Filter Large-scale Engine Data using Apache Spark

Donato Pirozzi, Vittorio Scarano
Dipartimento di Informatica
Università degli Studi di Salerno, IT
dpirozzi@unisa.it, vitsca@dia.unisa.it

Steven Begg, Guillaume De Sercey, Andrew Fish, Andrew Harvey
School of Computing, Engineering and Mathematics
University of Brighton, UK
{S.M.Begg, G.DeSercey, Andrew.Fish, A.J.Harvey}@brighton.ac.uk

***Abstract*—This paper introduces a minimum viable software product to filter large datasets of engine data recorded during laboratory experiments of combustion engines. The aim is to support analysts in the identification and analysis of specific physical phenomenon within hours of recorded engine experimental data. Specifically, the tool has been designed considering the use case of identifying Low Speed Pre-Ignition events. This work describes the tool's graphical user interface and its scalable architecture based on mainstream web and big-data technologies as well as the practical application to pre-ignition events identification. The paper provides details on the architecture's performance, providing evidence of its scalability by increasing the number of available computing workers.**

***Index Terms*—Data Visualisation, Data Exploration, Combustion Engine Experimental Data, Big-data Technologies**

I. Introduction

In the engineering field data loggers collect huge amounts of data over time. In the automotive field, engineers develop and test engines by operating them on engine *test beds* [1], which allow engines to be operated in a safe and controlled manner for a variety of test regimes. The measurement of physical parameters are logged using data loggers. Engine test beds comprise of an array of sensors to continuously measure, and record, engine status values (e.g. engine speed, pressure, and temperature sensors) which can be used to analyse, in off-line mode, any phenomenon and performance comparisons. Due to the large number of sensors typically installed, the high logging frequency required and the extended period of test times, the data collected can be in the order of gigabytes per experiment. An industry or a research centre usually has many engine test beds running for 24 hours a day, hence, as in many engineering fields (e.g., simulation field [2]), data collected over time will generate big repositories of valuable assets which are strategically important for industries, and can be used to boost their competitiveness.

The analysis of this data is essential for engineers to fully study and understand a particular phenomenon. The analysis of the raw data itself is, currently, a difficult and time consuming process, due to the added difficulty of the separate data loggers splitting data over multiple directories and formats. This leads to a requirement for a flexible tool which intuitively, visually and interactively supports the researcher in getting insight into collected data. An engineer must be able to dynamically filter data according to specific formulae, to gain the correct insights into the results. Interactivity and dynamic filtering of large-scale datasets together require reasonable response times from the systems, thus the need to use big-data technologies to achieve high performances. The data collected also exhibits variety due to the differences in engines tested, the variation in test-bed conditions and their configurations.

This field poses different challenges, such as, data variety and heterogeneities, big data processing, human computer interaction (HCI) aspects, and involves multidisciplinary skills (i.e, Big-data expert, Engineers, Computer scientists, etc.). This paper focuses on the interactive filtering of large-scale engine experimental datasets using mainstream big-data technologies (i.e., Apache Spark). In particular, it introduces EnginXplora, a web-based front-end accessible through any web-browser, able to filter engine datasets and visualise retrieved results as well as analyse data through charts. The tool connects in the back-end to Apache Spark facilities to filter data. As a particular practical application area, the open engineering research question of the identification and understanding of LSPI events (Low Speed Pre-Ignition events) is discussed in Section III.

The paper contributions are: a detailed description of the scenario and its characterisation (Section III), a minimum viable prototype with a demonstration of the tool utility via the application of the scenario described earlier (Section IV), and an extensible and reusable software architecture to process large datasets (Section V) with a performance benchmark to evaluate its scalability. The paper concludes with a summary of the achieved results, ongoing and interesting future works.

