Penerapan Java Dynamic Compilation pada Metode Java Customized Class Loader untuk Memperbaharui Perangkat Lunak pada saat Runtime

Tory Ariyanto, Romi Satria Wahono and Purwanto
Fakultas Ilmu Komputer, Universitas Dian Nuswantoro
toryaquino@gmail.com, romi@romisatriawahono.net, purwanto@dsn.dinus.ac.id

Abstract: Proses pembaharuan perangkat lunak diperlukan untuk menjaga kehandalan sebuah perangkat lunak agar bisa berjalan dengan baik. Sebuah perangkat lunak yang memiliki tingkat operasional tinggi sehingga tidak diperbolehkan untuk melakukan restart, memerlukan sebuah metode dimana metode tersebut dapat melakukan proses pembaharuan dengan cepat tanpa melakukan restart. Untuk mengatasi hal tersebut dapat digunakan metode *Java Customized Class Loader* (JCCL). Penerapan JCCL untuk melakukan pembaharuan perangkat lunak dengan tidak memperbolehkan restart memiliki kendala yaitu lambatnya proses pembaharuan. Pada penelitian ini, akan diterapkan *Java Dynamic Compilation* (JDC) untuk meningkatkan efisiensi waktu metode JCCL dalam melakukan pembaharuan perangkat lunak tanpa melakukan restart (JCCL+JDC). Untuk menguji coba metode yang diusulkan, digunakan aplikasi permainan *snake game*, aplikasi permainan ini telah digunakan juga pada penelitian sebelumnya. Berdasarkan hasil eksperimen, metode yang diusulkan dalam penelitian ini (JCCL+JDC) memiliki waktu yang lebih efisien dari metode JCCL sebelum dioptimalkan dengan metode JDC. Waktu yang dihasilkan dari metode yang diusulkan menyesuaikan kompleksitas algoritma dari setiap *method* yang ditambahkan dalam proses pembaharuan.

Keywords: Pembaharuan Perangkat Lunak, *Java Customized Class Loader, Java Dynamic Compilation*

1 PENDAHULUAN

Pembaharuan perangkat lunak pada saat *runtime* atau yang disebut dengan *Dynamic Software Update* (DSU) adalah teknik melakukan pembaharuan perangkat lunak ketika perangkat lunak tersebut sedang berjalan tanpa menimbulkan waktu jeda, oleh karena itu efisiensi waktu atau mengurangi waktu tunggu dari proses pembaharuan perangkat lunak merupakan hal yang diperlukan (Seifzadeh, Abolhassani, & Moshkenani, 2012). DSU juga disebut sebagai *on-the-fly program modification*, *online version change*, *hot-swap* dan *dynamic software maintenance* (Orso, Rao, & Harrold, 2002).

Pembaharuan perangkat lunak pada umumnya dilakukan untuk memperbaiki kesalahan kode program dan prosedur yang terdapat pada sebuah perangkat lunak. Ketika memperbaharui perangkat lunak, pada umumnya harus melakukan *restart* pada perangkat lunak yang telah diperbaharui sebelum melihat hasil pembaharuannya. Hal seperti ini mengurangi nilai efisiensi bahkan mengganggu (Kim, Tilevich, & J, 2010). Setelah memodifikasi dan mengkompilasi kode program yang baru, akan lebih baik jika hasilnya dapat secara langsung dimasukkan ke dalam perangkat lunak yang sedang berjalan tanpa perlu menghentikan perangkat lunak atau menunggu proses yang sedang berjalan selesai. Sebagai contoh memodifikasi *event* yang berada dalam sebuah tombol, tidak perlu menutup seluruh perangkat lunak yang sedang berjalan atau menunggu proses lain selesai (Würthinger, Wimmer, & Stadler, 2013).

Untuk menangani permasalahan pembaharuan perangkat lunak secara dinamis dimana tidak memperbolehkan komputer untuk *restart* dan mengharuskan proses pembaharuan yang efisien agar tidak terjadi jeda, terdapat beberapa pendekatan atau metode. Metode tersebut adalah *Customized Java Virtual Machine* (CJVM), *Java Wrapper* (JW), Jav Adaptor (JA) dan *Java Customized Class Loader* (JCCL).

Menurut Pukall (Pukall et al., 2011) CJVM dapat menyelesaikan permasalahan mengenai pembaharuan perangkat lunak pada saat *runtime*, akan tetapi metode ini tergantung dengan versi dari *Java Virtual Machine* (JVM). Hal ini menyebabkan metode CJVM tidak fleksibel untuk diterapkan pada seluruh versi JVM. Masalah rendahnya tingkat fleksibilitas metode CJVM dapat diatasi oleh metode JW, metode ini tidak tergantung pada JVM versi tertentu sehingga lebih fleksibel. Terlepas dari kemampuan JW yang fleksibel, JW memiliki permasalahan pada kemampuan untuk memperbaharui perangkat lunak. JW hanya bisa mengganti isi dari *method* pada sebuah kelas Java. Dikarenakan metode JW hanya bisa mengganti isi *method*, maka kekurangan ini dicoba diatasi oleh metode JA. Untuk mengatasi masalah yang terdapat pada JW diusulkan metode JA dimana metode ini dapat mengatasi permasalahan terbatasnya kemampuan melakukan pembaharuan yang dimiliki JW. Metode JA dapat mengganti seluruh bagian dari kode program serta fleksibel untuk dapat diterapkan pada seluruh versi JVM. Walaupun JA fleksibel terhadap seluruh versi JVM, akan tetapi efisiensi proses pembaharuan dari metode ini bukan menjadi tujuan utama,tujuan utama dari metode ini adalah agar bisa mengganti seluruh bagian kode program dan tidak tergantung terhadap JVM versi tertentu. Karena keterbatasan efisiensi proses pembaharuan, maka meningkatkan efisiensi proses pembaharuan menjadi tujuan lebih lanjut dari penelitian tentang JA. Permasalahan perlunya mengurangi waktu pada proses pembaharuan perangkat lunak juga terdapat pada penelitian Thomas Wuthingler (Würthinger et al., 2013), dalam penelitiannya dilakukan pendekatan dengan metode CJVM dimana dalam penelitian tersebut mengurangi waktu eksekusi dari kode program yang lama, merupakan langkah yang dapat dilakukan dalam penelitian berikutnya.

Metode-metode di atas memiliki kode sumber yang tertutup sehingga tidak bisa diteliti oleh umum untuk dikembangkan lebih lanjut. Melihat hal tersebut para peneliti DSU dan perangkat lunak yang memiliki waktu operasional tinggi lebih banyak yang menggunakan metode JCCL, metode JCCL dapat berjalan pada seluruh versi JVM. Metode JCCL melakukan pembaharuan dengan memodifikasi C*lass Loader* yang dimiliki oleh Java. *Class Loader* merupakan kelas dalam Java yang digunakan untuk memuat dan mengatur kelas yang sedang berjalan pada JVM secara dinamis, teknik ini secara

umum banyak digunakan dalam melakukan pembaharuan perangkat lunak yang sedang berjalan (Zhang, 2007). Kelas ini dapat dimodifikasi untuk dapat melakukan proses pemuatan ulang kelas kedalam JVM secara dinamis. Metode ini telah digunakan oleh OSGi Service Platform, Oracles FastSwap dalam memperbaharui komponennya yang sedang berjalan di JVM serta Javeleon yang memungkinkan pembaharuan secara fleksibel dan netBeans yang berbasis perangkat lunak juga telah menerapkan metode ini (Pukall et al., 2011). Metode ini memiliki kode sumber terbuka untuk umum, sehingga JCCL dapat dikembangkan lagi agar waktu proses pembaharuan lebih cepat jika jumlah *method* yang terdapat pada perangkat lunak terus bertambah (Pukall et al., 2011).

