Review Paper Stress Testing pada Aplikasi Mobile

Farhan Ramadhani
Departemen Ilmu Komputer dan Elektronika Instrumentasi
Universitas Gadjah Mada
Yogyakarta, Indonesia
Farhanrd96@gmail.com

***Abstrak*—Menjaga performa dari suatu aplikasi mobile adalah suatu keharusan bagi seorang developer. Meningkatnya keberagaman dan kompleksitas dari populasi aplikasi mobile menjadi tantangan dalam upaya meningkatkan performa dari aplikasi. Serta sulitnya dalam memprediksi aktivitas user dimasa mendatang juga menjadi tantangan dalam upaya peningkatan performa. Stress testing adalah salah satu cara agar developer bisa menjaga dan memantau performa dari aplikasi. Paper ini akan membahas metode-metode yang digunakan dalam melakukan stress testing di sistem aplikasi berbasis mobile. Setiap metode Stress testing yang dijalankan akan menghasilkan informasi mengenai batasan aplikasi tersebut dalam kondisi stress (*Abstrak*)**

Keywords—mobile; testing; stress testing; (key words)

I. PENDAHULUAN

Perkembangan perangkat lunak atau keras yang berbasis kan mobile sudah sangat pesat. Terdapat 500 juta perangkat mobile yang akan meluncur ditahun 2012 dibanding dengan computer yang hanya berjumlah 150 juta [1].Oleh karena itu, Keberagaman dan kompleksitas dari perangkat mobile pun tidak terelakkan. Beragamnya software/hardware yang berkembang menyebabkan banyaknya aplikasi yang tidak bisa bertahan lama di lingkungan bisnis. Kegagalan ini disebabkan oleh pola pikir developer yang hanya berfokus pada fungsionalitas perangkat lunaknya saja, tanpa mempertimbangkan soal kompleksitas dan ketahanannya. Sehingga, kehadiran *bugs* pada aplikasi yang dikembangkan juga tidak terelakkan. Kasus inilah yang menjadi alasan kenapa proses pengujian/*testing* pada pengembangan perangkat lunak sangat diperlukan.

A. Keberagaman & Kompleksitas

Merujuk ke [1] terdapat factor keberagaman & kompleksitas akan hardware & software dari proses pengujian suatu system. Dalam proses mobile testing, penguji harus mempertimbangkan keberagaman dari perangkat lunak atau perangkat keras dari suatu handphone. Di beberapa negara, Penyedia layanan internet menyediakan *device* yang sudah diubah konfigurasi perangkat keras nya dari settingan Pabrik. Hal ini dilakukan untuk kepentingan komersil serta perusahaan. Kebijakan seperti ini mempunyai dampak terhadap kalangan *tester*/penguji. Dengan semakin bervariasinya konfigurasi suatu device/handphone yang beredar. Maka penguji akan semakin ditantang untuk memprediksi efek nya terhadap performa, keamanan dan *usability*. begitu juga dari segi *software/*perangkat lunak, penguji juga harus mengejar perkembangan dari system operasi mobile seperti Windows mobile, Android, Symbian dan Iphone. Hal ini bertujuan agar penguji bisa me-*maintain* ilmu dari alat test terhadap perkembangan perangkat lunak mobile yang cukup pesat.

Tidak hanya keberagaman dari *hardware* dan *software*. Kompleksitas juga menjadi tantangan bagi *tester*. Sebagai contoh,

- Perbedaan ukuran memory pada handphone sangat berpengaruh terhadap proses pengujian suatu aplikasi
- Kapabilitas dari *GPS receiver* dalam menerima sinyal dari berbagai lokasi
- Kapabilitas dari *akselerometer* dalam mentransformasikan kondisi layar dari potret ke lansekap atau sebaliknya.

Dari segi software, contoh kompleksitas yang terjadi adalah pengujian perangkat lunak dengan bandwidth koneksi internet yang berbeda menghasilkan hasil pengujian yang berbeda juga.

B. Test Environment

Proses pengujian system berbasis mobile juga harus dipertimbangkan lingkungan/*environment* dari prosesnya. Selvam R et al juga menguraikan bagaimana *environment* dari proses testing. terdapat 2 environment yang diuraikan oleh Selvam R et al. Yaitu pengujian pada *environment* emulator dan Handheld yang sebenarnya. Pengujian pada emulator menawarkan cost yang efektif karena bisa mencoba berbagai platform dan vendor yang ada. Namun hasil dan proses pengujian tidak bisa di simulasikan secara realistis. Seperti mengambil foto/video dan stabilitas performa mesin. Sedangkan Pengujian dengan menggunakan handheld yang sebenarnya, lebih menghasilkan hasil pengujian yang optimal pada scenario yang nyata. Tapi dengan cost yang lebih tinggi. Selvam R et al merekomendasikan pengujian dengan emulator terlebih dahulu, kemudian diakhiri dengan pengujian handheld yang kongkrit.

C. Test Levels

Proses testing menurut studi [1], terdiri dari beberapa level. Level-level tersebut diuraikan dalam bentuk point seperti berikut.

- **Unit testing** adalah test yang dilakukan oleh programmer. Bertujuan untuk menemukan dan memperbaiki error yang ada. Unit didefinisikan sebagai potongan code yang bisa di compile, link, dan

di load seperti *function/procedures*, *class*, dan *interface*.

