Deadline Danger Detection Solutions for an Industrial Batch System using Tivoli Workload Scheduler for z/OS

Emil Ernstsen, Peter Borch Greve

Abstract

The banking industry contains large batch systems, including many jobs to be run. This leads to scheduling and monitoring being essential techniques to improve these batch systems. This paper goes in-depth with monitoring of a batch system provided by the danish it company, BEC. BEC uses the Tivoli workload scheduler for z/OS, in order to schedule their batch days. We, therefore, seek to provide insights into TWS and the runs of batch days by creating the tool D3P. This tool is able to retrospectively analyze and give insights into the completed batches, as well as providing an overview of historical data for each individual job. Through this work we present the considerations that went into designing this tool, as well as the final results of experiments testing multiple slightly different solutions. Through these experiments we find that estimating execution times of individual jobs can be difficult as a wide variety of contexts can impact the execution time immensely. We also found that using our two IBM-inspired methods for calculating the latest starts of jobs provided equal results, but there is a potential for the D3P-method to provide better results given more deadline jobs and longer critical paths.

I. Introduction

In modern day banking and financial industries, batch systems play an essential role. This is a result of the immense amount of data that needs to be processed on a daily basis [1]. However, the size of these batch systems can make it difficult to locate and solve a problem, when one occurs in the system. This is what this paper seeks to address. We were given a case related to this by the Danish financial IT company BEC [2], who this project is made in collaboration with. Furthermore, we seek to provide value to the company BEC by looking further into the case given to us and thereby producing a piece of software, which is able to provide insight into BEC's batch system. We call the software system Dangerous Deadline Detection Program (D3P). D3P should have some sort of user interface that the BEC employees are able to interact with in order to gain this insight. D3P should also provide the tracking that results in a better understanding of what the problem with a job is or which context might result in the job being late. For D3P we also need some way to monitor if the batch is progressing acceptably. For this we present the deadline detection solution methods, which we call the *D3P-method* and *the IBM partial re-implementation*, both inspired by the already existing solution made by IBM, but modified to fit into BEC's system. The D3P method was developed to see if it could provide better tracking than the IBM partial re-implementation.

A. Case

The case was given to us by BEC, and insights into their system was found in an earlier project containing classified information [3]. The case focuses on BEC's use of IBM Workload Tivoli Scheduler for z/OS, as some of their batch jobs might finish later than their deadline. Therefore, we seek to produce a tool that will be able to monitor whether the batch progresses acceptably. For this tool we will compare two solutions to see which is best for monitoring how the batch progresses. We will, alongside the IBM solution, also present our own D3P solution, which is inspired by the IBM solution. This tool should, beside being able to monitor batch progress, also be able to provide profound insight into the system, such as into the history of the jobs. Throughout the project we had meetings with BEC to regularly verify whether the tool provides further insight into their system. Furthermore, we used these meetings to find what new insights might be needed in order to provide as much value as possible.

B. Related Work

As this paper focuses on a somewhat niche area, batch system monitoring and tracking in an industrial setting, the work related to this is not directly related but do share some aspects in the problem or the solution. The authors in [4] present batch systems as large and complex with little to no end-to-end transparency. They present their "if-what analysis" as a tool to compute the impact of a certain change in the system, whereas we seek to get insight into the system to locate what might need to be changed. In [5] they present a web based job monitoring and group-and-user accounting tool for the LSF Batch System. This is also somewhat different from what we want to achieve, as we do not necessarily need a web based tool, neither do we seek to make a tool for group-and-user accounting regarding the batch system at BEC. They also seek to be an online tool, which in our case would be a goal for future versions but not a necessity for this project. We rather seek to show a proof of concept regarding the live features, as well as retrospectively provide insight into the batch system of BEC. The

author in [6] presents the tool BOSS, which is a tool made for batch job monitoring and book keeping. This tool was made to accommodate the large amount of jobs needed for large physics experiments, where it is able to accommodate the different types of jobs that might be needed in this field. Here the job type is defined by the parameters selected for monitoring, and a set of executables that contain the algorithms to determine the values of the parameters. This is more or less related to what we want to address in this paper. This is the case where we, such as they did, wish to create a job monitoring system able to monitor jobs regarding an area of interest. BOSS was a tool for monitoring physics experiment jobs, whereas our tool will be able to monitor the batch system provided by BEC. From the paper about BOSS we see that a tool for monitoring jobs can deliver value regarding large batch systems as to gain insight into what is happening within the batch system.

C. Outline

Section II presents the Tivoli Workload Scheduler for z/OS. This is the heart of BEC's daily batch operations. Section III-A presents how IBM is currently handling their deadline danger detection and how a critical path works. In Section III-C we present our modified solution to how we could calculate the critical path.

II. Tivoli Workload Scheduler for z/OS

BEC uses Tivoli Workload Scheduler for z/OS (TWS) [7], which is at the heart of their daily batch operation. This means that a complete change of scheduler is hard to pull off due to how ingrained it is in their daily operations and employees alike. From this conclusion it is clear that to avoid the costly and risky change of scheduler, a solution that can interface with TWS is more desirable in the context of BEC. As there is no intent to replace TWS, we will need to properly understand the behavior that it exhibits in order to employ the proper monitoring and analyzing strategies. Therefore, we will focus on the aspects of TWS that are important in the context of monitoring and analyzing done by D3P.

To do this we will start by defining the components that TWS uses:

a) ***Job****:* A job is a representation for the program that must be executed and thus internally holds the instructions. However, on the scope of the batch execution, the internal instruction set, data references, etc. is of no importance. However, what does exert an impact is the expected execution time, actual execution time, and the historical development of the execution time of the individual job. Thus, these are the attributes that we seek to monitor and provide feedback on suboptimal tendencies that may occur.

