Using Analytics to Understand and Control Batch Operations - A Case Study

Maitreya Natu
Tata Research Development and Design Centre,
Pune, India.
Email: maitreya.natu@tcs.com

Vaishali Sadaphal
Tata Research Development and Design Centre,
Pune, India.
Email: vaishali.sadaphal@tcs.com

***Abstract*—This paper focuses on batch environment of an enterprise. We use various data sources such as dependency data, alert and ticket data, and run history data of the jobs in batch systems to build a model of batch operations. We enrich the model with various attributes derived by graph analysis of dependency data, time-series analysis of execution history of jobs, and text and statistical analysis of alerts data. We present reactive and proactive approaches for improving stability of batch operations using this model and demonstrate their effectiveness through a real-world case-study.**

I. Introduction

Batch systems play a crucial role in the banking and financial industries. While the literature contains several studies for effectively managing the online transactional systems, the less glamorous, but equally important, offline batch processing environments have received little attention. In this paper, we focus on such batch environments of an enterprise. We applied these concepts in the batch system of a major bank in the US. We demonstrate the proposed ideas with examples from this real-world case-study.

Batch processing environment for an enterprise is characterized by the following facts:

(1) A batch includes a set of *jobs* with *precedence relations*. Precedence relation determines the order of job invocations. In event that a job has more than one predecessors, then it can be initiated only after the completion of *all* of its predecessor jobs. (2) Batch jobs are grouped into *streams* and *sub-streams*. Each stream refers to a business function. (3) Business requirements enforce a set of constraints on: the earliest time when a stream can start, and the latest time by which all the business critical jobs within a stream must complete.

Managing these systems is challenging because of the inherent scale and complexity. First, for most enterprises, the batch size (in terms of number of jobs and precedence relations) is quite large. For example, a nightly batch in the presented case-study involves approximately 200,000 jobs serving 50 different business streams. Each stream produces tens of thousands of alerts every month. Second, the number of precedence relationships between jobs are very large. Jobs are inter-related with complex inter-dependencies that often span across geographies, time-zones, and business streams and sub-streams. Each business stream is often managed by a separate team. Because of this, most often the support teams lack an end-to-end understanding of complete batch operations.

However, the good news is that the batch operations are captured in sufficient details using various monitoring tools. Various data sets are available to better understand and control operations. Some frequently available information about batch operations is dependency data, run history of jobs, alerts generated for jobs, and user-generated issues. Secondly, batch operations are more deterministic than transactional systems. There is a fixed set of jobs and deterministic precedence relationships and schedules.

In order to better understand and manage these systems, a critical requirement is the construction of a comprehensive end-to-end model that acts as a blueprint of the operations. In this paper, we first propose our approach to construct this model. The model has following key properties:

- *Hierarchical modeling:* The model captures the hierarchy of business functions, to applications, to infrastructure.
- *Attributes from a variety of data sources:* We explore various data sources to annotate system components such as past execution history, alerts generated in the past, system dependencies, outages, Service Level Agreements (SLAs), among others.
- *Attributes derived from analytics:* We enrich the model with various attributes derived by graph analysis of dependency data, time-series analysis of execution history of jobs, and text and statistical analysis of alerts data.

We next propose an analyzer that is built over the model to extract interesting observations that provide insights into the areas of improvement. In this paper, we focus on improving stability of batch operations. To ensure stable operations, we first present the reactive solutions to detect the unstable areas and narrow down the root-cause of the operations. We next present preventive solutions to proactively monitor the operations to timely avoid any potential outages or performance problems.

II. Real world data set

We have used following data sets for the case-study while developing the ideas presented in this paper. We have masked the customer sensitive information.

Inter-job dependencies: We have extracted information of inter-job dependencies from Autosys scheduler. We analyzed data of 200,000 jobs from 50 business streams. We have done cross-stream analysis and then done deep-dive into one specific business stream (stream ABC). This stream consists of 4,450 jobs. The below data sets were collected for a duration of 6 months for stream ABC.

Run-history of batch jobs: This data contains the details of job execution such as start time, end time, cpu time, wait time, run time, and machine name.

Alerts: The batch schedulers can be configured to generate various types of alerts to monitor the batch jobs. Some typical alerts that are generated while monitoring batch jobs are: *job failed*, *job taking too long to execute*, and *job not started*. An alert is often associated with different severity levels such as high, medium, and low. Alerts are observed and acted upon by a support team. Many alerts are false or only informational, and hence are not acted upon. An alert, when acted upon, is converted into ticket. We analyzed the data consisting of 30,000 alerts and 10,000 tickets produced in a duration of 6 months. We exported this data from the alert logging tool, Sockeye.

