KONSEP BARU SISTEM WAKTU-NYATA DAN ALGORITMA PENJADWALAN EDELF (*EARLIEST DEADLINE EARLIEST LIVELINE FIRST*)

Fazmah Arif Yulianto[1]; Kuspriyanto[2]

[1]Jurusan Teknik Informatika STTTelkom, Bandung; [2]Departemen Teknik Elektro ITB, Bandung
[1]fay@stttelkom.ac.id; [2]kuspriyanto@lskk.ee.itb.ac.id

Abstrak

Paper ini menawarkan konsep baru dalam memodelkan sistem waktu-nyata, dengan memasukkan *liveline* disamping *deadline* yang merupakan ciri model konvensional. Hal ini dilakukan karena pada kenyataannya, sebagian sistem waktu-nyata tidak hanya dibatasi oleh batas waktu maksimal, tapi juga dibatasi oleh batas waktu tercepat dalam penyajian respon. Dengan adanya *liveline*, maka beberapa hal harus turut pula disesuaikan, antara lain : strategi waktu pemrosesan data dan penyajian respon serta algoritma penjadwalan proses. Dengan berasumsi bahwa sistem waktu nyata terdiri dari dua modul, yakni : *calculator* dan *actuator*, maka diperkenalkan strategi tanpa jeda dan strategi dengan jeda antara eksekusi dua modul tersebut. Algoritma penjadwalan yang dimodifikasi adalah EDF (*Earliest Deadline First*) menjadi EDELF (*Earliest Deadline Earliest Liveline First*). Dengan menggunakan contoh kasus, diperlihatkan bahwa EDELF memberikan performansi rata-rata yang lebih baik dibanding EDF

Kata kunci : sistem waktu nyata, *liveline*, EDELF

Abstract

This paper propose a new concept of real-time system model, by adding liveline to the conventional model which is include only deadline. It is because some types of real-time system restricted by both lattest and earliest response time. The consequence of adding liveline is a modification of several aspects in real-time model. Assumed that real-time system consists of two main moduls (calculator and actuator), we now have two strategies to process data and to 'actuate' the result. First, without delay between the execution of these two moduls, and the second is where there is delay inbetween. EDF (*Earliest Deadline First*) scheduling algorithm has been modified to became EDELF (*Earliest Deadline Earliest Liveline First*). The case study shows that EDELF average performance is better than EDF.

Keywords : real-time system, liveline, EDELF

1. Pendahuluan

Sistem waktu-nyata banyak digunakan dalam pengendalian kerja perangkat keras maupun eksekusi proses oleh komputer. Biasanya terdapat sensor yang melakukan pemantauan lingkungan secara periodik. Komputer kemudian memberikan respon terhadap kondisi lingkungan dengan mengirimkan sinyal (perintah) ke aktuator yang selanjutnya melakukan tindakan bagi lingkungan.

Selain kejadian-kejadian periodik, mungkin juga muncul kejadian tidak terduga atau kejadian yang muncul sewaktu-waktu saja. Pada semua kasus tersebut, terlihat adanya waktu yang membatasi penyajian respon. Kemampuan komputer untuk dapat memenuhi batasan tersebut ditentukan oleh sumber daya yang dimilikinya untuk dapat melakukan komputasi yang diperlukan dalam waktu tertentu.

Jika ada sejumlah kejadian muncul pada interval waktu yang tidak terlalu lama, maka komputer harus dapat melakukan penjadwalan komputasi agar penyajian respon tidak melanggar batas waktu yang telah ditetapkan. Jika tidak tersedia cukup sumber daya, maka ada beberapa akibat mungkin muncul, yakni : tidak ada kerugian sedikitpun, ada efek negatif meski tidak terlalu besar atau mungkin juga akan menimbulkan bahaya yang sangat besar. Dalam kenyataannya, kadang respon terhadap kejadian-kejadian yang muncul tidak cukup hanya dibatasi oleh *deadline* saja, melainkan juga dengan batas waktu paling awal suatu respon disajikan dan masih dianggap valid.

Dalam paper ini, awalnya dipaparkan konsep waktu-nyata konvensional dan kekurangannya, kemudian dilanjutkan dengan penjelasan konsep baru yang ditawarkan. Berkaitan dengan masalah penjadwalan dalam konsep yang baru, pada bagian berikutnya dipaparkan kelemahan dari algoritma EDF (*Earliest Deadline First*) sebagai algoritma penjadwalan yang dianggap paling optimal. Algoritma baru yang diusulkan kemudian dipaparkan, dilengkapi dengan beberapa contoh kasus.

