Meningkatkan Kinerja *Java Virtual Machine* dengan Mengoptimalkan Peran *Garbage Collector*

Amil Ahmad Ilham*, Muhammad Niswar*, Ansar Suyuti, Indra Bayu Amirullah***
*Teknik Informatika Fakultas Teknik Universitas Hasanuddin
** Teknik Elektro Fakultas Teknik Universitas Hasanuddin

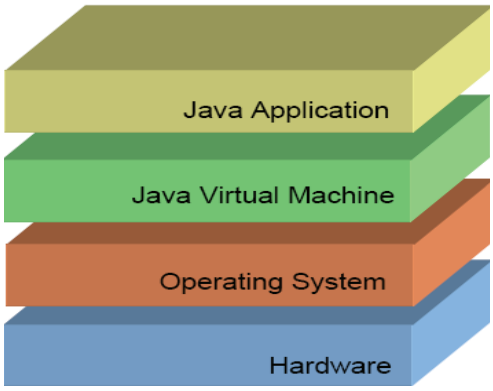
Abstrak

Pada penelitian ini, kinerja *Java Virtual Machine* (JVM) dalam mengeksekusi aplikasi Java berusaha ditingkatkan dengan cara mengoptimalkan peran Garbage Collector (GC). GC adalah salah satu komponen di dalam JVM yang bertugas untuk mencari obyek-obyek Java yang sudah tidak digunakan lagi oleh program (*garbage*) tapi masih menempati ruang pada memori. Selanjutnya GC akan membersihkan ruang memori tersebut untuk digunakan oleh obyek Java yang baru. JVM secara otomatis menjalankan GC jika ruang memori yang tersedia tidak cukup lagi untuk menampung obyek baru yang dibuat oleh Java aplikasi. Untuk mengoptimalkan peran GC dalam meningkatkan kinerja JVM, GC tidak hanya difungsikan untuk membersihkan ruang memori dari *garbage*, tapi juga difungsikan untuk mengubah urutan-urutan obyek yang masih digunakan oleh program di memori, sedapat mungin menyesuaikan dengan cara program mengakses obyek-obyek tersebut. Hal ini dimaksudkan agar prosessor komputer lebih cepat mendapatkan data dari memori sehingga eksekusi program menjadi lebih cepat. Hasil penelitian ini menunjukkan bahwa dengan urutan-urutan obyek yang tepat, kinerja JVM meningkat, ditandai dengan waktu eksekusi aplikasi Java yang menjadi lebih singkat.

Kata kunci: *JVM, GC, memori, obyek, garbage*

I. Pendahuluan

Java adalah bahasa pemrograman modern yang berorientasi obyek yang menggabungkan fitur –fitur yang baik dari bahasa berorientasi obyek sebelumnya seperti Smalltalk dan C++. Java mendukung pengembangan program-program yang *virus-free* , *tamper-free* (*secure*) dan *robust*. Program Java dijalankan oleh suatu "mesin" yang disebut dengan *Java Virtual Machine* [1] yaitu suatu lapisan perangkat lunak yang menerjemahkan kode-kode Java ke dalam bahasa yang dimengerti oleh perangkat keras komputer. Dengan sistem seperti ini, aplikasi Java hanya dikodekan satu kali tapi dijalankan pada berbagai macam sistem operasi dan *platform* yang berbeda-beda. Gambar 1 menunjukkan posisi aplikasi Java dan JVM pada sistem komputer.



Gambar 1: Lapisan aplikasi Java dan JVM pada sistem komputer

Salah satu cara meningkatkan kecepatan eksekusi aplikasi Java adalah dengan meningkatkan kinerja JVM. Ada beberapa komponen JVM yang berpengaruh terhadap kinerja JVM, salah satu diantaranya adalah *Garbage Colletor*.

