Agenti Mobili per la gestione delle applicazioni native in sistemi distribuiti*

Rocco Aversa Beniamino Di Martino Salvatore Venticinque

Dipartimento di Ingegneria dell'Informazione
Seconda Università degli Studi di Napoli
via Roma, 29 - 81031 Aversa
{salvatore.venticinque,rocco.aversa}@unina2.it, beniamino.dimartino@unina.it

ABSTRACT

Viene qui presentato un servizio ad agenti mobili per la gestione di programmi nativi in esecuzione su sistemi distribuiti ed eterogenei. Un agente è in grado di interfacciarsi all'applicazione nativa attraverso un cruscotto che consente di richiamare funzioni ad hoc sull'applicazione da controllare. Il gestore dell'applicazione può ricevere comandi dall'utente in maniera interattiva oppure da un sistema che in maniera automatica applica politiche di gestione delle risorse, ad esempio per il bilanciamento del carico[1]. Il fine è il progetto e lo sviluppo di un servizio di gestione, portabile su architetture eterogenee, in grado di migrare la sua esecuzione e quella dell'applicazione controllata da un nodo all'altro nella rete.

1. INTRODUZIONE

Un sistema distribuito è costituito da un insieme di risorse connesse in rete di tipo eterogeneo. Allo stato dell'arte esistono diverse insfrastrutture hardware e software che mettono a disposizione dell'utente servizi per l'esecuzione in remoto delle sue applicazioni. Tuttavia la gestione dell'eterogeneità continua ad essere un problema aperto. Da un lato esistono ambienti e linguaggi indipendenti dall'architettura della macchina utilizzata che possono essere utilizzati per introdurre maggiore flessibilità [2], d'altra parte la maggior parte delle applicazioni per il calcolo scientifico sono implementate in linguaggi come FORTRAN o C, e sono compilate per l'architettura target al fine di ottimizzare le performance. Quindi un requisito fondamentale per la realizzazione di un tale servizio è quello di evitare la riscrittura del programma in un diverso linguaggio. Il servizio di gestione è implementato come agente mobile, ed è in grado di migrare ed eseguire le funzionalità di gestione su un qualunque nodo del sistema su cui è possibile spostare l'applicazione da monitorare e/o controllare. I programmatori, al fine di utilizzare il servizio, possono estendere la propria applicazione senza modificarne il codice originale, ma semplicemente ridefinendo quei metodi che ne specializzano il comportamento al verificarsi di particolari eventi. Il modello di gestione supporta attualmente checkpointing, sospensione, ripristino, migrazione e supervisione. É in fase di sviluppo una interfaccia WSRF che consente l'integrazione del servizio in un architettura Grid. In particolare è già in fase di sviluppo l'integrazione di tale servizio nella piattaforma MAGDA (Mobile Agents based GriD Architecture) [3].

*Questo lavoro è stato supportato dalla Giunta Regionale della Campania – Assessorato alla Ricerca Scientifica , nell'ambito del progetto di ricerca "Magda una piattaforma ad agenti mobili per il Grid Computing", finanziato ai sensi della Legge Regionale n. 05 del 28/03/2002.

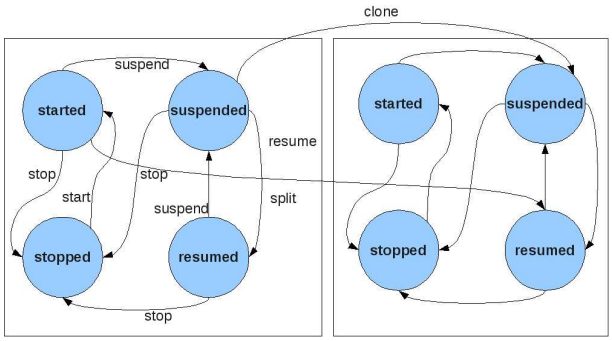

Figure 1: Modello di gestione dell'applicazione

2. REQUISITI DEL CRUSCOTTO

Per il modello di gestione è stato definito il ciclo di vita dell'applicazione rappresentato in Figura 1 In particolare la gestione di ogni evento attraverso le transizioni rappresentate in figura sono stati tradotti nelle funzionalità realizzate dal cruscotto:

