Domain_metric

August 23, 2021

[1]:
```
##DESCRIPTION
# This notebook calculates the so called "Polygons" to describe how a system␣
↪under test reacts to a set of performance tests.
```

[2]:
```
#install.packages("RColorBrewer", repos='http://cran.us.r-project.org')
#install.packages("gridExtra")
#install.packages("getPass")
#install.packages("RPostgreSQL")

library("RColorBrewer")
library(ggplot2)
library(gridExtra)
library(getPass)
library(RPostgreSQL)
library(dplyr)
library(stringr)
```

```
Loading required package: DBI


Attaching package: ‘dplyr’


The following object is masked from ‘package:gridExtra’:

    combine


The following objects are masked from ‘package:stats’:

    filter, lag


The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union
```

[3]:
```
db_connection <- DBI::dbConnect(dbDriver(drvName = "PostgreSQL"), dbname =␣
 ↪"pptam", host="db", port="5432", user="postgres", password="postgres")
dbGetQuery(db_connection, "SELECT id::text, name FROM projects")
```

A data.frame: 2 × 2

	id <chr>	name <chr>
1	270ef577-5fbf-4200-a61c-46b6afebc74b	Demo Project
2	275b9bae-076d-446b-90a7-82328241d6b5	todolist

[4]:
```
# Define the name of the project to analyze
project_name <- "todolist"

project_id = dbGetQuery(db_connection, str_glue("SELECT id::text FROM projects␣
 ↪WHERE name='{project}'", project = project_name))$id
```

[5]:
```
sql_operational_profile = "
    SELECT users, frequency FROM operational_profile_observations
        WHERE operational_profile = (SELECT id FROM operational_profiles WHERE␣
 ↪project = ?project)"
operational_profile <- dbGetQuery(db_connection, sqlInterpolate(db_connection,␣
 ↪sql_operational_profile, project = project_id))
```

