What Were They Thinking:

Modeling Think Times for Performance Testing

Tom Wilson

1 Introduction

When modeling users interacting with a system, the concept of think time will arise. Besides developing a workload model and recording scripts, a think time analysis is necessary in order to have an accurate test and appropriate system throughput.

Think time is the time the user spends between system tasks (i.e., interactions with the system). Example tasks include: selecting from a menu, entering data into a text field, reviewing a list of results before selecting one or more of them, and navigating to a completely new task. These tasks are more than the physical motion required to interact with the system's user interface; they include the thinking time that causes the user to make these choices. What cannot be determined when examining production data is whether or not the user is truly interacting with the system during the time between system tasks; idle times are hidden within think times.

This paper details a start-to-end think time analysis. It extends the analysis that appears in [Wil10a]. The analysis uses real production transaction data to derive think times. After processing these data, the analysis constructs a think time model for use in a load generator. Then, the analysis attempts to refine the model to provide more accurate times for given situations. The more accurate the model, the more confidence there will be in the analysis of the performance test. Finally, an operations analysis is performed as a check.

A proprietary transaction system that supports logistics and maintenance for military equipment provides example data. For the purpose of anonymity, most system details are omitted. We do not need to understand the system in order to analyze the think time data. Ultimately, the think time data will be fed into a performance test for the next generation system.

2 Data Mining

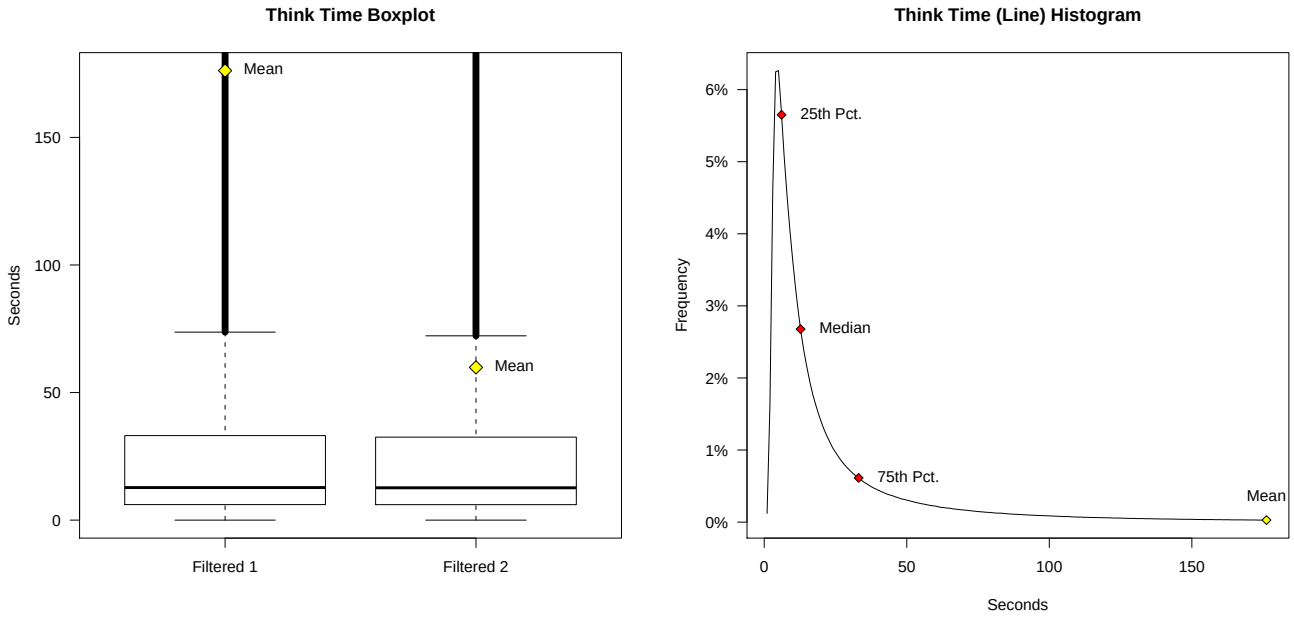
We took 8 months of transaction logs and separated the data according to the users' sessions. A sequence of think times was computed from the sequence of transactions within each session. There are over 13 million think times in the source data (note that this is really too much data). When we examine the computed data, we discover three problems: (1) some of the times are negative or zero, (2) some of the positive times are very close to zero, and (3) some of the times are very large. Negative think times can occur when two or more transactions are generated by the same user action. In such cases, the earlier one finishes after the next one has already started. Negative think times can also occur when a user submits another transaction before the previous has finished. This is often due to a slow response time, but can also be a result of abandoning the previous transaction due to change of mind. Similar situations can lead to times equal to 0. In the case of multiple transactions being generated, we do not need to model this; if the new system does this, it will happen automatically. In the case of abandonment, we are not interested in modeling the user changing his mind. So, in both cases, we can eliminate negative or zero think times. Less than 1% of the values fall into this category.

