioramento medio di efficienza dal 76% al 88%.

Il principale obiettivo raggiunto è il significativo calo di inefficienza rispetto alle soluzioni MiniMax e Equilibrio di Nash.

Questa soluzione può essere applicata in generale a tutti i sistemi Smart Grid. Utilizzando questa soluzione, in varie forme di attuazione è possibile prevedere il comportamento di un sistema generale distribuito, come una rete intelligente Smart Grid, dato che ogni Agente si comporta razionalmente e reagisce solo ad un segnale di prezzo, e costruire un sistema basato su strategie di Agente, che può ottenere ad esempio (sempre con riferimento all'esempio testé considerato) il miglior profilo di accumulo di energia dati i prezzi di mercato variabili.

Elenco dei riferimenti

[1] Massimo Orazio Spata, Giuseppe Pappalardo, Salvatore Rinaudo, Tonio Biondi: "Agent-Based Negotiation Techniques for a Grid: The Prophet Agents", e-Science 2006, 149.

[2] Massimo Orazio Spata: "A Nash-Equilibrium Based Algorithm for Scheduling Jobs on a Grid Cluster", WETICE 2007: 251-252.

[3] I. Foster, C. Kesselman, and S. Tuecke: "The Anatomy of the Grid: Enabling Scalable Virtual Organization", International Journal of High Performance

Computing Applications, 15(3):200-222, 2001.

[4] WWW. JADE a Java Agent Development Framework http://jade.tilab.com/

[5] Foundation for Intelligent Physical Agents, FIPA Agent Management Specification & Support for Mobility Specification, Geneva, Switzerland, October 2000. Available at: http://www.fipa.org

[6] Amjad Mehmood, Abdul Ghafoor, H. Farooq Ahmed, Zeeshan Iqbal: "Adaptive Transport Protocols in Multi Agent System", pp.720-725, Fifth International Conference on Information Technology: New Generations (itng 2008), 2008

[7] Massimo Orazio Spata, Salvatore Rinaudo: "A scheduling Algorithm based on Potential Game for a Cluster Grid" JCIT Journal of Convergence Information Technology, (2009)

[8] F. Patrone (Author) "Decisori razionali interagenti", pp. 5 - 12, 46, Una introduzione alla Teoria dei Giochi - Edizioni PLUS, Pisa, 2006 ISBN: 88-8492-350-6

[9] John Nash, "Equilibrium points in n-person games", 48-49, Proceedings of the National Academy of Sciences, 36, 1950

[10] Sun Yan, Li Cun-lin, "The Minimax Principle of Non-cooperative Games with Fuzzy Payoffs" International Conference of Information Science and Management Engineering (ISME), 2010 Publication Year: 2010 , Page(s): 390 - 394

Naturalmente, fermo restando il principio dell'invenzione, i particolari di realizzazione e le forme di attuazione potranno variare, anche in modo significativo, rispetto a quanto qui illustrato a puro titolo di esempio non limitativo, senza per questo uscire

dall'ambito dell'invenzione così come definito dalle rivendicazioni annesse.

RIVENDICAZIONI

1. Procedimento per schedulare (10) lo svolgimento di lavori con le risorse di un sistema Multi-Core (6), il procedimento comprendendo produrre un modello di sistema basato su agenti rappresentativi di detti lavori, detti agenti essendo suscettibili di implementare una strategia di schedulazione dello svolgimento dei lavori con le risorse del sistema Multi-Core (6) tramite rispettive mosse disponibili corrispondenti alla scelta di una determinata risorsa del sistema Multi-Core (6) per lo svolgimento di un determinato lavoro, in cui detti agenti sono configurati per implementare detta strategia come strategia che integra strategie MiniMax e di Equilibrio di Nash.

2. Procedimento secondo la rivendicazione 1, in cui detti agenti sono configurati per attuare strategie MiniMax e di Equilibrio di Nash in modo iterativo, identificando una soluzione ottimale derivante dall'iterazione.