A job can be uniquely identified by a 3-tuple (a_id, ia_time, $opno$) Where a_id is the application id, ia_time is the input arrival time and $opno$ is the operation number that the job has in its application. Although these three attributes are enough to uniquely identify jobs we opt to use a 4-tuple which in addition includes the job name: (job_name, a_id, ia_time, $opno$). The reason for this is that BEC has chosen to additionally group jobs into batchgroups and the jobs associated department, both of which are defined in their naming convention on the job name. Although, this has no inherent differences for the user of D3P, various groupings and search becomes easier to implement.

b) ***Application****:* An application is a collection of jobs with unique operation numbers (opno). Note here that opno has no impact on the order of which jobs, in the given application, are executed. Although, they have additionally uses, in the context of this project none of this functionality is deemed important.

c) ***Daily plan****:* The daily plan is a subset of all jobs, and is the set of jobs that are to be executed in the given batch run.

d) ***Dependencies****:* Jobs are connected by a series of dependencies that each define the last jobs only that must be finished before any given job, thus creating an order to the execution of jobs. There are two different types of dependencies, namely internal and external dependencies. All dependencies from jobs in one application to a job in another is considered external, while *almost* all dependencies from a job within an application to a job within the same application is considered internal. The exceptions here are that dependencies between jobs in the same application can be explicitly marked as external. The dependencies between jobs are, however, not static, not only because the relations between jobs can change daily, but also because there are exceptions that can invalidate any given dependency. These exceptions are as follows:

1) If either the child or parent job is not present in the daily plan
2) If the childs ia_time is strictly greater than the parents ia_time
3) if the parent and child jobs are in the same application, but the dependency is marked as an external dependency and the childs ia_time is greater than or equal to the parents ia_time

The reason for exception rule *3)* is to ensure that a dependency can be given from the first job, to execute in an application, to the last job, to execute in the same application. This is needed as there are no restrictions that all jobs in an application cannot be executed multiple times in the same batch as well as an application from a previous, unfinished, batch can be executed at the same day as current batch. However, that these multiple runs are not executed concurrently is a desired feature, as the jobs within the application would most likely work on the same areas of memory, likely leading to race conditions.

e) ***Deadlines****:* A job can also be associated with a deadline, although it is not required. Deadlines are always specified with an offset in days from the ia_time date and a time specifying the time on the calculated day that the deadline is set.

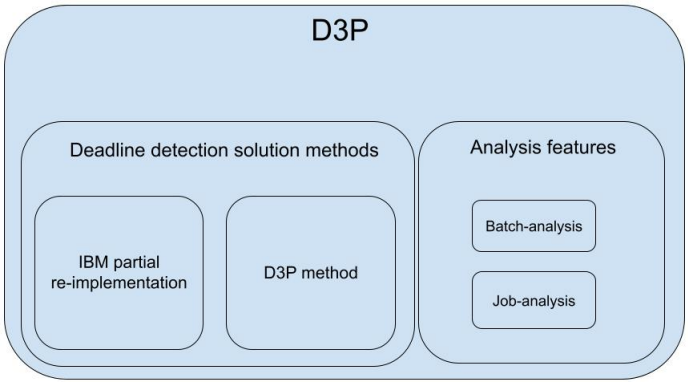

Fig. 1: D3P components

III. Deadline Delay Detection Solutions

In this section we present the various methods used to determine when a warning will be thrown. We have three methods we look into that are all very similar but do distinguish themselves in some key areas, which will be the main focus of this section. However, to properly understand how these methods work, we will first explain some of the underlying concepts; latest start, critical network and critical path which all three methods are build on.

a) ***Latest start****:* The major pillar behind these solutions is the concept of a latest start. The latest start is, as implied by the name, the latest a job can start before it is certain that a deadline job, which is dependent on this job, will not be able to meet the deadline. Note here that this means not all jobs will necessarily have a latest start which can be calculated, as it is not certain that either job or any (transitive) child job has a deadline.

The latest start is calculated by starting from a deadline job, then the latest start (LS) is equal to the deadline minus the expected execution time. Then recursively for all children C we calculate the LS as shown in equation 1.

$$LS = parent\_LS - expected\_execution\_time \tag{1}$$

b) ***Critical network****:* The critical network (CN), see Figure 2, is given for all deadline jobs by the set of all jobs from which a latest start is calculated from that deadline job. This means that the CN is defined as all jobs that, exceeding their latest start, will result in the job not being able to meet the deadline.

c) ***Critical path****:* A critical path is a subset of the CN, where for each level of predecessors in the CN, the process chooses the most critical one and includes it in the critical path. The most critical one is the predecessor with the latest end time.

d) ***The three methods****:* In the following sections we will describe the three methods for looking at latest start that we have explored. First of, we will describe the method that is an existing part of TWS, as this has been the major inspirational feature for the following methods. As will become clear through the following sections, however, we have no option to test this on the batches BEC runs. This leads us to look into the two methods we have created, which both are a part of the D3P tool, as can be seen in Figure 1. Here we have the IBM partial re-implementation, which is made for the purpose of mimicking, as close as possible, the method used in the existing TWS solution. Lastly, we will present the slightly modified D3P method and how this intends to improve how latest starts are used.

A. Features in the IBM Solution (IBM-original)

The existing solution for latest start monitoring, build into TWS, contains a wide variety of features that can help monitor *and* assist in correcting issues and delays that can result in missed deadlines. Especially the possibility to actively make changes to the daily plan, priorities, and more at runtime, gives the existing IBM solution incredible power as a tool. However, as mentioned earlier, we are not able to test nor observe this solution in action, as BEC does not utilize it. As such even though they can hold great potential, all the features that require being able to make changes or input to the system in any way is collected in a feature as ”automation”, as seen in Table I. In this table we also see one of the reasons the existing IBM solution is not used by BEC; namely the use for knowing the exact (or somewhat precise) execution time of each job. However, obtaining this knowledge can be a daunting task, and to our knowledge currently BEC does not have a method nor the possibility to obtain these.

