Digital Fleet Management: A Scalable Cloud Framework Based on Data-Driven Prediction Models

Sherin Thomas[1], Abhishek Dubey[2], Daniel Viassolo[3], Magson Zanette[4]

[1,2]*Schlumberger, Pune India Technology Center, Maharashtra, 411006, India*
SThomas35@slb.com
ADubey4@slb.com

[3]*Schlumberger, 1430 Enclave Parkway, Houston, TX 77077, USA*
DViassolo@slb.com

[4]*Schlumberger, 555 Industrial Blvd, Maildrop 4, Sugar Land, TX 77478, USA*
MZanette@slb.com

ABSTRACT

A scalable Digital Fleet Management System can be leveraged by organizations with high-volume high-value assets. In such scenarios, predictive analytics for tool health becomes central, as it enables decision-making in terms of planning, maintenance, end-of-life replacement, tool selection, etc. An end-to-end solution spans all the way from gathering live tool data to visual representations of tool health.

Long-term fleet management can be accomplished through a consistent evaluation of the fleet performance profile. Predictive analysis can anticipate maintenance needs and resultant downtimes, and in turn it helps improve scheduling of procurement and distribution of the fleet.

Overall, such a framework can be divided into two focus areas: framework deployment; sustenance and algorithm development. The former area focuses on all the topics related to developing an end-to-end solution architecture, scaling and deploying it on demand, and maintaining it going forward. The later area focuses on defining risk index, developing Machine Learning (ML) algorithms for different tools, defining a single comparison metric and deciding on when to trigger the automated re-training for each tool. This paper focuses on raising and solving all key questions for building such a framework.

1. INTRODUCTION

Fleet management systems are about managing a fleet of thousands of downhole tools based on tool health condition and other variables – a common use case in Oil & Gas Services. Fleet Management continues to have its relevance in an organization's competitive business growth as it facilitates cost and time savings (Alsyouf, 2012). The impact of unplanned downtime in case of industries like Oil & Gas (Moir, Niculita & Milligan, 2018) is significant, as it leads to production outage and in turn revenue loss. The increased availability of monitoring data can be exploited to build data-driven applications that can increase the productivity by efficient fleet maintenance. The maintenance and failure histories of the tools combine to form a weighty source for predicting the failure probabilities. Facilitating the flow of large data through a structured and automated channel enables fast and reliable identification of impending failures, eliminating the human-errors, bias and assumptions. The state-of-the-art technologies of Machine Learning, Cloud Computing and Big Data are ramping up the possibilities of efficient fleet maintenance.

In this paper, an end-to-end automated scalable cloud framework is described in detail, which integrates failure prediction models for each asset in the large fleet of tools. Based on historical tool data, the models generate tool risk indices (one index per asset) which correlate to the probability of tool failure during near-future field jobs. These risk indices can be used for optimal asset-to-job mapping. They also help in de-risking field operations by identifying tools for overhaul or retirement. The proposed method integrates the tasks of fetching data from 200,000+ tools, performing feature engineering, modeling via ML, and visualizing into a cloud pipeline. Framework scalability becomes a key requirement as fleet size increases or decreases over time to match market demands. The framework also allows for the easy addition of new ML models to the platform by citizen data scientists, who are not cloud experts. Finally, it is shown how this framework provides systematic steps for sustenance of such large cloud platform.

Before we start getting into details of the framework, let us try to raise all the questions which we must solve through this framework.

1. What should be a generic architecture of such a deployment and sustenance framework, that can
   - Automatically scale up and down on demand
   - Show all the activities and issues in real time
   - Be configurable without code
2. How to enable easy and stable onboarding of any new tool model by data scientists to the framework
3. How to store final output keeping in mind that
   - The final output data can be huge
   - Easy integration with visualization tool is a key factor
   - Requirement for inbuilt role level access management

Questions related to the algorithm development side of the framework are:

1. How to define and interpret the Tool Risk Index
2. For a given tool, how to compare various algorithm's prediction accuracies (with different output scale) to select one
3. How to compare predictions and model training across different tool families
4. How and when to trigger the auto model re-training for any tool

Any large-scale fleet management system should provide a clear answer to both the above group of questions at various stage of project life cycle from development, to production/ commercialization and finally to sustenance. The framework proposed in this paper tries to answer all these questions and covers all the critical components an organization would need to build such a large-scale framework.