III. Constructing the model of batch operations

The data exported from any standard batch scheduler contains rich job dependency information of the batch operations. We first use this information to construct a multi-level graph of business functions, jobs, and infrastructure.

A. Modeling the entities

We construct three main entities in the system model:
Stream: A stream refers to a specific business function and is often a collection of jobs. The batch operations are most often managed at stream level. The job schedules, hosts, and service level agreements are defined at the stream level. Furthermore, the support teams are also defined at the stream level.
Job: A job refers to a program that performs a specific task for the stream. Jobs are typically of three types: (a) command jobs that run a specific system command, (b) file-watcher jobs that wait for availability of a vendor feed or data feed and process the feed, (c) box jobs that refers to a logical job that is a grouping of tightly coupled jobs.
Host: A host refers to the virtual and physical machine on which the job executes.
Data feed: The file-watcher jobs depend on the availability of data feeds which often refer to location of specific filenames.

B. Modeling the relationships

The entities in batch systems are inter-dependent in several ways.
Inter-job dependency: The batch operations are defined as a sequence of batch jobs. Inter-job relationships are defined in the form of a precedence relationship. A relationship $A- > B$, indicates that job B can start only after job A finishes. Often a job depends on more than one jobs. For instance, $A- > B$, $C- > B$, indicates that job B can finish only after job A and job C finish. This represents an AND dependency with the parent jobs. Similarly, other relationships such as OR, NOT, and XOR relationships are also present between parent jobs and the child job.
Stream-job grouping: Each job is assigned to one specific stream. The stream-job mapping captures this relationship.
Inter-stream dependency: Jobs of one stream are often dependent on the jobs of other streams. Such dependencies can be used to derive inter-stream dependencies. A relationship can be defined between stream X and stream Y indicating that m jobs of stream X depend on n jobs of stream Y.
Job-host dependency: This dependency captures the host on which the job is executed. The host entity captures the mainframe as well as distributed environment. In case of virtualization, a host further captures the mapping of the logical instance to the physical host. Note that this relationship can change dynamically from day to day and needs to be updated in case of changes.
Job-data feed dependency: This dependency captures the availability dependency between the file-watcher jobs and the data files.

C. Modeling the attributes

The above entities are then annotated with attributes derived from various data sources such as dependencies, past execution history, business SLAs, past alerts history, and past outage history.

- Dependency attributes: We mine the dependency data to capture various dependency attributes such as fan-in, fan-out, parent nodes, child nodes, upstream tree, and downstream tree. In addition, we also compute interesting subgraphs such as connected components and cliques.
- Run-history attributes: We use the run-history information of the jobs to populate attributes such as the start time, end time, and run time of the job. We capture the time-series of these attributes as well as the statistically aggregated values such a the minimum, maximum, median, average, and standard deviation. In addition, we also capture the history of execution instances that violated SLAs. The job-level run-history metrics are also aggregated to connected components, and streams.
- Alerts attributes: We next annotate the jobs with information about the alerts generated in the past. Each job is associated with the time-series of alerts generated in the past. In addition, we also enrich a job with statistically aggregated values such as average number of alerts generated by a job in a month, temporal pattern of days of high alerts, etc. An alert that is acted upon by a resolver is converted into a ticket. Further, each ticket is associated with a severity of low, medium, and high, based on its business impact. We capture tickets information for each job with attributes such as average number tickets generated per month, the severity distribution of tickets, the average number of business outages or SLA violations per month, etc. The job-level alert attributes are also aggregated to connected components, and streams.
- System metrics: The hosts are enriched with time-series and statistically aggregated values of system metrics such

as CPU utilization, memory utilization, paging, IO, etc.

- Data-feed metrics: Each file is enriched with time-series and statistically aggregated values of the size, read time, and write time.

IV. Reactive analysis to improve stability

We next use the model explained in previous section, to develop solutions to improve stability of the batch operations. In this section, we present solutions for reactive analysis. We identify the areas that frequently observe stability issues and then perform a root-cause analysis using attributes derived from dependencies, alerts, and run-history.

A. *Detect areas of instability*

We first localize problems to connected components. We first identify connected components that generate high severity tickets for problem management.

Within the identified connected component, we prioritize jobs for problem management. To prioritize jobs, for each job, we compute a ratio of number of high severity tickets to number of alerts generated. We select jobs for problem management that have a very large ratio of high severity tickets to alerts. These jobs are further subjected to root cause analysis.

