Analisi e miglioramento dell'efficienza di algoritmi di scheduling real-time per architetture multiprocessore

Juri Lelli

26 maggio 2010

Introduzione e scenario di applicazione

I sistemi multiprocessore trovano al giorno d'oggi largo utilizzo nelle più diverse configurazioni hardware, dalle server workstations ai dispositivi mobili di ultima generazione. La sempre crescente domanda in termini di miglioramento delle performance non trova infatti risposta in tecniche volte all'aumento della capacità computazionale del singolo processore [18]. Si cercano invece soluzioni distribuite, ad esempio attraverso applicazioni multithread, che, suddividendo il carico tra le CPU del sistema, riescano a compiere il lavoro richiesto in minor tempo e con un maggior risparmio energetico.

I moderni sistemi operativi devono, dal canto loro, essere in grado di fornire alle applicazioni gli strumenti per decidere in che modo suddividere le attività ed i meccanismi per controllarne il funzionamento. Oltre a ciò è sempre più importante poter coniugare le esigenze dei sistemi in tempo reale (*real-time systems*), caratterizzate da vincoli temporali, ed usi più generali, come quelli dei sistemi *general-purpose*. Analizzando le prime, è poi possibile suddividere i sistemi in tempo reale in *hard real-time*, in cui il mancato rispetto dei vincoli temporali può comportare il verificarsi di eventi disastrosi per il sistema, e *soft real-time*, in cui tale fenomeno non comporta il verificarsi di situazioni critiche, ma solo il degrado della qualità del servizio (QoS).

La fusione tra queste tendenze trova terreno fertile nel mondo dei sistemi operativi il cui codice sorgente viene rilasciato dagli autori, e dalla comunità di sviluppo, come "Open/Free Software". La libera e gratuita disponibilità del codice e la possibilità di apportarvi modifiche ha reso interessante, sia per la comunità scientifica che per le aziende, provare ad inserire, all'interno di sistemi tradizionalmente general-purpose, elementi e meccanismi tipici di quelli real-time. Tale approccio è particolarmente interessante soprattutto per i sistemi soft real-time, in cui processi real-time e non real-time devo-

no spesso poter coesistere ed essere gestiti dinamicamente all'interno dello stesso sistema operativo.

L'utilizzo sempre più diffuso del noto sistema operativo libero GNU/Linux (e nello specifico del suo kernel *Linux*) sulle più diverse piattaforme hardware comporta la necessità di adattare un sistema nato come *General Purpose Operating System* (GPOS) ad ambienti che spesso richiedo l'utilizzo di un *Real-Time Operating System* (RTOS). Quando infatti (ad esempio nel caso di telefoni cellulari o smart-phone su cui eseguono applicazioni *time-sensitive* come lettori audio/video) dimensioni, capacità computazionale, consumo di energia e costi sono particolarmente stringenti, bisogna necessariamente fare un uso efficiente delle risorse del sistema e, al tempo stesso, riuscire a rispettare i vincoli real-time.

Sul mercato è da tempo possibile trovare versioni modificate del kernel Linux con supporto ai sistemi in tempo reale [13, 19, 15] , ma queste sono spesso non aperte e non possono quindi trarre vantaggio dalla grande comunità di sviluppo dello standard kernel. Recentemente è stata invece sviluppata, presso il Real-Time System Laboratory (ReTiS Lab [12]) della Scuola Superiore Sant'Anna di Pisa, un'implementazione dell'algoritmo "Earliest Deadline First" (EDF [14, 17]) all'interno del kernel Linux [9, 16, 8]. EDF è un noto algoritmo di scheduling real-time a priorità dinamica basato su un concetto semplice: il processo con *deadline* (vincolo temporale che identifica l'istante di tempo entro il quale l'istanza corrente deve terminare) più imminente è quello che deve eseguire per primo. Si riesce inoltre ad assicurare l'isolamento temporale tra processi (ogni applicazione presente nel sistema che richieda garanzie di tipo soft real-time deve avere a disposizione una partizione delle risorse disponibili) utilizzando, congiuntamente ad EDF, anche l'algoritmo "Constant Bandwidth Server" (CBS [2, 1]).

Partendo da una implementazione pienamente utilizzabile solo su sistemi monoprocessore, si è giunti poi ad estendere tale implementazione a sistemi multiprocessore. Tale modifica è fondamentale perché permette di controllare finemente l'allocazione delle risorse del sistema e rende possibile una gestione efficace dell'ambiente durante il funzionamento dinamico. Senza tale estensione esistono inoltre situazioni in cui le risorse del sistema potrebbero rimanere sotto utilizzate (si veda [7] per alcuni esempi) andando ad incidere negativamente sul rapporto versatilità del sistema/costo dell'hardware, dove con *versatilità* si intende la capacità del sistema di risolvere problemi diversi mediante lo stesso hardware.