Even though we do not know the best way to go about obtaining the required knowledge of the execution times, it is clear that the task is relatively large. This is not only due to the sheer amount of unique jobs that can be part of each batch, but also as each unique job can run in a vast amount of different contexts like the amount of data the job needs to process, how many concurrent jobs that share its special resources, and what time of month or year it runs. The result of this is a high barrier of entry to utilize the features, and sparks the motivation for constructing a subset of the features with a lower barrier

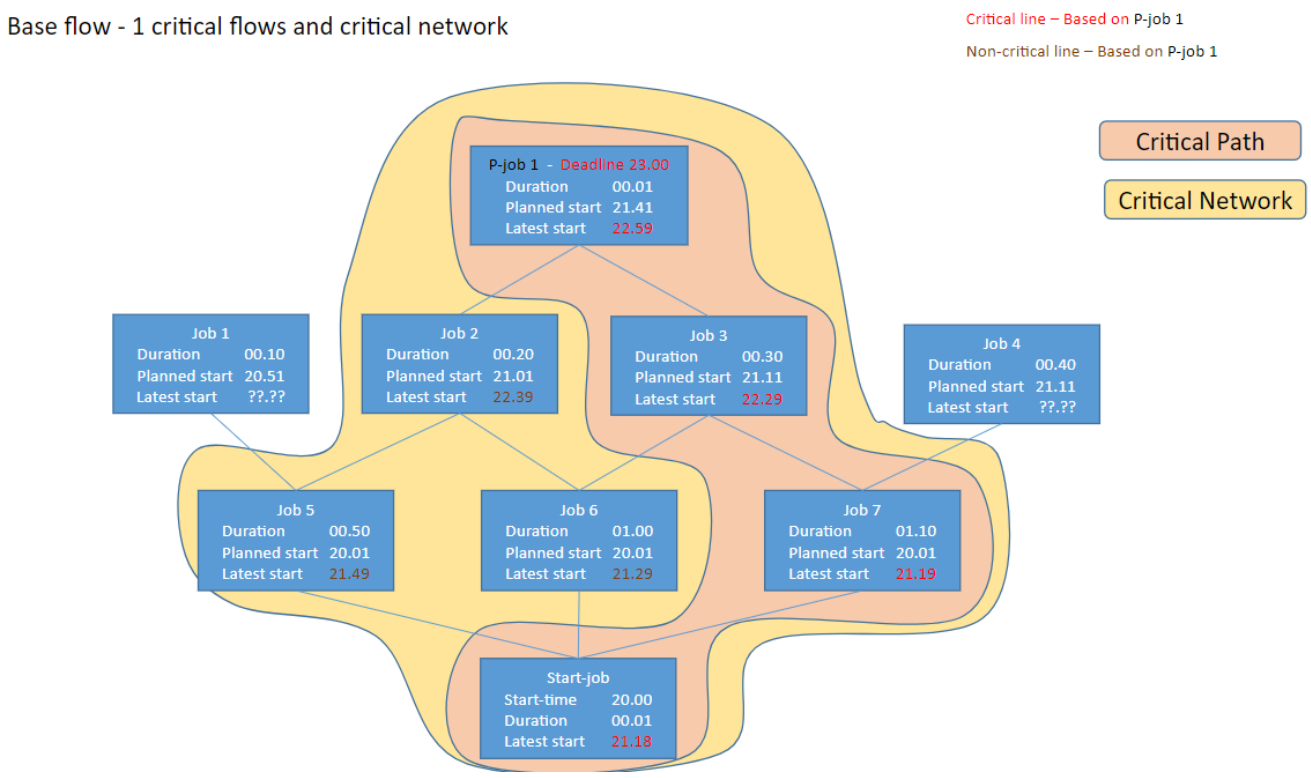

Fig. 2: Latest start example

of entry. This is what is displayed in the table as the IBM re-implementation, which is a solution we have created to mimic the functionality of latest starts and critical paths, as seen in the IBM solution.

The main monitoring feature we wish to mimic is to continuously monitor that no job exceeds its latest start, as this gives an early warning system when jobs will exceed their deadline earlier in the batch. Having this information earlier can help provide the insight to either fix or mend the issue by changing the priorities or other measures.

a) Example of latest starts as done by IBM: To give an example of how the latest starts are both calculated and used, we have a small fictional set of jobs as part of the daily plan, as seen in Figure 2. In this figure we see boxes representing individual jobs and lines between them indicating dependencies. The first job to be executed will be the start job located in the bottom of the figure, as the remaining jobs are either directly or transitively dependent on this job. On the figure we have a single job with a deadline **P-job 1**. Here we see that the latest start of each job in the critical network is calculated, as explained in Section III, recursively with root at the deadline job. For the IBM solution a warning will then be thrown when a job with a latest start has not yet started by that time, for each deadline job that is transitively dependent on this job.

B. IBM Partial Re-implementation (IBM-re-imp)

Although the re-implementation mimics the original IBM solution, not having access to precise execution times is a limiting factor that we have to work around. To achieve similar results, we seek to work with estimates of expected execution time. These estimates are calculated from historical data, but as they are estimates we need to acknowledge the uncertainty in these and both try to minimize it, as well as express the uncertainty to the user. We choose to express this uncertainty through our warning system, such that we will give two different levels of warnings when monitoring whether jobs exceed their latest starts. As such, a given job J with an estimated execution time of X minutes with an uncertainty of Y minutes, will calculate all latest starts using only X. However, when checking if a job has exceeded its latest start, a low certainty level warning (yellow warning) will be thrown according to equation 2, similarly the high certainty level warning (red warning) will be thrown according to equation 3.

$$yellow\_warning: \ current\_time > latest\_start(J) - Y \tag{2}$$

$$red\_warning: \ current\_time > latest\_start(J) + Y \tag{3}$$

C. Dangerous Deadline Detection Program method (D3P-Method)

The D3P method as a solution is, like the IBM-re-imp, inspired by the IBM solution. Where the D3P method differs compared to the IBM solution is regarding the ***Latest Start***, where both the IBM and IBM-re-imp solutions only accommodate one latest start per job even in the case of overlapping critical networks.

