Paper 3899-2019

Dynamic Call Center Simulation in SAS®: Revamping the Business Resiliency Process at JPMorgan Chase

Jay P. Carini, Amy Pielow, and Joel C. Weaver, JPMorgan Chase & Co.

ABSTRACT

The JPMorgan Chase Operations Research and Data Science Center of Excellence (ORDS CoE) has started a multi-year project to provide the internal Business Resiliency team with a simulation-based application to supplement current practices regarding resiliency testing and analysis. In the event of an outage, Business Resiliency seeks key insights about impacted locations: What will happen to service level during an outage? How will mitigation strategies impact service level (add headcount, reduce volume, and/or processing time)? The approach leverages simulation-based modeling (via SAS/OR® software) to estimate the expected impacts to service level due to an outage. The dynamic design of the model enables users to simulate any combination of numerous call centers and/or locations, with the ability to customize mitigation scenarios to compare with the “do-nothing” scenario. This presentation highlights the methodology employed through SAS, including: applying SAS/OR and PROC CPM as the simulation engine; deriving survival functions (PROC LIFETEST) to model abandonment behavior; fitting historical handle time data to probability distributions by call type and building a handle time function (PROC FCMP) for use in model parameterization; improving model performance through parallel processing with MP CONNECT; and generating output statistics through bootstrapping with PROC SURVEYSELECT.

INTRODUCTION

JP Morgan Chase (JPMC) inbound call centers currently employs a large, geographically-dispersed workforce responsible for handling Consumer & Community Banking (CCB) calls. When disruptions to one or more of these call centers occur, customers attempting to reach JPMC may experience atypical, lengthy wait times as a result of the drop in available resources to handle calls. To minimize service interruptions, JPMC artificially simulates these disruptions to estimate customer impacts and then creates contingency plans based on those findings. Leveraging simulation software to model these disruptions creates business value by both allowing for more what-if contingencies and mitigating business impacts.

JPMC’s Operations Research & Data Science Center of Excellence (ORDS CoE) developed a tool to predict key performance indicators (KPI) that may change during a resiliency event. After considering various simulation packages, ORDS decided to utilize the CPM Procedure within SAS/OR. This procedure schedules tasks based on given time and resource constraints, task priority and task duration information.

The scheduling algorithm effectively models each forecasted call queue relative to agent schedule constraints based on given expected call parameters (average call duration, arrival pattern), and customer behavior (willingness to wait before abandonment). There are numerous benefits to the PROC CPM approach:

- Easy to learn, build, and maintain
- Ability to simulate site shutdowns by each individual location
- Ability to assign resource availability by location and demand by time
- Designed to specify operating hours across days and shifts – accounting for weekends and holidays
- The capability to both forecast, as well as "back-test" the simulation using actual inputs instead of forecasts to dynamically gauge model fitness over time
- Easily incorporate multiple data sources through various SAS database connection methods

Call center agents can be single-skilled or multi-skilled in taking calls corresponding to different products and services offered by JPMC. When an inbound call arrives, it is either immediately routed to an available agent with the appropriate skillset or enters a queue when no agents are available. Calls in queue are answered based on priority level and then in the order they were received within that priority. The simulation models each of these queues and call centers, and is used to measure performance metrics when a location (and its resources) is removed from service, as shown in Figure 1.

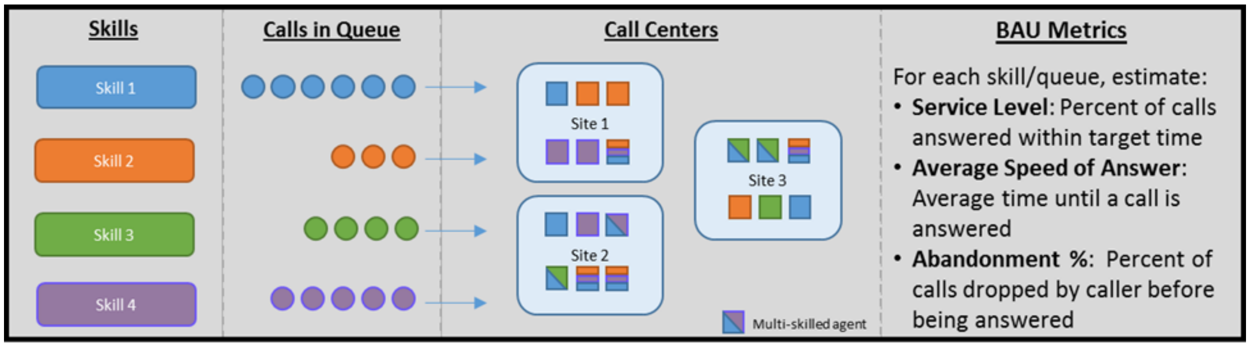

Skills
Calls in Queue
Call Centers
BAU Metrics
Skill 1
Skill 2
Skill 3
Skill 4
Site 1
Site 2
Site 3
Multi-skilled agent
For each skill/queue, estimate:
• Service Level: Percent of calls answered within target time
• Average Speed of Answer: Average time until a call is answered
• Abandonment %: Percent of calls dropped by caller before being answered

Figure 1. Simulation Process (BAU)