2. RELATED WORK

Fleet Management has its past rich with several researches to develop dynamically scalable, distributed and fully automated systems that can enable timely information gathering, knowledge sharing and maintenance scheduling. Variants of maintenance strategies are subjected to different studies in the past, including corrective, time-based, condition based (Kothamasu, Huang & VerDuin, 2006) and predictive maintenance. The predictive process is defined to have steps including preprocessing, fault prognosis and post-action prognosis (Lee, Lapira., Bagheri, & Kao, 2013). To deliver these functionalities of predictive systems, cloud-based frameworks were developed for different industries, delivering PHM as a service (Lee, Yang, Lapira, Kao & Yen, 2013, Mounir, Guo, Panchal, Mohamed, AbouSayed & Abou-Sayed, 2018). Apiletti, Barberis, Cerquitelli, Macii, Macii, Poncino and Ventura (2018) proposed a similar architecture in Industry 4.0, utilizing the distributed cloud services. A framework based on Amazon Web Services (AWS) was developed by (Mahmud, Iqbal & Doctor, 2016) to facilitate big data analysis and data visualization in health-shock prediction.

In the approach proposed in this paper, the big data analytics capabilities (Mohammadpoor & Torabi, 2018) are leveraged by integrating them with Google cloud services, which provides a flexible, low-cost and secure framework for predictive analysis and visualization. Data-driven approaches employing pattern recognition and machine learning (Schwabacher & Goebel, 2007) are integrated to the cloud framework for predicting a risk factor for each of the tools. The complex computational tasks on big data are delivered using the cloud computing platform (Ji, Li, Qiu, Awada, & Li, 2012), by choosing the optimal infrastructure and services. This framework provides the advantage of spinning up and down resources dynamically on demand, depending on various factors like data volume, cost of operation, complexity, time, etc. It also ensures the reliability and security of the system, by using the minimal standard cloud services. The proposed pipeline enables the use of predefined variants of machine learning technologies in cloud (Pop, 2016) along with custom made models. Almost all the large scale PHM papers focus only on the cloud and distributed computing, while the problem specific PHM/ ML papers focuses on the problem formulation and solution. This paper provides a novel framework which covers both the aspects of building such a large complex system, irrespective of the underlining technology, cloud/ computing vendor, Machine Leaning algorithms. We also provide an equivalent solution with respect to the suggested framework using Google Cloud Platform, but a similar solution can be developed with any of the major cloud providers today. We not only provide a framework, but also define the individuals and components that are needed in the overall framework covering everyone from Subject Matter Expert (SME), Data Scientists, Cloud Engineers to the IT support.

3. PROPOSED METHOD

To address and solve all the questions raised from framework deployment and sustenance side, we present a high-level architecture diagram for an end-to-end Digital Fleet Management System. Let us first analyze how the platform interacts with each of the main outside entities.

The **Project Data Pond** gets continuous data from the organization's Data Lake through a standard ingestion/ subscription method. The data ingestion/ subscription is usually handled at the organization level and not at the project/team level through a separate dedicated Data Platform team.

The **Data Scientist** having read-only access to the project data pond pulls the data for the tool which he/she needs to develop the machine learning model for. Once they have explored and developed the Machine Learning model for one tool, they package their code as separate training, validation and testing module. This code is then pushed to the project code repository.

The **Platform Sustenance Engineer**, schedule and monitors the platform jobs, using the framework orchestration/ monitoring platform. Unless there are major component or flow changes required, platform should be configurable just by simple parameter updates.

The **End User/ Product**, depending on the permission level should be able to consume the data through RESTful API. This ensures the smooth integration across all time of data consumers, like any visualization tools, web application, mobile application, etc.