Walaupun metode JCCL dapat berjalan pada seluruh versi JVM akan tetapi metode ini mengalami penurunan efisiensi seiring dengan bertambahnya jumlah *method* pada kelas yang akan diperbaharui (Pukall et al., 2011). Dalam penelitian Zhang tentang *Dynamic Update Transactions*, digunakan metode JCCL yang dikombinasikan dengan metode *Java Reflection* untuk melakukan proses pembaharuan perangkat lunak secara dinamis pada seluruh versi JVM (Zhang, 2007). Pada penelitian tersebut proses pembaharuan dilakukan terhadap seluruh bagian program ketika *runtime* sehingga membutuhkan proseses pembaharuan yang besar pada program yang sedang berjalan sehingga berdampak pada berkurangnya efisiensi proses pembaharuan (Pukall et al., 2011).

Untuk meningkatkan efisiensi eksekusi kode program, terdapat metode yang bisa digunakan, yaitu *Java Dynamic Compilation* (JDC). JDC adalah metode yang telah digunakan oleh Sun hotSpot dan IBM's Jalapeno untuk mengoptimasi serta meningkatkan efisiensi eksekusi kode Java pada saat *runtime* sehingga waktu eksekusi program menjadi lebih efsien (Hayden, Smith, & Foster, 2012). JDC meningkatkan kinerja JVM dengan cara menjalankan kode atau *method* Java yang telah dikompilasi akan tetapi belum dimuat ke dalam JVM sebelumnya (Fong, 2012). Berdasarkan hal tersebut, dalam penelitian ini akan digunakan metode JDC untuk meningkatkan efisiensi JCCL dalam melakukan proses pembaharuan perangkat lunak pada saat *runtime*.

2 PENELITIAN TERKAIT

Pada penelitian Zhang (Zhang, 2007), dilakukan pendekatan dengan menggunakan metode JCCL berbasis *Java Reflection*. Penelitian ini menyelesaikan permasalahan dari penelitian sebelumnya, dimana proses pembaharuan hanya terbatas memperbaharui *method* pada kelas yang sama dan harus mengubah versi dari JVM sehingga harus melakukan modifikasi terhadap setiap versi JVM. Metode ini menggantikan *Class Loader* Java, Sehingga perangkat lunak yang akan melakukan proses pembaharuan pada saat *runtime* harus menerapkan *Class Loader* yang telah dimodifikasi agar bisa melakukan pembaharuan pada seluruh Kelas dan *method* yang ada tanpa melakukan perubahan struktur atau versi dari JVM. Terdapat dua tahap dalam menggunakan metode JCCL, tahap pertama adalah memanggil objek menggunakan *Class Loader* sendiri dan tahap kedua mengubah *method* dengan menggunakan *Java Reflection*.

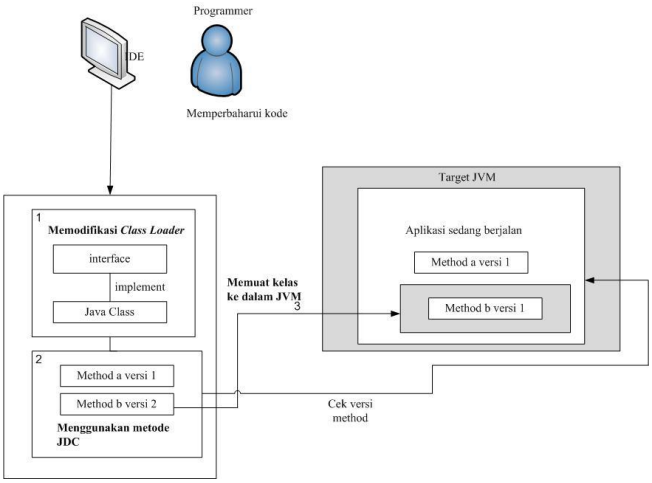
Pendekatan pembaharuan perangkat lunak menggunakan JCCL juga dilakukan dalam penelitiannya Kai (Gregersen & Koskimies, 2012). Penelitian ini memfokuskan pembuatan sebuah *tool* yang diintegrasikan ke dalam *framework* tertentu, sehingga perlu modifikasi ulang apabila terdapat *framework* baru. Dalam penelitian tersebut, memungkinkan beberapa versi dari objek untuk secara bersama berada dalam sistem yang sedang berjalan. Hal ini dapat terjadi dengan cara menggunakan Java Bootstrap atau disebut juga dengan (Javeleon) yang menciptakan *Class Loader* baru untuk setiap versi baru dari perangkat lunak yang akan diperbaharui. Cara keja dari metode yang diusulkan ini adalah dengan mengubah objek yang sudah berjalan di dalam JVM kedalam versi yang lebih baru.

Perbedaan yang terjadi pada metode JCCL berbasis Java Reflection dan metode JCCL berbasi Java Boostrap (Javeleon) adalah pada metode pembaharuan yang dilakukan, metode yang pertama melakukan proses pembaharuan dengan memuat ulang seluruh *method* sedangkan metode yang kedua melakukan proses pembaharuan dengan mengubah objek yang sudah berjalan pada JVM kedalam versi yang lebih baru.

3 METODE YANG DIUSULKAN

Class Loader standar yang dimiliki Java merupakan kelas yang terdapat pada Java untuk digunakan sebagai kelas yang bertugas memuat kelas dari program yang dibuat ke dalam JVM. Kelas ini dapat dimodifikasi dan mengontrol apa saja yang harus dilakukan oleh JVM. Dalam penelitian ini, *Class Loader* standar yang dimiliki Java akan dimodifikasi dengan menambahkan metode JDC sehingga dapat digunakan untuk memperbaharui perangkat lunak pada saat *runtime* dengan efisien. Proses pembaharuan perangkat lunak akan menjadi efisien karena metode JDC akan memeriksa *method* mana yang diperbaharui, sehingga hanya *method* itulah yang akan dimuat ulang ke dalam JVM.

Metode yang diusulkan dalam penelitian ini adalah JCCL berbasis JDC seperti pada Gambar 1.



Gambar 1 Metode yang Diusulkan

Perbedaan dengan metode sebelumnya yang menggunakan pendekatan JCCL terletak pada teknik pembaharuan kelas. Metode yang diusulkan yaitu JCCL berbasis JDC hanya memuat ulang *method* yang diperbaharui saja, *method* lama yang tidak diperbaharui dibiarkan tetap berjalan pada JVM tanpa dimuat ulang seperti pada Tabel 1.