II. Related Work

This Section describes scientific literature about recent tools to explore engine data. VISPLORE [3] is a system to visually explore multirun engineering data. It provides a wide range of well-known visualisations (e.g., histograms, scatter plots and parallel coordinates). It adopts an unguided approach, supporting many tasks in one system, where users are completely free to configure the visualisations as they desire. Recently VISPLORE authors described the challenges they encountered in introducing this system within the automotive industry [3]: (1) the lack of user guidance and (2) the lack of integration with other software systems in the industry. In order to address these issues, the authors switched to a guided approach, specifically designing a dashboard to perform one well-defined task, providing pre-configured visualisations, whilst permitting expert users to re-parametrise them. Based on this reported experience, along with our previous experience [4], in order

to design our EnginXplora, we chose the main user task within our scenario to be the identification of LSPI events.

ExploraTool [2] is a system for the visual exploration of engineering simulation data. In particular it uses a space filling visualisation Ellimap (a Treemap variant) which uses ellipses to represent groups of dataset items. ExploraTool provides an overview of the dataset, allowing its navigation and suggesting further facets to use in the exploration of the dataset. It is mainly a facet-based exploration system so it works by semi-automatically categorising simulation features. It does not support filtering based on the numerical data but only on the textual items' characteristics. In contrast, this paper provides a practical scenario where the exploration is primarily based on the filtering of numerical data, and the exploration through facets is initially inapplicable because the dataset is predominantly numerical. The tool EnginXplora reuses the idea of grouping items filtered from datasets, but instead of using the Ellimap visualisation, which was specifically designed to emphasize available facets and support data exploration, it uses Euler Diagrams to specifically display the relative sizes and relationships of (numerically) filtered datasets, showing their intersections to give a hint as to the items that are in common. In addition the groups of dataset items are dynamically generated based on filters defined by mathematical formula declared by the user.

III. Scenario: Pre-Ignition Events

This section introduces a live scenario of EnginXplora exploitation to identify a specific *abnormal combustion event*, technically named Low-Speed Pre-Ignition (LSPI), within a dataset of engine experiments. An LSPI event occurs when the charge in the combustion chamber of an engine stochastically auto-ignites prior to the introduction of an ignition source [5]. This early combustion, coupled with a characteristic rapid heat release, results in a knock amplitude of almost two orders of magnitude larger than conventional knock. The underlying cause of this pre-ignition event is still not fully understood.

During the tests, engines are constantly monitored by sensors, which continuously record the physical parameters, such as in-cylinder pressure. This stored dataset enables off-line analysis to be performed. In particular, with regard to the LSPI events, the aim is to analyse a huge amount of collected data from the sensors, in order to provide a tool to support analysts and researchers in their task of understanding the fundamental causes of this phenomenon.

Typically in a test programme, engine performance measurements are continuously recorded at a specified operating conditions throughout the individual engine cycle and averaged over many consecutive engine cycles. The process is repeated for the next operating condition and so forth until a detailed map of optimised engine operating envelope is derived. Abnormal combustion is observed in some regions of the operating map, for example, close to the engine "knock" limit or in the special case of LSPI. These phenomena can occur sporadically in the time series as a single instance or repetitively when initiated by a single event. In both cases, knock and LSPI are characterised by shock waves that produce very high in-cylinder gas pressures within the combustion chamber. The starting point, rate and magnitude of the pressure rise observed during an abnormal cycle can be used to help identify a knock or LSPI event.

IV. EnginXplora GUI for Engine Data

EnginXplora's Graphical User Interface (GUI) is mainly divided in three logical parts (Fig. 1): Overview of the Filters, Filter Results View, and Item Details. This section describes the tool GUI, its main functions of creating, removing and altering the filters as well as the visualisation of the charts.

A. Overview of the Filters

The tool provides an initial overview of the applied filters and their results at a glance through the Euler diagram (left part of Fig. 1). Each circle in the diagram is a pictorial representation of a filter. The circle pictorially encloses dataset items, which satisfy the filter formula, and the circle area is proportional to the number of items which satisfy the filter. For example, Fig. 1 shows three filters labelled as: `Peak>100`, `Peak>120`, and `LSPI`. The *peak* is the pressure in bar measured within the engine chamber. LSPI events are characterised by very high peak pressure. Analysts define filters to reduce the quantity of data and try to analyse the phenomenon (e.g., pre-ignition events in Section III).