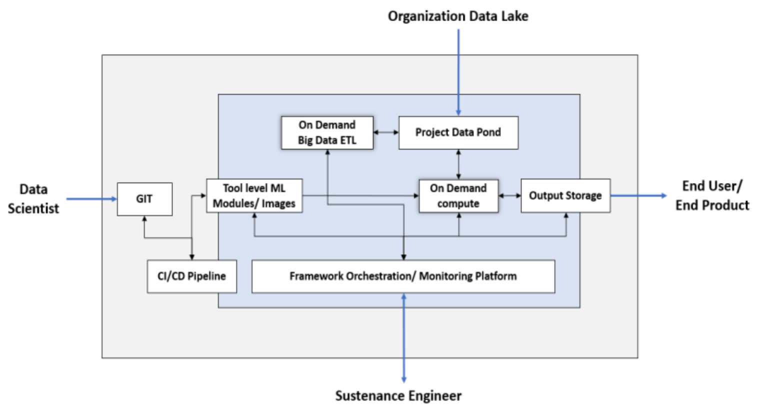

Figure 1. High Level Architecture

3.1. Platform Architecture

The proposed architecture in Figure 1 shows the components required to build the platform. The organization can choose to deploy it on their on-premises infrastructure, on cloud platform, or can choose to go with a hybrid setting. For our fleet management platform, we decided to deploy the above architecture on Google Cloud Platform, one of the leading cloud platform vendors. Figure 2 shows the equivalent architecture diagram for our platform.

3.1.1. Tool Family Level Model Images with CI/ CD

This component is required for two main reasons. First, it lets the Data Scientists to add their ML models to the framework without much knowledge of cloud and actual architecture. Second, this is required for submitting the individual model training, validation and testing for all tools to run on demand in parallel. Once the data scientists have developed the

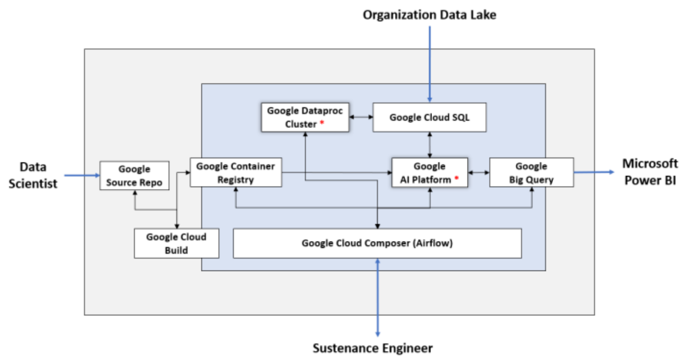

Figure 2. Platform Architecture on GCP

model, all they need to do is package the different code modules for training, testing and validation into a predefined structure and commit to the central project code repository. After the code is pushed and the approver approves the merge to the main branch of the project Repo, the Cloud Build runs the standard basic test cases. Once the test cases are passed the Cloud Build builds the model Docker container image and saves it to the Google Container Registry. The advantage of running a model in a container is increased portability of the model, instead of restricting our Data Science team from selecting specific tools or technology limited by our platform. The models in containers are easily deployable to multiple different platforms including different hardware, operating systems, with flexible sets of software packages defined in the containers.

3.1.2. On Demand Big Data ETL

Once the data is ingested into the Project Data Pond which can have global schema at the organization level, what is required is to separate the data for each tool for the Machine Learning tasks. This is the place where most of the common big data extract, transform and load (ETL) operations takes place. This component is brought up on daily scheduled hour for big data segregation and preprocessing and is brought down after that. This can be Google DataFlow in case of almost real time data ETL or can be Google Dataproc for more Batch ETL. As we are running our tool risk index prediction pipeline only once a day, we choose to go with Dataproc here.

3.1.3. On Demand ML Training

Depending on the orchestration platform setting if required the training and validation model images for different tools are pulled out and scheduled in parallel for all these tools on AI platform. At the same time for all the remaining tools, the prediction images are also scheduled in parallel on cloud AI platform. All of these training, validation and prediction jobs are run as a separate job on Cloud AI Platform and we are just charged for the amount of hours we run these jobs. So we do not own or manage the compute machine, we just submit the jobs, gets the results and pay only for the compute duration. Apart from on demand pay as we use jobs, these AI platforms are highly scalable to run many parallel jobs at once.