Tabel 1 Perbedaan Dengan Penelitian Sebelumnya

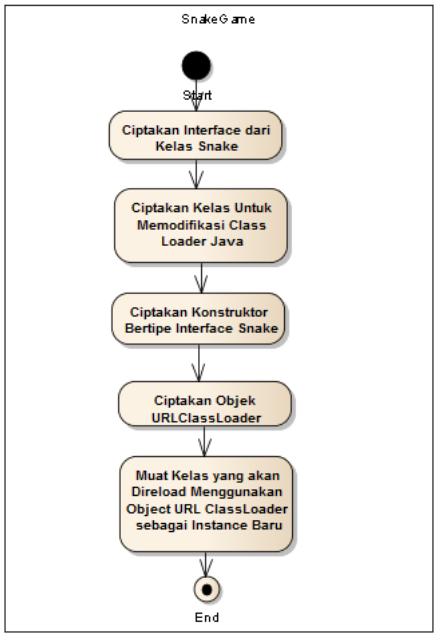
Penelitian	Metode	Perbedaan Dengan Penelitian Lainnya
Java Customized Class Loader berbasis Java Reflection (S. Z. S. Zhang dan L. H. L. Huang, 2007)	JCCL berbasis Java Reflection	Pembaharuan dilakukan dengan memuat ulang seluruh *method* yang terdapat pada JVM
Java Customized Class Loader berbasis Java Bootstrap (Gregersen dan Allan Raundahl, 2012) Koskimies dan Kai, 2012	JCCL berbasis Java boostrap	Pembaharuan dilakukan dengan mengubah objek yang terdapat pada JVM kedalam versi yang baru
Metode yang diusulkan (JCCL+JDC)	JCCL berbasis JDC	Pembaharuan dilakukan hanya dengan memuat ulang *method* yang diperbaharui saja, *method* lama yang tidak diperbaharui dibiarkan tetap berjalan pada JVM tanpa dimuat ulang

Metode yang diusulkan dalam penelitian ini memiliki langkah-langkah pembaharuan sebagai berikut:

1. Memodifikasi *Class Loader*
2. Menggunakan Metode JDC
3. Memuat Kelas ke dalam JVM

1. Memodifikasi Class Loader

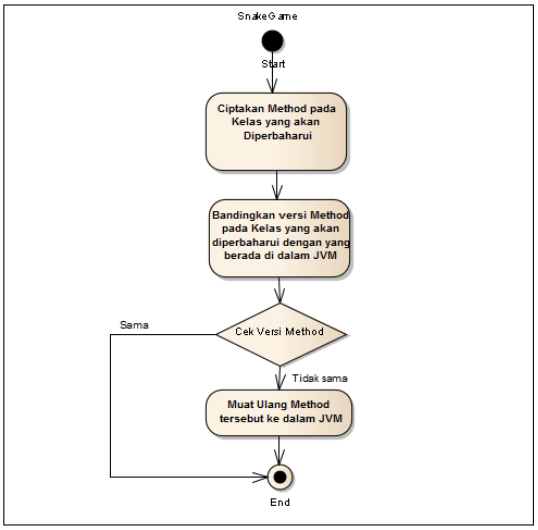
Dalam penelitian ini, *Class Loader* standar yang dimiliki Java perlu dimodifikasi agar dapat digunakan untuk memuat kelas kedalam JVM pada saat *runtime* dengan waktu seminimal mungkin. Modifikasi yang dilakukan memanfaatkan kelas yang terdapat pada Java yaitu *URL Class Loader*. Adapun cara penggunaan *Class Loader* standar yang dimiliki Java dan *URL Class Loader* adalah dengan membuatkan *interface* pada kelas yang akan diperbaharui dan selanjutnya *interface* tersebut akan digunakan oleh *Class Loader* yang dimodifikasi sebagai tipe konstruktor. *URL Class Loader* bertugas memanggil kelas yang akan dimuat ke dalam JVM. Setelah menciptakan objek *URL Class Loader*, selanjutnya objek tersebut bertugas memuat ulang kelas dari program yang akan diperbaharui ke dalam JVM menggunakan *method loadClass* dan menciptakan *instance* baru Gambar 2.



Gambar 2 Diagram Aktifitas Memodifikasi *ClassLoader*

2. Menggunakan Metode JDC

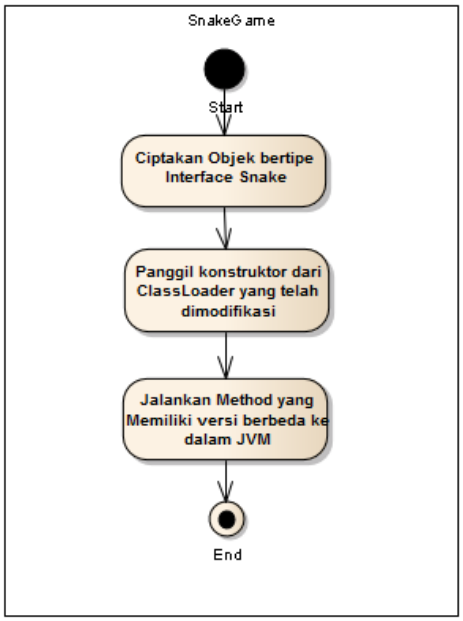
Dalam menggunakan metode JDC, kelas yang akan dimuat ke dalam JVM dibuatkan versi pada setiap *method*. Hal tersebut berfungsi, agar *method* yang tidak memiliki perubahan tidak dijalankan di dalam JVM. *Method* yang akan dijalankan di dalam JVM hanya *method* baru yang sebelumnya belum pernah dimuat ke dalam JVM. Di dalam Gambar 3 terdapat proses untuk mengecek apakah versi *method* yang terdapat di dalam kelas utama yang sudah dimuat di dalam JVM sama dengan versi *method* yang terdapat pada kelas yang akan diperbaharui. Jika sama maka *method* yang terdapat di dalam kelas yang akan diperbaharui tersebut dianggap sebagai *method* lama dan tidak akan dimuat ulang ke dalam JVM, jika versi *method* tersebut berbeda dengan yang ada di dalam JVM maka *method* tersebut dianggap sebagai *method* baru dan akan dimuat ulang ke dalam JVM.



Gambar 3 Diagram Aktifitas Penggunaan Metode JDC

3. Memuat Kelas ke Dalam JVM

Setelah kelas yang akan diperbaharui memiliki *method* dengan kode untuk mengecek versi, langkah terakhir adalah memuat kelas yang akan diperbaharui ke dalam JVM dengan menciptakan objek dari kelas yang akan diperbaharui meggunakan JCCL yang telah dimodifikasi dan ditambahakan metode JDC seperti pada Gambar 4.



Gambar 4 Diagram Aktifitas Memuat Kelas ke dalam JVM

4 HASIL PENELITIAN

Untuk mengetahui efisiensi waktu dari metode yang diusulkan, maka dilakukan ujicoba terhadap metode JCCL sebelum dioptimalkan dengan JDC dan sesudah dioptimalkan dengan JDC mengunakan data uji *snake game.* Perangkat lunak ini didapat dari situs pengembang perangkat lunak Java

Proses pengujian dilakukan sebanyak 30 kali percobaan menggunakan tiga jenis komputer yang memiliki spesifikasi berbeda. Komputer yang digunakan memiliki spesifikasi yang dapat mewakili dari masing-masing periode komputer, hal ini untuk menguji apakah metode yang diusulkan dapat berjalan dengan baik pada komputer spesifikasi tinggi,rendah dan sedang. Kemudian dari masing-masing spesifikasi komputer dilakukan 10 kali percobaan sehingga menghasilkan total 30 kali percobaan. Adapun ketiga spesifikasi komputer yang digunakan tertera pada Tabel 2.