The Overview of the Filters shows the relationship among filters and their retrieved results. Circles can intersect each other. Two intersecting circles represent filters which retrieve some common items. A circle A completely inside another circle B is a containment relationship ($A \subseteq B$) which means that the filter A retrieves a subset of the results retrieved by the filter B. For instance, in Fig. 1, the set of results retrieved with the filter `Peak>120` (one result with pressure of 162.92 [bar]) is a subset of the results retrieved with the filter `Peak>100` (two results with pressure of 162.92 [bar] and 110.26 [bar]). They are represented with two circles, where `Peak>120` is fully contained within the circle `Peak>100`.

The Overview of the Filters is interactive. In particular, every time the mouse hovers over a circle the tool shows additional information (i.e., the number of items) in a tool tip. All the circles and intersections are clickable.

B. Create, remove and change filters

During the data analysis, the engineer can, at any time, create a new filter from scratch, or change or remove an existing one. Specifically a **filter** is a mathematical formula, which is applied to each item of the dataset and generates a boolean value to decide whether the item is filtered or not.

EnginXplora provides a window called "*Filter Details*" to directly specify the mathematical filter formula. Fig. 2 shows a simple formula `peak > 120` based on the peak pressure threshold to filter out the items. The engineer can provide a formula that involves multiple variables. For instance, the set of events named `LSPI` (Fig. 1) have been filtered using the formula $casoc < caign$, which involves two variables:

