

# A Model Based Approach to Predict Batch Performance & Validate Night Batch Window

Manoranjan Dash  
Infosys Technologies Ltd  
Bangalore  
Manoranjan\_Dash@infosys.com

## ABSTRACT

Is my night batch window enough to accommodate the business end of day (EOD) activities? What kind of architectural decisions or business initiatives should one take? What would be the hardware requirement? These are some difficult questions which are not very straight forward to answer. Consolidation of business in recent times, along with high business growth, has lead to huge data churn during the night batch window which demands a proactive mechanism to answer these questions. This paper presents a model based approach to predict batch performance that helps in validating the night batch window.

## 1. INTRODUCTION

With the advent of local business models following the global standards, every single enterprise wants its applications to be available earlier in the morning and later at night. That results in a squeezed night batch window. Continuity of business, disaster recovery, and backups may squeeze it further. To add to the woe, excessive high growth in business, due to consolidation, makes the night batch window even more vulnerable to time constraints. Hence the growing demand of proactive evaluation of batch performance during off-business hours through informative business/ software architectural decisions cannot be ruled out. This paper outlines a case study where one of the largest supermarket chains wanted to validate whether the night batch window, as decided by business requirements, is enough for IT to process all the night batch tasks. The validation was done through a model based approach to predict individual batch performance and then a critical path analysis to validate the end-to-end chain. To standardize for a general audience, retail domain specific terminologies have been avoided in this paper wherever possible.

## 2. BACKGROUND

### 2.1. Application overview

Figure 1 depicts a high level view of the enterprise applications. The legacy applications are used by depot and store systems to record daily transactions that happen during day time. During the night batch window, data is moved to a central system where consolidation and data massaging is done by several batch processes. The consolidated and massaged data is then uploaded to the data warehouse system through an ETL process. Finally scheduled reports are generated and sent to respective parties for analysis before the next day activity starts. This is called “day 1 for 2 reporting” i.e. the transactions that happened on day 1 are

processed during night to be available for reporting on Day 2. This enables the customer to track the stock position at depots and stores more effectively for better accounting of stock and to minimize stock loss.

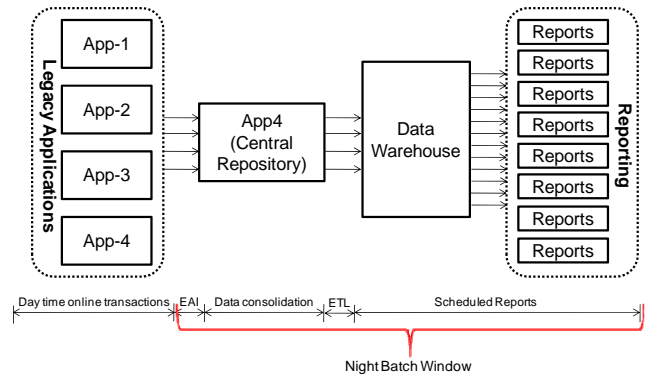


Figure 1: Sample application flow diagram

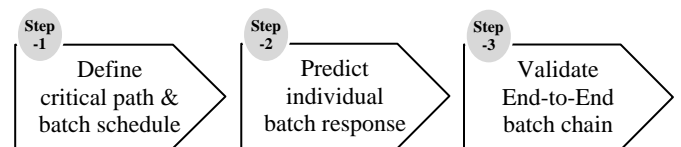
### 2.2. Objective

The objective of the exercise was to:

Validate whether the night batch window is enough for end-to-end chain of batch executions (In other words, a feasibility study of “day 1 for 2 reporting”)

## 3. APPROACH & METHODOLOGY

At a high level the following steps are performed to validate the night batch window performance.



Subsequent sections elaborate each step

Section 4: Talks about the critical path and batch schedule.

Section 5: Talks about the batch performance prediction. The critical path may have many components/ layers e.g. EAI, batch, ETL, reporting etc. Only batch performance prediction is covered in this paper.

Section 6: Talks about how to validate the night window performance once the critical path is defined, the batch schedule is drawn and the response time prediction model is ready.