2. Konsep Baru Sistem Waktu-Nyata

2.1 Konsep Konvensional

Salah satu definisi dari sistem waktu–nyata adalah suatu sistem dimana kebenaran dari suatu operasi

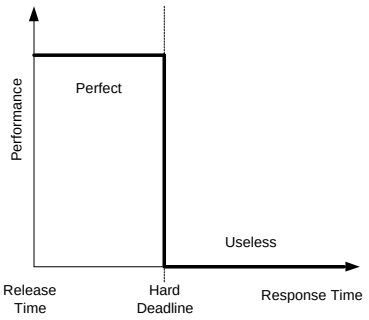
tidak hanya ditentukan oleh dari kebenaran logika dari hasil operasi saja, tapi juga tergantung pada waktu saat hasil itu disajikan[1]. Seringkali disebutkan bahwa tujuan utama komputasi waktu-nyata adalah mengeksekusi suatu proses kritis kemudian menyajikan hasilnya dengan dibatasi oleh *deadline* yang telah didefinisikan sebelumnya. Eksekusi akan dimulai segera setelah seluruh sumber daya (data) yang dibutuhkan tersedia.

Gambar 1 menunjukkan hubungan antara sistem Real-Time dengan lingkungan (*Data-Producer System* dan *Response-Consumer System*), dimana:

- *Release Time* (T_{Rel}) adalah saat tersedianya seluruh data yang dibutuhkan oleh *Real-Time System* untuk dapat beroperasi,
- *Communication Time* (T_{Com}) adalah rentang waktu yang dibutuhkan untuk mengirimkan data ke *Real-Time System*, atau mengirimkan respons ke *Response-Consumer System*,
- *Processing time* (T_{Proc}) adalah rentang waktu yang dibutuhkan oleh *Real-Time System* untuk beroperasi,
- *Response Time* (T_{Res}) adalah waktu penyajian respons ke *Response-Consumer System* (*Release Time* < *Response Time* ≤ *Deadline*),
- *Deadline* (T_{Dead}) adalah waktu batas akhir penyajian respons untuk dapat dianggap valid.

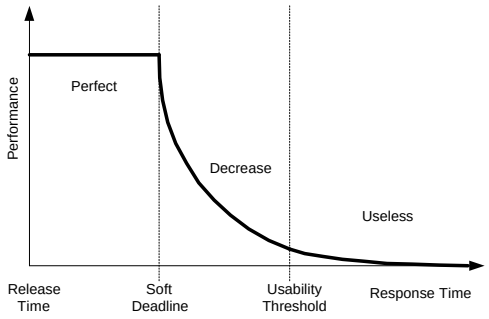
Grafik performansi (dengan *deadline* ditentukan oleh *user*) terhadap response time sistem waktu-nyata digambarkan dengan fungsi densitas, terlihat pada Gambar 2. Terlihat bahwa jika *response time* berada antara *release time* dan *deadline*, maka user akan menganggap '*perfect*', sebaliknya jika melebihi *deadline* akan dianggap '*useless*'.[8] Namun demikian perlu diingat bahwa *deadline* bukanlah selalu merupakan titik ekstrim pemisah '*perfect*' dengan '*useless*'. Tergantung dari tujuan sistem dan dampak yang mungkin terjadi jika terlanggar, *deadline* dapat dibedakan jadi dua macam, yakni : *hard deadline* dan *soft deadline*. *Deadline* jenis pertama persis seperti yang telah dikemukakan diatas. Sementara *deadline* jenis kedua lebih 'lunak', terlihat pada Gambar 3.

Disini terlihat bahwa meskipun telah melewati *deadline*, *response time* dari sistem masih dapat diterima meski tidak tergolong '*perfect*'. Tapi penerimaan ini bukan berarti tanpa batas toleransi. Ada suatu batas lain (juga dispesifikasikan sebelumnya) yang memisahkan mana *response time* yang masih dapat diterima dan mana yang sudah tergolong '*useless*'. Bentuk penurunan grafik (kecekungan atau kecuraman) setelah titik *soft deadline* ditentukan pula oleh user (secara subjektif).



