Analisi del framework XCat come ambiente di supporto alla programmazione di applicazioni ad alte prestazioni per ambienti grid-enabled

Tiziano Fagni
Gruppo di ricerca HPC (High Performance Computing)
Istituto ISTI, Consiglio Nazionale delle Ricerche (CNR), Pisa, Italy

05/12/2003

Sommario

Disporre di strumenti di programmazione potenti e flessibili è un grande vantaggio per qualsiasi programmatore. Questo è ancora più vero se l'ambiente di runtime target è rappresentato dalle griglie computazionali (notevolmente complesse a causa della loro natura fortemente eterogenea) e le applicazioni da sviluppare sono molto sofisticate e ad alte prestazioni. L'articolo descrive ed analizza XCat, un framework CCA-compliant per lo sviluppo di software high-performance su griglia, che mira a facilitare la fase di costruzione e il mantenimento di software di questo tipo. La libreria, ai fini di questi obiettivi, offre paradigmi di programmazione avanzati basati sul meccanismo di "porte dinamiche" per la comunicazione tra componenti ed inoltre permette l'uso di servizi di scripting per la scrittura di applicazioni di controllo. Nel corso dell'articolo saranno inoltre analizzate le performance dell'ambiente in varie condizioni operative, in modo da valutare correttamente la bontà dell'efficienza complessiva fornita dal sistema.

1 Introduzione

Una griglia computazionale è una sorta di grosso calcolatore parallelo e virtuale che mette a disposizione dei potenziali utilizzatori un insieme dinamico di risorse hardware e software, connesse in qualche modo ed opportunamente virtualizzate, in grado fornire un enorme capacità di calcolo a costi praticamente irrisori. Queste insieme di caratteristiche fanno si che il potenziale delle griglie computazionali sia molto elevato e che il loro successo dipenda dalla disponibilità di strumenti avanzati di programmazione che permettano di sfruttare con relativa facilità tutte queste risolse di calcolo.
Un modello di programmazione per le griglie consiste in una serie di tool, librerie, convenzioni e protocolli atti a "virtualizzare" le risorse disponibili e direttamente utilizzabili dal programmatore per rendere meno gravoso lo sviluppo di applicazioni grid-aware. Ovviamente, maggiore sarà il livello di astrazione dei tool di sviluppo, minore sarà lo sforzo per il generico programmatore. In questa ottica, negli ultimi anni hanno avuto una sempre maggior diffusione i cosiddetti "ambienti di programmazione a componenti". Lo sviluppo di codice tramite i componenti comporta una serie di vantaggi rispetto ad un modello di programmazione classico. In primo luogo i componenti (per definizione) incapsulano al loro interno tutto il codice necessario al loro funzionamento rendendo in tal modo la programmazione "più pulita", inoltre questo aspetto influisce positivamente anche sul riuso del codice stesso facendo aumentare la produttività di un programmatore. Infine, la caratteristica di essere oggetti in qualche modo autonomi, permette in modo rapido e con relativa facilità (se supportati da un

buon framework) di comporre componenti per creare applicazioni anche complesse.
In passato sono state proposte varie tecnologie per architetture a componenti (ad esempio CORBA, JavaBeans, Microsoft DOM, Microsoft .NET, ecc.) ma nessuna di queste si è rivelata adatta allo sviluppo di applicazioni ad alte prestazioni. Per questo motivo è stato proposto lo standard CCA, una specifica aperta che mira a facilitare lo sviluppo di applicazioni "intensive"[1] promovendo caratteristiche che facilitino la generazione di codice efficiente e il riuso del codice stesso.
In questo articolo analizzeremo XCat, una implementazione di CCA per ambienti distribuiti/grid. Nella sezione 2 descriveremo le caratteristiche dello standard CCA, focalizzandoci in particolare sul funzionamento del meccanismo delle porte come metodo di comunicazione tra i componenti. Nella sezione 3 vedremo le caratteristiche essenziali dell'architettura di XCat mentre nella sezione 4 descriveremo la procedura da seguire per installare con successo la libreria. Nella sezione 5, infine, analizzeremo i risultati sperimentali ottenuti durante la fase di testing dell'ambiente.

2 Il modello CCA

L'architettura a componenti CCA (Common Component Architecture)[4, 3, 1] è stata sviluppata come progetto congiunto internazionale tra numerosi laboratori di ricerca e istituzioni accademiche con lo scopo di definire un'architettura comune e standard con la quale costruire applicazioni scientifiche a larga scala a partire da componenti software ben testati. Inizialmente, agli inizi del 1999, lo standard era stato pensato per sviluppare applicazioni di tipo "intensive" da far girare su supercomputer paralleli, successivamente con la diffusione delle piattaforme di Grid Computing sono state realizzate implementazioni di CCA che tenessero conto anche di questo tipo di ambienti (XCat ne è un esempio). Inoltre, vi è stata anche una evoluzione dello standard per quanto riguarda la compatibilità dei componenti CCA-aware: si è passati da una fase iniziale nella quale era richiesta una compatibilità a livello dei sorgenti alla fase attuale nella quale la compatibilità richiesta è binaria (quindi senza necessità di ricompilazione dei componenti su ciascuna architettura hardware/software utilizzata).