Tabel 2 Spesifikasi Komputer Untuk Uji Coba Metode Terhadap Data Uji

Spesifikasi	Prosesor	Memori	Sistem Operasi
Tinggi	Core I3-2330M 2.20 GHz	2.00 GB	Windows 7
Sedang	Core 2 duo 2.10 GHz	1.99 GB	Windows XP
Rendah	Pentium 4 2.4 GHz	1 GB	Windows XP

Pengujian dengan tiga spesifikasi komputer diatas dilakukan terhadap masing-masing *method* yang dimiliki metode JCCL dan JCCL+JDC yaitu (doUpdate,reload dan loadClass). Adapun hasil pengukuran efisiensi dari masing masing *method* adalah sebagai berikut:

4.1 doUpdate

Berdasarkan pengujian yang telah dilakukan terhadap *method* doUpdate yang dilakukan menggunakan komputer spesifikasi tinggi, sedang dan rendah didapatkan hasil bahwa grafik *method* doUpdate dari metode JCCL+JDC lebih stabil dibanding dengan grafik hasil *method* doUpdate dari method JCCL yang terus naik. Hal tersebut disebabkan karena pada metode JCCL *method* doUpdate akan mengeksekusi seluruh *method* yang terdapat di dalam kelas, sehigga jika ditambah *method* maka waktu eksekusi akan terus meningkat. Hal demikian tidak terjadi pada metode yang diusulkan yaitu JCCL+JDC, pada metode JCCL+JDC *method* yang dieksekusi hanyalah *method* yang ditandai sebagai *method* baru saja (Kazi, Chen, Stanley, & Lilja, 2000). Sehingga waktu eksekusi yang dihasilkan tergantung dengan alghoritma masing-masing *method,* jika *method* yang akan dieksekusi memiliki alghoritma yang lebih panjang dari *method* sebelumnya maka waktu yang dihasilkan akan lebih tinggi.

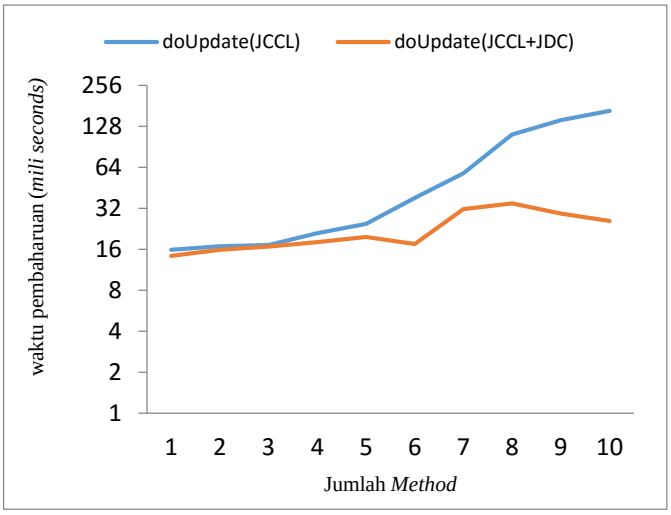
4.1.1 Komputer Spesifikasi Tinggi

Dalam pengujian yang dilakukan menggunakan komputer spesifikasi tiggi terhadap *method* doUpdate, didapatkan hasil bahwa waktu yang dihasilkan metode yang diusulkan yaitu JCCL+JDC memiliki waktu yang lebih efisien. Sedangkan metode JCCL sebelum dioptimalkan dengan metode JDC memiliki waktu yang terus meningkat. Hasil pengujian ini terlihat seperti ada Tabel 3 serta grafik perbedaan kedua *method* terlihat pada Gambar 5.

Tabel 3 Hasil Eksperimen *Method* doUpdate Pada Komputer Spesifikasi Tinggi

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	doUpdate	**doUpdate**
1	15,87	14,3
2	16,8	15,8
3	17,2	16,7
4	21	18
5	24,6	19,7
6	38	17,5
7	57,9	31,5
8	111	34,7
9	142	29,2
10	166	25,8

Method doUpdate dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih tinggi jika dilihat setiap kali pengujian dengan menambahkan jumlah method. Akan tetapi jika dilihat dari keseluruhan pengujian, *method* doUpdate memiliki fluktuasi waktu yang lebih stabil sedangkan method doUpdate dari metode JCCL memiliki fluktuasi waktu yang terus meningkat. Pada Tabel 3 diatas, terlihat bahwa pada percobaan yang ke-enam dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah hal tersebut dikarenakan method yang ke-enam memiliki alur yang lebih singkat dibandingkan method yang lainnya. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 0,63% dimana *method* JCCL+JDC lebih unggul.



Gambar 5 Grafik Perbandingan *method* doUpdate Pada Komputer Spesifikasi Tinggi

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 4 Nilai t hitung yang dihasilkan adalah 2,353 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,043 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara method doUpdate pada masing-masing metode.

Tabel 4 Hasil Uji Beda *Method* doUpdate Pada Komputer Spesifikasi Tinggi

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	38.71700	52.03145	16.45379	1.49594	75.93806	2.353	9	.043

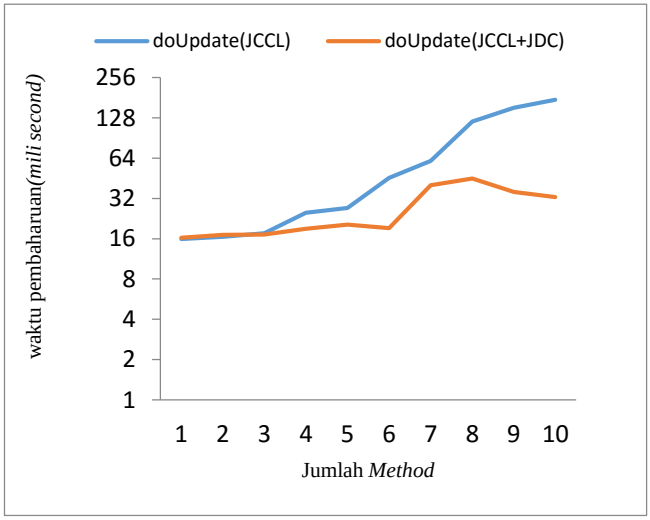
4.1.2 Komputer Spesifikasi Sedang

Metode yang diusulkan yaitu JCCL+JDC juga menunjukkan hasil yang bagus Gambar 6 ketika dilakukan pengujian menggunakan komputer spesifikasi sedang terhadap *method* doUpdate. Pada Gambar 6, grafik metode JCCL+JDC memiliki waktu yang lebih efisien dibanding metode JCCL pada setiap kali jumlah *method* ditambahakan. Hasil pengujian ini terlihat seperti ada Tabel 5 serta grafik perbedaan kedua *method* terlihat pada Gambar 6.

Tabel 5 Hasil Eksperimen *Method* doUpdate Pada Komputer Spesifikasi Sedang

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	doUpdate	**doUpdate**
1	15,87	16,3
2	16,5	17,1
3	17,6	17,2
4	25	19
5	27,2	20,4
6	45,7	19,2
7	61,2	40,2
8	120	45,1
9	152	35,7
10	175	32,8

Method doUpdate dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih tinggi jika dilihat setiap kali pengujian dengan menambahkan jumlah *method*. Akan tetapi jika dilihat dari keseluruhan pengujian, *method* doUpdate memiliki fluktuasi waktu yang lebih stabil sedangkan *method* doUpdate dari metode JCCL memiliki fluktuasi waktu yang terus meningkat. Pada Tabel 5 , terlihat bahwa pada percobaan yang ke enam dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah hal tersebut dikarenakan *method* yang ke enam memiliki alur yang lebih singkat dibandingkan *method* yang lainnya. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 0,59% dimana *method* JCCL+JDC lebih unggul seperti yang terlihat pada Gambar 6.



