Scalable Data Analysis of BlueGene/L Supercomputer Logs Using Apache Spark

Rishabh Sinha
Scalable Cloud Programming
Masters of Science in Cloud Computing
National College of Ireland
Dublin, Ireland
Email: x21171203@student.ncirl.ie URL: www.ncirl.ie
Video (Project Presentation, Demo, Q&As) URL: https://tinyurl.com/mwrxzakp

***Abstract*—The rapid expansion of data generation in the contemporary world necessitates efficient processing and analysis methods. This report delves into the analysis of a substantial log dataset from the BlueGene/L supercomputer, leveraging the scalability and power of the Apache Spark framework. By implementing a systematic methodology, we address diverse inquiries involving event counts, average durations, and temporal patterns within the log data. This report utilizes Spark's distributed processing capabilities, the project explores several crucial aspects of the log data. Our analysis reveals insights into fatal log entries, occurrences of specific events, and time-based trends. The methodology includes data acquisition, preparation, and transformation, employing both RDDs and DataFrames. The resulting insights provide valuable information about system behavior, highlighting potential anomalies and operational patterns. For effective data visualization, results are converted to Pandas DataFrames for graphical representation using Python's matplotlib library. The integration of Spark's efficient computing power and Python's visualization capabilities creates a comprehensive approach for analyzing and interpreting large-scale log datasets. This project report demonstrates the significance of scalable data processing and analysis in understanding complex system behaviors. The approach showcases the potential of Apache Spark as a robust tool for handling and deriving insights from extensive log data, making it an indispensable asset for data-driven decision-making and operational enhancement.**

***Index Terms*—Apache Spark, SQL, Python, PySpark, RDD**

I. Introduction

The amount of data being generated in today's world is increasing at an unprecedented rate. According to the report by 'Statista', the total amount of data created, captured, copied, and consumed globally is forecast to increase rapidly, reaching 180 zettabytes by 2025 [1]. High-performance computing (HPC) systems are heavily instrumented and generate logs containing information about abnormal events, such as critical conditions, faults, errors and failures, system resource utilization, and about the resource usage of user applications and are generated in huge volumes and with huge velocity. Scalable computing is a key feature for big data analysis and for applications that need to analyze very large and real-time data available from data repositories, social media, sensor networks, smartphones, the Web, and HPC systems in this case. Scalable big data analysis today can be achieved by parallel implementations that are able to exploit the computing and storage facilities of high-performance computing (HPC) systems and cloud platforms. [2].

There are various tools and technologies that can be utilized such as Hadoop which is an open-source software platform that stores and processes big data in a distributed computing environment across hardware clusters such as with the use of Map Reduce. This distribution allows for faster data processing. The framework is designed to reduce bugs or faults, be fault tolerant, be scalable, and process all data formats. Map Reduce is a disk-based computing model whereas another alternative is Apache Spark which provides real-time in-memory processing. Apache Spark is an open-source distributed computing system used for big data processing. It can process large amounts of data in parallel across a cluster of computers. Spark is designed to be fast and general-purpose, with support to various different libraries. [3]

In addition to these tools mentioned above, there are other tools that can be used for HPC systems such as Performance Analysis Tool for HPC and Big Data Applications on Scientific Clusters [4] which helps identify performance bottlenecks or debug the performance issues in large-scale scientific applications and scientific clusters by using state-of-the-art open-source big data processing tools.

This paper presents a statistical analysis of the BGL log data set, which records the events and activities of the BlueGene/L supercomputer. The BlueGene/L is a joint research project between IBM and the Lawrence Livermore National Laboratory, and it consists of 131,072 processors and 32,768GB of memory [5]. The BGL log data set contains detailed information about the runtime of the supercomputer, such as timestamps, dates, node identifiers, message types, system components, log levels, and message contents. The BGL log data set spans from '2005-6-03' to '2006-01-04' and has '4747963' records in total. The data set is expected to grow rapidly as the supercomputer continues to operate. The objective of this paper is to acquire, store, pre-process, and do the data computation task on the BGL log data using scalable programming. The dataset contains information about the events and activities of the BlueGene/L supercomputer at Lawrence Livermore National Labs. The BGL log data is converted into a structured format that enables scalable data

processing and analysis using Apache Spark with RDD, SQL, and Python Dataframe API. These technologies are used to perform various tasks such as data acquisition, transformation, storage, pre-processing, and computation on the BGL data set. The paper also presents the insights derived from the BGL log data, which reveal the performance, behavior, and characteristics of the BlueGene/L HPC system.

