JURNAL EMACS
[Engineering, MAthematics and Computer Science] Vol.3 No.2 May 2021: 79-85
e-ISSN: 2686-2573
DOI: 10.21512/emacsjournal.v3i2.7386

Pengembangan Simulasi Kejadian Diskret Berbasis Paket *Simmer* pada R

I G.A. Anom Yudistira*
Statistics Department, School of Computer Science,
Bina Nusantara University,
Jakarta, Indonesia 11480
i.yudistira@binus.ac.id

*Correspondence: i.yudistira@binus.ac.id

Abstract *– This study aims to describe the various capabilities of the simmer package on R, especially in running a discrete event simulation model, then develop a DES simulation model building technique, which is effective and can represent real systems well, and explore the simulation output on this simmer, both in statistical summary form and parameter estimation. The method used in this research is the literature study, with descriptive and exploratory approaches. Model development is more effective when it is carried out starting from simple models, to more complex forms step by step, and describing the system using a flow chart. Replication for simulations is easy to perform, so as to get standard error values for model parameter estimators. The stages in developing a discrete event simulation model with a simmer, start with compiling a simple flowchart to a more complex form, and replication is carried out. The simmer output in the form of data.frame makes it very easy to further process the output. The simple R API on simmer will also make it easier to simulate.*

Keywords*: Simulation; DES; simmer; R programming.*

Abstrak *– Penelitian ini bertujuan untuk men-deskripsikan berbagai kemampuan package simmer pada R khususnya dalam menjalankan model simulasi kejadian diskret, mengembang-kan suatu teknik pembangunan model simulasi DES, yang efektif dan bisa merepresentasikan sistem nyata secara baik, dan mengeksplorasi keluaran simulasi pada simmer ini, baik dalam bentuk ringkasan statistik dan pendugaan parameternya. Metode yang digunakan dalam penelitian ini adalah metode kepustakaan, dengan pendekatan deskriptif dan eksploratif. Pengembangan model lebih efektif bila dila-kukan mulai dari model yang sederhana, hingga bentuk-bentuk yang lebih kompleks secara langkah demi langkah, dan mendeskrip-sikan sistem menggunakan diagram alir. Replikasi untuk simulasi mudah dilakukan, sehingga mendapatkan nilai galat baku untuk penduga parameter model. Tahap-tahap dalam mengembangkan model simulasi ke-jadian diskret dengan simmer, dimulai dengan menyusun bagan alir sederhana sampai dalam bentuk yang lebih kompleks, serta dilakukan replikasi. Keluaran simmer dalam bentuk* `data.frame`*, sangat memudahkan dalam mengolah keluaran tersebut lebih lanjut. R API pada simmer yang sederhana juga akan memudahkan dalam melakukan simulasi.*

Kata kunci*: Simulasi; DES; simmer; Pemograman R.*

I. PENDAHULUAN

Model simulasi digunakan untuk mempelajari suatu sistem, khususnya apabila sulit / mahal atau berbahaya jika mempelajari sistem secara langsung (Banks, 2014). Disamping itu sistem nyata yang kompleks, hampir tidak mungkin dipelajari dengan model analitik, untuk itu simulasi merupakan pilihan terakhir yang tidak bisa di tawar lagi. Selanjutnya Shannon (1975), memberikan definisi terhadap simulasi yaitu suatu proses untuk mendesain model sistem nyata dan melakukan eksperimen terhadap model tersebut, untuk tujuan memahami perilaku sistem atau mengevaluasi berbagai strategi atau seperangkat kriteria.

R merupakan suatu bahasa pemrograman fungsional, yang berbasis sumber terbuka *(open source*), sehingga bisa diakses secara gratis. Bahasa R ini sangat dikenal dikalangan ilmuwan dan statistikawan, bahkan merambah penggunaannya diberbagai bidang terapan. Dikalangan akademisi, R banyak digunakan untuk membantu dalam aspek komputasi untuk memecahkan masalah penelitian. Dalam hal kajian simulasi, R memiliki kekuatan pada simulasi monte-carlo (*static stochastic simulation techniques*). Kelemahan utama R sebelum tahun 2017,