Obiettivi

Il lavoro di tesi, che ha reso possibile l'utilizzo su sistemi multiprocessore dell'algoritmo di scheduling real-time EDF/CBS, è tutt'altro che concluso. Utilizzando l'implementazione corrente è possibile decidere su quanti e quali

processori del sistema un processo debba eseguire. In letteratura si parla di scheduling *completamente partizionato* quando si associa staticamente ad ogni processo un solo processore tra quelli disponibili, contrariamente a quello che avviene utilizzando scheduling di tipo *globale*, dove tutti i processi possono eseguire su tutti i processori e, quando necessario, migrare da un processore all'altro. Sono ovviamente possibili anche soluzioni intermedie; si possono infatti creare *clusters* di processori indipendenti dal resto del sistema.

Nel presente piano di ricerca si propone da un lato di estendere ulteriormente l'implementazione dell'algoritmo di scheduling in modo da aumentarne l'efficienza, dall'altro uno studio approfondito delle performance dello stesso considerando tutta la gamma delle piattaforme hardware e delle applicazioni software che ne potrebbero beneficiare. Sarà inoltre interessante analizzare come il meccanismo di interesse si inserisca all'interno di sistemi soft real-time basati su *reservation* con condivisione di risorse (ad esempio FRESCOR [10] e AQuoSA [3]).

Il programma verrà realizzato secondo le seguenti fasi:

- Studio preliminare per evidenziare i problemi di scalabilità che la gestione attuale dei processi potrebbe incontrare; conseguentemente, ricerca e studio di strutture dati utilizzabili per risolvere tali problemi. Allo studio preliminare apparterrà anche l'individuazione dei sistemi soft real-time che potranno permettere una valutazione comparativa delle scelte implementative possibili e una stima reale degli overhead che i meccanismi di interesse introducono nel sistema.

- Valutazione delle possibili implementazioni, scelta e realizzazione di quella che garantisce maggiore efficienza, migliore scalabilità e minori overheads.

- Verifica pratica della soluzione proposta ed implementata.

Studio preliminare

Con lo studio preliminare si cercherà in un primo momento di individuare le eventuali criticità dell'attuale implementazione dei meccanismi di scheduling sopra descritti. A tal fine sarà interessante muoversi secondo due strade ortogonali. Lo spazio delle moderne architetture multiprocessore è molto vasto, sarà quindi interessante vedere come il sistema si comporta in ambienti diversi da quello di sviluppo (ad esempio architetture SMP con elevato numero di processori, NUMA o dispositivi embedded). Un aspetto caratterizzante una specifica architettura come, ad esempio, la gerarchia delle caches dei processori può influire infatti in maniera notevole sulla efficacia delle decisioni di scheduling. Fissando un tipo di hardware specifico

potrà poi essere interessante variare il tipo di applicazione software in modo da tenere in considerazione il più alto numero di utilizzi possibili.

Il metodo di collezione dei dati ed analisi dei risultati delle prove sperimentali sarà esso stesso oggetto di studio, in quanto non esistono al momento attuale pratiche e strumenti universalmente considerati utili ed attendibili. Uno dei problemi di maggiore rilevanza è infatti come estrarre informazioni sul funzionamento del sistema senza influire sul funzionamento stesso. Diversi approcci sono possibili, dalle più invasive modifiche al kernel per collezionare i dati da spazio utente, alla meno pesante raccolta di informazioni tramite lettura dei *performance counters* (registri speciali votati al controllo delle performance) presenti in quasi tutti i moderni processori.

Una volta raccolti un numero sufficiente di dati sarà possibile ricercare strutture dati che possano risolvere i problemi resi evidenti dall'analisi sperimentale. Una valutazione comparativa a livello teorico tra tutte le soluzioni possibili sarà necessaria già a questo stadio, in modo da evitare la scelta di una implementazione difficilmente realizzabile in pratica.

Progetto e realizzazione

Si conta, in questa fase, di implementare una o più delle strutture dati analizzate nella fase precedente. Anche se la fase di studio preliminare avesse ristretto la scelta ad una sola realizzazione sarà interessante implementarne diverse, anche solo come prototipi, in modo da poter effettivamente dimostrare la bontà della scelta fatta. Se, in alternativa, i dati sperimentali andassero contro quanto sostenuto dalla teoria, questo necessariamente porterebbe sia ad una revisione del modello teorico che alla modifica delle scelte implementative.