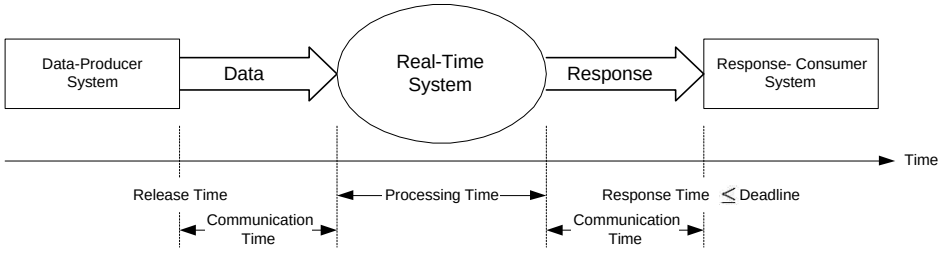
Gambar 2. Performansi dengan HardDeadline

Dengan adanya *deadline* ini, sistem waktu-nyata kadang lekat dengan anggapan sebagai 'usaha untuk mengeksekusi proses secepat mungkin'. Anggapan ini tidak sepenuhnya salah, karena kebanyakan sistem (perangkat keras, perangkat lunak dan algoritma) yang tersedia hampir selalu kewalahan dalam melayani permintaan eksekusi proses dengan adanya *deadline*, sehingga mempercepat proses eksekusi menjadi salah satu solusi.



Gambar 3. Performansi dengan SoftDeadline

Ternyata tidak semua aplikasi hanya dibatasi oleh *deadline* (batas waktu maksimal *response time*), ada juga aplikasi yang juga dibatasi oleh batas waktu terawal/tercepat respons dapat diberikan (selanjutnya akan disebut *liveline* [T_{Live}]). Sebagai contoh, sistem pengendali peluncur rudal darat ke udara (A) sebagai penghancur rudal musuh (B). Sistem pengendali ini harus memberikan perintah peluncuran kepada



Gambar 1. Model konvensional sistem waktu-nyata

peluncur rudal pada waktu yang tepat, begitu radar mendeteksi adanya rudal musuh yang masuk ke wilayah udara dalam jangkauannya. Perintah ini tidak hanya tidak boleh terlambat, tapi juga tidak boleh terlalu cepat, karena kalau salah satunya saja terjadi maka rudal (A) tidak akan dapat meledakkan rudal (B) diudara. Bahkan sangat mungkin akan ada dua kerugian besar yang diderita, satu karena meledaknya rudal (B) disasaran dan yang kedua adalah meledaknya rudal (A) disuatu tempat entah dimana. Contoh ini mengisyaratkan adanya kasus *hard liveline – hard deadline*.

2.2 Liveline

Dengan diperkenalkannya *liveline*, maka diagram hubungan sistem mengalami sedikit modifikasi sebagaimana terlihat pada Gambar 4. Apakah ini berarti konsep sistem *real-time* yang dikenal selama ini salah? Tidak! Bukannya salah, hanya saja belum bisa melingkupi semua kasus. Dengan model baru ini, model sistem *real time* konvensional merupakan salah satu kasus dimana *liveline* berhimpit dengan *release time.*

Kasus-kasus yang bisa digambarkan dengan model sistem *real-time* baru ini menjadi lebih komprehensif, yakni :

- *liveline = release time; hard deadline*
- *hard liveline; hard deadline*
- *soft liveline; hard deadline*
- *liveline = release time; soft deadline*
- *hard liveline; soft deadline*
- *soft liveline; soft deadline*
- *hard liveline;* tanpa *deadline*
- *soft liveline;* tanpa *deadline*

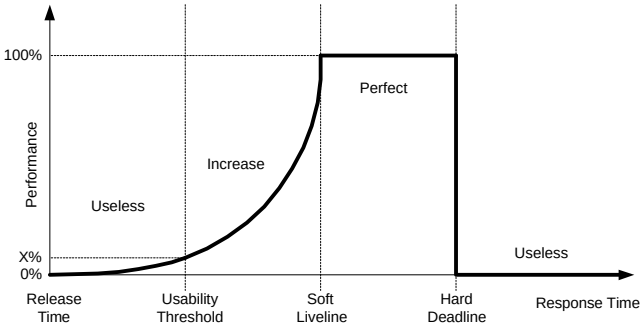
Grafik performansi untuk kasus *soft liveline; hard deadline* terlihat pada Gambar 5.

3. Strategi Waktu Pemrosesan Data dan Penyajian Respons

Diasumsikan prosesor dalam *Real-Time System* terdiri dari dua modul utama, yakni : *Calculator* dan *Actuator*. *Calculator* bertugas melakukan komputasi terhadap data sesuai dengan fungsi yang telah dispesifikasikan. *Actuator* akan menyajikan hasil perhitungan *Calculator* sebagai respon.

