 ISSN 2089-5490

ANALISIS SISTEM PENJADWALAN REAL-TIME MULTICORE PADA VIRTUALISASI RT-XEN 2.0

Dhanny Rukmana Manday[1],
Achmad Imam Kistijantoro, ST, M.Sc, Ph.D[2]

***Abstrak*— Virtualisasi telah menyebabkan peningkatan performansi pada prosesor multicore. Integrasi sistem real-time sebagai mesin virtual diatas platform multicore juga menimbulkan tantangan untuk memenuhi persyaratan kinerja penjadwalan real-time. RT-Xen 2.0 sebagai integrasi sistem mesin virtual real-time pada platform multicore menjawab tantangan dalam hal memenuhi kebutuhan suatu sistem agar dapat menjalankan banyak task secara bersamaan tanpa terjadi saling mendahului antar setiap task yang berjalan dengan menggunakan algoritma EDF untuk dynamic-priority schedulling. Algoritma EDF bekerja lebih baik dalam menyelesaikan task secara rt-global maupun rt-partition.**

Kata Kunci— virtualisasi, multicore, RT-Xen 2.0, Algoritma EDF, penjadwalan real-time.

I. PENDAHULUAN

Seiring perkembangan waktu, penggunaan sebuah piranti server komputer terdedikasi dengan satu sistem operasi tunggal saat ini dirasa kurang efisien, mengingat perkembangan CPU memungkinkan prosesor melakukan multiprogramming sehingga banyak proses dapat berjalan secara real-time. Akibatnya sistem operasi dituntut dapat membuat komputer lebih produktif, oleh karena itu perlu adanya mekanisme penjadawalan proses-proses yang ada pada sistem [1]. Untuk memenuhi kebutuhan sistem operasi yang beragam dalam sebuah piranti server komputer maka dibuatlah konsep virtualisasi. Virtualisasi merupakan strategi atau teknik yang memungkinkan pusat data untuk mengkonsolidasikan infrastruktur server fisik dengan menempatkan server virtual pada sejumlah kecil server fisik yang lebih kuat. Teknologi virtualisasi menyediakan mesin virtual yang dinamis dibawah satu sistem operasi host (sistem operasi utama yang berjalan pada perangkat keras fisik) bertujuan untuk menggunakan resource hardware seefisien mungkin [2] [3]. RT-Xen 2.0 salah satu tool untuk membuat mesin virtualisasi dikembangkan oleh University of Cambridge Computer Laboratory. RT-Xen 2.0 adalah software open source dan gratis digunakan dan dimodifikasi. Dengan RT- Xen 2.0, beberapa mesin virtual dapat bekerja bersama pada sebuah piranti server komputer tanpa saling mempengaruhi. Tujuan penelitian ini adalah untuk menguji sistem penjadwalan RT-Xen 2.0 real-time dengan menggunakan algoritma penjadwalan EDF (earliest deadline first), adapun kriteria untuk mengukur dan optimasi kinerja penjadwalan real-time pada penelitian ini adalah dengan jumlah dan besar task yang terus ditingkatkan dengan menerapkan konsep budget, period, dan deadline [3].

II. KONSEP REAL-TIME PADA SISTEM OPERASI

A. Sistem Operasi

Sistem operasi erat kaitannya dengan penyelesaian proses dan hubungan proses dengan sumber daya (resources). Hal tersebut adalah mekanisme yang harus dimiliki sistem operasi yang disebut penjadwalan. Penjadwalan adalah ketentuan secara matematis yang diterapkan dalam penyelesaian proses baik secara single thread atau multi thread dan bertujuan optimalisasi eksekusi proses terhadap sumber daya fisik dan abstrak [6].