### Disclaimer:

The content presented in this paper is purely based on my experience and it may not be a standard approach for all possible scenarios

## 4. DEFINE CRITICAL PATH

A simple definition for the critical path would be “A series of dependent batch jobs that has a direct impact to the night batch window and may affect the availability of the online application”. There can be multiple independent chains of batch jobs. Each batch job has an elapsed time and dependencies to other batches. Through a high level batch evaluation review technique each chain can be evaluated for estimated total time based on which the longest time path should be identified. Batches lying on that path define the critical path

A primary characteristic of the critical path is that time added to a batch on the critical path will extend the night batch window by that amount. Figure 2 illustrates a sample night batch window flow with critical path. Out of path1 and path2, if path2 takes the longest time to complete and impacts the online window then it becomes critical path.

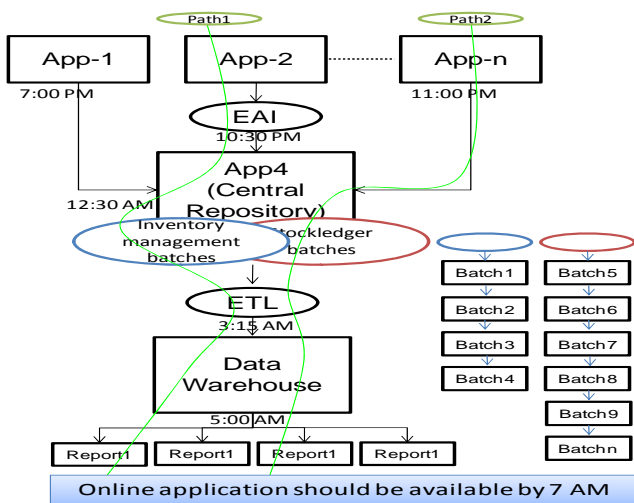


Figure 2: Critical path

The batch schedule provides an estimated timeline of all batches to be run in which system and when. A sample batch timeline is given in figure 3.

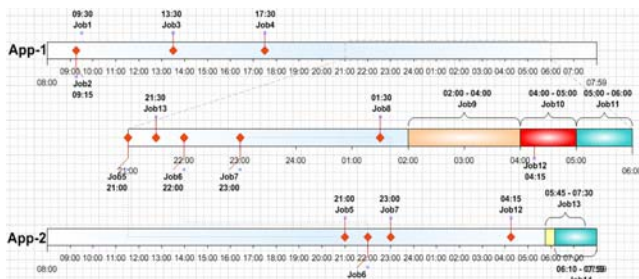


Figure 3: Sample batch timeline

For more details on the critical path please refer to IBM Redbook ‘System/390 MVS Parallel Sysplex Batch Performance’ at <http://www.redbooks.ibm.com/abstracts/sg242557.html>

*How are critical path and batch timeline information used?*

- The identified batches on the critical path in step-1 are only considered for batch performance prediction in step-2
- The defined batch timeline in step-1 is used as input while validating the end-to-end batch chain in step-3

## 5. PREDICT BATCH PERFORMANCE

### 5.1. Possible ways to predict

There can be multiple ways to predict batch performance. Please note that a single option or hybrid of them can be used to predict batch performance. Following are some of the options.

1. Do a proof of concept (PoC) of batches and extrapolate based on ‘to be’ data volume as per workload model (workload model is explained later).
2. Get benchmark numbers of the batches from core application vendor (if applicable) and extrapolate.
3. Get performance metrics from other similar projects and extrapolate as per workload model.
4. Profile the batches to get details of SQLs executed, apply base response time and overhead and estimate as per workload model.

*Which option was considered and why?*

Due to cost constraint option 1 was ruled out, and we did not get any data points for option 2 and 3. Finally option 4 was considered for prediction of batch performance.

### 5.2. Steps to predict batch performance

The approach adopted to predict batch response time is as follows. All the batches in context are written in Pro\*C with embedded Oracle SQL statements inside to carry out data retrieval.

