movimento italiano modellazione e simulazione

3° Convegno Tecnico Scientifico

Torino 3, 4, 5 Novembre 2003

A grid computing-based architecture for on demand movie rendering

Griglie di calcolo per il Ray-Tracing

Mario Pissardo
Politecnico di Torino, DAUIN
mario.pissardo@polito.it

Yvonne Carpegna
Politecnico di Torino, DAUIN
yvonne.carpegna@virgilio.it

Bartolomeo Montrucchio
Politecnico di Torino, DAUIN
montru@polito.it

Andrea Sanna
Politecnico di Torino, DAUIN
sanna@polito.it

Claudio Demartini
Politecnico di Torino, DAUIN
demartini@polito.it

ABSTRACT

In questi ultimi anni stanno acquisendo sempre più importanza sistemi in grado di fornire elevate performance utilizzando tecniche di elaborazione distribuita, soprattutto nelle applicazioni caratterizzate dall'utilizzo di ingenti quantità di risorse e da considerevoli tempi di elaborazione.

Il grid computing, ad esempio, può essere utilizzato in numerosi settori dell'elaborazione, dal data mining al calcolo distribuito. Nonostante gli evidenti vantaggi, in ambito industriale questo tipo di soluzione è ancora poco diffuso. Un settore che può trarre grandi benefici dall'adozione di tale architettura è sicuramente quello della computer grafica applicata al rendering di scene tridimensionali complesse.

In questa presentazione verrà proposta un'architettura che utilizza il grid computing per realizzare il rendering on-demand di filmati multimediali consentendo anche a settori esterni alla ricerca di poter usufruire dei vantaggi derivanti dall'impiego di queste tecnologie. Tramite un'applicazione Web vengono completamente resi trasparenti aspetti complessi quali la condivisione di risorse, l'elaborazione parallela tra più cluster e la sicurezza tipici di un ambiente distribuito. L'architettura proposta presenta, infatti, un'interfaccia semplice ed intuitiva per la sottomissione dei parametri e della scena da renderizzare, notificando successivamente all'utente, tramite l'invio di una mail, il completamento dell'operazione e la disponibilità del filmato per il download.

Particolare attenzione è stata rivolta alla scelta di componenti software open-source che permettono sia di ridurre sensibilmente i costi relativi alle licenze sia di avere il controllo completo delle funzionalità offerte dal software, il tutto senza rinunciare a qualità e performance.

Introduzione

Il processo di *rendering* di animazioni computerizzate complesse e di qualità elevata richiede una notevole potenza di calcolo e spesso lunghi periodi di elaborazione. Gli algoritmi di *rendering* utilizzati [1], potendo generare *frame* in modo indipendente, si rivelano particolarmente adatti a sfruttare i benefici introdotti da architetture distribuite in grado di parallelizzarne il procedimento. Per velocizzare ulteriormente il processo di *rendering*, si possono inoltre utilizzare particolari tecniche che sfruttano il concetto della *correlazione*, ad esempio temporale, per realizzare in modo concorrente e parallelo le scene. Per queste ragioni, negli ultimi anni, sono state presentate numerose soluzioni di architetture distribuite e parallele specifiche per questo settore. Una classificazione delle principali tecniche di *rendering* parallelo può essere trovata in [2] ed in [3]. Tra i progetti recentemente proposti, è importante citare *Chromium* [4] mentre numerosi prodotti commerciali, come ad esempio 3D Studio Max [http://www.3dmax.com], Maya [http://www.aliaswavefront.com] ed altri, sfruttano l'elaborazione distribuita su reti locali per migliorare le *performance*. Altre importanti iniziative sono: *Entropia* [http://www.entropia.com] che raccoglie e sfrutta la potenza di elaborazione dei PC non utilizzati di uffici e dipartimenti, e *Butterfly* [http://www.butterfly.net] il primo prodotto *on-line*, basato su tecnologia *grid*, per l'industria dei video-game. Quest'ultimo sviluppato con l'aiuto di *IBM* ed utilizzando il *framework open-source* del *Globus Project*, gestisce contemporaneamente moltissimi giochi ciascuno dei quali con milioni di giocatori sparsi per tutto il mondo.

