PREDICTIVE DATA ANALYTICS FOR COMPLEX SYSTEM-OF-SYSTEMS

STUDENTS: BENJAMIN AZEVEDO, CHENYANG BAO, CHANG LIU

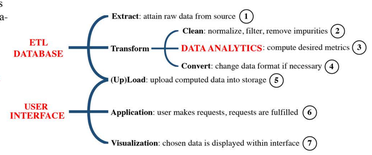
29

Mission Objective/Implementation

At Lockheed Martin some of the most advanced technologies in the world are developed. All that tech hinges around one commonality: data. Yet, what is vitally more important than data is the information extracted from it, hence data analytics. In this project we developed a seamless user interface for data manipulation, fault investigation, and predictive analysis of complex system-of-system performance trends. To accomplish that goal, our entire scope was encompassed by three main objectives:

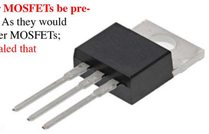
- Establish an ETL DATABASE to handle several terabytes of raw data
- Develop a DATA ANALYTICS capability to perform predictive health monitoring and assist in failure analysis
- Develop a streamlined USER INTERFACE to visualize and extract parsed data sets

Our three objectives were then parsed into seven generalized sub-tasks to better assess progress throughout our implementation process:



Background

In 2011 engineers at the NASA Ames Research Center had a question: **Can the remaining useful life (RUL) of power MOSFETs be predicted?** (The MOSFET used for experimentation was the International Rectifier, Thru-Hole, IRF520Npbf TO-220AB). As they would come to understand, it is indeed possible. Die-attach degadation was determined to be the primary failure mode of power MOSFETs; however, die-attach degradation is not a quantifiable metric. After further development of their experments, it was revealed that ON-State drain-to-source resistance (a quantifiable metric) increased as die-attach degraded under high thermal stress. Therefore, **ON-State drain-to-source resistance** was used as the primary precursor of device failure.



The data provided by NASA Ames was collected using accelerated aging, which was caused by thermal overstress and power cycling. Each MOSFET was run to failure. Health and failure analysis can be monitored through the following metrics: **incipient fault, standard fault, catastrophic failure, and RUL.**

Extract-Transform-Load Process

Figure 1

Figure 2

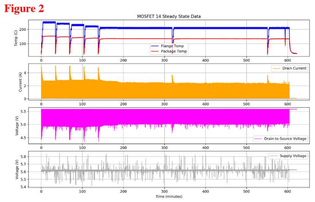


Figure 3

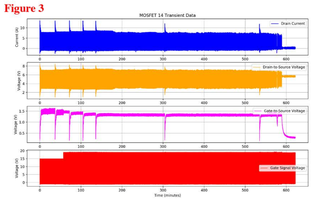
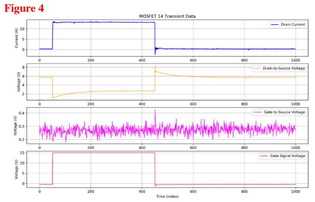


Figure 4



The dataset consisted of 104 separate files that accounted for 42 MOSFETs worth of data, which required 7.98 GB of storage. To better comprehend the content of data, we first built our ETL process to handle this small volume of data, then developed a scalable architecture to handle larger sets.

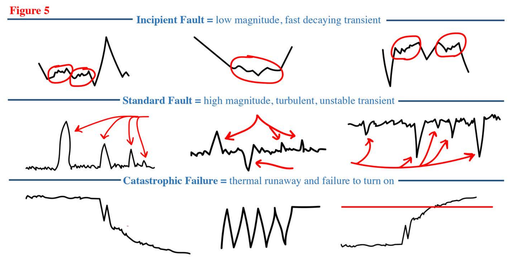
The transformation of our raw data into a cleaned form required six steps:

1. Plotting all collected parameters for a single MOSFET to determine any trends or relationships between parameters
   a. On initial glance we were able to make a number of assumptions:
      i. Pulse Wave Modulating data is comprised of "instructions" (fig 1)
      ii. Transient acts as high-speed 'bursts' of steady state data (fig 3)
      iii. Degradation is apparent (fig 1, 2, and 3)
      iv. 'Start-up' data is related between parameters (fig 1, 2, and 3)
      v. Drain Current and Drain-to-Source Voltage will require filtration (fig 2 and 3)
      vi. Gate-to-Source Voltage dives before failure, followed by Flange Temp, and Package Temp (fig 2 and 3)
      vii. Supply Voltage is noisy (fig 2)
      viii. Gate Signal Voltage seems to be the most dependable parameter for determining 'gate state' (ON-state or OFF-state) (fig 3)
   b. From the assumptions above we had the following takeaways and performed the following actions:
      i. PWM contains unimportant parameters for health diagnostics and failure analysis and can be ignored
      ii. Graph bursts and use only transient data for normalization
      iii. Normalize each run to Gate-to-Source Voltage
      iv. 'Start-up' data needs to be normalized (refer to iii.)
      v. Filter out all OFF-state data, and filter through sampling the averages of every minute worth of data
      vi. The diving parameters are important for RUL predictions
      vii. Supply Voltage is an unimportant parameter and can be ignored
      viii. Gate Signal Voltage will be used to determine ON-State
2. Removing OFF-state data using the Gate Signal Voltage
   a. Threshold value of 10V as determined by MOSFET datasheet
3. Normalizing ON-state data to related Gate-to-Source Voltage values
   a. $R_{ds} = R_{ds} * \frac{1}{\sqrt{V_{gs}}} * c$
      i. The constant c was determined through calibration
4. Applying additional damping to 'start-up' data
   a. $R_{ds(start-up)} = R_{ds} * \frac{1}{V_{gs}} * d$
      i. The constant d was determined through calibration
5. Sampling minute 'chunks' of ON-state data
   a. Sample Rate = samples / (testTime / 60)
6. Applying a shift to the entire data set, to normalize about the x-axis
   a. Subtract by the first value of every sampled dataset