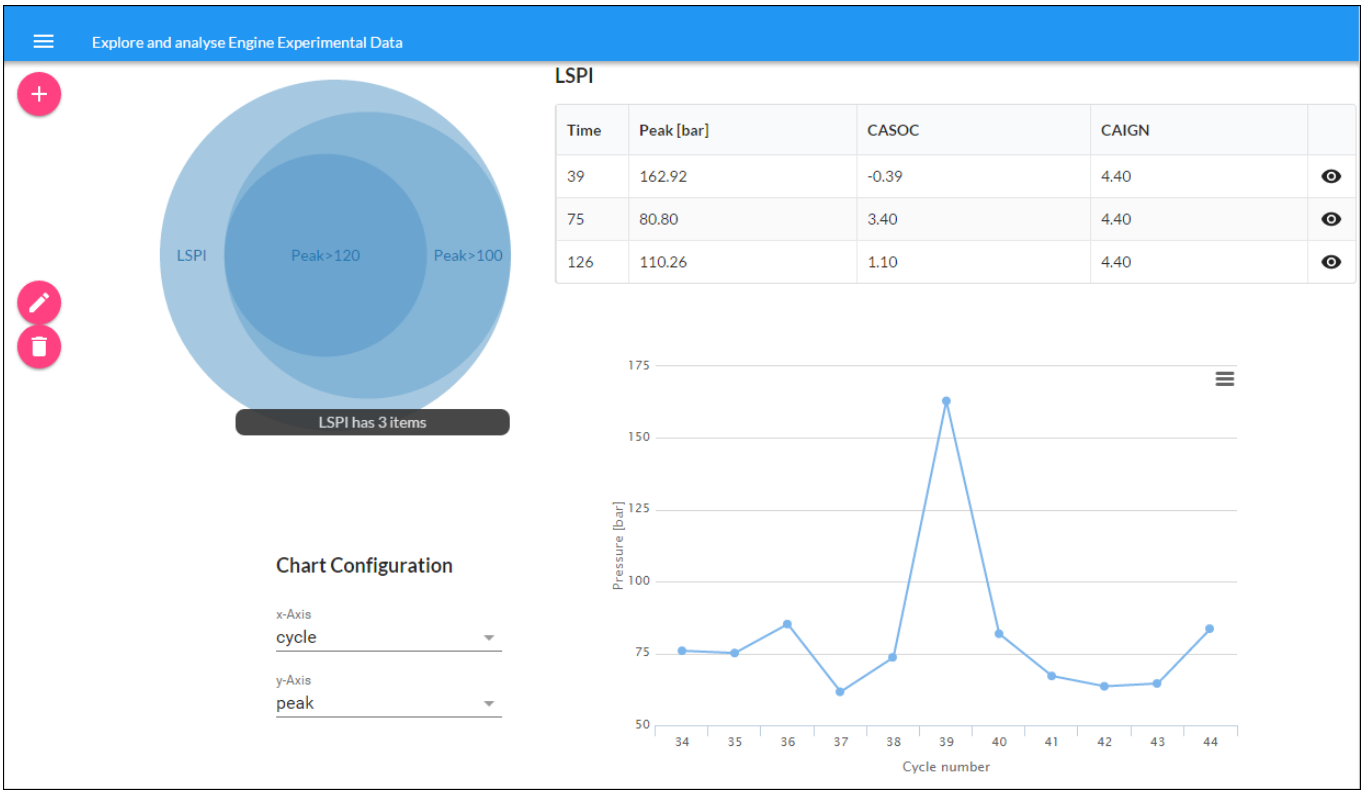


Fig. 1. EnginXplora's Graphical User Interface (GUI). The tool picture shows three filters. Two filters apply a threshold on the *Cylinder 1 Pressure* values which are $Peak > 100$ and $Peak > 120$, since pre-ignition events may be characterised by very high pressures. The filter labelled by LSPI applies the formula $casoc < caign$. The view on the left shows a pictorial representation of the filter results through an Euler Diagram; each formula retrieves a set of items from the dataset that is represented as a circle in the diagram. The table on the top right is showing the engine events filtered by the LSPI identifying pre-ignition events. Among the three filtered events shown in the table, the engineer decided to further explore the first event, and she/he clicked on the first row in order to display the cycles around the selected cycle number 39 in the interactive chart.

the Crank Angle Start Of Combustion (CASOC) and the Crank Angle point of IGnition (CAIGN), both in degrees. A characteristic of a LSPI event is that the combustion (CASOC) starts before the spark plug is triggered (CAIGN), hence the pre-ignition occurs.

In order to identify LSPI events engineers may begin with the use of the mean and standard deviation of the peak pressure. An example of a formula that uses mean and standard deviation is $(peak > mean + 4.7 * stddev)$. Both `mean` and `stddev` are reserved keywords in the system, that are replaced with the exact values during the processing, so within the formula they become a threshold. Both `mean` and `stddev`, as well as other values, are pre-computed and are available with the dataset of the experiment. The word `peak` is a variable indicating the actual peak pressure value. Thus, this filter is independently executed for each peak pressure value in the dataset, returning for each one a boolean value to indicate whether the value is greater than the threshold peak pressure. The filter is executed for each value independently, enabling the high scalability by just partitioning the dataset items in multiple blocks distributed across multiple computing nodes that apply the filter (see Section V for more details and technological implementation). Finally, in order to foster the reuse of existing filters, in the Filter Details window, the user

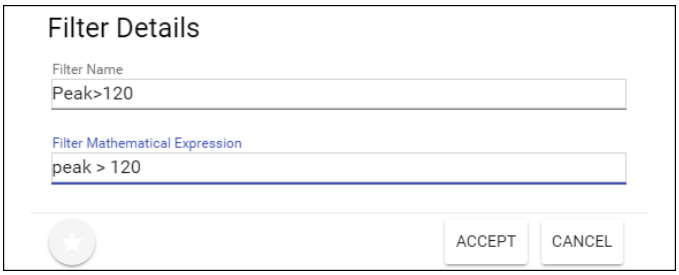


Fig. 2. Filter Dialog where the user enters the mathematical expression to filter the dataset items. The formula in the example filters all the cycles where the pressure in the engine chamber is greater than 120 [bar]. The formula will be parsed and executed in parallel on all items of the large dataset.

can star the filter by clicking on the star icon, inserting it in a repository of named reusable formulas.