Very short think times greater than 0 can be a result of finishing transactions triggering subsequent transactions. This situation is similar to the one where the times were negative and are not indicative of user behavior. Short times can also be a result of user anticipation. Because of familiarity with the application, the user does not review the result of the previous transaction and quickly submits the next transaction to move on. We do this when we "manually batch process" (i.e., we do the same thing over and over again). These times are valid. However, we cannot distinguish them from the first situation unless we can recognize which transaction pairs are truly related. Only 0.1% of the times are less than 1 second and greater than 0.

Very large think times may have nothing to do with the user "thinking". Large values are often a result of the user doing work away from the system. Numerous possibilities exist: interacting with a different system or application on this system, talking on the phone, leaving the room, or taking a break. For a performance test where the test

duration is limited, generating a think time longer than the test is pointless. So, we will eliminate times greater than an hour, which is our test interval length, and will not even want to generate any times that large. Only 0.4% of the times are greater than one hour. We will note that the largest value is over 26 days!

Figure 1 shows some statistical charts for the think time data. Figure 1a shows a boxplot for two data sets. The maximum y-value on the chart has been limited so that the boxes are not flattened. The "Filtered 1" column shows the spread of the think times, where the negative and zero times are removed. The "Filtered 2" column shows the spread of think times, where the times over an hour are also removed. The impact of these large times on the mean is significant: There are a small percentage of large times, but they can be very large! The two boxplots do not look very different because the outliers that we removed are off the chart.



(a) **(b)**

Figure 1: Chart (a) shows a boxplot of the think times for both filtered data sets, while chart (b) shows the same data as a (line) histogram. Each chart also shows important statistics such as the median and mean.

Figure 1b shows the distribution of the "Filtered 1" data using a histogram that is drawn with a line rather than bars (because very narrow bins are used). The percentiles that correspond to the boxplot's quartiles are shown, as well as the mean. The distribution for the "Filtered 2" would look almost the same except for the mean's location. In [Wil10a], we concluded that the distribution was lognormal without fitting the data to the distribution. For a more thorough analysis, we will perform the fit.

3 Distribution Fitting

Distribution fitting is a process of selecting the best distribution model from the models constructed. The goal of distribution fitting is to identify the distribution family and the parameters associated with the distribution (i.e., location, scale, and/or shape). For a discussion on distributions and distribution fitting, refer to [Wil10b] (in fact, this same problem is the example used in that paper).

Some tools, such as R ([R D09]), may have facilities to do most of the distribution fitting work. The R package *fitdistrplus* ([DMPDD10]) contains such functions. However, too much data can cause the analysis to be slow.[1] Table 1 shows a statistical summary for the think times for each month as well as the size of each data set. Each data set seems to be similar (although some of the similarity is due to rounding), and each is still inconveniently too large.

So, we selected one weekday of one month and removed the data that was not between 8:00 and 17:00; the remaining interval is representative of the time frame that we want to model. Then we randomly sampled the data to obtain a smaller set. The summary of these data is shown as the "-" entry in Table 1. The summary statistics

[1] Also, when a PDF image file is being created, the file ends up being very large. A fixed resolution format will not have this problem.

Table 1: Think Time Summary by Month

Mo.	**Min**	Q_1	Q_2	Q_3	**Avg**	**Max**	**Size**
3	1.00	6.20	12.96	33.16	66.86	3600	1,304,146
4	1.00	7.20	14.73	35.05	60.16	3598	1,212,268
5	1.00	7.23	14.84	35.88	61.86	3600	1,592,149
6	1.00	6.31	13.35	33.82	59.95	3600	1,874,687
7	1.00	5.72	11.97	31.11	58.06	3600	2,048,770
8	1.00	5.51	11.31	29.83	55.66	3600	959,340
9	1.00	5.63	11.51	30.73	58.64	3600	2,081,692
10	1.00	5.70	11.70	31.39	59.10	3600	2,090,495
-	1.00	5.17	11.84	29.92	48.54	2831	1,585

are "close" to the other data sets. We could increase the sample size to get a higher mean and max, but this is good enough to demonstrate the remainder of the approach.

Figure 2 shows the graphs for the lognormal distribution fit. The QQ-plot and PP-plot are the graphs of interest. The QQ-plot looks like it does not follow the line, but where it deviates from the line is beyond the 95^{th} percentile (an observation not derived from the figure). The PP-plot shows that the distribution is a good fit for the data since the probability points are very close to the diagonal line. We will not bother showing fits for other distributions.