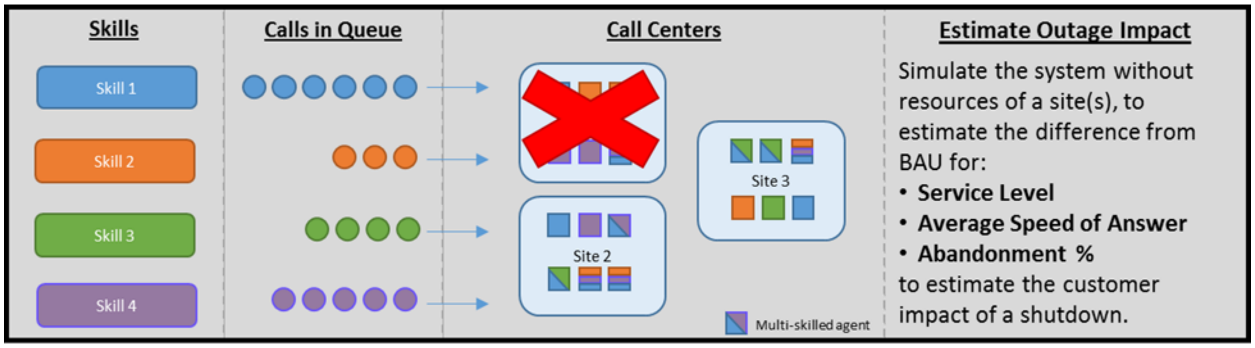

Skills
Calls in Queue
Call Centers
Estimate Outage Impact
Skill 1
Skill 2
Skill 3
Skill 4
Site 2
Site 3
Multi-skilled agent
Simulate the system without resources of a site(s), to estimate the difference from BAU for:
• Service Level
• Average Speed of Answer
• Abandonment %
to estimate the customer impact of a shutdown.

Figure 2. Simulation Process (Outage)

METHODOLOGY

PROC CPM is a SAS procedure to control and plan projects based on task duration, resource availability, and task precedence. While not a direct parallel to a call center, the project management tool can find feasible schedules of calls being answered (*tasks*) by available resources (*agents*). Agents are defined as a resource that can complete a task if they are trained on that type of call.

To create the input datasets necessary for PROC CPM, we used several other statistical methods within SAS to estimate the system processes outlined below:

1. **Call arrival patterns:** The distribution of calls coming into the system over a half-hour interval. For example, calls arriving mostly at the beginning of an interval and then tapering off versus calls arriving evenly throughout the interval.
2. **Call duration:** The average length of a call in seconds (commonly referred to as average handle time (AHT).
3. **Abandonment time:** The amount of time a caller is willing to wait before hanging up.

CALL ARRIVAL PATTERNS - DISTRIBUTION FITTING

Forecasts are provided for call volumes on a queue-level basis at thirty minute intervals. The volume then needs to be disseminated into individual calls with arrival times in the applicable thirty-minute time interval. Based on common queuing theory principles, the inter-arrival times (IAT) of events entering a system typically follow an exponential distribution. Utilizing historical call data, the IAT (length of time between call arrivals) was fit to several common probability distributions using the UNIVARIATE procedure as shown below:

```
ods output GoodnessOfFit=FitStats FitQuantiles=FitQuants
ParameterEstimates=ParmEsts ;
proc univariate data=call_history noprint;
     var IAT;
     histogram / normal gamma (theta=est)  lognormal (theta=est)
     exp (theta=est) ;
run;
```

Based on that analysis it was determined that historical IAT followed an exponential distribution in this system as well. Because the PROC CPM tasks are given a starting time, an arrival time was needed as opposed to time between calls. Since we take the number of arrivals and length of time as given, we can treat this as a Poisson point process on the real line, which means call arrival times will be uniformly distributed within the interval. Each call in the volume forecast is assigned a random (uniformly-distributed) arrival time during the thirty minute interval, which is defined in the model as the earliest start time for that task (atype = ‘sge’).

CALL DURATION – DISTRIBUTION FITTING

The task duration for each task (call) is based on the AHT for each call type during a given interval. Depending on the call type, the distribution of call handle times may be different. Similarly to the IAT analysis, historical call durations were fit to several known probability distributions using PROC UNIVARIATE. Additionally, the NPAR1WAY procedure was used when necessary to assess goodness of fit to a sample dataset from the Erlang distribution. The sample code below shows the options used:

```
proc npar1way data=AHT_Fit edf noprint;
     by call_type ;
     class sample;
     var aht;
     output out=erlangstats edf ;
run;
```

The data for several call types adequately fit a single distribution well (Normal, Exponential, Gamma, etc.). However, other calls types fit better when modeled as a mixture distribution. For example, a call type might be represented well as a mixture of normal distribution and exponential distribution.

Once an adequate distribution or mixture of distributions was determined for each call type, a permanent SAS function was developed to simplify the coding and dynamic nature of the process. Using the FCMP procedure, a parameterized function was created with the necessary inputs for generating a random variate for each call (task record) from the applicable probability distribution. The sample code below outlines the process:

```
proc fcmp outlib= sasuser.functions.simmod;
function handletime(INT_AHT, param_1, param_2, ... , param_n);

     -- Insert function logic here --

return(value);
endsub;
run;
```

Running the SAS code above creates a permanent function that call be called from any SAS program that first specifies the function library in an options statement. The sample options statement below needs to be executed before attempting to use the new function:

```
options cmplib=sasuser.functions;

data get_AHT;
set sample;
HT = handletime(200, 1, 0.5, 4);
run;
```

The distribution fitting process is repeated on a regular basis to ensure the handle time function is referencing the most up-to-date distribution details for a given call type.

ABANDONMENT TIME - SURIVIVAL ANALYSIS

The CPM procedure allows the user to specify a maximum delay before supplemental resources are employed. In the case of a call center, this maximum delay is used to represent the willingness to wait on the line before the customer abandons the call (see discussion in PROC CPM section below for more details of this unique application of PROC CPM).

