DOKUMEN KAJIAN TEKNIS "RESTART" UNTUK MENYELESAIKAN PERMASALAHAN DITINJAU DARI SISI APLIKASI

Periode Penilaian Semester I Tahun 2019

Oleh:

Budi Satrio
NIP 198503082010121009
Sekretariat Jenderal

Tim Penilai Instansi Pusat

Jabatan Fungsional Pranata Komputer

Kementerian Keuangan R.I.

Jakarta, April 2019

Dokumen Kajian Teknis Restart untuk Menyelesaikan Permasalahan Ditinjau dari Sisi Aplikasi dan Keterhubungannya dengan Sistem Lain
Tanggal : 22 April 2019
Disusun oleh :

NAMA/ NIP	JABATAN	TANDATANGAN
Budi Satrio/ 19850308 201012 1 009	Pranata Komputer Bidang Pengembangan Sistem Informasi	

Penanggung Jawab:

Kepala Bidang Pengembangan
Sistem Informasi,

Yusuf Nurrohman
NIP 19750802 199502 1 001

Diterbitkan oleh:
Pusat Sistem Informasi dan Teknologi Keuangan
Jalan Lapangan Banteng Timur Nomor 2–4, Jakarta 10710
Situs Web http://pusintek.kemenkeu.go.id
Telephone +62.21.3449230 external 4100
Faksimile +62.21.3519655
E-mail servicedesk@kemenkeu.go.id

LEMBAR PENGESAHAN

Dengan menandatangani lembar ini semua pihak menyatakan telah membaca, memahami, dan menyetujui isi kajian teknis Restart untuk Menyelesaikan Permasalahan Ditinjau dari Sisi Aplikasi dan Keterhubungannya dengan Sistem Lain.

Diperiksa Oleh:

Nama/ NIP	Jabatan	Tandatangan	Tanggal
Yusuf Nurrohman/ 19750802 199502 1 001	Kepala Bidang Pengembangan Sistem Informasi		23/4/19

Disetujui Oleh:

Nama/ NIP	Jabatan	Tandatangan	Tanggal
Herry Siswanto/ 19710322 199603 1 002	Kepala Pusat Sistem Informasi dan Teknologi Keuangan		23/4/19

DAFTAR ISI

DAFTAR GAMBAR

BAB I. PENDAHULUAN

Dokumen kajian teknis ini dibuat dengan tujuan memberikan gambaran umum secara keseluruhan mengenai permasalahan aplikasi yang dihadapi baik di *Data Center* (DC) Kementerian Keuangan maupun di *Data Recovery Center* Kementerian Keuangan. Kemudian, dokumen ini juga menyediakan langkah solusi mengenai restart sebagai jawaban dari beberapa kasus yang dihadapi baik di DC maupun DRC.

A. Latar Belakang

Sebagai unit Teknologi Informasi dan Komunikasi (TIK) Pusat yang fungsi utamanya memberikan layanan TIK di lingkungan Kementerian Keuangan, Pusat Sistem Informasi dan Teknologi Keuangan (Pusintek) menyadari pentingnya keandalan dalam menjaga ketersediaan layanan TIK. Hal ini tercermin salah satunya yaitu keikutsertaan Pusintek dalam standarisasi internasional TIK, antara lain ISO 20000 tentang standar Pengelolaan Layanan TI dan ISO 27001 tentang Sistem Manajemen Keamanan Informasi.

Selain meraih sertifikasi standar layanan, Pusintek juga memberikan kemampuan terbaiknya untuk menghadapi bermacam-macam gangguan yang muncul baik di DC maupun DRC Kementerian Keuangan. Salah satu contoh

gangguan adalah seringnya permasalahan yang muncul mengenai lambatnya aplikasi dan salah satu solusi tercepatnya adalah melakukan *restart*.

Restart yang dilakukan oleh Tim Teknis Pusintek bentuknya bisa beraneka ragam, mulai dari *restart server* fisik / *Virtual Machine* (VM), *restart service* aplikasi / basis data, *restart* perangkat jaringan, ataupun *restart* perangkat *security*. Dalam melakukan *restart* ini, seringkali tidak disertai dengan landasan berpikir mengapa *restart* dilakukan dan tidak ada analisis mendetail mengenai penyebab permasalahan, sehingga tim teknis kurang melakukan eksplorasi untuk menyelesaikan permasalahan selain dengan *restart*.

Karena *restart* terbukti cukup ampuh mengatasi permasalahan-permasalahan yang terjadi, perlu dilakukan analisis lebih jauh mengenai konsep *restart* itu sendiri, analisis permasalahan, dan decision making untuk melakukan *restart*.

B. Ruang Lingkup

Dokumen kajian ini akan fokus kepada permasalahan-permasalahan ditinjau dari kacamata aplikasi. Maka dari itu, dokumen kajian ini melingkupi beberapa tiga hal utama, yaitu apa saja contoh dari konsep *restart* di dunia nyata, bagaimana cara mengidentifikasi atau menganalisis suatu permasalahan dari sisi aplikasi, dan dasar apa yang digunakan sebagai *decision making* untuk melakukan tindakan *restart*.

Dokumen kajian ini tidak membahas mengenai cara-cara menyelesaikan permasalahan aplikasi dan cara-cara untuk melakukan *restart* dengan baik dan benar. Untuk menyelesaikan permasalahan aplikasi, dibutuhkan dokumen tersendiri yang terperinci, lengkap, dan detail, karena jumlah dari cara-cara untuk menyelesaikan permasalahan aplikasi akan bertambah secara eksponensial sejalan dengan kenaikan tingkat kompleksitas aplikasi. Sedangkan variasi dari cara-cara untuk melakukan restart jumlahnya juga akan sangat banyak sesuai dengan teknologi, jenis perangkat, dan vendor yang menyediakan teknologi / perangkat tersebut.

Dengan dijelaskannya ruang lingkup kajian ini, diharapkan dokumen ini akan lebih fokus menggali konsep *restart* pada aplikasi, analisis permasalahan, serta *decision making* untuk melakukan *restart*.

BAB II. LANDASAN TEORI

Pada beberapa poin dibawah, dijelaskan konsep-konsep dasar dan teori-teori yang meliputi pelaksanaan *restart* sebagai solusi dari permasalahan teknis.

A. *Restart*

Sesuai dengan penjelasan Kamus Besar Bahasa Indonesia (KBBI), *start* artinya adalah memulai (melakukan sesuatu)[1], sehingga, walaupun tidak tercantum dalam KBBI, *restart* dapat dijelaskan sebagai proses untuk mengulangi *start*. Di dalam dunia IT, konsep *restart* disebut juga dengan *reboot*, dan terbagi menjadi dua, yaitu *cold reboot* dan *warm reboot*. *Cold reboot* adalah proses mengulangi kembali untuk menyalakan komputer atau sistem dengan adanya pemutusan sambungan listrik, sehingga perangkat keras akan mengalami proses mati sebentar tanpa dialiri listrik[2]. Sedangkan *warm reboot* adalah proses mengulangi kembali untuk menyalakan komputer atau sistem tanpa adanya pemutusan sambungan listrik[3]. Kedua proses ini mempunyai kesamaan yaitu bagaimana caranya untuk mengembalikan *state* komputer menjadi seperti semula pada awal sebelum komputer itu digunakan.

Dalam dunia nyata, *restart* ini perlakuannya akan berbeda pada setiap jenis perangkat, ataupun dari jenis perangkat yang sama namun berbeda merek.

Sebagai contoh, pada sistem operasi Windows, yang dimaksud dengan *restart* adalah *cold reboot*, namun perangkat keras *motherboard* nya masih teraliri listrik. Sedangkan pada sistem operasi Linux, *restart* adalah *warm reboot* tanpa melakukan *restart* pada perangkat keras.

Penyebab dilakukannya *restart* dapat terbagi menjadi tiga, yaitu kesengajaan (*deliberate*), kegagalan listrik (*power failure*), dan tanpa alasan jelas (*random*). Pada poin pertama, kesengajaan, maksudnya adalah pengguna dengan sengaja melakukan *restart*. Hal ini dapat terjadi karena pengguna menemui permasalahan sehingga solusinya adalah *restart*, atau memang karena setelah melakukan install perangkat lunak sehingga langkah selanjutnya adalah *restart*. Pada poin kedua, kegagalan listrik, maksudnya adalah setiap komputer atau sistem, jika mengalami kegagalan listrik diharuskan untuk melakukan *cold reboot* pada saat listrik kondisinya sudah normal kembali. Pada poin terakhir, *random*, maksudnya adalah kondisi *restart* yang dilakukan karena tidak sengaja. Hal ini bisa terjadi karena fitur sistem operasi yang melakukan *restart* sendiri karena terjadi *memory error*, atau *segmentation fault*, maupun *system crash*.

B. *Recursive Restartability*

Recursive Restartability merupakan suatu konsep yang diperkenalkan oleh George Andea dan Armando Fox untuk melakukan *restart* dengan lancar dan mulus pada level manapun tanpa harus mengganggu keseluruhan jalannya sistem[4].

C. *Recoverable Programming Models*

Recoverable Programming Models merupakan suatu konsep yang diperkenalkan oleh Dejan Milojicic dan rekan, untuk menganalisis bagaimana sebaiknya aplikasi menghadapi gangguan-gangguan pada perangkat keras, sehingga menyebabkan perlunya *restart*[5].