3.1.4. Output Storage and Visualization

Finally, the output of the validation and prediction jobs on AI Platform are pushed onto the Google Big Query. We store the tool level risk index, features used for modelling, other important tools level details and the validation score of each training job in Big Query. Whereas all this information can be used by any end product or user by simple REST call, provided they are authenticated for the data, the validation job score is further used by orchestration platform to decide on whether any tool family needs re-training on not.

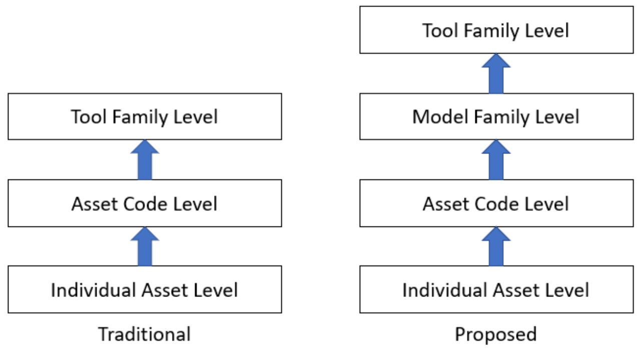

Figure 3. Tool hierarchy

3.1.5. Workflow Orchestration

In order to manage the entire workflow, we use an orchestration tool that can, with minimal code or configuration, schedule the tasks and provides a rich interface, allowing the administrator to visualize, monitor and troubleshoot tasks. We use Google Cloud Composer (build on top of Apache Airflow) for this purpose. With its cross-platform support, it manages the entire process orchestration, coordination and provide an inbuilt monitoring web application, which not only provides the real time job/ platform status, but also keeps history of all past jobs.

3.2. Fleet Modeling

Once we have built an end-to-end scalable framework like described above with the help of cloud or infrastructure team, we are just halfway done for the job. The most important task now is to systematically on board the massive asset fleet of the organization on to the platform. And if we provided one tool family per data scientist, given the size of our fleet, it would take us years to onboard all the tools. This is where it becomes important for any organization to break down and define the tool modelling steps as given next.

3.2.1. Tool Hierarchy

As shown in left side of Figure 3, usually individual assets belong to an *asset code* family. Based on type of job/ measurement (nuclear, resistivity, induction, etc.) performed by each asset family, they are combined into different *Tool families*.

A lot of times there is only a minor difference between various asset code families, like a tool revision. Hence, we can combine various asset code families into a broader group called *Model families*, so that we can model them together using Machine Learning, as shown in right side of Figure 3. Though most of the times it is at model family level where we would deploy each of the Machine Learning model, but based on the fleet properties, it can differ and can be at other levels as well.

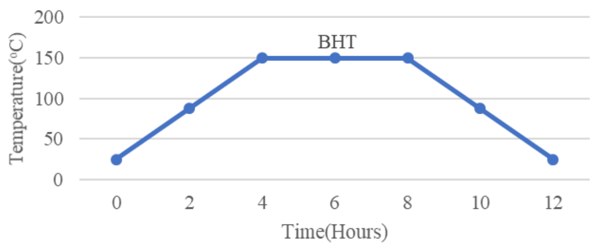

Figure 4. Arrhenius hours calculation

For our fleet of 2,000,000 individual tools, on an average 400-500 tool were present at an asset code level and 5-10 asset codes were combined at a model family level.

3.2.2. Job Aggregation vs No Aggregation

Now at a model family level we must again decide whether the tools show the failure trends on an individual tool level or not. For all the model families which do not show the trends at individual tool level, we should go back to our SME and understand how we can aggregate the individual tools data for failure trends.