Nella fase di progetto prima e di realizzazione poi sarà importante anche tener conto della possibilità di integrazione con sistemi volti a rendere disponibili, all'interno del kernel Linux, meccanismi per il controllo della qualità del servizio (ancora come esempio, AQuoSA). Non è ancora chiaro se l'introduzione di tutte le modifiche descritte pregiudichi in qualche modo il corretto funzionamento dei framework esistenti; studiare inoltre quali benefici questi possano trarre dai nuovi meccanismi, progettare e quindi realizzare gli adattamenti necessari per una efficace integrazione sarà quindi parte molto interessante dell'attività di ricerca.

Infine, visto che nelle applicazioni multithread il problema della gestione delle risorse condivise è di particolare interesse, sarà necessario considerare come tale problema debba essere affrontato dai meccanismi di scheduling implementati. Si pensa quindi che possa essere necessario progettare la realizzazione di strumenti che rendano il processo di scheduling conscio dell'esistenza di tali risorse, in modo che esso possa prendere decisioni che tengano conto delle problematiche relative.

Verifica dei risultati

In questa ultima fase tutto quanto è stato realizzato sarà oggetto di verifica. Gli stessi test sul sistema effettuati nella fase di studio preliminare saranno adesso ripetuti per accertare la validità delle scelte implementative. Una volta ottenuti risultati significativi, questi potranno anche essere messi a confronto con quanto ottenuto in ricerche indipendenti (si vedano ad esempio [6, 4, 5]) in modo da poterne confermare o confutare le conclusioni.

Assumendo che il sistema abbia guadagnato in termini di performance globale, sarà poi interessante valutare se questo si comporti realmente secondo quanto stabilisce la teoria dello scheduling EDF globale. A tal fine potranno essere utilizzati strumenti già esistenti [11] o in alternativa si provvederà ad implementarne di nuovi.

Ultimo test interessante, anche in vista di possibili sviluppi futuri della ricerca sull'argomento, sarà analizzare come il sistema si comporti quando coeasistano all'interno di esso processi assegnati staticamente ad alcuni processori e altri liberi invece di spostarsi su tutti i processori disponibili.

Riferimenti bibliografici

[1] L. Abeni and G. Buttazzo. Resource reservations in dynamic real-time systems. *Real-Time Systems*, 27(2):123–165, 2004.

[2] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[3] AQuoSA. Aquosa - "adaptive quality of service architecture" (for the linux kernel). `http://aquosa.sourceforge.net/index.php`.

[4] B. Brandenburg and J. Anderson. On the implementation of global real-time schedulers. In *Proceedings of the 30th IEEE Real-Time Systems Symposium*, pages 214–224, December 2009.

[5] B. Brandenburg, J. Calandrino, and J. Anderson. On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169, December 2008.

[6] J. Calandrino, J. Anderson, and J. Baumberger. A hybrid real-time scheduling approach for large-scale multicore platforms. In *Proceedings of the 19th Euromicro Conference on Real-time Systems*, pages 247–256, July 2007.

[7] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, chapter 30: A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms. Chapman Hall/CRC Press, 2004.

[8] Dario Faggioli. sched: Sched_deadline v2. `http://lkml.org/lkml/2010/2/28/107`.

[9] Dario Faggioli, Michael Trimarchi, Fabio Checconi, and Scordino Claudio. An edf scheduling class for the linux kernel. In *Proceedings of the 11th Real-Time Workshop (RTLW)*, October 2009.

[10] FRESCOR. Framework for real-time embedded systems based on contracts. `http://www.frescor.org/index.php?page=FRESCOR-homepage`.

[11] UNC's Real-Time Group. `http://cs.unc.edu/~mollison/unit-trace/`.

[12] ReTiS Lab. Scuola superiore sant'anna di pisa. `http://retis.sssup.it`.

[13] Wind River Linux. `http://www.windriver.com/products/linux/`.

[14] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[15] montavista. `http://www.mvista.com/real_time_linux.php`.

[16] Evidence S.r.l. Sched_deadline. `http://www.evidence.eu.com/content/view/313/390/`.

[17] John A. Stankovic, K. Ramamritham, M. Spuri, and G. Buttazzo. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publishers, 1998.

[18] techradar.com. Arm: dual core mobiles coming in 2010. `http://www.techradar.com/news/phone-and-communications/mobile-phones/arm-dual-core-mobiles-coming-in-2010-645417?src=rss&attr=all`.

[19] Timesys. `https://linuxlink.timesys.com/3/Linux`.