3. Procedimento secondo la rivendicazione 1 o la rivendicazione 2, in cui detta strategia è implementata massimizzando la probabilità di ottimizzare il tempo di completamento di un determinato lavoro per uno di detti agenti.

4. Procedimento secondo una qualsiasi delle rivendicazioni 1 a 3, comprendente:

- identificare per ogni agente un insieme di strategie determinato dall'insieme dei carichi iniziali di ogni risorsa del sistema Multi-Core (6), e
- definire il carico complessivo su ogni risorsa come

somma dei tempi di lavorazione dei lavori che scelgono quella particolare risorsa.

5. Procedimento secondo una qualsiasi delle rivendicazioni 1 a 4, comprendente:

- ricercare nell'ambito di detto modello di sistema soluzioni di Equilibrio di Nash, selezionando le soluzioni ritrovate attraverso un Agente Nash (11),

- in assenza di soluzioni di Equilibrio di Nash restituire la soluzione MiniMax per la matrice gioco iniziale (*M*),

- ricercare nell'ambito di detto modello di sistema soluzioni MiniMax, selezionando le soluzioni ritrovate attraverso un Agente MiniMax (12),

- giocare un nuovo gioco tra tutti gli Agenti selezionati con le soluzioni Nash (11) e MiniMax (12) ritrovate,

- per ogni Agente Nash (11) confrontare la sua soluzione con una soluzione di un Agente MiniMax (12),

- se i parametri del vettore dell'Agente Nash (11) corrispondono ai parametri del vettore dell'Agente MiniMax (12) e non ci sono altre soluzioni, restituire questa soluzione,

- altrimenti per ogni soluzione Nash, e per ogni soluzione MiniMax iterare il problema del Dilemma del Prigioniero, costruire una nuova matrice 2x2 con la soluzione Nash nella prima linea e la soluzione MiniMax nella seconda riga,

- applicare la soluzione di Equilibrio Nash alla nuova matrice e restituire questa soluzione,

- ottimizzare la soluzione ottenuta, giocando un nuovo gioco di controllo tra la soluzione restituita e la soluzione MaxiMin.

6. Dispositivo schedulatore (3) configurato per attuare il procedimento secondo una qualsiasi delle rivendicazioni 1 a 5.

7. Griglia computazionale comprendente una pluralità di risorse (6) per lo svolgimento di lavori computazionali, detta griglia computazionale comprendendo uno schedulatore (3) secondo la rivendicazione 6.

8. 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 5.

CLAIMS

1. A method for scheduling (10) the performance of tasks using the resources of a Multi-Core system (6), the method including producing a system model based on agents representative of said tasks, said agents configured to implement a scheduling strategy of performing said tasks with the resources of the Multi-Core system (6) via respective available moves corresponding to the choice of a given resource of the Multi-Core system (6) to perform a given task, wherein said agents are configured to implement said strategy as an integrated MiniMax/Nash Equilibrium strategy.

2. The method of claim 1, wherein said agents are configured to implement in an iterative manner MiniMax and Nash Equilibrium strategies, by ideintifying an optimal solution deriving from iteration.

3. The method of claim 1 of claim 2, wherein said strategy is implemented by maximizing the probability of optimizing the time of completing a given task for one of said agents.

4. The method of any of claims 1 to 3, including:

- identifying for each agent a set of strategies determined by the set of the initial workload on each resource in the Multi-Core system (6), and
- defining the total load on each resource as the sum of the performance times of the tasks that select said resource.