[6]:
```
plot_width = 6
plot_height = 4
plot_font_size = 13

sql_all_data = "
    SELECT tests.name::text AS test_id, test_sets.name::text AS test_set_id,␣
 ↪test_properties.value::numeric AS users, metrics.abbreviation AS metric,␣
 ↪items.name AS item_name, results.value AS item_value
        FROM results
        INNER JOIN tests ON results.test = tests.id
        INNER JOIN items ON results.item = items.id
        INNER JOIN test_properties ON (test_properties.test = tests.id AND␣
 ↪test_properties.name = 'load')
        INNER JOIN metrics ON results.metric = metrics.id
        INNER JOIN test_set_tests ON (test_set_tests.test = tests.id)
        INNER JOIN test_sets ON (test_sets.id = test_set_tests.test_set AND␣
 ↪test_sets.project = tests.project)
        WHERE tests.project = ?project AND metrics.abbreviation IN ('art',␣
 ↪'sdrt', 'mix', 'fc', 'rc')"
all_data = dbGetQuery(db_connection, sqlInterpolate(db_connection,␣
 ↪sql_all_data, project = project_id))


list_of_microservices = as.data.frame(unique(all_data[,5]))
no_of_microservices = nrow(list_of_microservices)
```

```
test_users_metric<-unique(all_data[,c(1:4)])


all_data$Endpoint = all_data$item_name
#all_data
plot_data <- all_data[all_data$metric == 'art',]
y_max = max(plot_data$item_value)
art_scaled = ggplot(data=plot_data[plot_data$test_set_id == 'scaled-cutoff',],␣
 ↪aes(x = users, y = item_value, color=Endpoint)) + geom_line(aes(group =␣
 ↪Endpoint)) + ylim(c(0,y_max)) + ylab('avg response time (ms)') +␣
 ↪geom_point() + labs(title = 'Scaled Cutoff - Average Response Time') +␣
 ↪theme_bw() + theme(text = element_text(size = plot_font_size))
art_fixed = ggplot(data=plot_data[plot_data$test_set_id == 'fixed-cutoff',],␣
 ↪aes(x = users, y = item_value, color=Endpoint)) + geom_line(aes(group =␣
 ↪Endpoint)) + ylim(c(0,y_max)) + ylab('avg response time (ms)') +␣
 ↪geom_point() + labs(title = 'Fixed Cutoff - Average Response Time') +␣
 ↪theme_bw() + theme(text = element_text(size = plot_font_size))
ggsave(plot=art_scaled, filename="art_scaled.png", width=plot_width,␣
 ↪height=plot_height)
ggsave(plot=art_fixed, filename="art_fixed.png", width=plot_width,␣
 ↪height=plot_height)

plot_data <- all_data[all_data$metric == 'sdrt',]
y_max = max(plot_data$item_value)
sdrt_scaled = ggplot(data=plot_data[plot_data$test_set_id == 'scaled-cutoff',],␣
 ↪aes(x = users, y = item_value, color=Endpoint)) + geom_line(aes(group =␣
 ↪Endpoint)) + ylim(c(0,y_max)) + ylab('sd response time (ms)') + geom_point()␣
 ↪+ labs(title = 'Scaled Cutoff - SD Response Time')+ theme_bw() + theme(text␣
 ↪= element_text(size = plot_font_size))
sdrt_fixed = ggplot(data=plot_data[plot_data$test_set_id == 'fixed-cutoff',],␣
 ↪aes(x = users, y = item_value, color=Endpoint)) + geom_line(aes(group =␣
 ↪Endpoint)) + ylim(c(0,y_max)) + ylab('sd response time (ms)') + geom_point()␣
 ↪+ labs(title = 'Fixed Cutoff - SD Response Time')+ theme_bw() + theme(text =␣
 ↪element_text(size = plot_font_size))
ggsave(plot=sdrt_scaled, filename="sdrt_scaled.png", width=plot_width,␣
 ↪height=plot_height)
ggsave(plot=sdrt_fixed, filename="sdrt_fixed.png", width=plot_width,␣
 ↪height=plot_height)

for(i in unique(all_data$test_id)) {
    for(j in unique(all_data$item_name)) {
        cur_set <- all_data[all_data$test_id == i & all_data$item_name == j,]
        fc <- cur_set[cur_set$metric == 'fc',]
        rc <- cur_set[cur_set$metric == 'rc',]
        fr <- fc$item_value / rc$item_value
        fc$metric = 'fr'
        fc$item_value = fr
        all_data = rbind(all_data, fc)
    }
}
#all_data

plot_data <- all_data[all_data$metric == 'fr',]
y_max = max(plot_data$item_value)
fr_scaled = ggplot(data=plot_data[plot_data$test_set_id == 'scaled-cutoff',],␣
 ↪aes(x = users, y = item_value, color=Endpoint)) + geom_line(aes(group =␣
 ↪Endpoint)) + ylim(c(0,y_max)) + ylab('failure rate') + geom_point() +␣
 ↪labs(title = 'Scaled Cutoff - Failure Rate') + theme_bw() + theme(text =␣
 ↪element_text(size = plot_font_size))
fr_fixed = ggplot(data=plot_data[plot_data$test_set_id == 'fixed-cutoff',],␣
 ↪aes(x = users, y = item_value, color=Endpoint)) + geom_line(aes(group =␣
 ↪Endpoint)) + ylim(c(0,y_max)) + ylab('failure rate') + geom_point() +␣
 ↪labs(title = 'Fixed Cutoff - Failure Rate') + theme_bw() + theme(text =␣
 ↪element_text(size = plot_font_size))
ggsave(plot=fr_scaled, filename="fr_scaled.png", width=plot_width,␣
 ↪height=plot_height)
ggsave(plot=fr_fixed, filename="fr_fixed.png", width=plot_width,␣
 ↪height=plot_height)

all_data[all_data$users == 1 & all_data$metric == 'fc',]
```