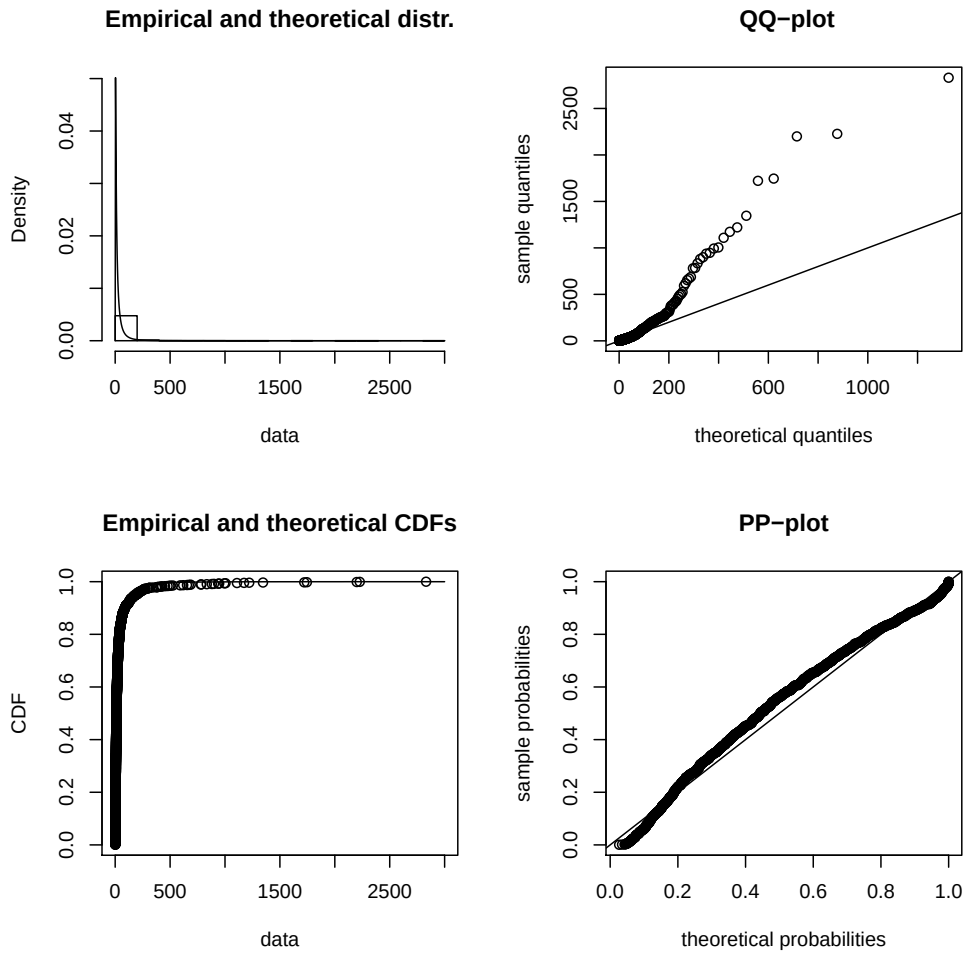


Figure 2: Visual results for the lognormal distribution fitting.

The distribution fitting function can also tell us the location and scale parameters that are associated with a lognormal distribution: meanlog and sdlog. These parameters are the mean and standard deviation of the logarithm

of the data. For our data, the values are: meanlog = 2.659 and sdlog = 1.325. These values can be used in the lognormal distribution's random number generator to generate think times.

4 Think Time Categories

When we apply the think time analysis to our performance test, we encounter two problems: (1) our testing tool only supports a uniform distribution and (2) a random number generated from our distribution might not make sense in many situations. For the first problem, the work-around is to use the random number to index a table of think times after scaling it by the size of the table. The data in the table can conform to any distribution we want. The tables can be updated without any impact to the testing scripts. The accuracy of the distribution can be controlled by the size of the table.

The second problem concerns where large and small times are applied. We would prefer not to see think times of a few minutes between steps in a sequence where a time of only a few seconds would be more typical, nor very small times between unrelated tasks. Blindly using random numbers from our distribution will be correct when the numbers are viewed as a whole, but not when viewed on a situational basis. We could have avoided this problem by having more resolution in the source data (i.e., knowing which category a think time belongs to). In our case, we would be reverse-engineering scripts from the transaction data.

Table 2 lists the think time categories pertinent to the scripts for our system. Understanding the think time types is not critical, but we will give a short description of each. *Tab* is used when the user selects one of several tabs, each leading to a different dialog on the screen. *Menu* is used when the user selects an item from a pull-down menu. *Search Options* is used when the user selects or checks options associated with searching. *View* is used when the user looks at data on various screens besides search results. *Update* is used when the user modifies retrieved data. *Search Results* is used when the user reviews results from a search. *Create* is used when the user enters new data. *Inter-task* is used between iterations of a script.