5. The method of any of claims 1 to 4, including:

- searching Nash Equilibrium solutions within said system model, selecting the solutions found by means of a Nash Agent (11),

- in the absence of Nash Equilibrium solutions returning the MiniMax solution for the initial game matrix (*M*),

- searching MiniMax solutions within said system model, selecting the solutions found by means of a MiniMax Agent (12),

- playing a new game among all the Agents selected with the Nash (11) and MiniMax (12) solutions found,

- for each Nash Agent (11), comparing its respective solution with a solution of a MiniMax Agent (12),

- if the parameters of the Nash Agent (11) vector correspond to the parameters of the MiniMax Agent (12) vector, and no other solutions exist, returning this solution,

- otherwise, for each Nash solution and for each MiniMax solution, iterating the Prisoner's Dilemma problem, generating a new 2x2 matrix with the Nash solution in the first line and the MiniMax solution in the second line,

- applying the Nash Equilibrium solution to the new matrix and returning this solution,

- optimizing the solution obtained, by playing a new control game between the solution returned and the MaxiMin solution.

6. A scheduler device (3) configured to implement the method of any of claims 1 to 5.

7. A computing grid including a plurality of resources (6) to perform computing tasks, said computing grid including a scheduler (3) according to claim 6.

8. A computer program product, loadable in the memory of at least one computer and including software code portions configured to perform the method of any of claims 1 to 5.