1. Define what a simple / medium / complex SQL statement is. A combination of join and data condition can be used to define the complexity of an SQL statement.
2. Profile the new/ modified batch processes. Find approximate number of SQLs and their complexity based on batch requirements/ design or manually going through the existing Pro\*C batches and called PL/SQL procedures/ functions. Consider Addition/ deletion of SQLs due to batch customization for existing batches.
3. Find out the multiplier for SQL executions (inside the batch process). Based on logic and workload volume the same SQL may be fired multiple times for different records through loops. In the said case we had only 2 levels of looping
  - If an SQL is fired outside any loop, then it will be executed once, so the multiplier is taken as 1.
  - If an SQL is executed from the outer loop (1st loop), then it is executed based on the logic for 1st loop. E.g. in case of a parent-child record relationship the outer loop can be based on parent.
  - If an SQL is executed from the inner loop (2nd loop), then the number of executions in most of the cases is same as the workload volume. E.g. in case of a parent-child record relationship the inner loop can be based on the child.
4. Calculate base response time of different types of SQLs
  - Create some sample scripts and execute them in the test environment to measure the response time of different types of SQLs.

- Take existing SQL performance from production statspack/ awr reports and analyze them to find the base response time.
- Do PoC of a batch program in the test environment to calculate the base response time.

Based on the results from above a best judgment can be made to find out the appropriate base response time.

5. Calculate batch response time
  - Calculate the response time of all SQLs in the batch.
  - Put additional SQL overhead (During profiling we may miss out some SQLs. To counter that some additional overhead should be there).
  - Put other overheads (Pro\*C Code, File read write etc).
6. Thus the response time we get is the response time when the program is executed serially (i.e. with a single thread).
7. Decide the number of threads for the batch programs based on the batch window allotted in the batch schedule.
8. Predict the response time of the batches for the projected future workload.
9. Validate the approach with multiple checkpoints for more accuracy.
10. Identify potential bottleneck batches and recommend alternate solution.

### 5.3. Workload Model

The workload model represents how an application is expected to be used in live scenarios. It captures transaction use-cases, expected transaction volume, usage patterns, peak load, typical mix, future growth etc. This data helps in estimating the data churn that would be happening during night the batch window and helps in taking alternate architectural decisions. In this exercise we collected the workload for the batches that are executed in the night window. Some of the parameters captured are:

- Batch Name
- No of Threads
- Scheduled/ triggered
- Batch start time
- Dependency
- Data characteristics
- Functionality
- New batch/ enhancement
- Frequency of execution
- Batch end time
- Average & peak volume
- Future growth

There were multiple source systems (stores & depots) from which data is transferred during the night batch window. Data volume per day and data distribution based on transaction type from each source system was captured during workload analysis. This data also tells us the average and peak amount of data processing to be done by different batch processes during the night batch window.

#### How is workload data used?

- Accuracy of data volume and data distribution based on transaction type from all source systems is crucial as this data is fed into the batch performance prediction model.
- The multiplier for SQLs as described in section 5.2 (point 3) is derived from the workload model.

Figure 4 shows a sample of different types of data that flows during the night batch window.

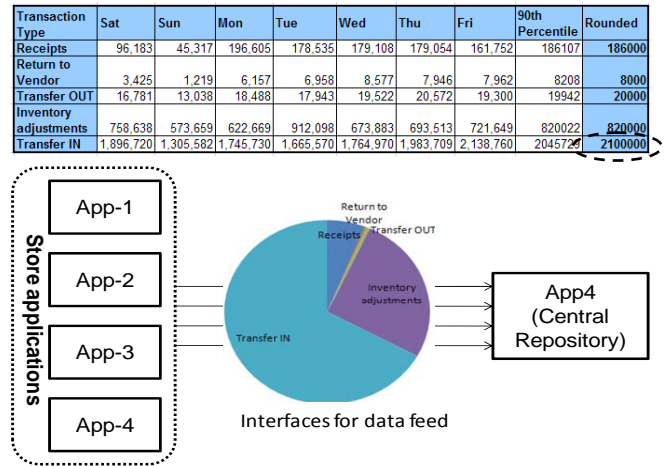


Figure 4: Workload from different applications

### 5.4. Batch performance prediction model

The Pro\*C batches used were data centric and the processing revolves around a lot of SQL statements. That's why the SQL types, their complexity, and the looping level for individual batches were captured through analysis/ profiling. The number of executions of the SQLs depends on the workload. Figure 5 shows a partial snapshot of the model.