This paper is structured as follows. In Section 2, we survey some of the previous studies that have utilized HPC logs for various research purposes and we critically examine their approaches. In Section 3, we present the methods that we employed in this paper to process and analyze the BGL log data using Spark and its libraries. In Section 4, we report the results of our data analysis and the insights that we derived from them. Finally, we conclude the paper with a summary of our findings and contributions.

II. Related Work

The statistical analysis of HPC logs has been an active research area in recent years. Many studies have been conducted to explore the characteristics and patterns of HPC logs and to develop efficient and scalable methods for processing and analyzing them. In particular, MapReduce, Hadoop, and Spark have been widely used as the underlying technologies for statistical analysis of HPC logs due to their ability to handle large-scale data processing and their support for parallel computing.

The paper titled "Analyzing and predicting job failures from HPC system logs" [6] addresses the challenges of analyzing and predicting job failures in high-performance computing (HPC) systems using system logs [6]. The author discusses the challenges posed by the increasing volume and complexity of data generated by HPC systems, as well as the need for scalable and efficient processing frameworks. The paper highlights the importance of tools and technologies such as Hadoop, MapReduce, and Spark in addressing the challenges of big data analytics. It discusses the advantages and limitations of these frameworks and their suitability for different types of applications. The author emphasizes the need for scalable programming models and systems that can handle the large-scale and real-time nature of HPC log data and discussed about the importance of scalability and performance for efficient data processing and analysis techniques that can handle the massive volumes of log data generated by HPC systems. The paper implements discusses the scalability and performance characteristics of frameworks like Spark, which is known for its ability to process large amounts of data in parallel across a cluster of computers. The paper discusses the challenges and opportunities in analyzing and predicting job failures from HPC system logs. The author describes the process of converting the log data into a structured format that enables scalable data processing.

In [6] the author also addressed the challenges of analyzing and predicting job failures in high-performance computing (HPC) systems using system logs. The author discusses the challenges posed by the increasing volume and complexity of data generated by HPC systems, as well as the need for scalable and efficient processing frameworks. The paper highlights the importance of tools and technologies such as Hadoop, MapReduce, and Spark in addressing the challenges of big data analytics with the advantages and limitations of these frameworks and their suitability for different types of applications. Scalability and performance characteristics of frameworks like Spark are also discussed in this paper.

Similarly, the paper [7] discussed the challenges and insights related to big data analytics, MapReduce, and Spark in the context of analyzing HPC system logs. The paper highlights the importance of analyzing HPC system logs, which contain information about abnormal events, system resource utilization, and the resource usage of user applications [7]. The authors emphasize the need for scalable approaches to analyze and correlate these logs to gain detailed insights into system behavior and its impact on applications. One key insight discussed in the paper is the use of statistical and data mining techniques for analyzing HPC system logs [7]. The authors mention the use of monitoring frameworks like Ganglia and Nagios for understanding system resource usage and detecting abnormal system conditions. They highlight the need for postmortem analysis of system logs to extract statistical properties and failure characteristics. The paper utilizes frameworks like MapReduce and Spark for scalable data processing and analysis [7]. The author discussed the advantages and limitations of frameworks like Spark in terms of scalability and processing speed.

The paper titled "A comprehensive performance analysis of Apache Hadoop and Apache Spark for big data on low-end clusters" [8] provides a comprehensive analysis of the challenges and insights related to big data analytics, MapReduce, and Spark in the context of low-end clusters [8]. The paper highlights the increasing volume and complexity of digital data generated by various sources, such as sensors, mobile devices, and social media platforms, leading to the emergence of big data challenges. Comparison of Apache Hadoop and Apache Spark for big data processing on low-end clusters [8] is discussed and the performance is evaluated comparison of Apache Hadoop and Apache Spark for big data processing on low-end clusters [8] evaluated using various workloads and datasets utilizing benchmarking techniques. They analyse parameters such as execution time, throughput, and speedup to assess the scalability and efficiency of Hadoop and Spark. The paper also discusses the programming models and systems used in big data analysis. It compares the type of parallelism and level of abstraction provided by different frameworks, highlighting their programming capabilities for expressing parallel operations and hiding low-level details. The authors emphasize the importance of selecting the appropriate programming solution based on skills, hardware availability, and application domains and conducted experiments using the HiBench suite and select specific workloads, such as WordCount and TeraSort, to represent different types of jobs. The results of these experiments demonstrate the superior performance of Spark over Hadoop in terms of execution time,

throughput, and speedup.