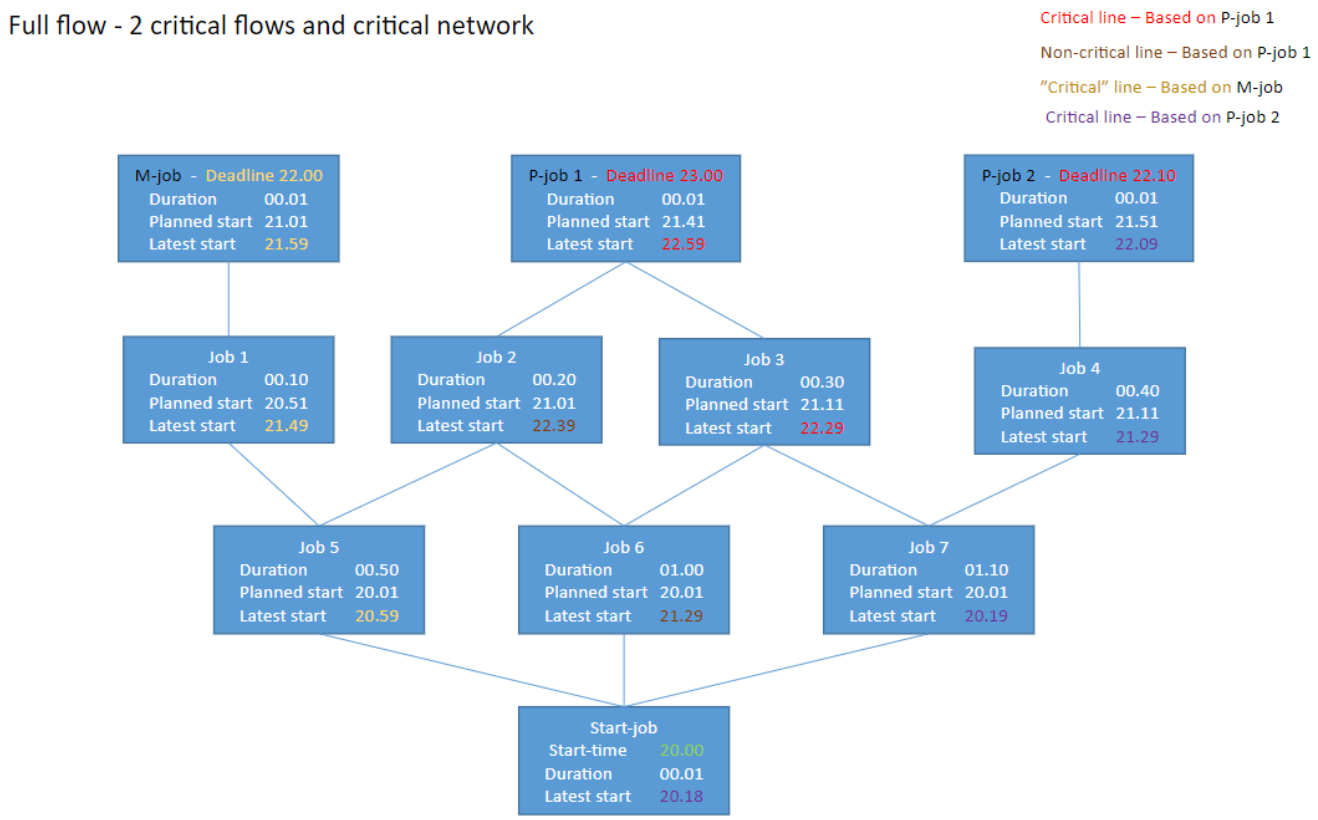

Fig. 3: Latest start example

Feature	IBM-re-imp	D3P-method	IBM-original
Can use exact job duration	✓	✓	✓
Can use historic data to estimate job durations	✓	✓	×
Warning thrown when job exceeds latest start	✓	✓	✓
Warning thrown when job running longer than exact job duration	×	×	✓
Warning thrown when job ends in error	✓	✓	✓
Multiple latest starts	×	✓	×
Automation	×	×	✓

TABLE I: Feature Table

To understand the impact of having only a single latest start per job versus having multiple, we will take the example from Figure 2 and expand it so that we will have overlapping critical networks from different deadline jobs. This expansion can be seen in Figure 3. In this figure we have three jobs with deadlines: M-job, P-job 1, P-job 2. We can see that the critical networks from P-job 1 and P-job 2 overlaps at job 7 and the start job. However, due to the changes in deadlines and accumulated duration until this overlap, results in two different latest starts, one for each deadline job. However, in the Existing IBM solution and the IBM-re-imp, having multiple latest starts for a job is not possible. This means that a choice of what latest start to use must be made. However, no good choice can be made. The choice that is made by the existing IBM solution and the IBM-re-imp is to always choose the earliest latest start, this however comes with an edge case that is not handled. This can be seen when we look at job 7 from the figure the latest start from P-job 1 would be 21:19, while the latest start from P-job 2 is the selected value of 20:19, as marked on the figure. However, lets assume that the job starts at 20:45. This means that P-job 2 should no longer be able to meet its deadline while P-job 1 can. Although this distinguishing between which job is going to be late and which will not is not possible from the information of a single latest start. This means that as the earliest latest start is selected both P-job 1 and P-job 2 would get a warning indicating that none of the jobs will make it by the deadline even though this is not the case.

Given that the option to have multiple latest start on each job is a feature which is missing even in IBM-re-imp, we will in this paper conduct experiments on the impact of being able to have multiple latest start for each job for BEC. This is also why we have decided to create the IBM-re-imp, since this can be used as a reference point. Furthermore, as both D3P-method and IBM-re-imp work with estimates and uncertainty, any changes in result, if any, can be attributed to this change in having the option for multiple latest starts on each job.

