Expressing Latent Demand as a Single Number

Jim Horne
Lowe's Companies, Inc.

Latent demand is an important factor in using computers efficiently but it can be difficult to determine and show. Understanding how much exists can influence tuning decisions but how do you communicate it? This paper presents a method to express latent demand on z/OS as a single number that can be used to help determine if further action needs to be taken.

Latent demand can be called a desire that is not currently being satisfied because no suitable good or service is available. This is a concept familiar to almost everyone. For computer performance analysts and capacity planners it is especially relevant, because latent demand means there is work that cannot be done and tasks that cannot be completed in a timely manner. This paper will look at the tools IBM provides for measuring this unsatisfied work and present a methodology to express it as a single number.

Getting the numbers

For many years IBM has provided numbers showing the amount of work that was In and Ready (IR) in the SMF type 70 SMF70Rnn variables. These variables provide buckets with counts of how many times n number of tasks are IR in an RMF interval. SMF70R00 has the count of how many times there are no tasks IR, SMF70R01 has the count of how many times one task is IR, on up to SMF70R14, which has the count of how many times fourteen tasks are IR. The last SMF70Rnn variable, SMF70R15, provides the count of how many times fifteen or more tasks are IR. These buckets worked well in the past but have become inadequate to show the true pattern of ready work in today's world – one where many shops have more than fourteen processors and can expect most of their IR work to be shown in SMF70R15.

In z/OS 1.7 IBM provided a new set of variables, SMF70Q00 through SMF70Q12. These variables – or buckets as we will now call them – work differently than the older buckets do, and they provide more useful information. Each bucket has a count of how many times in an RMF interval the count of ready tasks compares to the number of engines available. SMF70Q00 has the count of how many times the number of ready tasks was less than or equal to the number of engines available. SMF70Q01 has the count of how many times the number of ready tasks was one more than the number of available engines. Higher buckets encompass larger ranges and the last bucket, SMF70Q12, has the count of how many times the number of ready tasks was more than eighty greater than the number of available engines. Figure 1 has the complete list of buckets and their granularity.

Figure 1. SMF70Qnn Count of In and Ready tasks compared to available engines

SMF70Q00	<=N
SMF70Q01	=N+1
SMF70Q02	=N+2
SMF70Q03	=N+3
SMF70Q04	4<=N<=5
SMF70Q05	6<=N<=10
SMF70Q06	11<=N<=15
SMF70Q07	16<=N<=20
SMF70Q08	21<=N<=30
SMF70Q09	31<=N<=40
SMF70Q10	41<=N<=60
SMF70Q11	61<=N<=80
SMF70Q12	N>80

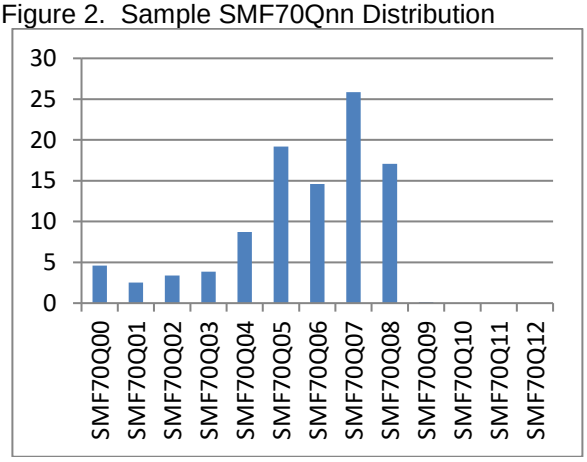
N=available engines

This provides more granularity than the old buckets, and offers immediate possibilities in determining latent demand since buckets SMF70Q01 through SMF70Q12 show that there is more work than can currently execute because there are not enough engines to run it. In Cheryl Watson's Tuning Letter 2007 No. 3 (Watson, 2007), she discusses this topic and offers a method to calculate the average number of tasks waiting on engines. While her technique shows the power of the information in these buckets, this paper will take a different approach. Using the PIF technique detailed by Charles Loboz, Steve Lee and James Yuan in their 2009 CMG paper, "How do you measure and analyze 100,000 servers – daily?" (Loboz, Lee, & Yuan, 2009), we will compute a single number showing the amount of latent demand in a single RMF interval.

Consolidating the buckets

What Loboz, Lee and Yuan did was construct a histogram of data for CPU consumption, putting observations into buckets that corresponded to the amount of CPU consumed. Then they computed the weighted sum of those buckets and determined the logarithm of that number, arriving at a much smaller number that could be rated in an easy to understand way. They called this number a PIF, Performance Impact Factor. It may seem hard to understand at first, but by breaking it down into its parts we will see how we can use this technique to compute a single number for latent demand.