- **Integration testing** adalah test yang memverifikasi *interface* dan desain software dengan cara memaparkan kekurangan interaksi antar komponen, contoh di system operasi Android adalah integrasi antar *Activity*.
- **System testing** adalah pengujian terhadap seluruh system yang terintegrasi apakah sesuai dengan *requirement* yang ditentukan oleh aplikasi-nya.
- **User interface testing** adalah pengujian terhadap QWERTY keyboard, layar sentuh, resolusi & orientasi layar, aksesoris yang kompatibel dengan handheld, touchpad, dan GUI (ukuran tombol)
- **Regression testing** adalah proses eksekusi ulang dari subset test *acceptance* setelah terdapat beberapa perubahan. Perubahan yang dimakud seperti *bug-fixes* dari developer untuk update system. Test regresi mendesain scenario pengujian yang berfokus pada perubahan tersebut (*localized*)
- **Acceptance testing** adalah sebuah kumpulan test dalam skala besar. Bertujuan untuk memverifikasi produk tertentu berada pada spesifikasi yang dibutuhkan. Aplikasi yang lolos dari pengujian ini sering dijadikan benchmark terhadap aplikasi yang lain.

Berdasarkan point-point yang sudah diuraikan, level unit test merupakan pengujian dengan level terendah. Dan hanya level Unit test yang dipaparkan implementasi-nya oleh Selvam R et al. Implementasi ini akan dibahas pada paper review ini dibagian tertentu.

D. *Test Scope*

Proses testing menuruts studi [1], terdiri dari beberapa ruang ligkup. Ruang lingkup tersebut diuraikan dalam bentuk point seperti berikut.

- **Functional testing,** prosedur yang mem-verifikasi fungsionalitas yang mendasar dari system mobile, contohnya fungsionalitas dari system operasi, browser, wireless carrier, lokasi dan bahasa. Prosedur ini sangat esensial.
- **Performance testing,** mengujikan aplikasi dengan berbagai macam kondisi seperti pada koneksi Wi-Fi, koneksi 3G/2G, tidak ada simcard, mode Pesawat, dan koneksi dari PC melalui USB.
- **Security testing,** pengujian ini mempertimbangkan proses perpindahan data yang sensitive/penting serta *user privileges* atau *application execution permissions.*
- **Usability testing,** pengujian ini bisa dikaitkan dengan permasalahan dari user experience suatu system.
- **Load/stress testing,** pengujian ini bertujuan untuk menemukan error, hangs, dan deadlocks yang bisa saja terjadi selama penggunaan yang fungsional atau masa pengujian. Pengujian ini dilakukan dengan melakukan suatu operasi dengan beban tertentu, dan perulangan yang banyak.
- **Localization testing,** proses pengujian ini berlangsung dengan berfokus pada bahasa dan globalitas-nya. Contohnya adalah menerjemahkan user interface sesuai dengan budaya/bahasa tertentu.
- **Synchronization testing,** merupakan pengujian terhadap fitur singkronisasi pada suatu aplikasi. Singkronisasi adalah fitur untuk melakukan pergantian, perubahaan data antar 2 aplikasi yang berbeda

Pada paper review ini akan lebih berfokus pada pengujian stress atau *stress testing.* Sebelum membahas tentang metode *stress testing* yang ada, diperlukan pengetahuan tentang parameter *stress testing* dan *stress testing* itu sendiri.

E. *Stress testing*

Stress testing adalah pengujian yang dilakukan untuk menentukan stabilitas suatu system. Pengujian yang dilakukan berada diatas batas operasional pada kondisi normal. *Stress testing* akan memperlihatkan bagaimana *behavior* dari system jika suatu scenario pengujian dikenakan pada system tersebut. Skenario dari *stress testing* memerlukan sebuah parameter. Parameter ini sangat mempengaruhi akurasi dan reliabilitas dari stress testing.

F. *Parameter Stress testing*

Dalam proses analisis *stress testing*, penguji harus mengetahui parameter-parameter yang digunakan dalam proses stress testing. Parameter ini sangat mempengaruh akurasi dan reliabilitas dari stress testing [2]. Meng Zhang et al bagaimana menentukan metode-metode yang digunakan untuk mengestimasi parameter pengujian menggunakan stokastik proses dan teori antrian. Berikut metode-metode yang digunakan oleh studi [2]:

- Metode untuk mengestimasi jumlah user yang *concurrent*

Dalam pemahamannya, jumlah user yang concurrent diarahkan ke *login session*. Login session adalah waktu tertentu pada saat user login hingga user berhenti/logout. Selama selang waktu tersebut, system akan membuat user *session* untuk setiap *login session*, maka jumlah user yang *concurrent* pada saat momen ke- t bisa d definisikan sebagai total waktu untuk semua *login session* pada momen ke- t. Maka rumus matematika-nya :

$$Rata - rata\ jumlah\ user\ yang\ concurrent\ (C) = \frac{total\ waktu\ semua\ login\ session}{T} \quad (1)$$

Jika terdapat *n login session* pada waktu T, dan rata-rata lama nya 1 *login session* maka persamaan diatas menjadi :

$$C = \frac{nL}{T} \quad (2)$$

Pada proses *stress testing*, informasi jumlah user concurrent paling tinggi/maksimum juga di perlukan. Sehingga dilakukan pendekatan dengan Probabilitas Poisson. Probabilitas Poisson sering digunakan untuk membuat model dari jumlah kedatangan user pada waktu tertentu. Dimisalkan jumlah *login session* per waktu adalah distribusi probabilitas poisson dengan parameter λ, maka

$$P(\text{jumlah } login\ session \text{ per waktu} = \text{x}) \frac{e^{-\lambda}\lambda^{x}}{x!} \quad (3)$$

Menurut sifat dari persamaan distribusi probabilitas poisson, parameter λ juga menghitung nilai rata-rata, sehingga cocok dengan definisi rata-rata jumlah user yang concurrent (C) dari persamaan sebelumnya. Sehingga jika C di substitusikan ke persamaan distribusi probabilitas poisson menjadi :

$$P(\text{jumlah } concurrent \text{ use per waktu} = \text{x}) \frac{e^{-C}C}{x!} \quad (4)$$