Using historical data, the ORDS CoE generated survival curves to form the basis for estimating the maximum acceptable delay of calls. To begin the analysis, the historical call data was organized as demonstrated in the sample dataset in Figure 3 below.

Call_Group	Records	WaitTime	Abandoned
Call Group A	1000	0	0
Call Group A	100	0	1
Call Group A	2000	1	0
Call Group A	200	1	1
…	…	…	…
Call Group N	X	Z	0
Call Group N	X	Z	1

Figure 3. Sample Dataset for Survival Analysis

Each record represents the number of historical calls (*Records*), by Call Type (*Call Group*), by wait time (*WaitTime,* expressed in seconds), that Abandoned (*Abandoned* = 1) or waited for an agent to answer, i.e. a handled call (*Abandoned* = 0).

The LIFETEST procedure is applied to the data to develop a relationship between wait time and potential to abandon for each call type. An example of a survival curve produced from the procedure is provided in Figure 4 below. The curve represents the percentage of callers expected to wait for an agent as a function of wait time. For example, at wait times near zero close to 100% of callers are expected to wait for an available agent.

```
Proc Lifetest data= SampleDataset
outsurv=survout
method=LT
NINTERVAL=MaxInterval;
time WaitTime*ABANDON(0);
Strata Call_Group;
Freq Records;
run;
```

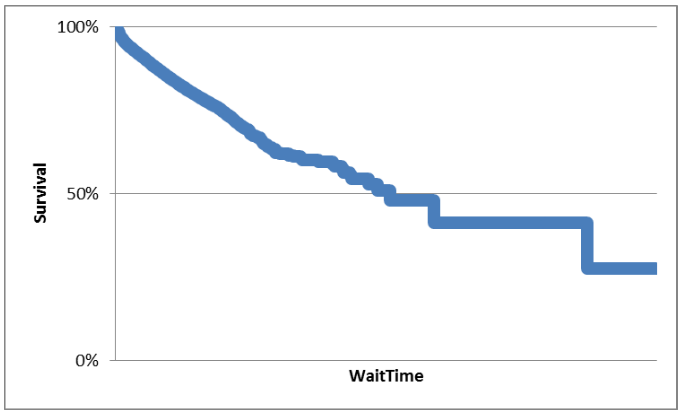


Figure 4. Sample survival curve

In order for this information to be ingested into the model, ORDS CoE created an expression to estimate the maximum wait time for each caller in the simulation. This expression was created using a linear regression estimation of Wait Time as a function of the –ln(Survival Rate). Figure 5 provides a graphical representation of this relationship, including an estimated trend line.

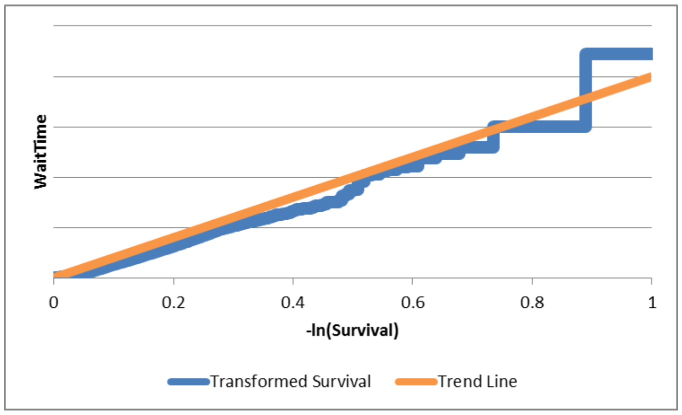


Figure 5. Sample transformation of survival curve

This transformation allows us to use a uniform random variable to simulate a survival percent and from that generate a corresponding wait time. Each call in the simulation is then assigned a maximum wait before abandon (unique for call type) using the following logic:

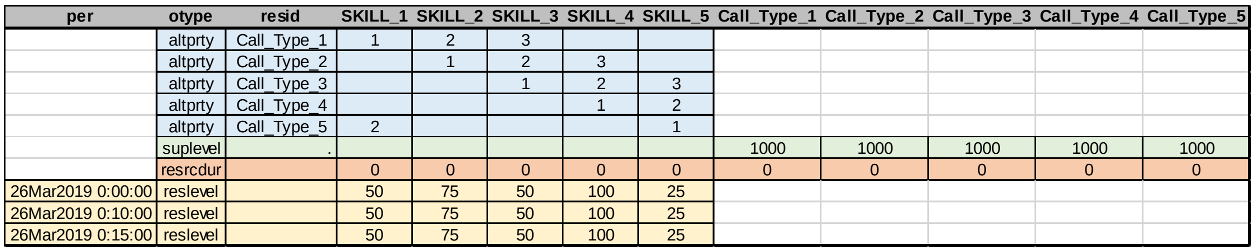
```
RandSeed = Rand('uniform');
NegLog = -LOG(RandSeed);
Max_Wait_Before_Abandon = RegressionParameter * NegLog;
```

PROC CPM

The three analyses feed into the PROC CPM processes part of the activity (task) dataset. In the task dataset, each row represents a call with duration, time of arrival (earliest start), maximum time until abandonment and necessary resource (agent with required skill set).

The call center system of record provides estimated head counts for each call type and interval. We update this distribution based on historic patterns of non-productive time (breaks, lunch, etc.) to generate resource availability dataset. Figure 6 represents a subset of the resource dataset.