Day	1	2	3	4	5	6	7	8	9	10	11	12
Date	2021-03-22	2021-03-23	2021-03-24	2021-03-25	2021-03-29	2021-03-30	2021-03-31	2021-04-05	2021-04-06	2021-04-07	2021-04-08	2021-04-12
Jobs in batch	48048	48672	48686	17857	50435	51395	40879	18164	49807	49799	32222	41251
Deadline Jobs	400	401	400	114	398	398	335	184	398	398	283	308

TABLE II: Amount of jobs in batch day and amount of deadline jobs in batch day

Day	1	2	3	4	5	6	7	8	9	10	11	12
Deadline jobs w. at least one overlap in CP	48	48	50	26	50	48	4	30	48	50	4	22
Deadline jobs avg overlap in CP	0.7	0.7	0.7	2.2	0.7	0.7	0.1	1.4	0.7	0.7	0.0	0.1
Deadline jobs avg overlap in CP w. at least one	5.6	5.6	5.4	9.5	5.4	5.6	1	8.3	5.6	5.4	1	1
Max overlap in CP	14	14	14	14	14	14	1	14	14	14	1	1

TABLE III: Critical path statistics

IV. Experiments

As the two solution methods have now been presented, we can begin to compare them. Afterwards, we conduct experiments where the performance of the two will be compared, as well as experiments to gain more in-depth knowledge of the data on a particular batch day.

A. Setup

In order to compare the two solutions, we set up experiments to see how the two fare compared to each other. Depending on the experiment, different parameters were set up to either measure the performance of the two solutions or to get a better overview of the data. The parameters can be split into two categories, *warnings* and *critical path*. The *warnings* parameters will measure how each solution fares regarding the handling of warnings, as explained throughout Section III. The *critical path* parameters are primarily used to get a better understanding of the data regarding a specific day. As we focus on detection of warnings, we therefore experiment with different functions for calculating the *Estimation Method*, which is the method for when a warning should be detected. For this estimation method we will run experiments with three different functions.

The first one will calculate a simple mean function of standard deviations of all historic runs for each specific job, we call this "***Mean***". The second one will calculate a standard deviation of the historic standard deviations, we call this "***StandardDev***". The third will calculate a mean of the standard deviation of standard deviations, we call this "***MeanStandardDev***". The equations for calculating these can be seen in Equations 4 through 7, where σ denotes the standard deviation of a given set.

$$Mean(Job) = \sum (hitorical\_job\_durations(Job)) \ / \ historical\_job\_durations(Job).len \tag{4}$$

$$StandardDevSet(Job) = \{\sigma(x) \mid x \ = \ all\_historic\_job\_durations \ \textbf{and} \ x.len \ = \ historical\_durations(Job).len\} \tag{5}$$

$$MeanStandardDev(Job) = \sum (StandardDevSet(Job)) \ / \ StandardDevSet(Job).len \tag{6}$$

$$StandardDev(Job) = \sigma(StandardDevSet(Job)) \tag{7}$$

Table II shows the amount of jobs that a given batch day contains alongside the amount of deadline jobs for that given day. The amount of jobs and deadline jobs will not differ for the batch day throughout all the experiments.

B. Critical Path

In Table III, we see the critical path statistics. These statistics are for each day and should be kept in mind when we compare the D3P method and the IBM-re-imp. The critical path statistics are important in this matter, as the D3P method seeks to improve the results by handling the edge case, explained in Section III-C, better than the IBM-re-imp. The critical path statistics describe how intertwined the deadline jobs are in their critical paths. In the table, *CP* stands for critical path. *Deadline jobs w. at least one overlap in CP* describes how many deadline jobs has a member in their critical path which they share with another deadline jobs critical path. *Deadline jobs avg overlap in CP* shows the average amount of overlaps in the critical path for all deadline jobs, whereas *Deadline jobs avg overlap in CP w. at least one* shows the average overlap for deadline jobs that has at least one overlap. *Max overlap in CP* shows the maximum amount of overlap for a single deadline job on the given day.

Solution method	D3P-method			IBM-re-imp		
Estimation method	Mean	StandardDev	MeanStandardDev	Mean	StandardDev	MeanStandardDev
Day	6	6	6	6	6	6
Amount of jobs w. red warning	12	12	11	12	12	11
Amount of jobs w. yellow warning	42	285	289	42	287	291
Amount of correct red	12	10	8	12	10	8
Amount of wrong red	0	2	3	0	2	3
Percent correct red	100%	83%	73%	100%	83%	73%
Amount of correct yellow	18	204	207	18	204	207
Amount of wrong yellow	24	81	82	24	83	84
Percent correct yellow	43%	72%	72%	43%	71%	71%
Avg time red was thrown before deadline	0	204	210	0	204	210
Avg time yellow was thrown before deadline	123	9	11	123	22	24

TABLE IV: Estimation method comparison

Time	5 Minutes	2 Minutes	1 Minute	30 seconds
Day	6	6	6	6
Amount of jobs w. red warning	12	12	12	12
Amount of jobs w. yellow warning	42	43	44	44
Amount of correct red	12	12	12	12
Percent correct red	100%	100%	100%	100%
Amount of correct yellow	18	13	8	8
Percent correct yellow	43%	30%	18%	18%
Avg time red was thrown before deadline	0	1	0	0
Avg time yellow was thrown before deadline	123	121	119	119