Il prototipo da noi proposto [5] è basato anch'esso su tecniche di *grid computing* per la realizzazione di un'architettura distribuita per il rendering [6] (come per il progetto *Entropia)*, ed utilizza esclusivamente software open-source (come parzialmente fatto da *Butterfly*). Prima di descrivere il progetto è opportuno definire brevemente cos'è una *grid*. Come sottolineato recentemente da Ian Foster [7], uno dei fondatori del grid computing, non esiste una definizione unica di *grid* ma essa deve essere definita in termini di applicazioni [8], risultati scientifici e benefici per il mondo business e non solo come una mera architettura tecnologica. Una *grid* deve garantire agli utenti un *accesso trasparente* a risorse distribuite geograficamente tramite un insieme ben definito di servizi. Il termine *risorsa* deve essere inteso nel senso più generico possibile, includendo: computer, archivi, strumenti ed apparecchiature *on-line*, ecc. Questa tecnologia viene infatti impiegata per gestire specifici tipi di risorse ed applicazioni; per esempio, un insieme eterogeneo di elaboratori che fornisce, in modo collaborativo, servizi di elaborazione distribuita, viene indicato con il termine di *computational grid* [9]. In questa presentazione viene infatti mostrata una soluzione, presentata per la prima volta in [5], che sfrutta questo concetto per effettuare il *ray-tracing* di sequenze di *frame*.

Esistono altri tipi di grid come il *data-grid*, pensato per la condivisione e la gestione di archivi di grandi quantità di dati, ed il *service-grid*, progettato per fornire servizi complessi e non compatibili con le risorse di una singola macchina, come ad esempio certe applicazioni multimediali. Una rigorosa ed esaustiva classificazione delle architetture grid si può trovare in [10]. Solo recentemente le tecnologie *grid* sono state sfruttate per applicazioni di *rendering* ad alta qualità; in particolare, *Sun Microsystems* [http://www.sun.com] e *Side Effects Software* [http://www.sidefx.com] hanno presentato una versione distribuita del software *Houdini* durante la conferenza *SIGGRAPH 2002* [6].

Il nostro progetto propone un'architettura per il *rendering* distribuito ad alte performance in grado di elaborare sequenze di *frame on-demand*. Il sistema presenta un'interfaccia Web molto intuitiva che consente agli utenti di accedere ai servizi grid e sottomettere le scene del filmato da *renderizzare*. Il principale aspetto innovativo di questa soluzione rispetto ai prodotti commerciali indicati precedentemente, è l'uso di solo software *open-source* per la realizzazione dell'infrastruttura di elaborazione.

Il Globus Toolkit [http://www.globus.com][11] è stato scelto come *framework* di sviluppo per la realizzazione dell'architettura *grid*, mentre per il *rendering* delle animazioni sono state utilizzate le librerie di *ray-tracing* di *Pov-ray*. La *computational grid*, da noi realizzata, coordina opportunamente un insieme di *cluster* indipendenti, distribuiti ed eterogenei consentendo agli utenti di sottomettere le sequenze da elaborare e di ricevere la notifica per il *download* del filmato finale. L'architettura gestisce in modo trasparente la complessità delle interazioni tra i cluster su cui è installato *Pov-Ray*, come ad esempio le procedure di autenticazione e sicurezza, la schedulazione dei processi e la ricomposizione del filmato a partire dalle singole elaborazioni distribuite.

Le sezioni successive descrivono in dettaglio l'architettura proposta, sottolineando le problematiche incontrate durante l'implementazione e mostrando il modo in cui i vari componenti software/hardware sono stati organizzati. Infine, sono indicati i risultati sperimentali ottenuti ed i possibili sviluppi futuri.

Architettura proposta

Come detto precedentemente, le architetture per il *rendering* di filmati, come ad *esempio 3D Studio Max*, sono spesso proprietarie. Inoltre, questi prodotti sono normalmente utilizzati solo se le scene da elaborare non sono troppo complesse e non richiedono strumenti hardware/software particolari. L'ambiente in cui operano, infatti, è limitato dalle risorse a disposizione, solitamente pochi *personal computer*, sui quali parallelizzare la fase di *rendering*. Non è possibile, ad esempio, condividere risorse che non appartengono alla stessa *LAN* (Local Area Network).