adalah tidak tersedianya suatu *package* untuk memecahkan masalah simulasi kejadian diskret (*discrete event simulation, DES*) yang handal. Sehingga pemrograman DES dengan menggunakan R menjadi sangat sulit. Sejak tahun 2017 dikembangkan *package* `simmer` (Ucar dan Smeets, 2017), yang merupakan *package* DES untuk R yang memungkinkan pemodelan berorientasi proses tingkat tinggi, sejalan dengan simulator modern lainnya. Ucar, Smeets dan Azcorra (2017), menyebutkan bahwa, paket simmer memudahkan untuk membangun model simulasi kejadian-diskrit pada R. Paket dirancang sebagai kerangka kerja berorientasi proses yang generik namun handal, yang ditulis dalam C++, sehingga eksekusinya relatif cepat dan robust. Paket simmer juga dilengkapi dengan kemampuan monitoring yang otomatis, dengan keluaran defaultnya dalam bentuk tabel. Paket ini juga menyediakan API untuk R yang kaya dan fleksibel, yang berpusat di sekitar konsep lintasan, yaitu jalur umum dalam model simulasi untuk lintasan entiti. Fungsi `trajectory()` digunakan untuk membangkitkan objek dengan kelas `trajectory`. Hanya saja belum dikembangkan teknik untuk membangun model DES berbasis simmer, sehingga efektif untuk merepresentasi-kan sistem yang kompleks.

1.1 Tujuan

Penelitian ini bertujuan untuk (1) mendeskripsikan berbagai kemampuan *package* simmer pada R untuk menjalankan model simulasi kejadian diskret (DES), (2) mengembangkan suatu teknik pembangunan model simulasi DES, yang efektif dan bisa merepresentasikan sistem nyata secara baik. Serta (3) mengeksplorasi keluaran simulasi dengan menggunakan *package* simmer ini, baik dalam bentuk ringkasan statistik dan pendugaan parameternya. R yang memiliki kekuatan pada analisis statistik dan grafik, akan sangat menarik apabila dikombinasikan dengan kemampuan *package* simmer. Sinergi package simmer dan package – package lain pada R tentu akan menarik untuk dianalisis dan dibahas dalam penelitian ini.

Pesaing utama simmer adalah SimPy (Tim SimPy, 2017) dan SimJulia (Lauwens, 2017), yang masing-masing dikembangkan untuk Bahasa Python dan Julia.

II. METODOLOGI PENELITIAN

Metode yang digunakan dalam penelitian ini adalah metode kepustakaan, dengan pendekatan deskriptif dan eksploratif.

Langkah-langkah dalam penelitian ini adalah:

1) Telaah pustaka untuk mendapatkan pemahaman yang lengkap terhadap R API (antarmuka pemrograman aplikasi). R API untuk simmer adalah berbentuk fungsi-fungsi R.
2) Telaah pustaka untuk mendapatkan pemahaman yang lengkap terhadap struktur lingkungan dari kelas simmer.
3) Membangun bagan atau flowchart yang menggambarkan sistem
4) Membangun script R yang dalam bentuk yang paling sederhana
5) Meningkatkan kompleksitas sistem satu tingkat
6) Apakah cukup mewakili sitem, jika belum kembali kelangkah 5)
7) Selesai

2.1 Batasan

Proses pemodelan dan simulasi disajikan pada yaitu dimulai dari perumusan masalah, sampai pada pengambilan keputusan. Penelitian ini di-batasi hanya pada aspek pembangunan model, khususnya pada pengembangan model simu-lasi. Aspek pengumpulan data, pengujian asumsi-asumsi model dan pelaksanaan percobaan terhadap model diluar ruang lingkup penelitian ini.

III. HASIL DAN PEMBAHASAN

Sistem Kejadian Diskret yang paling sederhana adalah sistem antrian dengan server tunggal (antrian tunggal). Deskripsi sistemnya adalah sebagai berikut: Suatu sistem mempunyai server tunggal yang melayani entiti untuk suatu transaksi. Bagannya adalah sebagai berikut,

Gambar 1. Sistem sederhana (server tunggal)