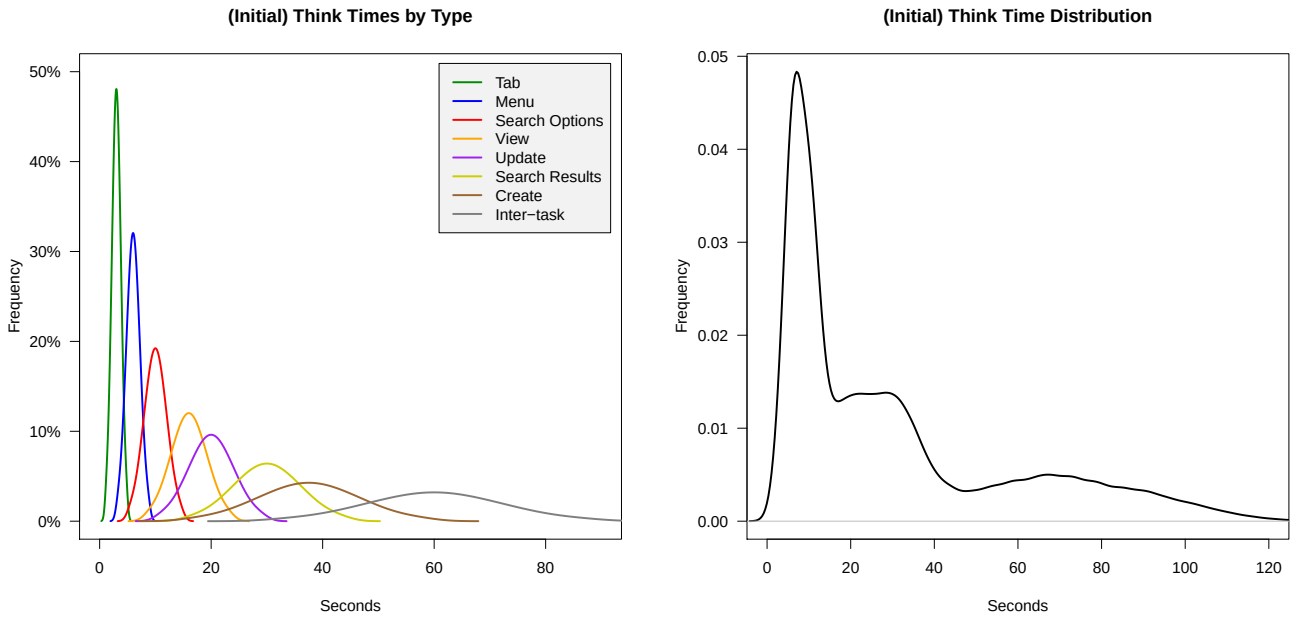
Table 2: Think Time Categories and Ranges (*=Times are in Seconds)

	Initial*		**Final***			**Distr.**	**Est.**
Category	**Min**	**Max**	**Min**	**Mid**	**Max**	**Type**	**Count**
1 - Tab	1	5	1		5	Norm	16,126
2 - Menu	3	9	3		9	Norm	23,194
3 - Search Options	5	15	5		12	Norm	18,968
4 - View	8	24	5		20	Norm	11,601
5 - Update	10	30	10		30	Norm	7,347
6 - Search Results	15	45	10	20	90	Tri	27,309
7 - Create	15	60	15		60	Norm	3,506
8 - Inter-task	30	90	30	120	930	Tri	19,314

We have chosen to associate a range with each action, and base the think times for each action on a normal distribution. The "Initial" values appeared in [Wil10a] and were defined when the script development was still on-going (we will discuss the remainder of the table later in this section). Script 2 in Section 6 describes how the think time tables are generated from these values. The hope is that, when all of the times are collected from a test, a distribution results that compares to the original data. As stated in [Wil10a], this would be difficult to predict without knowing the frequency of each category. Those frequencies need to be computed from the scripts and the workload mix. Figure 3a shows what each category looks like.

After a test is run, think times are computed from the transaction data. We encountered issues in computing think times with the new system similar to those encountered with the current system. There were many cases where two consecutive transactions were generated from one user action. The times between such transactions are small (less than 1 second), and so these times are filtered out (we know this is the case because no think time is less than one second). Figure 3b shows what the resulting think time distribution looks like. The problem with the distribution's appearance is a result of the non-uniform frequencies of each category. So, the category distributions require some refinement. At first, this was done via trial-and-error. Eventually, an improved approach was created.

The "Search Results" category was determined to be the biggest contributor to the bulge in the distribution around 25 seconds. Also, the distribution was not generating large enough values in the "Inter-task" category. These two categories were changed to a triangular distribution so that they were skewed rather than normal. The "Final" columns of Table 2 show the new distribution values as well as a "Distr. Type" column indicating what distribution is used (normal versus triangular).



(a) **(b)**

Figure 3: Chart (a) shows the initial distributions for each think time type. Each distribution is normal with the hope that the accumulation will result in a lognormal distribution. Chart (b) shows the distribution from the performance test run.

Each script has an associated number of users. These numbers are derived from a workload analysis. Each script also has a count for each think time category. Table 3 shows an excerpt of the script data, detailing the number of users and the number of each think time category for each script. The extremes are shown in the excerpt.

Table 3: Script Data (Excerpt)

Script	Users	Think Time Category								Cycle Time	#
		1	2	3	4	5	6	7	8		
Script 001	42	1		1			1		1	411.86	9
Script 002	11	1	1				1		1	413.84	9
Script 003	11	1	1	1	3		1		1	461.29	8
Script 102	15	2	3	3		1	3	2	1	628.98	6
Script 131	108	1	1	1			1		1	433.15	8
Script 132	12	1	1				1		1	410.17	9
Script 133	35				1				1	372.57	10

A particular think time category will occur for each instance in the script for each user executing the script. This amount is further multiplied by the number of iterations of each script. The number of times a script executes is a function of its cycle time (or duration) and the interval over which it is run. The cycle time depends on the details of the script. Our interval is one hour.