\* For better readability the excel snapshot is broken into 2 parts

Sr. #	Program	Type	SQLs outside Loop			SQLs in Loop 1 (Main loop)		Multiplier
			Simple	Medium	Complex	Simple	Medium	
1	Batch1	select	3	0	0	5	0	14048
		insert	0	0	0	0	0	14048
		update	0	0	0	0	0	14048
		delete	0	0	0	0	0	14048
		select	3	0	0	6	0	213
2	Batch2	insert	0	0	0	2	0	213
		update	0	0	0	3	0	213
		delete	0	0	0	0	0	213
		select	3	0	0	6	0	91304
3	Batch3	insert	3	0	0	2	0	91304
		update	0	0	0	2	0	91304
		delete	0	0	0	0	0	91304
		update	0	0	0	0	0	91304

Y10 Avg		Y10 Peak		When Executed with single thread (No Parallel execution)		
Simple	Medium	Complex	Multiplier	Response Time (Sec)	Total Response Time (Sec)	Including overheads (Sec)
25	2	0	59000	582	892	1505
5	0	0	59000	103		
8	2	0	59000	207		
0	0	0	0	0		
25	4	0	11	11	7625	
5	0	0	25	25		
4	2	0	40	40		
0	0	0	2100000	2100000		
30	4	0	2100000	2100000		
5	0	0	2100000	2100000		
4	2	0	2100000	2100000		
0	0	0	2100000	2100000		

Taken from workload

Overhead  
 1) File read/ write overhead, Pro\*C, PL/SQL Code execution and looping overhead (assumed xx%-Based on PoC)  
 2) Additional SQL + profiling error margin is assumed to be xx% (After model revalidation)

Figure 5: Model to predict response time

#### Pseudo code for batch3

```

Begin
  Select col1 from tab1 where col3=c1;
  Loop --1st loop
    Select col1, col3 from tab2 where col4=c2;
    Select col3 from tab3 where col1=c3;
  Loop --2nd loop
    Select col1, col3 from tab2 where col4=c2;
  End loop;
End;

```

Let's explain the above model for batch3. SQL statement types considered were select, insert, update and delete. Before the start of any loop there were 3 SQL statements and all were simple selects. In this case we had a parent-child relationship between the

records. On an average there were 23 childs per parent (this information was collected during the workload modeling phase). There were 6 simple selects, 2 simple inserts and 2 simple updates in the outer loop (1st loop). Finally there were 30 simple selects, 4 medium selects, 5 simple inserts, 4 simple updates and 2 medium updates in the inner loop (2nd loop). For the inner loop the multiplier would be the workload volume (in this case 2,100,000) because for each record this set of SQL statements would be executed by the batch. The formula to calculate the response time is given below.

Total response time = (total number of simple selects \* simple select response time + total number of medium selects \* medium select response time + total number of complex selects \* complex selects response time)

Base response time of different SQL types is captured earlier. Thus the total response time we get is the response time of all the SQLs for the batch workload and is nothing but the sum of response time for selects, inserts, updates and deletes. On top of that a percentage overhead is added for Pro\*C/ PLSQL code execution, file read/ write and possible error due to profiling. The model created thus can be used for doing 'what-if' analysis.

### 5.5. Threading scheme

The batch schedule has to be designed based on allotted night batch window. Once the batch schedule is laid down based on the batch response time visibility for individual batches, the multi-threading scheme has to be decided for individual batches where applicable. Simultaneously considerations should also be made for optimal infrastructure usage.