Dua strategi yang dapat diterapkan untuk menjamin $T_{Live} \leq T_{Res} \leq T_{Dead}$:

1. Mengatur saat dimulainya eksekusi proses, jika tidak ada jeda waktu antara proses komputasi dan aktuasi.
2. Mengatur (menunda) proses aktuasi, jika dimungkinkan adanya jeda waktu antara proses komputasi dan aktuasi. Konsekuensi adanya jeda waktu (disebut dengan *Buffer Time* [T_{Buf}]), maka harus disediakan buffer untuk menyimpan sementara hasil perhitungan *Calculator* untuk kemudian disajikan oleh *Actuator* pada saat yang tepat. Jika digunakan pendekatan ini, ada hal-hal yang perlu diperhatikan, antara lain :
   - Ukuran buffer yang diperlukan
   - Masa *Idle* Antara *Buffering* dan *Actuation* boleh jadi nol (tidak ada jeda waktu antara).
   - Algoritma penjadwalan harus dapat menentukan waktu mulainya proses kalkulasi dan waktu mulainya proses aktuasi.



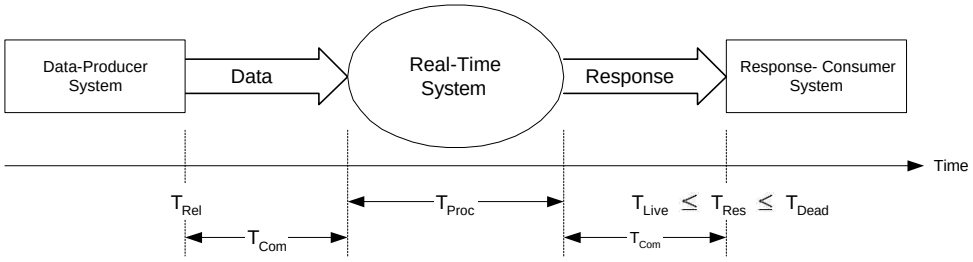
Gambar 5. Grafik performansi kasus SoftLiveline; HardDeadline

4. Algoritma Penjadwalan

4.1 Kekurangan EDF

4.1.1 Prioritas hanya ditentukan oleh *deadline*

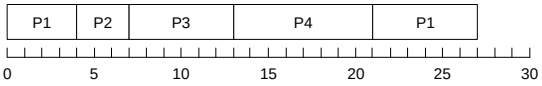
Meski dikatakan EDF (*Earliest Deadline First*) merupakan algoritma penjadwalan yang optimal untuk prosesor tunggal (jika sejumlah *task* tidak dapat dijadwalkan dengan EDF, maka tidak ada algoritma lain yang dapat menjadwalkannya)[8][5][2], tetapi EDF tidak bisa digunakan dalam kerangka model sistem *real-time* yang baru. Hal ini terutama karena EDF hanya memperhatikan *deadline* sebagai satu-satunya variabel penentu prioritas tiap proses. Sementara pada model sistem baru, *liveline* juga patut mendapat perhatian.



Gambar 4. Model baru sistem waktu-nyata

Tabel 1. Spesifikasi proses kasus 1

Proses	Waktu Kedatangan	Waktu Eksekusi	*Liveline* Absolut	*Deadline* Absolut
P1	0	10	0	30
P2	4	3	4	10
P3	5	6	16	25
P4	6	8	6	25

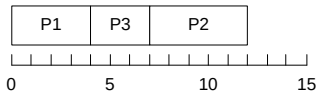


Gambar 6. Jadwal eksekusi proses pada kasus 1

Untuk proses kasus 1 sesuai Tabel 1, dengan EDF akan didapatkan skema penjadwalan pada Gambar 6. Semua proses memang selesai sebelum *deadline*, namun kalau dicermati lebih jauh ternyata P3 disajikan terlalu dini, yaitu pada unit waktu 13, sementara *liveline* absolutnya 16.

Tabel 2. Spesifikasi proses kasus 2

Proses	Waktu Kedatangan	Waktu Eksekusi	*Liveline* Absolut	*Deadline* Absolut
P1	0	4	0	6
P2	3	5	6	16
P3	4	3	8	14



Gambar 7. Jadwal eksekusi proses pada kasus 2

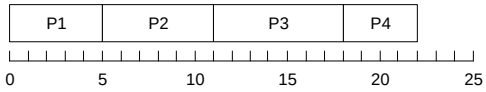
Untuk proses kasus 2 sesuai Tabel 2, dengan EDF akan didapat skema penjadwalan pada Gambar 7. Meskipun tampaknya tidak ada masalah berkenaan dengan deadline dan utilisasi prosesor pun sangat tinggi, namun lagi-lagi dijumpai masalah karena respon P3 disajikan terlalu dini.