Table 4 illustrates how the cycle time (in seconds) is computed. Each script generates a series of transactions. Transactions may be separated by think times. Our scripts place the "Inter-task" think time at the top of the script so that think time is still accounted for should a script encounter an error and start over. This is not important to us here as long as the category occurs in the list. The cycle time estimate consists of both transaction estimates and think time estimates. For the think times, the average for the interval is used.[2]

Each script's cycle time (in seconds) is shown in Table 3. The number of times the script is executed in one hour is also shown (the "#" column). This number is rounded to the nearest integer. The estimated number of times

[2]The average for a triangular distribution is $(min + mid + max)/3$.

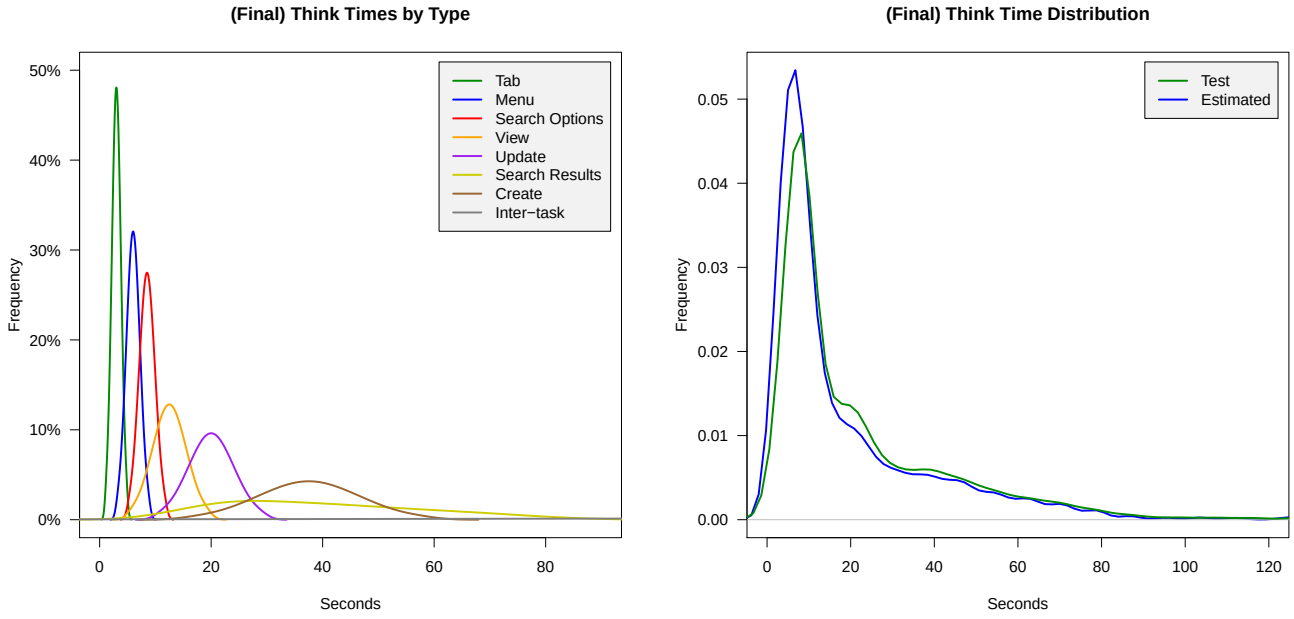
Table 4: Script 1 Cycle Time

Think Time	Transaction	Time
Inter-task		360.00
Tab		3.00
	Transaction 139	0.10
Search Options		8.50
	Transaction 141	0.23
	Transaction 11	0.03
Search Results		40.00
Script 1 Total		411.86

each think time category occurs is shown in Table 2.

The think time category distribution and frequency (i.e., the count) can be used to predict the total think time distribution without running a test. The think time distribution can be estimated by generating random times according to each distribution, using the number of times is estimated in Table 2. The accumulation of all times can be graphed as before. Script 4 in Section 6 shows how simple this task is.

Figure 4a shows what each think time category looks like based on the data in Table 2. Figure 4b shows the predicted and actual test think time distributions. One notable difference is that the test contains think times for the entire run. The times could have been limited to the test interval, but the difference is not significant. Another minor issue is that the think time tables are not large enough. This does not result in a visually identifiable artifact.



(a) **(b)**

Figure 4: Chart (a) shows the final think time distributions for each type. Each distribution is either normal or triangular with the intent that the accumulation will result in a lognormal distribution. Chart (b) shows distributions of the estimate and the actual test.

One flaw in correlating the distributions concerns consecutive think times. Consecutive think times occur most often at the top and bottom of the script. Sometimes a category precedes the "Inter-task" category (located at the bottom of the script) and usually one follows it. The estimate handles them separately while the test combines them. The result is that some small times are combined with some large times. This is probably why the test lacks the expected shape in the peak. Otherwise, the two distributions look very similar

But how does the think time model compare with the real data? If we were to fit the data to a lognormal distribution (like we did Figure 2), the disparity would be noticeable. Our distribution parameters are: meanlog =

3.231 and sdlog = 1.469. Other statistical summary data are in Table 5. The conclusion is basically this: Most of the think time categories need to be moved left and made narrower. The "Inter-task" category needs to generate much larger values occasionally. This will not affect the quartiles and will not affect the average significantly since most of other values will decrease in value after altering the other category distribution values.