B. Konsep Penjadwalan Real-Time

Penjadwalan real-time pada dasarnya untuk menentukan urutan di mana berbagai task yang harus diambil untuk dieksekusi/dijalankan oleh sistem operasi. Setiap sistem operasi menggunakan satu atau lebih penjadwalan task untuk mempersiapkan jadwal eksekusi berbagai task yang dibutuhkan untuk dijalankan. Setiap penjadwalan task ditandai oleh algoritma penjadwalan yang mengaturnya [4].

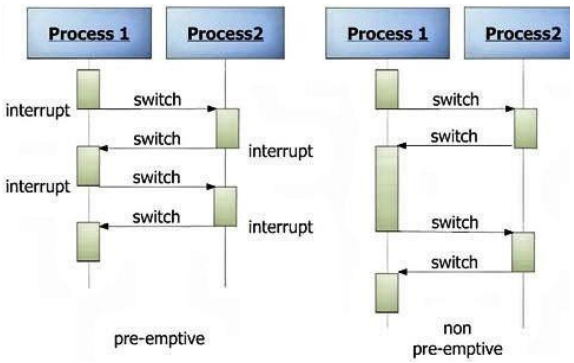
Penjadwalan real-time harus menghasilkan respon yang tepat dalam batas waktu yang telah ditentukan. Jika respon komputer melewati batas waktu tersebut, maka terjadi degradasi performansi atau kegagalan sistem. Penjadwalan real-time adalah sistem yang kebenarannya secara logis didasarkan pada kebenaran hasil-hasil keluaran sistem dan ketepatan waktu hasil-hasil tersebut dikeluarkan [9].

Dalam strategi penjadwalan, sistem real-time dibagi atas dua, yaitu:

- Preemptive
- Non preemptive

Perbedaan cara penyelesaian proses yang dilakukan strategi preemptive dan non preemptive dapat dilihat pada gambar 1.

 ISSN 2089-5490



Gambar 1. Cara penyelesaian pada preemptive dan non preemptive [11] [12]

III. KONSEP ALGORITMA PENJADWALAN EDF

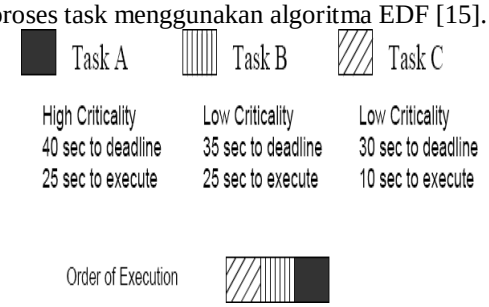
A. Konsep Algoritma EDF

Earliest deadline first (EDF) adalah algoritma penjadwalan dinamis yang digunakan dalam sistem real-time yang menempatkan proses dalam antrian berdasarkan prioritas. Setiap kali penjadwalan terjadi (selesai menjalankan task, task baru dirilis) maka antrian akan dicari untuk proses yang paling dekat dengan batas waktu deadline tesebut. Sehingga proses ini yang akan dieksekusi berikutnya.

Earliest deadline first (EDF) merupakan algoritma penjadwalan yang optimal pada uniprosesor preemptive, karena algoritma ini akan menjadwalkan sekumpulan task/job yang masing-masing task/job tersebut memiliki waktu kedatangan, persyaratan eksekusi, tenggat waktu dan menjamin semua task/job tersebut selesai dengan tenggat waktu masing-masing task/job tersebut.

B. Pembagian task berdasarkan deadline

Implementasi EDF akan mempertahankan semua task yang siap untuk dieksekusi dalam antrian. Setiap task baru akan dimasukkan pada akhir antrian. Setiap node dalam antrian akan berisi batas waktu mutlak (absolutely deadline) task tertinggi. Pada setiap titik preemption, seluruh antrian akan diposisikan dari awal untuk menentukan tugas memiliki tenggat waktu terpendek. Gambar 2 berikut menjelaskan proses task menggunakan algoritma EDF [15].



Gambar 2. Sekumpulan task yang dijadwalkan pada algoritma EDF

Alur program scheduling algoritma EDF dapat dilihat dibawah ini [13].