Karena menggunakan distribusi probabilitas poisson maka nilai variansi dan ekspektasi nya sama dengan nilai C. Kemudian dengan menggunakan persaman Chebyshev $P\{|X - C| < \varepsilon\} \geq 1 - \frac{C}{\varepsilon^2}$, dan menentukan nilai ε menjadi $4\sqrt{C}$ dan $5\sqrt{C}$ maka di dapatkan asumsi bahwa batas bawah dari peluang kejadian mendekati 1 mempunyai peluang yang mendekati 1 juga. Maka didapat nilai *concurrent* user yang maksimum yaitu :

$$\hat{C} \approx C + 4\sqrt{C}\hat{C} \approx C + 4\sqrt{C}$$
$$\hat{C} \approx C + 5\sqrt{C} \quad (5)$$

Selain itu, penulis juga membandingkan metode yang ia ciptakan dengan penulis lain. Dalam paper, penulis membandingkan analisis dari kedua metode. Kesimpulan yang penulis ini tuliskan adalah metode yang ia ciptakan lebih bisa diandalkan, karena metode yang menjadi saingan menggunakan pendekatan probabilitas distribusi normal untuk menghitungkan jumlah concurrent user yang maksimum, sedangkan distribusi normal tidak ideal ketika menghadapi sifat numerical dari jumlah *concurrent* user

- Metode untuk mengestimasi *throughput*

Throughput merupakan jumlah *request* yang harus diproses setiap waktunya. Secara umum, *throughput* diukur berdasarkan $\frac{request}{time}$ atau $\frac{jumlah\ page\ views}{time}$. berikut rumus untuk menghitung *throughput:*

$$F = \frac{N_{vu}\ x\ R}{T} \quad (6)$$

Dengan F adalah *throughput*, N_{vu} adalah jumlah user yang *concurrent*, R adalah jumlah *request* per user. Dan T merupakan waktu test.

- Metode untuk mengestimasi pengguna *think-time*

Think-time atau Sleep-time adalah waktu interval yang terjadi antara 2 *request*. Karena pada dasarnya tidak mungkin ketika user mengoperasikan suatu system dengan melakukan request secara non-stop. Untuk menentukan waktu *think-time* lebih *reasonable* penulis menyajikan rumus berikut:

$$R = \frac{T}{T_s} \quad (7)$$
$$R = jumlah\ request\ per\ user$$
$$T = waktu\ test$$
$$T_s = pengguna\ thinking - time$$

Dengan mensubstitusikan rumus diatas ke rumus *throughput* maka didapat:

$$T_s = \frac{N_{vu}}{F} \quad (8)$$

Untuk mencapai tujuan test, maka persamaan diatas didasarkan atas unit waktu. Selain itu, ide "*zero think-time*" bisa dilakukan untuk menguji system dibawah tekanan yang sangat berat, maka ide ini bisa diadopsi. Meskipun tidak begitu realistis.

- Metode untuk mengestimasi penambahan *users-arrival*

Terdapat 2 metode dalam hal *users-arrival.* Pertama, adalah user diakses secara bersamaan atau serentak. Yang kedua adalah jumlah pengaksesan user naik secara bertahap. Meskipin dari kedua metode ini mengakses user dengan jumlah yang sama. Namun, hasil dari pengujian stress akan berbeda.

Misalkan *user's-arrival* dilakukan dengan pendekatan distribusi poisson. Rata-rata user's arrival adalah λ, nilai *service rate (jumlah request per waktu)* adalah tetap sebagai μ, dan ada 1 sistem untuk *service.* Dengan menggunakan teori antrian. Kita bisa menentukan rata-rata waktu respon $\overline{W}$ menjadi:

$$\overline{W} = \frac{1}{\mu} + \frac{\lambda}{2\mu(\mu - \lambda)} \quad (9)$$

Kemudian λ bisa ditentukan sebagai:

$$\lambda = \frac{(2\overline{W}\mu - 2)\mu}{2\overline{W}\mu - 1} \quad (10)$$

Rumus diatas digunakan untuk menetukan rata-rata penambahan user yang concurrent.

Setelah menentukan metode-metode untuk mengestimasi parameter dalam *stress testing*, Selanjutnya adalah melakukan pengujian stress terhadap system *"Grid-based Railway Cargo Information System".* Sistem ini melacak data dari kereta api yang terhubung pada layanan organisasi dan layanan kargo, Analisis statistika pendapatan kargo, dan pemasaran kargo. User dari system ini adalah Kementrian Kereta api, Biro Kereta api dan Khalayak umum. Setelah system dianalisis dan user disurvey, maka didapat:

1. Rata-rata jumlah user perharinya adalah 600 user
2. Rata-rata *login session* user adalah 4 jam per 8 jam kerja

3. Rata-rata setiap user mengoperasikan 500 *queries* pada *login session*-nya.
4. Sistem bisa mengatasi 5 request user dalam waktu yang bersamaan.
5. Rata-rata waktu respon yang bisa di tolerir oleh user adalah 10 detik.

Dengan menggunakan metode estimasi yang sudah dijelaskan, maka kita bisa menentukan hal-hal seperti berikut:

1. Rata-rata jumlah user yang *concurrent:*
   600 x 4/8 = 300
2. Jumlah maksimum user yang *concurrent:*

$300 + 4\sqrt{300} = 360$

3. *Throughput*

$$\frac{300x500}{4x60x60} \approx 10\ (page\ wiews\ per\ detik)$$

4. *Think-time*

300/10 = 30 Second

5. Penambahan *user's arrival*

$$\lambda = \left[\frac{(2x5x10 - 2)x5}{2x5x10 - 1}\right] = 4$$

Menurut analisis dari system yang akan di uji, maka ada 2 hal yang harus diperhatikan pada saat *stress testing:*

1. Waktu respon halaman harus kurang dari 10 detik dengan jumlah *concurrent* user yang di rata-ratakan.
2. System harus 48 jam stabil dengan jumlah *concurrent* user yang maksimum. Stabil yang dimaksud adalah tidak pengurangan *resource* dari system dan perubahan tingkat respon user yang signifikan.