Table 5: Think Time Summary for the Test

Min	Q_1	Q_2	Q_3	**Avg**	**Max**	**Size**
1.00	7.97	18.37	53.64	88.24	968.10	182,231

Changing the values is fairly easy and we can quickly estimate the distribution. We conclude that the approach has been demonstrated without refining the values further.

5 Operational Analysis

Operational analysis uses simple equations to assess system performance. It is synonymous with queueing theory. When performing an operational analysis, it is important to know whether a system is open or closed. Real systems are usually open or have properties of both (refer to [SWHB06]), while performance tests tend to be closed. A key relationship in operational analysis is: $N = X * (R + Z)$, where N is the number of users, X is the throughput, R is the response time, and Z is the think time. The equation can be rearranged to solve for any variable when the other three are known. [Gun10] discusses using the think time to vary the number of users in a test.

Table 6 contains all data for all subsequent discussions. A value in the table has been colored blue to denote that it has been measured or red to denote that it has been computed from the other values. For this analysis, we have a one-hour sample from the existing system that is different from data used in earlier sections of this paper. The "Orig" column contains the data of interest. The computed think time Z is much higher than we expect. We need to figure out what is wrong.

Table 6: Operational Analysis Data

Param		**Orig**	**Test**	**Conv**	**Proj1**	**Proj2**
N	Users	801	2,455	257	3,000	3,000
T	Transactions	18,973	158,178	18,973	221,475	178,512
R	Response Time (secs)	1.76	0.50	1.76	0.50	0.50
X	Throughput (transactions/sec)	5.27	43.94	5.27	61.52	49.59
Z	Think Time (secs)	150.22	55.37	47.00	48.26	60.00
	Transactions/User	23.69	64.43	73.82	73.82	59.50
A	Activity Interval (mins)	17.22	53.86	60.00	-	-
O	Observed Think Time (secs)	52.07	50.51	-	-	-

Other parameters include the number of transactions (T), the activity interval (A), and the observed think time (O). The activity interval is the time between the beginning of the first transaction and the end of the last transaction. This is similar to the session duration, but is only equal to it when the entire session falls within the hour interval. The observed think time is the average think time computed from the logged data. We would like Z and O to be nearly equal.

The one hour sample contains 18,973 transactions submitted by 801 unique users. However, not all of these users interacted with the system for the entire hour like the users would in a performance test. So, we need to figure out how many equivalent users there are.

Figure 5 shows a plot of the users' transactions during the hour interval. The users are ordered by the start times of the first transactions. It is clear that most users are not interacting during the whole interval. What is not so obvious is when some users' last transactions are. If we ordered the users by the latest end times, a similar (but horizontally-flipped) shape would exist. Further analysis reveals that the average activity interval A is only 17.22 minutes. For a performance test, the users should interact for most of the hour. Perhaps the number of users should be scaled by 17.22/60.

We decided to compute the average activity interval for one of our tests. The "Test" column shows the relevant data. We need to note that the test is performed on the new system, which has a lower response time. The activity interval is 53.86, and we know that most of our users interact with the system on both sides of the hour-long sampling