D. *Decision Making*

Decision Making adalah proses berpikir secara logis untuk menentukan keputusan dari beberapa pilihan yang tersedia[6]. Menurut Irham Fahmi, *decision making* adalah proses penelusuran masalah yang berawal dari latar belakang masalah, identifikasi masalah hingga kepada terbentuknya kesimpulan atau rekomendasi dalam pengambilan keputusan[7]. Teknik untuk memutuskan sesuatu ini terbagi menjadi dua, yaitu *group* dan *individual decision making*.

Pada *group decision making*, keputusan diambil secara bersama-sama, memperhatikan setiap anggota dalam kelompok. Pada bagian ini, keputusan dapat diambil melalui beberapa cara, diantaranya yaitu konsensus dan *voting*. Konsensus adalah pengambilan keputusan dengan cara permufakatan bersama (mengenai pendapat, pendirian, dan sebagainya) yang dicapai melalui kebulatan suara[8]. Konsensus biasanya dipilih untuk menghindari kelompok pendukung dan penentang, sehingga diharapkan keputusan yang dipilih akan disetujui oleh semua pihak, apabila ada satu suara yang tidak setuju maka tidak dapat dilakukan pengambilan keputusan. Sedangkan *voting* adalah proses pengambilan keputusan

dengan memperhatikan pihak mayoritas dan minoritas. Sebagai contoh, jika pihak yang menyetujui suatu pilihan keputusan berjumlah lebih dari 50 persen, maka keputusan dapat diambil sesuai dengan pilihan tersebut.

Pada *individual decision making*, pengambilan keputusan dapat dilakukan dengan banyak cara, seperti *cost benefit analysis* ataupun *decision support systems*. *Cost benefit analysis* adalah metodologi yang digunakan untuk mengukur apakah suatu keputusan menghasilkan keuntungan yang maksimal atau tidak[9]. Pada metodologi ini, pembuat keputusan harus menimbang untung rugi dari setiap pilihan keputusan, setelah itu dilakukan analisis pilihan mana yang akan memberikan keuntungan maksimal. *Decision support system* merupakan merupakan suatu sistem interaktif berbasis komputer, yang membantu pengambil keputusan melalui penggunaan data dan model-model keputusan untuk memecahkan masalah-masalah yang sifatnya semi terstruktur dan tidak terstruktur untuk mempertinggi efektivitas pengambilan keputusan[10]. Pada metode ini, pengambilan keputusan dibantu oleh sistem informasi yang interaktif yang menampilkan parameter-parameter yang dibutuhkan untuk pengambilan keputusan.

E. Framework

Framework atau kerangka kerja adalah sejumlah pemikiran, konsep, ide atau asumsi yang digunakan untuk mengorganisasikan proses pemikiran tentang

sesuatu atau situasi[11]. Untuk menyelesaikan suatu permasalahan, *framework* dibutuhkan sebagai kerangka berpikir agar proses penyelesaian menjadi lebih logis dan terstruktur.

1. Framework for Analysing Criticality

Dalam papernya, Bo Edvardsson dan Inger Roos, menjelaskan bahwa diperlukan sebuah *framework* untuk menganalisis kritikalitas dari sebuah insiden[12]. Beberapa parameter dalam *framework* tersebut adalah *Time*, *Memory*, dan *History*. *Time* artinya perlunya untuk mencatat waktu kapan permasalahan terjadi, kapan permasalahan mulai dianalisis, dan kapan waktu eksekusi penyelesaian permasalahannya. *Memory* adalah pentingnya mencatat opini-opini pengguna dan pengeksekusi sebagai bahan untuk analisis. Terakhir, *history*, adalah perlunya untuk menyimpan segala macam data, baik waktu, maupun opini-opini, sehingga dapat kita lihat rekam sejarahnya. Dengan adanya *history* tersebut, penganalisis dapat menggunakannya sebagai bahan analisis untuk menyelesaikan permasalahan yang mungkin terjadi secara berulang.

BAB III. KONDISI SAAT INI

Pada bagian ini, dicontohkan beberapa kasus yang solusinya adalah dengan melakukan *restart*. Kasus ini terjadi di dunia nyata, baik di luar Kementerian Keuangan, maupun di dalam Pusintek sendiri, dan digunakan sebagai contoh untuk memperlihatkan bahwa *restart* bukan merupakan sesuatu yang harus dihindari.

A. Kasus di luar Kementerian Keuangan

Beberapa contoh kasus kritikal yang melibatkan restart diantaranya yaitu, FBI, Google Home, Nasa Pathfinder, dan Rudal Pertahanan Patriot.

Contoh kasus pertama yaitu Biro Investigasi Amerika Serikat atau *Federal Bureau Investigation* (FBI) memerintahkan kepada setiap rumah tangga untuk melakukan *restart router modem* rumah tangga mereka[13]. Hal ini dimaksudkan untuk menghapus *malware,* program kecil yang merugikan pengguna, yang menginfeksi setiap *router* di setiap rumah. Kejadian ini bisa terjadi karena *router* pada setiap rumah spesifikasinya minim sehingga rentan disusupi oleh *malware*. Dari kasus ini terlihat bahwa *restart* akan mengembalikan kondisi / *state router* kembali seperti semula sebelum terkena infeksi *malware*.

Contoh kasus kedua yaitu Google menganjurkan kepada setiap pemilik Google Home dan Chromecast untuk melakukan *restart* apabila terjadi *glitch* atau

kesalahan-kesalahan kecil yang tidak berdampak signifikan[14]. Hal ini dimaksudkan untuk menghapus kesalahan dalam perangkat lunak yang terus menumpuk selama keadaan menyala. Saat perangkat Google tersebut di *restart*, ia akan kembali pada keadaan semula sebelum terjadi *glitch*.

Contoh kasus ketiga yaitu mesin penjelajah planet Mars, *Mars Pathfinder,* yang dibuat oleh Badan Antariksa Amerika Serikat, NASA, memiliki fasilitas *restart* perangkat keras dan perangkat lunaknya apabila terjadi kesalahan. Fitur ini ternyata berguna dan dijalankan pada saat *Mars Pathfinder* gagal mengeksekusi program secara tepat waktu pada saat pendaratan di Mars[15]. Hal ini makin memperjelas bahwa *restart* merupakan suatu fitur yang harus disediakan dalam penggunaan perangkat keras, maupun perangkat lunak.

Contoh kasus terakhir yaitu sistem peluncuran peluru kendali Patriot oleh pasukan Amerika Serikat dalam Perang Teluk ternyata harus di *restart* setiap 8 jam sekali karena sesuatu *bug*, atau kesalahan pemrograman[16]. Untuk memperbaiki sistemnya demi menghindari *restart*, dibutuhkan waktu yang lama dan teknisi langka dan berpengalaman. Saat sistem tersebut diperbaiki, ternyata menyebabkan 28 prajurit meninggal dan 98 luka-luka. Hal ini menunjukkan bahwa ketika waktu dan resource untuk memperbaiki suatu kesalahan tidak sebanding dengan melakukan *restart*, maka *restart* bisa dilakukan secara berkala.

Gambar 1. Contoh sistem peluru kendali Patriot

B. Kasus Internal Kementerian Keuangan

Dari hasil perekaman tiket yang tercatat pada aplikasi ITSM (*Information Technology Service Management*) Pusintek, didapatkan 761 satu kasus yang dekripsi dan *resolve* nya mengandung kata "*restart*". Hal ini mencakup sekitar 4% dari total tiket yang direkam dari tahun 2017 sampai September 2018. Jika dikelompokkan, terdapat 321 tiket tentang *restart* server, atau sekitar 42%, 198 tiket mengenai *restart* aplikasi (sekitar 24%), dan sisanya yaitu 117 *restart* jaringan, 103 *restart* infrastruktur, dan 22 *restart* database[17]. Untuk melihat lebih dalam mengenai kasus *restart* dari sisi aplikasi, terdapat beberapa contoh kasus

yaitu lambatnya akses aplikasi Nadine (Naskah Dinas Elektronik) dan E-Performance.

Pada kasus aplikasi Nadine, saat pengguna login ke dalam aplikasi, yang tampil hanya halaman loading yang tidak kunjung selesai. Jika ditelisik dari sisi aplikasi, penggunaan sumber daya *Central Processing Unit* (CPU) sangat tinggi, dan jika dilihat dari sisi database, juga mengalami hal serupa. Pada kasus seperti ini yang dijadikan sebagai solusi adalah dengan cara melakukan *restart service Internet Information Services* (IIS) sebagai web server dan melakukan *restart service database*.

Hal yang sama juga terjadi pada kasus aplikasi E-Performance, namun bedanya adalah issue lambatnya aplikasi Nadine muncul pada waktu random, sedangkan issue lambatnya aplikasi E-Performance muncul pada saat akhir masa penilaian perilaku dan Indeks Kinerja Utama. Solusi dari permasalahan ini juga mirip, yaitu dengan melakukan *restart service web server* dan *restart service* basis data. Namun yang membuat issue aplikasi ini berbeda adalah adanya kesempatan untuk memperbaiki struktur *database* dengan menambahkan *index* dan menghilangkan *query join* yang berat.

C. Alasan Restart

Beberapa alasan dilakukannya restart dapat dikelompokkan menjadi dua hal utama, yaitu *refreshing system* dan *solving heisenbug*.

Alasan pertama mengenai *refreshing system*, maksudnya adalah *restart* dapat mengembalikan *resource* yang *stuck* dan membersihkan *state* yang *corrupt*, Sehingga, sistem akan berjalan dengan kondisi semula sebelum *error* terjadi, walaupun terdapat kemungkinan *loss of data integrity* yang berarti kehilangan integritas data setelah restart dilakukan.