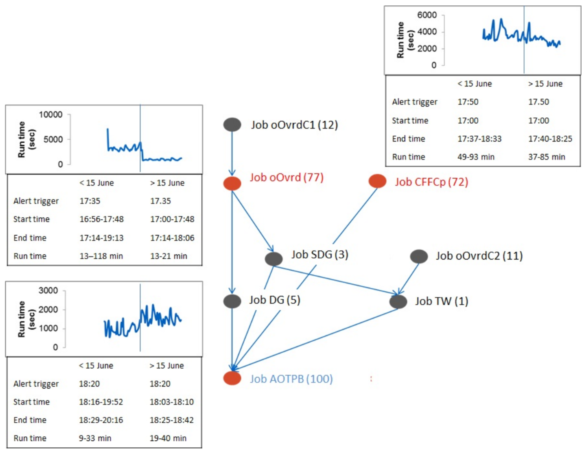
Case-study: In the case-study, for the stream ABC, one connected component with 1,690 jobs is identified for problem management.

We identified top 13 jobs on the basis of ratio of number of high severity tickets and number of alerts.These 13 jobs covered 32% of high severity tickets. The top job AOTPB ran 64 times in three months, generated 105 alerts out of which 100 were converted into high severity tickets in three months. It has ticket count to alert count ratio of 0.95.

B. *Perform root-cause analysis*

We next perform root-cause analysis of the identified problematic jobs. To deal with the large number of jobs and large volume of metrics data, we propose a multi-resolution root-cause analysis.

- *Dependency based localization:* A blind analysis across all jobs is highly compute-intensive and carries the risk of accidental correlations. Hence, we exploit upstream tree of the target job.
- *Coarse grained analysis:* In the identified upstream jobs, we perform a coarse-grained analysis using alerts data to localize the potential causes to the jobs that generate temporally correlated alerts with the target job.
- *Fine grained analysis:* The coarse grained analysis identifies the cause only to the job level. However, there are large number of parameters related to a job such as the start time of the job, runtime of the job, type of alert it produces, the machine on which it runs etc. Correspondingly, the cause of delay in the target job can be due to delayed start of the job, larger runtime of the job etc. It is important to identify the potential causal parameter within an identified job. Hence, we perform time-series correlation, and change-point correlation between the performance metric of the target job and the causal metrics of potential causes.



	< 15 June	> 15 June
Alert trigger	17:35	17.35
Start time	16:56-17:48	17:00-17:48
End time	17:14-19:13	17:14-18:06
Run time	13–118 min	13-21 min

	< 15 June	> 15 June
Alert trigger	18:20	18:20
Start time	18:16-19:52	18:03-18:10
End time	18:29-20:16	18:25-18:42
Run time	9-33 min	19-40 min

	< 15 June	> 15 June
Alert trigger	17:50	17.50
Start time	17:00	17:00
End time	17:37-18:33	17:40-18:25
Run time	49-93 min	37-85 min

Fig. 1. Upstream jobs of target job AOTPB and their run time.

Case-study: From the case-study, we present root cause analysis of a target job AOTPB.

The target job AOTPB has only 48 upstream jobs. By considering only upstream jobs we narrow down the scope from 4,450 jobs to 48 jobs. Further, we select only those upstream jobs that produce alerts that are temporally correlated to the alerts produced by target job AOTPB. In the case-study, out of 48 jobs, only 7 jobs produce alerts. This brings down the number of potential causal jobs from 48 jobs to only 7 jobs. Figure 1 shows 7 upstream jobs that generate alerts. Only 2 jobs out of 7 jobs have temporally correlated alerts with target job AOTPB. This brings down the scale of the jobs from 4,450 to only 2 for which we perform fine grained analysis.

We next performed fine-grained analysis on the identified two potential causes. The runtime of target job AOTPB, and upstream causal jobs oOvrd and CFFCp are shown in Figure 1. We observed that all the three jobs showed a change on 15th June. We identified following change points.

1) Target job AOTPB: (i) The target job AOTPB was generating alert *'delayed start'* before 15th June. It started generating alert *'delayed finish'* after 15th June. (ii) Also, the the job started taking longer time after 15th June.
2) Upstream job oOvrd: (i) Before 15th June, the job oOvrd was generating alert 'delayed finish'. After 15th June, it stopped generating this alert. (ii) Also, before 15th June the job was running on machine xxx1. The machine changed to xxx2 (a faster machine) after 15th June. (iii) The job started taking smaller time to run after 15th June. Before 15th June, the job was taking around 4000 sec to run. After 15th June, it was taking around 50 sec to run.
3) Upstream job CFFCp: (i) Before 15th June, the job was taking 49 to 93 sec. After 15th June it showed a decreasing trend in runtime.