For modelling the former type model families, each tool job acts as one data point for the training algorithm, whereas for modelling the later type of model families, each bucket of the aggregator index act as one data point. For example, Arrhenius equivalent hours (Laidler, 1984) is one of the most common aggregator indices. So, in this case, first we compute the Arrhenius equivalent hours spent by a tool in each job. Then we must fix the bucket bin size, as a hyper parameter, to aggregate all the tool jobs in the bin as one data point for the modelling algorithm. This kind of model family level aggregation becomes even more important for the tools which have a smaller number of failures, and these failures are mostly related to long term aggregated tool history.

We calculate the Arrhenius equivalent hours by splitting tool's operating time into 3 sections, as shown in Figure 4. The first section shows a rise in temperature, the second section has a constant temperature equal to the Bore Hole Temperature (BHT) and the final section shows a drop in the temperature back to the initial one.

$$t_{spent} = t_{operation}.AF \qquad (1)$$

where AF is the Acceleration Factor determined as,

$$AF = e^{\frac{Ea}{h}\left(\frac{1}{T_{ref}}-\frac{1}{T_o}\right)} \qquad (2)$$

where Ea is the activation energy (eV), h is the Boltzmann's constant (8.617343×10-5 $eV/°K$), T_{ref} is the reference temperature (e.g., 150℃ for some tool families) in Kelvin and $T_o$is the operating temperature in Kelvin.

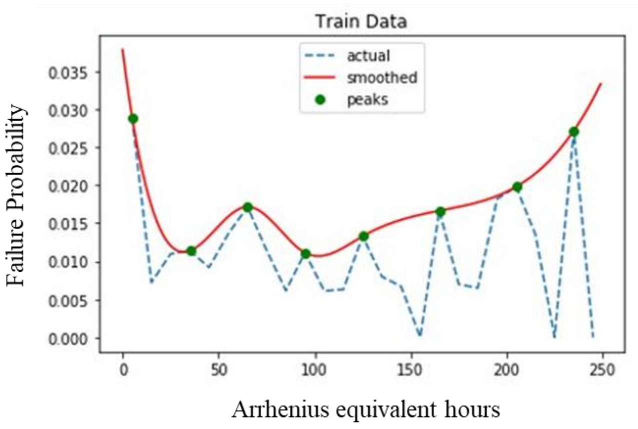

Figure 5. Smoothing failure probabilities using cubic spline interpolation to eliminate outliers

We sample the temperature at equal intervals, with the distribution as in Figure 4, and calculate the total Arrhenius equivalent hours spent by the tool, in each of the three sections.

The overall idea is to model the trend of the tool failures. Though every sub sample of tools might give the same failure probability trend, but the failure actual values might differ at the subset level. For this kind of issues, we restore to learning the average (or peak) failure trend, rather than the actual failure trend. The values are interpolated using Linear/Cubic Spline interpolation methods as shown in Figure 5.

3.2.3. Tool Risk Index

One of the most important decisions is the definition of Risk Index based on how we want to use/interpret this risk index. While exactly interpreting the risk index as the probability of the tool failure would greatly reduce the tool modelling approaches, a more general approach is to use the risk index to rank the tools in the order of its failure probability. The former one should be called the Tool Risk Probability, while the later one must be called the Tool Risk Index.

We decided to go with Tool Risk Index, as almost every time modelling the exact tool risk probability was not possible with the data in hand, but in general tool ranking was still possible to good levels. Eventually we all must move from Tool Risk Index to Tool Risk Probability, but this would mostly be driven by how well we capture the data for every tool in the fleet.

3.2.4. Model Validation Metric

The actual data with such a system log is tool failure information and by default each job can either be in healthy or failed state, which makes this a binary class data. We can choose to aggregate each job, say at weekly level, and find the ratio of number of failure jobs to total number of jobs, making it more like a regression data problem. Also, based on the problem formulation, different error metrics would be more appropriate. For such a wide and complex range of tool families the Data Scientist should be free to formulate this problem anyway possible and we must select a metric which not just gives us a way to compare different models for same tool family but also a way to compare models for different tool families. We must look for a metric which overall gives a higher risk index to risky tools and lower risk index to the healthy tools. This metric should be more focused on the mistake we do by giving higher risk index to a healthy tool or other way around; i.e., absolute values are not as important as their relative values. This is where the idea of using the Swap Ratio as a model quality metric comes into consideration.