TABLE V: Time interval table

C. Estimation Methods

To find the best *Estimation Method* for each individual solution, we conducted experiments where both the D3P method and the IBM-re-imp made use of each *Estimation Method*. The experiments were conducted for each solution, using every method throughout all the batch days, as presented in Table II. Table IV shows the results for the D3P method and IBM-re-imp using each method. In order to compare the results, day 6 was chosen, as this day contains the most amount of jobs. If we first take a look at the D3P method and the three estimation methods, we see that ***"Mean"*** and ***"StandardDev"*** find the largest amount of red warnings, 12, where ***"Mean"*** finds them with 100% accuracy, whereas ***"StandardDev"*** has about 83%. This indicates that ***"Mean"*** will be best for finding the amount of jobs with red warnings. What is also noteworthy is that the average time red was thrown before the deadline is remarkably larger with ***"StandardDev"*** and ***"MeanStandardDev"***, being 204 and 207 minutes, respectively, whereas it is 0 minutes with the ***"Mean"***. We speculate this to be due to the wrong red warnings. Furthermore, this table shows that ***"Mean"*** has the lowest amount of yellow warnings, 42, whereas the ***"StandardDev"*** and ***"MeanStandardDev"*** methods had 285 and 289, respectively. This might indicate that the two latter mentioned methods might be better, especially if we also take a look at the percent correct yellow, where they both have about 72% accuracy and ***"Mean"*** only has about 43%. However, even though the percent correct is better, we also see a noticeable rise in amount of wrong yellow warnings, from 24 in ***"Mean"*** up to 207 in ***"MeanStandardDev"***, and a fairly large decrease in average time yellow thrown before deadline, from 123 minutes down to 9 and 7 minutes in ***"StandardDev"*** and ***"MeanStandardDev"***. This is due to how we calculate the ***"StandardDev"***, as we aggregate the data points for jobs with similar sample sizes. These results indicate that such an aggregation of data points might not be valid and thereby invalidating the estimation methods ***"StandardDev"*** and ***"MeanStandardDev"***. There can be a multitude of reasons why this aggregation method does not work. The most likely reason being similar execution times of jobs that run equally often is not sound. It could also be that there are no correlation between job that can help determine the execution time for any individual job. Due to the uncertainty on how any aggregation could be done and that the results do not support our attempts of data aggregation, we decide to move forward with distinguishing jobs uniquely, which is only supported by the ***"Mean"*** method.

D. Query Time Intervals

When deciding how often to query TWS, we wanted to continuously be able to monitor the progress and, therefore, decided that we wanted to query with 5-minute intervals. However, we wanted to see the effect that querying more often would have. We, therefore, conducted experiments with shorter time intervals. The results can be seen in Table V. The experiments were conducted with intervals of 5 minutes, 2 minutes, 1 minute, and 30 seconds. The tendencies shown in Table V suggest that

Day	1	2	3	4	5	6	7	8	9	10	11	12
Amount of jobs w. red warning	10	11	15	4	11	12	7	6	11	12	13	6
Amount of jobs w. yellow warning	30	30	34	8	33	42	39	10	33	33	26	24
Amount of correct red	10	11	13	2	11	12	7	6	11	12	10	6
Amount of wrong red	0	0	2	2	0	0	0	0	0	0	3	0
Percent correct red	100%	100%	87%	50%	100%	100%	100%	100%	100%	100%	77%	100%
Amount of correct yellow	16	16	16	2	17	18	22	6	16	17	15	14
Amount of wrong yellow	14	14	18	6	16	24	17	4	17	16	11	10
Percent correct yellow	53%	53%	47%	25%	57%	43%	56%	60%	48%	51%	57%	58%
Avg time red was thrown before deadline	0	0	170	440	0	0	0	1	0	60	35	0
Avg time yellow was thrown before deadline	63	63	134	329	114	123	29	277	64	121	14	7
Avg time correct red was thrown before deadline	0	0	425	440	0	0	0	0	0	0	95	0
Avg time correct yellow was thrown before deadline	133	133	251	437	230	207	62	683	123	230	11	13
Avg time wrong red was thrown before deadline	0	0	0	0	0	0	0	1	0	60	0	0
Avg time wrong yellow was thrown before deadline	2	2	2	3	5	11	3	6	2	18	16	2

TABLE VI: D3P & IBM-re-imp

shorter time intervals produce worse results. This claim is supported by the results seen in the percent correct yellow, where we see 43% of the yellow warnings are correct in 5 minute intervals and drops to 30% with 2 minute intervals and further drops to 18% when having 1 minute intervals. Furthermore, it is worth noticing that the percentage does not drop further when having 30 second intervals, as we only work with minutes in D3P. The reason for the drop in percent correct yellow is likely due to some jobs with short execution time and with relatively large standard deviation according to the execution time.

Example: Job A has an expected execution time of 1 minute and a standard deviation of 1 minute. If the deadline for job A is 8:00, the latest start would be 7:59 and we would throw a yellow warning at 7:58.

With 5 minutes the warning in the example would not necessarily be thrown unless we were querying at 7:58. Shorter time intervals are, therefore, more prone to jobs with large standard deviations relative to their execution time. This claim is also supported by the decreasing average time of when yellow warnings are thrown before the deadline, as seen in Table V.

E. D3P-method vs IBM-re-imp

For this experiment we wanted to test the two solutions against each other by comparing them across different parameters. The experiment was conducted to see which method would serve as the best one to use in D3P. The experiment showed identical results, therefore, we can see the results for both methods in Table VI. The reasons why they show identical results are due to two main reasons. The first reason is that the methods are fairly identical and the difference between them being the ability to handle multiple latest starts, as shown in Table I. This ability allows D3P to handle the edge case, which was explained and exemplified in Section III-C. This leads to the second reason, which is amount of deadline jobs and the length and overlaps of the critical path. By how the D3P-method work then it should show better results when there are many deadline jobs, the critical paths overlap, and the critical paths overlap deep in the dependencies. As shown in Table II, we see that we have about 400 deadline jobs for a majority of the batch days, which does not amount to much when a batch day can contain up to more than 50,000 jobs. Furthermore, we see in Table III, that the amount of deadline jobs with overlaps is at most 50, which does not allow D3P to show its promise, as we also see that average amount of overlaps in the critical path, when a deadline job contains at least one overlap, is at max 9.5.

With these results and these considerations in mind, we argue that the method we will use in D3P should be the D3P-method. The reasoning for this is that the results for the two methods are identical, but D3P is able to handle an edge case which IBM-re-imp is not and, therefore, it should be best using the D3P-method going forward.

V. D3P TOOL

The D3P tool was made to be used by BEC. This tool was developed with the intent to provide insights into their batch systems and the individual jobs. The tool has a simple UI, as seen in Figure 4, which could be further developed. However, a simple UI is all that is needed for the proof of concept of this tool.