The paper titled "Addressing big data problem using Hadoop and MapReduce" [9] discusses the challenges and insights related to big data analytics, Hadoop, and MapReduce. It emphasizes the need for scalable storage, processing, and analysis capabilities to handle huge amounts of data. The authors discussed the architecture and components of Hadoop, including the Hadoop Distributed File System (HDFS) and the MapReduce programming model. They highlight the advantages of Hadoop in terms of fault tolerance, scalability, and processing large datasets in a distributed manner. The paper also discusses the challenges in big data analytics, such as data variety, velocity, and complexity [9]. The authors emphasize the need for efficient data processing and analysis techniques that can handle the diverse and rapidly changing nature of big data. They highlight the role of MapReduce in enabling parallel processing and distributed computing for big data analytics and use Hadoop and MapReduce for scalable data processing and analysis [9]. The authors describe the process of data ingestion, storage, and processing using Hadoop's distributed file system and the MapReduce programming model. They discuss the advantages of Hadoop's fault tolerance and scalability in handling large-scale data processing. The challenges of scalability and performance are also addressed in the paper [9]. The authors discuss the need for scalable frameworks that can handle the increasing data volumes and provide efficient data processing and analysis. They highlight the advantages of Hadoop's distributed computing model and MapReduce's parallel processing capabilities. The paper provides insights into the challenges and opportunities in big data analytics, Hadoop, and MapReduce [9]. It discusses the architecture and components of Hadoop, the role of MapReduce in enabling parallel processing, and the challenges in big data analytics. The paper highlights the advantages of Hadoop and MapReduce in terms of fault tolerance, scalability, and distributed computing for big data processing and analysis.

The paper titled "Benchmarking Apache Spark and Hadoop MapReduce on Big Data Classification" by Tekdogan et al. (2021) focuses on benchmarking and comparing the performance of Apache Spark and Hadoop MapReduce for big data classification tasks [10]. The paper provides insights into the challenges and advantages of these frameworks in the context of big data analytics. The authors discuss the concept of big data mining and the need for efficient data processing techniques to extract valuable information from large volumes of unstructured data. They highlight the importance of benchmarking frameworks like Spark and MapReduce to evaluate their performance and scalability in handling big data classification tasks. The paper utilizes Amazon's Elastic MapReduce (EMR) as the project environment for benchmarking Spark and MapReduce [10]. The authors configure an m5.xlarge cluster with a master and multiple slave nodes to perform the benchmarking experiments. They use HiBench, a benchmark suite for big data processing, to compare the performance of Spark and MapReduce. The experiments conducted in the paper show that Spark outperforms MapReduce in terms of execution time for big data classification tasks [10]. The authors observe a growth pattern in execution time for both Spark and MapReduce as the input size increases, but Spark exhibits shorter time durations compared to MapReduce. However, it is noted that Spark consumes more memory compared to MapReduce. It also discusses the impact of tuning parameters on the performance of Spark and MapReduce [10]. The authors aim to identify the parameters that have the highest impact on the performance of these frameworks. They use a trial-and-error approach to tune the parameters under different experimental settings. The paper provides insights into the performance and scalability of Spark and MapReduce for big data classification tasks [10]. It highlights the advantages of Spark in terms of execution time, but also notes its higher memory consumption. The paper emphasizes the importance of benchmarking frameworks and tuning parameters to optimize the performance of big data analytics tasks.

The paper titled "A Big Data Analytics Framework for HPC Log Data: Three Case Studies Using the Titan Supercomputer Log" [11] focuses on solving similar challenges as mentioned above. It emphasizes the challenges of handling large-scale log data and the importance of efficient data processing techniques. One key insight discussed in the paper is the development of a big data analytics framework for HPC log data using Spark and MapReduce [11]. The authors present three case studies that demonstrate the application of this framework to analyze the log data from the Titan supercomputer. The paper discussed the challenges in analyzing HPC log data, such as the unstructured and voluminous nature of the data [11]. The authors highlight the need for scalable frameworks like Spark and MapReduce to handle large-scale log data and enable efficient data processing and analysis and how Spark and MapReduce can be used for scalable data processing and analysis of HPC log data [11]. The authors describe the process of data acquisition, transformation, and computation using these frameworks. The advantages of Spark's in-memory processing and MapReduce's distributed computing model are also discussed by the author in this paper. The authors discuss the scalability of Spark and MapReduce in handling large-scale log data and compare their performance in terms of execution time and efficiency. They highlight the advantages of Spark's in-memory processing for faster data processing.