Gambar 6 Grafik Perbandingan *method* doUpdate Pada Komputer Spesifikasi Sedang

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan pada Tabel 6 Nilai t hitung yang dihasilkan adalah 2,352 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,043 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara *method* doUpdate pada masing-masing metode. Hasil pengujian ini terlihat seperti pada Tabel 6 serta grafik perbedaan kedua *method* terlihat pada Gambar 6.

Tabel 6 Hasil Uji Beda *Method* doUpdate Pada Komputer Spesifikasi Sedang

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	39.30700	52.84393	16.71072	1.50473	77.10927	2.352	9	.043

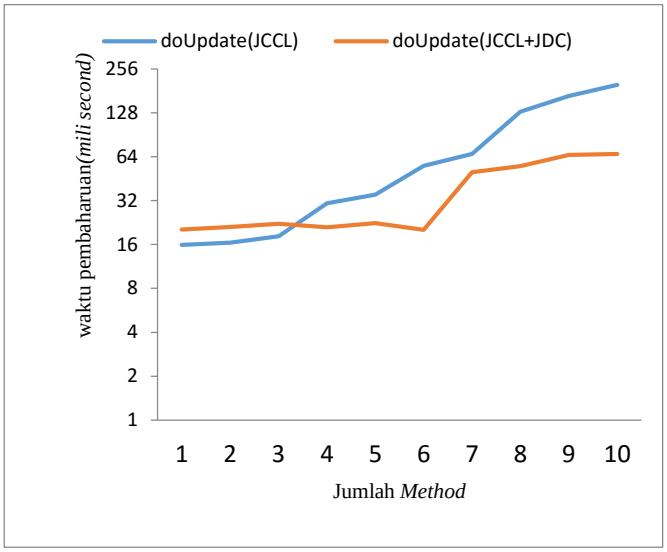
4.1.3 Komputer Spesifikasi Rendah

Pada pengujian yang dilakukan terhadap komputer spesifikasi rendah, metode yang diusulkan yaitu JCCL+JDC memiliki waktu yang kurang efisien pada awal pengujian tambar 7, hal tersebut disebabkan karena spesifikasi komputer yang rendah digunakan untuk memperbaharui *method* yang memiliki alghoritma panjang. Namun ketikan jumlah *method* terus ditambahkan, metode yang diusulkan yaitu JCCL+JDC menunjukkan hasil waktu yang bagus dibandingkan metode JCCL sebelum dioptimalkan dengan JDC, hal ini disebabkan karena metode yang diusulkan memiliki kemampuan untuk mengeksekusi hanya *method* baru saja.

Tabel 7 Hasil Eksperimen *Method* doUpdate Pada Komputer Spesifikasi Redah

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	doUpdate	**doUpdate**
1	15,87	20,3
2	16,5	21,1
3	18,2	22,2
4	30,7	21
5	35,2	22,4
6	55,5	20,2
7	66,9	50,2
8	130	55,1
9	167	65,7
10	199	66,8

Method doUpdate dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih tinggi jika dilihat setiap kali pengujian dengan menambahkan jumlah *method*. Akan tetapi jika dilihat dari keseluruhan pengujian, *method* doUpdate memiliki fluktuasi waktu yang lebih stabil sedangkan *method* doUpdate dari metode JCCL memiliki fluktuasi waktu yang terus meningkat. Pada Tabel 7 , terlihat bahwa pada percobaan yang ke 4,5 dan 6 dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah hal tersebut dikarenakan *method* yang ke-empat memiliki alur yang lebih singkat dibandingkan *method* yang lainnya. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 0,50% dimana *method* JCCL+JDC lebih unggul seperti yang terlihat pada Gambar 7.



Gambar 7 Grafik Perbandingan *method* doUpdate Pada Komputer Spesifikasi rendah

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 8 nilai t hitung yang dihasilkan adalah 2,393 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,040 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara method doUpdate pada masing-masing metode.

Tabel 8 Hasil Uji Beda *Method* doUpdate Pada Komputer Spesifikasi Rendah

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	36.98700	48.88519	15.45885	2.01664	71.95736	2.393	9	.040

4.2 Reload

Hasil pengujian yang dilakukan terhadap *method* reload menggunakan komputer spesifikasi tinggi, sedang dan rendah menunjukkan hasil bahwa grafik *method* reload pada metode JCCL memiliki waktu yang lebih efisien dibanding *method* reload pada metode JCCL+JDC. Hal tersebut disebabkan karena *method* reload pada metode JCCL+JDC memiliki alghoritma tambahan untuk mengecek versi dari *method* yang akan diperbaharui sehingga mengkonsumsi waktu untuk melakukan hal tersebut. Sedangkan *method* reload pada metode JCCL tidak memiliki alghoritma tambahan untuk mengecek versi dari *method* yang akan diperbaharui, sehingga waktu yang dihasilkan lebih efisien.

4.2.1 Komputer Spesifikasi Tinggi

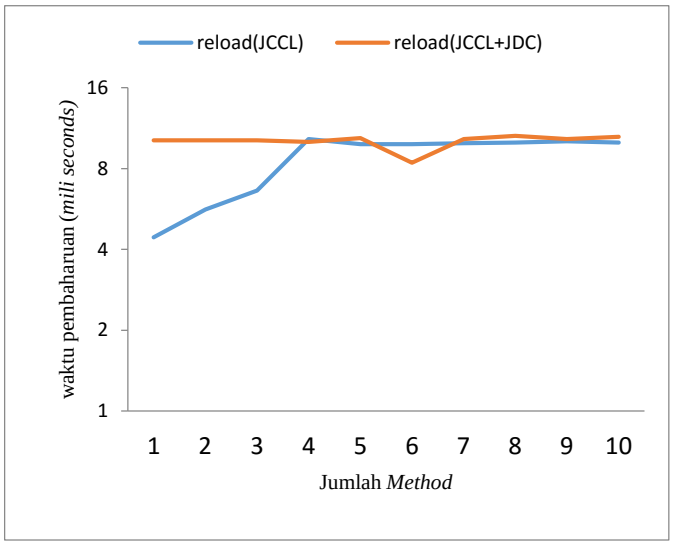
Pada pengujian menggunakan komputer spesifikasi tinggi, seperti terlihat pada Gambar 8. metode JCCL lebih baik dalam konsumsi waktu dibandingkan dengan metode yang diusulkan yaitu JCCL+JDC. Hal tersebut dikarenakan pada metode JCCL+JDC terdapat alghoritma untuk mengecek versi dari *method* yang akan diperbaharui. Sehingga alghoritma ini akan

mengkonsumsi waktu lebih lama dibanding metode JCCL sebelum ditambahkan metode JDC. Hasil pengujian ini terlihat seperti ada tabel 9 serta grafik perbedaan kedua *method* terlihat pada gambar 8.

Tabel 9 Hasil Eksperimen *Method* reload Pada Komputer Spesifikasi Tinggi

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	reload	**reload**
1	4,44	10,2
2	5,64	10,2
3	6,63	10,2
4	10,3	10,04
5	9,86	10,4
6	9,87	8,41
7	9,95	10,3
8	9,99	10,6
9	10,1	10,3
10	10	10,5

Method reload dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebi tinggi, hal ini dikarenakan didalam *method* reload pada penelitian ini ditambahkan dengan metode JDC agar proses pembaharuan yang dilakukan *method* doUpdate lebih efisien. Pada Tabel 9, terlihat bahwa pada percobaan yang ke enam dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah hal terseut dikarenakan *method* yang ke empat memiliki alur yang lebih singkat dibandingkan *method* yang lainnya. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 72 % dimana *method* JCCL lebih unggul karena tidak memiliki alghoritma untuk mengecek versi *method*.