CCA è stato pensato per superare i limiti nello sviluppo di applicazioni ad alte prestazioni delle architetture a componenti esistenti (CORBA, DOM, JavaBeans, ecc.). Per ottenere questo obiettivo, sono stati considerati vari requisiti che devono essere presi in considerazione da un ambiente di questo tipo:

Caratteristiche dei componenti I componenti sono pensati per generiche applicazioni ad alte prestazioni quindi possiamo avere una moltitudine di componenti con caratteristiche diverse: da quello a grana grossa a quello a grana fine, da quello che usa un paradigma di programmazione SPMD a quello che usa un modello multithread con memoria condivisa;

Eterogeneità In pratica dovrebbe essere possibile costruire nuove applicazioni componendo fra loro componenti scritti in linguaggi diversi e in esecuzione su architetture differenti (intendendo con questo architetture hardware, sistemi operativi e ambienti di runtime diversi);

Componenti locali e remoti Utilizzare un servizio di un componente locale (cioè in esecuzione all'interno dello stesso spazio di indirizzamento dell'applicazione) deve costare sola la chiamata ad una funzione. Nel caso di componenti remoti si devono invece sfruttare protocolli a 0-copia per il trasferimento dei dati e, ove possibile, anche sfruttare ottimizzazioni o caratteristiche particolari offerte dallo strato di networking sottostante;

Alte prestazioni In generale l'ambiente deve essere il più possibile efficiente quindi, ove applicabile, è necessario realizzare protocolli di comunicazione ottimizzati sfruttando caratteristiche particolari dell'architettura hardware e del sistema operativo sottostante, evitare extra copie, sincronizza-

[1] Con il termine "intensive" si intendono applicazioni che usano in modo pesate la CPU, l'I/O dei dati oppure entrambe queste caratteristiche.

zioni inutili e qualsiasi altra cosa "costosa" che non è strettamente necessaria;

Integrazione I componenti devono essere direttamente utilizzabili (salvo eventualmente una ricompilazione) su ambienti runtime CCA differenti in esecuzione su architetture eterogenee.

La specifica CCA è definita in modo *lightweight* ovvero è uno standard poco stringente, che definisce il comportamento di massima di un componente e che lascia agli sviluppatori delle singole implementazioni[2] la libertà di realizzare certe generiche funzionalità (menzionate nella specifica) nella maniera che ritengono più opportuna. In particolare, CCA non definisce:

- come realizzare il framework ovvero quale linguaggio di programmazione, quale eventuale libreria esterna o quale architettura logica interna utilizzare;
- come l'utente potrà interagire con il sistema per costruire, eseguire e controllare applicazioni;
- come i componenti potranno essere creati, connessi e trovati all'interno dell'ambiente di runtime.

Il concetto più importante fissato dalla specifica CCA è il modo con il quale i componenti possono comunicare (e quindi in un certo senso "agganciarsi) tra loro. Lo standard definisce il concetto di composizione di componenti mediante l'uso delle cosiddette "porte". Queste non sono altro che una descrizione di una interfaccia che rappresenta un certo insieme di operazioni disponibili su un particolare oggetto. L'interfaccia definita su una particolare porta dovrà poi essere utilizzata secondo il paradigma *uses/provides* a seconda che il componente considerato voglia utilizzare o voglia fornire un determinato servizio. Per chiarire questo concetto, prendiamo il componente di esempio rappresentato in figura 1 a). Questo implementa due porte di tipo *provides* e quindi fornisce i servizi dichiarati all'interno di queste due interfacce. La prima porta definisce una sola funzione (*setImage()*) che ha lo scopo di impostare l'immagine da elaborare successivamente mentre l'altra porta *provides* definisce altre tre operazioni che agiscono direttamente sull'immagine stessa. Il componente, inoltre, per poter funzionare correttamente, dichiara anche di dover utilizzare una porta di tipo *uses* che contiene al suo interno una funzione chiamata *doFFT()*. Essendo quest'ultimo un requisito fondamentale per il corretto funzionamento di questo modulo software, è necessario trovare un altro componente che implementi la corrispondente porta *provides* e possa in questo modo essere composto con il componente precedente. Questo "gioco a incastro" tra componenti mediante le porte uses e provides è rappresentato in figura 1 b) dove viene modellata una ipotetica applicazione grafica che fa uso del componente descritto precedentemente. Il modello descritto è piuttosto flessibile e permette con relativa facilità di poter costruire "metacomponenti" (ovvero componenti di componenti) a partire da quelli "primitivi".

[2]Attualmente esistono varie implementazioni di framework CCA che sono ottimizzate per particolari ambienti operativi: CCaffeine per un tipo di programmazione SPMD, XCat per ambienti distributi/grid, Uintah per ambienti multithread con memoria condivisa.