The paper titled "Leveraging Comprehensive Data Analysis to Inform Parallel HPC Workloads" focuses on the challenges and insights related to big data analytics, MapReduce, and Spark in the context of parallel high-performance computing (HPC) workloads [12]. The paper provides a comprehensive analysis of the data generated by parallel HPC workloads and the techniques used to inform workload optimization. The paper emphasizes the challenges of analyzing and understanding the behavior and performance of parallel HPC workloads. The authors propose a methodology that combines statistical analysis, visualization, and machine learning techniques to analyze and understand the characteristics of parallel HPC workloads. The paper also discusses the challenges in analyzing and optimizing parallel HPC workloads, such as work-

load characterization, performance prediction, and workload balancing [12]. The paper proposes a performance prediction model based on machine learning techniques that can estimate the execution time of parallel HPC workloads. The paper also proposes a workload-balancing strategy based on data analysis techniques that can balance the workload distribution among different nodes. The paper uses comprehensive data analysis techniques to analyze and inform parallel HPC workloads [12]. The authors describe the process of data collection, preprocessing, analysis, and visualization using tools and techniques such as R, Python, and machine learning algorithms. The paper uses frameworks like MapReduce and Spark for scalable data processing and analysis. The authors discuss the scalability of the proposed methodology and its applicability to large-scale parallel HPC workloads. They highlight the advantages of using Spark for scalable data processing and analysis. They also compare the performance of MapReduce and Spark in terms of execution time and efficiency.

The paper titled "A Comparative Survey of Big Data Computing and HPC: From a Parallel Programming Model to a Cluster Architecture" [13] provides a comprehensive analysis of big data computing and high-performance computing (HPC) with a focus on parallel programming models and cluster architectures [13]. The paper discusses the challenges and insights related to big data, MapReduce, and Spark. It emphasizes the need for novel architectures, programming models, and systems to address the complexity and high velocity of big data. The authors mention that big data analysis techniques can extract useful information and produce valuable knowledge for various domains, including science, the economy, and health. The paper discusses the MapReduce and Spark frameworks in terms of programming models. It mentions that Spark is commonly used for developing in-memory applications, such as interactive queries and batch processing, and provides a wide range of libraries for various applications. The authors highlight the advantages of Spark, such as its flexibility and fast processing speed, compared to Hadoop. They also mention that MapReduce is suitable for batch processing only and may not be efficient for highly iterative applications. The paper discusses the challenges and considerations in selecting the best programming solution for big data analysis based on factors such as skills, hardware availability, application domains, and community support. It compares different frameworks, including Hadoop, Spark, and Flink, and highlights their advantages and limitations. The paper also provides a comparative analysis of big data computing and HPC from the perspective of the parallel programming model layer, middleware layer, and infrastructure layer [13].

These papers have explored characteristics and patterns in HPC logs, focusing on efficient processing and analysis methods, and mostly technologies like MapReduce, Hadoop, and Spark have been applied due to their capacity for large-scale data handling and parallel computing. The paper "Analyzing and predicting job failures from HPC system logs" [6] highlights the importance of scalable processing frameworks and discusses the benefits and drawbacks of frameworks like Hadoop and Spark. While emphasizing Spark's ability to process vast data volumes, it also acknowledges limitations in terms of memory consumption. Similarly, [7] delve into big data analytics with MapReduce and Spark, stressing scalability and correlation of logs for system insights. [8] evaluates Hadoop and Spark on low-end clusters, showcasing Spark's superior execution time while noting its higher memory use. Another work underscores Hadoop's scalability and fault tolerance for big data, acknowledging challenges in data variety [9]. A performance comparison in [10] finds Spark outperforming MapReduce in classification tasks, albeit with higher memory consumption. Addressing HPC logs, a framework paper develops Spark and MapReduce solutions for analysis, focusing on scalable data handling. Another study combines analysis and machine learning for optimizing parallel HPC workloads, using MapReduce and Spark for scalability. Lastly, a comparative survey [13] emphasizes the need for novel architectures and programming models for big data analysis, highlighting Spark's flexibility and speed compared to MapReduce's batch processing.

III. Methodology

As reviewed in the 'Related Work' section we will be using the Apache Spark framework to do data computation tasks over the BGL data set. In this section, we outline the systematic approach taken to acquire, store, pre-process, and perform data computation tasks using the Spark framework within a Jupyter Notebook environment.

The objective is to analyze the extensive log dataset generated by the BlueGene/L supercomputer at Lawrence Livermore National Labs. This high-performance computing system boasts an impressive configuration of 131,072 processors and a substantial 32,768GB of memory. The raw log file 'BGL.log', which encapsulates a wealth of runtime information is downloaded from the Zenodo [14] 'Loghub' data sets a collection of the system logs. The BGL log data set contains detailed information about the runtime of the supercomputer, such as timestamps, dates, node identifiers, message types, system components, log levels, and message contents. The BGL log data set spans from '2005-6-03' to '2006-01-04' and has '4747963' records in total. The dataset contains records about the events and activities of the BlueGene/L supercomputer at Lawrence Livermore National Labs.