The number of threads to be configured for a batch is a function of the time window allotted for the batch and 'response time of the batch assuming a single thread processes the data'. The following formula is used to estimate the number of threads.

$$(R_1 / T_W) = (N_T * S) \dots\dots\dots (1)$$

$$N_T = (R_1 / T_W) / S \dots\dots\dots (2)$$

Number of threads required ( $N_T$ ) = [Total response time assuming a single thread is used ( $R_1$ ) / Target response window ( $T_W$ )] / Linear scalability factor ( $S$ )

Where

$R_1$  = Total response time assuming a single thread is used

$T_W$  = Target response window

$N_T$  = Number of threads to configure

$S$  = Scalability factor (e.g. 0.9) #derived based on PoC

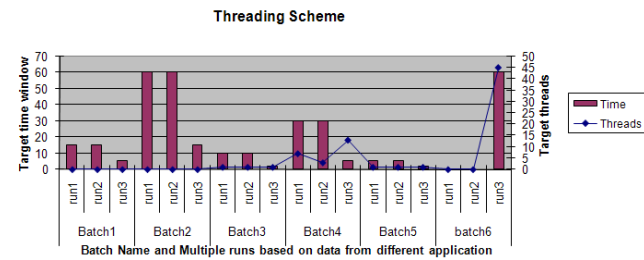


Figure 6: Threading scheme

Let's explain the threading scheme for batch 6. As per the model:

Total execution time if only a single thread processes the data = 142931 seconds = 2382 minutes

Target time window to finish the batch job as per batch schedule = 60 minutes

$$No. \text{ of threads would be } = (2382 / 60) / 0.9 = 44.1 \sim 45 \text{ threads}$$

Figure 6 represents a relation between the time window and the number of threads required to meet the defined time window. For instance as calculated above, to execute Batch6-Run3 in 60 minutes we need to configure 45 threads. Please note beyond a certain number of threads the batch may not scale. A parallel qualitative analysis should be done to identify possible bottlenecks in batch scalability.

### 5.6. Model validation

Once the model is established it should be validated through multiple checkpoints and tweaked further for better accuracy levels. In real life simulating a perfect validation scenario may not be possible due to multiple constraints. The validation criteria that should be used depend directly on what can be availed for validation. The following techniques were adopted for model validation.

- Validate the SQL profiling error margin through PoC.
- Validate the code execution overhead.
- Validate scalability factor through PoC.
- Validate the impact of SQL base response time fluctuation and re-baseline for more accurate results.
- For enhancement projects apply the model to any existing batch process and validate the response time of the model versus actual production response time.
- Direct extrapolation based on amount of data churn during the night batch window if it's not a green field application.

Maturity and better accuracy level of the model is achieved over a period of time through iterative tweaking of the model input parameters based on multiple usage of the model.

## 6. VALIDATE NIGHT BATCH WINDOW

### 6.1. Approach for batch window validation

Figure 7 represents a high level approach that was adopted for night batch window validation.

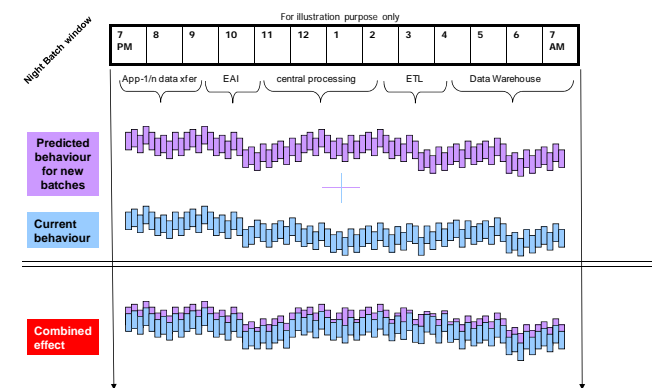


Figure 7: Approach for end to end chain validation

The end to end chain of the night batch window includes a series of batch jobs/ interfaces that must run at a particular time (e.g.

daily) and must complete in a given time frame as defined in the critical path and the batch schedule. After predicting the batch response time each distinct set is evaluated against the batch schedule to validate whether the night batch window is enough for critical path batches.

## 6.2. End to End chain validation

A very high level data flow in this context was OLTP=>EAI=>central processing=>ETL=>Data Warehouse. There were multiple validation points (e.g. 1.flat file generation from OLTP application, 2.data transfer through EAI layer, 3.data consolidation and massaging in central processing layer, 4.extract and load to the data warehouse system through ETL layer, and finally 5.report generation in data warehouse system that enables 'day 1 for 2 reporting' before start of next day online activities). Each layer was validated against the allocated time window with due consideration to possible multi-threading schemes and infrastructure requirements, and it was identified that one particular layer (layer 1 as depicted in figure 8) would take 45 minutes to 1 hour more for the processing. That means either the night batch window has to be extended by one hour in the morning, or we need to shut down the online application 1 hour earlier in the evening. Doing so may have a huge impact on business. Now at an early stage we were able to predict that the night batch window is not enough with the current architecture/design options, which is fantastic. But, is there an alternate solution? What should we do so that we achieve 'day 1 for 2 reporting' without squeezing online window?