4.1.2 Asumsi : *Deadline* selalu berjenis *Hard-Deadline*

Kekurangan lain dari algoritma EDF seperti telah dipaparkan di atas adalah belum terlihatnya penanganan penjadwalan untuk kumpulan proses dengan *soft-deadline*. Selama ini EDF selalu diaplikasikan dengan asumsi semua proses memiliki *hard-deadline*. Dengan mempertimbang-kan pula jenis *deadline* (*soft* atau *hard*), mungkin akan didapat jadwal yang lebih optimal, dibandingkan selalu mengasumsikan semua *deadline* berjenis *hard*.

Tabel 3. Spesifikasi proses kasus 3

Proses	T_{Rel}	T_{Exec}	*Deadline*	Jenis *Deadline*	*Deadline Usability Threshold*
P1	0	5	6	*Soft*	9
P2	3	6	10	*Soft*	26
P3	4	7	12	*Soft*	15
P4	14	4	19	*Soft*	21

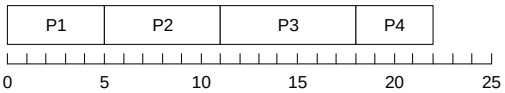


Gambar 8. Jadwal eksekusi proses pada kasus 3

Kumpulan proses untuk kasus 3 sesuai Tabel 3, jika dijadwalkan dengan EDF akan didapat jadwal seperti pada Gambar 8. P1 dapat dijadwalkan dan memenuhi *deadline*-nya. P2 juga terjadwalkan, meski tidak dapat memenuhi *deadline* namun masih dapat memenuhi *Usability Threshold*-nya. Ada dua proses yang tidak terjadwalkan karena tidak terpenuhi *Usability Threshold*-nya, yaitu P3 dan P4.

Tabel 4. Spesifikasi proses kasus 4

Proses	T_{Rel}	T_{Exec}	*Deadline*	Jenis *Deadline*	*Deadline Usability Threshold*
P1	0	5	6	*Soft*	20
P2	3	6	10	*Hard*	10
P3	4	7	12	*Soft*	17
P4	14	4	19	*Soft*	25



Gambar 9. Jadwal eksekusi proses pada kasus 4

Untuk proses kasus 4 sesuai Tabel 4, jadwal yang didapat jika dijadwalkan dengan EDF terlihat pada Gambar 9. P1 dapat dijadwalkan dan memenuhi *deadline*-nya. P2 tidak terjadwalkan karena tidak dapat memenuhi *hard-deadline*-nya. Ada dua proses yang juga tidak terjadwalkan, yaitu P3 dan P4, karena tidak terpenuhi *Usability Threshold*-nya,.

4.2 *Earliest Deadline Earliest Liveline First*

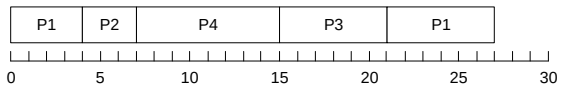
Hanya dengan dua contoh kasus di atas sudah cukup menunjukkan perlunya algoritma penjadwalan baru yang juga memperhatikan *liveline.* Algoritma baru ini dapat dikatakan merupakan modifikasi EDF.

Secara garis besar algoritma EDELF sama dengan EDF, yakni prosesor akan lebih dulu mengeksekusi proses yang memiliki prioritas lebih tinggi. Prioritas tetap dinamik sejalan dengan waktu dan kedatangan proses, perbedaannya hanyalah terletak pada aturan penentuan prioritas. Pada EDELF digunakan aturan sebagai berikut :

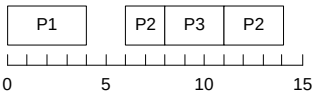
1. Jika $T_{Dead}(A) = T_{Dead}(B)$ maka:
   a. Jika TLive(A) = TLive(B) maka A dan B memiliki prioritas yang sama
   b. Jika TLive(A) < TLive(B) maka A memiliki prioritas yang lebih tinggi
2. Jika $T_{Dead}(A) < T_{Dead}(B)$ maka:
   a. Jika TLive(A) = TLive(B) maka A memiliki prioritas yang lebih tinggi
   b. Jika TLive(A) < TLive(B) maka A memiliki prioritas yang lebih tinggi

3. Jika TLive(A) > TLive(B) maka :
   a. Untuk t < T_{Live}(A): B memiliki prioritas yang lebih tinggi
   b. Untuk T_{Live}(A) ≤ t ≤ T_{Dead}(A): A memiliki prioritas yang lebih tinggi.