Kemudian stress testing diterapkan pada 30 halaman web. Kemudian hasil nya adalah 21 halaman memiliki waktu respon dibawah 10 detik, sisanya tidak. Grafik dibawah menampilkan hubungan jumlah *concurrent* user dengan waktu respon (detik).

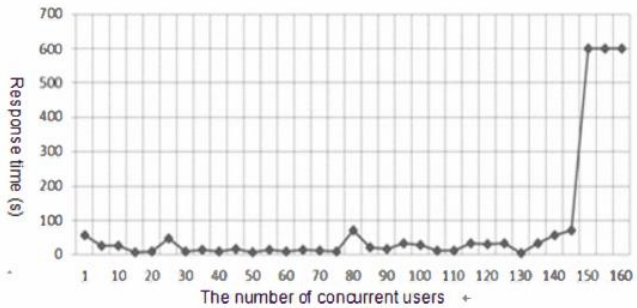


Figure 1. Relational graph between the number of concurrent users and response time

Figure.1 Grafik antara jumlah konkuren user dan waktu respon.

Seperti yang sudah diuraikan dibagian sebelumnya. Proses testing bisa dijalankan pada 2 *environment.* Yaitu pada emulator dan *device* yang nyata. Terdapat beberapa aplikasi emulator yang bisa mendukung proses *stress testing*.

G. Stress testing dengan aplikasi emulator

Merujuk ke studi [3], penulis melakukan pengujian terhadap situs *m-commerce* berbasis jQuery yaitu ebay (https://m.ebay.in). website ini diuji menggunakan Aplikasi *Webserver stress tool. Webserver stress tool* adalah aplikasi *HTTP-client /server* yang didesain untuk mencari titik kritis yang menimbulkan *error* pada aplikasi *web based m-commerce.* Aplikasi ini akan membantu penguji untuk mempelajari batas dari *traffic* webserver dan melihat respon webserver ketika melebihi batas-nya [3]. *Webserver stress tool* akan mensimulasikan sejumlah user untuk mengakses *website* melewati *HTTP/HTTPS* secara bersamaan. User akan melakukan *request* terhadap suatu webpage dari URL. Setiap user akan disimulasikan dengan session yang berbeda-beda. Aplikasi ini menawarkan tiga tipe *stress test*, yaitu:

1. *Click*
   Click merupakan pengujian dengan mensimulasikan user untuk menginisiasi click dengan jumlah tertentu.
2. *Time*
   Test ini akan menerapkan beban yang konstan untuk waktu yang ditentukan.
3. *Ramp*
   test ini juga menerapkan beban dengan waktu tertentu. Namun beban akan disimulasikan dari 1 user teringkrementasi hingga jumlah user yang sudah ditentukan. Proses ini berjalan hingga 80% waktu pengujian berjalan. Sisa-nya 20%, akan mensimulasikan seluruh user yang ada.

Berikut hasil dari pengujian yang dilakukan oleh aplikasi *Webserver stress tool.*

- Hasil test *Click*

Table II Results per user

User No.	Clicks	Hits	Errors	Avg. Click Time [ms]	Bytes	kbit/s	Cookies
1	24	24	0	25,710	3,521,426	45.66	
2	44	44	0	12,615	8,643,082	124.58	
3	67	67	0	7,628	10,188,885	159.50	
4	40	40	0	11,986	1,316,934	21.97	
5	6	6	0	69,307	882,693	16.98	
6	24	24	0	15,083	4,808,549	106.27	
7	26	26	0	12,266	3,954,914	99.21	
8	5	5	1	52,894	152,202	4.60	
9	3	3	0	76,264	389,857	13.63	
10	11	11	0	15,244	2,201,850	105.05	

Table I Hasil per User

Tabel diatas menunjukkan hasil dari test *click* yang dijalankan aplikasi *webserver stress tool* per user-nya. User menginisiasi *click* dengan jumlah yang sesuai pada kolom **Clicks, Hits** adalah proses HTTP request yang sudah dikirim dan di respon oleh server. **Errors** adalah error yang terjadi dalam proses pengujian. **Avg. Click Time** adalah rata-rata waktu yang dihitung ketika proses *request* selesai alias semua data/*pages* sudah ditransfer ke *client*. Dan **Bytes & kbit/s**

adalah kolom yang mendata ukuran halaman yang di request serta bandwidth dari network-nya. Hanya ditemukan 1 error pada proses test *Click* per user. Yaitu pada user ke-depalan. Table III merupakan hasil *click* test per URL-nya. Pada setiap URL yang di uji, aplikasi akan mensimulasikan user untuk menginisiasi *click* pada URL tersebut. Namun, di dalam paper tidak disebutkan bagaimana penguji melakukan *sequencing* pada URL-nya. Berdasarkan Aplikasi Webserver Stress tool proses *sequencing* sendiri terdiri dari 4 jenis, yaitu:

1. User memilih URL per *click*-nya secara random
2. User meng-*click* URL yang sama.
3. User akan meng-*click* semua URL secara berurutan
4. User akan men-*click* URL x, kemudian URL random, dan terakhir URL y.

Table III Results per URL

URL No.	Name	Clicks	Errors	Errors [%]	Time Spent [ms]	Avg. Click Time [ms]
1		30	2	6.67	906,912	32,390
2		76	1	1.32	910,490	12,140
3		91	0	0.00	798,865	8,779
4		43	2	4.65	348,747	8,506

Tabel II Hasil per URL

Setiap kolom bisa direpresentasikan secara harfiah. Dan ditemukan beberapa error pada URL tertentu.