Gambar 8 Grafik Perbandingan *method* reload Pada Komputer Spesifikasi Tinggi

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 10 nilai t hitung yang dihasilkan adalah 1,861 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,096 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara *method* reload pada masing-masing metode.

Tabel 10 Hasil Uji Beda *Method* reload Pada Komputer Spesifikasi Tinggi

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	-1.39200	2.36567	.74809	-3.08430	.30030	-1.861	9	.096

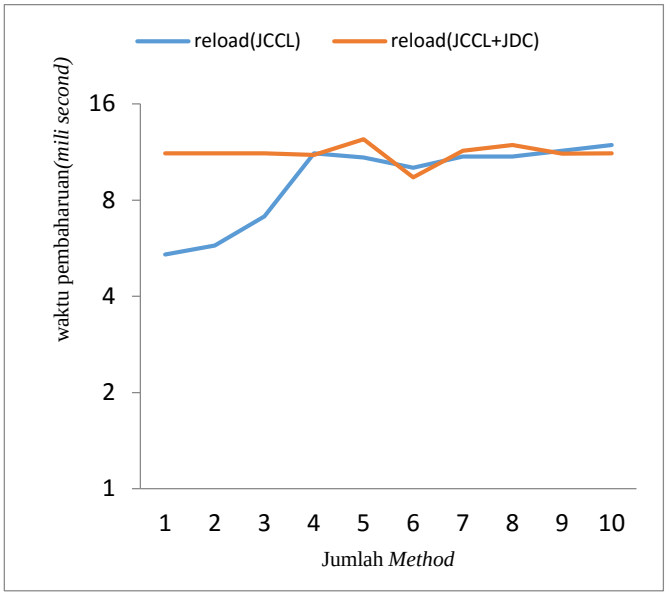
4.2.2 Komputer Spesifikasi Sedang

Hal yang sama terjadi ketika pengujian dilakukan pada komputer spesifikasi sedang. Pada pengujian menggunakan komputer spesifikasi sedang, seperti terlihat pada Gambar 9. metode JCCL lebih baik dalam konsumsi waktu dibandingkan dengan metode yang diusulkan yaitu JCCL+JDC. Perbedaan hanya terjadi pada waktu yang dihasilkan. Waktu yang dihasilkan lebih tinggi dibanding hasil waktu yang dihasilkan dari komputer spesifikasi tinggi, hal ini disebabkan karena spesifikasi komputer yang digunakan lebih rendah. Hasil pengujian ini terlihat seperti pada Tabel 11 serta grafik perbedaan kedua *method* terlihat pada Gambar 9.

Tabel 11 Hasil Eksperimen *Method* reload Pada Komputer Spesifikasi Sedang

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	reload	**reload**
1	5,41	11,2
2	5,77	11,2
3	7,12	11,2
4	11,2	11,08
5	10,86	12,4
6	10,1	9,42
7	10,94	11,4
8	10,95	11,9
9	11,4	11,17
10	11,9	11,2

Method reload dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih tinggi, hal ini dikarenakan didalam *method* reload pada penelitian ini ditambahkan dengan metode JDC agar proses pembaharuan yang dilakukan *method* doUpdate lebih efisien. Pada Tabel 7, terlihat bahwa pada percobaan yang ke enam dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah hal terseut dikarenakan *method* yang ke empat memiliki alur yang lebih singkat dibandingkan *method* yang lainnya. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 79,13 % dimana *method* JCCL lebih unggul karena tidak memiliki alghoritma untuk mengecek versi *method*.



Gambar 9 Grafik Perbandingan *method* reload Pada Komputer Spesifikasi Sedang

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan tabel 12 nilai t hitung yang dihasilkan adalah 2,078 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,067 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara method reload pada masing-masing metode.

Tabel 12 Hasil Uji Beda *Method* reload Pada Komputer Spesifikasi Sedang

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
					Lower	Upper			
Pair 1	jccl - jccljdc	-1.65200	2.51373	.79491	-3.45022	.14622	-2.078	9	.067

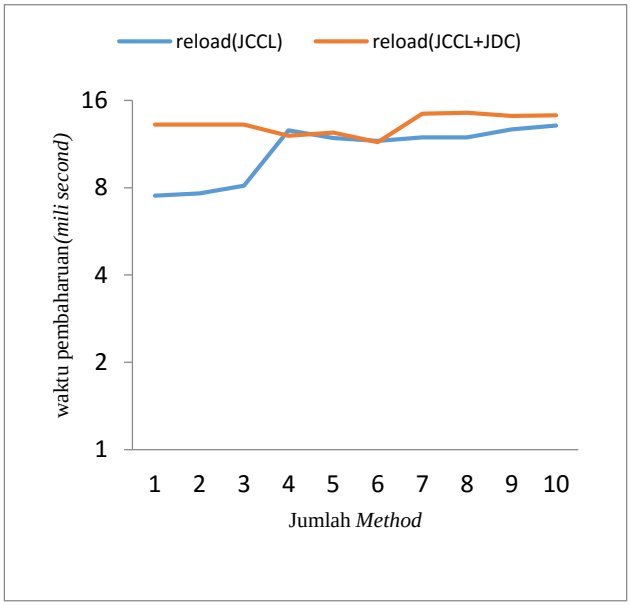
4.2.3 Komputer Spesifikasi Rendah

Hasil pengujian yang dilakukan dengan menggunakan komputer spesifikasi rendah Gambar 10, menghasilkan waktu yang lebih tinggi dibandingkan hasil pengujian menggunakan komputer spesifikasi tinggi dan sedang. Hal ini karena spesifikasi komputer yang digunakan lebih rendah dibanding spesifikasi komputer yang digunakan untuk menguji sebelumnya yaitu tinggi dan rendah. Hasil pengujian ini terlihat seperti pada Tabel 13 serta grafik perbedaan kedua *method* terlihat pada Gambar 10.

Tabel 13 Hasil Eksperimen *Method* reload Pada Komputer Spesifikasi Rendah

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	reload	**reload**
1	7,51	13,2
2	7,65	13,2
3	8,12	13,2
4	12,6	12,09
5	11,86	12,4
6	11,6	11,47
7	11,94	14,4
8	11,92	14,5
9	12,7	14,15
10	13,1	14,2

Method reload dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih tinggi, hal ini dikarenakan didalam *method* reload pada penelitian ini ditambahkan dengan metode JDC agar proses pembaharuan yang dilakukan *method* doUpdate lebih efisien. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 85,19 % dimana *method* JCCL lebih unggul karena tidak memiliki alghoritma untuk mengecek versi *method*.



Gambar 10 Grafik Perbandingan *method* reload Pada Komputer Spesifikasi rendah

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 14 nilai t hitung yang dihasilkan adalah 3,231 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,010 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara *method* reload pada masing-masing metode.

Tabel 14 Hasil Uji Beda *Method* reload Pada Komputer Spesifikasi Rendah

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	-2.38100	2.33053	.73698	-4.04816	-.71384	-3.231	9	.010

4.3 loadClass

Method loadClass ini bertugas untuk memasukkan kelas yang sudah diperbaharui kedalam JVM. Berdasarkan hasil percobaan yang dilakukan menggunakan komputer spesifikasi Tinggi,sedang dan rendah, didapatkan hasil bahwa grafik *method* loadClass pada metode yang diusulkan yaitu JCCL+JDC memiliki waktu yang lebih rendah atau efisien pada pengujian menggunakan komputer spesifikasi tinggi,sedang dan rendah. Kelas ini memiliki tugas yang ringan karena hanya bertugas memasukkan kelas yang berisi *method* yang telah diperbaharui ke dalam JVM. *Method* loadClass pada metode yang diusulkan yaitu JCCL+JDC memiliki waktu yang lebih unggul karena pada metode yang diusulkan, kelas yang dimasukkan hanya berisi *method* baru saja.