Queste ed altre motivazioni ci hanno spinto a progettare un'architettura in grado di condividere efficacemente risorse appartenenti a reti diverse ed indipendenti. Il software *open-source* utilizzato, inoltre, garantisce un elevato livello di sicurezza pur mantenendo basso il valore del *TOC* (Total Cost of Ownership). La scelta di utilizzare tecnologie *grid* per questo progetto è stata anche motivata dalla semplicità di sviluppo del *framework Globus*, utilizzabile per la realizzazione di molte applicazioni verticali, e dalla possibilità di integrazione con soluzioni *web-based*. Questo tipo di applicazione può così essere considerata *interna* ad una singola azienda, quando la potenza di calcolo viene fornita da un dipartimento ad un altro, oppure *globale*, quando le risorse appartengono a sedi aziendali localizzate in città differenti o quando questo tipo di collaborazione viene realizzata tra organizzazioni differenti (caso difficilmente realizzabile utilizzando software proprietario). Inoltre, utilizzando un'applicazione *web-based* si ottengono i seguenti vantaggi:

- semplicità di accesso alle risorse da parte degli utenti;
- separazione completa tra l'interazione tra le risorse della *grid* e gli utenti finali; è possibile infatti creare organizzazioni dinamiche che forniscono potenza di calcolo ai loro clienti in funzioni delle specifiche esigenze.

L'idea base dell'intera architettura è quindi fornire potenza di elaborazione *on-demand* agli utenti in modo semplice ed intuitivo. Questa soluzione vuole infatti mostrare come le tecnologie *del grid computing* possano integrarsi efficacemente con i servizi aziendali, dapprima tramite una semplice interfaccia *web*, per poi evolvere direttamente in un *web service* a disposizione delle varie applicazioni aziendali. Uno schema ad alto livello dell'architettura è mostrato in fig.1 in cui si evidenza come, dal punto di vista dell'utente, l'operazione di *rendering* sia fornita completamente *on-demand* da un server Internet. L'utente, infatti, invia al server i propri file da elaborare utilizzando un comune *browser* e, quando l'elaborazione si è conclusa (viene informato tramite l'invio di una mail), può effettuare il download del filmato finale. Come si può vedere dalla fig.1, le comunicazioni con il *server web* avvengono tramite connessioni protette (HTTPS) e, sfruttando i servizi base del framework *Globus*, vengono inoltre garantite anche tutte le interazioni tra cluster e tra questi e l'application server. Si deve notare infine come questa soluzione, che separa il fornitore della potenza computazionale da chi la utilizza, permetta di gestire il *payment* del servizio direttamente tramite il server web, senza coinvolgere gli amministratori dei cluster.

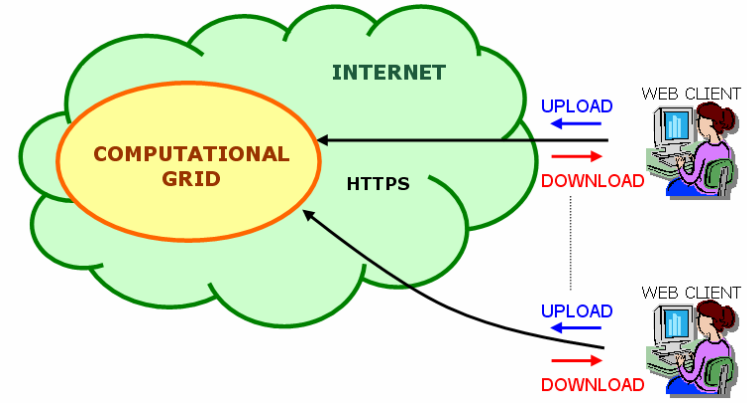

Figura 1 Schema ad alto livello dell'architettura proposta

In fig.2 si evidenza la struttura della *computational grid* che nella figura precedente era inglobata (dal punto di vista dell'utente) in Internet.