In order to remove or change a filter, the user must click on the labelled circle representing the filter, and click on the trash or pencil icon to remove or edit the filter respectively.

C. Filter Results View

The Filter Results View is a table that lists the dataset filtered items for a particular filter formula (right side of Fig. 1). Clicking on a filter (circle) or intersection within the Euler

Fig. 3. Architecture Data flow. The test-bed engine (on the left) records data, which are stored in the HDFS file system. An analyst can send a filter query to the WebServer via his/her browser. Then Apache Spark filters the data based on the query and returns results to the WebServer, which is responsible for the presentation of the data.

diagram triggers the update of the list of items. Of course, a particular filter formula can retrieve many items from the dataset, and in the worst case it retrieves all items; hence the list of results splits the items across multiple pages so that only a small portion of data is shown.

D. Item Details and Chart Visualisation

After the data filtering, the next task for the analyst is to analyse the results shown in the Filter Results View (top right of Fig. 1). Each result can be selected by clicking on the black eye icon. For instance, in Fig. 1, the user has selected the engine cycle number 39. The tool has updated the chart, showing the pressure on the y-axis and the cycles on the x-axis, displaying ten cycles with the selected value in the middle. The chart is configurable and a user can change the x-axis and y-axis using the drop down lists on the chart left-side.

V. System Architecture

This Section describes the EnginXplora architecture shown in Fig. 3. The engineer runs one or multiple experiments on the test-bed engine, which records the data from the engine sensors (e.g., pressure) and stores it on the file system. There is no de-facto standard on how to store engine experimental data into files (e.g., file format, number of files); this is dependent upon the data logger and the logging software used during the experiments, each having its own standard. Hence, native files are saved in their own format and their own ad-hoc naming conventions. The measured physical quantities are often similar, in that they include the same sensor type and locations, but, they can be displayed in different formats.

The architecture is able to load and read data from Matlab files (*.mat), one of the most commonly used raw formats in commercial engine data logging systems. These files can be enormous; to give an example, a logging of all sensors' data for 18 seconds at 2000 rpm (considered low speed testing) takes around 30MB on the file system. An engine can run for several hours with various configurations and test conditions, generating large-scale datasets. To study a physical phenomenon these large-scale datasets must be analysed, a non-trivial task on a general purpose computer.

A *Data lake* is a repository that stores a massive quantity of data in raw native format [6]. This is an emerging trend in the context of big-data where data remains in its original format and has to be read and processed many times. Hence, the system proposed in this paper follows the data lake, experimental data remains in raw formats within the repository and the tool never changes them. The tool, when needed, stores the additional metadata provided by engineers or computed results in separate metadata files in open format (i.e., XML).

In order to support large-scale experimental data analysis, the architecture runs on a cluster configured with Hadoop and Apache Spark. In particular, the engineer must copy the experimental files on the Hadoop Distributed File System (HDFS) of the Intranet HPC resources. HDFS [7], [8] is a fault-tolerant file system able to store huge amounts of data and is particularly suitable to be used with Apache Spark.

The architecture is based on mainstream open source technologies, so it can run both on a private cluster or an external third party High Performance Computing (HPC) service (e.g., Amazon Web Services). The use case described in this paper assumes that the HPC resources are accessible through the Intranet for two main reasons: data confidentiality and the cost to transfer the large-scale data from the private laboratory to the on-line service. Data confidentiality is an important legal aspect, which must be considered when determining the technology used. Research laboratories often run engine experiments for external third party companies, so they are sometimes unable to transfer externally the confidential data out of the internal infrastructures. Once transferred on the HDFS, the files are available for processing through the tool.