After correlating changes in all the jobs, we conclude following:

- Before 15th June, the cause of delayed start of target job AOTPB was (1) *delayed finish* of job oOvrd, and (2) *delayed finish* of job CFFCp.
- After 15th June, the runtime of job oOvrd and job CFFCp reduced. And the cause of delayed finish of target job AOTPB was increase in its own runtime.

V. Proactive analysis to improve stability

In the previous section, we presented reactive solutions to detect stability issues and perform root-cause analysis. However, proactive solutions can help prevent the problem in the first place. It is important to monitor the batch operations to get timely warnings and take corrective actions in order to avoid business outages. The operations are monitored using two ways - (a) by real-time monitoring of the job execution status, (b) by configuring the job scheduler to produce alerts on observing anomalies. However, various challenges are faced in these two approaches. Furthermore, if incorrectly designed, these operations fail to be effective. In this section, we present analytics based approaches to improve monitoring of batch operations.

A. Real-time monitoring of job execution

The large scale of batch jobs presents a challenge in real-time monitoring of job execution. Monitoring thousands of jobs becomes a daunting task. However, not all jobs are equally critical. In this section, we first present different heuristics to identify subset of jobs that should be proactively monitored.

1) Identification of critical jobs for online monitoring: We present 4 heuristics to identify jobs that can have a significant impact on the overall operations of the stream:

1) Business-critical jobs : The obvious choice for the jobs to be proactively monitored is the jobs that perform a business-critical task and have a fixed Service Level Agreement (SLA) for timely completion.
2) Upstream to business-critical jobs: Only monitoring a business-critical job in not enough. It is important to monitor all the jobs on which a business-critical jobs depends. Monitoring of these upstream jobs can thus give a proactive indication of potential delay or failure of a business-critical job.
3) Jobs with high fan-in and fan-out: By analyzing the properties of the inter-job dependencies, we can identify jobs that can become potential bottlenecks due to high incoming dependencies and outgoing dependencies.
4) Jobs connecting multiple streams: The dependencies of jobs often span across business streams. Identification of jobs that have incoming or outgoing dependencies on multiple streams can help get proactive warnings across streams.

2) Construction of a subgraph of critical jobs: Once the critical jobs are identified, the next challenge is to derive a subgraph of inter-dependencies of critical jobs. Many critical jobs are connected to each other via paths that often contain many non-critical jobs. To reduce the scale, it is essential to compute a subgraph consisting of only critical jobs that show derived inter-dependencies between the critical jobs.

This can be done by creating a folded graph such that only critical jobs are retained along with their upstream and downstream dependencies. A folded graph is generated such that an edge is present between two critical jobs if their exists a path between them in the original graph.

VI. Related work

Batch systems have primarily been addressed in the past literature for scheduling of batch jobs. Authors in [1] formulate the batch-scheduling problem and present several scheduling heuristics for multi-objective optimization. Work has also been done to derive critical nodes and critical paths by analyzing inter-job dependencies. Authors in [4] have presented graph-mining algorithms to derive the frequent subgraphs of jobs and inter-job dependencies.

The problem of root-cause analysis has been addressed by various researchers for a variety of domains [3], [2]. In this paper, we customize the root-cause analysis solutions by considering the properties and most commonly available datasets in batch systems.

VII. Conclusion

This paper presents our experience in using analytics to understand, control, and optimize batch operations. We presented a real-world case-study and used various data sources for analysis such as dependency data, alert and ticket data, and run history data of the jobs in batch systems. We presented reactive and proactive approaches for improving stability of batch operations and demonstrated their effectiveness through a real-world case-study.

As part of our ongoing research, we are working on various open-problems in the analysis of batch systems such as (a) defining optimal scheduling recommendations such that all SLAs are met, (b) analysis of system capacity to compute available headroom, and time to saturation, (c) what-if analysis to predict impact of system changes such as increase in workload or change in infrastructure.

References

[1] Rushi Agrawal and Vaishali Sadaphal. Batch systems: Optimal scheduling and processor optimization. In *HiPC*, 2011.

[2] K. Appleby, G. Goldszmidt, and M. Steinder. Yemanja-a layered event correlation system for multi-domain computing utilities. *Journal on Network and Systems Management*, 10(2):171–194, Jun., 2002.

[3] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, 2004.

[4] M. Natu, V. Sadaphal, S. Patil, and A. Mehrotra. Mining frequent subgraphs to extract communication patterns in data-centres. In *International Conference on Distributed Compoting and Networks (ICDCN), Bangalore*, 2011.