II. Garbage Collector (GC)

Garbage Colletor (GC) memberikan manfaat yang besar bagi programmer karena menggantikan fungsi programmer dalam mencari dan menghapus obyek-obyek yang sudah tidak digunakan lagi (*garbage*) oleh program. GC menghilangkan kesalahan-kesalahan yang sering dibuat oleh programmer seperti menghapus obyek yang masih digunakan oleh program (*premature frees*), menghapus obyek yang sudah tidak ada lagi (*double frees*), atau tidak menghapus obyek yang sudah tidak digunakan lagi (*memory leaks*) sehingga memori yang digunakan oleh JVM cepat penuh. Dengan peran seperti ini, GC dapat meningkatkan produktivitas programmer. Tabel 1 memperlihatkan perbedaan pengaturan obyek pada bahasa pemrograman Java dengan C/C++.

Pada penelitian ini, kinerja JVM dalam mengeksekusi aplikasi Java berusaha ditingkatkan dengan cara mengoptimalkan peran GC yaitu dengan memanfaatkan GC selain untuk mencari *garbage* juga mengatur kembali urutan-urutan obyek yang masih digunakan oleh program sehingga obyek-obyek yang saling berhubungan satu sama lain menempati ruang memori yang berdekatan. Obyek-obyek ini selanjutnya ditempatkan pada memori khusus yang disebut dengan *cache* [2] yaitu suatu memori yang memiliki kapasitas terbatas tetapi dapat diakses dengan sangat cepat. Jika prosessor membutuhkan data dari memori dan data tersebut tersedia di *cache*, maka prosessor akan langsung mengeksekusi instruksi yang membutuhkan data tersebut. Namun jika data tidak tersedia di *cache*, maka prosessor harus menunggu beberapa lama sampai data yang dibutuhkan tersedia di *cache*. Dengan demikian, salah satu cara untuk mempercepat eksekusi suatu program adalah dengan cara menyiapkan data-data yang dibutuhkan oleh program pada cache.

Table 2 menunjukkan contoh yang sangat sederhana bagaimana urutan-urutan obyek di memori berpengaruh terhadap jumlah *cache misses* yaitu keadaan di mana data yang dibutuhkan oleh prosessor tidak tersedia di cache. Pada contoh kasus 1, diasumsikan bahwa urutan obyek di memori berturut-turut adalah A, C, F, H, J dan K. Satu *line cache* dapat menampung maksimal 2 obyek. Misalkan obyek A menempati *line* 1, obyek C dan F menempati *line 2* serta obyek H dan J menempati *line 3*. Ketika program mengakses berturut-turut obyek A, C, dan F, maka prosessor bisa langsung mendapatkan data yang dibutuhkan karena obyek A, C dan F sudah berada di *cache*. Pada contoh kasus 2, diasumsikan bahwa urutan obyek di memori berturut-turut adalah A, C, H, J, K dan F sehingga keadaan *cache* mengalami perubahan yaitu obyek A menempati *line 1*, obyek C dan H menempati *line 2*, serta obyek J dan K menempati *line 3*. Ketika program mengakses obyek A, C dan F, prosessor harus menunggu beberapa saat karena obyek F tidak berada di *cache*. Hal ini memperlambat ekesekusi program.

Karena program Java sangat bergantung pada obyek dan referensi, maka urutan-urutan obyek di memori menjadi sangat penting sehingga diharapkan dengan mengubah urutan-urutan obyek sesuai dengan yang dibutuhkan program dapat mempercepat eksekusi program Java.

III. Eksperimen

Dalam penelitian ini GC dimanfaatkan selain untuk membersihkan obyek-obyek yang tidak digunakan lagi oleh program, juga sekaligus mengurutkan kembali obyek-obyek yang masih digunakan oleh program. Pada experimen ini, ada dua metode pengurutan obyek yang dilakukan yaitu metode *Breadth First* (BF) dan metode *Depth First* (DF). Kedua metode ini berbeda dalam menempatkan obyek-obyek yang

memiliki hubungan *parent-child* di memori. Dengan metode BF, semua obyek yang memiliki status sebagai *children* dari suatu obyek ditempatkan secara berurutan di memori, sedangkan dengan metode DF, hanya satu obyek yang berstatus sebagai *children* ditempatkan berdekatan dengan obyek *parent*. Tabel 3 menunjukkan perbedaan urutan obyek berdasarkan metode BF dan DF.