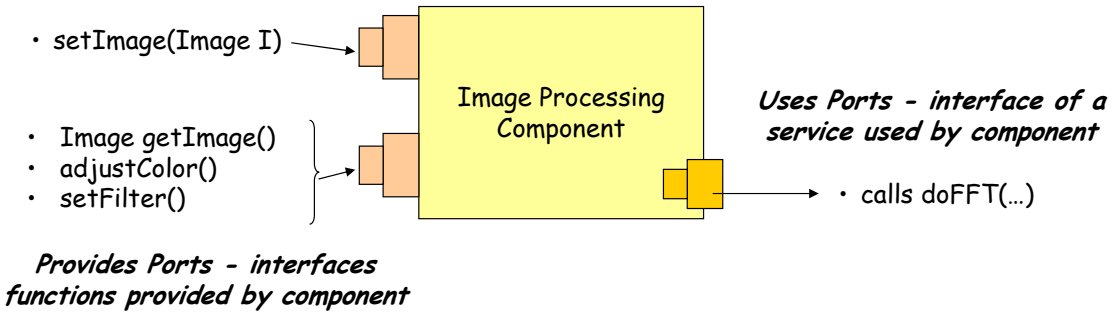
3 La libreria a componenti XCat

Una implementazione CCA che gode di una buona popolarità nella comunità scientifica è rappresentata dal software XCat[2], un framework ideato e sviluppato dal team di Dennis Gannon della Indiana University (USA) per la realizzazione di applicazioni ad alte prestazioni in ambiente distribuito/grid. Il middleware, attualmente arrivato alla versione 2.0.2-RC1, è stato scritto in Java (anche se è in fase di realizzazione una versione analoga in C++) e permette di scrivere componenti CCA-compatibili direttamente utilizzabili in applicazioni che supportano questa specifica.

3.1 Architettura

XCat, essendo pensata per ambienti di calcolo distribuiti, cerca di rimanere in linea con l'attuale stato dell'arte: in particolare, quindi, oltre a supportare i classici ambienti distribuiti omogenei ed eterogenei,

a) Porte uses-provides di un componente



b) Esempio di costruzione di una ipotetica applicazione grafica

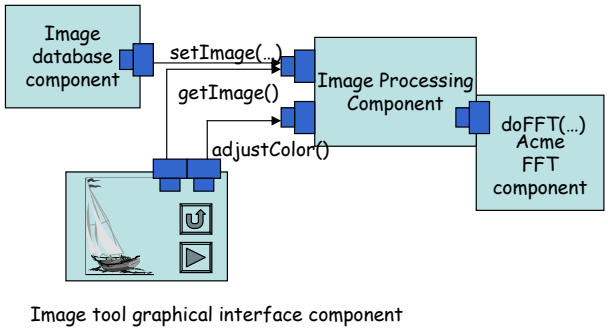


Figura 1: Rappresentazione del modello di interazione “uses/provides port”

permette di scrivere applicazioni direttamente eseguibili all'interno di una griglia computazionale (realizzata mediante il software Globus). Il supporto per quest'ultimo tipo di ambiente, come vedremo, è ancora piuttosto limitato ed inoltre obsoleto rispetto all'ultima versione rilasciata di Globus (la versione 3, che risulta differente dalle versioni precedenti sia come modello dell'architettura sia come compatibilità). Attualmente il team di sviluppo di XCat sta cercando di fare il porting del framework su quest'ultima versione di Globus ma, tuttora, hanno confermato di essere in uno stato di avanzamento embrionale e di non sapere quando potranno rilasciare una nuova versione del loro software.
L'architettura di XCat è basata sul modello dei Web-Service che in questi ultimi anni ha riscosso un notevole successo e sembra stia diventando uno standard "de facto" nella realizzazione di middleware che realizzano ambienti grid. In particolare, il modello si basa su uno stack a cinque livelli:

- **Framework**. Rappresenta lo strato più ad alto livello del modello e fornisce tutti i servizi avanzati necessari per costruire ed eseguire le applicazioni. In XCat questo è rappresentato dalla realizzazione della specifica CCA e da vari servizi primitivi offerti agli applicativi (vedi sezione 3.2).

- **Discovery**. Questo livello è necessario per pubblicare e recuperare i servizi offerti dagli applicativi ed utilizza un registry per tenere traccia di tutte le funzionalità da esportare. Nell'ambito dei Web-Service il protocollo di riferimento è rappresentato da UDDI (Universal Description, Discovery, and Integration) mentre in XCat si è scelta una strada alternativa, adottando un registry di tipo LDAP (Lightweight Directory Access Protocol).

- **Descrizione**. Rappresenta il livello utilizzato per descrivere i servizi disponibili (interfacce disponibili, protocolli di trasporto supportati, ecc.). Nei Web-Service lo standard è di utilizzare il linguaggio WSDL (Web Service Description Language). In XCat le interfacce dei componenti vengono invece descritte tramite un sottoinsieme del WSDL, utilizzato anche per generare il codice "collante" necessario ai componenti scritti in linguaggi di programmazione differenti per comunicare tra loro in modo corretto.