A data.frame: 5 × 7

	test_id <chr>	test_set_id <chr>	users <dbl>	metric <chr>	item_name <chr>	item_ <dbl
252	todolist-scaled-cutoff-001	scaled-cutoff	1	fc	ToDo-Create	0
257	todolist-scaled-cutoff-001	scaled-cutoff	1	fc	ToDo-Delete	0
262	todolist-scaled-cutoff-001	scaled-cutoff	1	fc	ToDo-Get-All	0
267	todolist-scaled-cutoff-001	scaled-cutoff	1	fc	ToDo-Get-Single	0
272	todolist-scaled-cutoff-001	scaled-cutoff	1	fc	ToDo-Update	0

[7]:

```
test_users_metric[list_of_microservices[,1]]<-NA
```

[8]:

```
#If the tests occur too fast, it might be that some services have no data. This␣
 ↪case is not handled, yet.

for (i in 1:nrow(test_users_metric)) {
    search_test_id <- test_users_metric[i,1]
    search_metric <- test_users_metric[i,4]

    for (j in 1:no_of_microservices) {
        search_microservice <- list_of_microservices[j,]
        row <- filter(all_data, test_id == search_test_id & metric == search_metric & item_name == search_microservice)

        if (dim(row)[1] > 0) {
            found_value = row$item_value

            if (length(found_value) == 1) {
                test_users_metric[i,j+4] <- found_value
            }
        }
    }
}
raw_data <- test_users_metric
raw_data
```