Heisenbug adalah kesalahan pada aplikasi, terutama pada level pemrograman, yang sulit untuk direproduksi atau dilakukan kembali, sehingga tidak ada solusi lain selain *restart*[18]. Maka dari itu, maksud dari alasan kedua adalah *restart* dapat mengatasi *heisenbug* tersebut.

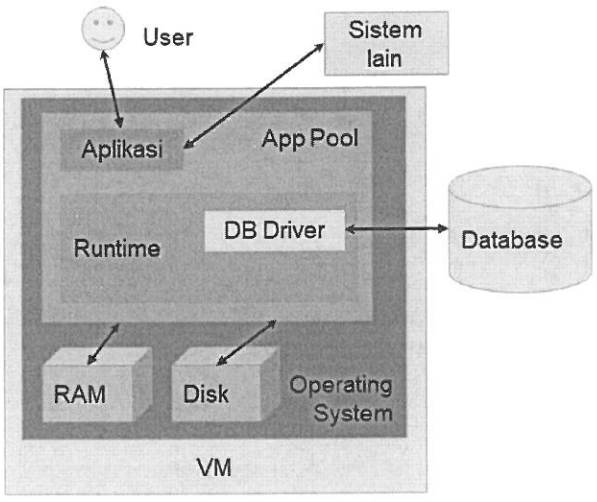
Selain dari dua alasan utama tersebut, *restart* merupakan solusi yang cepat dalam keadaan khusus. Maksud dari hal ini adalah ketika penyelesaian suatu masalah membutuhkan waktu yang lama, atau membutuhkan *resource* yang besar, maka cara paling efektif adalah dengan melakukan *restart*. Sebagai contoh yaitu saat aplikasi E-Performance lambat, untuk memperbaiki hal tersebut membutuhkan waktu sekitar dua minggu, sementara aplikasi tidak boleh ada downtime karena esok hari adalah masa terakhir penilaian. Pada kasus seperti ini, cara paling cepat adalah dengan melakukan *restart*, walaupun secara sumber daya sudah tersedia untuk memperbaiki permasalahan lambat tersebut tetapi waktunya tidak mencukupi.

BAB IV. PEMBAHASAN DAN ANALISIS

Bagian ini akan fokus pada analisis aplikasi secara detail termasuk gambaran umumnya, potensi permasalahan yang muncul pada aplikasi, serta pengelompokan permasalahan tersebut.

A. Gambaran Umum Aplikasi

Aplikasi berjalan di atas bahasa pemrograman yang juga berjalan di atas *runtime* sebuah *Operating System*. Hal ini merupakan sebagian kecil dari sudut pandang aplikasi dari sisi Sistem Operasi. Maka dari itu, untuk lebih mengetahui bagaimana aplikasi berjalan, perlu kita lihat detail aplikasi sesuai dengan gambar 2.



Gambar 2. Detail proses di dalam sebuah aplikasi

Pada gambar 2, dapat kita lihat bahwa Pengguna atau *User* mengakses aplikasi yang berada dalam *application pool* yang di dalamnya berisi *runtime* untuk menjalankan proses bisnisnya. *Application pool* adalah sebuah kelompok proses yang memiliki konfigurasi tertentu yang digunakan oleh aplikasi yang berjalan di dalam *application pool* tersebut[19]. Fitur ini berguna untuk mengisolasi sebuah aplikasi atau mengelompokkan beberapa aplikasi yang mirip, agar tidak mengganggu kinerja dari aplikasi lain pada *server* yang sama. *Runtime* adalah kumpulan fungsi-fungsi dalam bentuk objek yang siap untuk dieksekusi oleh bahasa pemrograman[20] Sebagai contoh, *runtime* dari bahasa Java memiliki fungsi untuk menuliskan ke dalam disk, mengambil input dari keyboard, ataupun menampilkan sesuatu pada layar monitor. Bagaimana kita menggunakan fungsi-fungsi tersebut diatur oleh proses logis dalam aplikasi yang kita buat.

Application pool dan *runtime* ini berjalan di atas *Operating System*. Ini berarti fungsi-fungsi pada *runtime* akan ditranslasikan menjadi bahasa mesin oleh Sistem Operasi. Sesuai dengan definisi dari Stallings, Sistem Operasi adalah kumpulan perangkat lunak *low level* yang bertugas menjadi jembatan antara aplikasi / program / perangkat lunak *high level* dengan perangkat keras[21]. Untuk dapat menjalankan suatu proses dalam perangkat keras, dibutuhkan bahasa mesin untuk berkomunikasi dengan perangkat tersebut. Adalah tugas Sistem Operasi untuk mentranslasikan proses logis dari aplikasi ke dalam perangkat keras.

Sistem Operasi ini berjalan pada sebuah server, atau kumpulan perangkat keras yang terdiri dari *Central Processing Unit* atau otak dari komputer, *Random Access Memory* sebagai tempat meletakkan temporary state saat program berjalan, dan *Storage* sebagai tempat meletakkan file-file yang diperlukan oleh program atau pengguna. Seiring dengan perkembangan zaman, server ini dapat digantikan oleh proses virtualisasi dari kumpulan perangkat keras. Hal ini biasa kita sebut dengan *Virtual Machine* (VM) dari layanan *Cloud*. Dengan adanya *cloud* ini, pemilik infrastruktur dapat membuat server secara *real-time*, sesuai dengan kemampuan infrastrukturnya, tanpa terbebani oleh perangkat-perangkat keras yang harus ada di dalam sebuah server.

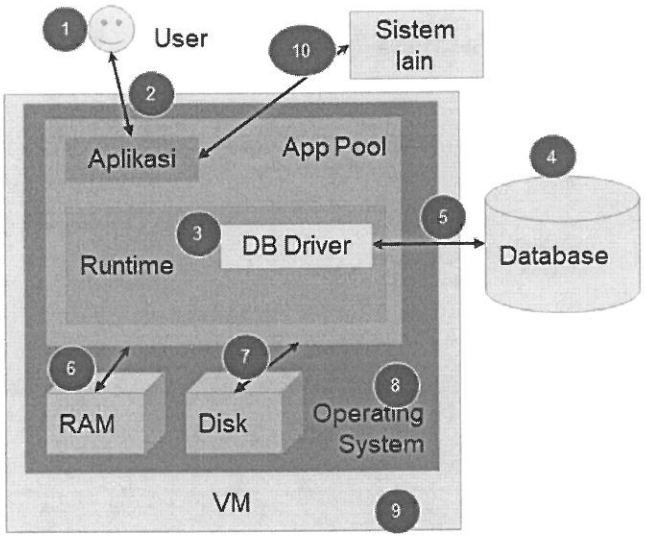
Selain itu, aplikasi juga harus dilihat dari sisi luarnya, yaitu hubungannya dengan sistem lain dan basis data nya. Hubungan dengan sistem lain maksudnya adalah apakah dalam menjalankan proses bisnisnya, aplikasi membutuhkan data atau *response* dari sistem lain. Jika ya, maka sistem lain tersebut akan berpengaruh besar terhadap jalannya aplikasi. Sebagai contoh, jika aplikasi kita berhubungan dengan hasil dari sistem Human Resource Information System (HRIS) untuk mendapatkan nama pengguna, maka pada saat HRIS tersebut sedang mengalami gangguan, maka aplikasi kita akan terkena dampaknya.

Tidak hanya sistem lain, aplikasi juga berhubungan dengan basis data untuk menyimpan data-data yang akan digunakan dalam proses bisnis sebuah aplikasi. Sesuai dengan penjelasan Beynon-Davies bahwa basis data adalah tempat

penyimpanan data aplikasi, baik berbentuk *structured model* ataupun *non structured model*, termasuk sistem manajemennya, agar dapat digunakan oleh aplikasi[22].

B. Potensi Permasalahan

Dari penjelasan di poin sebelumnya, kita dapat melihat bahwa aplikasi saling terhubung dengan bagian-bagian lain sesuai dengan gambar 2. Dari banyaknya hubungan tersebut, dapat kita identifikasi mana saja potensi permasalahan yang muncul dari sebuah aplikasi. Untuk itu perlu kita lihat pada gambar 3 di bawah ini.



Gambar 3. Potensi permasalahan dalam sebuah aplikasi

Dari gambar 3, dapat diidentifikasi 10 potensi permasalahan aplikasi, berikut adalah penjelasannya.

1. **Permasalahan pada pengguna**

Untuk mengakses sebuah aplikasi, pengguna dapat menggunakan komputer maupun *smartphone*, kemudian jika aplikasi yang diakses berbentuk web, maka pengguna akan menggunakan *browser*. Dari hal ini dapat diidentifikasi permasalahan yang muncul saat komputer atau *smartphone* atau *browser* mengalami kendala. Umumnya, pengguna akan menyalahkan aplikasi saat terjadi kesalahan, namun perlu dipastikan bahwa kondisi komputer / *smartphone* / *browser* pengguna sudah terbebas dari kesalahan.

Contoh yang paling umum terjadi adalah ketika pengguna mengalami kendala saat menggunakan aplikasi tertentu, namun setelah diteliti permasalahan tersebut disebabkan karena pengguna menggunakan browser yang telah *out of date*, sehingga aplikasi tidak dapat berjalan dengan sempurna pada browser seperti itu.