- Blue section: shows the call types in the *resid* field that agents (Skill_1 – Skill_5) can answer. The values in Skill_1 – Skill_5 represent the agent priority for the given call type.
- Yellow section: shows the number of available agents for each skill during the given interval (*per*).
- Green section: sets the number of supplementary resources for each call type to be effectively unbounded. These are the resources that correspond to an abandoned call.
- Orange section: specifies that each resources has a *resrcdur* value of zero (fixed duration effect).



per	otype	resid	SKILL_1	SKILL_2	SKILL_3	SKILL_4	SKILL_5	Call_Type_1	Call_Type_2	Call_Type_3	Call_Type_4	Call_Type_5
	altprty	Call_Type_1	1	2	3							
	altprty	Call_Type_2		1	2	3						
	altprty	Call_Type_3			1	2	3					
	altprty	Call_Type_4				1	2					
	altprty	Call_Type_5	2				1					
	suplevel	.						1000	1000	1000	1000	1000
	resrcdur		0	0	0	0	0	0	0	0	0	0
26Mar2019 0:00:00	reslevel		50	75	50	100	25					
26Mar2019 0:10:00	reslevel		50	75	50	100	25					
26Mar2019 0:15:00	reslevel		50	75	50	100	25					

Figure 6. Sample Resource Dataset

The other inputs into the procedure specify calendars, shifts, and holidays. Descriptions of the other input datasets and their relationships to one another can be found in SAS paper 2621-2018, "Improving Scheduling and Strategic Planning at JPMorgan Chase's Card Production Center". A sample code for the PROC CPM execution can be seen below:

```
proc cpm data=tasks resin=resources workdata=shifts
calendar=calendar out=savec resourcesched=Resourceschedule noutil
interval=dthour;
     activity Task;
     calid Cal;
     duration NumHour;
     aligndate adate;
     aligntype atype;
     res  Skill_1 - Skill_&j.
     / period=per obstype=otype resid=resid schedrule=ACTPRTY
     ACTIVITYPRTY=FinalPriority schedrule2=shortdur
     actdelay=maxdelay delayanalysis  awaitdelay;
run;
```

EFFICIENCY STRATEGIES

Simulations of the call center environment can include millions of calls over a multi-week time horizon with thousands of agents. Although this could be solved in a single call to PROC CPM, several methods of code optimization are employed to improve model tractability. These methods, described below, leverage the structure of the problem, SAS parallel processing, and statistical sampling theory.

TIME SUBSETTING

In lieu of modeling the entire time horizon in a single call of PROC CPM, individual thirty minute blocks of time are run and then stitched together at the end of the process. This method has several benefits; primarily, the PROC CPM code processes fewer calls at once and is faster overall. An additional benefit is that each time period being modeled matches the underlying time frame used in the volume forecast.

To correctly model the continuous process, after each half hour run the code calculates all calls still in queue and those calls being worked. The calls in queue are given highest priority and their maximum wait duration is reduced by the amount of time spent in queue in the previous interval. Agents in call at the end of the interval are kept in the same task for the remaining duration at the start of the next interval. This is done with basic data and PROC SQL calls.

PARALLEL PROCESSING

Representing the call center behavior using probability distributions for various call attributes and applying survival curves to mimic a customer's willingness to wait on hold are examples of methods to improve the simulation's representation of reality. However, these methods also inherently introduce random variability into the model. With that said, running the simulation model a single time for a particular use case will only yield one example of a possible predicted outcome based on the scenario inputs. A single simulation run does not produce results that are necessarily a good representation of reality.

In order to produce results that will more likely encompass a close representation of reality, we decided to run the model multiple times and summarize the output metrics. Due to the complexity of the environment being modeled, it was difficult to run multiple simulations in a reasonable timeframe. Depending on the simulated outage, the call volume and number of impacted sites can vary greatly. A simple scenario could take minutes while a more complex scenario could take hours. With some simulations taking multiple hours to execute, it would take days to generate enough simulations for the appropriate calculations of output statistics.

The solution was to run multiple simulations at the same time through parallel processing. Utilizing parallel processing quickly increased the output rate for simulation runs. Parallel processing can be invoked using the following example code:

```
option autosignon=yes sascmd="!sascmd" sysrputsync;
%let pwork=%sysfunc(pathname(work));

*Parallel Process 1;
%syslput pwork="&pwork."/remote=sim1;
rsubmit process=sim1 wait=no sysrputsync=yes;
libname simpath &pwork.;

-- Insert SAS code here --

%sysrput path1=%sysfunc(pathname(work));
endrsubmit;


*Parallel Process 2;
%syslput pwork="&pwork."/remote=sim2;
rsubmit process=sim2 wait=no sysrputsync=yes;
libname simpath &pwork.;

-- Insert SAS code here –-

%sysrput path2=%sysfunc(pathname(work));
endrsubmit;

waitfor _all_ sim1 sim2;

libname sess1 "&path1.";
libname sess2 "&path2.";

data alldata;
set sess1.data sess2.data ;
run;
```

When working with parallel processing it can be useful to pass macro variables back and forth between the local and remote hosts. Macro variables can be passed from the local host to the remote host using the %syslput macro statement. Alternatively, the %sysrput macro statement will pass the value of a remote macro variable to a macro variable on the local host.

The example above is set up to pass the path of the original session’s work directory to each spawned parallel session. In doing this, data sets generated as part of the original session can be ingested and manipulated separately in each parallel session. Additionally, the path for the work directory from each parallel session is being passed back to the local session. This allows the local session to reference datasets generated as part of the parallel process. In practice, there are multiple macro variables that can be passed between hosts to maximize the functionality of parallel processing. Figure 7 outlines the performance benefits of parallel processing. Assuming each simulation task takes 1 hour, switching from serial processing to complete parallel processing would save 5 hours processing time for the example below. Due to system performance and available memory, it might be difficult to run all tasks in parallel. However, running multiple parallel processes with smaller serial jobs will still significantly reduce the overall processing time.

