Università degli Studi Roma Tre
Facoltà di Scienze M.F.N.

Tecniche per il calcolo scientifico distribuito in Java

Sintesi della tesi di Laurea in Matematica
di Cristiana Vigliaroli

Relatore: Prof. Marco Liverani

a.a. 2000/2001 – Novembre 2001

Negli ultimi anni abbiamo assistito a delle grandi innovazioni per ciò che riguarda l'approccio automatico, con l'ausilio di un calcolatore, alla soluzione di problemi di carattere gestionale o di rilevanza computazionale e numerica. L'Informatica e l'industria che segue e sollecita le innovazioni di carattere teorico e le intuizioni innovative, ultimamente ci hanno abituato non soltanto ad assistere ad una rapidissima evoluzione delle capacità di calcolo delle macchine, ma anche alla nascita di nuovi modelli, di nuove architetture e di nuovi approcci alla risoluzione di problemi mediante un computer.

Accanto alla produzione di calcolatori sempre più veloci (la potenza di un normale Personal Computer oggi è forse superiore a quella di un *mainframe* degli anni settanta), sono state sviluppate nuove "tecnologie abilitanti" e nuove modalità operative. Per tecnologie abilitanti intendiamo tutti quegli strumenti tecnici che consentono di accedere a nuove possibilità nell'uso di una macchina e nella progettazione di nuovi servizi. Solo pochi anni fa era impensabile l'attuale diffusione di telefoni cellulari, mentre oggi troviamo abbastanza normale poter utilizzare la rete Internet da ogni parte del mondo. Anzi: quando ci troviamo a non poter utilizzare la rete, ci risulta difficile l'uso dello stesso computer. Senza la possibilità di condividere file ed informazioni con altri utenti della rete, sembra venir meno l'utilità di queste macchine.

Una delle innovazioni principali quindi è proprio quella costituita dalla diffusione di massa della rete Internet e dal potenziamento delle linee dati che collegano fra loro i nodi della rete. Oggi, grazie a queste tecnologie, possiamo utilizzare indifferentemente applicazioni grafiche che operano su una macchina di una università italiana, collegandoci da un terminale (di fatto un Personal Computer o un Macintosh) dalla stanza di un albergo negli Stati Uniti. Fino a pochi anni fa eravamo al massimo in grado di leggere la posta elettronica, e non senza qualche disagio!

In questo contesto si inserisce lo sviluppo di un nuovo linguaggio di programmazione, il linguaggio Java, progettato appositamente per trarre il massimo del beneficio dall'uso della rete (sia essa una "rete locale" o anche una "rete geografica" come Internet).

In una prima fase Java è stato interpretato da molti soltanto come un linguaggio mediante il quale si potessero arricchire di componenti interattive le pagine di un sito Web. L'uso che se ne è fatto è stato principalmente quello di realizzare le cosiddette *applet Java* che effettivamente permettevano di estendere le scarse possibilità offerte dal linguaggio HTML utilizzato per codificare le pagine Web.

Oggi Java è evoluto al punto da essere considerato uno dei principali linguaggi per lo sviluppo di applicazioni in ambito industriale: tutti i prodotti più diffusi (pensiamo ad esempio ai database) consentono la realizzazione di funzionalità aggiuntive mediante il linguaggio Java. In molti contesti le applicazioni scritte in Java stanno sostituendo vecchi programmi realizzati in Cobol, in Fortran ed anche in C.

Il successo di questo linguaggio è da ricercarsi nelle sue caratteristiche intrinseche che effettivamente semplificano di molto la realizzazione di funzionalità altrimenti estremamente complesse. Quali sono queste caratteristiche? Innanzi tutto il fatto che Java è un linguaggio ad oggetti: non è certo l'unico, nè il migliore, sotto questo punto di vista, però questa caratteristica di base già da sola basterebbe a farlo preferire ad altri linguaggi, visto che consente di ingegnerizzare in modo più elegante le applicazioni ed i programmi.

Ma non solo: Java fornisce diverse *classi* di oggetti per la realizzazione di procedure che sfruttano in pieno i servizi offerti dalla rete. Questo si traduce in una stretta integrazione con la tecnologia Web, ma soprattutto nella disponibilità di funzioni (che chiameremo *metodi* nella terminologia *object oriented* di Java) che permettono di utilizzare i *socket* del protocollo TCP/IP ed il protocollo RMI (*Remote Method Invocation*) per la progettazione di applicazioni *distribuite* di alto livello.