1: scurr the currently running VCPU on this PCPU
2: idleVCPU the idle VCPU on this PCPU
3: snext idleVCPU
4: burn_budget(scurr)
5: for all VCPUs in the RunQ do
6: if VCPU's new period starts then
7: reset VCPU.deadline, replenish VCPU.budget
8: move VCPU to the appropriate place in the RunQ
9: end if
10: end for
11: for all VCPUs in the RunQ do
12: if VCPU.cpu_mask & this PCPU ≠ 0 then
13: if VCPU.budget > 0 then
14: snext VCPU
15: break
16: end if
17: end if
18: End For
19: if (snext = idleVCPU snext.deadline > scurr.deadline) (scurr ≠ idleVCPU) (scurr.budget > 0) vcpu_runnable(scurr) then
20: snext scurr
21: end if
22: if snext ≠ scurr then
23: remove snext from the RunQ
24: end if
25: return snext to run for 1 ms
End

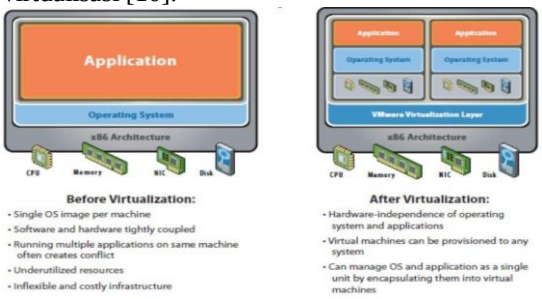
IV. VIRTUALISASI

Virtualisasi/virtualization adalah sebuah teknik atau cara untuk membuat sesuatu dalam bentuk virtualisasi, virtualisasi juga merupakan salah satu strategi untuk mengurangi konsumsi daya pusat data dengan mengemulasikan perangkat fisik komputer dan membuatnya seolah-olah perangkat tersebut tidak da (disembunyikan) atau bahkan menciptakan perangkat yang tidak ada menjadi ada.

Virtualisasi memungkinkan pusat data untuk mengkonsolidasikan infrastruktur server fisik dengan menempatkan server virtual pada sejumlah kecil server fisik yang lebih kuat, sehingga menggunakan energi listrik yang sedikit dan menyederhanakan pusat data. Selain mendapatkan penggunaan hardware yang lebih baik, virtualisasi mengurangi penggunaan ruang pusat data, membuat komputasi dengan penggunaan daya yang lebih baik, dan mengurangi kebutuhan energi pada pusat data. Banyak perusahaan menggunakan virtualisasi untuk membatasi konsumsi energi dari pusat data [5][7]. Gambar 3 menjelaskan kosep

 ISSN 2089-5490

virtualisasi [10].

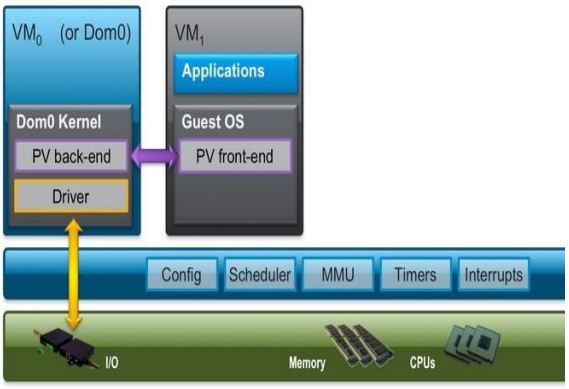


Gambar 3. Konsep sebelum dan sesudah virtualisasi pada sistem operasi

A. Konsep Virtualisasi Pada RT-Xen 2.0

Salah satu tujuan RT-Xen 2.0 adalah melakukan pendekatan yang optimal untuk desain suatu hypervisor dengan membiarkan hypervisor ini berjalan dengan kompleksitas yang rendah dengan tidak mengurangi nilai fleksibilitas dan performa CPU itu sendiri. Sehingga dapat dikatakan bahwa Xen menyajikan konsep virtualisasi mesin yang sangat mirip dengan platform hardware arsitektur prosesor tanpa menciptakan salinan teknik arsitektur prosesor tersebut. Ini juga merupakan inti dari paravirtualization. Karena konsep paravirtualization tidak memerlukan beberapa modifikasi guest OS [10].