- Hasil test *Ramp*

Test *Ramp* yang dilakukan menghasilkan grafik figure II. Grafik ini bersifat dinamis, sehingga kita bisa melihat lebih detil lagi pada bagian grafik tertentu. Fitur ini akan membantu penguji untuk mengidetifikasikan *errors* dan *bottlenecks. Bottlenecks* adalah momen dimana server tidak lagi merespon *request* dari *client.* Grafik yang ditampilkan adalah:

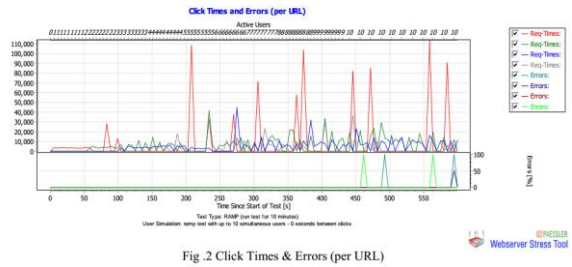


Fig .2 Click Times & Errors (per URL)

Figure.2 Click Times & Error (per URL)

Grafik ini memperlihatkan hasil dari pengujian *Ramp* 10 akses user. Antar click user terdapat jeda 0 detik. Grafik ini juga menampilkan tingkat *error* yang dialami oleh user ketika men-*download* halaman web yang di*request.* Untuk setiap URL (1 URL diwakili warna yang berbeda), grafik ini menampilkan ***request times of clicks*** dan persentase *error.* Diketahui *error* muncul pada saat aplikasi mulai mensimulasikan 10 user secara bersamaan pada akhir waktu test.

Selain [3], studi [4] juga melakukan pengujian dengan metode stress testing. Pengujian dilakukan terhadap system *home care agencies.* System ini akan terhubung ke *smartphone* pegawai untuk mengedit data dari pasien secara langsung. Sistem akan diuji dengan sebuah aplikasi desktop yang dikembangkan oleh penulis. Aplikasi penguji ini mensimulasikan beberapa akses user dalam waktu yang silmultan atau bersamaan. Aplikasi ini sudah terintegrasi dengan sebuah Web service. ***Time response*** dari Web service inilah yang nanti nya akan dijadikan tolak ukur untuk menentukan perangkat keras yang tepat [4]. Aplikasi desktop ini mengukur dan membuat :

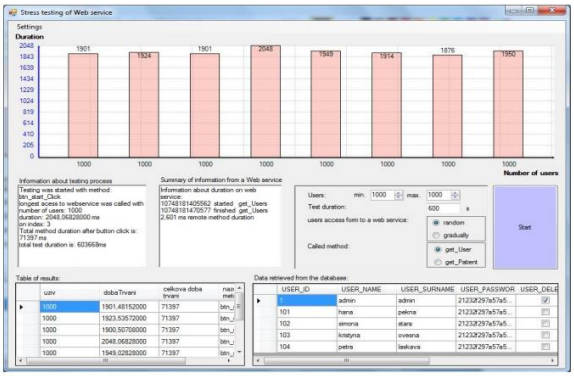
1. Durasi dari 1 ***call*** oleh sebuah method
2. Durasi dari sebuah call untuk user tertentu
3. Durasi respon dari web servis untuk jumlah user tertentu serta waktu perjalanan oleh web servis
4. Membangun koleksi data yang diperlukan dari database
5. Cara koleksi data berpindah dari web servis ke aplikasi.
6. Durasi dari proses ***start*** hingga aplikasi selesai ***running***.
7. 

Figure 4 Appearance of test application

Figure 3 Tampilan Aplikasi testing

Cara kerja aplikasi ini adalah dengan menggenerate jumlah user yang sudah ditentukan. Untuk menggenerate user, aplikasi menyediakan opsi apakah user ingin di generate secara progresif(***gradually***) atau secara random. Jika opsi progresif di pilih maka aplikasi akan menggenerate user dengan ***user_id*** yang memiliki nilai terkecil. Jumlah user yang di simulasikan akan di-***increment*** hingga akhirnya mencapai jumlah ***maximum.*** Pada percobaan opsi progresif, menghasilkan grafik seperti berikut :

Figure 4 Overhead webservice pada percobaan pertama

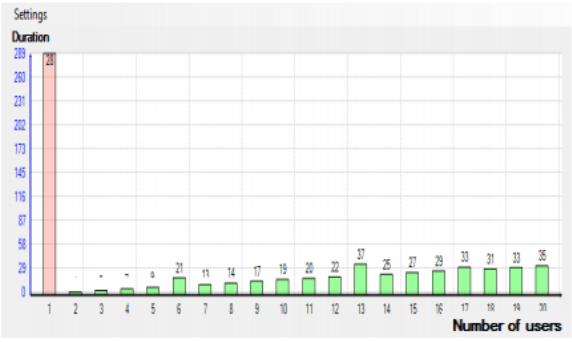


Figure 1: Test conducted after the start - in the first column shows the bias of measurement when establishing a connection to the server and compile a Web service

Figure 5 hasil test tanpa webservice overhead

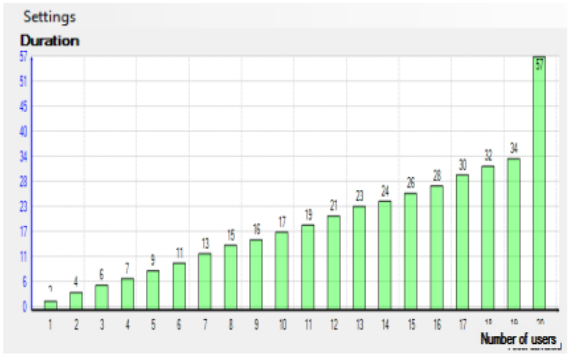


Figure 2 Test without the overhead of Web service - it displays the test output obtained by measuring with active connection with the server and compiled the Web service

Figure 4 dan Figure 5 mewakili hasil dari pengujian yang di jalankan aplikasi dengan mesimulasikan user secara prograsif. Grafik menunjukkan adanya lonjakan durasi pada saat aplikasi menggenerate 1 user. Menurut penulis, hal ini disebabkan adanya jeda untuk menghubungkan koneksi server dengan web servis. Secara detil, Penulis menjelaskan bahwa figure 4 memperlihatkan hasil tes setelah beralih ke aplikasi testing.