Una applicazione distribuita non è altro che un programma frammentato in due o più componenti distinte che possono essere eseguite su calcolatori differenti, ma collegati in rete fra loro e coordinati da una applicazione di livello superiore che stabilisce la distribuzione del carico di lavoro sulle diverse macchine. A differenza di quanto avviene nel caso delle applicazioni *parallele*, in questo caso ogni macchina esegue operazioni differenti, specializzandosi anzi, nell'esecuzione di procedure ben precise messe a disposizione degli altri calcolatori che contribuiscono all'esecuzione del "calcolo distribuito".

Questa tesi fornisce una descrizione del lavoro di progettazione ed implementazione di un modello di calcolo distribuito applicato alla risoluzione di un problema di carattere numerico classico. In particolare ci siamo occupati di verificare la praticabilità di una architettura di calcolo distribuito, realizzata in linguaggio Java sfruttando la tecnologia RMI, per la soluzione approssimata di un sistema di equazioni differenziali ordinarie (EDO) con i metodi di Eulero e di Runge Kutta. Sia il problema che i metodi utilizzati sono argomenti standard, ben consolidati e naturalmente privi di aspetti innovativi: è chiaro quindi che sono da considerarsi solo strumentali alla verifica di un modello di calcolo, quale appunto quello "distribuito", realizzato mediante il protocollo RMI.

La tesi si articola su tre capitoli: nel primo viene presentato il linguaggio Java e la metodologia di programmazione ad oggetti (OOP, *Object Oriented Programming*). Abbiamo approfondito i concetti di incapsulamento, ereditarietà e polimorfismo, tipici della programmazione ad oggetti. Viene inoltre proposta una sintesi delle caratteristiche fondamentali del linguaggio Java, presentando alcune distinzioni tra le tipologie di programmi che possono essere realizzati (es.: le *applet*, le *applications*, ecc.) e affrontando le problematiche che un approccio *object oriented* implicano nella soluzione di un problema mediante un calcolatore.

Java è un linguaggio progettato da Sun Microsystems partendo da zero, non fa riferimento a nessun altro linguaggio di programmazione, gli è stata data una struttura simile al C per permetterne un facile apprendimento. È un linguaggio nato per operare in ambiente di rete, viene data quindi particolare importanza agli aspetti relativi alla sicurezza, alla connessione mediante protocolli TCP/IP, alla portabilità su diverse piattaforme e alla possibilità di progettare applicazioni che operano in modalità distribuita, consentendo cioè la cooperazione tra componenti software distinte che operano su più calcolatori. Per quanto riguarda la "portabilità" del software, Java propone una architettura innovativa basata sul concetto di macchina virtuale e di bytecode: la *Java Virtual Machine* (JVM). Con questa architettura Sun propone un software in grado di simulare il comportamento di un calcolatore "astratto", ma con caratteristiche sempre uguali su più architetture hardware e software che permetta quindi l'esecuzione dello stesso programma su calcolatori differenti (un PC Windows, un Macintosh, ecc.). I programmi scritti in Java non vengono dunque eseguiti dal computer ospite, ma dalla JVM: il linguaggio *bytecode*

è il linguaggio macchina della Java Virtual Machine; per ottenere un programma codificato in bytecode è necessario compilare il programma sorgente con un compilatore Java. Naturalmente poi per riutilizzare poi lo stesso programma eseguibile in formato bytecode su macchine differenti non sarà necessario disporre ogni volta di un compilatore Java, ma soltanto della Java Virtual Machine compatibile con la macchina. Questa è una delle novità fondamentali della struttura Java: anche prima della nascita di Java era possibile compilare lo stesso programma su piattaforme diverse in quanto erano stati creati dei compilatori per quasi tutte le piattaforme; ciò comportava che, per far eseguire lo stesso programma su piattaforme diverse, bisognava ricompilarlo per produrre un codice in linguaggio macchina che fosse compatibile con la piattaforma. La ricompilazione del programma era possibile solo se il programma era stato scritto utilizzando un linguaggio base, senza poter sfruttare la potenza e le caratteristiche di una determinata piattaforma. Java è stato progettato per offrire buone prestazioni anche su sistemi con processori poco potenti; se da una parte è vero che Java è un linguaggio interpretato, dall'altra il bytecode Java, più vicino alla macchina di un linguaggio di alto livello, è stato progettato con cura per essere facilmente tradotto in linguaggio macchina dalle alte prestazioni.