- **Messaggio**. Questo livello viene utilizzato per codificare/decodificare i dati da trasmettere sulla rete. Questa operazione (detta anche di marshaling/unmarshaling) è necessaria per poter scambiare dati in modo trasparente tra applicazioni/componenti in esecuzione su architetture o sistemi operativi differenti. In XCat questo compito è svolto da XSoap, una libreria che realizza (in Java e C++) il modello RMI di Java sfruttando SOAP (Simple Object Access Protocol) come meccanismo di trasporto. L'utilizzo di tale libreria permette a XCat di sfruttare alcune ottimizzazioni quali ad esempio l'utilizzo, nel caso di componenti scritti in Java, di un meccanismo di creazione automatica degli stub e degli skeleton necessari per l'invocazione remota di un metodo.

- **Trasporto**. Questo livello indica la tecnologia utilizzata per trasportare i messaggi all'interno di un'applicazione (parallela, distribuita, ecc.). Come già detto nel punto precedente, in XCat viene sfruttato SOAP come protocollo di comunicazione.

3.2 Servizi avanzati disponibili all'interno del framework

Per rendere la vita più facile ai programmatori e fornire un ambiente di runtime flessibile e ad alto livello, XCat offre alcuni servizi avanzati direttamente utilizzabili dalle applicazioni e dai componenti realizzati con l'ausilio del framework.

Il servizio *CreationService*

Il CreationService è un componente CCA della libreria che mette a disposizione un insieme di funzionalità per la creazione di componenti. I suoi servizi possono essere invocati da qualsiasi componente CCA per istanziare/rimuovere altri componenti CCA.

I meccanismi offerti da XCat per la creazione di nuovi componenti sono di quattro tipi e dipendono essenzialmente da dove e come vogliamo istanziare un componente:

- **local**. Utilizzando questo meccanismo il nuovo componente verrà istanziato come thread separato all'interno dello spazio di indirizzamento del componente chiamante.
- **proc**. Specificando questo meccanismo il nuovo componente sarà creato sulla stessa macchina del componente chiamante però in un processo separato.
- **rsh/ssh**. Questi due meccanismi permettono di creare un componente utilizzando rispettivamente i tool *rsh* e *ssh*.
- **gram**. Questo meccanismo consente di instanziare il nuovo componente utilizzando il modulo Gram di Globus.

Ovviamente, in fase di creazione di un componente, oltre al meccanismo da usare per la istanziazione, è possibile specificare altri parametri fondamentali quali la locazione dell'eseguibile, la macchina su cui creare il componente e i parametri da linea di comando da passare al componente.

Il servizio *ConnectionService*

Questo servizio è necessario per connettere le porte di tipo *uses* di un componente con le porte di tipo *provides* di un altro componente. Il meccanismo di connessione è dinamico nel senso che è possibile a runtime aggiungere o rimuovere connessioni tra i componenti. Il servizio mette inoltre a disposizione un meccanismo che permette ad un componente di esporre funzionalità (porte *provides*) di altri componenti, come se fossero proprie.
Per descrivere le interfacce di una porta si possono utilizzare due metodi. Il primo consiste nel descrivere l'interfaccia della porta mediante un file interpretabile da un compilatore SIDL (il software Babel) il quale provvederà a generare tutto il codice "colla" necessario (stub, skeleton, eventuali conversione dei dati, ecc...) per far comunicare correttamente due componenti che utilizzano la stessa porta. L'altro metodo consiste nello sfruttare le informazioni fornite dal runtime di Java per generare stub e skeleton in modo dinamico. Nella pratica il primo metodo è inutilizzabile in quanto il software Babel non supporta pienamente il linguaggio Java e quindi tutte le prove ed i test sono stati fatti utilizzando il secondo metodo.

IL servizio *NamingService*

Il servizio di Naming realizza un registro in cui memorizzare le informazioni relative ai componenti istanziati che rendono disponibili le proprie funzionalità. Attualmente l'operazione di lookup di un componente non è associata ad alcun meccanismo di sicurezza, in futuro il team di sviluppo di XCat prevede di inserire una forma di ACL (Access Control List) per il controllo dell'accesso ad un componente.

3.3 Modelli di programmazione

In XCat l'unità minima di computazione è rappresentata da un componente, il quale ha il compito di informare il "mondo esterno" delle funzionalità che esporta e di cui ha bisogno. Ovviamente per poter essere integrati correttamente all'interno dell'ambiente, tale compito è riservato anche ai meta-componenti (componenti composti da vari componenti opportunamente collegati).
Per collegare e coordinare i componenti vengono create le cosiddette applicazioni di controllo. In XCat esistono sostanzialmente due possibili soluzioni.