A data.frame: 105 × 9

	test_id	test_set_id	users	metric	ToDo-Create	ToDo-I
	<chr>	<chr>	<dbl>	<chr>	<dbl>	<dbl>
1	todolist-fixed-cutoff-010	fixed-cutoff	10	rc	116.0000000	65.0000
2	todolist-fixed-cutoff-010	fixed-cutoff	10	fc	0.0000000	0.00000
3	todolist-fixed-cutoff-010	fixed-cutoff	10	art	350.9913793	274.661
4	todolist-fixed-cutoff-010	fixed-cutoff	10	sdrt	44.4444444	39.2592
5	todolist-fixed-cutoff-010	fixed-cutoff	10	mix	0.1608877	0.09015
26	todolist-fixed-cutoff-020	fixed-cutoff	20	rc	240.0000000	123.000
27	todolist-fixed-cutoff-020	fixed-cutoff	20	fc	0.0000000	0.00000
28	todolist-fixed-cutoff-020	fixed-cutoff	20	art	150.3750000	183.211
29	todolist-fixed-cutoff-020	fixed-cutoff	20	sdrt	26.6666667	36.6666
30	todolist-fixed-cutoff-020	fixed-cutoff	20	mix	0.1644962	0.08430
51	todolist-fixed-cutoff-030	fixed-cutoff	30	rc	355.0000000	184.000
52	todolist-fixed-cutoff-030	fixed-cutoff	30	fc	0.0000000	0.00000
53	todolist-fixed-cutoff-030	fixed-cutoff	30	art	143.2028169	151.065
54	todolist-fixed-cutoff-030	fixed-cutoff	30	sdrt	32.2222222	35.9259
55	todolist-fixed-cutoff-030	fixed-cutoff	30	mix	0.1609977	0.08344
76	todolist-fixed-cutoff-040	fixed-cutoff	40	rc	467.0000000	247.000
77	todolist-fixed-cutoff-040	fixed-cutoff	40	fc	0.0000000	1.00000
78	todolist-fixed-cutoff-040	fixed-cutoff	40	art	246.0342612	268.558
79	todolist-fixed-cutoff-040	fixed-cutoff	40	sdrt	64.4444444	68.5185
80	todolist-fixed-cutoff-040	fixed-cutoff	40	mix	0.1619279	0.08564
101	todolist-fixed-cutoff-050	fixed-cutoff	50	rc	588.0000000	312.000
102	todolist-fixed-cutoff-050	fixed-cutoff	50	fc	5.0000000	5.00000
103	todolist-fixed-cutoff-050	fixed-cutoff	50	art	295.2517007	297.182
104	todolist-fixed-cutoff-050	fixed-cutoff	50	sdrt	75.1851852	63.1481
105	todolist-fixed-cutoff-050	fixed-cutoff	50	mix	0.1608755	0.08536
126	todolist-fixed-cutoff-060	fixed-cutoff	60	rc	693.0000000	367.000
127	todolist-fixed-cutoff-060	fixed-cutoff	60	fc	22.0000000	19.0000
128	todolist-fixed-cutoff-060	fixed-cutoff	60	art	530.5685426	485.138
129	todolist-fixed-cutoff-060	fixed-cutoff	60	sdrt	124.4444444	132.592
130	todolist-fixed-cutoff-060	fixed-cutoff	60	mix	0.1618025	0.08568
376	todolist-scaled-cutoff-050	scaled-cutoff	50	rc	588.0000000	315.000
377	todolist-scaled-cutoff-050	scaled-cutoff	50	fc	0.0000000	0.00000
378	todolist-scaled-cutoff-050	scaled-cutoff	50	art	205.8707483	186.209
379	todolist-scaled-cutoff-050	scaled-cutoff	50	sdrt	51.2962963	42.9629
380	todolist-scaled-cutoff-050	scaled-cutoff	50	mix	0.1614498	0.08649
401	todolist-scaled-cutoff-060	scaled-cutoff	60	rc	705.0000000	377.000
402	todolist-scaled-cutoff-060	scaled-cutoff	60	fc	0.0000000	0.00000
403	todolist-scaled-cutoff-060	scaled-cutoff	60	art	167.8695035	157.488
404	todolist-scaled-cutoff-060	scaled-cutoff	60	sdrt	38.5185185	38.5185
405	todolist-scaled-cutoff-060	scaled-cutoff	60	mix	0.1612166	0.08621
426	todolist-scaled-cutoff-070	scaled-cutoff	70	rc	826.0000000	457.000
427	todolist-scaled-cutoff-070	scaled-cutoff	70	fc	2.0000000	4.00000
428	todolist-scaled-cutoff-070	scaled-cutoff	70	art	199.0605327	198.925
429	todolist-scaled-cutoff-070	scaled-cutoff	70	sdrt	41.4814815	45.9259
430	todolist-scaled-cutoff-070	scaled-cutoff	70	mix	0.1627586	0.09004
451	todolist-scaled-cutoff-080	scaled-cutoff	80	rc	923.0000000	507.000
452	todolist-scaled-cutoff-080	scaled-cutoff	80	fc	7.0000000	6.00000
453	todolist-scaled-cutoff-080	scaled-cutoff	80	art	249.1614301	223.824
454	todolist-scaled-cutoff-080	scaled-cutoff	80	sdrt	41.8518519	44.8148
455	todolist-scaled-cutoff-080	scaled-cutoff	80	mix	0.1605217	0.08817

[9]:

```
tests <- unique(raw_data[,1:3])

max_no_of_users <- max(raw_data[,3])
min_no_of_users <- min(raw_data[,3])

user_load <- operational_profile[,1]
user_load
access_count <- operational_profile[,2]
max_no_of_requests <- max(user_load)
scale_factor <- max_no_of_users/max_no_of_requests
scaled_user_load <- floor(scale_factor * user_load)
# Due to different profile, both are supposed to be the same
scaled_user_load <- user_load
scaled_user_load
```

```
1. 0 2. 1 3. 2 4. 3 5. 4 6. 5 7. 10 8. 11 9. 12 10. 13 11. 15 12. 16 13. 17 14. 18 15. 19 16. 20 17. 21
18. 22 19. 23 20. 24 21. 25 22. 26 23. 27 24. 28 25. 30 26. 35 27. 40 28. 42 29. 45 30. 50 31. 55 32. 60
33. 65 34. 70 35. 75 36. 80 37. 105 38. 85 39. 90 40. 95 41. 100
```

```
1. 0 2. 1 3. 2 4. 3 5. 4 6. 5 7. 10 8. 11 9. 12 10. 13 11. 15 12. 16 13. 17 14. 18 15. 19 16. 20 17. 21
18. 22 19. 23 20. 24 21. 25 22. 26 23. 27 24. 28 25. 30 26. 35 27. 40 28. 42 29. 45 30. 50 31. 55 32. 60
33. 65 34. 70 35. 75 36. 80 37. 105 38. 85 39. 90 40. 95 41. 100
```