Altre caratteristiche interessanti del linguaggio Java, sono le seguenti:

- *Abstract Windows Toolkit.* Java fornisce una astrazione dell'interfaccia utente a finestre, in modo tale da distaccarsi dal particolare modello di interfaccia utente proposto dal sistema operativo Microsoft Windows, o dal System della Apple; questo aspetto non è di secondaria importanza, visto che uno dei limiti principali alla portabilità del software è costituito proprio dalla impossibilità di ricondurre una particolare interfaccia utente ad un'altra.

- *Multi threading.* È la caratteristica di alcuni moderni linguaggi di programmazione e sistemi operativi, che consente di scindere il flusso di un programma in più "thread", ossia in sottoprogrammi parzialmente indipendenti, in grado di eseguire compiti differenti in parallelo.

- *Remote Method Invocation.* È la modalità con cui Java rende possibile la realizzazione di applicazioni distribuite su più macchine, che comunicano tra di loro (per lo scambio dei dati e la sincronizzazione dei processi) mediante il protocollo RMI.

È questo l'aspetto che per noi costituisce l'elemento di maggiore rilevanza: il protocollo RMI e la modalità con cui può essere progettata ed implementata una applicazione distribuita.

Il secondo capitolo della tesi affronta nello specifico la descrizione del protocollo RMI (*Remote Method Invocation*), l'aspetto del linguaggio Java di maggiore rilevanza per il nostro obiettivo di realizzare una architettura di calcolo distribuita. In questo capitolo infatti viene descritta la modalità con cui può essere progettata ed implementata una applicazione distribuita che sfrutti i metodi offerti da RMI. RMI consente di definire delle classi di oggetti che potranno essere istanziate su una macchina differente da quella che sta eseguendo il programma: in questo modo si viene a creare una relazione di tipo client/server tra un computer (che chiameremo *client*) che si occupa di eseguire il programma vero e proprio, eventualmente gestendo anche l'interfaccia utente, ed un altro computer (che chiameremo *server*) che avrà il compito di gestire degli oggetti e quindi di eseguire i metodi che operano su di essi. La relazione client/server non deve necessariamente essere "esclusiva": uno stesso server può fornire servizi a più di un client e, al tempo stesso, un client può utilizzare oggetti differenti messi a disposizione da più server. In maggiore dettaglio possiamo dire che l'architettura RMI richiede la presenza di tre componenti fondamentali:

- RMI Server: è l'applicazione Java che mette a disposizione degli oggetti e dei metodi remoti;
- RMI Registry: è un componente del server RMI che fornisce informazioni ai client relativamente ai metodi remoti pubblicati dal server ed alla modalità con cui tali metodi devono essere invocati;
- RMI client: è il programma che richiama ed esegue i metodi remoti disponibili sul server.

In fig. **??** è riportata una schematizzazione delle componenti dell'architettura RMI e dei flussi di comunicazione esistenti.

Figura 1: Architettura RMI

Il server è costituito, come abbiamo visto, da due componenti che operano parallelamente: RMI registry ed RMI server. Il primo è un componente che gira su ogni computer su cui è attivo un server RMI e si occupa di mantenere la lista degli oggetti che sono attivi su quel computer. Tramite questo componente, il client è in grado di ottenere, da ciascun computer che esegue un server RMI, l'elenco degli oggetti che sono disponibili su quella macchina.

Una delle caratteristiche peculiari di RMI è infatti la possibilità di scaricare dinamicamente il codice delle classi che non sono già installate sulla macchina chia-

mante, rendendo possibile l'estensione delle funzionalità dell'applicazione in modo dinamico. Un'altra caratteristica fondamentale di RMI è la possibilità di attivare gli oggetti remoti in funzione di una richiesta del client. L'attivazione remota comporta che, se un processo server non è in esecuzione, ma un client ne richiede i servizi, RMI è in grado di lanciare l'esecuzione del server in modo automatico.