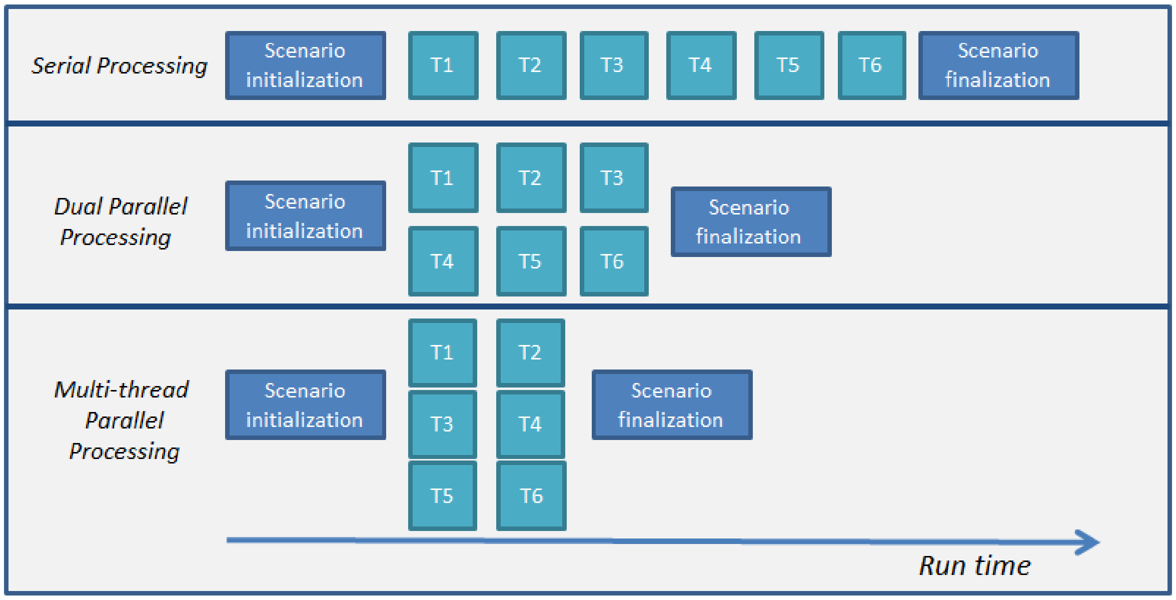


Figure 7. Serial vs. Parallel Processing Illustration

BOOTSTRAPPING

The use of parallel processing increased the number of simulations we were able to run in a reasonable amount of time. However, the sample size (number of simulations) was not large enough to support a robust statistic derived from traditional inferential statistics. The KPI's could instead be calculated on a larger sample size built from many more bootstrap samples. Instead of providing output statistics as a single point estimate from a small sample, we could provide estimates with confidence intervals for each KPI.

Developing confidence intervals for any of the simulation output statistics starts with generating N bootstrap samples of size m, where N is a sufficiently large sample size and m is the number of simulation runs. Table 1 represents example output for a particular KPI, average speed of answer (seconds), for an example simulation scenario. In this example there were 10 iterations for the simulation. Therefore, all bootstrap samples should have 10 values.

Table 1. Sample Simulation Output (Average Handle Time, seconds)

Simulation Output	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run 7**	**Run 8**	**Run 9**	**Run 10**
Average Speed of Answer (Call Type A)	9	20	13	16	11	10	15	12	7	21

To simplify the explanation, this example only generates 20 bootstrap samples. In reality, hundreds or thousands or samples would be created. The bootstrap samples are created using simple random selection with replacement. This is accomplished with PROC SURVEYSELECT and specifying the method of sampling as URS (unrestricted random sampling).

```
proc surveyselect data=sim_KPI method=URS  samprate=1 reps=20
out=bootsamps noprint;
strata call_type;
run;
```

The URS method samples records with equal probability and with replacement. That means that any given value from the original sample could appear multiple times in a single bootstrap sample. For example, as shown in Table 4, 11 appears twice in bootstrap sample 1, but there was only one 11 in the original sample.

Specifying the *samprate*= option tells PROC SURVEYSELECT the portion that should be sampled. Setting samprate equal to 1, indicates a sample size of 100% (10 records in our example). The *reps*= options specifies how many replications, or in our case, the number of bootstrap samples we would like. Lastly, the *strata*= option specifies that sampling should occur independently for each group identified by the named variable. In our case, we would like to generate bootstrap samples separately for each call type.

Table 2 shows example input and Table 3 shows example output data sets for PROC SURVEYSELECT as outlined above.

Table 2. Sample Input Dataset (Sim_KPI)

Call_Type	ASA
A	9
A	20
A	13
A	16
A	11
A	10
A	15
A	12
A	7
A	21
B	5
B	3
B	6
B	7
B	3
B	9
B	6
B	8
B	3
B	10

Table 3. Sample Output Dataset (bootsamps)

Call_Type	Replicate	ASA	NumberHits
A	1	11	2
A	1	13	1
A	1	10	2
A	1	16	3
A	1	7	1
A	1	9	1
A	2	13	3
A	2	11	1
A	2	7	1
A	2	16	2
A	2	9	1
A	2	12	1
A	2	21	1

The output dataset in Table 3 is only showing the first two replications for the call type A strata. The actual dataset would have replicate values of 1 – 20 for both strata A and B. The two variables worth noting in the output dataset are *Replicate* and *NumberHits*. *Replicate* corresponds to the bootstrap sample and *NumberHits* indicates how many times the ASA value in that record was selected for the group. The number of records in the output dataset for each replication will not necessarily be equivalent to the samprate value (100% or 10 records in our case), because values can be selected more than once when creating the bootstrap samples. However, the sum of the records for *NumberHits* for a given replication should be equal to 10. With basic data manipulation of the output dataset, you can create your bootstrap samples in simple view as depicted in Table 4.