[10]:

```
##Create aggregate values (by fifty) of the user frequency from␣
 ↪"operational_profile"
steps <- 10

# calculate_aggregated_values <- function() {
    access_frequency <- access_count/sum(access_count)
access_frequency
    by_fifty <- which((scaled_user_load %% steps) == 0)
by_fifty
    no_of_aggregated_rows = length(by_fifty)

    binProb <- c()
    for (i in 1:no_of_aggregated_rows) {
        if (i==1) {
            binProb[i] <- sum(access_frequency[1:by_fifty[i]])
        } else {
            binProb[i] <- sum(access_frequency[(by_fifty[i-1]+1):by_fifty[i]])
        }
    }

aggregated_values_from_operational_profile <-␣
 ↪matrix(c(scaled_user_load[by_fifty], binProb), ncol=2,␣
 ↪nrow=no_of_aggregated_rows, dimnames=list(c(1:no_of_aggregated_rows),␣
 ↪c("Workload (number of users)", "Domain metric per workload")))
# }
# aggregated_values_from_operational_profile <- calculate_aggregated_values()
aggregated_values_from_operational_profile
```

```
1. 0.00793650793650794 2. 0.0291005291005291 3. 0.0158730158730159 4. 0.0158730158730159
5. 0.0158730158730159 6. 0.0211640211640212 7. 0.00529100529100529 8. 0.0105820105820106
9. 0.00529100529100529 10. 0.00529100529100529 11. 0.0105820105820106 12. 0.0105820105820106
13. 0.0211640211640212 14. 0.0185185185185185 15. 0.0185185185185185 16. 0.0185185185185185
17. 0.0185185185185185 18. 0.0158730158730159 19. 0.0132275132275132 20. 0.00264550264550265
21. 0.0105820105820106 22. 0.00793650793650794 23. 0.00793650793650794 24. 0.0105820105820106
25. 0.0264550264550265 26. 0.0343915343915344 27. 0.0555555555555556 28. 0.0264550264550265
29. 0.0529100529100529 30. 0.0317460317460317 31. 0.0502645502645503 32. 0.0661375661375661
33. 0.0740740740740741 34. 0.0687830687830688 35. 0.0476190476190476 36. 0.0423280423280423
37. 0.00264550264550265 38. 0.0449735449735450 39. 0.0396825396825397 40. 0.0158730158730159
41. 0.00264550264550265
```

```
1. 1 2. 7 3. 16 4. 25 5. 27 6. 30 7. 32 8. 34 9. 36 10. 39 11. 41
```

A matrix: 11 × 2 of type dbl

	Workload (number of users)	Domain metric per workload
1	0	0.007936508
2	10	0.103174603
3	20	0.119047619
4	30	0.113756614
5	40	0.089947090
6	50	0.111111111
7	60	0.116402116
8	70	0.142857143
9	80	0.089947090
10	90	0.087301587
11	100	0.018518519