Our cleaned data was then loaded to S3 cloud storage for future use in the front-end and data analytics portion.

Data Analytics Algorithms

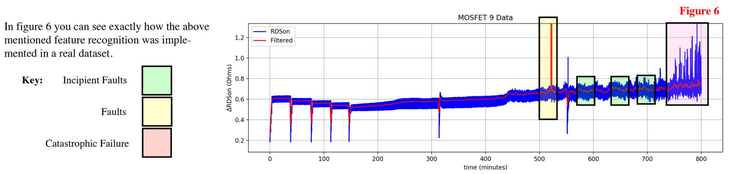
Figure 5



The prediction of remainig useful life (RUL) centers around three main health-related parameters: Incipient Faults as low magnitude, fast decaying transients; Standard Faults as high magnitude, turbulent, and instable transients; and most important catastrophic failure, which for our purposes (and those of the NASA Ames researchers) used a crisp failure rate of 0.05 for drain-to-source resistance (meaning that once the drain-to-source resistance (Rds) has increased by 0.05 ohms, the device is assumed to have failed). In figure 5 you can see examples of how those features appear in data (for clarity, the middle graphic under catastrophic failure represents the gate signal failing in the ON-state).

In figure 6 you can see exactly how the above mentioned feature recognition was implemented in a real dataset.

Figure 6



Four different algorithms were used to predict failure of the MOSFETs:

1. Basic Regression: computed using only the current Rds value; in the form y = ax
2. Linear Regression: computed using the average trend of all past Rds values; in the form y = ax + b
3. Exponential Regression: computed using the curvature of all past Rds values; in the form y = ax^b
4. Recurrent Neural Network: forecasting computed using all available parameters (drain current, drain-to-source voltage, gate-to-source voltage, gate signal voltage, flange temperature, and package temperature)

In all algorithms, the basic strategy was to use past and present information in order to extrapolate into the future to predict RUL (determined when drain-to-source resistance crosses the threshold of 0.05 ohms) as seen in figure 7. For reference, in figure 7, the red line represents basic regression, the blue line represents linear regression, the green line represents exponential regression, and the purple line represents the recurrent neural network. Since RUL predictions were measured in units of time, a perfect trend line was seen as a 45 degree angle, as displayed in figure 8. Figure 9 gives a more detailed explanation of the RUL values collected in figure 8. Given the nature of the various algorithms we expected their rate of error to increase in the following order (least to greatest): recurrent neural network, exponential regression, linear regression, and finally basic regression. However, this is not always the case as seen in MOSFET 8.

Figure 7

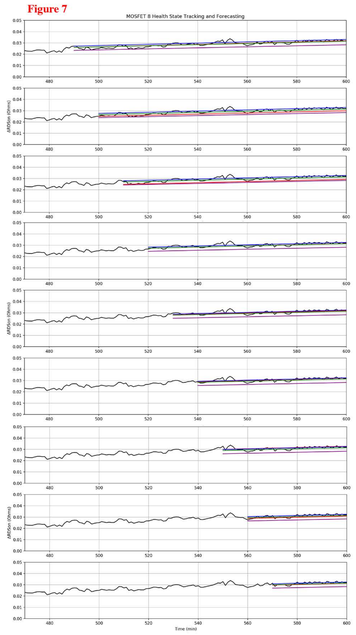


Figure 8

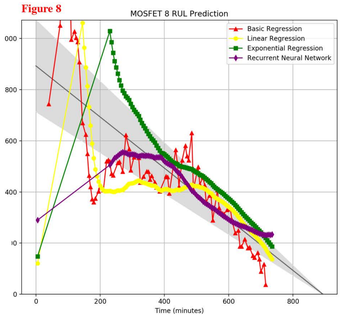


Figure 9

Time	*RUL*	*BasicR*	*LinearR*	*ExpR*	*RNN*
300	593	510.06	434.12	732.81	547.23
350	543	479.19	429.04	627.02	541.31
400	493	446.54	410.58	547.2	520.24
450	443	472.7	409.26	500.22	466.55
500	393	468.17	423.2	477.06	387.27
550	343	288.27	387.39	427.02	337.23
600	293	336.89	341.28	373.31	291.68
650	243	214.75	269.64	306.47	263.65
Average Error		2.74%	3.01%	20.30%	0.96%

Web Service and Storage

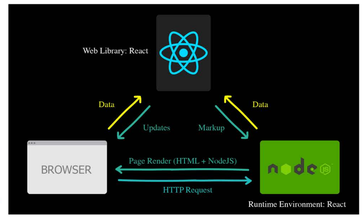
This system enables cloud (remote) interaction with data and visualization without the need to repeatedly deploy on local machines. To fulfill this, we use:
- AWS S3 for storage, which supports remote access (both read and write), security level enable, etc.
- Heroku for deployment, as it has flexibility to customize unique DevOps workflow needs.