Il primo modello usa direttamente le API Java messe a disposizione da XCat: si tratta di scrivere una semplice applicazione di controllo in Java che usi direttamente le funzionalità messe a disposizione dai servizi base del framework. Infatti, tramite il *Creation Service* è possibile creare ed ottenere i riferimenti a nuove istanze di oggetti in esecuzione. Il *Connection Service* può essere invece usato per connettere le porte *uses* e *provides* dei componenti da utilizzare per modellare l'applicazione considerata. Il *Naming Service*, infine, può essere utilizzato per registrare od ottenere riferimenti ad istanze di componenti "inte-

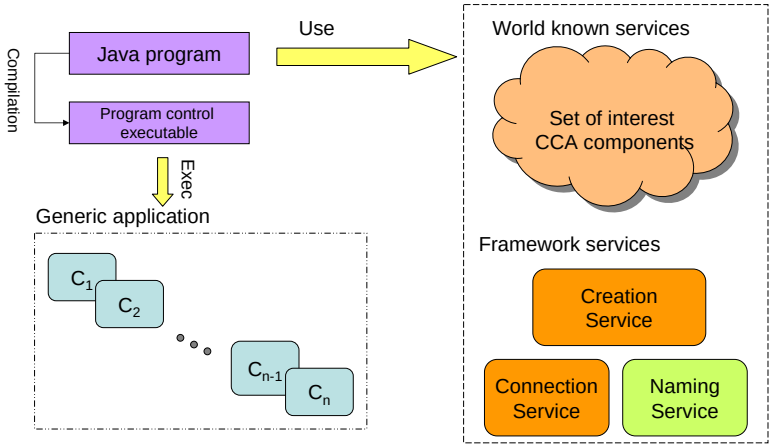


Figura 2: Modello di programmazione che usa un programma di controllo Java

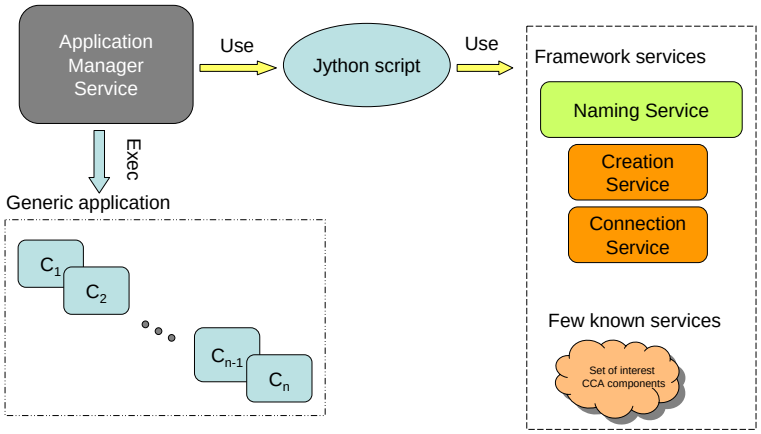


Figura 3: Modello di programmazione che usa un programma di controllo Jython

ressanti"[3] per il programma che stiamo modellando. L'idea appena descritta può essere sintetizzata nel disegno in figura 2 dal quale si evince che il modello può essere utilizzato per realizzare applicazioni piuttosto statiche, delle quali si conosci già a priori il tipo di comportamento e le risorse (componenti) utilizzate.
Un approccio più flessibile e dinamico, che supera le limitazioni del modello precedente, è rappresentato dal secondo meccanismo di programmazione messo a disposizione da XCat: l'utilizzo di script di tipo Jython (figura 3). L'oggetto *Application Manager Service* è un interprete Jython che ha il compito di eseguire uno script che descrive un'applicazione (o parte di un'applicazione) da modellare. Esso, come nel caso del modello precedente, ha accesso a tutti i servizi e le funzionalità messe a disposizione dall'ambiente ma permette anche di poter eseguire "al volo" più di uno script che può andare ad influire sul comportamento dell'applicazione da eseguire. Quest'ultimo aspetto rappresenta il punto chiave per poter fornire agli sviluppatori di applicazioni di una certa complessità un meccanismo flessibile e dinamico con cui modellare a proprio piacimento il comportamento del software sviluppato con XCat.

4 Installazione di XCat

Il primo passo da eseguire per installare il framework è quello di scaricarsi il pacchetto *tgz* dei sorgenti dal sito internet di XCat (http://www.extreme.indiana.edu/xcat/tutorial/).
La versione utilizzata per svolgere i test è la 2.0.2-RC1 ed è compatibile unicamente con la specifica JVM (Java Virtual Machine) 1.3. Il pacchetto contiene al suo interno quasi tutte le librerie ed applicazioni esterne (ad esempio XSoap, Ant, Jython, ecc.) necessarie al suo corretto funzionamento. Gli unici pacchetti da scaricare "a mano" e da integrare con i sorgenti di XCat sono quelli relativi alla gestione delle comunicazioni di tipo SSL (Secure Socket Layer). Quest'ultima cosa non dipende tanto da XCat quanto dalla JVM utilizzata che non supporta in modo nativo (a differenza delle versioni ≥ 1.4) le comunicazioni di tipo sicuro.
Una volta in possesso del pacchetto *tgz*, è necessario scompattarlo all'interno di una directory con il seguente comando:

```
tar xzf XCat-2.0.2-RC1.tgz
```

Successivamente è necessario scaricare il pacchetto che aggiunge alla JVM 1.3 la possibilità di eseguire le comunicazioni di tipo SSL. È sufficiente aprire un browser e ridirigerlo verso l'indirizzo `http://java.sun.com/products/jsse/index-103.html`. Da qui potete facilmente raggiungere la sezione download e ottenere il pacchetto con i *jar* necessari. Dopo averlo salvato localmente sulla vostra macchina potete scompattarlo in una directory temporanea e quindi copiare tutti i file contenuti all'interno dell'archivio nella directory `$JAVA_HOME/jre/lib/ext/` (dove `JAVA_HOME` è la directory radice del SDK di Java). A questo punto non rimane altro da fare che compilare il framework XCat. Posizionarsi all'interno della directory dove precedentemente è stato scompattato il pacchetto *tgz* ed eseguire:

```
./make.sh
```

Il comando costruirà la libreria e posiziozerà tutti i *jar* creati all'interno della directory `build`.

