PCD Module 2.5

Elementi di progettazione di programmi concorrenti

Passi nella progettazione

Analizzare il problema per identificare concorrenze sfruttabili: decomposizione task e dati e loro dipendenze

Task

- Sequenza di istruzioni che opera assieme come gruppo (parti di un algoritmo/programma)
- Specifico "pezzo" di lavoro che deve essere necessariamente portato a termine come responsabilità di un qualche agente (definizione più astratta/OO)

Scelta dell'architettura concorrente

- Mappatura task <-> agenti - incapsulano logica e controllo dell'esecuzione del task
- Risorse condivise che incapsulano il risultato del lavoro di un agente (entità passive/monitor)
- Coordinazione media-risorse per definire l'interazione tra agenti e gestire le dipendenze (entità passive/monitor)
- Protocolli di scambio messaggi

Analisi del problema

Decomposizione dei task

Problema decomposto in una collezione di task indipendenti o quasi, principio divide et impera. Task sufficientemente indipendenti cosicchè gestire le dipendenze necessiti solo di una piccola frazione di tutto il tempo di esecuzione del programma.

Es. sistemi producer/consumer

Decomposizione dei dati

Possono essere facilmente decomposti in unità che possono operare relativamente indipendentemente. Prime unità di dato identificato, quindi task relativi a una unità possono essere identificati --> Decomposizione dei task segue la decomposizione dei dati.

Scala bene con il numero di processori disponibili. Esempio: moltiplicazione di matrici.

Analisi delle dipendenze

Analisi dipendenze relative ai task, per **raggruppare e ordinare** i task relativi ai tipi di dipendenze e massimizzare le performance.

Tipi di dipendenze:

- **temporali**
- **data/risorse**

Livello di progettazione - alcune classi concettuali

Parallelismo di risultato

Parallelismo visto in termini di risultati del programma. Progettazione del sistema intorno alla struttura dei dati o alla risorsa resa come risultato finale.

- otteniamo il parallelismo computando contemporaneamente tutti gli elementi del risultato
- strettamente correlati al modello di decomposizione dei dati

Ogni agente è assegnato a produrre un pezzo del risultato: tutti lavorano in parallelo fino alla naturale limitazione imposta dal problema. Le strutture dati adatte sono progettate per avvolgere il risultato (dati) in costruzione, incapsulando problemi di mutex e sincronizzazione.

Esempio costruzione casa: singole parti costruite singolarmente in parallelo da diversi lavoratori.

Parallelismo specialistico

Parallelismo visto in termini di un insieme di specialisti collegati in una rete logica di qualche tipo

Il parallelismo risulta da tutti i nodi della rete logica (tutti gli specialisti) attivi contemporaneamente. Ogni agente è assegnato per eseguire un tipo specifico di attività. Tutti lavorano in parallelo fino alla naturale limitazione imposta dal problema

I mezzi di coordinamento (protocolli di comunicazione, strutture dati condivise) vengono utilizzate per supportare la comunicazione e il coordinamento degli specialisti (caselle di messaggi, lavagne, servizi per eventi, ecc.). L'approccio è opposto rispetto al parallelismo dei risultati.

Esempio casa: parallelismo visto come singole unità di lavoro (elettricisti, carpentieri, idraulici, ecc.)

Parallelismo "agenda"

Parallelismo visto come agenda di task del programma. Progettazione del sistema intorno ad un particolare ordine del compito, dopodichè si assegnano molti lavoratori ad ogni passo, i quali possono possibilmente svolgere diversi passaggi in parallelo (flusso di lavoro). Ogni agente è assegnato per aiutare con l'elemento corrente all'ordine del giorno: tutti lavorano in parallelo fino alla naturale limitazione imposta dal problema. Le strutture dati condivise sono progettate per gestire i dati consumati e prodotti dalle attività di agenda (esempi: buffer limitati)

Esempio casa: agenda dei lavori, con lavoratori assegnati ad ogni fase a comptii diversi (lavoratori generalisti).

Recap

- I confini tra le tre classi non sono rigidi, si possono anche mixare
- Nonostante ciò, le tre classi individuano tre modi diversi di lavorare (rispettivamente: forma del lavoro finito, suddivisione crew di lavoro, lista di task)
- L'approccio può essere applicato ricorsivamente seguendo decomposizione task+dati

Uso delle classi concettuali

Per scrivere un programma concorrente:

- scegliere la classe di concetto più naturale per il problema;
- scrivere un programma utilizzando il metodo più naturale per quella classe concettuale; e
- se il programma risultante non è accettabilmente efficiente, trasformarlo metodicamente in una versione più efficace passando da un metodo più naturale ad un più efficiente

Dalle classi concettuali alle architetture + pattern

Alcune architetture possono aiutare a colmare il divario tra classi concettuali e implementazione.

Master-Workers

L'architettura è composta da un agente principale e da un insieme probabilmente dinamico di agenti di lavoro, che interagiscono con un mezzo di coordinamento adeguato che funge da sacco di attività:

- **agente principale**: decompila l'attività globale in sottotask, assegna i sottotask inserendo la loro descrizione nella bag e raccoglie i risultati delle attività
- **agente di lavoro**: ottiene l'attività da fare dalla bag, eseguire i compiti e comunica i risultati (variante con strutture di monitoraggio adatte)
- **bag di task resources**: tipicamente implementato come una blackboard o un buffer limitato

Fork-Join

Applicando ricorsivamente l'architettura del master worker i lavoratori stessi sono padroni, producendo compiti e raccogliendo risultati

- approccio efficace per implementare strategie concorrenti di divide-et-conquer
- può essere realizzato sia utilizzando un sacco di attività o direttamente

Filter-Pipeline (pattern basati su flussi)

L'architettura è composta da una catena lineare di agenti che interagiscono con alcune risorse del buffer o delle risorse del canale o del buffer limitato

- **agente generatore**: l'agente che inizia la catena, generando i dati elaborati dalla pipeline
- **agente filtrante**: un agente intermedio della catena, che consuma informazioni d'ingresso da una pipe e produce informazioni in un altra pipe
- **agente sink**: l'agente che chiude la catena, raccogliendo i risultati

Esempio: image processing

Announcer-Listeners (coordinazione basata su eventi)

L'architettura è composta da un agente di annunciatore e da un insieme dinamico di agenti di ascolto, che interagiscono con un servizio di evento

- **agente annunciatore**: annuncia l'avvenimento di un evento sul servizio degli eventi
- **agenti ascoltatori** registrati sul servizio dell'evento in modo da essere informati del verificarsi di eventi interessanti per l'ascoltatore
- **event-service resource**: disattiva l'interazione degli annunciatori-ascoltatori e raccoglie e invia eventi