- *Avvio dell'applicazione*: deve essere data la possibilità all'utilizzatore dell'infrastruttura di poter avviare l'applicazione su un nodo del cluster;
- *Sospensione dell'applicazione*: si deve garantire l'opportunità di poter sospendere temporaneamente l'applicazione senza perdere il contenuto informativo fino ad allora elaborato;
- *Ripristino dell'applicazione*: l'utilizzatore deve essere

in grado di ripristinare l'esecuzione dell'applicazione, a seguito di una sospensione, ripartendo dal punto di elaborazione precedentemente raggiunto;

- *Terminazione dell'applicazione*: deve essere data la possibilità di poter definire una condizione di corretta terminazione anche quando essaè eseguita da più processi;
- *Checkpointing*: nell'ambito di utilizzo su cluster, deve essere possibile effettuare degli snapshot dello stato dell'applicazione ai fini di consentire il restart a seguito di guasti sul sistema. Questa funzionalità può consentire la costruzione di piattaforme fault-tolerant;
- *Spostamento dei processi*: si deve garantire l'opportunità di poter spostare l'esecuzione dei processi su vari nodi del cluster senza perdere contenuto informativo precedentemente elaborato. Attraverso questa azione è possibile implementare politiche di load-balancing;
- *Clonazione dei processi e suddivisione dei tasks*: nell'ambito dell'utilizzo della piattaforma distribuita per il calcolo parallelo, deve essere data la possibilità all'utente o al sistema di effettuare clonazioni di interi processi e di poter suddividere i tasks tra più cloni in modo da poter combinare le capacità di calcolo delle diverse macchine;
- *Gestione della terminazione*: si deve garantire la possibilità di poter comunicare l'avvenuta terminazione di un lavoro da parte di un processo a tutti gli altri che concorrono all'esecuzione parallela di una stessa applicazione.

I requisiti non funzionali considerati per il progetto sono:

- *portabilità del codice*: è richiesta la portabilità del codice su varie architetture e/o sistemi operativi per consentire l'utilizzo della funzionalità di spostamento dei processi tra i nodi del cluster;
- *utilizzo di codice nativo*: si vuole permettere l'utilizzo di codice nativo per sfruttare a pieno le peculiarità della macchina sulla quale si ha intenzione di eseguire l'applicazione realizzata;
- *gestione dell'applicazione nativa*: è richiesta la gestione dell'applicazione nativa su qualsiasi nodo della piattaforma essa si trovi;
- *gestione dell'eterogeneità del sistema*: pur utilizzando codice nativo, si vuol dare la possibilità di consentirne l'esecuzione su una piattaforma eterogenea e distribuita.

3. ARCHITETTURA SOFTWARE

Come mostrato in Figura 2 l'architettura del servizio di gestione è articolata su tre livelli, ognuno composto di diversi elementi. Un primo insieme di componenti realizza un servizio portabile mediante la tecnologia ad Agenti Mobili. Agenti distribuiti assumono diversi ruoli. Essi implementano l'interfaccia utente, le funzionalità di supervisione e controllo, la gestione dell'applicazione. Un sistema di monitoring rileva i