L'unico punto di ingresso/uscita della *grid* è il collegamento tra il *web server* ed i *browser* dei client. In questo modo l'intera *grid* può essere vista come un'unica rete in cui le risorse cooperano tra di loro. In particolare, in fig.2 si possono vedere le funzioni specifiche dei componenti progettati. L'elemento che interagisce direttamente con i client tramite richieste HTTPS è, come già sottolineato, il *web server* che si occupa anche di interrogare il *grid broker*. L'unico punto di ingresso/uscita della grid è il collegamento tra il *web server* ed i *browser* dei client. In questo modo l'intera *grid* può essere vista come un'unica rete in cui le risorse cooperano tra di loro. In particolare, in fig.2 si possono vedere le funzioni specifiche dei componenti progettati. L'elemento che interagisce direttamente con i client tramite richieste HTTPS è, come già sottolineato, il *web server* che si occupa anche di interrogare il *grid broker*.

L'unico punto di ingresso/uscita della grid è il collegamento tra il *web server* ed i *browser* dei client. In questo modo l'intera *grid* può essere vista come un'unica rete in cui le risorse cooperano tra di loro.

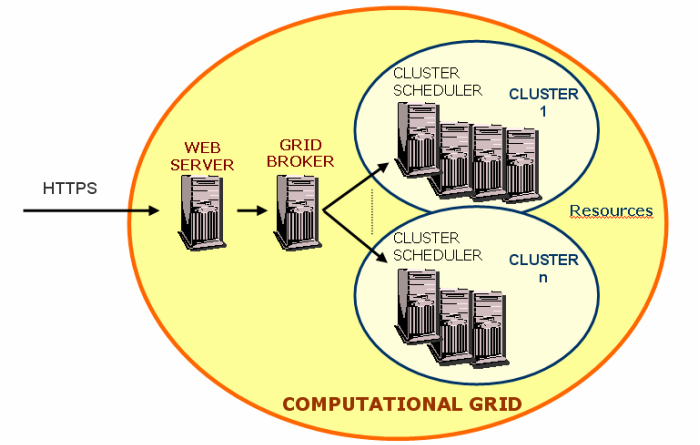

Figura 2 Computational grid internal architecture

In particolare, in fig.2 si possono vedere le funzioni specifiche dei componenti progettati. L'elemento che interagisce direttamente con i client tramite richieste HTTPS è, come già sottolineato, il *web server* che si occupa anche di interrogare il *grid broker*. Poiché si vuole effettuare il *rendering* di singoli *frame*, che poi andranno a comporre il filmato finale, e non è semplice conoscere a priori la complessità di un *frame* (anche se esistono tecniche a riguardo), il *grid broker* sottometterà un certo numero di *frame* (considerando la loro complessità costante) a ciascun *cluster* solo in funzione della disponibilità di risorse libere in quel momento. Affinché questa tecnica sia efficiente e si evitino colli di bottiglia nella rete è necessario che l'elaborazione dei singoli *frame* richieda un tempo di calcolo superiore a quello necessario per la schedulazione e la ricezione dei risultati, condizione normalmente soddisfatta.

Implementazione

L'architettura descritta nella sezione precedente è stata successivamente implementata in modo da verificare su di un prototipo reale la correttezza delle scelte intraprese. Nella fig.3 vi è un ulteriore dettaglio della realizzazione.

Per trasferire i file sorgenti al *server web* si utilizza un comune *browser* (ad esempio Opera 6 per Windows e Netscape 6 per Linux). Sia il *web server* (si è scelto Apache, sicuramente il più diffuso e veloce su piattaforma Linux) sia le applicazioni client del framework **Globus** sono state installate su di un server ad alta disponibilità ed affidabilità (si è scelto un server Webvision della Flextel [http://www.flextel.it]). La configurazione hardware del server è la seguente: *dual Pentium III 800MHz* con *1GB di RAM* e *36GB* di spazio su disco (RAID 0). Il sistema operativo è *Linux* (RedHat 7.2) ed il framework per l'implementazione della computational *grid* è *Globus Toolkit 2.0* (la versione 3.0 non era ancora disponibile all'inizio del progetto), largamente diffuso e testato che assicura servizi di qualità elevata.