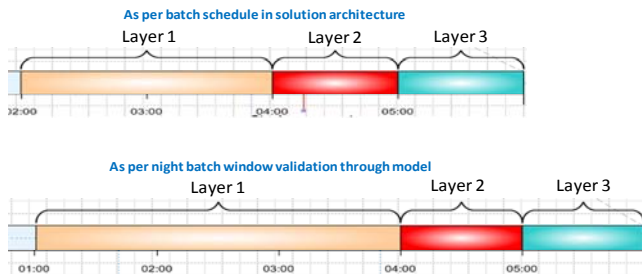


Figure 8: Night window validation result

## 6.3. Solutions proposed

A number of possible solutions mentioned below were put forward with pros and cons to finalize the best solution.

1. *Squeeze the online window by 1 hour:-* Impact on business and competitors would have an edge.
2. *'Day 1 for 2 reporting' reports to be available after 1 hour from start of day online activities:-* This was violating the basic purpose of going for 'day 1 for 2 reporting' because before start of day activity stores and depots should know the stock position and other details to take necessary actions.
3. *A major revamp of batch architecture:-* This had an impact on overall cost and timeline.
4. *Through more hardware:-* Apart from cost impact there was an uncertainty on whether some critical batches would scale beyond 70 threads or not.
5. *Optimize the critical batches:-* Some of the existing batches were profiled and an assessment was done to identify performance bottleneck areas. It was believed that after

tuning, we probably would be able to complete the batches within the night batch window. However, this solution alone is not scalable for future growth.

6. *Drip feed instead of single feed:-* As per current batch design, data from stores and depots is moved to the central system after online hours in a single go which is called 'single feed.' A drawback in this approach is it may not scale when the data volume increases. Also through capacity planning it was identified that the infrastructure usage would be very much skewed. During online hours 20% to 25% of the infrastructure would be used where as during night window, the usage would be around 80% to 90% because of intensive batch processes. Then we thought instead of moving all the data in a single go could the data be moved from stores and depots incrementally through multiple feeds ('drip feed') during online activity. After impact analysis we came to a conclusion that with minimal changes to the batch architecture/ design this can be achieved, and this is a scalable approach. Moreover, the daytime idle infrastructure would be properly utilized

Option 1 and 2 were not acceptable to business where as option 3 and 4 were not considered for cost impacts. Finally it was a unanimous decision to go ahead with both 5 and 6 for better performance, scalability and optimal resource usage.

## 7. CONCLUSION

Generating a prediction model that delivers results with good accuracy is a complex task. Many a times I have seen people asking why do we need to create a prediction model in the first place. If there is a problem, add more hardware, and the batch performance would increase, and the night batch window can be achieved. To defend, that's not always true. What happens if your batch is not designed to be multi threaded? What happens if the batch does not scale beyond a certain number of threads due to the batch architecture? What happens if at a later stage you realize that the resource utilization during the night window is skewed? What happens if you have to change the batch architecture at a later stage? It would have a huge impact on business in terms of cost, timeline, time-to-market etc. So, it is strongly recommended to validate the night batch window performance proactively through the batch response time prediction model which helps in making proper business/ design decisions at an early stage of the performance driven software development life cycle.

## 8. REFERENCES

- System/390 MVS Parallel Sysplex Batch Performance – IBM redbooks
- Batch window management using critical path analysis by Linwood Merritt
- More Batch, Less Time – DB2 Regional User Group Tour 2008
- An analytic performance model of a multi-programmed batch-timeshared computer by John E. Neilson
- Reducing the batch window to start EURO processing by Neil McMenemy
- Batch tuning experiences- Things that go bump in the night by John Shuman
- Modeling batch response by Glen Becker & Charlie Winston