Figure 2. Sample SMF70Qnn Distribution



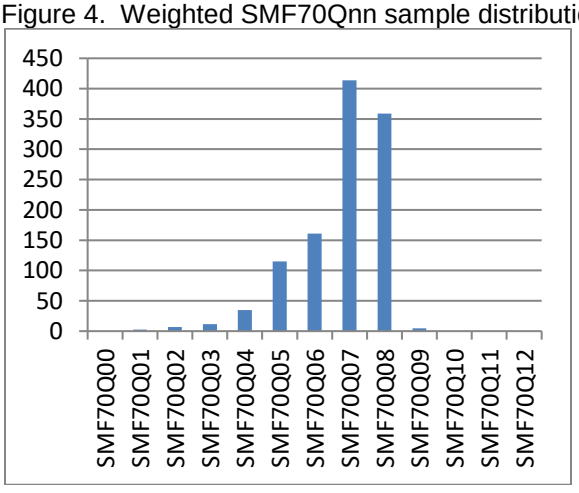
Looking at Figure 2, we see a possible distribution of SMF70Qnn values. The more to the right of the graph something is, the more tasks are waiting, indicating more latent demand. This means we need to weight the buckets so the higher the ratio of tasks to engines is, the more important the buckets get. Loboz, Lee and Yuan had to use queuing theory to help guess at weights that might work. Unlike them, we have a ready built set of weights available to us – the number of tasks waiting relative to the number of available engines in the SMF70Qnn variables. Figure 3 has the list of weights and Figure 4 shows how the weighted distribution looks.

Figure 3. SMF70Qnn and weights

Bucket	Tasks	Weight
SMF70Q00	<=N	0
SMF70Q01	=N+1	1
SMF70Q02	=N+2	2
SMF70Q03	=N+3	3
SMF70Q04	4<=N<=5	4
SMF70Q05	6<=N<=10	6
SMF70Q06	11<=N<=15	11
SMF70Q07	16<=N<=20	16
SMF70Q08	21<=N<=30	21
SMF70Q09	31<=N<=40	31
SMF70Q10	41<=N<=60	41
SMF70Q11	61<=N<=80	61
SMF70Q12	N>80	81

N=available engines

Figure 4. Weighted SMF70Qnn sample distribution.



Building the Formula

Now we are ready to look at reducing the buckets to a single number by summing them together. Mathematically this looks like $\frac{\sum_{i=0}^{12}(h_i w_i)}{\sum_{i=0}^{12} h_i}$, where h_i is the value in each bucket and w_i is the weight associated with that bucket. There is no need to get excited if this seems hard to understand. It gets simpler when we realize that $\frac{h_i}{\sum h_i}$ is the percent each bucket contributes to the whole. This simplifies the formula to $\sum_{i=0}^{12} p_i w_i$, where p_i is each bucket's percent of the whole and w_i is its weight. Now that we see this, let us leave the esoteric world of mathematics and enter the world of programming languages where we can actually use these formulae.

If you are an MXG® user, the language of choice to work with things is SAS®. MXG has already computed the percentages of the whole for each of the SMF70Qnn variables and replaced the values of SMF70Q00 through SMF70Q12 with their percentages. This means the SMF70Qnn values are ready to sum without any additional work. However, if you are not using MXG, you can sum SMF70Q00 through SMF70Q12 and divide each variable by that number and then multiply by 100 to calculate its percentage. For the rest of this paper we will assume you are using MXG and already have SMF70Qnn expressed as percentages.

So we now have a number – but what does it mean? If all tasks were in the SMF70Q12 bucket (more than 80 greater than available engines), our number would be 8100. We got this number by multiplying this bucket's percent of the whole (100%) by its weight (81) and again by 100 If all tasks were in SMF70Q00, our number would be 0 (100*0*100). The difference between these extremes is too large to deal with easily, especially if we want to be able to show management numbers that are easy to grasp. What Loboz, Lee and Yuan did was take the logarithm of their sums. If we do the same, we first need to throw away any sum less than 1 because the logarithm of a number between 0 and 1 is a negative number. Since our sums range from 0 to 8100 this does not present a problem because we will not lose any definition of latent demand that we care about by ignoring such a small number. But the logarithm of 8100 is 3.91, while the logarithm of 1000 is 3. This does not appear to be granular enough to be of use to us. However, if we look at other numbers, we see that the natural logarithm (logarithms using the number base *e*) of 8100 is 9. Applying natural logarithms, we see that the other whole numbers between 0 and 9 provide the granularity that base 10 logarithms did not.