Segiempat menunjukkan suatu aktivitas sedangkan bentuk bulat/oval menunjukkan status sistem. Setiap aktivitas yang diawali oleh status antri, yang artinya aktivitas tersebut membutuhkan sumberdaya. Entiti datang dengan suatu proses kedatangan tertentu (waktu konstan atau acak), kemudian waktu kedatang-annya (tiba) dicatat sebagai *start time*. Entiti menuju server untuk melakukan proses transaksi, tetapi apabila server tidak tersedia (sedang melayani entiti lain), maka entiti masuk dalam antrian. Entiti mendapat layanan server selama waktu tertentu, dicatat sebagai *activity time* (konstan atau acak). Setelah entiti selesai mendapatkan proses layanan, maka akan keluar dari sistem kembali ke lingkungan, dicatat sebagai *end time*. Waktu *start time*, *end time* dan *activity time* akan dicatat (dimonitor) oleh simmer secara otomatis. Catatan waktu ini dapat diperoleh dengan menggunakan fungsi `get_mon_arrival`, yang disajikan dalam objek `data.frame` (tabel). Sistem ini mem-punyai dua jenis input simulasi yaitu waktu kedatangan / antar kedatangan, dan waktu aktivitas server (lamanya server melayani). Misalnya kedatangan entiti menyebar eksponesial dengan laju entiti per unit waktu dan lama layanan menyebar eksponensial dengan laju entiti per unit waktu (angka 0.4 dan 0.45 hanyalah sebagai gambaran agar bisa dieksekusi oleh R). Lintasan entiti adalah tiba mengambil sumber daya (*seize*) proses pelayanan yang mebutuhkan waktu (*timeout*) melepas sumber daya (*release*) keluar dari sistem. *Script* R untuk lintasan ini adalah sebagai berikut:

```
library(simmer) # memuat package simmer
env <- simmer("DES01") # membuat objek berkelas
simmer
# mendefiniskan input simulasi
# peubah acak Waktu Antar Kedatangan (AK)
AK <- function() round(rexp(n=1, rate=0.40),3)
# peubah acak lamanya layanan
Lama <- function() round(rexp(n=1, rate=0.45),3)
```

```
# mendefinisikan lintasan entiti
lintas <- trajectory() %>%
  seize("server") %>%
  timeout(Lama) %>%
  release("server")
```

Script R untuk simulasi, selalu didahului dengan pendefinisian lingkungan simmer (`env <- simmer("DES01")`). Nama lingkungan-nya adalah "DES01" dan disimpan pada objek `env`.

Kemudian dilanjutkan dengan pendefinisan input simulasi. Koding untuk lintasan terdiri dari 3 fungsi utama, yaitu penciptaan kelas `trajectory`, `seize`, `timeout` dan `release`. Apabila aktivitas tidak membutuhkan sumber-daya, maka fungsi `seize`, dan `release` dihilangkan. Input simulasi untuk lamanya aktivitas server melayani suatu entiti ("Lama"), dijadikan sebagai nilai argumen untuk fungsi `timeout`. Setelah mendefinisikan lintasan entiti, pada lingkungan simulasi `env` ditambah-kan sumberdaya (resource) dengan nama "Server", sejumlah yang tersedia (dalam hal ini 1). Fungsi R (pada paket simmer) untuk melakukan ini adalah `add_resource`. Kemudian linkungan `env`, ditambahkan fungsi untuk membangkitkan entiti, beserta waktu kedatangan / antar kedatangan, yaitu meng-gunakan fungsi `add_generator`. *Script* secara lengkap adalah sebagai berikut:

```
env %>%  # env adalah nama objek
         # lingkungan yang telah
         # didefinsikan diawal
  add_resource("server", 1) %>%
         # jumlah server hanya 1
  add_generator("entiti", lintas, AK) %>%
  run(50)
# simulasi dijalankan tidak
# melebihi 50 unit waktu
```

Hasil monitoring simulasi dapat diambil dengan fungsi `get_mon_`, yang hasilnya berupa data.frame (tabel).

```
output.arr <- get_mon_arrivals(env)
output.resc <- get_mon_resources(env)
```

Fungsi `get_mon_attributes()`, akan menghasilkan nilai-nilai *attribute* dari simulasi. Pada kasus sekarang ini belum menggunakan fungsi *attribute*.