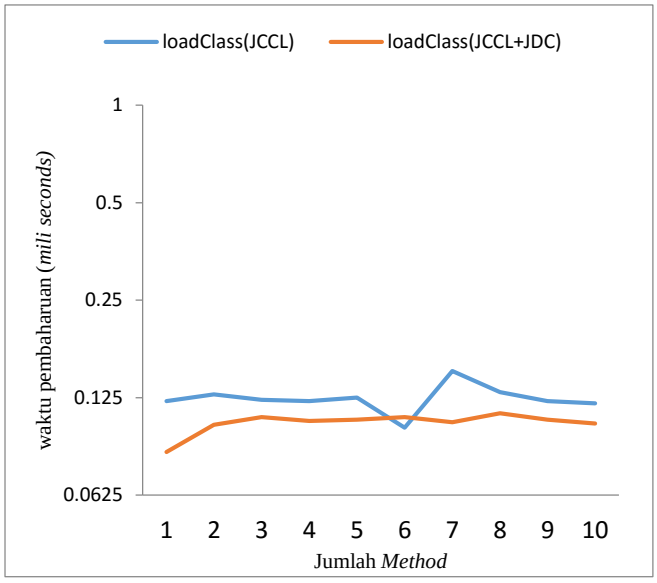
4.3.1 Komputer Spesifikasi Tinggi

Pengujian yang dilakukan menggunakan komputer spesifikasi tinggi Gambar 11, menunjukkan hasil bahwa metode JCCL+JDC memiliki fluktuasi waktu yang lebih baik dibanding metode JCCL. Hasil pengujian ini terlihat seperti pada Tabel 15 serta grafik perbedaan kedua *method* terlihat pada Gambar 11.

Tabel 15 Hasil Eksperimen *Method* loadClass Pada Komputer Spesifikasi Tinggi

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	loadClass	**loadClas**
1	0,122	0,085
2	0,128	0,103
3	0,123	0,109
4	0,122	0,106
5	0,125	0,107
6	0,101	0,109
7	0,151	0,105
8	0,13	0,112
9	0,122	0,107
10	0,12	0,104

Method loadClass dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih tinggi tetapi ada juga yang memiliki waktu lebih rendah, hal ini dikarenakan *method* loadClass bertugas memuat ulang kelas yang diperbaharui ke dalam JVM dengan menyertakan metode tambahan (JDC) agar kelas yang dimuat hanyalah kelas yang baru saja. Waktu yang dihasilkan oleh loadClass tergantung dari alghoritma setiap *method* yang ditambahkan serta spesifikasi komputer yang digunakan untk percobaan. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 0,15% dimana *method* JCCL+JDC lebih unggul.



Gambar 11 Grafik Perbandingan *method* loadClass Pada Komputer Spesifikasi Tinggi

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 16 nilai t hitung yang dihasilkan adalah 4,371 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,002 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara method reload pada masing-masing metode.

Tabel 16 Hasil Uji Beda *Method* loadClass Pada Komputer Spesifikasi Tinggi

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	.01970	.01443	.00456	.00938	.03002	4.317	9	.002

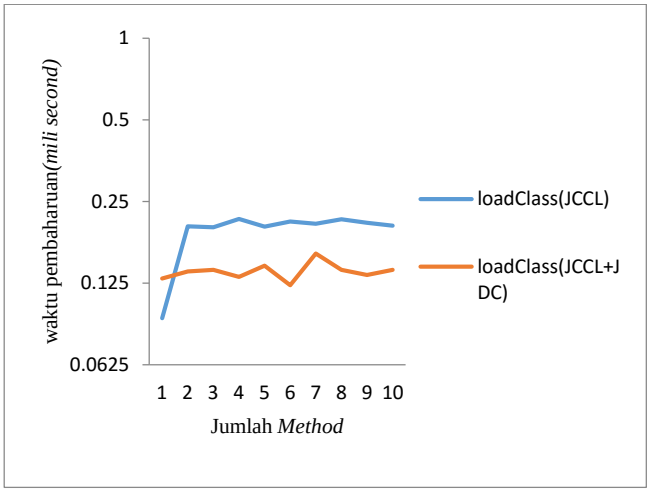
4.3.2 Komputer Spesifikasi Sedang

Pada pengujian *method* loadClass menggunakan komputer spesifikasi sedang Gambar 12, dihasilkan waktu dari metode yang diusulkan yaitu JCCL+JDC lebih unggul dibandingkan waktu metode JCCL. Hasil pengujian ini terlihat seperti pada Tabel 17 serta grafik perbedaan kedua *method* terlihat pada Gambar 12.

Tabel 17 Hasil Eksperimen *Method* loadClass Pada Komputer Spesifikasi Sedang

Jumlah *Method*	JCCL	JCCL+JDC
	loadClass	loadClass
1	0,093	0,13
2	0,203	0,138
3	0,201	0,14
4	0,216	0,132
5	0,202	0,145
6	0,211	0,123
7	0,207	0,161
8	0,215	0,14
9	0,209	0,134
10	0,204	0,14

Method loadClass dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah, hal ini dikarenakan *method* loadClass bertugas memuat ulang kelas yang diperbaharui ke dalam JVM dengan menyertakan metode tambahan (JDC) agar kelas yang dimuat hanyalah kelas yang baru saja. Waktu yang dihasilkan oleh loadClass tergantung dari alghoritma setiap *method* yang ditambahkan dan juga spesifikasi komputer yang digunakan untuk melakukan percobaan. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 0,29% dimana *method* JCCL+JDC lebih unggul.



Gambar 12 Grafik Perbandingan *method* loadClass Pada Komputer Spesifikasi Sedang

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 18 nilai t hitung yang dihasilkan adalah 5,132 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,001 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara method reload pada masing-masing metode.

Tabel 18 Hasil Uji Beda *Method* loadClass Pada Komputer Spesifikasi Sedang

	Paired Differences					t	df	Sig. (2-tailed)
	Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1 jccl - jccljdc	.05780	.03562	.01126	.03232	.08328	5.132	9	.001

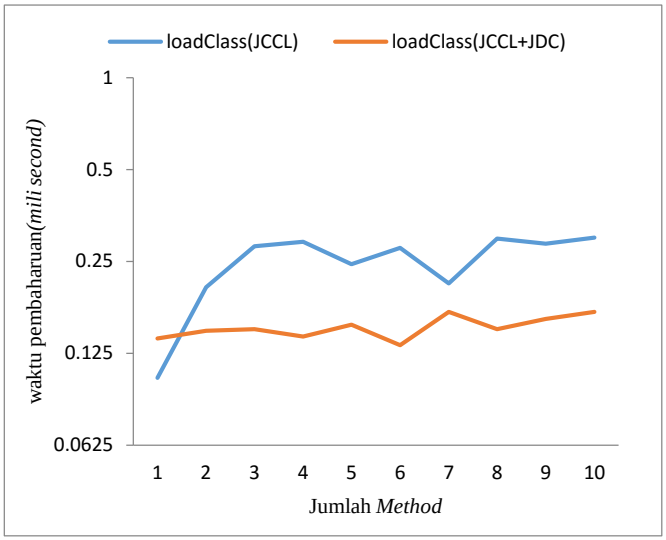
4.3.3 Komputer Spesifikasi Rendah

Pada pengujian *method* loadClass menggunakan komputer spesifikasi sedang Gambar 12, dihasilkan waktu dari metode yang diusulkan yaitu JCCL+JDC lebih unggul dibandingkan waktu metode JCCL. Pada pengujian yang dilakukan menggunakan komputer spesifikasi rendah ini grafik yang dihasilkan memiliki fluktuasi waktu yang kurang stabil dibandingkan pengujian menggunakan komputer spesifikasi rendah dan tinggi. Hasil pengujian ini terlihat seperti ada Tabel 19 serta grafik perbedaan kedua *method* terlihat pada Gambar 13.