5 Test e valutazione prestazioni

Per valutare le prestazioni del framework abbiamo svolto una serie di test mirati a misurare i parametri critici di un ambiente per applicazioni high-performance. L'architettura hardware/software utilizzata per svolgere gli esperimenti è stata un cluster di workstation con le seguenti caratteristiche:

- cluster di workstation Linux (con RedHat 9) formato da 1 nodo front-end e da 8 nodi di lavoro[4]. Ciascun nodo è caratterizzato da:

[3] Componenti che vengono spesso utilizzati all'interno dell'applicazione o di cui si ignora, a tempo di compilazione, la locazione fisica.

[4] Il nodo front-end è utilizzato per accedere al cluster dall'esterno mediante connessioni di tipo SSH. Su questo nodo vengono sviluppate (compilate) le varie applicazioni da testare mentre il testing vero e proprio viene solitamente eseguito (come nel nostro caso) sui nodi di lavoro.

  - 2 processori Intel Pentium Xeon a 2 Ghz e 1 Gbyte di memoria RAM;
  - Rete dati dedicata[5] a 100 MBit;
- Globus 2.0[6] come ambiente runtime di griglia;
- JDK 1.3.1_09 della Sun come ambiente di runtime Java.

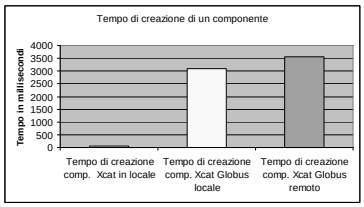
Nel proseguo della sezione mostreremo e discuteremo i principali risultati ottenuti dalle valutazioni sperimentali.

Tempo di accesso ai componenti

In un ambiente high-performance come XCat possiamo individuare sostanzialmente tre operazioni critiche in termini di tempo relativamente alla creazione/accesso di un particolare componente. La prima riguarda il tempo di creazione ad una specifica locazione di una nuova istanza di un componente. La seconda operazione invece è quella di misurare il tempo per ottenere un identificatore univoco di un componente che sappiamo gia essere stato creato in precedenza. L'ultima operazione critica è il tempo necessario per eseguire la chiamata di un metodo (o servizio) esportato da un componente. Nei nostri test tutte le operazioni sopra descritte sono state considerate ed i risultati ottenuti vengono mostrati di seguito.

In figura 4 a) è stato misurato il tempo di creazione di un nuovo componente per varie configurazioni[7]. La prima configurazione riguarda un componente creato in locale utilizzando il meccanismo *proc*, la seconda un componente creato in locale utilizzando il meccanismo *gram*, la terza un componente utilizzando sempre il meccanismo *gram* ma creandolo su una macchina remota rispetto all'applicazione considerata. Dai risultati sperimentali si nota che il tempo di creazione utilizzando *gram* è molto alto, anche quando i componenti sono creati in locale. Non è chiaro se il problema dipende dall'implementazione di XCat o dallo strato di Globus sottostante. In ogni caso si tratta di overhead molto significativi che inficiano notevolmente sulle caratteristiche di high performance a cui XCat ambisce.

a) Tempo necessario per istanziare un nuovo componente



b) Tempo di lookup di un componente esistente e gia registrato all'interno del Naming Service

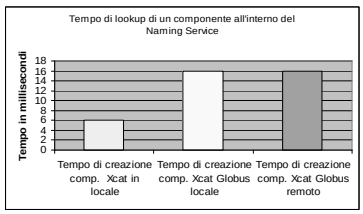


Figura 4: Tempo di creazione e di lookup di un componente

Nel grafico in figura 4 b) viene invece mostrato il tempo di accesso al Naming Service per ottenere il riferimento ad un componente esistente. In questo caso, invece, si ottengono tempi abbastanza bassi e pressoché identici in tutte e tre le configurazioni (le stesse del caso precedente). Questo comportamento è ragionevole e dovuto al fatto che il NamingService è un processo separato che deve essere interrogato. In queste condizioni l'unico overhead può essere dovuto al tempo di trasmissione della richiesta sulla rete ma visto che ci aspettiamo un numero di byte trasferiti piccolo e che la rete è dedicata e abbastanza veloce, possiamo considerare che questo tempo sia trascurabile.

In figura 5 infine viene mostrato il grafico relativo al tempo necessario per invocare un metodo su un certo componente. Osserviamo che, come nei grafici precedenti, sono stati considerati le "solite" tre configurazioni. Anche in questo caso i risultati lasciano un

[5]Solo gli 8 nodi interni.