RT-Xen memiliki kontrol terhadap setiap vm yang diatur melalui dom0. Dom0 berfungsi seperti instalasi Linux standar, sehingga dapat dijalankan aplikasi modus pengguna, seperti yang digunakan untuk mengelola lingkungan Xen serta dipasang user driver yang diperlukan untuk mendukung platform perangkat keras. Karena kemampuan untuk mengkompilasi dan menjalankan hampir semua perangkat keras dengan driver Linux yang tersedia, hal ini memberikan fleksibilitas yang lebih besar kepada organisasi TI dengan pilihan akan jaringan fisik dan perangkat penyimpanan maupun mengijinkan Xen untuk diterapkan pada hampir setiap lingkungan x86. Gambar 4 menunjukkan interaksi hardware driver RT-Xen 2.0 pada domain0 dan domainU [15].



Gambar 4. Interaksi hardware driver pada domain0 dan domainU

V. PERANCANGAN VIRTUALISASI RT-XEN 2.0

Pada tahap ini dilakukan perancangan virtualisasi dengan menggunakan konsep penjadwalan real-time multicore pada RT-Xen 2.0.

A. Perancangan vm1 dan vm2

Perancangan vm1 dan vm2 bertujuan untuk membandingkan hasil eksekusi real-time task baik secara rt- global maupun rt-partition, dengan melakukan tiga skenario yang berbeda untuk masing-masing vm dan akan dihitung distribusi nilai overhead untuk masing-masing vm.

B. RT-Global dan RT-Partition

Perbedaan kebutuhan penjadwalan pada multicore membutuhkan strategi khusus untuk mencapai tujuan real- time system. Penjadwalan secara rt-global memungkinkan seluruh vcpu yang ada pada sistem dipakai untuk mencapai tujuan real-time system, sedangkan penjadwalan secara rt- partition hanya memungkinkan satu vcpu pada sistem dipakai untuk mencapai tujuan real-time system.

C. Konfigurasi Hardware dan Software

Untuk mencapai tujuan eksperimen yang diinginkan, kami memakai Intel i7 dengan dua core @ 1.9 GHz (PCPUs) yang sudah mendukung VT (virtualization technology) dengan memory RAM sebesar 8 GB, adapun untuk software menggunakan sistem operasi ubuntu 12.04 LTS 64 bit, hypervisor RT-Xen 2.0 dan memakai kernel versi 3.10 sebagai domain0, dan 64 bit Ubuntu image pada domU (vm1 dan vm2) secara paravirtualisasi.

~/liblitmus#./base_task1			
N = 22		Loop = 20 times	
Iteration	Result of task_1	Interval Time (s)	Execution Time (s)
1	4194304	0,100175	0,016756
2	4194304	0,082991	0,031454
3	4194304	0,06847	0,03153
4	4194304	0,06842	0,031749
5	4194304	0,068199	0,031541
6	4194304	0,068388	0,031583
7	4194304	0,068385	0,031508
8	4194304	0,068436	0,031535
9	4194304	0,068418	0,031742
10	4194304	0,0682	0,031539
11	4194304	0,068409	0,031541
12	4194304	0,068407	0,031627
13	4194304	0,068324	0,031716
14	4194304	0,068228	0,031829