Pada figure 5 terjadi perbedaan durasi yang mencolok. Perbedaan tersebut diperlihatkan pada jumlah user 19 dan 20. Dijelaskan didalam paper bahwa perbedaan ini kemungkinan disebabkan oleh server yang mendapatkan *request* yang lebih penting pada saat itu. Sehingga, request dari aplikasi stress testing tidak langsung di eksekusi.

Pada percobaan simulasi user secara random. Aplikasi akan memilih user dengan nomor ***user_id*** yang random antara jumlah ***minimum & maximum,*** proses ini akan diulang hingga period dari testing berakhir.Proses testing melibatkan 2 mesin database yang berbeda. Sebuah PC dengan processor 2x4GHz, 6 GB DDR2 RAM memory, dan yang kedua adalah PC dengan 850Mhz AMD Processor, dan 512 MB memory SDRAM. Berikut table dari tipe test yang diujikan, setiap tipe dilakukan dengan 5 kali perulangan

Table III tipe pengujian

Method of users generating	Users range	Test duration/step
2x4GHz, 6GB DDR2 RAM		
gradually	1 – 30	step = 1
gradually	1 – 1000	step = 50
gradually	100 – 200	step = 10
gradually	200 – 500	step = 20
gradually	500 – 800	step = 20
gradually	500 – 1000	step = 50
gradually	800 – 1000	step = 10
gradually	1 – 100	30 s, 60 s, 120 s
gradually	1 – 200	30 s, 60 s, 120 s
gradually	1 – 500	3 min, 5 min, 10 min
gradually	1 – 700	3 min, 5 min, 10 min
gradually	1 – 800	3 min, 5 min, 10 min
random	1 – 900	3 min, 5 min, 10 min
random	1 – 1000	3 min, 5 min, 10 min
random	1000	10 min
AMD 850MHz, 512MB SD RAM		
gradually	1 – 30	step = 1
gradually	1 – 1000	step = 50
gradually	100 – 200	step = 10
gradually	200 – 500	step = 20
gradually	500 – 800	step = 20
gradually	500 – 1000	step = 50
gradually	800 – 1000	step = 10
gradually	1 – 200	120 s
random	1 – 700	10 min
random	1 – 1000	10 min
random	1000	10 min

Table 1 Types of performed tests

Figure 6 menunjukkan hasil percobaan simulasi 500-1000 user yang di generate secara progresif, dapat dilihat bahwa antara durasi dan jumlah user saling berbanding lurus. Artinya semakin banyak jumlah user yang di simulasikan semakin banyak pula durasi yang diperlukan. Selain itu, adanya sedikit perbedaan antar iterasi dari setiap proses simulasi user, penulis menjelaskan bahwa perbedaan tersebut disebabkan oleh jeda waktu yang terjadi ketika ada pergantian proses pada server, dan bias juga ***network latency.*** Diketahui perbedaan durasi antara simulasi 500 users adalah 2.5 detik berdasarkan mesin. Dan 1000 user sebanyak 5 detik.

Figure 6 500-1000 user secara progresif

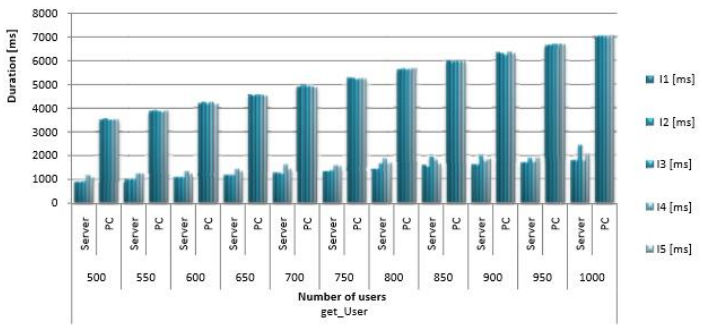


Figure 3 Contingency graph for 500 - 1000 users - shows the results of all five iterations of measurement and both the PC machine type and the server machine type

Figure 7 1000 user secara random pada mesin Server

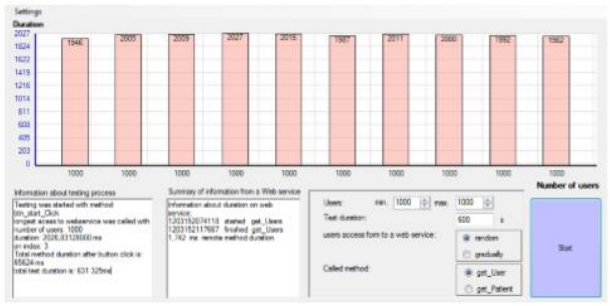


Figure 5 Chart of results for the machine of server type - Iteration 5 - simulating 1000 simultaneous access to a web service at each step, the duration of the processing approaches are not very different from each other and reach a value of about 2 seconds

Figure 6 Result of measurement for the machine of PC type - Iteration 5 - also simulates 1000 simultaneous access to a web service but on a weaker machine, access time of up to seven seconds

Figure 8 1000 user secara random pada mesin Server

Berdasarkan grafik hasil test di atas, figure 7 merupakan output grafik dari test yang dikenakan pada mesin server (2x2GHz 6GB memory). Durasi dari waktu respon berkisar 2 detik. Sedangkan untuk hasil pengujian pada mesin PC (850 MHz, 512 MB) seperti pada figure 8, berkisar 7 detik di setiap iterasinya. Kemudian penulis menyatakan bahwa beban yang dipikul oleh ***Webservice*** berbanding lurus dengan jumlah akses user per 100 milisecond- nya.