2. **Permasalahan antara pengguna dengan aplikasi**

Pada kasus ini, hubungan antara komputer / *smartphone* / *browser* dengan aplikasi dapat menjadi sumber permasalahan, biasanya terkait dengan jaringan. Dalam hal ini perlu dipisahkan antara jaringan yang menghubungkan pengguna dengan data center tempat aplikasi tersebut berjalan, serta jaringan yang berada di dalam data center itu sendiri.

Karena bisa jadi gangguan terjadi di dalam konfigurasi jaringan di data center seperti adanya *Anti DDoS*, atau *firewall* maupun *hardening*.

Anti DDoS merupakan sistem atau perangkat yang mampu mendeteksi serangan *Denial of Service*, sehingga dapat dengan cepat memblok koneksi internet dari pengakses[23]. *Firewall* adalah sistem keamanan jaringan yang mengontrol lalu lintas data keluar dan masuk berdasarkan rules[24]. *Hardening* merupakan proses pengamanan sistem dengan cara mengurangi hal-hal yang memicu terbukanya celah keamanan[25]. Contohnya yaitu melakukan *uninstall* perangkat lunak yang tidak digunakan, menghapus *user login* yang tidak perlu, dan *disable service* yang tidak digunakan.

Sedangkan gangguan yang kerap terjadi pada jaringan antara pengguna dengan data center biasanya dipengaruhi oleh infrastruktur tempat pengguna melakukan akses ke aplikasi. Sebagai contoh, pengguna yang berada di pusat kota akan lebih cepat mendapatkan akses dibanding pengguna yang berada di daerah terpencil, hal ini disebabkan karena jangkauan sinyal atau internet yang digunakan pada pusat kota lebih baik daripada di daerah terpencil.

Ketika pengguna mendapatkan kendala saat mengakses sebuah aplikasi, maka perlu diteliti apakah kondisi jaringan yang menghubungkan pengguna dengan aplikasi sedang mengalami gangguan.

3. Permasalahan pada aplikasi

Bagian ini merupakan yang paling penting karena aplikasi adalah tujuan utama pengguna untuk melakukan proses bisnisnya. Terdapat banyak isu pada aplikasi, namun pada umumnya berkutat pada *performance, compatibility,* dan *configuration*[26].

Performance maksudnya adalah kinerja dari aplikasi, seberapa responsif aplikasi jika diakses dalam waktu yang lama, atau dalam melayani jumlah pengguna yang sangat besar. Hal ini akan berhubungan dengan *logic programming*, kemampuan CPU, RAM, dan *storage*. Seorang *developer* aplikasi seharusnya sudah mengantisipasi permasalahan yang muncul terkait dengan kinerja aplikasi dengan membuat *logic programming* dengan baik dan benar. Sementara untuk CPU, RAM, dan *storage* akan bergantung pada kemampuan infrastruktur tempat aplikasi berjalan.

Compatibility adalah kesesuaian antara versi atau jenis aplikasi dengan environment tempat aplikasi itu berjalan, seperti *runtime, driver,* atau Sistem Operasinya. Ketidaksesuaian yang muncul dapat

menyebabkan aplikasi tidak dapat berjalan dengan sempurna. Sebagai contoh, ketika aplikasi yang dibuat untuk .NET *Framework* 4.7 dijalankan pada *server* dengan .NET *Framework* 4.5, maka terdapat kemungkinan fungsi-fungsi yang tidak berjalan dengan semestinya.

Configuration adalah daftar settingan atau konfigurasi yang digunakan aplikasi saat berjalan di dalam infrastruktur. Konfigurasi ini dapat berupa *path* dari basis data atau *driver* yang digunakan, bisa juga berupa settingan berapa jumlah maksimal objek yang dapat ditangani oleh *pool* dari web server. Konfigurasi ini penting karena untuk dapat memaksimalkan kerja aplikasi, konfigurasinya harus sesuai dan optimal. Sebagai contoh, jika aplikasi berjalan pada web server apache dengan konfigurasi standar dari saat diinstall, maka otomatis aplikasi yang berjalan di atasnya akan mengalami gangguan ketika *request* yang datang secara bersamaan berjumlah lebih dari yang tertulis dalam konfigurasinya.

4. **Permasalahan pada basis data**

Pada kasus ini, permasalahan aplikasi dapat disebabkan oleh basis data yang digunakannya. Karena basis data pada dasarnya berjalan pada sebuah server / VM dengan konfigurasi tertentu, maka permasalahan basis data dapat terbagi menjadi permasalahan perangkat keras dan perangkat lunak.

Perangkat keras basis data dapat berupa CPU, RAM, maupun *storage* khusus. Sebagai contoh, *cluster* Oracle menggunakan storage khusus untuk tempat *Extract, Transform*, dan *Load* nya. Ketika perangkat keras mengalami masalah, maka akan berdampak pada aplikasi.

Perangkat lunak basis data ini maksudnya adalah *service* yang berjalan sebagai basis data, seperti mysqld pada *database* MySQL, atau SQLServer *instance*. Ketika *service* ini bermasalah disebabkan oleh desain database yang tidak optimal, maka aplikasi akan terpengaruh dampaknya. Sebagai contoh, ketika terdapat proses bisnis untuk melakukan join dua tabel dengan setiap tabelnya memiliki lebih dari sepuluh juta *record*, maka otomatis proses ini akan memakan waktu yang akan mempengaruhi performa aplikasi.

5. Permasalahan pada aplikasi dengan basis data

Pada kasus ini, hubungan antara aplikasi dengan basis data dapat menjadi sumber permasalahan dan umumnya terkait dengan jaringan. Ketika aplikasi lancar dan basis data dalam kondisi baik, namun konektivitas antara keduanya terdapat gangguan, maka dapat dipastikan aplikasi akan mengalami gangguan. Sebagai contoh, saat koneksi antara aplikasi dan database terganggu akibat adanya *network loop*, yaitu kondisi dimana setiap switch membroadcast ke *switch* yang lain mengakibatkan

loop paket data[27], maka aplikasi akan menampilkan *error* bahwa basis data tidak dapat diakses.

6. Permasalahan pada aplikasi dengan *Random Access Memory*

Random Access Memory (RAM) merupakan tempat untuk meletakkan data dan *machine code* secara acak, untuk digunakan oleh sistem operasi saat sedang berjalan[28]. RAM ini akan terisi dengan data dan kode mesin saat sistem operasi digunakan, dan apabila terjadi *power off*, isi RAM akan terhapus. Sistem operasi yang baik akan selalu menjaga isi dan kondisi RAM agar tidak mudah penuh dan tidak terjadi *corrupt*. Namun, dari sisi aplikasi, karena aplikasi dibuat oleh pengembang, maka terdapat kemungkinan-kemungkinan terjadinya kasus RAM yang penuh ataupun corrupt, apabila pengembang tidak memiliki kesadaran terkait RAM yang digunakan ketika aplikasi berjalan.

Sebagai contoh, pada saat aplikasi melakukan *join* data untuk menghasilkan suatu daftar, jika *join* dilakukan pada aplikasi, maka *object* nya akan banyak terbentuk di RAM, sehingga dapat memicu RAM yang penuh. Tetapi jika join dilakukan di *database*, lalu diambil oleh aplikasi, maka RAM yang digunakan akan lebih kecil daripada metode sebelumnya.

7. **Permasalahan pada aplikasi dengan *Storage***

Storage adalah tempat penyimpanan yang bersifat permanen, dalam artian ketika terjadi *power off*, maka data akan dapat diakses kembali oleh Sistem Operasi. Karena storage ini digunakan oleh Sistem Operasi, makan akan banyak sistem-sistem kecil yang menggunakannya sehingga mengakibatkan prosess *Input Output* (IO), atau baca tulis dari dan ke dalam storage, akan saling bergantung antara aplikasi dengan sistem operasinya. Ketika terjadi gangguan pada storage, baik itu berupa corrupt atau IO yang tinggi, maka aplikasi akan terkena dampaknya.

Sebagai contoh, ketika aplikasi sedang diakses dan pada saat yang bersamaan terjadi proses *scan antivirus*, maka aplikasi akan mengalami kelambatan karena proses IO digunakan lebih banyak oleh *scan antivirus*.

8. **Permasalahan pada Sistem Operasi**

Sistem Operasi terdiri dari berbagai macam jenis, mulai dari *Windows*, *Linux/Unix* dan variannya, serta Apple. Setiap jenis OS tersebut memiliki fitur dan fungsi yang pada dasarnya adalah sama, yaitu mengatur hubungan antara CPU, RAM, *storage*, dan aplikasi sehingga dapat digunakan oleh pengguna. Karena OS ini beraneka ragam dan beraneka versi, maka harus dipastikan pada jenis dan versi yang mana aplikasi dapat berjalan optimal. Jika aplikasi berjalan pada sistem operasi yang tidak optimal, maka dapat dipastikan gangguan akan lebih sering muncul.

Sebagai contoh, apabila aplikasi E-Performance berjalan pada *Windows* XP, maka aplikasi tersebut tidak akan berjalan dengan optimal, atau bahkan tidak berjalan sama sekali.

9. Permasalahan pada *Server / Virtual Machine*

Karena Sistem Operasi berjalan diatas perangkat keras berupa *server* atau perangkat lunak berupa *virtual machine*, maka setiap gangguan yang terjadi di dalamnya akan berimbas ke aplikasi. Gangguan ini dapat berupa kesalahan pada perangkat kerasnya, seperti *storage* yang *corrupt* atau *motherboard* yang rusak, atau berupa kesalahan pada perangkat lunaknya, seperti versi *Vsphere* yang sudah *outdated* karena belum menggunakan *patch* terbaru.