Dengan algoritma EDELF, maka dua contoh kasus di atas masing-masing akan dijadwalkan sebagaimana pada pada Gambar 10 untuk kasus 1 dan pada Gambar 11 untuk kasus 2. Pada kedua jadwal di atas tidak terlihat adanya pelanggaran baik terhadap *deadline* maupun *liveline*. Pada kasus 2 utilitas prosesor tidak setinggi saat digunakan algoritma EDF (prosesor *idle* pada unit waktu 4 sampai 6). Namun demikian hal ini masih dapat dikatakan sebagai *trade-off* yang layak untuk sebuah jaminan tidak terlanggarnya *liveline*.



Gambar 10. Jadwal kasus 1 dengan EDELF



Gambar 11. Jadwal kasus 2 dengan EDELF

4.2.1 EDF untuk proses dengan *soft-deadline*

Asumsi pukul rata bahwa semua *deadline* berjenis *hard*, bisa jadi digunakan untuk menyederhanakan masalah atau agar selalu didapat performansi maksimal (100%) pada eksekusi tiap proses yang terjadwalkan. Untuk bisa melakukan penjadwalan terhadap sekumpulan proses yang memiliki *soft-deadline*, diperlukan informasi tentang batas waktu yang memisahkan *response-time* yang masih dapat diterima (*usable*) dengan yang sudah tidak dapat diterima (*useless*). Pada grafik *performance vs response time* yang digambarkan diawal makalah ini, batas ini disebut sebagai *Usability Threshold* (T_{Use}).

Dengan turut mempertimbangkan *Usability Threshold* (T_{Use}), maka algoritma EDF dapat dimodifikasi sebagai berikut :

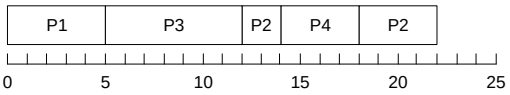
Setiap kali diperlukan penentuan prioritas dan penjadwalan (ada proses yang baru tiba atau berakhirnya pengerjaan suatu proses) :

1. Jadwalkan semua proses dalam buffer (semua proses yang telah tiba dan belum selesai) dengan EDF, prioritaskan proses dengan *hard-deadline* dibanding proses dengan *soft-deadline*
2. Jika seluruh proses yang memiliki *hard-deadline* terjadwalkan, namun ada proses dengan *soft-deadline* tidak terjadwalkan dengan EDF, maka untuk setiap proses yang belum terjadwalkan:
   - Cari P, dimana P adalah proses yang memiliki T_{Use} paling besar
   - Jadwalkan P selambat mungkin
3. Geser satu per satu proses yang baru saja dijadwalkan kekiri (kearah t = 0) hingga didapat ruang kosong yang paling minimal dengan urutan yang tidak berubah.

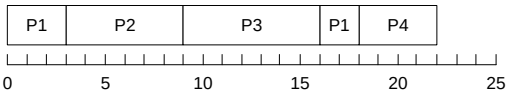
Langkah nomor 1 di atas bertujuan untuk menjamin terpenuhinya semua *hard-deadline* dan memaksimalkan performansi proses dengan *soft-deadline* bila ada yang dapat terjadwalkan dengan EDF. Langkah nomor 2 menentukan urutan eksekusi proses dengan *soft-deadline* yang belum terjadwalkan. Langkah terakhir, nomor 3, dilakukan untuk mengoptimalkan performansi semua proses dengan *soft-deadline* yang bisa dijadwalkan dengan algoritma ini.

Kasus 3 jika dijadwalkan dengan *modified* EDF akan menghasilkan jadwal seperti pada Gambar 12. Dari jadwal tersebut, memang hanya P1, P3 dan P4 yang dapat dipenuhi *deadline*-nya, sementara T_{Res} P2 melampaui *deadline*. Namun satu hal yang patut dipertimbangkan adalah tidak ada proses yang dijadwalkan melampaui T_{Use}-nya.