Any authorized engineer can use the web browser installed on his/her workstation to access to the engine experimental data through the tool GUI (Fig. 1). This can be done in the same Intranet or also through an Internet connection because the architecture has been designed to leave and process the data on the server-side using the cluster facilities. In addition, the GUI shows only a small subset of the available data and generated analysis results, using the pagination on the tables of data and exploiting the information seeking mantra [9] "*overview first, zoom and filter, then details-on-demand*" as described in other recent works [2], [4], [10]. The tool's web-side uses standard Web 2.0 technologies (i.e., HTML, JavaScript, JSON, SVG, WebComponents), extensively described in literature exploiting Restful services (for more further details, see the architecture of ExploraTool architecture presented in [2]).

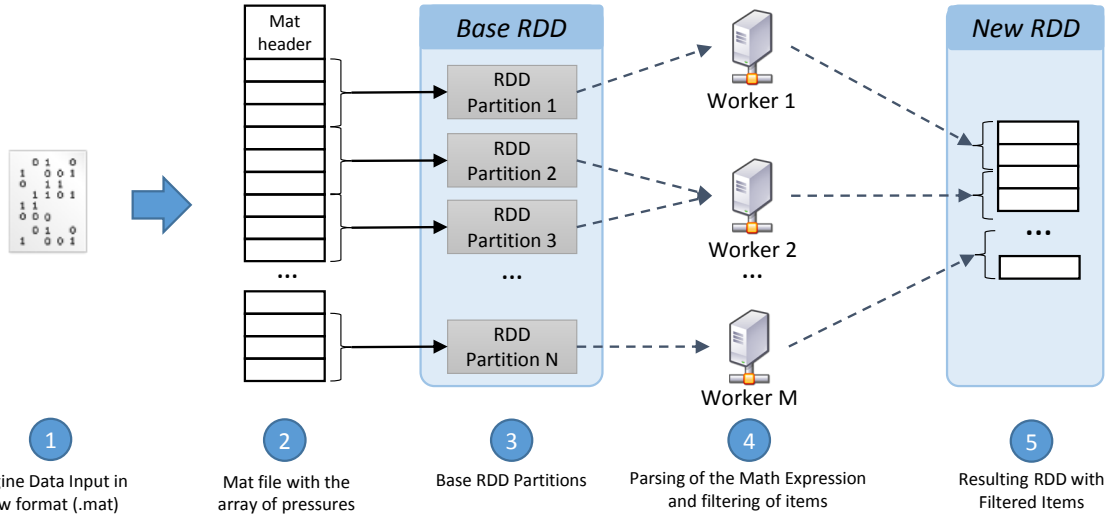


Fig. 4. Filtering of dataset items in parallel using Apache Spark. The steps are: (1) the custom JMATRW library reads the *.mat input file; (2) the Apache Spark *custom reader* called *JMatFileRecordReader* removes the header from the mat file; (3) creates a BaseRDD splitting the mat file into multiple partitions; (4) the custom action executes on each worker parsing the mathematical expression, and applies it for each dataset item (array cell); (5) the new RDD contains the filtered data sent to the WebServer to be shown in the Web Browser.

A. *Parallel Data Processing*

The software stack installed on the cluster is depicted in Fig. 5. At the bottom level there is the hardware, such as physical storage to store the engine data in raw format. These raw files are managed by the HDFS and upon it there is the Apache Spark core. In this paper, the raw data are Matlab files containing huge arrays and matrices of data recorded from the engine sensors. A central Apache Spark concept is the Resilient Distributed Dataset (RDD), which is a collection of data items partitioned across the cluster nodes. Spark RDD is optimised for in-memory processing especially for iterative computations, resulting in faster performance than the Hadoop Map/Reduce. Hence, each cluster computing node (worker node) will have and process a piece of dataset items in parallel.

Fig. 3 shows the process and sequence of steps performed when the user creates a new filter by providing a filter formula. The filtering engine receives the filtering inputs, provided by the users through the GUI and process data accordingly.