 ISSN 2089-5490

15	4194304	0,068116	0,03162
16	4194304	0,068328	0,031831
17	4194304	0,068122	0,031588
18	4194304	0,068345	0,031795
19	4194304	0,068132	0,031877
20	4194304	0,0681	0,031608
The average interval time (s)		0,07062965	
The average execution time (s)		0,03089845	

VI. SKENARIO PERCOBAAN

Adapun untuk skenario percobaan dibagi menjadi tiga bagian, yaitu:

1. Menjalankan task menggunakan algoritma EDF, dengan looping sebanyak 20 kali dan mengukur nilai overhead untuk tiap task.
2. Menjalankan task pada vm1 dan satu task konstan pada vm2 dengan looping sebanyak 20 kali.
3. Menjalankan task pada vm1dan satu task non real-time
4. pada vm2 sebanyak 30 kali.

VII. HASIL IMPLEMENTASI DAN PENGUJIAN

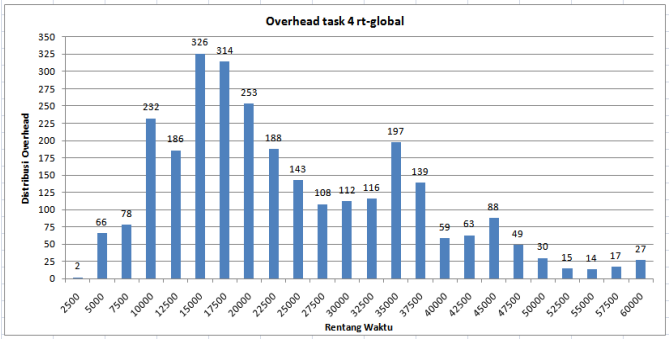
Hasil akhir dari penelitian ini adalah perbandingan performa real-time system pada rt-global maupun rt-partition. Dengan menerapkan kerangka kerja waterfall untuk setiap langkah-langkah (milestone) untuk mencapai hasil yang diinginkan. Untuk memberikan hasil yang lebih maksimal untuk setiap task yang dibuat kami menggunakan kerangka base_task yang disediakan LITMUS RT. Sebelum melakukan percobaan pada RT-Xen 2.0, kami melakukan kalibrasi setiap CPU pada guest OS sebesar 1 ms, dan menghitung besar nilai overhead yang dihasilkan menggunakan st_trace yang disediakan LITMUS RT.

Pengujian dengan skenario pertama, kedua, dan ketiga Pada penelitian ini, pengujian dilakukan dengan skenario pertama, hasilnya dapat dilihat pada tabel 1, tabel 2, dan tabel 3 dibawah ini:

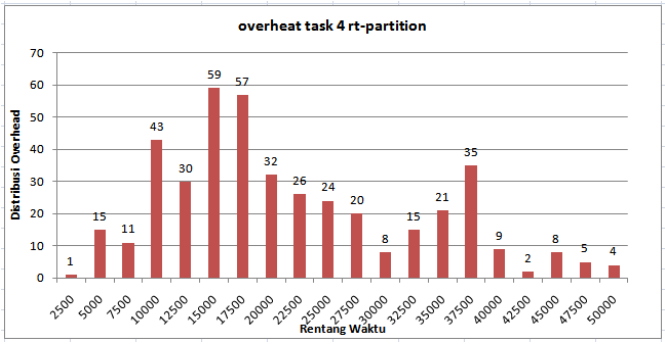
Tabel 1. Hasil algoritma EDF dengan task n pada vm1