Latency

While Loboz, Lee and Yuan called their number a PIF (Performance Impact Factor), the number we have computed is much more specialized and uses weights that follow from the values IBM placed on the buckets. Let us call the number we get with this simpler process *latency*. The formula to calculate latency is straightforward. First we compute the weighted sum of all SMF70Qnn variables, except SMF70Q00, which has a weight of zero,

```
summ70q =   (1*smf70q01) + (2*smf70q02)  + (3*smf70q03) +
            (4*smf70q04) + (6*smf70q05)  + (11*smf70q06) +
            (16*smf70q07)+ (21*smf70q08) + (31*smf70q09) +
            (41*smf70q10)+ (61*smf70q11) + (81*smf70q12) ;.
```

Then we compute latency itself, using the value of summ70q. Since the logarithm of 0 cannot be computed we put in a check for that. We also make sure that we ignore negative logarithms, since they represent real values too small to worry about. This gives us,

```
IF summ70q = 0 THEN latency = 0;
ELSE               latency = MAX(LOG(summ70q),0;.
```

Note that the LOG function in SAS refers to natural logarithms instead of using LN, as you might expect.

Let us look at latency values we might get in a "perfect world." Figure 5 is a table of latency values that would result if all time except that shown in the table was in SMF70Q00 (and so contributing zero to latency). For example, in the 25% column, we see a value of 5.01 corresponding to variable SMF70Q05, which has a weight of 6. This means that 25% of the time we had between 6 and 10 tasks waiting on engines and the rest of the time we have no tasks waiting on engines. The minimum latency shown is zero, for a value of SMF70Q01 at 1% of the time and no wait 99% of the time. The maximum latency is 9 (rounded to two digits past the decimal point). This can only be achieved when SMF70Q12 is at 100%, indicating that more than 80 tasks were waiting for engines the entire interval.

Figure 5. Latency values for single values of SMF70Qnn buckets

Variable	Weight	1%	10%	25%	50%	75%	90%	100%
SMF70Q01	1	0.00	2.30	3.22	3.91	4.32	4.50	4.61
SMF70Q02	2	0.69	3.00	3.91	4.61	5.01	5.19	5.30
SMF70Q03	3	1.10	3.40	4.32	5.01	5.42	5.60	5.70
SMF70Q04	4	1.39	3.69	4.61	5.30	5.70	5.89	5.99
SMF70Q05	6	1.79	4.09	5.01	5.70	6.11	6.29	6.40
SMF70Q06	11	2.40	4.70	5.62	6.31	6.72	6.90	7.00
SMF70Q07	16	2.77	5.08	5.99	6.68	7.09	7.27	7.38
SMF70Q08	21	3.04	5.35	6.26	6.96	7.36	7.54	7.65
SMF70Q09	31	3.43	5.74	6.65	7.35	7.75	7.93	8.04
SMF70Q10	41	3.71	6.02	6.93	7.63	8.03	8.21	8.32
SMF70Q11	61	4.11	6.41	7.33	8.02	8.43	8.61	8.72
SMF70Q12	81	4.39	6.70	7.61	8.31	8.71	8.89	9.00

This gives us some idea of what values can mean in the real world. Very rarely will the buckets be so sparsely populated but the table does given us a clear idea of what is bad and good. In Figure 5 latency is shaded into red, yellow and green for bad, warning and good values – traffic lighting, in other words. The choices made in this table, 0-5 shown in green, 5-7 shown in yellow and 7-9 shown in red, are for illustration only but they show how simple it is to make value judgments in a meaningful way now that the number is small enough to see.

Summarizing Latency – a single number for larger intervals of time

You can pick a summary value using average, percentiles or other method that results in a single number or you can compute the total latency for any interval you like. To compute the latency for longer periods of time than a single RMF interval proceed as follows.

If you are using the raw SMF variables simply include all the intervals you want to summarize. If you are using MXG, you need to do a little more work. First, multiply each SMF70Qnn variable by the value of DURATM for that interval because you cannot add percentages. SAS code using arrays to help would look like

```
ARRAY q70(*) smf70q00-smf70q12;
ARRAY sum70(*) sum70q00-sum70q12;
DO ictr = 1 TO DIM(q70);
  sum70(ictr) = q70(ictr) * duratm;
END;
```

This code can be inserted in a DATA step. Then you could execute a PROC MEANS to sum these variables across your desired range of intervals. Code to sum an entire shift might look like

```
PROC MEANS DATA=yourdata NOPRINT;
   BY date shift system; * you must calculate date;
   VAR sum70q: duratm;
   OUTPUT OUT=yoursumm(DROP=_TYPE_ _FREQ_) SUM=;
RUN;
```

Finally, divide each summed SMF70Qnn by the summed DURATM to get a percentage again and use the same method we already know for calculating single intervals. Code to put the SMF70Qnn variables back to percentage in a DATA step might look like

```
FORMAT smf70q:          6.2;
ARRAY sum70(*) sum70q00-sum70q12;
ARRAY q70(*) smf70q00-smf70q12;
DO ictr = 1 TO DIM(q70);
  q70(ictr) = sum70(ictr) / duratm;
END;
```