Apache Spark has well-known facilities to automatically split textual based big data files, allowing different types of partition schemes, such as the split by lines or by structured records. Engine experimental data are **binary mat files** containing arrays or matrices of measures. Apache Spark does not have a library to parse mat files and split them in partitions. JMATRW[1] is a custom open source library released with LGPLv3 license, specifically designed and developed for this work to read a binary mat file and split its content in multiple partitions to create an Apache Spark RDD. A textual splitting can not be used because 1) the mat files are binary files so there is no available concept of textual lines or records, 2) the mat files have a variable length header portion at their beginning that must be removed and must not be in any of the partition, 3) the file can not be randomly partitioned, but it must be partitioned to have consistent blocks of double values, avoiding the splitting of value bytes into two parts. The JMATRW library has been written using Scala and Java programming languages.

Once the RDD has been created with the partitioned array (Fig. 5), each worker receives a partition of the dataset and the mathematical formula provided by the user through the Filter Window Details (Fig. 2) to run against each dataset item in the partition. Thus, workers run the same actions on their received dataset partition. Hence, each worker filters the data in its partition generating another partition with the array of filtered double numbers. All the resulting partitions together make the filtered results.

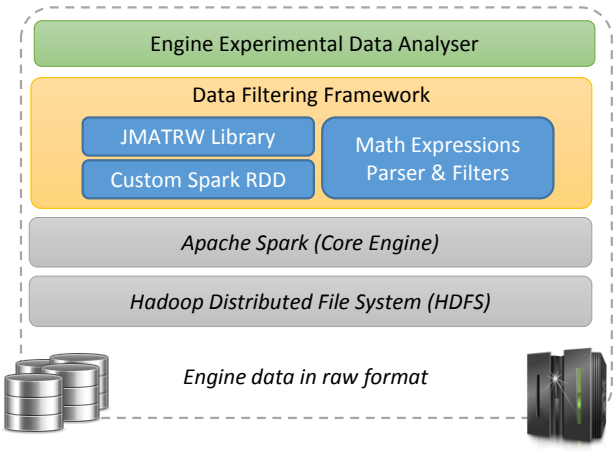
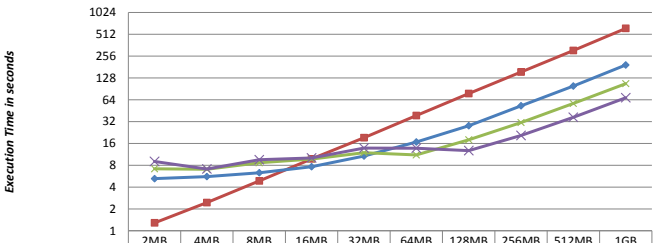


Fig. 5. Stack of software deployed on the cluster. The top two layers have been specifically designed to run the filtering of engine data.

[1]JMATRW to read, parse and split *.mat files has released open source with LGPLv3 license on GITHUB https://github.com/donpir/JMATRW.

B. Performance Evaluation

This Section reports on the architecture's performance benchmark to evaluate the architecture's scalability. Each performance test takes as input the mat input file that contains *the array of numbers*, the *filter* to run over this dataset and the *number of computing workers* to use. The output is the time taken to filter data items. For the benchmark, the file sizes have been increased by a power of two, starting from 2MB up to 1GB. Filtering on each file has been executed in serial and in parallel with two, three and four Apache Spark workers. Each test has been executed ten times, measuring the test execution time, and calculating the mean of the ten runs.



	2MB	4MB	8MB	16MB	32MB	64MB	128MB	256MB	512MB	1GB
Serial Execution	1,28	2,45	4,86	9,79	19,19	38,82	77,84	154,44	305,58	616,85
1 Worker	5,24	5,57	6,29	7,63	10,74	16,73	28,02	52,929	99,02	193,22
2 Workers	7,16	7,04	8,67	9,72	11,93	11,10	17,92	31,03	57,01	106,71
4 Workers	9,00	7,10	9,56	10,11	13,83	13,72	12,71	20,61	36,71	68,20

Fig. 6. Benchmark data of a filter execution over datasets with different sizes from 2MB up to 1GB. Both scales are logarithmic.