Sebagai contoh, aplikasi portal Kemenkeu mengalami gangguan, namun setelah dicek gangguan itu ternyata disebabkan oleh perangkat lunak administrasi *virtual machine* (*Vsphere)* yang digunakan sudah outdated, sehingga tidak dapat berfungsi optimal.

10. Permasalahan pada aplikasi dengan sistem lain

Pada kasus ini, perlu dianalisis dengan sistem mana saja aplikasi terhubung. Sistem lain tersebut bisa berupa *Single Sign On, get & receive data* dari API, serta process *Synchronous / Asynchronous* dengan sistem lain.

Single Sign On (SSO) merupakan sebuah sistem yang bertugas sebagai pintu *login* dari aplikasi-aplikasi yang berbeda[29]. Jika pengguna telah masuk ke dalam aplikasi satu melalui SSO, maka ia akan dapat masuk ke sistem lain tanpa harus melakukan *login* lagi. Ketika terjadi permasalahan pada SSO, maka pengguna akan mengalami gangguan saat melakukan login.

Application Programming Interface (API) adalah tata cara aplikasi untuk saling berkomunikasi antara satu dengan yang lain, termasuk didalamnya adalah protokol komunikasi, format yang digunakan, dan bagaimana interaksinya[30]. Sebagai contoh, aplikasi E-Performance mendapatkan data pegawai dengan cara mengakses API dari aplikasi HRIS dengan menggunakan protokol HTTP dan format data menggunakan *Representational State Transfer* (REST). Ketika terjadi permasalahan pada API, seperti kondisi API yang dituju sedang lambat / error, maka aplikasi tidak akan dapat berjalan dengan sempurna.

Synchronous/Asynchronous adalah cara aplikasi untuk menyelesaikan sebuah logis dari proses bisnis[31]. Synchronous adalah proses eksekusi dalam aplikasi yang berjalan secara seri, dimana eksekusi proses ke-2 akan dijalankan saat aplikasi telah selesai memproses eksekusi ke-1. Asynchronous adalah proses eksekusi secara paralel, dimana eksekusi proses ke-2 dapat dijalankan walaupun proses eksekusi ke-1

belum selesai. Ketika aplikasi menggunakan proses synchronous, terdapat kemungkinan pengguna tidak akan mendapatkan hasil secara cepat, karena eksekusi proses menunggu sampai eksekusi proses sebelumnya selesai.

Sebagai contoh, aplikasi penerimaan CPNS terhubung dengan sistem BKN untuk mendapatkan data peserta CPNS. Ketika sistem BKN mengalami gangguan, maka aplikasi dapat terkena dampaknya, yaitu muncul error bahwa data peserta CPNS tidak dapat ditemukan.

C. Pengelompokkan Permasalahan Aplikasi

Dari 10 potensi permasalahan yang telah diidentifikasi pada subbab 4.2, terdapat beberapa kesamaan sumber permasalahan yang muncul. Beberapa kesamaan ini dapat dikelompokkan menjadi lima jenis sumber permasalahan, yaitu *Server*, Jaringan, Hubungan dengan Sistem lain, Basis Data, dan Aplikasi. Dengan adanya pengelompokan ini, setiap gangguan yang muncul dapat dianalisis secara mendalam pada setiap sumber.

1. Analisis dari sisi Server

Pada poin ini, sumber permasalahan dilihat dari sisi *Server*, baik itu berbentuk fisik maupun VM, perangkat kerasnya berupa RAM, CPU, dan *Storage*, serta perangkat lunaknya yaitu Sistem Operasi (OS) atau *Vsphere*.

Pertama, analisis permasalahan di sisi *Server* dapat dilihat dari kondisi perangkat kerasnya. Sebagai contoh yaitu *storage* yang *corrupt*, *Network Attached Storage* (NAS) yang lambat, ataupun CPU dan RAM yang kecil. Solusi dari permasalahan ini terbagi menjadi dua, yaitu memperbaiki kondisi sekarang, atau melakukan *restart* perangkat. Seperti pada contoh kasus diatas, jika NAS lambat maka dapat diperbaiki dengan cara mengganti storage tersebut atau dengan melakukan restart NAS.

Kedua, analisis sumber permasalahan dilihat dari kondisi perangkat lunaknya. Sebagai contoh yaitu OS yang digunakan belum menggunakan patch terbaru, atau OS yang digunakan adalah versi lama yang sudah tidak disupport oleh pengembang OS, bisa juga terdapat permasalahan virus atau malware pada OS. Solusi dari permasalahan ini yaitu dengan memperbaikinya atau melakukan *restart* OS.

2. Analisis dari sisi Jaringan

Pada bagian ini, permasalahan aplikasi dapat dilihat dari sisi Jaringan, sehingga perlu dilakukan analisis terdapat perangkat jaringan apa saja yang terhubung ke aplikasi serta bagaimana konfigurasinya. Beberapa contoh perangkat yang mempunyai efek terhadap lancarnya komunikasi aplikasi melalui Jaringan yaitu *Firewall, Packet Scanning*, dan *Web Application Filter*. Sedangkan beberapa contoh konfigurasi Jaringan yaitu bagaimana jalur komunikasi aplikasi, apakah langsung ke dalam server,

ataukah melewati filter terlebih dahulu, kemudian apakah terdapat port yang tidak dibuka karena alasan keamanan.

Solusi untuk permasalahan ini terbagi menjadi dua, yaitu dengan cara memperbaiki perangkat dan konfigurasi, atau dengan melakukan restart perangkat dan mengembalikan konfigurasi semula dimana aplikasi sudah pernah berjalan.

3. **Analisis dari sisi Hubungan dengan Sistem Lain**

Karena aplikasi yang besar sudah pasti akan berhubungan dengan sistem lain, maka pada poin ini, permasalahan dalam aplikasi dapat dilihat dari seberapa besar gangguan sistem lain yang berpengaruh terhadap aplikasi kita. Sebagian contoh hubungan dengan sistem lain adalah penggunaan *Single Sign On, get & receive data* dari API, serta process *Synchronous / Asynchronous* dengan sistem lain.

Solusi untuk permasalahan ini dapat dilihat dari sumber permasalahannya. Jika perbaikan sistem lain atau sistem kita sendir bisa dilakukan secara cepat maka sebaiknya diperbaiki. Sebagai contoh, saat API HRIS mengalami error yang menyebabkan gangguan pada aplikasi E-Performance, maka kita dapat memperbaiki API HRIS tersebut. Jika perbaikan sistem lain tidak bisa dilakukan, maka langkah selanjutnya adalah dengan melakukan *restart* sistem lain, atau *restart* aplikasi kita

sendiri saat terjadi gangguan. Namun perlu diingat jika gangguan sistem lain tersebut terjadi karena kesalahan pemrograman, maka solusi utamanya adalah dengan melakukan perbaikan dari sisi kita untuk menghindari error pada saat melakukan komunikasi dengan sistem lain.

4. Analisis dari sisi Basis Data

Pada bagian ini, permasalahan aplikasi dapat dilihat dari sisi Basis Data nya baik itu dari perangkat keras berupa CPU, RAM, dan *Storage* dari *server* Basis Data, atau dari perangkat lunaknya seperti versi Basis Data, fitur-fitur yang dinyalakan, dan kondisi Basis Datanya yang meliputi indexing, jumlah *record*, atau jumlah tabel.

Untuk perangkat keras, analisisnya hampir sama dengan analisis pada poin 4.3.1 karena berhubungan dengan *Server*. Sebagai contoh, storage yang digunakan harus cukup untuk mengatasi lonjakan jumlah data, karena log yang timbul dari Basis Data akan berjumlah sangat besar saat aplikasi digunakan mendekati deadline.

Untuk perangkat lunak, analisis harus dilakukan secara mendalam terkait sistem, struktur, maupun kondisi Basis Datanya. Pertama, sistem dalam hal ini yaitu jenis Basis Data apa yang digunakan, apakah MySQL, SQL Server, Oracle, atau PostgreSQL. Kemudian versi berapakah yang digunakan, mengingat versi lama biasanya terdapat bug ataupun celah

keamanan. Kedua, struktur basis data adalah bagaimana arsitektur dari Basis Data aplikasi, apakah menggunakan Relasional atau Non Relasional. Kemudian apakah desain tabelnya sudah menggunakan normalisasi, index, maupun view atau stored procedure. Terakhir, kondisi Basis Data adalah bagaimana performa dari Basis Data saat aplikasi digunakan, seperti berapa jumlah record data dan apakah terdapat data yang bermasalah.

Jika gangguan terjadi pada perangkat keras maka solusinya sama seperti dengan solusi *Server*, yaitu dengan memperbaiki atau mengganti perangkat keras, atau melakukan *restart* perangkat kerasnya. Jika gangguan terjadi pada perangkat lunaknya, solusinya adalah dengan memperbaikinya atau melakukan *restart service* Basis Data.