Tabel 19 Hasil Eksperimen *Method* loadClass Pada Komputer Spesifikasi Rendah

Jumlah *Method*	JCCL	JCCL+JDC
	loadClass	loadClass
1	0,104	0,14
2	0,206	0,148
3	0,28	0,15
4	0,29	0,142
5	0,245	0,155
6	0,277	0,133
7	0,212	0,171
8	0,297	0,15
9	0,286	0,162
10	0,299	0,171

Method loadClass dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah, hal ini dikarenakan *method* loadClass bertugas memuat ulang kelas yang diperbaharui ke dalam JVM dengan menyertakan metode tambahan (JDC) agar kelas yang dimuat hanyalah kelas yang baru saja. Waktu yang dihasilkan oleh loadClass tergantung dari alghoritma setiap *method* yang ditambahkan dan juga spesifikasi komputer yang digunakan untuk melakukan percobaan. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 0,39% dimana *method* JCCL+JDC lebih unggul.



Gambar 13 Grafik Perbandingan *method* loadClass Pada Komputer Spesifikasi rendah

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 20 nilai t hitung yang dihasilkan adalah 5,127 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,001 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara method reload pada masing-masing metode.

Tabel 20 Hasil Uji Beda *Method* loadClass Pada Komputer Spesifikasi Rendah

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	.09740	.06008	.01900	.05442	.14038	5.127	9	.001

5 KESIMPULAN

Dari hasil penelitian yang dilakukan mulai dari tahap awal hingga proses pengujian, dapat disimpulkan bahwa Jumlah *method* yang terus bertambah pada perangkat lunak yang akan diperbaharui mempengaruhi efisiensi proses pembaharuan pada metode JCCL. Hal ini terbukti berdasarkan hasil pengujian metode JCCL bahwa grafik yang dihasilkan metode JCCL terus meningkat seiring bertambahnya jumlah *method*.

Dengan menerapkan metode JDC pada metode JCCL maka efisiensi proses pembaharuan perangkat lunak pada saat *runtime* meningkat walaupun jumlah *method* terus bertambah, hal ini menjawab pertanyaan penelitian: Apakah dengan menerapkan metode JDC pada metode JCCL dapat meningkatkan efisiensi proses pembaharuan perangkat lunak pada saat runtime apabila jumlah *method* terus bertambah. Hal tersebut didukung dari hasil pengujian metode yang diusulkan yaitu JCCL+JDC, dimana setiap pertambahan *method* memiliki fluktuasi waktu yang lebih stabil dibandingkan pertambahan *method* yang dilakukan pada metode JCCL dimana pada metode JCCL waktu yang dibutuhkan untuk melakukan proses pembaharuan terus meningkat.

Berdasarkan uji t yang telah dilakukan terhadap metode yang diuji yaitu JCCL dan JCCL+JDC yang dilakukan menggunakan komputer spesifikasi tinggi,sedang dan rendah serta dilakukan percobaan sebanyak 30 kali yang dilakukan pada tiga unit komputer dengan spesifikasi yang berbeda, dimana pada setiap komputer dilakukan penambahan *method* setiap kali percobaan dilakukan, maka didapat data perbedaan yang siknifikan antara kedua metode tersebut. Dalam hal ini metode JCCL+JDC memiliki waktu yang lebih efisien walaupun jumlah *method* terus bertambah.

DAFTAR REFERENSI

Fong, Y. L. A. S. (2012). Heuristic optimisation algorithm for Java dynamic compilation. *Engineering and Technology*, (February), 307–312. doi:10.1049/iet-sen.2011.0144

Gregersen, A. R., & Koskimies, K. (2012). Javeleon : An Integrated Platform for Dynamic Software Updating and its Application in Self- * systems. *Transactions on Software Engineering*.

Hayden, C. M., Smith, E. K., & Foster, J. S. (2012). Kitsune : Efficient , General-purpose Dynamic Software Updating for C. *OOPSLA*.

Kazi, I. H., Chen, H. H., Stanley, B., & Lilja, D. J. (2000). Techniques for obtaining high performance in Java programs. *ACM Computing Surveys*, *32*(3), 213–240.

Kim, D. K., Tilevich, E., & J, C. (2010). Dynamic Software Updates for Parallel High Performance Applications. *Dept. of Computer Science*.

Orso, A., Rao, A., & Harrold, M. (2002). A Technique for Dynamic Updating of Java Software. *18th IEEE International Conference on Software Maintenance*, *pp649658*.

Pukall, M., K, C., G, S., Grebhahn, A., Schr, R., & Saake, G. (2011). J AV A DAPTOR – Flexible Runtime Updates of Java Applications. *Softw. Pract. Exper.*, 1–33. doi:10.1002/spe

Seifzadeh, H., Abolhassani, H., & Moshkenani, M. S. (2012). A survey of dynamic software updating. *Wiley Online Lirary*. doi:10.1002/smr

Würthinger, T., Wimmer, C., & Stadler, L. (2013). Unrestricted and safe dynamic code evolution for Java ☆. *Science of Computer Programming*, *78*(5), 481–498. doi:10.1016/j.scico.2011.06.005

Zhang, S. (2007). Type-Safe Dynamic Update Transaction. *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, (60673116).

BIOGRAFI PENULIS

Tory Ariyanto. Menyelesaikan pendidikan S1 Teknik Informatika di Sekolah Tinggi Managemen Informatika (STMIK) Widya Pratama Pekalongan Indonesia, S2 Magister Teknik Informatika di Universitas Dian Nuswantoro Semarang Indonesia. Saat ini menjadi Dosen Pemrograman Komputer di STMIK Widya Pratama Pekalongan Indonesia. Minat penelitian saat ini adalah Software enginering.

Romi Satria Wahono. Menempuh pendidikan S1, S2 di bidang computer science di Saitama University, Jepang, dan Ph.D di bidang Software Engineering di Universiti Teknikal Malaysia Melaka. Saat ini menjadi dosen di Fakultas Ilmu Komputer di Universitas Dian Nuswantoro, Indonesia. Founder dan CEO PT Brainmatics Cipta Informatika, sebuah perusahaan pengembangan software di Indonesia. Minat penelitian saat ini adalah software engineering dan machine learning. Professional member dari ACM, PMI and IEEE.

Purwanto. Menempuh pendidikan S1 di universitas diponegoro Semarang Indonesia, S2 di **Purwanto**. Menempuh pendidikan S1 di universitas diponegoro Semarang Indonesia, S2 di Sekolah Tinggi Teknologi Informasi Benarif Indonesia dan Ph.D di multimedia university Malaysia. Saat ini menjadi dosen pascasarjana Magister Teknik Informatika di Universitas Dian Nuswantoro Semarang, Indonesia.