~/liblitmus#./base_task4			
vm 1			vm2
N = 25		Loop = 20 times	N = 24
Result of task on vm1	Interval Time (s)	Execution Time (s)	Result of constant task on vm2
67108864	0,100064	0,111204	16777216
67108864	0,000014	0,089025	16777216
67108864	0,00002	0,088204	16777216
67108864	0,011291	0,08756	16777216
67108864	0,012402	0,088739	16777216
67108864	0,011225	0,087926	16777216
67108864	0,01204	0,088139	16777216
67108864	0,011791	0,088636	16777216
67108864	0,011365	0,087581	16777216
67108864	0,012384	0,089028	16777216
67108864	0,010912	0,089163	16777216
67108864	0,010861	0,088763	16777216
67108864	0,011141	0,088438	16777216
67108864	0,011562	0,088239	16777216
67108864	0,011655	0,088128	16777216
67108864	0,011831	0,087372	16777216
67108864	0,012613	0,08787	16777216
67108864	0,012151	0,088061	16777216
67108864	0,011846	0,087308	16777216
67108864	0,012696	0,089262	16777216
The average interval time (s)			0,01499320
The average execution time (s)			0,0894323

Nilai Overhead pada tiap task rt-global maupun rt-partition Pada bagian ini akan ditunjukkan nilai overhead tiap task rt- global maupun rt-partition. Didapatkan pada Gambar 5, dan
gambar 6.



Gambar 5. Nilai overhead yang dihasilkan pada rt-global



Gambar 6. Nilai overhead yang dihasilkan pada rt-partition

 ISSN 2089-5490

VIII. KESIMPULAN

Berdasarkan hasil analisa perbandingan kinerja dari setiap skenario percobaan yang telah dilakukan, dapat diambil kesimpulan sebagai berikut :

Penjadwalan menggunakan algoritma EDF untuk dinamic-priority scheduling baik secara rt-global maupun rt-partition dapat berjalan dengan baik diatas vm.

Nilai overhead yang dihasilkan rt-global lebih tinggi daripada nilai overhead yang dihasilkan rt-partition mengingat pada rt-global seluruh vcpu yang ada dipakai untuk menjalankan proses task. Namun hasil eksekusi terhadap task dapat berjalan dengan baik dan memnuhi deadline.

REFERENSI

[1] Abdel-Hamid, T., and Madnick, S.: Software Project Dynamics: An Integrated Approach, Upper Saddle River, NJ: Prentice Hall, 1991

[2] Albert M. K. Cheng (2002): Real-time systems, Scheduling, analysis and verification. Wiley Publishing.

[3] B. Brandenburg, "Scheduling and Locking in Multiprocessor Real-Time Operating Systems", PhD thesis, UNC Chapel Hill, 2011.

[4] B. Brandenburg and J. H. Anderson, "On the Implementation of Global Real-Time Schedulers," in RTSS. IEEE, 2009.

[5] Bernard, G., & Scheffy, C. (2007): Virtualization For Dummies. Wiley Publishing.

[6] Blumofe, R.D., & Leiserson, C.E.: "Scheduling Multithreaded Computations by Work Stealing," Proc. 35th Annual Symp. on Foundations of Computer Science, IEEE, pp. 356-368, Nov. 1994.

[7] Bovet, D.P., and Cesati, M.: Understanding the Linux Kernel, Sebastopol, CA: O'Reilly & Associates, 2000.

[8] Cheriton, D.: "The V Distributed System," Commun. of the ACM, vol. 31, pp. 314-333, March 1988.

[9] Coffman, E.C., Elphick, M.J., and Shoshani, A.: "System Deadlocks," Computing Surveys, vol. 3, pp. 67-78, June 1971.

[10] Chris, W., Erick, M. Halter (2005): "Virtualization From Desktop to the Enterprise". Apress.

[11] Naghibzadeh, M., (2005): "Operating System Concept and Technique". IUniverse

[12] Silberschatz, A., (2004): "Operating System Concepts". Prentice Hall.

[13] Sisu, et all., Sisu X., Meng X., Lint T.X. Phan, (2014), "Real-Time Multi-Core Virtual Machine Scheduling in Xen".

[14] Tanenbaum, A. S (1994). Modern Operating Systems. Prentice Hall PTR.

[15] Von Hagen, W. (2008). "Professional Xen Virtualization". Indianapolis: Wiley Publishing, Inc