5. Analisis dari sisi Aplikasi

Mengacu pada poin 4.2.3, bahwa analisis aplikasi harus dilihat dari *performance, compatibility*, dan *configuration* nya. Pertama, *performance*, jika aplikasi tidak berkinerja secara maksimal, maka perlu di cek apakah *logic programmingnya* sudah baik, apakah sudah menerapkan *cache*, atau apakah aplikasi menggunakan resource yang besar. Kedua, pengecekan kesesuaian antara aplikasi dengan *environment* yang digunakan untuk menjalankan aplikasi. Apakah *runtime* yang digunakan sudah sesuai dengan *runtime* yang diinstall pada server, apakah OS yang digunakan support dengan fitur-fitur aplikasi, atau apakah *library* yang digunakan

sudah sesuai versi nya dengan *library* yang digunakan pada saat pengembangan aplikasi. Terakhir, pengecekan konfigurasi aplikasi, apakah konfigurasi *web server* aplikasi masih standar atau sudah *tuned*, apakah konfigurasi aplikasi masih dalam mode *debug / production*, atau apakah konfigurasi driver yang digunakan sudah maksimal.

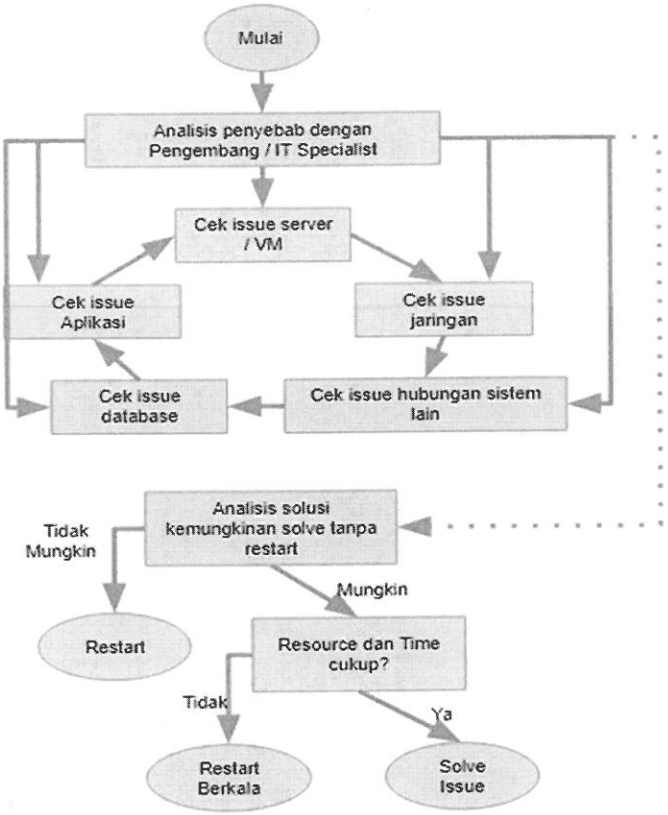
Jika terjadi permasalahan dari sisi aplikasi, maka yang perlu dilakukan adalah melihat konfigurasi terlebih dahulu, kemudian melihat sisi *web server / environment* aplikasi, dan terakhir adalah dengan melihat *core aplikasi*, apakah terdapat sumber data atau waktu yang cukup untuk memperbaikinya. Jika perbaikan tidak mungkin untuk dilakukan dilihat dari *time* dan *resource*, maka langkah terakhir adalah dengan melakukan *restart* aplikasi, termasuk di dalamnya adalah *restart web server* atau *restart* OS.

BAB V. RESTART DECISION FRAMEWORK

Berdasarkan analisis permasalahan aplikasi yang telah dibahas pada poin 4, terdapat dua solusi utama gangguan aplikasi, yaitu dengan memperbaikinya atau melakukan *restart*. Karena kajian ini berfokus pada *restart*, maka langkah yang dapat dilakukan untuk melakukan *restart* secara efektif sekaligus meminimalisir efek negatif dari *restart* adalah dengan membuat suatu *framework* untuk mengambil keputusan mengenai restart.

Restart Decision Framework (RDF) adalah sebuah *framework* yang dibangun untuk mengatasi permasalahan yang solusinya adalah restart. *Framework* ini akan mencakup analisis permasalahan aplikasi dari 5 sisi, analisis solusinya, serta langkah-langkah pengambilan keputusannya. Karena *framework* ini akan unik untuk setiap permasalahan aplikasi, dan ada kemungkinan permasalahan aplikasi akan berulang, maka diperlukan sebuah *form* yang sifatnya adalah untuk mencatat informasi dari banyak analisis yang telah dilakukan. *Form* ini berfungsi sebagai *knowledge* yang dapat digunakan oleh setiap pengguna jika terjadi permasalahan yang pernah terjadi. Untuk lebih memahami, diperlukan contoh kasus yang bisa diselesaikan dengan *framework* ini.

Berikut adalah gambar detail RDF:



Gambar 4. Restart Decision Framework

Dari gambar 4, dapat dilihat bahwa RDF terbagi menjadi tiga alur besar, yaitu Analisis Penyebab, Analisis Solusi, dan Eksekusinya apakah Restart, Restart Berkala, atau memperbaikinya.

A. Analisis Penyebab

Pada alur ini, pengelola aplikasi diharapkan untuk berdiskusi dengan pengembang aplikasi atau IT *Specialist* untuk mengetahui secara lebih detail mengenai permasalahan dalam aplikasi. Dalam melakukan analisis ini, pengelola aplikasi dan pengembang / IT *Specialist* wajib untuk menelusuri ke lima sisi potensi

permasalahan dalam aplikasi, yaitu *Server*/VM, Jaringan, Hubungan dengan Sistem Lain, Basis Data, dan Aplikasi.

Proses analisis penyebab ini diharapkan bisa berjalan secara cepat (tidak lebih dari dua hari) dengan tujuan untuk memberikan solusi terbaik dalam waktu yang singkat. Salah satu faktor pendukung analisis yang cepat adalah kesigapan dari tim *Server*, Jaringan, Basis Data dan Aplikasi yang selalu bersiaga di ruangan yang sama. Jika keempat tim tersebut berada dalam satu ruangan, sebagai contoh ruang *Operating Command* (OC), maka proses analisis ini akan berjalan cepat dikarenakan mudahnya memperoleh informasi dalam satu pintu.

Output dari alur ini adalah hasil identifikasi dan analisis secara detail mengenai permasalahan aplikasi dengan mempertimbangkan ke lima sisi potensi permasalahan dalam aplikasi.

B. Analisis Solusi

Pada alur ini, pengelola aplikasi harus berdiskusi dengan pengembang / IT *Specialist*, serta manajemen untuk memberikan solusi permasalahan aplikasi dengan mempertimbangkan *resource* dan *time* yang tersedia. Terdapat beberapa kemungkinan dari analisis solusi ini, diantaranya yaitu aplikasi tidak bisa di *solve* sama sekali, aplikasi dapat di *solve* namun sumber daya dan waktu tidak mencukupi, atau aplikasi dapat diperbaiki dengan memperhitungkan kemampuan sumber daya dan waktu.

Kemungkinan pertama yaitu permasalahan aplikasi ini tidak bisa diperbaiki sama sekali. Solusi jika terjadi permasalahan seperti ini adalah dengan melakukan *restart* secepatnya dengan memperhatikan kemungkinan bahwa isu ini bisa kembali terjadi. Untuk memilih *restart* mana yang akan dilakukan, perlu koordinasi dengan pengembang dan tim *Server*, Jaringan, Basis Data, dan Aplikasi. Sehingga *restart* akan lebih spesifik dan efisien. Sebagai contoh permasalahan aplikasi yang disebabkan oleh lambatnya proses di web server akan lebih baik jika yang di *restart* adalah *service* dari aplikasi itu sendiri, bukan restart *operating systemnya.*

Kemungkinan kedua yaitu permasalahan aplikasi dapat diperbaiki, namun sumber daya dan waktu tidak mencukupi. Solusi untuk permasalahan ini adalah dengan melakukan *restart* secara berkala ketika permasalahan terjadi. Penentuan kapan dilakukan *restart* ini akan berbeda antara aplikasi satu dengan aplikasi lainnya, oleh karena itu dibutuhkan koordinasi dengan pengembang / IT *Specialist* untuk mengetahui pada jam / hari / minggu / bulan apa *restart* sebaiknya dilakukan. Pada umumnya, variasi dari *restart* berkala ini akan terbagi menjadi dua, yaitu konstan / *flat* dan dinamik. *Flat* dalam hal ini adalah *restart* dilakukan secara berkala dengan jangka waktu yang sama, sebagai contoh *restart* basis data HRIS setiap senin, rabu, dan jumat jam 11 malam. Sedangkan dinamik adalah proses *restart* yang reaktif, atau mengikuti kondisi-kondisi tertentu. Sebagai contoh, *restart service* E-Performance dilakukan setiap dua jam dimulai dari tiga

hari menjelang deadline pengisian penilaian perilaku, selain hari itu tidak ada restart.

Kemungkinan terakhir yaitu permasalahan aplikasi dapat diperbaiki dengan memperhitungkan sumber daya dan waktu. Pada langkah ini tidak dilakukan *restart*, namun yang dilakukan adalah memperbaiki permasalahan aplikasi. Perbaikan ini dapat dilakukan pada salah satu atau lebih dari lima sisi potensi permasalahan aplikasi. Sebagai contoh perbaikan modul monitoring agar tidak lambat membutuhkan kerja sama antara tim basis data untuk membuat *index*, serta tim aplikasi untuk mengganti *algoritma* monitoringnya menjadi lebih efisien.