A. Intended Use

The intended use of the tool, that is the D3P software, is to provide insight into different aspects of the batch system that BEC has provided. This tool should provide features that make it possible for BEC to gain insights needed to solve issues within the batch system. The tool, however, is not intended to solve the issues, and is a read-only tool without the capabilities to make changes within the batch system. It is a monitoring tool that should be able to provide retrospective analysis of a given batch day's run. This retrospective analysis should provide insights into the individual jobs, delays on jobs, and the critical paths. This will be further elaborated upon in the following section.

B. D3P Features

The two main feature categories for D3P lie within the retrospective analysis and the live tracking, which we will further elaborate upon.

1) Retrospective Analysis: As we are required to work with historical data to calculate the estimated execution time used for live monitoring the batch, we also worked on some features that can help BEC gain an overview of completed batches and overall statistics for these.

a) Job insight: The first feature for retrospective is on the individual job level. Here we will display for the user the general information that is known based on the historical data, such as:

- The average execution time
- The sample size
- The standard deviation
- Max and Min execution times
- Amount of times the jobs end in error

All these metrics are to provide insight into any job, such that if a job runs longer than expected it can be evaluated in context to historical data. However, this is not perfect due to impact on these parameters, which can be caused by the specific context of any given day. Through dialogue with BEC we decided to give D3P the ability to also list each day individually to give the user the option to manually review execution times in the context of the specific day it was set to run.

b) Delays on jobs: All of the metrics listed above have to do with execution times and how the runs went. BEC, however, expressed the need to also have insight into *when* the jobs are running. More precisely, to be able to see tendencies if jobs start later in the batch than expected. To define a metric that is generic across days is difficult, as the scenario changes daily, resulting in a wide number of reasons that a job is executed early or late in the batch. To normalize this into a more comprehensive metric, we decided to look at the the completion time of jobs compared to the input arrival time of the job on that day. By averaging these durations from input arrival to completion, we will get a single number that can express if the job is run early or late in the batch.

c) Critical path insight: In addition to the *job insight* and the *delays on jobs*, we also provide insight into the critical path for all the defined deadline jobs in the system. When the user selects a deadline job in D3P, it will show the critical path. D3P is also able to display a resource scheme of jobs using processing power in intervals when neither the deadline job nor its critical network is progressing. As it is not necessarily a bad thing that there is no progress on a critical network for any given deadline job, we also provide a list of deadline jobs of which critical networks do have progress in that time interval. This is done, as whether it is an issue to not have progress on a critical network for a deadline job, is solely determined by where the resources are then being used. This is a difficult concept to define, which is why it is left up for manual review.

2) Live Tracking: One of the major selling points of the IBM-original solution, mentioned in Section III-A, is the possibility for automatically applying changes to keep a batch on time, but also all of the insights, on possibilities that can come with an early warning system. Although it is not a possibility for D3P to automatically apply any changes to priority etc, it can still give the option to do so by the information of what deadline job is in danger of exceeding the deadline. However, the data from the running batch that we are provided are extracted through software called XINFO, a built-in tool for TWS, which currently in the context for BEC does not allow to extract any data live, but only with a few hours delay which renders live tracking unusable. Although getting data from TWS live should be possible, it is not feasible for BEC to establish a pipeline for this within the given time frame.

Although the option for live tracking is not possible this time around, all the value is not lost as the batch day and step by step progression will have to be simulated to provide the insights described in Section V-B1. So by simulating the day by continuously making more and more data visible to the system as it would have been provided during the actual batch run, we can still provide a retrospective overview similar to a complete log of warnings given throughout the batch. The insight this can provide is that, should a deadline be exceeded, then it is easier to find the reason why this happened after the completion of the batch. This can be a hard task as the reason for this could have happened anywhere in the critical network, be it delays, jobs running longer than expected, or even jobs ending in error. Hopefully, a warning telling you much earlier in the batch that the deadline at this point can no longer be met, can substantially narrow down the search for the root cause of the exceeded deadline. As such the live tracking feature can also be used retrospectively.

VI. Conclusion

Throughout this paper we explored the batch system provided by BEC and the Tivoli Workload Scheduler for z/OS, a tool BEC uses for scheduling their batch jobs throughout their batch day. We then looked into deadline danger detection solutions, where three different solutions were presented and the features each solution support were presented in Table I. Through experiments we found several key results. Firstly, we found that the only valid estimation method we made was ***"Mean"***. Secondly, we found that shorter time intervals for querying did not provide better results. Thirdly, we found that the D3P-method and IBM-re-imp provided the same results given the current data and amount of deadline jobs. However, D3P can potentially provide better results as it is able to accommodate an edge case. After the experiments we presented the D3P