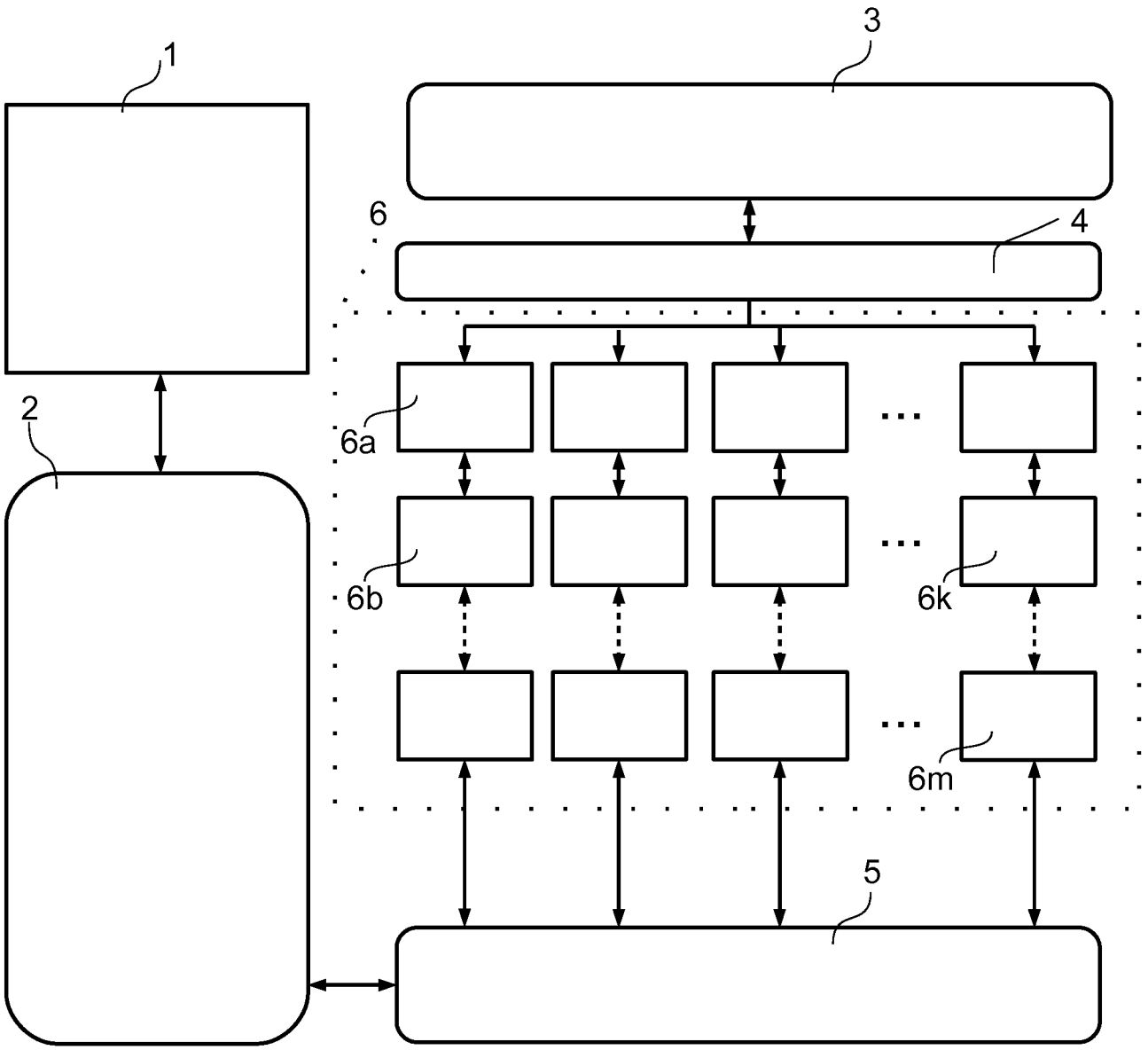

Fig. 1

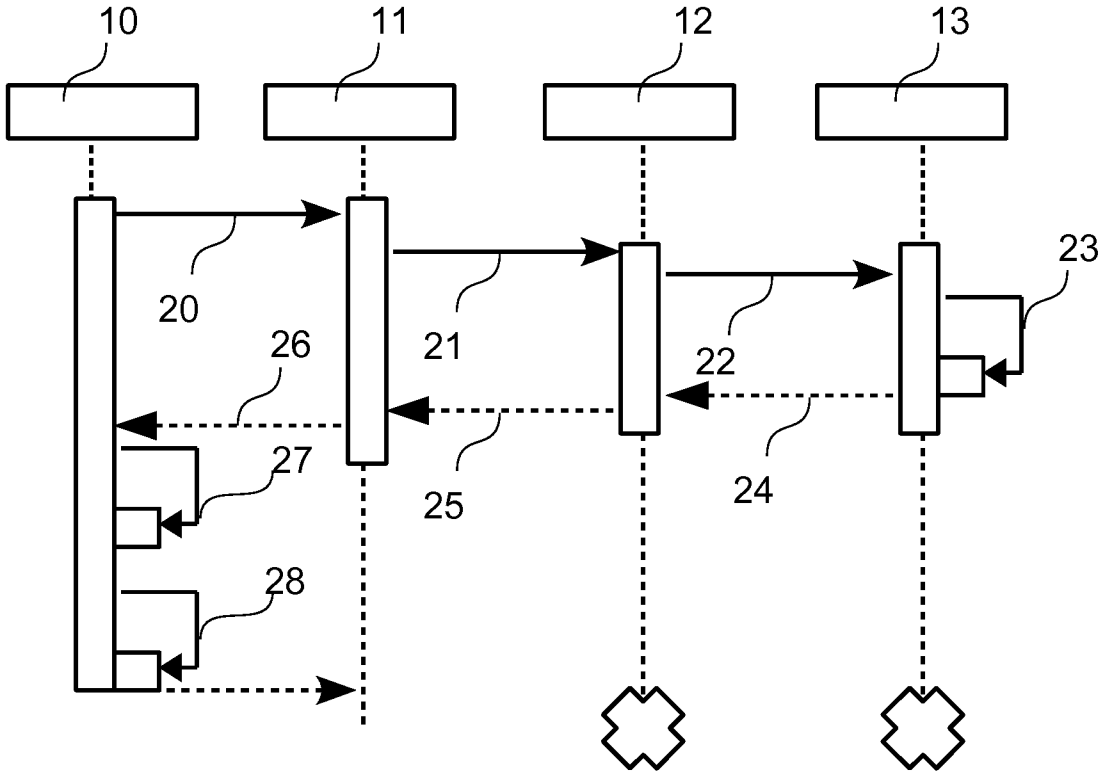