For testing of models, we define a *testing window*, and take all the data before this window as the training data. The data scientists are free to choose any modelling approach and the models for the training data, and then we compare the models using the status of the first job in the testing window. A higher risk index should imply a failure job and a lower risk index imply the normal healthy job.

In Figure 6, let's say we have N tools, for each tool gray dot is healthy job, colored (blue, read & green) dot is a failed job. Blue is failed job in training window, red is first failed job in testing window and green is any failed job afterwards in or outside the testing window. For validation or testing, the job of interest is first job in testing window, if it is red, the tool failed in the first testing job and if it is gray, the tool was healthy in first testing job. In Figure 6, T3, T4, T6 are the failure tools and T1, T2, T5 are the healthy tools. A good model would provide higher risk index to T3, T4, T6 and lower risk index to T1, T2, T5. So, for 3 failed and 3 healthy

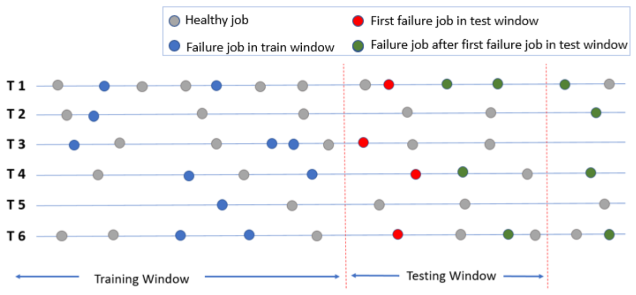
Healthy job
Failure job in train window
First failure job in test window
Failure job after first failure job in test window
T1
T2
T3
T4
T5
T6
Training Window
Testing Window

Figure 6. Train and test data split

tool jobs above we have 9 pairs, {(T3, T1), (T3, T2), (T3, T5), (T2, T1) … (T6, T5)}. We count the pairs which have higher risk index for the healthy jobs as we want to have least of these pairs. The ratio of these pairs to total pairs is called the Swap ratio, as these are the number of swaps the Tool Risk index must do to achieve the perfect model. We can use this as the testing and validation metric.

We use an advanced version of Swap Ratio defined above, called C-Statistics (Austin et al., 2012). With the expectation that the probability prediction for failed jobs should be greater than normal jobs, we get the count of jobs which are concordant, discordant or tie, with this assumption. We calculate an index value known as the c-index, which can evaluate the discriminatory performance of the model as,

$$c - index = \frac{number\ of\ cordant\ pairs + 0.5 \times tie}{All\ pairs} \tag{3}$$

This concordance static is equal to the area under the Receiver Operating Characteristic (ROC) curve, with the values ranging from 0.5 to 1, where 1 indicates perfect discrimination and 0.5 corresponds to a model with no discrimination ability. This approach saves the need to fix arbitrary threshold values for classifying a prediction as failure or not, as it varies with use cases. Also, the c-statistics method considers all thresholds, as it is same as the area under the ROC curve. This approach is useful when there is no trade-off between the false positives and false negatives.

3.2.5. Model Re-training

For each model deployed to the framework, we keep track of Swap Ratios and C-statistics for each month. When the Data Scientists deploy the model for any model family, they are also providing the lower bound on the mean and the upper bound on the variance of these metric which should trigger the model re-training job for this model family on the platform. Usually the Data Scientist comes up with these values based on cross-validation on training data and confirms them with the SME for this model family. These metrics are available to the Sustenance Engineer and the Subject Matter Experts to keep updating in a timely manner based on the model performance and feedback from the actual field users. Based on the user feedbacks they can decide to go for the parameter re-adjustment or for a complete re-modelling for any model family.

4. EXPERIMENTAL RESULTS

The proposed architecture was implemented and tested for various tool families including electric power cartridges, Gamma Ray Neutron Sondes, Telemetry cartridges, Pump out modules, etc. The data was segregated and preprocessed to obtain desirable features as input vectors for each model families. Features including job information like BHT per job, pressure, density, failure details and maintenance records were utilized for modeling based on the tool characteristics. Various feature engineering techniques were experimented for each family, including dimensionality reduction, sampling and other custom-made features. Machine Learning models like Random Forest, SVM, Gradient Boost, Linear Regression, etc., were deployed for different tool families.