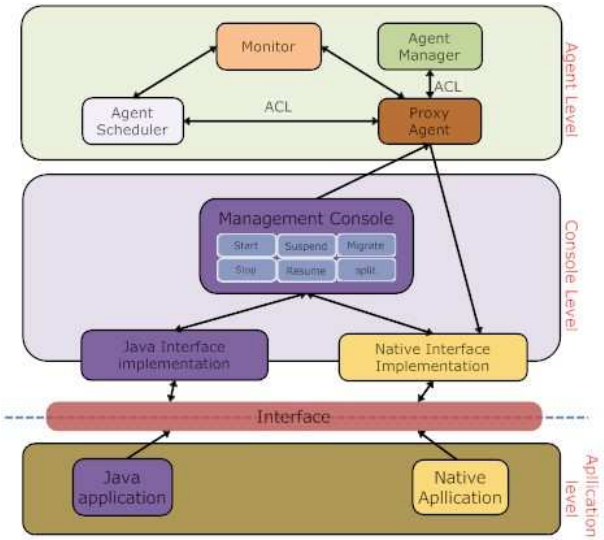

Figure 2: Architettura del servizio

cambiamenti di interesse delle condizioni di funzionamento del sistema e notifica questi eventi ad un Agente Scheduler. L'agente Scheduler reagisce a tali eventi per evitare che si abbia un degrado delle prestazioni. Esso comunica con gli Agenti Proxy e li coordina. Un agente Proxy è responsabile di una istanza di esecuzione di un'applicazione. Esso riceve richieste via messaggi ed attua le azioni di gestione utilizzando una cruscotto realizzato in Java. Tale cruscotto software rappresenta una interfaccia astratta tra il proxy e l'applicazione. L'interfaccia è ovviamente indipendente dal tipo di applicazione da controllare e dalla tecnologia con cui l'applicazione è realizzata. Le applicazioni native in particolare verrà collegata al cruscotto mediante una la tecnologia JNI (Java Native Interface) che supporta la realizzazione di metodi astratti di una classe java mediante il collegamento a run-time a librerie dinamiche native. Al fine di poter spostare l'esecuzione su architetture eterogenee, il programmatore deve ricompilare il codice per le diverse piattaforme target. Il risultato della compilazione consiste in una libreria dinamica che comprende il codice dell'applicazione e la le procedure software che definiscono i comportamenti attivati dai comandi di gestione. Tali librerie verranno salvate in un archivio remoto da cui l'Agente Proxy è in grado di scaricarle all'occorrenza.

4. TECNOLOGIE E REALIZZAZIONE

L'infrastruttura di gestione fa uso della tecnologia ad *agenti mobili* [4]. In particolare, per perseguire gli obiettivi di astrazione previsti da questo strato, il middleware in esame ha fatto uso della piattaforma **JADE** [5] che, come è richiesto, nasconde la struttura eterogenea e distribuita del cluster.

Una interfaccia per la gestione interattiva delle applicazioni realizza un cruscotto provvisto di una serie di pulsanti al cui nome è associata la funzionalità invocata sull'applicazione. L'intera interfaccia è stata realizzata mediante il linguaggio **Java** che garantisce l'assoluta portabilità del codice.