Spesifikasi perangkat yang digunakan pada eksperimen adalah komputer Pentium 4, 3.20 GHz, tanpa mengaktifkan fungsi *Hyperthreading*. Sistem memori pada komputer memiliki 64 btye DL1 dan *L1 cache line size*, 8KB *4-way set associative L1 data cache*, a 512KB *unified 8-way set associative L2 cache*, dan 2GB memori utama. Sistem operasi yang digunakan adalah 32bit Linux yang sudah di-compile dengan *perfcrt* [4], sehingga bisa digunakan untuk mengakses *on-chip performance counters* untuk mengukur parameter-parameter kinerja yang diinginkan seperti jumlah L1 dan L2 cache misses serta DTLB misses. Aplikasi yang dijalankan pada eksperimen ini adalah aplikasi-aplikasi Java yang diambil dari SPECjvm98 [5] benchmark suite dan *open source* DaCapo [6] benchmark suite versi 2006-10-MR2.

IV. Hasil dan Pembahasan

Hasil penelitian awal menunjukkan bahwa ukuran obyek Java pada umumnya kecil sehingga beberapa obyek Java dapat disimpan secara bersamaan dan menempati ruang yang berdekatan pada cache. Tabel 4 memperlihatkan bahwa paling sedikit 40% obyek yang dibuat oleh aplikasi Java berukuran tidak lebih dari 32 byte.

Gambar 2 menunjukkan bahwa urutan obyek pada memory berpengaruh terhadap waktu eksekusi program. Ketika GC mengurutkan obyek berdasarkan metode DF, waktu eksekusi aplikasi _209_db berkurang 16% dibandingkan dengan ketika obyek diurut berdasarkan metode BF. Perbedaan waktu eksekusi akibat perbedaan urutan obyek di memori juga terjadi pada aplikasi 213_javac, _228_jack, _202_jess, _227_mtrt, antrl, bloat, fop, and hsqldb yaitu sekitar 3-7% dimana pada umumnya waktu eksekusi program berkurang ketika obyek diurut berdasarkan metode DF.

V. Kesimpulan

Berdasarkan hasil eksperimen dapat disimpulkan bahwa GC dapat digunakan untuk meningkatkan waktu eksekusi program dengan cara mengurutkan obyek sesuai dengan kebutuhan program. Ketika obyek diurutkan mengikuti metode DF, waktu eksekusi aplikasi Java semakin singkat dibandingkan dengan ketika obyek diurutkan dengan metode BF.

Daftar Pustaka

[1] J.E. Smith and R. Nair, *Virtual Machines: Versatile platforms for systems and processes*, USA: Morgan Kaufmann Publishers, 2005.

[2] J. Hennessy and D. Patterson, *Computer architecture: a quantitative approach*, San Mateo, CA. Morgan Kaufmann Publishers, 1995.

[3] B. Alpern, S. Augart, S. Blackburn, M. Butrico, A. Cocchi, P. Cheng, J. Dolby, S. Fink, D. Grove, M. Hind, K. McKinley, M. Mergen, J. Moss, T. Ngo, V. Sarkar, and M. Trapp, "The Jikes Research Virtual Machine Project: Buliding an Open-source Research Community," *IBM Systems Journal*, vol. 44, 2005, p. 399–417.

[4] "perfctr," *http://user.it.uu.se/~mikpe/linux/perfctr/*.

[5] "Standard Performance Evaluation Corporation," *http://www.spec.org/benchmarks.html#java*.

[6] S.M. Blackburn, R. Garner, C. Hoffman, A.M. Khan, K.S. McKinley, R. Bentzur, A.

Diwan, D. Feinberg, D. Frampton, S.Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J.E.B. Moss, A. Phansalkar, D. Stefanovic, T. VanDrunen, D. Von Dincklage, and B. Wiedermann, "The DaCapo Benchmarks: Java Benchmarking Development and Analysis," *In proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA `06)*, Portland, OR, USA: 2006, pp. 169-190.

[7] M. Arnold, S. Flink, D. Grove, M. Hind, and P. Sweeney, "Adaptive optimization in the Jalapeno JVM," *In proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA `00)*, 2000.