Table 4. Example Bootstrap Sample Output

Bootstrap Sample											Sample Mean
1	11	13	11	10	16	10	7	9	16	16	**11.9**
2	13	11	7	16	13	9	12	16	13	21	**13.1**
3	12	12	16	7	21	7	12	13	13	10	**12.3**
4	12	20	21	15	20	10	7	20	13	20	**15.8**
5	11	20	13	21	9	21	21	9	12	10	**14.7**
6	9	13	15	16	15	11	13	7	13	16	**12.8**
7	20	21	13	13	21	12	7	13	16	12	**14.8**
8	12	10	16	21	9	9	11	20	12	16	**13.6**
9	9	13	11	16	10	12	15	15	20	16	**13.7**
10	15	21	21	15	16	21	10	7	21	11	**15.8**
11	11	7	12	12	21	16	16	13	15	21	**14.4**
12	11	20	7	12	20	7	15	9	16	20	**13.7**
13	10	12	21	12	7	7	16	10	13	12	**12.0**
14	11	16	12	13	20	13	20	7	16	20	**14.8**
15	16	13	16	7	7	12	10	20	21	15	**13.7**
16	16	10	15	15	21	7	12	21	12	12	**14.1**

17	13	15	11	7	15	10	10	9	12	15	**11.7**
18	20	13	13	12	21	12	15	13	9	10	**13.8**
19	7	9	21	12	9	11	10	9	12	15	**11.5**
20	20	9	10	11	12	11	13	9	15	21	**13.1**

Once you have generated the desired number of bootstrap samples (Table 4), you can begin calculating the desired statistics and confidence intervals. For this example, we are calculating the mean with a 90% confidence interval. The following steps outline the process for this example:

1. Calculate the sample statistic (mean) for each bootstrap sample
2. Order the statistic from smallest to largest
3. Remove the top and bottom 5% ($^{\propto}/_{2}$) from the list.
   ~~11.5~~, 11.7, 11.9, 12.0, 12.3, 12.8, 13.1, 13.1, 13.6, 13.7, 13.7, 13.7, 13.8, 14.1, 14.4, 14.7, 14.8, 14.8, 15.8, ~~15.8~~
4. The remaining minimum and maximum values represent your lower and upper confidence limits, respectively. **(11.7, 15.8)**

MODEL USAGE

The process flow for usage of the SAS model can be shown by four main categories (Figure 8). The model is accessed by the business users through an IT supported user interface (Display 1).

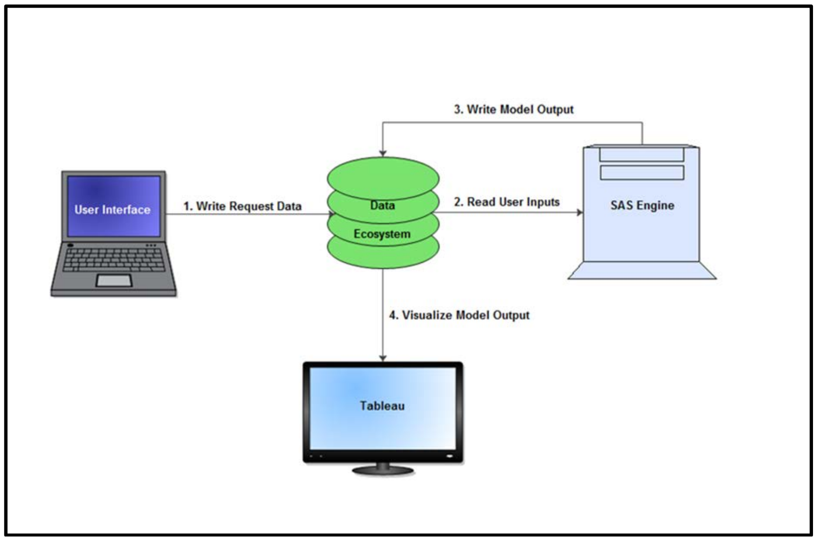
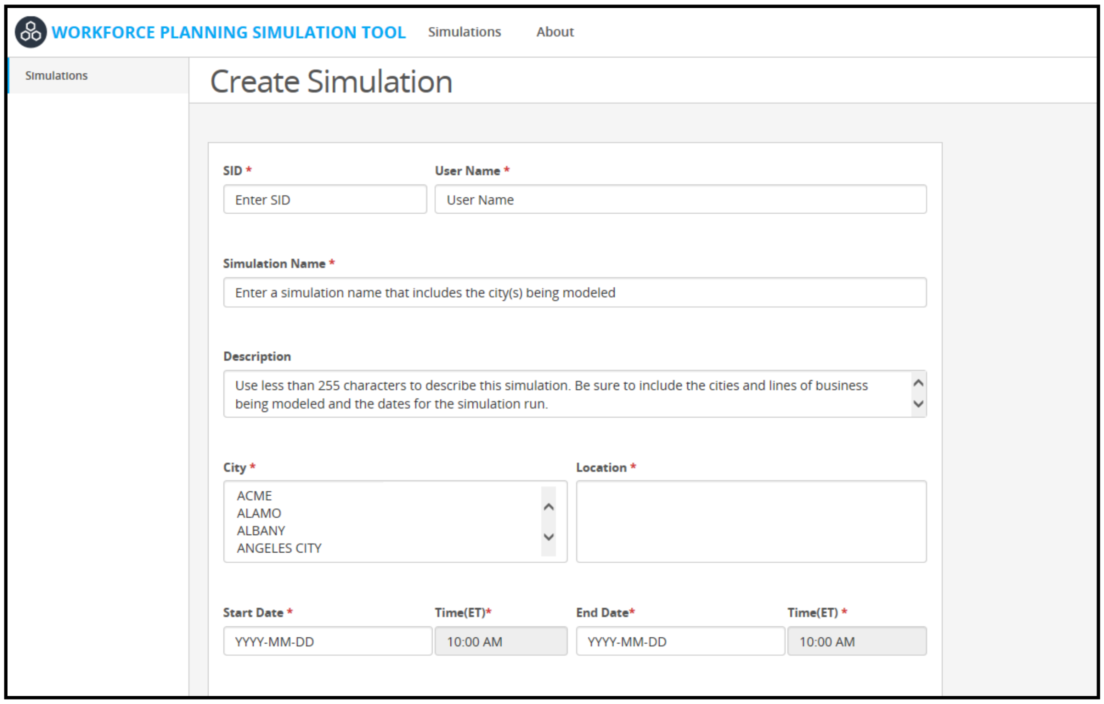