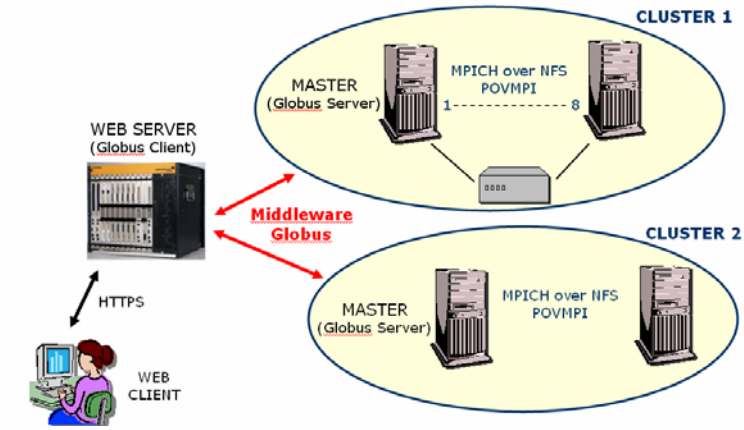

Figura 3 Architettura implementata

Per quanto riguarda la sicurezza ad esempio, *Globus* fornisce, tra i suoi servizi di *core*, il modulo *GSI* (Globus Security Infrastructure) che gestisce le autorizzazioni e gli accessi utilizzando un'infrastruttura a chiave pubblica basata su X.509.v3 e sui protocolli SSL(Secure Socket Layer Protocol)/TLS(Transport Layer Security) per l'autenticazione. *GSI* definisce inoltre alcune estensioni per la generazione e la gestione dei *certificati proxy*, molto utilizzati per la realizzazione di scenari *grid*. Un altro modulo dei servizi di *core* è *GASS* (Globus Access to Secondary Storage) che consente di trasferire eseguibili e file di input/output tra i nodi della grid. Nella nostra architettura, questo modulo viene utilizzato dal *grid broker* per distribuire i file *Pov-Ray*, organizzati in un unico file compresso, tra i *cluster* della *grid*. Il livello di sicurezza in questi trasferimenti è garantito dai servizi del *GSI*.

Su ciascun cluster i dati ed i file sono distribuiti tramite NFS ed elaborati in concorrenza dalle librerie di *PovMPI* (versione distribuita di *Pov-Ray* che utilizza MPI) e *MOSIX* [http://www.mosix.org]. *MOSIX* infatti, pur non avendo un sistema a code, consente tramite tecniche di migrazione dei processi, di ottenere il bilanciamento di carico dinamico anche nella fase successiva alla schedulazione. La versione di *PovMPI* utilizzata per il *rendering* dei *frame* è basata su *PovRay 3.1g* siccome, al momento in cui il progetto è iniziato, essa era l'unica a fornire la versione distribuita per MPI.

Il primo cluster, come riportato in fig.3, è composto da 8 *personal computer*: 4 bi-processori *AMD Athlon 1600+* con *512 MB di RAM* e 4 mono-processori *Athlon 1600+ con 512 MB di RAM*. I computer sono connessi in una LAN tramite uno *switch 3Com Gigabit* a 24 porte. Il secondo cluster è composto, invece, da due *Pentium III 733 mono-processore con 256 MB di RAM* connessi con uno *switch Fast Ethernet*. I due cluster sono connessi tra loro e con il *server Flextel* tramite una LAN Ethernet (10Mbit/s). L'asimmetria tra i due cluster è stata utilizzata per mostrare come l'architettura proposta possa ottenere un buon bilanciamento del carico anche in presenza di cluster con potenza computazionale e connessioni di rete molto differenti.