Sul client operano le seguenti componenti:

1. il sistema operativo della macchina (Windows, Linux, Mac OS, ecc.);
2. la Java Virtual Machine relativa al sistema operativo della macchina;
3. il programma Java compilato in formato bytecode.

Su ogni server sono presenti le seguenti componenti software:

1. il sistema operativo della macchina (Windows, Linux, Mac OS, ecc.);
2. la Java Virtual Machine relativa al sistema operativo del server;
3. il programma RMI Registry;
4. il programma RMI Server, configurato per attendere connessioni da parte dei client su una determinata porta del protocollo di rete TCP/IP.

RMI Server esegue le classi Java che implementano gli oggetti resi pubblici attraverso RMI; al tempo stesso l'applicazione che opera sul client è stata compilata includendo anche una classe che espone l'interfaccia dei metodi remoti; in altre parole l'applicazione client deve conoscere a priori la modalità di chiamata dei metodi remoti.

Il flusso di comunicazione tra il client ed il server RMI prevede i seguenti passi, che sono resi del tutto trasparenti al programmatore: egli non dovrà occuparsi di eseguirli perché è la stessa invocazione dei metodi remoti mediante RMI che produce l'attivazione delle seguenti operazioni da parte della Java Virtual Machine del client:

1. il programma Java (client) invoca un metodo che è stato dichiarato "remoto";
2. la Java Virtual Machine apre una connessione (grazie ai servizi di rete offerti dal sistema operativo ospite) con la macchina che ospita il server RMI; per far questo utilizza il protocollo di comunicazione TCP/IP ed una "porta" inutilizzata da altre applicazioni;
3. una volta stabilita una connessione con il server RMI, viene interrogato RMI Registry per ottenere i riferimenti interni relativi all'oggetto e al metodo invocato dal programma client;
4. con i riferimenti così ottenuti viene finalmente invocato il metodo sul server RMI;
5. la Java Virtual Machine del server RMI istanzia l'oggetto ed esegue il metodo su di esso, quindi restituisce i riferimenti all'oggetto istanziato o i dati di output del metodo invocato;
6. la Java Virtual Machine del client riceve l'output dal server e lo restituisce al programma client che lo utilizzerà opportunamente.

Infine, sempre nel secondo capitolo della tesi, abbiamo descritto i diversi passi che devono essere compiuti per costruire una applicazione basata sulla tecnologia RMI: è stato presentato un piccolo programma, a titolo di esempio, per il calcolo della media aritmetica tra due numeri letti in input. La lettura e la stampa del

risultato avvengono sul client, mentre il calcolo viene eseguito sul server mediante un oggetto remoto invocato con RMI.

Abbiamo presentato, nella parte conclusiva del capitolo, alcune distinzioni tra le tipologie di programmi che possono essere realizzati con il linguaggio Java: distinguiamo i programmi denominati *application* da quelli denominati *applet*. Le *application* sono programmi autonomi eseguibili dalla Java Virtual Machine, mentre le *applet* sono programmi che devono essere eseguiti dalla JVM di un web browser. Nell'esempio della media aritmetica abbiamo visto una *application*, mentre invece nel programma che presentiamo in questo capitolo, abbiamo utilizzato la tecnologia delle *applet*. Se dal punto di vista utente si distinguono le applet dalle application sulla base della modalità operativa con cui devono essere eseguite, dal punto di vista del programmatore la differenza è ben più sostanziale, ma, grazie anche alla modalità *object oriented* di Java, si riduce a poche differenze tecniche individuabili per lo più nella presenza del metodo `main()` nelle application (assente invece nelle applet) e nel fatto che l'implementazione di una applet estende la classe `Applet`. Per il resto i due tipi di programma sono molto simili, tanto da poter facilmente trasformare una applet in una application e viceversa.