[11]:

```
#Define the threshold for each service. The threshold is a vector computed as␣
 ↪avg+3*SD for the configuration with
#Users=2, Memory=4, CPU=1, CartReplica=1

data_of_min_user<-raw_data[raw_data$users==min_no_of_users,]
test_of_min_user<-tests[tests$users==min_no_of_users,]

avg <-data_of_min_user[data_of_min_user$metric=="art",][,-c(1:4)]
sd <- data_of_min_user[data_of_min_user$metric=="sdrt",][,-c(1:4)]
threshold<-data.frame(test_of_min_user,avg+3*sd)

#Check the first line of the dataframe thereshold: it must be one line
head(threshold)
data_of_min_user
```

A data.frame: 1 × 8

	test_id <chr>	test_set_id <chr>	users <dbl>	ToDo.Create <dbl>	ToDo.Delete <dbl>	ToD <db
251	todolist-scaled-cutoff-001	scaled-cutoff	1	299.0278	239.5	287.

A data.frame: 5 × 9

	test_id <chr>	test_set_id <chr>	users <dbl>	metric <chr>	ToDo-Create <dbl>	ToDo-Del <dbl>
251	todolist-scaled-cutoff-001	scaled-cutoff	1	rc	12.0000000	6.0000000
252	todolist-scaled-cutoff-001	scaled-cutoff	1	fc	0.0000000	0.0000000
253	todolist-scaled-cutoff-001	scaled-cutoff	1	art	197.9166667	191.16666
254	todolist-scaled-cutoff-001	scaled-cutoff	1	sdrt	33.7037037	16.111111
255	todolist-scaled-cutoff-001	scaled-cutoff	1	mix	0.1666667	0.0833333

[12]:

```r
#Exclude case with user = 2 from dataFile and check whether each service passes or fail: avg<threshol (Pass).
#Compute the relative mass for each configuration

tests_without_benchmark<-tests[!tests$users==min_no_of_users,]
raw_data_without_benchmark<-raw_data[!raw_data$users==min_no_of_users,]

avg<-raw_data_without_benchmark[raw_data_without_benchmark$metric=="art",-4]
sd<-raw_data_without_benchmark[raw_data_without_benchmark$metric=="sdrt",-4]
mix<-raw_data_without_benchmark[raw_data_without_benchmark$metric=="mix",-4]

#Check pass/fail for each service. the "mix" value is 0 if fail and mixTemp if pass. Compute the relative mass for each configuration
pass_criteria<-avg

calculate_relative_mass <- function() {
    relative_mass<-c()

    mix_of_passing_tests<-as.data.frame(matrix(nrow=nrow(tests_without_benchmark), ncol=ncol(raw_data_without_benchmark)-1))

    for(j in 1:nrow(pass_criteria)){
        mix_of_passing_tests[j,]<-mix[j,]
        for(i in 3:(2+no_of_microservices)){
            if(pass_criteria[j,i]>threshold[i]){
                mix_of_passing_tests[j,i]<-0
            }
        }
        relative_mass[j]<-sum(mix_of_passing_tests[j,3:(2+no_of_microservices)])
    }

    relative_mass
}

relative_mass <- calculate_relative_mass()

#Show first lines of passCriteria
head(pass_criteria)
```

A data.frame: 6 × 8

	test_id <chr>	test_set_id <chr>	users <dbl>	ToDo-Create <dbl>	ToDo-Delete <dbl>	ToDo- <dbl
3	todolist-fixed-cutoff-010	fixed-cutoff	10	350.9914	274.6615	243.15
28	todolist-fixed-cutoff-020	fixed-cutoff	20	150.3750	183.2114	182.78
53	todolist-fixed-cutoff-030	fixed-cutoff	30	143.2028	151.0652	164.51
78	todolist-fixed-cutoff-040	fixed-cutoff	40	246.0343	268.5587	265.17
103	todolist-fixed-cutoff-050	fixed-cutoff	50	295.2517	297.1827	212.72
128	todolist-fixed-cutoff-060	fixed-cutoff	60	530.5685	485.1390	346.17