Kasus 4 jika dijadwalkan dengan *modified* EDF akan menghasilkan jadwal seperti pada Gambar 13. Dari jadwal diatas, seakan P1, P3 dan P4 'dikorbankan' agar P2 dapat terpenuhi *hard-deadline*-nya. Namun hasil akhir menunjukkan bahwa P1, P3 dan P4 tetap dapat memenuhi T_{Use}-nya.



Gambar 12. Jadwal kasus 3 dengan EDF modifikasi

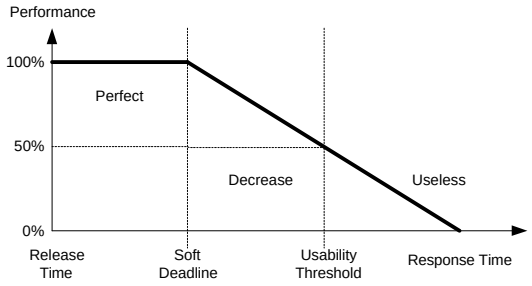


Gambar 13. Jadwal kasus 4 dengan EDF modifikasi

Jika fungsi garis atau kurva pada area *decrease* diketahui, maka dapat dihitung performansi eksekusi suatu proses yang telah dijadwalkan. Karena mungkin didapatkan jadwal dengan satu atau lebih proses yang T_{Res}-nya berada di area *decrease*, maka performansi (secara keseluruhan) mungkin tidak mencapai 100%.

Sebagai contoh, diketahui nilai performansi P (yang dapat diterima) dari proses-proses yang memiliki *soft-deadline* pada kasus 3 dan 4 sebagai fungsi dari T_{Res} sesuai Persamaan (1), dengan grafik sebagaimana yang diperlihatkan pada Gambar 14.

$$P(T_{Res}) = \begin{cases} 100\%; T_{Rel} \le T_{Res} \le T_{Dead} \\ 100\% - \dfrac{T_{Res} - T_{Dead}}{T_{Use} - T_{Dead}} \times 50\%; T_{Dead} < T_{Res} \le T_{Use} \end{cases} \quad (1)$$



Gambar 14. Grafik contoh performansi proses dengan SoftDeadline

Performansi masing-masing proses pada kasus 3 yang dijadwalkan dengan *modified* EDF di atas adalah sebagaimana ditunjukkan pada Tabel5. Sementara performansi masing-masing proses pada kasus 4 yang dijadwalkan dengan *modified* EDF ditunjukkan pada Tabel 6. Sebagai pembanding, performansi penjadwalan dengan EDF diperlihatkan pada Tabel 7.

Tabel 5. Performansi kasus 3 dengan *modified* EDF

Proses	T_{Res}	Performansi
P1	5	100%
P2	22	62,5%
P3	12	100%
P4	18	100%
Rata-rata		90,625%

Tabel 6. Performansi kasus 4 dengan *modified* EDF

Proses	T_{Res}	Performansi
P1	18	57,143%
P2	9	100%
P3	16	60%
P4	22	75%
Rata-rata		73,03575%

Tabel 7. Performansi dengan algoritma EDF

Proses	T_{Res}	Performansi
P1	5	100%
P2	11	96,875%
P3	18	0%
P4	22	0%
Rata-rata		49.21875%

4.2.2 EDELF untuk proses dengan *soft-deadline* dan *soft-liveline*

Algoritma penjadwalan EDELF yang disajikan di atas juga masih menggunakan asumsi semua *deadline* dan *livelife* berjenis *hard*. Dengan menambahkan informasi jenis *Liveline*, nilai *Liveline Usability Threshold*, dan performansi sebagai fungsi dari T_{Res} di area *Decrease* pada spesifikasi masing-masing task, mungkin dapat disusun algoritma penjadwalan *modified* EDELF (seperti halnya *modified* EDF). Eksekusi jadwal yang dihasilkan oleh algoritma ini diharapkan dapat memberikan performansi rata-rata yang lebih tinggi dibanding algoritma EDELF yang tidak dimodifikasi.