Each of these models were validated with the data over a specified time period and resulted in C-index values in the range of 0.8-0.9 on an average. The framework keeps track of the validation results for each month and this can be evaluated to decide on the need for model re-training based on expert opinions.

5. FRAMEWORK ESTIMATES

The Project team can further optimize the above framework in terms of development time and running time/ cost. Based on cost/ time requirements:

1. The platform can be scheduled to run on daily to monthly basis. For most of our model families, we run predictions once per day.
2. The ETL Engine (DataProc in our case) can run multiple small instances or a single large instance.
3. The tool ML training jobs are all submitted in parallel or can be combined by the code requirement level (like Python, R, Octave, etc., at higher level to the individual package requirements at lower level).

The entire framework can be developed in three parallel phases, which all play equal part in the success of the platform

1. Software Engineering team working on developing and setting up the cloud/on premises platform.
2. Product Owners working with end users (reliability engineer, job planners, etc.) for usability features to be included in the visualization tool.
3. Asset Experts working with Data Scientist to classify the tools and develop the ML models for each one of them.

The next step for such a system would be to move from daily to near real time predictions, where along with tool daily job history, we also analyze the real time tool sensor data. Though the training architecture for such a framework would exactly be the same, there would be changes in the prediction architecture. Along with Big Data ETL Engine we would also need a Streaming Analytics Engine (Example for GCP would be Cloud IoT Core with Dataflow).

6. CONCLUSION

For organizations with a very large fleet of assets, building a Digital Fleet Management Platform is a first major step towards data driven PHM. This demands deep technical expertise in multiple areas, like ML, Asset Level Domain Knowledge, Digital Infrastructure and Cloud offerings.

This paper provides guidelines on how to build a cost-effective framework with fast response times, plus secure and reliable operations to avoid unscheduled downtime with systematic fleet maintenance policies. We developed a generic framework that scales automatically with demand, monitors the activities continuously with easy configurations for tool onboarding, output storage and visualization. We also defined a risk assessment factor for the tool families that can be used to compare different tools based on a common metric. These tool-to-tool comparisons are scheduled on regular intervals within this framework to ensure minimal tool failures during jobs.

NOMENCLATURE

ML	Machine Learning
AI	Artificial Intelligence
AWS	Amazon Web Services
REST	Representational State Transfer
API	Application Programming Interface
SQL	Structured Query Language
BHT	Bore Hole Temperature
eV	Electron-volt
ROC	Receiver Operating Characteristic
ETL	Extract, Transform, Load
SME	Subject Matter Expert
SVM	Support Vector Machine
IoT	Internet of Things
GCP	Google Cloud Platform
CI	Continuous Integration
CD	Continuous Deployment
PHM	Prognostics and Health Management
IoT	Internet of Things

REFERENCES

Alsyouf, I. (2007). The role of maintenance in improving companies' productivity and profitability. *International Journal of Production Economics*. 105. 70-78. 10.1016/j.ijpe.2004.06.057.

Apiletti, D., Barberis, C., Cerquitelli, T., Macii, A., Macii, E., Poncino, M., & Ventura, F. (2018). iSTEP, an Integrated Self-Tuning Engine for Predictive Maintenance in Industry 4.0. 2018 *IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications* (ISPA/IUCC/BDCloud/SocialCom/SustainCom), 924-931.

Austin, P. C., & Steyerberg, E. W. (2012). Interpreting the concordance statistic of a logistic regression model: relation to the variance and odds ratio of a continuous explanatory variable. *BMC medical research methodology*, 12, 82. doi:10.1186/1471-2288-12-82.