C. Form Analisis

Dokumen ini pada dasarnya adalah pencatatan dari analisis penyebab dan analisis solusi yang telah dilakukan pada alur sebelumnya. *Form* analisis ini akan dikumpulkan sehingga dapat menjadi basis pengetahuan / *knowledge* jika dikemudian hari terjadi permasalahan aplikasi yang sama / mirip dengan permasalahan yang pernah ditangani.

Bentuk form ini sangat sederhana namun mencakup analisis penyebab dari lima sisi, analisis solusi, serta *contact person* dari pengelola dan pengembang / IT *Specialist*. Fungsi dari dituliskannya kontak tersebut adalah untuk memudahkan

koordinasi apabila terjadi permasalahan yang mirip dan sudah pernah dianalisis sebelumnya.

Untuk lebih mengetahui secara detail mengenai form ini, dapat kita lihat pada gambar 5.

Form Analisis Restart Decision Framework

*lingkari seperlunya

1. Nama Aplikasi			
2. Permasalahan			
3. Nama & Kontak Pengembang			
4. Nama & Kontak IT Specialist			
5. Analisis Penyebab	a. Server		
	b. Jaringan		
	c. Hubungan dengan Sistem Lain		
	d. Bssis Data		
	e. Aplikasi		
6. Analisis Solusi	a. Dapat solve tanpa restart	YA* (go to 6.b)	TIDAK* (go to 6.d: restart)
	b. Apakah resource dan time memadai	YA* (go to 6.d: solve)	TIDAK* (go to 6.c)
	c. Apakah restart dibutuhkan secara flat atau dinamik	FLAT (go to 6.d: restart berkala) setiap ..... jam/hari/minggu....	DINAMIK (go to 6.d: restart berkala) setiap .... (contoh mendekati deadline)
	d. Keputusan dan Penjelasan	Restart / Restart Berkala / Solve*	
	e. Langkah yang dilakukan	1..... 2..... 3..... 4....	

Tanggal Permasalahan diketahui

DD/MM/YYYY

Tanggal Analisis RDF selesai

DD/MM/YYYY

Gambar 5. Form Analisis Restart Decision Framework

Pada gambar 5, kita dapat melihat bahwa informasi aplikasi, permasalahan, pengembang dan IT *Specialist* diisi pada nomor 1 sampai 4.

Sedangkan nomor 5 a sampai e adalah analisis penyebab permasalahan aplikasi dilihat dari kelima sisi potensi permasalahan. Pada nomor 6 a sampai e, kita dapat melihat alur pengambilan keputusan apakah solusi dari permasalahan aplikasi tersebut adalah *restart*, *restart* berkala, atau memperbaikinya.

Dengan adanya form ini, setiap permasalahan yang muncul akan dicatat analisis penyebab dan analisis solusinya sehingga akan berguna ke depan apabila terjadi permasalahan yang sama / mirip dengan permasalahan yang sebelumnya pernah terjadi. Data analisis ini akan dikumpulkan menjadi *knowledge* yang dapat dianalisis juga dengan teknik *mining* dan *big data* untuk mendapatkan *trend* / prediksi mengenai permasalahan yang mungkin akan muncul ke depan.

D. Contoh penggunaan Form RDF

Untuk lebih mengetahui bagaimana Form RDF ini membantu mempercepat keputusan terkait permasalahan aplikasi, berikut adalah contoh penggunaannya pada kasus lambatnya akses E-Performance. Terdapat 2 langkah utama yang dilakukan, yaitu Analisis Penyebab dan Analisis Solusi.

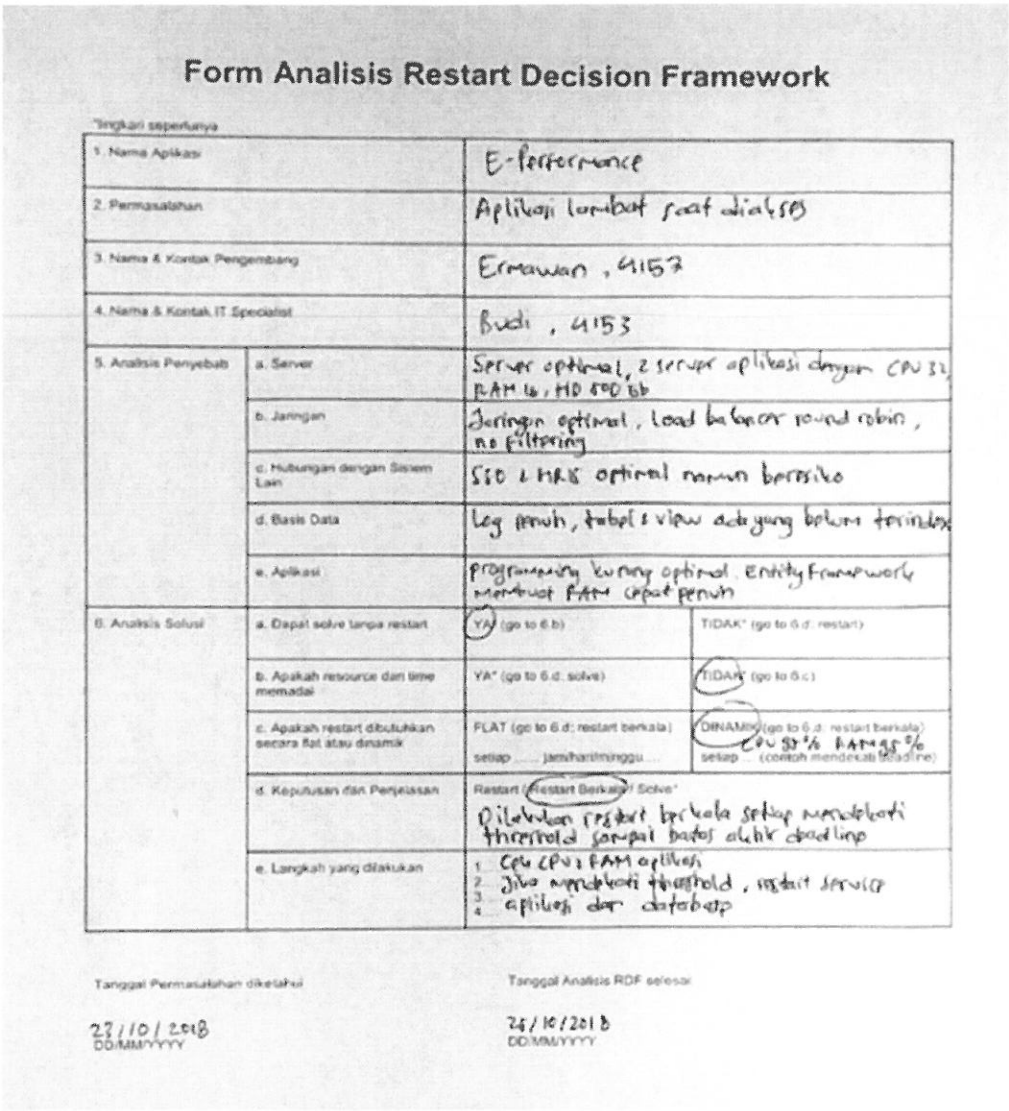
Pada tahap Analisis Penyebab, pengelola aplikasi berkoordinasi dengan pengembang aplikasi E-Performance atau IT *Specialist*. Kemudian, satu per satu melihat lima sisi potensi penyebab permasalahan. Dari sisi *Server* diketahui bahwa *server* aplikasi telah menggunakan spesifikasi maksimal yaitu dua server aplikasi dengan CPU 32 core dan *Harddisk* 500 Gb serta RAM 16 Gb. Dari sisi Jaringan

diketahui bahwa konfigurasi sudah baik dengan menggunakan *load balancer* F5 metode *round robin* serta tidak ada *Application Filtering* antar server aplikasi dengan database. Dari sisi Hubungan dengan Sistem lain, diketahui bahwa aplikasi rawan tidak bisa diakses ketika SSO dan HRIS mengalami masalah. Namun diketahui juga bahwa kondisi SSO sekarang ini sedang optimal dan HRIS tidak mengeluarkan banyak data. Dari sisi Basis Data diketahui bahwa basis data mengalami pertumbuhan log yang cukup pesat pada saat akses mendekati *deadline*, serta beberapa tabel dan view belum memiliki index, sehingga memperlambat pemrosesan aplikasi. Dari sisi Aplikasi diketahui bahwa beberapa pemanggilan *View* masih menggunakan teknologi *Entity Framework* yang belum optimal, sehingga perlu dilakukan optimalisasi dari sisi pemrograman. Selain itu, pemrograman yang kurang baik juga menyebabkan menimbunnya memori-memori sampah pada RAM yang mengakibatkan aplikasi tidak berjalan responsif. Dari tahap Analisis Penyebab ini diketahui bahwa potensi permasalahan muncul pada dua sisi, yaitu Basis Data dan Aplikasi.

Langkah selanjutnya adalah melakukan Analisis Solusi. Pada tahap ini, dilakukan analisis apakah permasalahan tersebut dapat diperbaiki tanpa melakukan restart, dan jawabannya adalah benar aplikasi dapat diperbaiki karena kita telah menemukan sumber permasalahan dan cara-cara untuk mengatasinya. Sehingga alur pengambilan keputusan akan maju ke pertanyaan apakah sumber daya dan waktu mencukupi untuk melakukan perbaikan mengingat *deadline*

sekitar dua hari lagi. Pada pertanyaan ini, jawabannya adalah tidak memungkinkan, mengingat *deadline* akan datang dalam dua hari kedepan sedangkan untuk proses perbaikan perlu dilakukan pengujian yang membutuhkan waktu lebih dari dua hari. Oleh karena itu, jawabannya adalah melakukan *restart* service aplikasi dan database secara berkala dengan mempertimbangkan *restart* akan dilakukan jika sistem dirasa sudah mulai melambat (*restart* berkala secara dinamik). Pada tahap ini juga ditentukan batas-batas kapan harus dilakukan *restart*, diantaranya yaitu ketika memori RAM sudah mencapai 95% dan CPU sudah mencapai 95%. Sehingga, kesimpulan pada tahap ini yaitu adalah melakukan *restart* secara berkala dengan metode dinamik.