Quando tutti i *frame* sono stati *renderizzati*, si procede a comprimerli in un unico file e successivamente a realizzare il filmato tramite un codificatore *MPEG-4 open-source* (*animmaker* 0.2.5 per la generazione del file AVI e *mencoder* per la codifica in MPEG-4). A questo punto l'utente può scaricare il filmato, sempre utilizzando i servizi di sicurezza forniti dal GSI. Nella fig.4

è riportata l'interfaccia di download (quella di upload è analoga) con la quale l'utente può anche verificare lo stato dell'elaborazione.

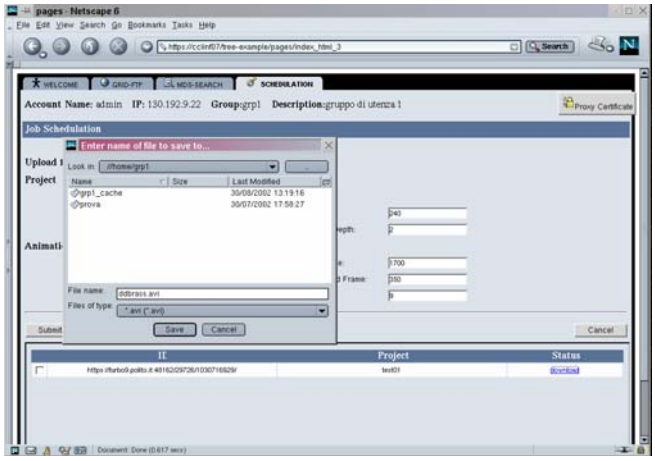

Figura 4 Screenshot dell'interfaccia di download

L'applicativo web è stato realizzato utilizzando il *CMF* (Content Management Framework) fornito da *Zope* [http://www.zope.org]. La scelta di questo strumento è stata guidata dall'ottima integrazione tra le librerie di *Globus* ed il linguaggio *Python* (tramite un modulo *CoG* – Commodity Grid) con cui *Zope* è scritto, riducendo in questo modo l'overhead complessivo. Inoltre *Zope Corporation* è una delle aziende in crescita e che riesce a produrre business tramite software *open-source*. In fig. 5 è mostrata la struttura a livelli dei software utilizzati sia lato client sia lato server.

L'idea base era creare un'interfaccia web all'applicazione in modo da semplificare al massimo la complessità della struttura *grid* sottostante e rendere l'applicazione stessa più vicina alle esigenze di un'utenza generica. Nella prossima sezione vedremo in particolare come la semplicità funzionale non vada a discapito della potenza computazione e delle performance ottenute.

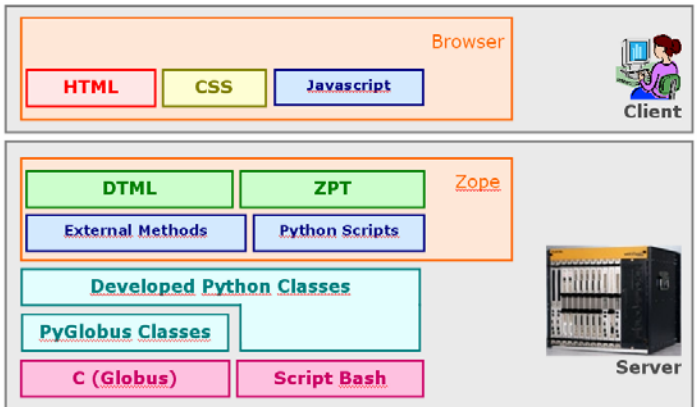

Figura 5 Linguaggi di programmazione utilizzati

Risultati sperimentali

L'architettura sviluppata è stata testata utilizzando uno scenario abbastanza complesso. I *frame* da elaborare sono stati presi da "*The Internet Raytracing Competition*" [http://irtc.og], una competizione internazionale di grafica 3D. In particolare, si è scelta la scena giunta al terzo posto nell'edizione Gennaio-Aprile 2002, siccome utilizzava *Pov-Ray 3.1g* ed i sorgenti erano liberamente disponibili. Il nome del lavoro di computer grafica di *Daniel Dressel è "Brass and Steel*" [http://www.irtc.org/anims/2002-04-15.html], da cui abbiamo estratto come test un centinaio di frame tra i più complessi. L'autore sostiene che l'intera scena ha richiesto circa 50 ore di elaborazione su di un *Athlon 1200*). Un esempio dei frame renderizzati è mostrato in fig.6.