In this project, the Spark Framework was utilized within a Jupyter Notebook environment using PySpark. PySpark is an analytics engine that allows for large-scale data processing in a distributed environment using high-level Python API and provides a PySpark shell for interactively analyzing our BGL data set. As it supports higher-level tools including Spark SQL for SQL and DataFrames and pandas API on Spark for pandas workloads it becomes easy to do analysis on our log dataset with '4747963' records (ongoing) and achieve the result in a scalable and optimized manner.

After reviewing the related paper like in [6] [8] [10] it can be seen that Spark gives better computation and higher performance because of Spark's ability for in-memory data

processing and its support to other technologies such as HDFS, Elasticsearch, Kafka, MongoDB, and plenty, plenty more and has tight integration with different components within itself to perform data computation tasks.

So it means if we configure it with the right nodes with enough memory in them, we can actually load our entire dataset, our entire distributed dataset into a computer's memory or RAM. As RAM is very highly optimized for read/writes and it's very close to the CPU, so it's quickly accessed and has a lot of bandwidth between RAM and the CPU. So the ability of Spark to do in-memory data processing will increase the speed and performance at which we can analyze our datasets. If the data is more than the size of the memory the data the remaining data is processed on disk. On the other hand, Hadoop needs to be integrated with the various other components for e.g. to process structured data on Hadoop Hive needs to be installed, for machine learning we need Mahout to be installed which is not included with Hadoop. This can give rise to various different problems such as component integration problems, technical dept in an organization, and also the data is processed over disk in the case of Hadoop.

Spark Core is the main engine that is responsible for scheduling, distributing datasets, monitoring running applications, and has APIs that define RDD. It works on top of cluster manager and can be installed over various cluster managers such as Hadoop YARN (very powerful cluster manager), and Apache Mesos, and can also use Sparks standalone schedular. Spark can fetch data from HDFS, Local File System, Cassandra, Amazon S3, etc., and support various different file formats and other Hadoop input formats. There are other Sparks components, for example, SparkSQL, MLib, etc that can be optimized further by optimizing Spark Core as these work on top of Spark Core and these being Spark's own components it does not require extra effort into installing and maintaining other components.

We have used SparkSQL in this project which is a Spark package for working with structured datasets and can help in programmatic data manipulation supported by RDDs in Python. RDD (Resilient distributed dataset) is a fault-tolerant dataset and in RDD we have two types of operations performed that is operations and actions. Transformations take data that is in an RDD and transform it from one form into another that's transforming the current RDD into a new RDD. Apache Spark is also known for its lazy transformation and lazy transformation means that execution does not start until an action is triggered. RDD maintains a record of transformations in DAG (Directed Acyclic Graph) to increase manageability and give better computation and higher performance and reduces some of the complexities of operations. There are some limitations of RDD that as it can not easily process data as easily as we can with SQL and also can not utilize Spark's "catalyst Optimiser". SparkSQL stack core has a "DataFrame API" which is based on RDD Dataframe and can also utilize Spark's "catalyst Optimiser". Spark's "catalyst Optimiser" decides which actions to perform in which sequence for the increased performance which is executed in the spark core execution engine.

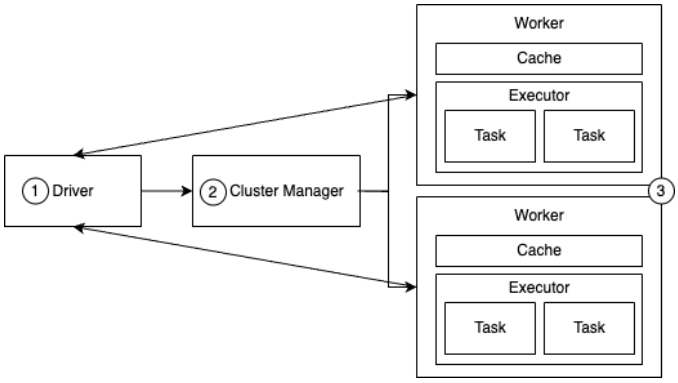

Fig. 1: Apache Spark Working Diagram [15]