Tabel 1: Pengaturan obyek pada bahasa pemrograman Java dengan C/C++

Bahasa	Membuat obyek	Menghapus obyek
C, C++	Programmer (malloc, new)	Programmer (free, delete)
Java	Programmer (new)	Garbage collector

Table 2: Contoh sederhana hubungan antara urutan obyek dengan *cache misses*

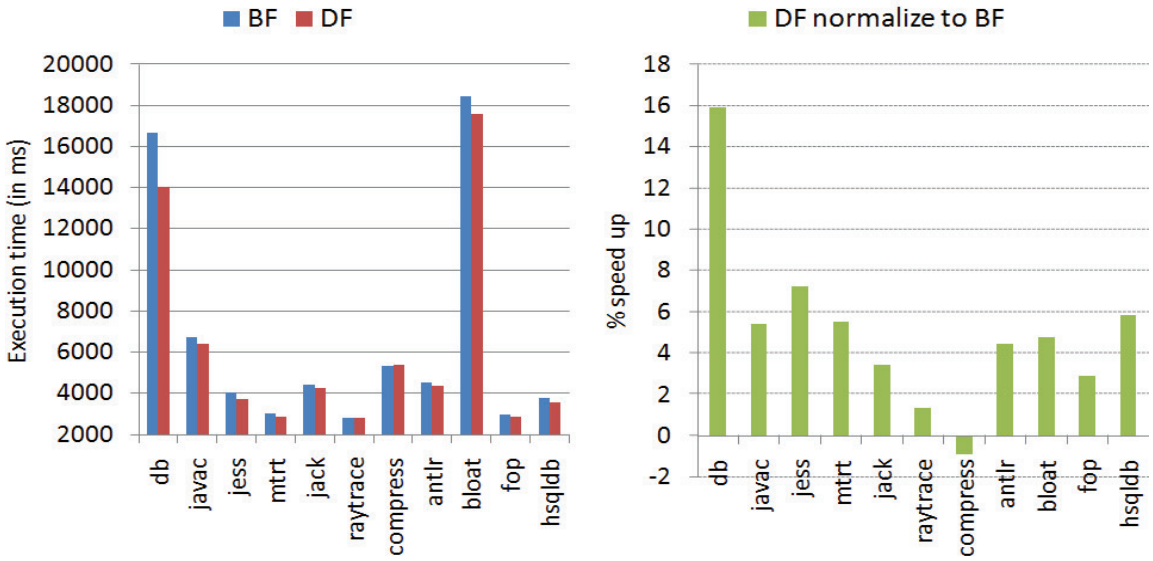
Kasus	Urutan obyek di memori	Keadaan *cache*	Obyek yang diakses oleh program	*Cache misses*
1	A C F H J K	* * Line 0 * A Line 1 C F Line 2 H J Line 3	A,C,F	0
2	A C H J K F	* * Line 0 * A Line 1 C H Line 2 J K Line 3	A,C,F	1

Table 3: Perbedaan urutan obyek berdasarkan metode BF dan DF

Metode	Urutan obyek
BF	*Children*
DF	*Parent – child*

Table 4: Distribusi ukuran obyek yang dibuat oleh aplikasi Java

Aplikasi	Jumlah Obyek	Ukuran Obyek (z)			
		z<=16 bytes	16<z<=32 bytes	32<z<=64 bytes	z>64 bytes
_209_db	3411032	87.17%	9.16%	1.88%	1.79%
_213_javac	6365225	30.84%	47.81%	15.04%	6.31%
_202_jess	8258710	41.36%	29.01%	26.33%	3.3%
_227_mtrt	9486343	35.04%	51.42%	10.39%	3.15%
_228_jack	10946169	31.65%	38.45%	25.18%	4.72%
_205_raytrace	6604512	9.51%	36.03%	44.86%	9.6%
_201_compress	3190165	3.18%	45.14%	37.95%	13.73%
Antlr	1642623	32.05%	38.75%	19.68%	9.52%
Bloat	4936276	36.74%	41.17%	10.47%	11.62%
Fop	5737593	26.79%	38.12%	22.48%	12.61%
Hsqldb	6336240	40.33%	43.26%	8.31%	8.1%



Gambar 2: Pengaruh urutan obyek terhadap waktu eksekusi program

Supported by IEEE Indonesia Section, IEEE APS/MTT Indonesia Joint Chapter, and IEEE Communication Society (Comsoc) Indonesia Chapter