A typical workflow will look like: User uploads their data file to S3 from a local machine, then they would then be able to conduct analysis and visualization of that same data in the Heroku application.

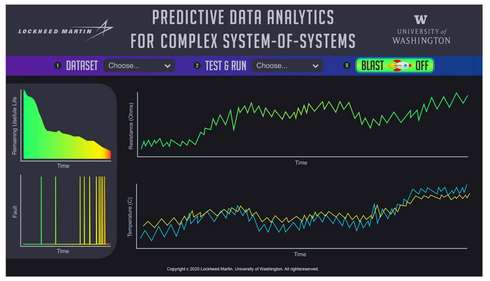
To implement our web service, we used a browser/server model. To be specific, this includes:
- React as the variable and reusable components and responsive layout improve the aesthetics and interactivity of the web page.
- NodeJS as development environment.
- Bootstrap for responsive grid system, extensive prebuilt components, and powerful JavaScript plugins.

For the server as backend, we use Python and Flask.
- Flask is a lightweight WSGI web application framework designed to make getting started quick and easy, with the ability to scale up to complex applications.

For interaction between frontend and backend, we chose Restful API.
- Restful API is based on representational state transfer (REST), an architectural style and approach to communications often used in web services development.



User Interface/Visualization



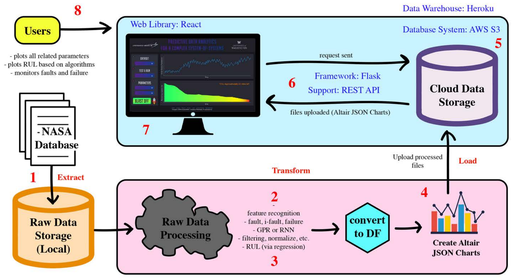
To provide data visualization, we used Altair, a wrapper for Vega-Lite, a JavaScript high-level visualization library. One of its most important features is its interactivity. For instance, you can add tooltips with one line of code and link the selection with another graph, which provides a straightforward interaction for users.

From this web application interface, a user is able to select their preferred dataset (currently there is only one, the MOSFET data), and which test they would like to run (the MOSFET dataset contains 42 different tests). After pressing the 'Blast Off' button, all parameters (drain current, drain-to-source voltage, gate-to-source voltage, gate signal voltage, flange temperature, and package temperature) are plotted against time. In addition, there are plots displaying RUL, incipient fault and standard fault indicator, as seen in the web display on the left.

System Overview

Following the full system from beginning to end:

1. Download files from NASA database to local machine
2. Process data on local machine by damping, filtering, normalizing, and shifting
3. Run drain-to-source resistance data through various feature recognition (incipient fault and standard fault) and prediction algorithms (basic regression, linear regression, exponential regression, and recurrent neural network)
4. Convert collected data into dataframe format and load to S3 cloud storage
5. Deploy application on Heroku
6. Python, Flask, and the REST API are used to communicate between the user interface and cloud storage
7. React, NodeJS, and Bootstrap together create our front-end web service
8. The user make requests in the user interface and the web service fulfills those requests



Future Implementations

There are always improvements to be made to engineering projets and our predictive data analytics web application is no exception. Moving forward we see any improvements to the project falling into one of three categories: scalability, functionality, and application to different data. We recommend the following changes:

1. Scalability
   a. Extract data using cloud storage like InfluxDB or AWS S3 that can handle terabytes of data
   b. Deploy application using AWS Elastic Cloud Computing (EC2)
   c. Improve scalability of data processing by implementing a service like Dask in conjuction with PANDAS API
2. Functionality
   a. Immediately transform data into a user-friendly format such as CSV or Parquet
   b. Refine data cleaning process. Contact professors and data analytics professionals for tips and tricks.
   c. Improved, novel approach to preditive algorithms
   d. Allow for serial input of data, with speed controls (10x, 100x, and even 1000x speed)
   e. Give user more freedom for plotting (allow any combination of parameters)
3. Application to Different Data
   a. Allow for processing of multiple different datasets

ELECTRICAL & COMPUTER ENGINEERING
UNIVERSITY of WASHINGTON

MENTORS: THOMAS MITTAN, PETER SULCS, MATTHEW NGUYEN, ELI SHLIZERMAN

SPONSOR: LOCKHEED MARTIN

LOCKHEED MARTIN