Fungsi `run(50)` menunjukkan bahwa simulasi dijalankan sampai 50 unit waktu. Jadi secara garis besar script R untuk simulasi kejadian diskret (DES), dengan menggunakan `simmer` terdiri dari empat komponen dasar, yaitu

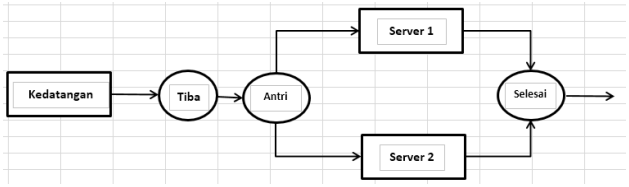
1) *Inisialisasi*
   a) Pendefinisan lingkungan simulasi ➔ `env <- simmer()`
   b) Pendefinisan input simulasi ➔ `AK <- function(); Lama <- function()`
   c) Pendefinisian lintasan ➔ `lintas <- trajectory()`
      …
      `# objek lintas berkelas`
      `# trajectory`
2) Penambahan resources dan pembangkitan kedatangan entiti pada lingkungan simulasi:
   a) `add_resources()`
   b) `add_generator()`
3) Menampilkan hasil simulasi, yang berupa tabel
   a) `get_mon_arrivals()`
   b) `get_mon_resources()`
   c) `get_mon_attributes()`

Hasil monitoring kedatangan (*arrivals*) dan sumberdaya (*resources*), dapat dilihat pada lampiran.

3.1 Peningkatan Kompleksitas Sistem

- **Penambahan jumlah server paralel identik dan antrian tunggal**

Sekarang *server* ditambah menjadi dua unit *server* yang identik. Entiti yang datang akan mengantri pada satu jalur antrian, dan segera akan mendapatkan layanan, jika minimal salah satu dari dari kedua *server* dalam keadaan *free* (*idle*) sehingga siap dipakai. Diagram sistem antrian dengan dua *server* paralel dan satu jalur antrian adalah sebagai berikut:



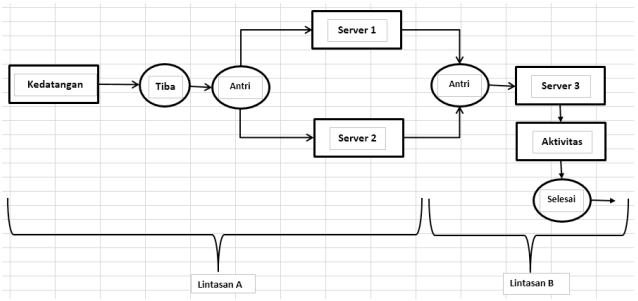
Gambar 2. Sistem Paralel Identik

Jumlah server yang paralel dapat diperluas menjadi n > 2 server. Perubahan hanya dilaku-kan pada penambahan jumlah (kapasitas) *resources*, yaitu menjadi dua, sedangkan kom-ponen lain tidak berubah.

```
env %>%
  add_resource("server", 2) %>%
# jumlah (kapasitas) "resources" = 2
  add_generator("entiti", lintas,
    AK) %>%
  run(50) %>% invisible
```

- **Penambahan server serial dan aktivitas tanpa *resources***

Apabila sistem dikembangkan dengan deskripsi sebagai berikut: Setelah entiti mendapatkan la-yanan disalah satu server 1 atau server 2, ke-mudian berlanjut secara serial ke server 3, setelah dari server 3 entiti melakukan aktivitas tanpa *reources*. Bagannya digambarkan seba-gai berikut:



Gambar 3. Sistem Serial

Pada bagan terlihat lintasan entiti dapat dibagi menjadi dua bagian, yaitu lintasan A dan lintasan B. Hal ini dilakukan semata-mata agar memudahkan dalam membuat programnya. *Script* untuk lintasan pun kita buat menjadi dua sebagai berikut:

```
# 1) Inisialisasi
env <- simmer("DES01")
AK <- function() round(rexp(n=1,
        rate=0.40),3)
Lama <- function() round(rexp(n=1,
        rate=0.45),3)
Lama3 <- function() 5
Lama4 <- function() 3
```

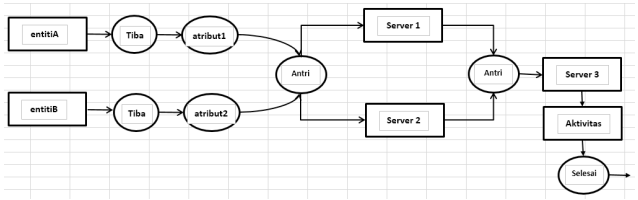
```
# 2) Pendefinisian lintasan
lintas_B <- trajectory() %>%
  seize("server3") %>%
  timeout(Lama3) %>%
  release("server3") %>%
  timeout(Lama4) # aktivitas tanpa
# resource (tidak ada antrian)

lintas_A <- trajectory() %>%
  seize("server") %>%
  timeout(Lama) %>%
  release("server") %>%
  join(lintas_B)

# 3) Penambahan resources dan pem-
#    bangkitan kedatangan entiti
env %>%
  add_resource("server", 2) %>%
  add_resource("server3", 1) %>%
  add_generator("entiti",
     lintas_A, AK) %>%
  run(100)

# 4) Mengambil hasil simulasi
output.arr <- get_mon_arrivals(env)
output.resc <- get_mon_resources(env)
```