Figure 8. Simulation Model Process Flow

The user's scenario request is then captured in the data ecosystem, where the SAS program can ingest appropriate model inputs.

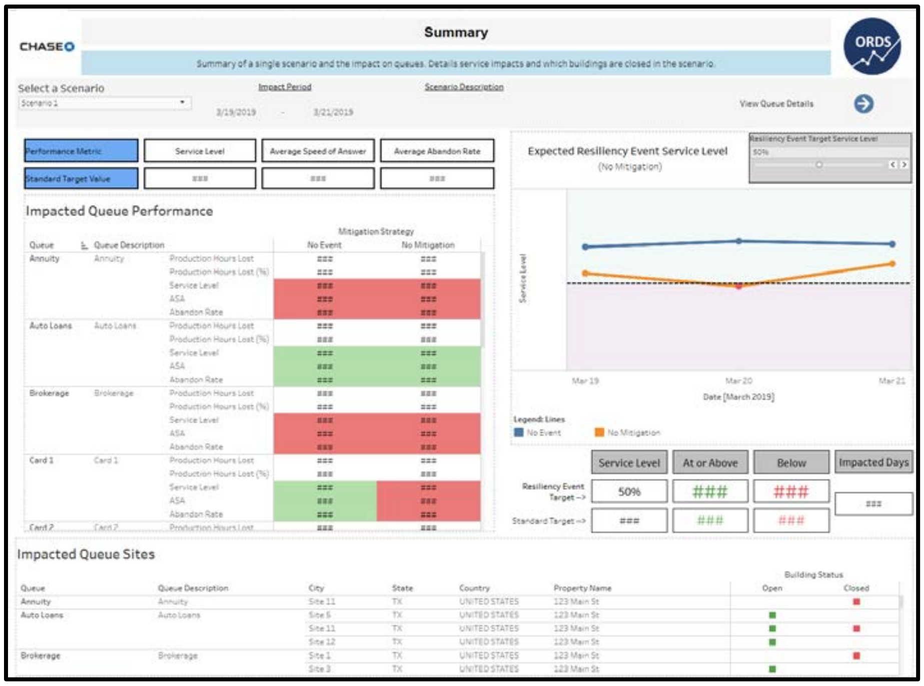


Create Simulation

SID *	Enter SID
User Name *	User Name
Simulation Name *	Enter a simulation name that includes the city(s) being modeled
Description	Use less than 255 characters to describe this simulation. Be sure to include the cities and lines of business being modeled and the dates for the simulation run.
City *	ACME ALAMO ALBANY ANGELES CITY
Location *	
Start Date *	YYYY-MM-DD
Time(ET)*	10:00 AM
End Date*	YYYY-MM-DD
Time(ET) *	10:00 AM

Display 1. Sample User Interface Screen

After executing the PROC CPM portion of the SAS script, the output summary is generated and written back to the data ecosystem. At that point, a live Tableau connection reflects the scenario output in a Tableau dashboard to the business users. Displays 2 and 3 show example mockups of the dashboard details.



Summary of a single scenario and the impact on queues. Details service impacts and which buildings are closed in the scenario.

Select a Scenario: Scenario 1

Impact Period: 3/19/2019 - 3/21/2019

Scenario Description

Performance Metric	Service Level	Average Speed of Answer	Average Abandon Rate
Standard Target Value	###	###	###