1.explains the working of Spark on the small-scale distributed deployment of Spark. First, is the "Driver" (marked as (1) in Figure 1.) which is basically a job or a code that is asking the question. "Driver" operates through a "Cluster Manager" (marked as (2) in Figure 1.) in a distributed mode and can be called a "Master Node" which keeps track of all the "Worker nodes" (marked as (3) in Figure 1.). "Driver", asks that question to the "Cluster Manager" and "Cluster Manager" then sends that question to the "Worker Nodes". "Worker Nodes" can be on top of HDFS or ElasticSearch nodes or other data storage. "Worker Nodes" has a built-in "cache" which is an advantage for frequently repeated Jobs to provide greater performance. Whether doing full-memory processing or not built-in "cache" has some level of in-memory cache to speed up data analysis. Another component of "Worker Nodes" is the "Executor" which performs the subtasks and can cache them if necessary and then return the results back to the "Driver". To scale out our cluster computing to achieve scalability we can add more "Worker Nodes" as necessary to handle large datasets and get quick results achieving scalability and performance.

A. Data Analysis

As a part of this project, I will be doing the data analysis on the BGL dataset and programmatically acquire, store, preprocess, and perform data computation tasks on the BGL data set answering the below question with the use of PySpark. I have also plotted the graph to give more visual insights into the data analysis. Also, I will be discussing the methodology that can be adapted for analysis on a similar set of questions:

1) How many fatal log entries in the month of September resulted from a "major internal error"?
2) For each day of the week, what is the average number of seconds over which "re-synch state events" occurred?
3) What are the top 5 most frequently occurring hours in the log?
4) Which node generated the smallest number of "APPUNAV" events?
5) On which date was the latest fatal kernel error resulting in an rts panic?

Data acquisition and preparation are two of the most important steps in big data analysis. Data acquisition involves collecting data from various sources such as databases or other sources. Here, in this case, we are reading the log data set from our local file system and converting it to RDD DataFrame. In the Data Preparation phase, the columns of the dataset are defined with the DataTypes to involve in further data computation tasks and analysis which is important as it can lead to incorrect conclusions or decisions. Data acquisition and preparation are common for all the questions to perform data analysis. This Phase is the most important and critical part of any data analytics pipeline. As a part of this project, the RDD Dataframe is saved as *'bgl_log_df'*. The same is described in **??**

Data Acquisition and Data Preparation

0: **Step 1:** create a SparkSession object.
0: Use *SparkSession.builder.appName* to set the name of the application as 'ScpProject'.
0: **Step 2:** Use SparkSession object to read the BGL.log file and store in *'bgl_log_df'* Data Frame. *'inferSchema=True'*
0: Use SparkSession object to read the BGL.log file and store it in Data Frame.
0: Set *'inferSchema=True'* and *header=False* as there is no header line in the BGL.log DataFrame.
0: **Step 3:** Use *toDF()* function is used to convert an RDD to a new RDD DataFrame with new column names that describe the BGL log data set records and its columns.
0: **Step 4:** Define the schema of the data set.
0: : Use *withColumn()* function to update the existing column in a DataFrame. *cast()* function can be used to cast the "*timestamp*", "*date*", and "*date_and_time*" columns of the "bgl_log_df" DataFrame to *TimestampType* and *DateType*.
0: −'alert_message_flag' as StringType
0: −'timestamp' as TimestampType
0: −'date' as DateType
0: −'node' as StringType
0: −'date_and_time' as TimestampType
0: −'node_repeated' as StringType
0: −'message_type' as StringType
0: −'system_component' as StringType
0: −'level' as StringType
0: −'message_content' as StringType
0: **Step 5:** Parsing the Timestamp and Datetype column
0: Use to_date(),to_timestamp() function to convert the columns to '*yyyy.MM.dd*' , '*yyyy-MM-dd-HH.mm.ss.SSSSSS*' format.
0: **Result** We should have a BGL Log RDD DataFrame and store it as *'bgl_log_df'* that will be used to perform data computation tasks. =0

Below is the analysis performed to find "How many fatal log entries in the month of September resulted from a "major internal error" recorded in the BGL log data set recorded from '2005-6-03' to '2006-01-04' and has '4747963' records in total. A new RDD DataFrame is created after filtering out the DataFrame to show only "FATAL" log entries. There are other distinct 'level' in the log dataset such as INFO, ERROR, WARNING, SEVERE, FATAL, Kill, and FAILURE. We need to apply another transformation on the created RDD DataFrame to show the record only for 'September' month from "Date" Column of the bgl log data set. Once we have RDD DF with all the records of fatal log entries in the month of September. Finally, we can use contains() function to fetch "major internal error" string from the 'message_content' column and store it in the new RDD Dataframe and perform count() action to show the count of the fatal log entries in the month of September resulted from a "major internal error". The above approach uses 'filter' design pattern using *filter()* and *count()* functions. **??**