Perhatikan komponen ke-2 pada program. Pendefinisian lintasan dengan menggunakan fungsi `trajectory()` dipecah dua menjadi Lintasan_A (`Lintas_A`) dan Lintasan_B (`Lintas_B`). Lintasan_B diletakkan pada bagian atas, sebelum Lintasan_A, karena fungsi `trajectory` dengan objek Lintasan_A, di dalamnya ada memanggil Lintasan_B. Jadi `trajectory` Lintasan_B harus dikenal lebih dahulu sebelum `trajectory` Lintasan_A dibaca. Hal ini terjadi karena program dibaca dari atas ke bawah. Aktivitas tanpa sumberdaya ada pada lintasan B, yang dilambangkan pada diagram dengan bentuk segiempat dengan nama "Aktivitas" yang tidak didahului oleh lingkaran oval antrian. Pada *script* diwakili oleh fungsi `timeout(Lama4)`.

3.2 Sistem dengan lebih dari satu jenis entiti

Misalkan setelah dipelajari lebih lanjut, ternyata ada dua jenis entiti yang masuk kedalam sistem yang diberi nama entitiA dan entitiB. Diasumsikan entitiA datang setiap 8 menit sekali, sedangkan entitiB datang setiap setiap 12 menit sekali. Dibatasi ada 100 entitiA dan 50 entitiB yang harus menyelesaikan lintasan. Lama waktu yang dibutuhkan untuk memproses entitiA (pada server 1 atau 2) adalah menyebar Eksponensial dengan rate 1/7 per unit waktu, sedangkan entitiB (pada server 1 atau 2) membutuhkan waktu yang juga menyebar eksponensial tetapi dengan rate 1/12 per unit waktu (nilai asumsi ini diberikan semata-mata, agar program dapat dijalankan). Untuk menyederhanakan model, diasumsikan bahwa entitiA datang secara konstan setiap 8 unit waktu sekali, sedangkan entiti B datang setiap 12 unit waktu sekali. Bagan lintasannya akan digambarkan seperi berikut ini:

Pada bagan diperlihatkan, bahwa ada dua lintasan baru, yaitu kedatangan entitiA dan entitiB. Pada program R objek lintasan ini diberi nama lintas_entA dan lintas_entB. Pada kedua lintasan ini, setiap entiti yang tiba akan diberi nilai atribut dengan nama "Lama", yang akan digunakan untuk menentukan lamanya layanan baik diserver 1 atau di server 2. Kompleksitas program, digambarkan oleh bagan lintasan berikut:



Gambar 4. Sistem dengan lebih dari satu jenis entiti

Script R diberikan sebagai berikut:

```
# 1) Inisialisasi
env <- simmer("DES01")

AK <- function() round(rexp(n=1, rate=0.40),3)
Lama3 <- function() 5
Lama4 <- function() 3

# 2) Pendefinisian lintasan
lintas_B <- trajectory() %>%
  seize("server3") %>%
  timeout(Lama3) %>%
  release("server3") %>%
  timeout(Lama4)

lintas_A <- trajectory() %>%
  seize("server") %>%
  timeout(Lama) %>%
  release("server") %>%
  join(lintas_B)

lintas_entA <- trajectory() %>%
  set_attribute("lama",
     function() rexp(1, 1/7)) %>%
  join(lintas_A)

lintas_entB <- trajectory() %>%
  set_attribute("lama",
     function() rexp(1, 1/12)) %>%
  join(lintas_A)

# 3) Penambahan resources dan
#    pembangkitan kedatangan entiti
env %>%  # env adalah nama objek
         # lingkungan yang telah
         # didefinsikan diawal
  add_resource("server", 2) %>%
  add_resource("server3", 1) %>%
  add_generator("ent_A",
    lintas_entA, at(cumsum(rep(8,
         100))), mon=2) %>%
  add_generator("ent_B",
    lintas_entB, at(cumsum(rep(12,
         50))), mon=2) %>%
  run() %>% invisible()

# 4) Mengambil hasil simulasi
output.arr <-
   get_mon_arrivals(env)
output.resc <-
   get_mon_resources(env)
output.attr <-
   get_mon_attributes(env)
```