Impacted Queue Performance

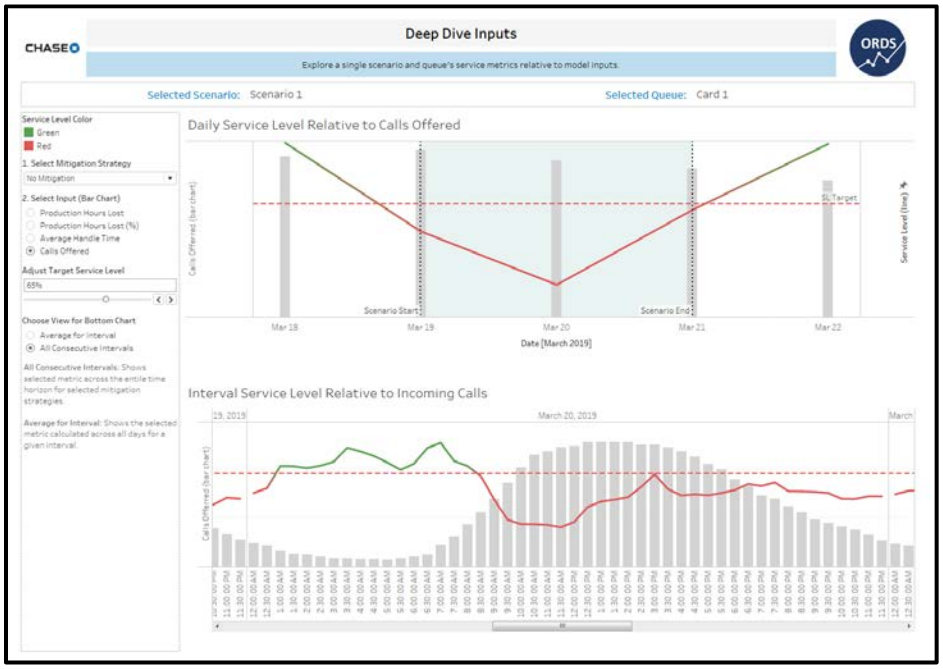
			Mitigation Strategy	
Queue	Queue Description		No Event	No Mitigation
Annuity	Annuity	Production Hours Lost	###	###
		Production Hours Lost (%)	###	###
		Service Level	###	###
		ASA	###	###
		Abandon Rate	###	###
Auto Loans	Auto Loans	Production Hours Lost	###	###
		Production Hours Lost (%)	###	###
		Service Level	###	###
		ASA	###	###
		Abandon Rate	###	###
Brokerage	Brokerage	Production Hours Lost	###	###
		Production Hours Lost (%)	###	###
		Service Level	###	###
		ASA	###	###
		Abandon Rate	###	###
Card 1	Card 1	Production Hours Lost	###	###
		Production Hours Lost (%)	###	###
		Service Level	###	###
		ASA	###	###
		Abandon Rate	###	###
Card 2	Card 2	Production Hours Lost	###	###

	Service Level	At or Above	Below	Impacted Days
Resiliency Event Target -->	50%	###	###	###
Standard Target -->	###	###	###	

Impacted Queue Sites

						Building Status	
Queue	Queue Description	City	State	Country	Property Name	Open	Closed
Annuity	Annuity	Site 11	TX	UNITED STATES	123 Main St		🟥
Auto Loans	Auto Loans	Site 5	TX	UNITED STATES	123 Main St	🟩	
		Site 11	TX	UNITED STATES	123 Main St	🟩	🟥
		Site 12	TX	UNITED STATES	123 Main St	🟩	
Brokerage	Brokerage	Site 1	TX	UNITED STATES	123 Main St		🟥
		Site 3	TX	UNITED STATES	123 Main St	🟩	

Display 2. Sample Summary Output Screen


Deep Dive Inputs

Explore a single scenario and queue's service metrics relative to model inputs.

Selected Scenario: Scenario 1 Selected Queue: Card 1

Service Level Color
🟩 Green
🟥 Red

1. Select Mitigation Strategy
No Mitigation

2. Select Input (Bar Chart)
Production Hours Lost
Production Hours Lost (%)
Average Handle Time
[✓ Calls Offered]

Adjust Target Service Level
65%

Choose View for Bottom Chart
Average for Interval
[✓ All Consecutive Intervals]

All Consecutive Intervals: Shows selected metric across the entile time horizon for selected mitigation strategies.

Average for Interval: Shows the selected metric calculated across all days for a given interval.

CHASE
ORDS
Daily Service Level Relative to Calls Offered
Calls Offered (bar chart)
Service Level (line)
SL Target
Scenario Start
Scenario End
Mar 18
Mar 19
Mar 20
Mar 21
Mar 22
Date [March 2019]
Interval Service Level Relative to Incoming Calls
19, 2019
March 20, 2019
March
Calls Offered (bar chart)

Display 3. Sample Interval-Level Output Screen

CONCLUSION

Using PROC CPM to represent the call centers at JPMC has several benefits, including giving developers access to the statistical and analytical capabilities in SAS within a single tool. Leveraging SAS allows the modeling process to easily ingest multiple sources of data and perform significant amounts of exploratory data analysis prior to beginning the CPM procedure. This upfront analysis and processing would be more difficult in other simulation software. Furthermore, the versatility of PROC CPM also makes it easy to represent most business processes, including those not fitting the traditional waterfall project management structure. This model is currently in development and testing, but augmenting current resiliency practices with simulations can provide significant business value.

REFERENCES

Baddeley, Adrian. “Spatial Point Processes and their Applications”. Accessed March 12. 2019. Available at https://pdfs.semanticscholar.org/f4a2/fe1b2a7b1a87b05f271dad1c7734ebcab686.pdf

Carini, Jay, Weaver Joel. 2017. “Improving Scheduling and Strategic Planning at JPMorgan Chase’s Card Production Center”.

Cassel, David. 2010. “BootstrapMania! Re-Sampling the SAS Way”. Proceedings of the SAS Global Forum 2010 Conference, Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings10/268-2010.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Jay Carini
JPMorgan Chase
jay.p.carini@chase.com

Joel C. Weaver
JPMorgan Chase
joel.c.weaver@chase.com

Amy Pielow
JPMorgan Chase
amy.pielow@chase.com