Data Analysis Task 2: How many fatal log entries in the month of September resulted from a "major internal error"

0: **Step 1:** Filter *'bgl_log_df'* Dataframe 'level' column to show only 'FATAL' logs in the month of 'September'filtered out from the 'date' column of the data set and store it in new RDD DataFrame.
0: *.filter ("level" == "FATAL") & ("date" == 9)*
0: **Step 2:** New RDD Dataframe is filtered to only show the records which contain the string "major internal error" in "message_content"
0: Use .contains("major internal error") and .filter function to filter the DataFrame and create a new RDD DataFrame
0: **Step 3:** Invoke count() action to count the number of entries in the DataFrame that was created in Step 3.
0: **Result** We will have the count of fatal log entries in the month of September resulting from a "major internal error. =0

In Task 6: **??** for the list of day of the week with the average number of seconds over which "re-synch state events" occurred. We need to first Filter *'bgl_log_df'* Dataframe to only show the records which contain the string "re-synch state event" in "message_content" and also extract the seconds from the event record entry in the dataset.se the *regexp_extract* from pyspark SQL function library to extract the re-synch state events that include all dcr 0x events and the seconds' value from the string and store the seconds' value in the new column in the new DataFrame. and then we can extract the 'Day' from the 'date_and_time' column and store it in the new column 'Day' to enable us to group the events day-wise. we need to drop the records from the data frame that doesn't record seconds as the seconds string was not available for that specific record entry. We then need to Perform GROUP BY 'DAY'and AGG(AVG seconds) and print aggregation in the new 'average_seconds' column and store it in the new DataFrame. This new data frame can be printed to show the list of days of the week with the average number of seconds over which "re-synch state events" occurred. Aggregation and filter design pattern is used to perform scalable data computation task to answer the task query.

Data Analysis Task 6: For each day of the week, what is the average number of seconds over which "re-synch state events" occurred

0: **Step 1:** Filter *'bgl_log_df'* Dataframe to only show the records which contain the string "re-synch state event" in "message_content"
0: Use the *regexp_extract* from pyspark SQL function library to extract the re-synch state events that include all dcr 0x events and the seconds' value from the string and store the seconds' value in the new column in the new DataFrame.
0: **Step 2:** Extract 'Day' from the 'date_and_time' column and store it in the new column 'Day'.
0: **Step 3:** Drop the NA from the 'seconds' column in the data frame.
0: Use *.na.drop* function
0: **Step 4:** Perform GROUP BY 'DAY' , AGG(AVG seconds) and print aggregation in new 'average_seconds' column and store in new DataFrame
0: **Step 5:** Print the DataFrame created in step 4.
0: **Result** We will have the list of day of the week with the average number of seconds over which "re-synch state events" occurred. =0

In task 12: **??** we need to Use the 'createOrReplaceTempView()' method to create a temporary view 'bgllogtable' of the DataFrame 'bgl_log_df'. From the created temporary view we need to select the hour of the date_and_time column and count the number of entries for each hour. we then need to group the results by hour and order them by Occurance_Count in descending order. Use 'session' object to execute an SQL query and display the results. Finally, it displays the results in descending order with the list of hours with the highest frequency first. Scalability and performance are managed by Spark Core and pysparkSQL in-built technology stack.

Data Analysis Task 12: What are the top 5 most frequently occurring hours in the log?

0: **Step 1:** Use the 'createOrReplaceTempView()' method to create a temporary view 'bgllogtable' of the DataFrame 'bgl_log_df'
0: **Step 2:** SELECT HOUR from (date_and_time) column AS hour_HH,
0: **Step 3:** COUNT ALL AS Occurance_Count
0: **Step 4:** GROUP BY HOUR(date_and_time)
0: **Step 5:** ORDER BY Occurance_Count DESC
0: **Step 6:** Use the 'session' object to execute an SQL query and display the results.
0: **Result** The result is displayed in descending order with the list of hours with the highest frequency first and hence we can get the top 5 most frequently occurring hours in the log. =0

In task 16: **??** we need to Use the 'createOrReplaceTempView()' method to create a temporary view 'bgllogtable' of the DataFrame 'bgl_log_df'. From the created temporary view we need to select the "node" column and count the number of entries for each node where the "alert_message_flag" is equal to 'APPUNAV' and then group the results by node and order them by the number of "num_appunav_events". Finally, it returns the number of "num_appunav_events" for each node and that answers the query of task 16.

Data Analysis Task 16: Which node generated the smallest number of "APPUNAV" events?