Pada komponen inisialisasi, tidak lagi ada pemberian nilai untuk objek `lama`, karena nilai ini ditentukan oleh nilai *atribute* setiap entiti yang didefinikan pada *trajectory* `lintas_entA` dan `lintas_entB`. Fungsi yang digunakan adalah `set_attribute`. Sedangkan pada komponen ke-3 (penambahan resources dan pembangkitan kedatangan) *argumen* `mon` pada fungsi `add_generator`,

diberi nilai 2, sehingga dapat dimonitor oleh fungsi `get_mon_attributes`, yang akan memberikan tabel untuk nilai-nilai atribut setiap entiti.

3.3 Replikasi

Proses kompleksitas model akan berlanjut terus, sehingga benar-benar mencerminkan sistem yang akan ditiru. Kompleksitas tersebut dapat berupa percabangan (*Branching*), *Lopps*, *Batching* dan *Reneging*. Penggunaan komponen-komponen itu tentu tergantung, apakah sistem memerlukan itu. Akan tetapi, setiap simulasi akan membutuhkan proses replikasi, yaitu pengulanngan simulasi sampai R kali (R > 100). Proses replikasi ini bertujuan untuk memperoleh simpangan baku dari pendugaan-pendugaan parameter sistem.

Berikut ini adalah *script* R untuk proses replikasi model, menggunakan fungsi `lapply` yang tersedia pada paket *base*. Script R berikut ini untuk melakukan pengulangan sebanyak 100 kali

```
# 3) Proses replikasi penambahan
#    resources dan pembangkitan
#    kedatangan entiti
env_repl <- lapply(1:100,
       function(j){
  simmer("DESRep") %>%
  add_resource("server", 2) %>%
  add_resource("server3", 1) %>%
  add_generator("ent_A",
    lintas_entA, at(cumsum(rep(8,
        100))), mon=2) %>%
  add_generator("ent_B",
    lintas_entB, at(cumsum(rep(12,
        50))), mon=2) %>%
  run() %>% invisible()
})

# 4) Mengambil hasil simulasi
output.arr <-
    get_mon_arrivals(env_repl)
output.resc <-
    get_mon_resources(env_repl)
output.attr <-
    get_mon_attributes(env_repl)
```

3.4 Analisis Output dengan "dplyr"

Paket "dplyr" sangat membantu dalam memanipulasi data dalam bentuk *data frame*, seperti membuat ringkasan data, mengolah kolom-kolom dan membentuk kolom baru dan seba-gainya. Pada proses replikasi simulasi diper-oleh hasil berupa *data frame*, yaitu `output.arr`, `output.resc`, dan `output.attr`. Script berikut akan mentransformasi output menjadi berkelas tbl, yaitu kelas data yang digunakan oleh paket dplyr.

```
out.arr <- as_tibble(output.arr)
out.resc <- as_tibble(output.resc)
out.arr <- as_tibble(output.attr)
sapply(list(out.arr, out.resc,
       out.attr), class)

# output
     [,1]          [,2]          [,3]
[1,] "tbl_df"      "tbl_df"      "tbl_df"
[2,] "tbl"         "tbl"         "tbl"
[3,] "data.frame" "data.frame" "data.frame"
```

Sekarang ketiga objek (`out.arr`, `out.resc`, dan `out.attr`) telah ditransformasi menjadi berkelas `tbl`, disamping juga berkelas `data.frame`, dan siap untuk dioleh dengan paket dplyr. Dimensi ketiga objek set data ter-sebut adalah:

```
sapply(list(out.arr, out.resc,
       out.attr), dim)
      [,1]  [,2]  [,3]
[1,] 15000 60000 15000
[2,]     6     9     5
```

Jadi `out.arr` memiliki 15.000 baris dan 6 kolom, `out.resc` dengan 60.000 baris dan 9 kolom, sedangkan `out.attr` memiliki 15.000 baris dan 5 kolom.