Hardware utilizzato (tipo e numero di CPU)	**Tempo di elaborazione** (hh:mm:ss)
tutta la Grid (12x Athlon 1600+ e 2x PIII 733)	0:16:49
mono-processone Athlon 1600+	3:22:46

Tabella 1: Risultati ottenuti; 100 frame di 320X200; lo speed-up per la grid è di 12.05, mentre il rapporto rispetto all'elaborazione su PC singolo è pari a circa 13

La prima configurazione è formata dall'impiego contemporaneo di tutte le risorse disponibili nella *Grid*. Come già indicato nella sezione precedente, si sono realizzati due cluster; uno composto da otto *personal computer Athlon* di cui 4 bi-processore, e l'altro formato da due mono-processore *Pentium III 733MHz*. Questa asimmetria è stata volutamente introdotta per valutare gli effetti di caratteristiche hardware differenti all'interno della *grid*. La seconda configurazione testata, invece, utilizza un singolo elaboratore di media potenza (mono-processore *Athlon 1600+*) utilizzato normalmente da chi effettua rendering. Il rapporto approssimato tra i tempi impiegati dalla seconda configurazione rispetto alla prima è di circa 13:1. Dalla Tabella 1 si può notare che lo *speed-up* complessivo della *grid* sia di 12.05, valore molto soddisfacente. Dai risultati ottenuti si può dedurre che anche una semplice grid, come quella da noi realizzata, può essere molto utile in presenza di rendering complessi in quando consente di ottenere una riduzione quasi lineare del tempo di elaborazione complessivo.

L'architettura proposta è attualmente un prototipo e saranno necessari ulteriori test con un numero di *cluster* maggiore per poter individuare eventuali colli di bottiglia. Il primo possibile punto critico del sistema è la struttura *master/slave*; anche se i nostri test non lo hanno mostrato, un numero molto maggiore di *frame* potrebbe evidenziare il problema. Una soluzione potrebbe essere l'utilizzo contemporaneo di più *server* con funzionalità di *grid broker*, scenario semplicemente realizzabile all'interno dell'architettura realizzata.

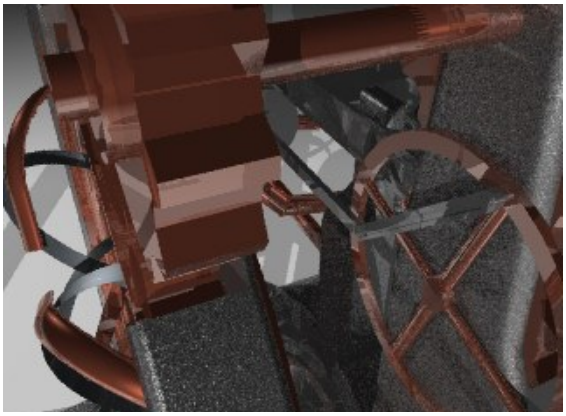

Figura 6 Esempio di frame renderizzato

L'utilizzo di un server che garantisce alta affidabilità, disponibilità e throughput limita ulteriormente i problemi di scala. *PovMPI* è un buon motore di rendering parallelo e non genera un collo di bottiglia nelle comunicazioni di rete, siccome il tempo di rendering è solitamente molto superiore al tempo di trasferimento. Nel nostro test infatti utilizziamo una risoluzione di 320x200, che richiede circa 2 minuti per ciascun frame. In scenari reali la risoluzione impiegata e conseguentemente i tempi di elaborazione sono simili o addirittura maggiori riducendo ulteriormente questo rischio. Un altro problema potrebbe essere il bilanciamento del carico dovuto alla stima della complessità dei frame. In particolari condizioni, non verificatesi nei nostri test, potrebbero esserci brevi intervalli di tempo in cui le macchine hanno carico differente. Ulteriori test verificheranno se problemi di questo tipo possano insorgere. Un ultimo problema potrebbe essere legato al tempo dovuto alla compressione del filmato, anche se nel caso specifico si è dimostrato del tutto irrilevante (inferiore a 10 secondi).