Kemudian studi [4] menjelaskan beberapa kesimpulan yang bisa ditarik secara eksplisit yaitu :

1. Mesin/Server yang digunakan sangat mempengaruhi hasil durasi dari pengujian
2. Waktu respon akan naik seiring dengan bertambahnya jumlah record database.
3. Waktu respon bias terpengaruhi oleh network latency
4. Durasi yang diperlukan untuk memproses *query* berbanding lurus dengan jumlah pengaksesan user serta mesin penyimpan database yang digunakan.

Pada bagian sebelumnya, Selvam R et al menjelaskan salah `satu ruang lingkup dari proses testing adalah *Unit testing*. Proses *Unit testing* juga diuraikan di dalam paper. *Unit testing* yang diterapkan menggunakan *automated testing*. Pengujian dilakukan pada sebuah aplikasi yang dikembangkan menggunakan *IDE Eclipse*. Proses pengujian didukung oleh framework unit testing dari java yang bernama *JUnit*.Mekanisme pengujian dilakukan dengan membuat class baru yang akan digunakan untuk menguji sebuah class atau disebut *test class.* Berdasarkan paper yang direview, Selvam R et al memaparkan potongan kode yang digunakan untuk menguji sebuah *Activity* dan *view*. Berikut potongan kode-nya:

```
package com.utest.helloandroid.test;
import com.utest.helloandroid.HelloAndroid;
import android.test.ActivityInstrumentationTestCase2;
import android.widget.TextView;
public class HelloAndroidTest extends
ActivityInstrumentationTestCase2<HelloAndroid> {
        private HelloAndroid mActivity; // the activity under
test
        private TextView mView; // the activity's TextView
(the only view)
        private String resourceString;
    public HelloAndroidTest() {
        super("com.utest.helloandroid", HelloAndroid.class);
    }
@Override
    protected void setUp() throws Exception {
    super.setUp();
    mActivity = this.getActivity();
    mView =
    (TextView)mActivity.findViewById(com.example.helloand
    roid.R.id.textview);
    resourceString =
    mActivity.getString(com.example.helloandroid.R.string.hell
    o);
        }
public void testPreconditions() {assertNotNull(mView);}
public void testText()
{assertEquals(resourceString,(String)mView.getText());}}
```

Constructor dari class test case mendeklarasikan 3 variabel yaitu *mActivity, mView(TextView), dan resourceString* bersifat *private*. Class *HelloAndroidTest()* akan mewarisi sifat class super nya yaitu class *HelloAndroid.* Kemudian code akan menjalankan 3 method, yaitu:

- *setUp(),* method ini akan membuat *context* dari *mActivity, mView(TextView), dan resourceString.*
- *testPreconditions(),* method ini akan melakukan *assertion* terhadap *mView(TextView).* jika fungsi *asserionnotnull()* tidak bernilai null, maka pengujian *mVIew* berhasil.
- *testText(),* method ini akan melakukan *assertion* dengan membandingkan variable *resourceString* dengan teks yang didapatkan dengan memanggil *getText()* pada *mView.* Jika string-nya sama, maka pengujian berhasil.

H. Stress testing pada Device Smartphone

Studi [5] melakukan penelitian tentang *software aging*. Untuk mengetahui ada atau tidak fenomena ***software aging***

pada Sistem Operasi Android. Maka [5] melakukan pengujian ***stress testing*** pada sebuah handheld dengan spesifikasi memory 1GB, 1,4GHz CPU, 32Gb memory penyimpanan dan Android versi 2.3. Pengujian dilakukan dengan sebuah aplikasi yang mengumpulkan informasi memory ***RAM*** dari ***route "/proc/meminfo"***. Setelah itu pengujian dilakukan selama 24 jam dan aplikasi tersebut akan merekam data ketersediaan memory setiap 5 menitnya. ***Stress testing*** dilakukan dengan 2 metode, yaitu stress test pada memory dengan beban tetap dan stress test pada memory dengan beban eksponensial :

- **Stress test memory dengan beban tetap.**

Pada metode ini, penhujian dilakukan dengan mengalokasikan objek dengan nilai yang tetap setiap 100milisecond nya. Terdapat 3 jenis data yang di deklarasikan yaitu "String (N_s)", "Double(N_d)", dan "Integer(N_i)" untuk dialokasikan. Ketiga data tersebut dijumlahkan menjadi $N^m_f(k)$. k adalah level dari memory stress yang ke-*k*. yang berarti jika waktu total dari pengujian adalah T miliseconds, berarti $\frac{T}{100} = k$

$$N^m_f(k) = N_s + N_d + N_i \qquad (11)$$

Pada percobaan, N_s di set 2450 bytes, N_d 17 bytes, N_i 8 bytes, Sehingga total-nya 2475 bytes atau 2.42 KiloBytes.

- **Stress test memory dengan beban eksponensial.**

Metode ini mengalokasikan objek dengan cara eksponensial, setiap level k akan di alokasikan N_e sebanyak 2^{k-1} kali. Berikut ekspresi matematika-nya

$$N_e^m(k) = \sum_{k=1}^{2^{K-1}} N_s + N_d + N_i \qquad (12)$$

- **Hasil Percobaan**

Dalam proses ***stress testing*** dengan beban tetap, ditemukan bahwa ketersediaan memory pada saat ***stress test***(96854 kb) 20900kb lebih sedikit daripada ketersediaan memory pada saat normal (117754kb). Dengan semakin menurunnya ketersediaan memory pada metode ini, maka Sistem operasi android memiliki suatu strategy agar bisa mempertahankan kestabilan system dari ***crash.*** Strategi ini disebut GC (Garbage Collection)[5].