3.5 Peringkasan Hasil Simulasi

Peringkasan berikut menghasilkan antrian terpanjang pada setiap server (resource), yaitu

```
out.resc %>%
  dplyr::select(resource, time,
          queue, system) %>%
  group_by(resource) %>%
  summarise(max_Q=max(queue))
# A tibble: 2 x 2
  resource max_Q
  <chr>    <int>
1 server       2
2 server3      8
```

Antian terpanjang terjadi pada server3 yaitu sampai 8 entiti. *Script* berikut ini memberikan rata-rata panjang antrian untuk setiap *resorce* per replikasi, yaitu

```
out.resc %>% group_by(resource,
     replication) %>%
summarise(across(c(queue, system),
     mean, .names = "Avg_{.col}"))

# A tibble: 200 x 4
# Groups:   resource [2]
   resource replication Avg_queue Avg_system
   <chr>          <int>     <dbl>      <dbl>
 1 server             1   0           0.767
 2 server             2   0.00333     0.753
 3 server             3   0.00333     0.773
 4 server             4   0           0.76
 5 server             5   0           0.727
 6 server             6   0           0.727
 7 server             7   0.00333     0.76
 8 server             8   0.00333     0.747
 9 server             9   0.00333     0.767
10 server            10   0.00333     0.753
# ... with 190 more rows
```

Bila ingin mendapatkan lamanya entity ada dalam antrian dan ada di dalam sistem, maka dapat digunakan perintah `mutate` pada *dplyr*, yaitu sebagai berikut,

```
Out.arr.new <- out.arr %>%
   dplyr::select(start_time, end_time,
                 activity_time, replication) %>%
   mutate(system_time=end_time - start_time,

          Q_time=round(system_time -
                   activity_time,7))

# A tibble: 15,000 x 6
   start_time end_time activity_time replication system_time Q_time
        <dbl>    <dbl>         <dbl>       <int>       <dbl>  <dbl>
 1         12     21.5          9.48           1        9.48   0
 2          8     26.5         14.2            1       18.5    4.27
 3         16     31.5          8.84           1       15.5    6.64
 4         24     36.5          8.15           1       12.5    4.34
 5         24     41.5         16.2            1       17.5    1.24
 6         32     46.5          8.46           1       14.5    6.02
 7         36     51.5          8.25           1       15.5    7.23
 8         40     56.5         10.3            1       16.5    6.16
 9         48     61.5          9.05           1       13.5    4.43
10         48     66.5         12.6            1       18.5    5.89
# ... with 14,990 more rows
```

Fungsi mutate menghasilkan kolom baru yaitu system_time yang memberikan lamanya entity di dalam system dan Q_time yang mem-berikan lamanya entity di dalam antrian, kemudian hasil ini disimpan pada objek data out.arr.new. Berdasarkan ouput terakhir ini, dapat dicari nilai rata-rata lamanya entiti di dalam system dan lamanya entiti di dalam antrian, yaitu sebagai berikut:

```
out.arr.new %>%
   group_by(replication) %>%
   summarise(across(c(system_time,
        Q_time),mean,
        .names="Avg_{.col}"))

# A tibble: 100 x 3
   replication Avg_system_time Avg_Q_time
         <int>           <dbl>      <dbl>
 1           1            25.0       14.8
 2           2            23.6       13.4
 3           3            23.8       13.4
 4           4            22.9       12.5
 5           5            21.5       11.5
 6           6            24.3       14.1
 7           7            23.8       13.5
 8           8            23.2       13.2
 9           9            26.2       15.9
10          10            26.4       16.2
# ... with 90 more rows
```

Jika yang diinginkan adalah rata-rata untuk keseluruhan replikasi, maka perintah di atas dapat dilanjukan menjadi sebagai berikut:

```
out.arr.new %>%
  group_by(replication) %>%
  summarise(across(c(system_time,
      Q_time), mean,
      .names="Avg_{.col}")) %>%
  summarise(across(c(Avg_system_time,
      Avg_Q_time), mean,
      .names="Avg_{.col}"))

# A tibble: 1 x 2
  Avg_Avg_system_time Avg_Avg_Q_time
                <dbl>          <dbl>
1                23.9           13.7
```

3.6 Pendugaan Interval Kepercayaan

Berdasarkan objek data out.arr.new diper-oleh rata-rata lamanya entiti di dalam sistem dan di dalam antrian, yang diberikan oleh kolom Avg_system_time dan Avg_Q_time. Kedua kolom tersebut dapat diambil dengan menggunakan fungsi attach.

```
out.arr.new %>%
  group_by(replication) %>%
  summarise(across(c(system_time,
     Q_time), mean,
     .names="Avg_{.col}")) %>%
  attach()

head(Avg_system_time,4)
[1] 25.02309 23.61274 23.81376 22.86765

head(Avg_Q_time,4)
[1] 14.76864 13.41091 13.35512 12.51495
```