Fig. 6 shows the effect of doubling the number of workers on execution time. The use of an additional worker has an execution cost due to their initialisation and the transfer of data chunks. This has significant effect for small files (i.e., less than 16MB), where the serial execution is faster than the use of at least one worker. Due to the initialization costs, we see that doubling the number of workers does not quite reduce the execution time by half. However, the benefits of increasing the number of workers overshadows the initialization costs as the file size increases.

VI. Conclusions and Future Work

This paper describes a web-based tool to filter large datasets of combustion engine experimental data. The practical application (see scenario in Section III) is the identification of pre-ignition events. Due to the quantity and type of data recorded during the experiments, its processing (filtering) requires the use of big data technologies. The software architecture adopted uses Apache Spark in the back end to process and filter data. This work explained the software architecture, which uses Apache Spark in the back-end to process data and apply the filter. The performance benchmark compared different configurations, using different input sizes and numbers of computing workers.

Despite the fact the the tool has been designed to process engine data, its applicability to a variety of different contexts where numerical data must be filtered and analysed through charts (e.g., financial data) is evident. This work represents the first brick for a web-based tool to support the analysis of large engine datasets and open questions remain. An open issue is the heterogeneity of data and formats recorded from the different engine test beds, which will be considered in future work by exploiting techniques experimented within other fields [11].

The identification and analysis of pre-ignition events in engine data is one of a plethora of phenomenon that could be examined. In terms of future work, the goal is the further exploration of a LSPI identification and analysis case study. The aim is to have a predictive tool to help identify under what conditions LSPI events may occur. Finally, it would be interesting to expand the filtering to include in-cylinder optical diagnostics taken during engine testing, alongside the numerical data gathered from current test bed sensors, for a more thorough analysis.

Acknowledgment

The authors would like to acknowledge and thank Dr. Richard Osborne of Ricardo UK Ltd. and Ricardo UK Ltd. for providing the engine test data used within this report and for funding the EngD project currently being carried out by Andrew Harvey.

References

[1] M. Mikulski and S. Wierzbicki, "The concept and construction of the engine test bed for experiments with a multi-fuel ci engine fed with cng and liquid fuel as an ignition dose," *Journal of KONES*, vol. 19, pp. 289–296, 2012.

[2] A. Fish, C. Gargiulo, D. Pirozzi, and V. Scarano, "Simulation repository visualisation and exploration," in *13th IEEE International Conference on Industrial Informatics (INDIN), Cambridge, UK, July 22-24*, 2015, pp. 832–837.

[3] S. Pajer and H. Piringer, "Challenges in creating task-tailored analytical dashboards for the automotive industry," in *Poster Proceedngs of IEEE Vis 2015 (VIS in Practice)*, 2015.

[4] A. Fish, C. Gargiulo, D. Malandrino, D. Pirozzi, and V. Scarano, "Visual Exploration System in an Industrial Context," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 567–575, 2016.

[5] C. Dahnz and U. Spicher, "Irregular combustion in supercharged spark ignition engines–pre-ignition and other phenomena," *International Journal of Engine Research*, vol. 11, no. 6, pp. 485–498, 2010.

[6] T. Priebe and S. Markus, "Business information modeling: A methodology for data-intensive projects, data science and big data governance," in *Big Data (Big Data), 2015 IEEE International Conference on*, Oct 2015, pp. 2056–2065.

[7] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, May 2010, pp. 1–10.

[8] J. Kim, T. Ashwin Kumar, K. George, and N. Park, "Performance evaluation and tuning for mapreduce computing in hadoop distributed file system," in *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*, July 2015, pp. 62–68.

[9] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *IEEE Symposium on Visual Languages,*, 1996, pp. 336–343.

[10] A. Fish, D. Pirozzi, and V. Scarano, "Visual and textual dataset exploration," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2015, pp. 309–310.

[11] C. Gargiulo, D. Malandrino, D. Pirozzi, and V. Scarano, "Simulation data sharing to foster teamwork collaboration," *Scalable Computing: Practice and Experience*, vol. 15, no. 4, pp. 309–329, 2014.