[6]Non è stata utilizzata l'ultima versione disponibile di Globus (la 3.0 al momento della scrittura dell'articolo) perchè la versione utilizzata di XCat era compatibile solo con la release 2.0.x.

[7]Per maggiori dettagli sulle varie configurazioni fate riferimento alle descrizioni contenute nella sezione 3.2 a pagina 5.

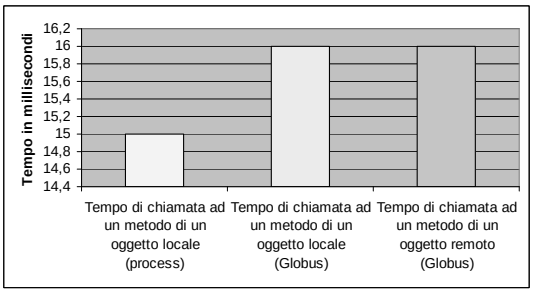


Figura 5: Tempo di chiamata di un metodo

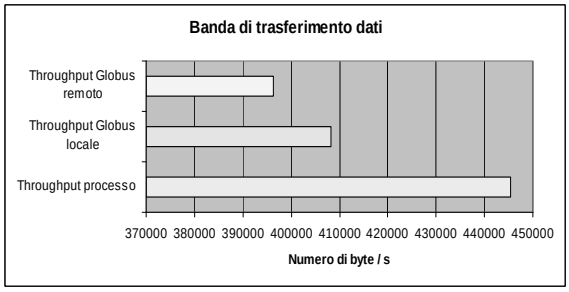


Figura 6: Banda di trasferimento dati tra due componenti

po' perplessi perché abbiamo ottenuto tempi praticamente equivalenti in tutte e tre i casi, dove invece ci saremmo aspettati per la prima configurazione (considerata la località del componente) tempi di gran lunga più bassi a quelli degli altri due casi.

Throughput nel trasferimento di dati tra due componenti

Per valutare la banda di trasferimento dati tra due componenti abbiamo sfruttato il meccanismo standard delle porte uses/provides invocando un metodo vuoto (che non faceva nulla!) su un certo componente, passando un parametro dalle dimensioni note (di 2000000 di byte) e misurando il tempo necessario per invocare il metodo stesso. I risultati ottenuti sono riportati in figura 6 e considerano le stesse configurazioni viste in precedenza. In questo caso le prestazioni sono veramente disastrose sia per le configurazioni locali che quelle remote. I pessimi risultati crediamo possano essere in parte dovuti alla cattiva implementazione della parte RMI del framework che in particolare non ottimizza le chiamate per componenti tra loro locali (in esecuzione sulla stessa macchina). Un altro fattore negativo potrebbe essere il fatto che libreria è implementata in Java e come è noto i programmi eseguiti sulle JVM non sono molto efficienti.

Scalabilità di una applicazione "intensive"

Per misurare i valori di scalabilità abbiamo realizzato un'applicazione basata sul modello di farm presentato in figura 7. Il processo è composto da un componente *emittitore* che ha il compito di passare i pacchetti da elaborare (prelevabili dalla coda dei lavori "Task list") ai generici *worker*. La richiesta di nuovi pacchetti da elaborare deve essere fatte direttamente dai worker all'emettitore. La simulazione dell'elaborazione di un pacchetto è fatta con un codice del tipo di quello rappresentato nel riquadro arancione della figura. Una volta che un worker ha terminato di elaborare un pacchetto di lavoro deve provvedere a trasmettere i risultati ottenuti ad un componente *collettore*. Infine, l'applicazione terminerà quando non vi saranno più pacchetti da elaborare.
I test sono stati svolti facendo variare alcuni parametri. Il numero dei pacchetti da elaborare è stata

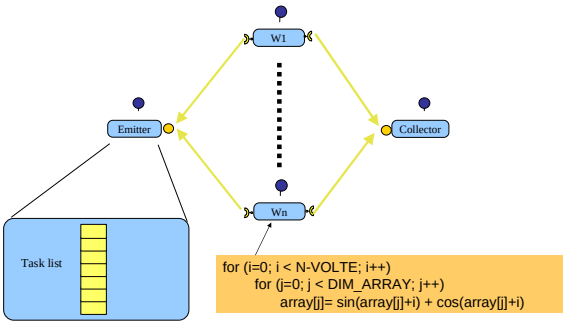
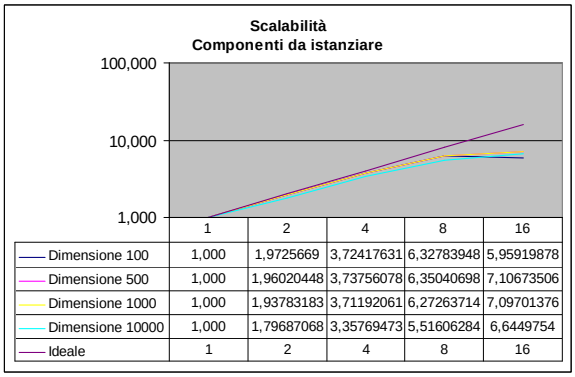