Secara garis besar, algoritma *modified* EDELF adalah sebagai berikut :

Setiap kali diperlukan penentuan prioritas dan penjadwalan (ada proses yang baru tiba atau berakhirnya pengerjaan suatu proses) :

1. Jadwalkan semua proses dalam buffer (semua proses yang telah tiba dan belum selesai) dengan EDELF, prioritaskan proses dengan *hard-deadline* dibanding proses dengan *soft-deadline*, demikian juga proses dengan *hard-liveline* dibanding proses dengan *soft-liveline*
2. Jika seluruh proses yang memiliki *hard-deadline* dan *hard-liveline* terjadwalkan, namun ada proses dengan *soft-deadline* dan/atau *soft-liveline* tidak terjadwalkan dengan EDELF, maka untuk setiap proses yang belum terjadwalkan:
   a. Untuk proses-proses yang memiliki *soft-deadline*, maka :
      - Cari P, dimana P adalah proses yang memiliki $T_{UseDead}$ paling besar
      - Jadwalkan P selambat mungkin
   b. Untuk proses-proses yang memiliki soft-liveline, maka :
      - Cari P, dimana P adalah proses yang memiliki TUseLive paling kecil
      - Jadwalkan P secepat mungkin
   c. Untuk proses-proses yang memiliki soft-deadline dan soft-liveline, maka :
      - Gunakan cara a atau b di atas dengan mempertimbangkan performansi maksimal yang dapat dicapai
3. Geser satu per satu proses yang baru saja dijadwalkan hingga didapat ruang kosong yang paling minimal dengan urutan tidak berubah :
   a. Untuk proses-proses pada poin 2.a. : geser ke kiri (ke arah t = 0)
   b. Untuk proses-proses pada poin 2.b. : geser ke kanan (ke arah t = ∞)

5. Kesimpulan

Dari paparan dan contoh kasus di atas, dapat disimpulkan bahwa Konsep sistem waktu-nyata konvensional tidak dapat memodelkan semua kasus sistem waktu-nyata yang mungkin ada. Dengan melakukan modifikasi terhadap konsep konvensional (menambahkan faktor *liveline*), maka konsep sistem waktu-nyata menjadi semakin umum. Kasus-kasus yang dapat dimodelkan dengan model konvensional menjadi bagian kasus yang dapat dimodelkan dengan konsep sistem waktu-nyata yang baru. Modifikasi konsep memungkinkan pemilihan strategi pemrosesan data (*calculate*) dan penyajian respon (*actuate*) sesuai dengan kebutuhan sistem. Dengan konsep sistem yang baru, algoritma penjadwalan proses yang dianggap matangpun menjadi tidak valid lagi, sehingga dibutuhkan modifikasi. Algoritma EDELF yang dibangun sebagai modifikasi algoritma EDF terbukti

(dengan contoh kasus) memiliki performansi yang lebih baik dibanding EDF pada kasus-kasus yang tidak dapat dimodelkan dengan akurat menggunakan konsep konvensional.

Paper ini menyajikan hasil penelitian tahap awal. Sebaiknya dikembangkan suatu penelitian yang merinci hubungan konsep baru tersebut dengan kebutuhan (*requirements*) sistem waktu-nyata, yang berkaitan dengan aspek *behavioral*, *temporal* dan *cost*; dan hubungan dengan batasan (*constraints*) sistem waktu-nyata, misal: tentang *task model* dan *timing constraints* [4]. Perlu pula dilakukan modifikasi berbagai algoritma alokasi sumber daya dan penjadwalan proses lain yang telah dianggap matang pada konsep konvensional, agar sesuai dengan konsep yang baru (dan kemungkinan besar bisa bersifat lebih umum).

Daftar Pustaka

[1] Bennet, S. 1994. *Real-Time Computer Control: An Introduction*, Prentice Hall.

[2] Bonuccelli, M. A. and C. M. Clo. *EDD Algorithm Performance Guarantee for Periodic Hard-Real-Time Scheduling in Distributed Systems*, Universita di Pisa

[3] Brandt, S. A., *Performance Analysis of Dynamic Soft Real-Time Systems*, University of California.

[4] Ekelin, C. and J. Jonsson. Oktober 1999. *Real-Time System Constraints: Where do they come from and where do they go?*. Virginia: Proceedings of the International Workshop on Real-Time Constraints. Alexandria.

[5] Goosens, J., R. Devillers and S. Funk. *Tie-breaking for EDF on Multiprocessor Platforms*. Universite Libre de Bruxelles and University of North Carolina.

[6] Ingram, D. *Soft Real-Time Scheduling for General Purpose Client-Server System*. University of Cambridge

[7] Joseph, M. (editor). 1996. *Real-Time Systems: Specification, Verification and Analysis*, Prentice Hall.

[8] Krishna, C.M, Kang G. S., 1997, *Real-Time Systems*, McGraw Hill.

[9] Shin, Youngsoo. Daehong Kim, and Kiyoung Choi, *Schedulability-Driven Performance Analysis of Multiple Mode Embedded Real-Time Systems.* University of Tokyo and Seoul National University.