Il cruscotto è stato sviluppato parte in Java, ed esporta i

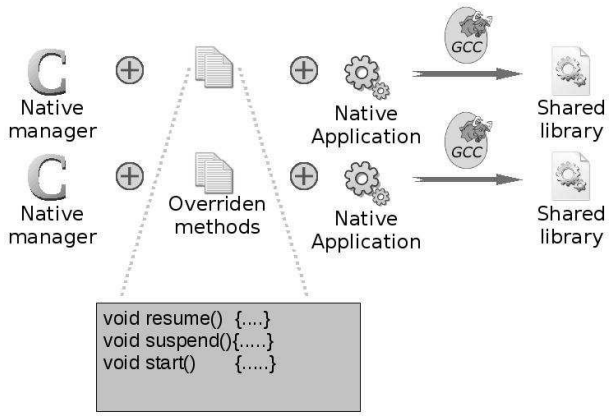

Figure 3: Generazione della libreria di gestione

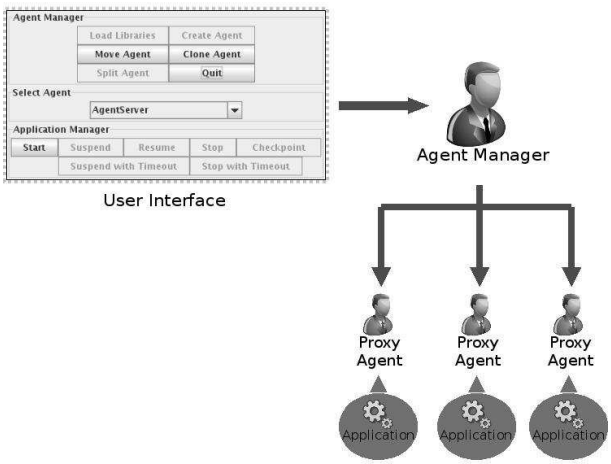

Figure 4: Cruscotto per il controlo interattivo

metodi per la gestione dell'applicazione da parte dell'Agente Proxy. L'implementazione nativa è stata sviluppata utilizzando il linguaggio C in standard POSIX . Per collegare i metodi del cruscotto universale alle funzioni native del manager è stata utilizzata la tecnologia **JNI** (Java Native Interface)[6].

5. ESTENSIONE DELL'APPLICAZIONE NATIVA E UTILIZZO DEL SERVIZIO

In definitiva, lo sviluppatore del programma nativo deve semplicemente ridefinire le suddette funzioni e linkare la sua applicazione con il manager nativo che gli viene fornito. In questo modo creerà una libreria dinamica contenente la propria applicazione che potrà essere governata attraverso il cruscotto Java. In Figura 3 sono rappresentati i diversi moduli oggetto che compongono la libreria dinamica collegata al cruscotto Java. I moduli rappresentati realizzano:

- Implementazione POSIX del cruscotto;
- Metodi che specializzano il comportamento dell'applicazione in risposta ai comandi di gestione;
- Codice applicazione.

L'interfaccia grafica dell'applicazione per il controllo interattivo è mostrata in Figura 4. Essa consente di caricare diverse versioni della libreria che realizza l'applicazione nativa, in modo tale da consentire al manager dell'applicazione di di identificare e utilizzare la libreria corretta semplicemente scaricando le varie dll da un sito ftp. Per ogni versione della libreria occorre specificare sistema operativo e processore per la quale è stata compilata. In questo modo il gestore, riconoscendo a runtime le caratteristiche della macchina sulla quale è in esecuzione, è in grado effettuare il download dal sito ftp della corretta libreria dinamica. Definita la posizione e le corrispondenze delle librerie, è possibile creare l'agente proxy, il quale, con questa configurazione, è in grado di spostarsi sulle diverse macchine e controllare l'esecuzione dell'applicazione. La comunicazione tra l'interfaccia utente, l'agente manager, e gli agenti responsabili della gestione di una particolare istanza di esecuzione avviene mediante scambio di messaggi.

6. REFERENCES

[1] R. Aversa, B. Di Martino, R.D., Venticinque, S.: Load balancing of mobile agents based applications in grid systems. In: Proceedings of 8 IEEE 17th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE (2008)

[2] S.Venticinque, Moscato, F., Martino, B.D., Aversa, R., Mazzocca, N.: Enabling mobile agents trasparent execution context migration through globus services. In: Proceedings of 1st Austrian Grid Symposium, Oesterreichische Computer Gessellschaft (2006) 256–266 J. Volkert, T. Farhringher, D. Kranzlmuller, W. Schreiner (eds.).

[3] R. Aversa, B. Di Martino, N.M.S.V.: Magda: A mobile agent based grid architecture. Journal of Grid Computing **4**(4) (2006) 395–412

[4] V. A. Pham, A.K.: Mobile software agents: an overview. IEEE Communications Magazine **36**(7) (1998) 26–37

[5] F. L. Bellifemine, G. Caire, A.P.G.R.: Jade: a white paper. (2003)

[6] Liang, S.: The Java Native Interface: Programmer's Guide and Specification. Prentice Hall PTR (1999)