Figura 7: Il modello di farm utilizzato per misurare la scalabilità di un'applicazione



	1	2	4	8	16
Dimensione 100	1,000	1,9725669	3,72417631	6,32783948	5,95919878
Dimensione 500	1,000	1,96020448	3,73756078	6,35040698	7,10673506
Dimensione 1000	1,000	1,93783183	3,71192061	6,27263714	7,09701376
Dimensione 10000	1,000	1,79687068	3,35769473	5,51606284	6,6449754
Ideale	1	2	4	8	16

Figura 8: Scalabilità per l'applicazione farm di esempio

calcolato utilizzano la formula

$$N_{pacchetti} = \frac{N_{puntiTotali}}{N_{puntiPerPacchetto}}$$

dove $N_{pacchetti}$ è il numero di pacchetti totale da elaborare, $N_{puntiTotali}$ è il numero totali di punti da elaborare nella simulazione e $N_{puntiPerPacchetto}$ è il numero di punti da elaborare contenuto in ogni singolo pacchetto. Abbiamo fissato il numero totale di punti da trattare per ogni singola simulazione ($N_{puntiTotali}$=800000) e abbiamo fatto variare gli altri due parametri. In questo modo abbiamo ottenuto un mix di configurazioni di testing che spaziavano tra I/O-bound e CPU-bound.

In figura 8 vengono riportati e tracciati i valori di scalabilità relativi alle configurazioni provate. Per semplicità, i componenti emettitore e collettore sono stati eseguiti sulla stessa macchina. Inoltre, si assume che i tempi riportati tengono anche di conto del tempo di creazione dei vari worker sulle macchine remote.

I test sono stati eseguiti fino ad un grado di parallelismo di 16 (distribuendo due worker per ciascun nodo di lavoro del cluster utilizzato) e le varie configurazioni riportate in figura con denominazione "Dimensione x" corrispondono a simulazioni nelle quali la dimensione dei pacchetti è x.

I risultati mostrano che siamo riusciti ad ottenere valori di scalabilità mediocri gia a partire da configurazioni con grado di parallelismo di 8 e che anche variando la dimensione dei pacchetti da elaborare non si ottengono miglioramenti significativi. I valori negativi ottenuti sono concordi con le performance evidenziate nei grafici precedenti e denotano una generale inefficienza del sistema in tutte le condizioni.

6 Conclusioni

In questo articolo abbiamo analizzato il framework XCat, una libreria a componenti per costruire applicazioni ad alte prestazioni in ambienti distributi/grid. Il framework è basato sulla specifica CCA e quindi utilizza il modello delle porte uses/provides per comporre i componenti tra loro. Questa caratteristica permette di modellare in modo naturale e semplificato applicazioni anche di una certa complessità. Il sistema inoltre fornisce all'utente/programmatore una serie di servizi ad alto livello ed una interfaccia di programmazione scripting che permettono di scrive-

re applicazioni di controllo in modo facile e veloce.
Purtroppo però non sono tutte rose e fiori. Infatti, dai nostri test emerge che l'obiettivo principale della libreria (quello di fornire un ambiente high-performance) non è stato raggiunto. I problemi riscontrati sono a vario livello e riguardano l'efficienza in generale del framework in qualsiasi condizione. Non è chiaro se questo dipenda dall'implementazione specifica di XCat oppure dalla scarsa efficienza di qualcuno dei tool (molto numerosi, per la verità) utilizzati direttamente del framework. Inoltre, va osservato che il supporto di griglia, oltre che inefficiente, è ancora piuttosto limitato e non da la possibilità di avere un controllo esplicito sulle operazioni eseguite. Un'altra nota dolente dell'implementazione attuale di XCat è che le interfacce delle porte possono essere specificate solo attraverso l'uso di classi Java. Questo limita l'utilità dello strumento nella creazione di applicazioni reali per almeno due motivi. Il primo è che i metodi esportati da una porta non possono fare uso di alcuni tipi di dato complessi generalmente usati in software scientifici (ad esempio numeri complessi, array dinamici multidimensionali, ecc.). L'altro aspetto negativo è che non è possibile utilizzare direttamente oggetti CCA scritti in altri linguaggi di programmazione e quindi siamo "limitati" ad utilizzare solo quelli scritti in Java. La limitazione di dover scrivere le interfacce in Java sembra per la verità più un problema di Babel (il compilatore SIDL utilizzato) che di XCat ma comunque resta il fatto che è un problema in futuro da risolvere per poter fornire uno strumento da poter utilizzare in maniera proficua.

Riferimenti bibliografici

[1] The Common Component Architecture Technical Specification - Version 0.5. http://cca-forum.org/bindings/old-0.5/.

[2] The XCat framework, a CCA-compliant library for distributed computing. http://www.extreme.indiana.edu/xcat/.

[3] Rob Armstrong, Dennis Gannon, Al Geist, Katarzyna Keahey, Scott R. Kohn, Lois McInnes, Steve R. Parker, and Brent A. Smolinski. Toward a common component architecture for high-performance scientific computing. In *HPDC*, 1999.

[4] The CCA forum home page. http://www.cca-forum.org/.