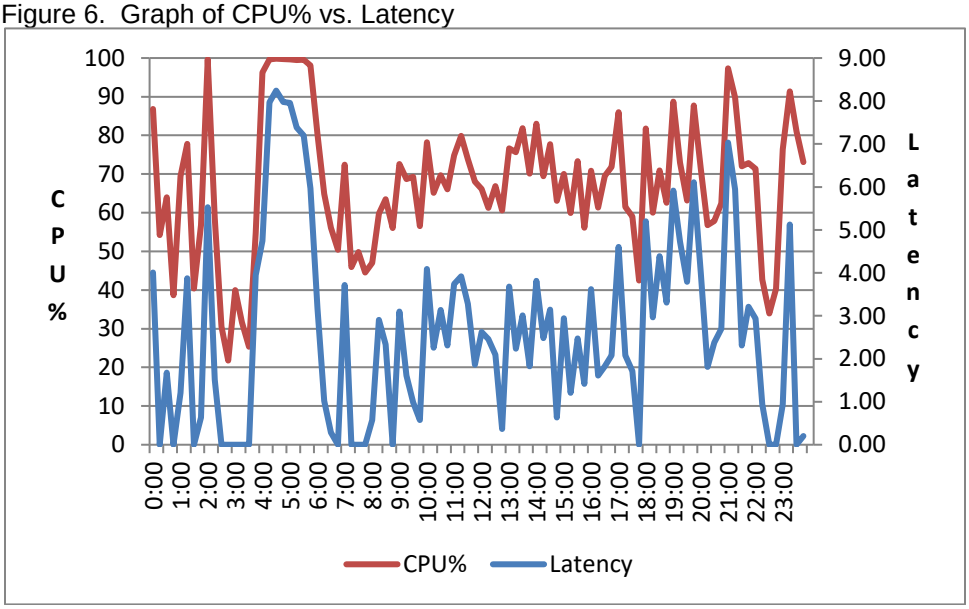
The code to sum the individual SMF70Qnn variables into a single variable is not included again because it is exactly the same as though this represented a single interval – which it does. The only difference is that the size of this interval is an entire shift rather than a single RMF interval.

Using Latency

Now we know how to calculate latency and it looks interesting but how can we take it one step further and use it in the real world? Alone it can be an indicator of pain, of how much a system is suffering because there is more work ready to run than it has engines available. That is what latency is intended to show. But how is it more useful than other measurements such as CPU percent busy? The answer is granularity.

Figure 6 demonstrates its usefulness by contrasting CPU percent busy, shown in red and graphed on the right axis against latency, shown in blue and graphed on the left axis, over the same one day period. On this hypothetical system, we are drawn immediately to the area of the graph showing the CPU running flat out. But latency, showing the amount of latent demand on this system, tells us more about what is going on. It peaks as CPU% first hits maximum, but it doesn't stay there; it drops even while the CPU stays at max. This graph immediately tells us where the worst point is, and shows that the system is working to relieve latent demand as resources become available. It also tells us that things could be worse because latent demand never hits maximum and even drops to zero several times during the day.

Figure 6. Graph of CPU% vs. Latency



This is excellent information and the graph makes it easy to see where to start looking for issues. If we want to know how much latency is present, we will need to know which buckets are contributing to this number – and in what proportions. Referring to Figure 5, we can see that to have a latency of 8 with only a single bucket contributing to it, the 31 to 40 bucket would be busy all the time. In other words, there could be somewhere between 31 and 40 tasks waiting on an engine to do work for the entire interval. For the smallest amount of time a single bucket could be taking up, there could be more than 80 tasks waiting on an engine somewhere between 25 and 50 percent of the interval. A flyover with detail percentages of each bucket's use can easily show this information, or we can put the bucket information for all intervals in a table that can be referred to as needed. We have the detail information readily available but we do not have to read it to see if we have an issue.

Conclusions

Once we know we want to investigate potential issues, latency by itself has reached its limits. We must perform a detailed investigation in order to determine what caused latency to be so high and see what we can do to address it – but now we know where to look and what to look for.

We now have greater ability to see latent demand on a system using IBM's SMF70Qnn variables. We have learned what these variables mean and understand how we can use the information available in them for analysis. We have learned how to take this fountain of information and reduce it to a single number so that we can see if further action is needed. We already know that to take action requires more work – but latency has shown us where and how to start.

References

Loboz, C., Lee, S., & Yuan, J. (2009). How do you measure and analyze 100,000 servers - daily? *Proceedings of the Computer Measurement Group's 2009 International Conference.*

Watson, C. (2007). *Cheryl Watson's Tuning Letter 2007 No. 3.*