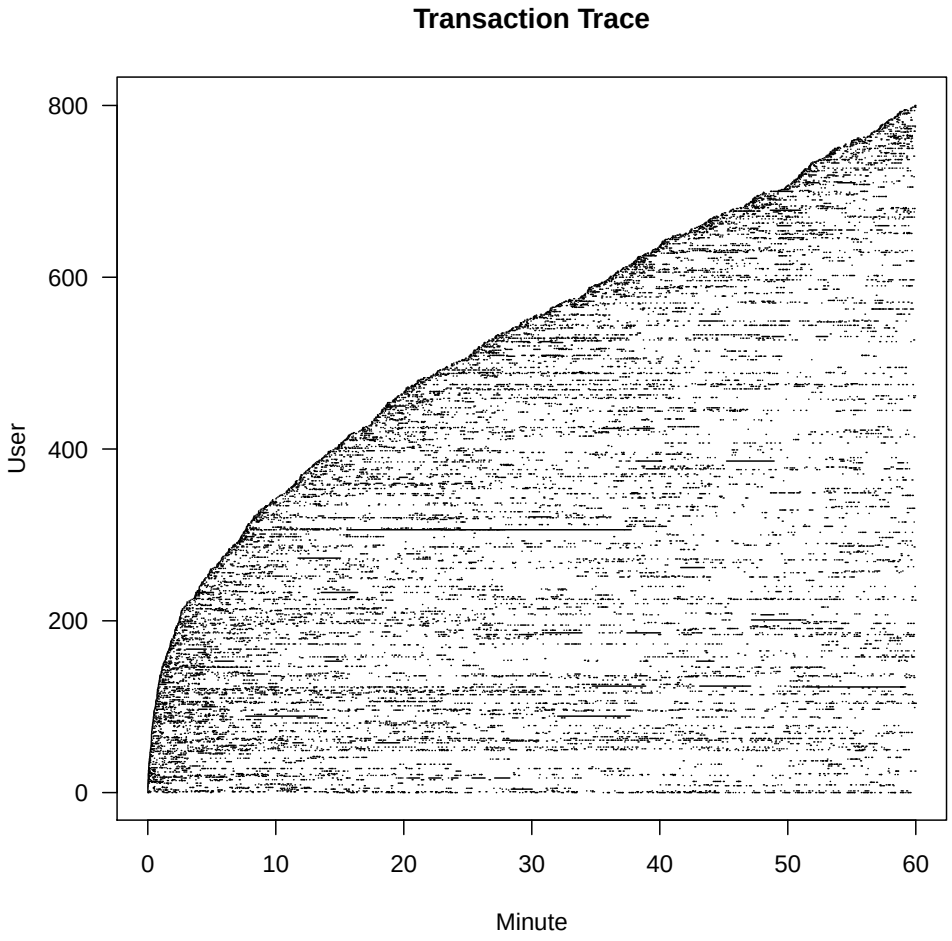


Figure 5: This chart plots each user's activities during the hour interval. The users are ordered by the start of their first transactions within the interval. What is not so obvious is when some users' last transactions are.

interval. This means that there are think times preceding the first transaction and following the last transaction for each user. This is ok. We just have to assume the same situation applies to some of the users in the original sample.

So, we decided to scale the number of original users by 17.22/53.86. There are 257 effective users. The "Conv" column shows the new data and results. This number of users gives a better think time estimate, although it is still a little low. What we are also interested in is the computed transactions/user.

The new system must support a larger user base. So, we created two projections for 3000 users. Whether or not the same response time can be achieved is not the goal of this analysis; the performance test will determine that. This analysis is focused on a proper think time model. For the first projection, we want the same transactions/user as we think we are seeing in the existing system (in the "Conv" column). This gives us an estimated think time that is comparable to the previous value. For the second projection, we want the think time to be 60 seconds because this is approximately what we observed in the earlier analyses.

The operational analysis helps us better understand what is actually happening in production so that we can have better confidence in our testing. Blindly using the transactions/user data from the "Orig" column of Table 6 to construct a load results in a load that is too light.

6 R Scripts

Some of the described work sounds more complicated than it really is, so we thought we would include some of the R scripts that were used. [Wil10a] contains a script similar to the one that creates Figure 1. It will not be repeated here.

Script 1 illustrates how simple the distribution fitting task is with the *fitdistrplus* R package. The package is not a standard one, so it must be downloaded and installed. It contains additional functions to compute goodness-of-fit measures, but we did not leverage these. In [Wil10b], the data were also fitted against an exponential distribution. We did not illustrate that fit here. In order to fit against the exponential distribution, replace *lnorm* with *exp*.

Script 1 Lognormal Distribution Fitting

```
library(fitdistrplus) # Load the fitdistrplus package
w = read.csv("think_time_sample.csv") # Read the data
# fit the data to a lognormal distribution
f1 = fitdist(w$Duration, "lnorm")
pdf("fit_lognormal.pdf") # create a PDF file for the graph
plot(f1) # plot the fit -- there are actually 4 graphs
dev.off() # close the file
f1 # print meanlog and sdlog
```

Script 2 generates the (initial) think time tables. The script creates 10 intervals: $(0.0, 0.1), (0.1, 0.2), \ldots, (0.9, 1.0)$ with midpoints in each interval. The *gen_norm_vals* function determines how many points would fall at the midpoint using the *dnorm* (normal distribution density) function. This number is scaled to an integer such that the counts for all intervals sum to 100 (the table size). Within each interval, uniformly spaced points are created (excluding the end points). Even though we have specified a normal distribution, our data are bounded by [0.05,0.95]. So, we do not have to worry about extreme values that are generated by this unbounded distribution.

The *gen_range* function scales the distribution values to an interval bounded by *min* and *max*. The result is a table with a normal distribution. A table is created for each think time category, and all tables are written out to a CSV (comma-separated values) file.

Script 3 is the remainder of the table generator script that creates the "Final" values. Function *gen_tri_vals* is similar to *gen_norm_vals* except that a triangular distribution is defined. An R package called *mc2d* contains triangular distribution functions. However, we encountered problems installing a dependent package and were not able to use the package. Therefore, the *gen_tri_vals* is a little longer and more complicated than necessary. The remainder of the script is similar to the end of Script 2.

Script 4 estimates the overall think time distribution based upon the think time tables and the counts for each category. The script reads the tables and defines the category counts (these values are the same as those in Table 2 and could have been stored in a file in order to avoid editing the script). Then it generates a sequence of random numbers with a length equal to the category's count. These numbers are scaled to table indices and used to index the category's table. All of the times are accumulated and plotted as a density curve like in Figure 4b. The only things omitted from this script are commands to read and plot the test data (the green line in Figure 4b) and to create the legend.

7 Conclusions

In this paper, we have performed a complete think time analysis. We started with production data from a real system and created a think time model for a new system. We added a level of detail to the model by defining think time submodels for several categories of think times. Then we tuned the submodels to produce the combined think time distribution that we expect to see. An operations analysis was performed as a check and this clarified some misconceptions about the production environment.