Ji, C., Li,Y., Qiu, W., Awada, U., & Li, K. (2012). Big Data Processing in Cloud Computing Environments. *Proceedings of the 2012 12th International Symposium on Pervasive Systems, Algorithms and Networks* (I-SPAN '12). IEEE Computer Society, USA, 17–23, doi:10.1109/I-SPAN.2012.9

Kothamasu, R., Huang, S. H., & VerDuin, W.H. (2006). System Health Monitoring and Prognostics – A Review of Current Paradigms and Practices. *International Journal of Advanced Manufacturing Technology* 28, 1012–1024). doi:10.1007/s00170-004-2131-6.

Laidler K. J. (1984). The development of the Arrhenius equation, *Journal of Chemical Education* 1984 61 (6), 494 doi:10.1021/ed061p494

Lee, J., Lapira E., Bagheri B., & Kao, H. (2013). Recent advances and trends in predictive manufacturing systems in big data environment. *Manufacturing Letters*, Volume 1, Issue 1, 2013, Pages 38-41, ISSN 2213-8463. doi:10.1016/j.mfglet.2013.09.005.

Lee, J., Yang S., Lapira E., Kao H., & Yen, N. (2013). Methodology and Framework of a Cloud-Based Prognostics and Health Management System for Manufacturing Industry. *Chemical Engineering Transactions* 33:205–210. doi:10.3303/CET1333035.

Mahmud S., Iqbal R., & Doctor F. (2016). Cloud enabled data analytics and visualization framework for health-shocks prediction. *Future Generation Computer Systems.* Volume 65, 69–181, ISSN 0167-739X, doi:10.1016/j.future.2015.10.014

Mohammadpoor, M., & Torabi, F. (2018). Big Data analytics in oil and gas industry: An emerging trend. *Petroleum*, ISSN 2405-6561, doi:10.1016/j.petlm.2018.11.001.

Moir, K., Niculita, O., & Milligan, W. (2018). Prognostics and health management in the oil & gas industry – a step change. In *Proceedings of the European Conference of the PHM Society* (Vol. 4 (1)).

Mounir, N., Guo, Y., Panchal, Y., Mohamed, I. M., Abou-Sayed, A., & Abou-Sayed, O. (2018). Integrating Big Data: Simulation, Predictive Analytics, Real Time Monitoring, and Data Warehousing in a Single Cloud

Application. *Offshore Technology Conference*. doi:10.4043/28910-MS.

Nguyen, H M., Cooper, E. W., & Kamei, K. (2011) Borderline over-sampling for imbalanced data classification. *International Journal of Knowledge Engineering and Soft Data Paradigms*. 3. 4-21. 10.1504/IJKESDP.2011.039875.

Pop, D. (2016). Machine Learning and Cloud Computing: Survey of Distributed and SaaS Solutions. *arXiv preprint arXiv:1603.08767 (2016).*

Schwabacher, M., & Goebel, K. (2007). A survey of artificial intelligence for prognostics. *AAAI Fall Symposium* - Technical Report.

BIOGRAPHIES

Sherin Thomas is a Data Scientist at Schlumberger Technology Center at Pune, India. She has her MTech. degree in Computer Science from Indian Institute of Technology, Hyderabad. Her main research interests are machine learning, Bayesian data analysis, information retrieval and prognostics & health management.

Abhishek Dubey is a Senior Data Scientist at Schlumberger Technology Center at Pune, India. He has a MS degree in Computer Science from Indian Institute of Science, Bangalore. His area of expertise is building large scale, end to end Machine Learning Products on cloud. His main area of research is Deep Learning for structured & unstructured data and prognostics & health management.

Daniel E. Viassolo is an Analytics Manager and Principal Data Scientist with IT Data & Analytics, Schlumberger. He is a technical expert in Machine Learning, Prognostics & Health Management, Optimization, and Control Systems. Daniel authored 27 US patents plus 35 papers while working on diverse applications across Oil & Gas and Power Generation domains. He has a PhD (Purdue University) and Business Leadership training (Texas A&M).

Magson Zanette is Analytics Manager working with Technology Lifecycle Management (TLM) division of Schlumberger. He is leading the PHM projects for TLM with the Product Centers at Schlumberger. He holds Electrical Engineering degree from Federal University of Santa Catarina (UFSC).