Nel terzo capitolo viene infine descritto il problema matematico affrontato (la risoluzione di sistemi di equazioni differenziali ordinarie), concentrando l'attenzione sugli algoritmi per il calcolo approssimato di tali sistemi. Come esempi abbiamo utilizzato l'equazione dell'oscillatore armonico e dell'oscillatore armonico smorzato, l'equazione di Van der Pol ed il modello "preda–predatore" descritti dai seguenti sistemi:

1. Il movimento del pendolo senza attrito, governato dalla seguente equazione:

$$y_1''(t) = -k \sin(y_1(t))$$

   Utilizzando la trasformazione che permette di passare da un'equazione del secondo ordine ad un sistema di due equazioni del primo ordine:

$$\begin{cases} y_1'(t) = y_2(t) \\ y_2'(t) = y_3(t) \\ \dots \end{cases}$$

   si ottiene il sistema equivalente:

$$\begin{cases} y_1'(t) = y_2(t) \\ y_2'(t) = -k \sin(y_1(t)) \end{cases}$$

2. L'equazione di Van der Pol:

$$\begin{cases} y_1^{'}(t) = y_2(t) - y_1^3(t) + y_1(t) \\ y_2^{'}(t) = -y_1(t) \end{cases}$$

3. Modello preda-predatore:

$$\begin{cases} y_1^{'}(t) = ay_1(t) - cy_1(t)y_2(t) - e(t)^2 \\ y_2^{'}(t) = -by_2(t) + dy_1(t)y_2(t) \end{cases}$$

Questi tre sistemi di equazioni sono stati implementati con due algoritmi differenti: il metodo di Eulero ed il metodo di Runge-Kutta, codificati dai seguenti pseudo codici:

- Metodo di Eulero esplicito.

  Per implementare il metodo di Eulero occorre assegnare in input i seguenti dati:

1. f la velocità, nel nostro caso i campi vettoriali $f_1(y_1, y_2); f_2(y_1, y_2)$;
2. y il dato iniziale, nel nostro caso $y_1(0), y_2(0)$;
3. h il passo di integrazione;
4. N il numero di iterazioni.

L'algoritmo procede secondo i seguenti passi:

1. Considerata la matrice $w_{i,j}$ con $1 \leq i,\ j \leq N$, pone:

$$w_{0,1} = y_1(0)$$

$$w_{1,1} = y_2(0)$$

2. Calcola $y(n+1) = y(n) + hf(y(n))$
3. Se $n < N$ sostituisce $y(n+1)$ a $y(n)$ e ritorna al passo 2 altrimenti si arresta.

- Algoritmo di Runge-Kutta

  Per implementare il metodo di Runge-Kutta, assegnamo in input gli stessi dati del metodo di Eulero:

  1. f la velocitá $(f_1(y_1, y_2); f_2(y_1, y_2))$;
  2. y il dato iniziale $(y_1(0), y_2(0))$;
  3. h il passo di integrazione;
  4. N il numero di iterazioni.

  L'algoritmo del quarto ordine procede secondo i seguenti passi:

  1. Considerata la matrice $w_{i,j}$ con $1 \leq i,\ j \leq N$, pone:

$$w_{0,1} = y_1(0)$$

$$w_{1,1} = y_2(0)$$

  2. Calcola la funzione $g(y(n), h)$ usando la seguente formula:

$$g(y(n), h) = (k_1 + 2k_2 + 2k_3 + k_4)/6$$

  con

$$\begin{aligned} k_1 &= f(y) \\ k_2 &= f(y + hk_1/2) \\ k_3 &= f(y + hk_2/2) \\ k_4 &= f(y + hk_3) \end{aligned}$$

  3. Calcola $y(n+1) = y(n) + hg(y(n), h)$
  4. Se $n < N$ sostituisce $y(n+1)$ a $y(n)$ e ritorna al passo 2, altrimenti si arresta.