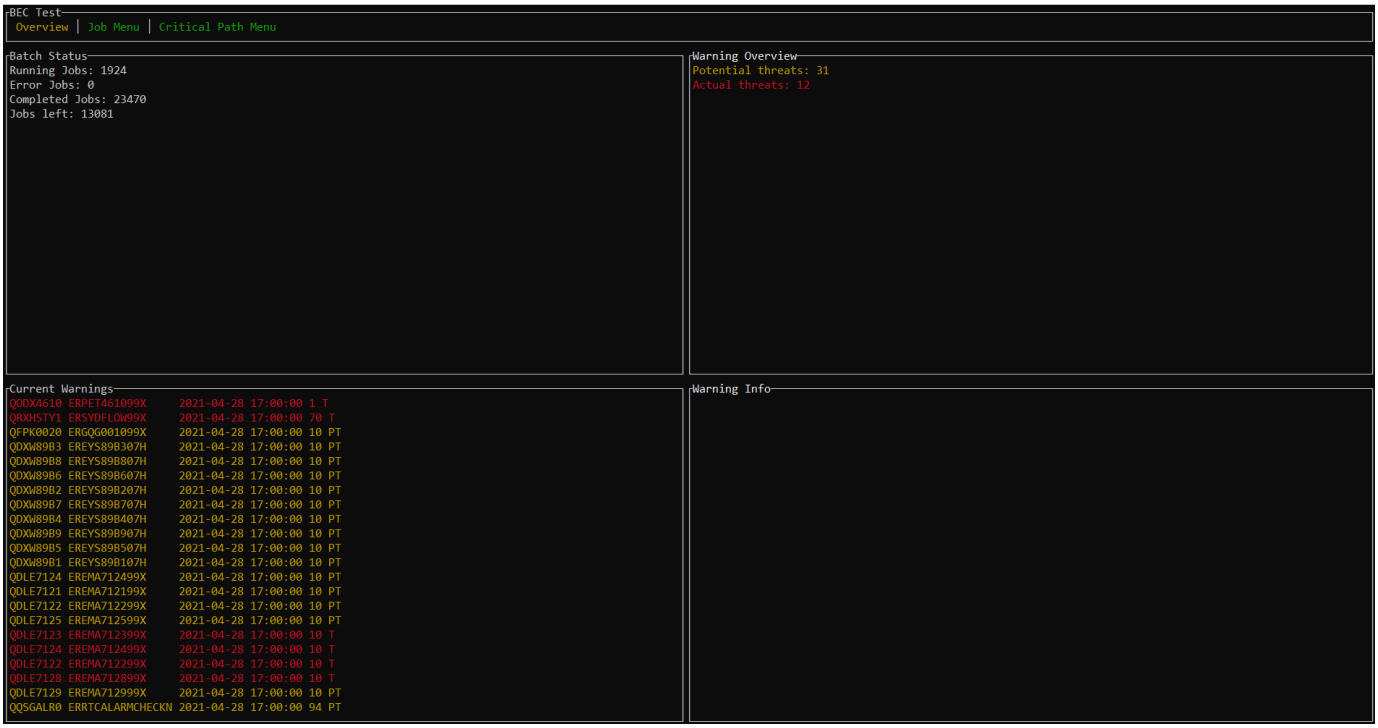

Fig. 4: The D3P tool

tool, which is the tool that should be able to provide insights to BEC. The tool is, as of now, able to provide retrospective analysis and also a kind of retrospective live tracking.

VII. Future Work

This paper presents the tool D3P, the underlying considerations, and the circumstances around it. However, D3P, as of now, is not a finished tool, but is a tool that can provide some benefits and be used as a proof of concept for further development. D3P can with more development provide even more insights and value to BEC, some of which is required for D3P being a commercially viable product. Furthermore, more experiments could provide a deeper understanding of how to adjust D3P as to get better results.

A. Livetracking

As mentioned in Section V-B2, live tracking is not supported as of now in D3P, as it is not possible to get live data from TWS. However, going forward it could provide essential value to have live tracking in D3P, as it would allow live monitoring of the batch system. This would make D3P able to throw live warnings, which BEC would be able to react to and thereby adjust jobs, such that their deadline is not exceeded as projected by D3P. As shown in Table I IBM-original has automation as a feature which the IBM-re-imp and D3P-method do not. However, with live tracking it would be possible to accommodate some of the problems that automation solve with live tracking, as D3P would now be able to provide BEC with the feedback needed, so they can take action to resolve issues manually.

B. Monitor jobs

As of now D3P uses the deadline jobs provided by TWS to throw warnings and it is an essential part of D3P. However, going forward these deadline jobs should not be a part of D3P. Instead a feature we call ”monitor jobs” should be introduced. This feature should allow for jobs to be set as monitor jobs, and be given a deadline which will not affect the run of the job but only the monitoring of it. This would allow BEC to set deadlines on problematic jobs that do not have a set deadline in TWS and allow for D3P to throw warnings according to these jobs rather than the deadline jobs from TWS. This feature would allow for more customized and in-depth monitoring of jobs in D3P.

C. No progress jobs

Going forward D3P should also be able to show what we call ”no progress jobs”. These jobs are defined as jobs that have dependencies which are all finished with their execution, but do not start their own execution. This would allow BEC to more easily find and identify problematic jobs or behavior within their batch system.

D. Time intervals

As mention in Section IV-D, we conducted experiments with the time intervals to see if it would benefit D3P to use shorter time intervals when querying. The results suggested that a shorter time interval would not provide better results. However, longer time intervals was not explored in these experiments and, therefore, going forward it could be of interest to see if longer time intervals would provide better results.

E. Better estimation method

In Section IV-A we presented three estimation methods and through the experiments in Section IV-C we found that the only viable estimation method to use, out of these three, is ***"Mean"***. Therefore, going forward, a better estimation method could be developed to better accommodate the different context the jobs run in. These contexts seem like they can more accurately be defined by which rules that define what days any given job run. However, as these rules are written in a language, used only internally by TWS, a parser would need to be constructed. Creating a parser for this language should be doable as the semantics are well defined by IBM. This would, as an example, allow us to better estimate a primo or ultimo job run.

REFERENCES

[1] R. T. Bedeley and L. S. Iyer, "Big data opportunities and challenges: the case of banking industry," *Proceedings of the Southern Association for Information Systems Conference*, vol. 1., April 2014.

[2] BEC, "Om bec," 2021. [Online]. Available: https://www.bec.dk/om-bec/

[3] E. Ernstsen and P. B. Greve, "Data analyses of scheduling of an industrial batch system," 2020.

[4] S. Batthala and et al., "If-what analysis to manage batch systems," *International Conference on Communication Systems and Networks (COMSNETS). IEEE*, vol. 8th, 2016.

[5] S. Sarkar and S. Taneja, "A job monitoring and accounting tool for the lsf batch system," *Journal of Physics: Conference Series*, vol. 331, no. 7, 2011.

[6] C. Grandi, "Boss: a tool for batch job monitoring and book-keeping," *Conference for Computing in High-Energy and Nuclear Physics*, March 2003.

[7] I. Corp., *IBM Tivoli Workload Scheduler for z/OS: Managing the Workload*, 9th ed. IBM, April 2017.