[13]:

```
#Compute the domain metric for each configuration
tests_without_benchmark$relative_mass<-relative_mass

absolute_mass<-c()
for(j in 1:nrow(tests_without_benchmark)) {
   ␣
 ↪absolute_mass[j]<-tests_without_benchmark[j,"relative_mass"]*aggregated_values_from_operati
 ↪aggregated_values_from_operational_profile[,1]),2]
}
tests_without_benchmark$absolute_mass<-absolute_mass

test_sets<-as.data.frame(unique(all_data[,2]))
colnames(test_sets)[1] <- "test_set_id"

set<-list()
domain_metric_list<-list()
for(i in 1:nrow(test_sets)){
    set[[i]]<-tests_without_benchmark[which(tests_without_benchmark[,2] ==␣
 ↪test_sets[i,1]),]
    domain_metric_list[[i]]<-set[[i]][,c(3,5)][order(set[[i]][,c(3,5)][,1]),]
}

#Uncomment this to show first lines of domain_metric_list
#head(domain_metric_list)
domain_metric_list
```

1. A data.frame: 10 × 2

	users <dbl>	absolute_mass <dbl>
1	10	0.04965546
26	20	0.08755181
51	30	0.08285402
76	40	0.05838452
101	50	0.07174343
126	60	0.03772266
151	70	0.00000000
176	80	0.00000000
201	90	0.00000000
226	100	0.00000000

2. A data.frame: 10 × 2

	users <dbl>	absolute_mass <dbl>
276	10	0.07660361
301	20	0.08788204
326	30	0.08345963
351	40	0.06620377
376	50	0.08170114
401	60	0.08544496
426	70	0.10561576
451	80	0.05127766
476	90	0.02181189
501	100	0.00300867

[14]:
```
#Compute Cumulative Domain metric: summing up absoluteMass over users for each␣
 ↪configuration
test_sets$domain_metric<-0
for(i in 1:nrow(test_sets)){
   ␣
 ↪test_sets[i,2]<-round(sum(tests_without_benchmark[which(tests_without_benchmark[,2]␣
 ↪== test_sets[i,1]),"absolute_mass"]),4)
}
domain_metric<-test_sets

domain_metric
```