Questo tipo di software e di architettura può quindi essere utilizzato in piccole società di rendering garantendo, a fronte di un modesto investimento, notevoli benefici derivanti dall'impiego delle nuove tecnologie

Conclusioni

I sistemi per l'elaborazione distribuita ad alte performance, conosciuti anche con il termine *computational grid*, stanno acquisendo sempre maggiore importanza soprattutto per applicazioni caratterizzate da tempi di elaborazione molto lunghi. Noi abbiamo proposto un'architettura di *grid computing* per la realizzazione di filmati *on-demand* che, tramite una semplice ed intuitiva interfaccia *web*, consentisse la massima astrazione e trasparenza nei confronti della complessità dell'infrastruttura. Tutto il software utilizzato è *open-source* riducendo così al minimo il *Total Cost of Ownership* dell'applicativo. Questo però non pregiudica l'impiego di software proprietario all'interno dell'architettura progettata qualora esso consentisse migliori prestazioni.

Attualmente l'implementazione dell'architettura è da considerarsi un prototipo ed è stato testato solamente su una *grid* ridotta (solo due *cluster*), anche se la struttura ed i protocolli usati dovrebbero garantire la possibilità di usufruire di un numero molto superiore di risorse. Una futura fase di test consentirà di individuare eventuali limiti strutturali verificandone l'impiego anche su

grid estese. Un possibile sviluppo sarà l'inserimento ed il test di altri motori di *rendering*. I test riguarderanno anche la possibilità di verificare la scalabilità del sistema su cluster di dimensioni più cospicue; lo scopo è infatti quello di utilizzare un cluster formato da 32 macchine.

Questo tipo di soluzione prova a fondere in un'unica architettura l'approccio proprietario *master/slave* di *Entropia* con progetti *open-source* per il *grid computing* come *Globus,* mostrando come quest'ultima tipologia di software possa essere impiegata anche a livello produttivo aziendale.

Il lavoro proposto può quindi essere considerato un primo passo verso l'uso di *soluzioni open-source* di *grid computing* in applicazioni produttive avendo curato molto approfonditamente, oltre alle performance, anche problemi di sicurezza e di autorizzazione.

Ringraziamenti

Questo lavoro è stato parzialmente sovvenzionato dal progetto *MIUR* (Ministero dell'Istruzione, dell'Università e della Ricerca) "*Metodologie e strumenti per la progettazione, lo sviluppo e la gestione delle reti di telecomunicazione avanzate*" e dal *CERCOM* (Center for Multimedia Radio Communications) del Politecnico di Torino.

Riferimenti

[1] J.D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes. "Computer Graphics, Principles and Practice". 2nd ed. In C, Addison-Wesley, 1997.

[2] S. Molnar, M. Cox, D. Ellsworth, H. Fuchs, "A sorting classification of parallel rendering". IEEE CG & Appl. July 1994 14(4), pp.23-32.

[3] T. W. Crockett, "An introduction to parallel rendering". Parallel Computing July 1997 23(7), pp.819-843.

[4] G. Humphreys et al., "Chromium, a stream-processing framework for interactive rendering on clusters". ACM Trans. on Graphics July 2002 21(3), pp.693-702.

[5] Y. Carpegna, M. Pissardo, B. Montrucchio, A. Sanna "A Grid Computing-Based Architecture for on Demand Movie Rendering", 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003), Orlando, USA, July 27-30, 2003

[6] Systems/Enterprise. "Side Effects & Sun achieved rendering across a compute farm". Grid today 1(7), July 2002.

[7] Ian Foster, "What is the grid? A three point check-list", Grid today 1(6), July 2002.

[8] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations". International Journal of High Performance Computing Applications 15(3).

[9] Geoffrey Fox and Dennis Gannon. "Computational grids" Computing in Science & Engineering 3(4), 2001.

[10] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing", Software-Practice and Experience, 32(2) pp. 135-164, 2002.

[11] Ian Foster, "The Globus Toolkit for Grid Computing". Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid. 15-18 May 2001.