Bentuk Form RDF pada kasus ini adalah sebagai berikut:



Form Analisis Restart Decision Framework

Lingkari seperlunya

1. Nama Aplikasi		E-performance	
2. Permasalahan		Aplikasi lambat saat diakses	
3. Nama & Kontak Pengembang		Ermawan , 4157	
4. Nama & Kontak IT Specialist		Budi , 4153	
5. Analisis Penyebab	a. Server	Server optimal, 2 server aplikasi dengan CPU 32, RAM 16, HD 500 Gb	
	b. Jaringan	Jaringan optimal, Load balancer round robin, no filtering	
	c. Hubungan dengan Sistem Lain	SSO & MAK optimal namun berisiko	
	d. Basis Data	Log penuh, tabel & view ada yang belum terindex	
	e. Aplikasi	Programming kurang optimal. Entity Framework membuat RAM cepat penuh	
6. Analisis Solusi	a. Dapat solve tanpa restart	YA (go to 6.b)	TIDAK* (go to 6.d: restart)
	b. Apakah resource dari time memadai	YA* (go to 6.d: solve)	TIDAK (go to 6.c)
	c. Apakah restart dibutuhkan secara flat atau dinamik	FLAT (go to 6.d: restart berkala) setiap ..... jam/hari/minggu	DINAMIS (go to 6.d: restart berkala) CPU 32 % RAM 95 % setiap ... (contoh mendekati deadline)
	d. Keputusan dan Penjelasan	Restart (Restart Berkala)/ Solve* Dilakukan restart berkala setiap mendekati threshold sampai batas akhir deadline	
	e. Langkah yang dilakukan	1. CPU CPU & RAM aplikasi 2. Jika mendekati threshold, restart service 3. aplikasi dan database 4	

Tanggal Permasalahan diketahui: 27/10/2018 DD/MM/YYYY

Tanggal Analisis RDF selesai: 26/10/2018 DD/MM/YYYY

Gambar 6. Contoh Isian Form Analisis *Restart Decision Framework*

BAB VI. KESIMPULAN, SARAN, DAN PENUTUP

A. Kesimpulan

Banyaknya contoh kasus di dunia nyata membuktikan bahwa *Restart* merupakan salah satu bentuk solusi untuk permasalahan-permasalahan tertentu dimana sumber daya dan waktu untuk memperbaiki permasalahan tersebut tidak dimungkinkan. Dari kacamata aplikasi, terdapat beberapa potensi permasalahan yang dapat disederhanakan menjadi lima sisi, yaitu: *Server*, Jaringan, Hubungan dengan Sistem Lain, Basis Data, dan Aplikasi itu sendiri. Ke lima sisi tersebut akan berguna pada saat melakukan analisis suatu permasalahan aplikasi.

Restart Decision Framework (RDF) dibangun dengan dasar lima sisi tersebut, yang bertujuan untuk memudahkan pengambilan keputusan apakah aplikasi layak untuk diperbaiki, dilakukan *restart*, atau dilakukan *restart* secara berkala. RDF mencakup analisis penyebab permasalahan, analisis solusinya, dan form untuk mencatat segala *knowledge* mengenai permasalahan aplikasi.

B. Saran

Beberapa saran dari adanya kajian ini terbagi menjadi dua hal utama. Pertama, sebaiknya kajian ini dapat ditindaklanjuti dengan sebuah keputusan yang bersifat mengikat mengenai pelaksanaan Restart Decision Framework,

sehingga akan membantu organisasi dalam melakukan restart dan mengambil analisis dan manfaatnya. Kedua, kajian ini harus disosialisasikan kepada seluruh pengelola infrastruktur Teknologi Informasi agar mengetahui prinsip-prinsip dalam melakukan analisis dan identifikasi permasalahan sistem.

C. Penutup

Diharapkan dengan adanya kajian ini, pengetahuan mengenai permasalahan aplikasi akan lebih terbuka, serta penanganannya akan lebih cepat dan mudah dilakukan dengan bantuan RDF. *Framework* ini akan membantu pengelola aplikasi mengambil keputusan dalam waktu cepat dengan memperhatikan sumber daya (*resource*) dan waktu (*time*) dalam mencapai tata kelola TIK *Data Center* Kementerian Keuangan yang lebih baik.

DAFTAR PUSTAKA

[1] Arti kata Start: https://kbbi.web.id/start

[2] Cooper, Jim (2002). *Using MS-DOS 6.22.* Que Publishing. pp. 24, 960, 964. ISBN 9780789725738.

[3] Ng, Wee Teck, Christopher M. Aycock, Gurushankar Rajamani, and Peter M. Chen. "Comparing disk and memory's resistance to operating system crashes." In *Software Reliability Engineering, 1996. Proceedings., Seventh International Symposium on*, pp. 185-194. IEEE, 1996.

[4] Candea, George, and Armando Fox. "Recursive restartability: Turning the reboot sledgehammer into a scalpel." In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pp. 125-130. IEEE, 2001.

[5] Milojicic, Dejan, Alan Messer, James Shau, Guangrui Fu, and Alberto Munoz. "Increasing relevance of memory hardware errors: a case for recoverable programming models." In Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system, pp. 97-102. ACM, 2000.

[6] Decision Making. BusinessDictionary.com. WebFinance, Inc. http://www.businessdictionary.com/definition/decision-making.html (accessed: September 17, 2018).

[7] Irham, Fahmi. "Manajemen Pengambilan Keputusan." *Bandung: Alfabeta* (2011).

[8] Arti kata Konsensus: https://kbbi.web.id/konsensus

[9] Fatah, N. (2012) Standar Pembiayaan Pendidikan. Bandung: Remaja Rosdakarya

[10] Scott Morton, Michael S. "Program management and interactive management decision systems." (1970).

[11] Zachman, John A. "A framework for information systems architecture." *IBM systems journal* 26, no. 3 (1987): 276-292.

[12] Edvardsson, Bo, and Inger Roos. "Critical incident techniques: Towards a framework for analysing the criticality of critical incidents." *International Journal of Service Industry Management* 12, no. 3 (2001): 251-268.

[13] Contoh kasus reboot router oleh FBI: https://krebsonsecurity.com/2018/05/fbi-kindly-reboot-your-router-now-please/

[14] Contoh kasus Chromecast: https://mashable.com/2018/06/27/google-home-chromecast-down/#XBPZPMf68squ

[15] Contoh kasus Mars Pathfinder: https://www.rapitasystems.com/blog/what-really-happened-to-the-software-on-the-mars-pathfinder-s pacecraft

[16] U.S. General Accounting Office. Patriot missile defense: Software problem led to system failure at Dhahran, Saudi Arabia. Technical Report GAO/IMTEC-92-26, 1992

[17] Total tiket (termasuk kemungkinan duplikasi tiket) adalah sekitar 18.182. Query langsung *database* ITSM oleh pengelola aplikasi ITSM pada tanggal 6 september 2018.

[18] Grottke, Michael, and Kishor S. Trivedi. "A classification of software faults." *Journal of Reliability Engineering Association of Japan* 27, no. 7 (2005): 425-438.

[19] Application Pools: https://docs.microsoft.com/en-us/iis/configuration/system.applicationhost/applicationpools/

[20] Appel, Andrew W. "A runtime system." *Lisp and Symbolic Computation* 3, no. 4 (1990): 343-380.

[21] Stallings, William, and Goutam Kumar Paul. *Operating systems: internals and design principles*. Pearson, 2012.

[22] Beynon-Davies, Paul. *Database systems*. Basingstoke, UK: Palgrave Macmillan, 2004.

[23] McDowell, Mindi. "Understanding denial-of-service attacks." *National Cyber Alert System, Cyber Security Tip ST04-015.2004* (2004).

[24] Boudriga, Noureddine. *Security of mobile communications*. Auerbach Publications, 2009.

[25] Kopp, Carlo. "Hardening your computing assets." *Asia/Pacific Open Systems Review. Computer Magazine Group, NSW under the title of "Information Warfare—Part* 2 (1997).

[26] Filippini, R. (1998). Trade-off and compatibility between performance: definitions and empirical evidence. *International Journal of Production Research*, *36*(12), 3379-3406.

[27] Turner, Jonathan S. "Packet switching loop-around network and facilities testing." U.S. Patent 4,486,877, issued December 4, 1984.

[28] Arti kata RAM: https://www.webopedia.com/TERM/R/RAM.html

[29] He, Jingsha. "System and method for single sign-on to a plurality of network elements." U.S. Patent 5,944,824, issued August 31, 1999.

[30] Chen, Mingtse, Anil K. Annadata, and Leon Chan. "Adaptive communication application programming interface." U.S. Patent 7,581,230, issued August 25, 2009.

[31] Cristian, Flaviu. "Synchronous and asynchronous." *Communications of the ACM* 39, no. 4 (1996): 88-97.