A data.frame: 2 × 2

test_set_id <chr>	domain_metric <dbl>
fixed-cutoff	0.3879
scaled-cutoff	0.6630

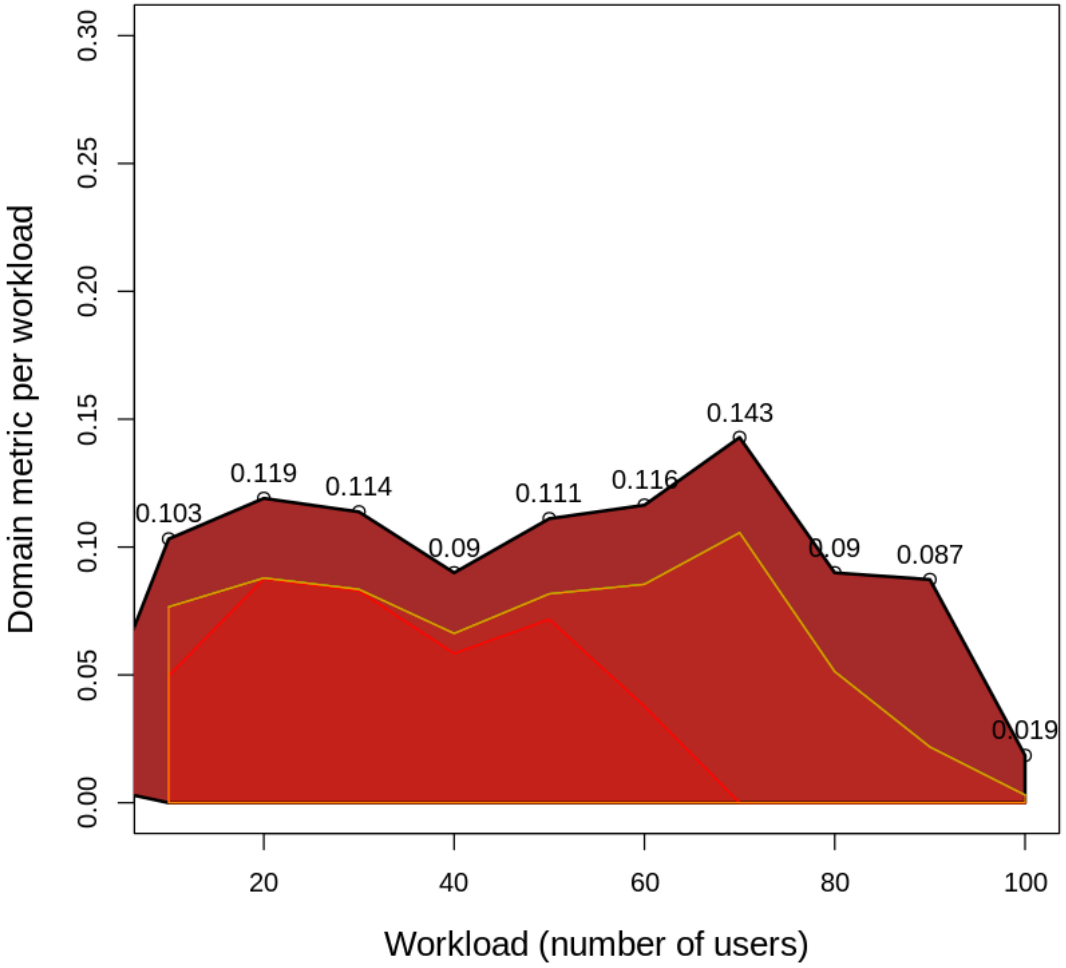
[15]:
```
#Plot operational_profile against domain metric for each configuration

plot(aggregated_values_from_operational_profile, xlim=c(steps,␣
 ↪max_no_of_users), ylim=c(0, 0.3),cex.lab=1.3)
polygon(c(steps,aggregated_values_from_operational_profile[,1],max_no_of_users),c(0,aggregated
 ↪col="brown", lty = 1, lwd = 2, border = "black")
color=heat.colors(11)
color_transparent <- adjustcolor(color, alpha.f = 0.2)

sorted_domain_metric<-domain_metric
k<-which(sorted_domain_metric[,2]==max(sorted_domain_metric[,2]))
#Green line whithin the polygon is the best domain matric line.
#It corresponds to the second line in the final table below

for(i in 1:nrow(test_sets)) {
    lines(domain_metric_list[[i]], type="l", col=heat.colors(11)[i])
    lines(domain_metric_list[[k]], type="l", col="green")
```

```
   ␣
 ↪polygon(c(steps,t(domain_metric_list[[i]][1]),max_no_of_users),c(0,t(domain_metric_list[[i]
 ↪col=color_transparent[i], lty = 1, lwd = 1 , border = rainbow(11)[i])
}

text(aggregated_values_from_operational_profile,labels =␣
 ↪round(aggregated_values_from_operational_profile[,2],3), pos=3, col="black")

graphics.off()
aggregated_values_from_operational_profile
aggregated_values_from_operational_profile[1:6,2]
```

A matrix: 11 × 2 of type dbl

	Workload (number of users)	Domain metric per workload
1	0	0.007936508
2	10	0.103174603
3	20	0.119047619
4	30	0.113756614
5	40	0.089947090
6	50	0.111111111
7	60	0.116402116
8	70	0.142857143
9	80	0.089947090
10	90	0.087301587
11	100	0.018518519

```
1   0.00793650793650794 2   0.103174603174603 3   0.119047619047619 4   0.113756613756614 5
 0.0899470899470899 6                                    0.111111111111111
```



```
[16]: DBI::dbDisconnect(db_connection)
```

TRUE

```
[ ]:
```

```
[ ]:
```