Galat baku (*standard error*) untuk lamanya entiti di dalam sistem dan di antrian, dapat didekati oleh simpangan baku dari rata-rata per replikasi, diperoleh sebagai berikut:

```
(SE_system <- sd(Avg_system_time))
[1] 1.64213

(SE_Q <- sd(Avg_Q_time))
[1] 1.620795
```

Sehingga *margin error* untuk kedua variable adalah 2 kali galat bakunya. Jadi penduga interval kepercayaan masing-masing untuk lamanya entiti di dalam sistem dan lamanya entiti di dalam antrian adalah sebagai berikut:

```
out.arr.new %>%
 group_by(replication) %>%
   summarise(across(c(system_time,
      Q_time), mean,
      .names="Avg_{.col}")) %>%
   summarise(across(
      c(Avg_system_time,
      Avg_Q_time), mean,
      .names="Avg_{.col}")) %>%
   attach()

c(Avg_Avg_system_time,
   Avg_Avg_Q_time)

[1] 23.90671 13.66237
```

Jadi pada kasus ini, interval kepercayaan rata-rata lamanya entiti di dalam sistem adalah:

```
Avg_Avg_system_time +
      c(-1,1)*2*SE_system
[1] 20.62245 27.19097
```

Sedangkan interval kepercayaan rata-rata lamanya entiti di dalam antrian adalah:

```
Avg_Avg_Q_time +
      c(-1,1)*2*SE_Q
[1] 10.42078 16.90396
```

IV. KESIMPULAN

R API pada *simmer* terdiri dari dua elemen utama, yaitu lingkungan simulasi *simmer* – yang diciptakan melalui fungsi simmer()-, dan objek lintasan (dibuat oleh fungsi trajectory()). Seperti yang telah dipaparkan dalam tulisan ini simulasi dengan *simmer*, berarti membangun lingkungan simulasi, dengan satu atau lebih lintasan-lintasan. Lingkungan simulasi akan menyimpan dan memonitor semua nilai dan perubahan-perubahan yang terjadi pada peubah acak dan atribut entiti selama simulasi berlangsung. Pengembangan simulasi terhadap sistem akan lebih mudah dilakukan, dengan membuat suatu bagan alir (*flowchart*) yang merepresentasikan sistem dengan jelas. Semua nilai-nilai peubah acak dan perubahan-perubahan yang terjadi terhadapnya, dapat diambil kembali dari lingkungan simulasi simmer, dengan menggunakan perintah get_mon_. Hasil dari perintah get_mon_ ini berupa objek dengan kelas data.frame, yang selanjutnya dapat diolah kembali dengan menggunakan fungsi-fungsi R lainnya terutama dplyr.

Agar hasil simulasi menjadi lebih menarik, akan lebih baik dikembangkan kembali pemba-hasan tentang simmer.plot dan dihubungkan dengan kemampuan *package* ggplot2, yang sangat baik dalam memvisualisaikan suatu dataset. Kemampuan *simmer* dalam memba-ngun simulasi sirkular (seperti sistem tranportasi pulang pergi untuk

angkutan umum massal), juga belum diekplorasi lebih jauh. Kompleksitas sistem belum sepenuhnya dieksplorasi pada tulisan ini seperti adanya layanan prioritas untuk entiti tertentu, *balking* dan *reneging*, serta entiti berbentuk *bulk*.

DAFTAR PUSTAKA

Banks, Jerry, John Carson II, Barry Nelson, and David Nicol. Discrete-Event System Simulation, 5th Edition. (2014) Pearson Education Limited, Edinburgh Gate, England

Lander, Jared P. (2017). R for Everyone: Advanced Analytics and Graphics 2nd Edition. Pearson Education, Inc. USA.

Smeets, Bart and Inaki Ucar. (2020). Intro-duction to simmer. *https://r-simmer.org/articles/simmer-01-introduction.html*

Ucar, Inaki, Bart Smeets, & Arturo Azcorra. (2019). simmer: Discrete-Event Si-mulation for R. Journal of Statistical Software, Vol 90, Issue 2. *https://www.jstatsoft.org/article/view/v090i02*