References

[DMPDD10] Marie Laure Delignette-Muller, Regis Pouillot, Jean-Baptiste Denis, and Christophe Dutang. *fitdistrplus*, 2010. R package version 0.1-3.

[Gun10] Neil Gunther. "Using Think Times to Determine Arrival Rates". `http://perfdynamics.blogspot.com/2010/05/using-think-times-to-determine-arrival.html`, 2010.

[R D09] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009.

Script 2 Think Time Table Generator

```
table_size = 100 # The table_size value should be much larger for an accurate table.
# define sampling points along a [0,1] interval
mins = seq(from=0, to=0.9, by=0.1)
mids = mins + 0.05
maxs = mins + 0.1
# gen_norm_vals generates a vector of 100 values that follow a normal distr.
# for every sampling interval, the number of points that fall in that interval
# are computed. then the interval is divided uniformly for that number of points
gen_norm_vals = function(mean, sd)
{
  counts = dnorm(mids, mean=mean, sd=sd) # compute freqs. for a normal curve
  counts = round(counts * table_size / sum(counts)) # normalize and scale by 100
  vals = numeric()
  for (x in 1:10)
  { # compute count[x] points between mins[x] and maxs[x] (exclusive)
    tmp = seq(from=mins[x], to=maxs[x], length.out=counts[x] + 2)
    vals = append(vals, tmp[2:(counts[x] + 1)]) # append to output vector
  }
  vals # return vector
}
# gen_range scales the vals vector by the specified interval
gen_range = function(min_value, max_value, vals)
{
  len = max_value - min_value
  x = round(min_value + vals * len, digits=3)
  x
}
norm_vals = gen_norm_vals(0.5, 0.2) # create a normal distribution vector
d = data.frame(Tab            = gen_range(1, 5, norm_vals),
               Menu           = gen_range(3, 9, norm_vals),
               Search_Options = gen_range(5, 15, norm_vals),
               View           = gen_range(8, 24, norm_vals),
               Update         = gen_range(10, 30, norm_vals),
               Search_Results = gen_range(15, 45, norm_vals),
               Create         = gen_range(15, 60, norm_vals),
               Inter_Task     = gen_range(30, 90, norm_vals))
write.csv(d, file="think_time_tables_initial.csv", row.names=FALSE)
```


[SWHB06] Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. "Open Versus Closed: A Cautionary Tale". In *Networked Systems Design and Implementation '06*, pages 239–252, 2006.

[Wil10a] Tom Wilson. "Data Mining User Behavior". *CMG MeasureIT*, September 2010.

[Wil10b] Tom Wilson. "Statistics for the Performance Analyst". *CMG MeasureIT*, November 2010.

Script 3 Think Time Table Generator (Continued)

```
# gen_tri_vals generates a vector of 100 values that follow a triangular distr.
# for every sampling interval, the number of points that fall in that interval
# are computed. then the interval is divided uniformly for that number of points
gen_tri_vals = function(a, c, b)
{
  len = b - a
  x = a + mids * len
  counts = numeric(length(mids)) # compute freqs. for a triangular distribution
  counts[x < c] = 2 * (x[x < c] - a) / ((b - a) * (c - a))
  counts[x >= c] = 2 * (b - x[x >= c]) / ((b - a) * (b - c))
  counts = round(counts * 100 / sum(counts) + 0.000001) # normalize and scale
  vals = numeric()
  for (x in 1:10)
  { # compute count[x] points between mins[x] and maxs[x] (exclusive)
    tmp = seq(from=mins[x], to=maxs[x], length.out=counts[x] + 2)
    vals = append(vals, tmp[2:(counts[x] + 1)]) # append to output vector
  }
  vals # return vector
}
tri_vals1 = gen_tri_vals(10, 20, 90) # create a triang. distribution vector
tri_vals2 = gen_tri_vals(30, 120, 930)
d = data.frame(Tab            = gen_range(1, 5, norm_vals),
               Menu           = gen_range(3, 9, norm_vals),
               Search_Options = gen_range(5, 12, norm_vals),
               View           = gen_range(5, 20, norm_vals),
               Update         = gen_range(10, 30, norm_vals),
               Search_Results = gen_range(10, 90, tri_vals1),
               Create         = gen_range(15, 60, norm_vals),
               Inter_Task     = gen_range(30, 930, tri_vals2))
write.csv(d, file="think_time_tables_final.csv", row.names=FALSE)
```

Script 4 Think Time Distribution Estimator

```
# This excerpt illustrates how to estimate the overall think time
num_tables = 8
v = read.csv("think_time_tables_final.csv")
n = length(v[,1]) # the length of a table (they're all the same length)
counts = c(16126, 23194, 18968, 11601, 7347, 27309, 3506, 19314)
times = numeric()
for (x in 1:num_tables) # loop through all tables
{ # generate a sequence of random numbers to index the lookup table v[x]
  y = round(runif(counts[x]) * (n-1)) + 1 # generate random and convert to indices
  times = append(times, v[y,x]) # v[x] is a column; y is a vector of rows within it
}
plot(density(times), xlim=c(0,120), ylab="Frequency", las=1, lwd=2,
     xlab="Seconds", main="(Final) Think Time Distribution", col="blue")
```