Fig. 2

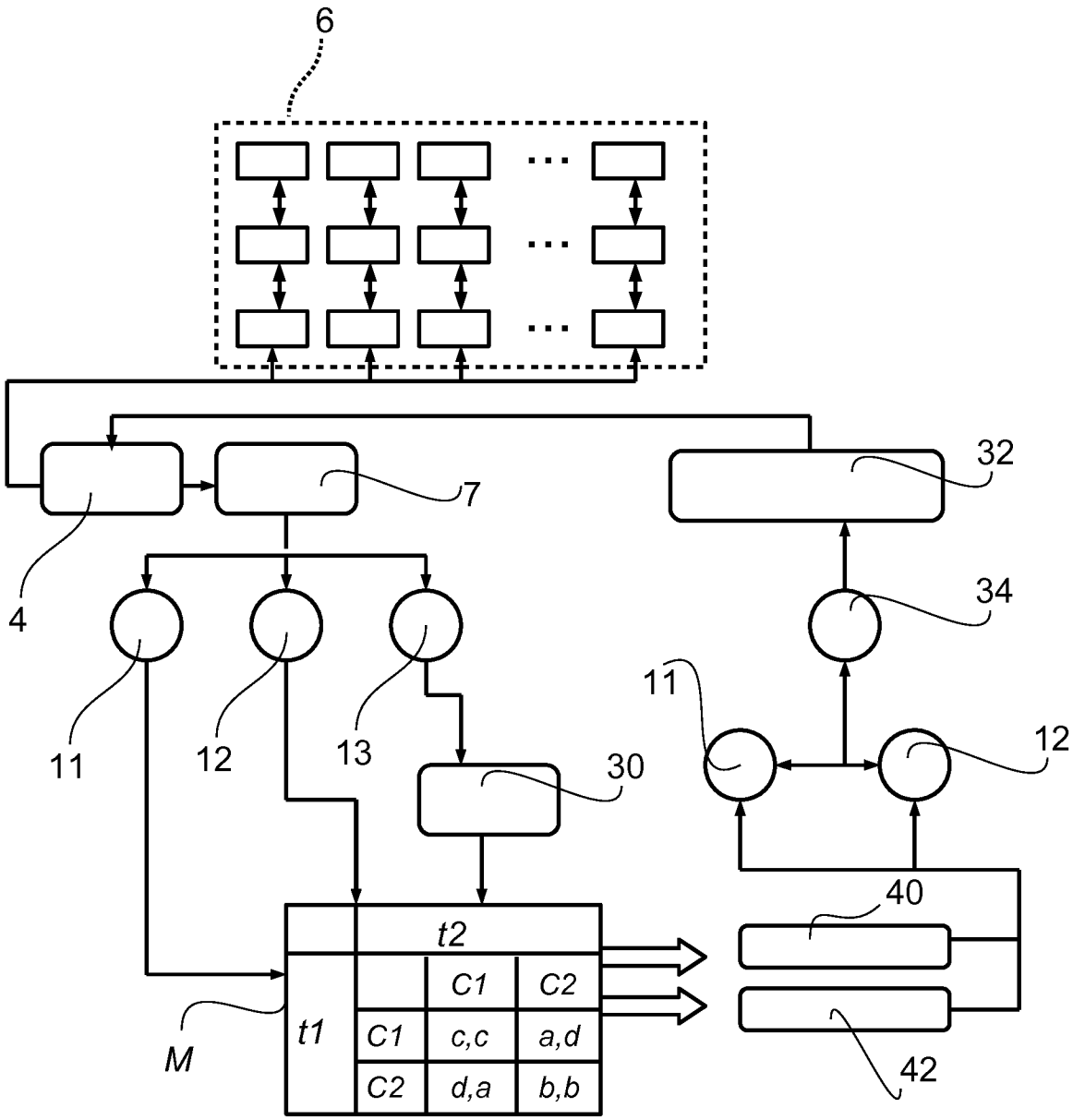

Fig. 3