Una volta analizzato il problema numerico, partendo da quanto visto nei due capitoli precedenti, viene proposta l'architettura di calcolo per la realizzazione di un software per la soluzione approssimata dei sistemi di equazioni differenziali citati in precedenza. Vengono presentate due architetture differenti in cui la frammentazione del programma avviene secondo due distinte modalità. La prima prevede di realizzare un *server* dedicato all'esecuzione di funzioni semplici a cui vengono forniti come parametri solo pochi dati. Tali funzioni vengono però richiamate numerose volte dal *client* che si occupa di coordinare l'intera procedura di calcolo. La

seconda architettura proposta è stata realizzata spostando sul server l'esecuzione dell'intero algortitmo risolutivo: in questo caso il client richiamerà una sola volta il *metodo* disponibile sul server, ricevendo da questo una grande mole di informazioni (le coordinate dei punti che definiscono la soluzione approssimata del sistema). Sperimentalmente abbiamo potuto verificare che la prima architettura è preferibile alla seconda nel caso in cui la connessione di rete è piuttosto veloce e si dispone di una macchina client di buona potenza. La seconda architettura invece, fornisce il massimo del beneficio quando viene utilizzata su client di scarsa potenza con una connessione di rete anche lenta; in questo caso però il server dovrà essere una macchina piuttosto potente.

La nostra sperimentazione non si è limitata a questo: nella parte iniziale del nostro lavoro abbiamo infatti provveduto ad implementare gli algoritmi risolutivi utilizzando linguaggi di programmazione differenti: ad una prima versione scritta in linguaggio Pascal ne abbiamo affiancate altre due scritte rispettivamente in linguaggio C ed in linguaggio Java. Abbiamo quindi confrontato (a parità di algoritmo di calcolo) l'efficienza delle tre codifiche: il risultato, peraltro abbastanza ovvio e prevedibile, ha portato alla conclusione che la codifica in C era la più efficiente, mentre la meno efficiente era proprio quella scritta in Java. Abbiamo però deciso di trascurare questo risultato, a fronte di alcune considerazioni che ci permettono comunque di preferire il linguaggio Java, nonostante sia in certi casi meno efficiente.

Queste considerazioni sono state poi messe in evidenza dal nostro lavoro e sono riportate diffusamente in questa tesi: mediante Java siamo in grado di distribuire facilmente l'applicazione su più CPU, mentre una analoga architettura di calcolo sarebbe improponibile in linguaggio Pascal e di difficile implementazione utilizzando il C. Inoltre, come vedremo più avanti, Java ci permette di scrivere applicazioni che possono essere eseguite senza alcuna modifica su macchine completamente diverse, facendo anche uso di output grafici che rendono la nostra applicazione facile da utilizzare e di grande impatto visivo.

Nella appendice della tesi sono riportati i listati in linguaggio java dei due programmi che implementano le diverse architetture di calcolo distribuito descritte nel terzo capitolo ed alcune stampe dell'output prodotto dal nostro programma.

Riferimenti bibliografici

[1] G. Ausiello, A. Marchetti-Spaccamela, M. Protasi, *Teoria e progetto di algoritmi fondamentali*, Franco Angeli, 1988.

[2] B. Blount, S.Chatterjee, *An evaluation of Java for numerical computing*, preprint, 1999.[1]

[3] L. Comi, *Java flash*, Apogeo, 1996.

[4] D. Flanagan, *Java in a Nutshell*, O'Reilly & Associates, 1996.

[5] M.Hirsch, S. Smale, *Differential equations, Dynamical System and Linear Algebra*, Academic Press, 1974.

[6] B.W. Kernighan, R. Pike, *Practical programming*, Addison-Wesley, 1999.

[7] M. Snir, J. Moreira, M. Gupta, L. Haibt, S. Midkiff, *Floating-point performance in Java*, preprint, "T.J. Watson Research Center" IBM, 1998.

[8] P. Naughton, *Il manuale Java*, Mc Graw-Hill, 1996.

[1] I *preprint* riportati in questa bibliografia sono tratti dagli atti della conferenza annuale *Java Grande Conference*, promossa dalla ACM, tenutasi nel Novembre del 1998 alla Stanford University di Palo Alto e nel Giugno 1999 a San Francisco.

[9] R.A. Plastock, G. Kalley, *Teoria e problemi di Computer Grafica*, Etas Libri, 1989.

[10] A. Quarteroni, R. Sacco, F. Saleri, *Matematica numerica*, Springer, 1998.

[11] P. van der Linden, *Just Java and Beyond* , Practice-Hall, 1998.

[12] P. Wu, S. Midkiff, J. Moreira, M. Gupta, *Efficient support for complex numbers in Java*, preprint, 1999.

[13] AAVV, *MokaBook*, a cura di G. Puliti, Hoops, 2001.