Figure 9 Grafik *Available Memory* dengan *fix workload*

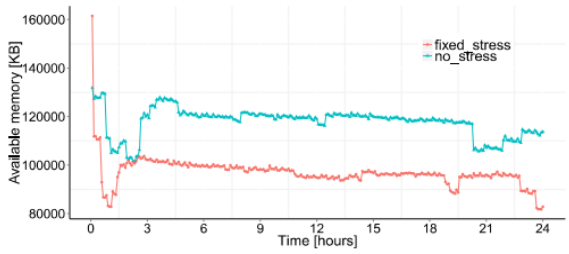


Figure 1. The result of available memory under memory stress test with fixed workload for 24 hours

Ketika memory yang tersedia sudah mendekati ***threshold***-nya. Maka GC akan melepaskan beberapa objek yang sudah teralokasi dari memory agar system bisa kembali stabil. Pada eksperimen ini, memory yang di alokasikan untuk aplikasi adalah 13767 KB. Ketika nilai *k* mencapai 8000, dan jumlah alokasi memory saat itu hamper mencapai 12000 KB. Maka Garbage Collection pun di aktifkan. Proses ini dipantau melalui Eclipse monitor. GC mengumpulkan sekitar 400 KB agar system tidak ***crash.*** Namun tetap saja kurva dari grafik 9 tidak mengalami penaikan. Bahkan, semakin menurun. Berdasarkan kejadian penulis mengungkapkan adanya gejala-gejala dari fenomena ***software aging.***

Pada pengujian dengan beban eksponensial, ***available memory*** pada saat ***stress test*** (52993 KB) 64761 KB lebih sedikit daripada keadaan normal (117754 KB). Berdasarkan percobaan, penulis menemukan adanya indikasi ***memory leaks*** pada k=245**.** Sehingga aplikasi ter-***suspend*** kemudian mengaktifkan GC. ***Memory leaks*** adalah sebuah objek yang gagal di bersihkan/di bebaskan dari memory oleh GC .

Figure 10 Grafik *Available Memory* dengan eksponensial *workload*

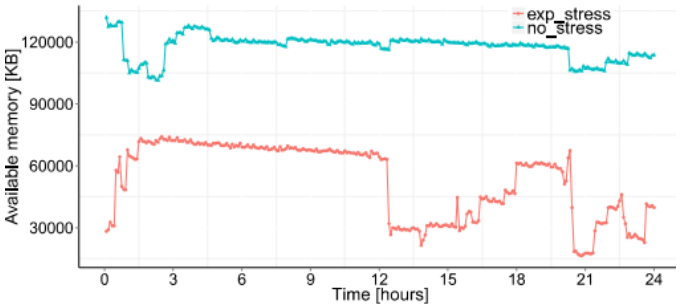


Figure 2. The result of available memory under memory stress test with xponential workload for 24 hours

Sama halnya dengan percobaan stress test beban tetap. Jumlah ***Available memory*** berdasarkan grafik 10 semakin menurun seiring nilai k- bertambah. Ditemukan saat nilai k= 13 dan jumlah memory yang di alokasikan sebanyak 19824.64 KB, dengan ***treshold*** senilai 14151 KB. Maka GC diaktifkan karena memory yang di alokasikan lebih besar dari ***threshold.*** Meskipun GC rutin di aktifkan setiap nilai alokasi memory Lebih besar dari ***threshold.*** Namun, kurva dari grafik 10 tetap mengalami penurunan.

Menurut hasil ke-2 percobaan diatas, system operasi android berkemungkinan menunjukkan indikasi/gejala ***software aging*** setelah dilakukan ***stress testing*** dengan level yang berbeda-beda.

II. KESIMPULAN

Melalui paper review ini beragam test scenario dari *stress testing* sudah dipaparkan. Setiap proses pengujian menghasilkan informasi mengenai batasan system dalam kondisi yang sudah ditentukan. Baik itu kondisi dari *environment* emulator maupun *device* yang nyata. Informasi inilah yang nanti-nya akan menjadi tolak ukur bagi *developer* untuk memantau dan meningkatkan performa dari aplikasinya.

REFERENSI

[1] S. R and D. V. K. , "Mobile Software Testing - Automated Test Case DEsign Strategies," *Internation Journal on Computer Science and Engineering (IJCSE)*, vol. 3, pp. 1450-1461, 2011.

[2] M. Zhang, H. Li and J. Xiao, "The Estimation Method of Commen Testing Parameters in Software Stress Testing," *International Conference on Computer Science and network Technology*, pp. 1468-1472, 2011.

[3] M. Asokan and D. P. Arul, "Stress Testing for J-query Based M-Commerce Mobile Web Applicatins Using Webserver Stress Tool," *Internaional Journal of Advanced REsearch in Computer Sciencce and Software Engineering*, vol. 11, no. 11, pp. 545-551, 2015.

[4] L. Motalova and O. Krejcar, "Stress Testing Data Access via a Web Service for Determination of Adequate Server Hardware for Developer Seoftware Solution," *The Second International Conference on Computer Engineering and Applications*, pp. 329-333, 2010.

[5] Y. Zhao, J. Xiang, S. Xiong, Y. Wu, J. An, S. Wang and X. Yu, "An Experimental Study on Software Aging in Android Operating System," *2nd International Symposium on Dependable Computing and Internet of Things*, pp. 148-150, 2015.