0: **Step 1:** Use the 'createOrReplaceTempView()' method to create a temporary view 'bgllogtable' of the DataFrame 'bgl_log_df'
0: **Step 2:** SELECT node, COUNT ALL AS num_appunav_events FROM bgllogtable temporary view.
0: **Step 3:** WHERE alert_message_flag = 'APPUNAV'
0: **Step 4:** GROUP BY node
0: **Step 5:** ORDER BY num$_appunav_events$;
0: **Step 6:** Use the 'session' object to execute an SQL query and display the results.
0: **Result** As a result it returns the number of "num_appunav_events" for each node in ascending order displaying which node generated the smallest number of "APPUNAV" events first. =0

In task 17: **??** we need to Use the 'createOrReplaceTempView()' method to create a temporary view 'bgllogtable' of the DataFrame 'bgl_log_df'. From the created temporary view we need to select the "date" column where the "level" is equal to 'FATAL' and the "message_content" contains the string 'rts panic'. It then orders the results by "date_and_time" in descending order. The first entry of the result will be the latest date on which a fatal kernel error resulting in an rts panic occurred.

Data Analysis Task 17: On which date was the latest fatal kernel error resulting in an rts panic

0: **Step 1:** Use the 'createOrReplaceTempView()' method to create a temporary view 'bgllogtable' of the DataFrame 'bgl_log_df'
0: **Step 2:** SELECT date FROM bgllogtable temporary view,
0: **Step 3:** WHERE level = 'FATAL' AND message_content LIKE '%rts panic%'
0: **Step 4:** ORDER BY date_and_time DESC;
0: **Step 5:** Use the 'session' object to execute an SQL query and display the results.
0: **Result** Prints the result in descending order. The first entry of the result will be the latest date on which a fatal kernel error resulting in an rts panic occurred. =0

1) Data Visualization: Further, the results of the above tasks that are performed using different design patterns such as aggregation patter, filter pattern, etc., or by using PysparkSQL can be visualized and plotted in graph format. The results from the PysparkSQL can be visualized by first converting them into

Pandas DataFrame to use the Python matplotlib library to plot the graph. We can also utilize GraphX to visualize the data.

In conclusion, this methodology outlines the systematic approach taken to address the posed questions. By carefully considering the intricacies of each inquiry, we explored a robust framework for analysis. The use of specific design patterns has played a pivotal role in enhancing the scalability of our computational efforts. We have adhered to proven design patterns that not only promote efficiency but also ensure the seamless expansion of our computing processes as needed. The methodological choices detailed here aim our dedication to achieving both precision and efficiency in addressing the tasks assigned as a part of this project. In the next section, we will discuss the results of the above methodologies when implemented.

IV. Results

Task 2: How many fatal log entries in the month of September resulted from a "major internal error"?

- Total number of logs in BGL dataset: total_logs_count = 4747963
- Total number of 'FATAL' logs in BGL dataset: fatal_logs_count = 855195
- Total number of 'FATAL' logs in BGL dataset in the month of September: fatal_logs_september_count = 81818
- Total number of 'FATAL' logs in BGL dataset in the month of September resulting "major internal error": **major_internal_error_september_count = 10**

Task 6: For each day of the week, what is the average number of seconds over which "re-synch state events" occurred?

day	average_seconds
Tuesday	8538.42
Friday	6954.14
Thursday	8412.63
Monday	14511.81
Wednesday	11170.69
Saturday	17872.34
Sunday	16091.97

TABLE I: Summary of Average Seconds by Day over which "re-synch state events" occurred

Task 12: What are the top 5 most frequently occurring hours in the log?

hour_HH	Occurrence_Count
11	388648
10	379141
12	354389
8	315017
9	308801

TABLE II: Hourly Occurrence Count

Task 16: Which node generated the smallest number of APPUNAV events?

- Total Number of nodes with "APPUNAV" events: smallest_num_appunav_events = **512**

node	num_appunav_events
R44-M1-NC-I:J18-U01	1
R64-M0-NC-I:J18-U01	1
R71-M1-NC-I:J18-U11	1
R76-M1-N4-I:J18-U11	1
R62-M0-N4-I:J18-U11	1
R42-M0-N4-I:J18-U01	1
R60-M1-N8-I:J18-U11	1
R41-M0-N0-I:J18-U01	1
R74-M1-N4-I:J18-U11	1
R71-M1-NC-I:J18-U01	1

TABLE III: Top 10 rows out of 512 with the node that generated the smallest number of APPUNAV events

Task 17: On which date was the latest fatal kernel error resulting in an rts panic?

The total number of fatal kernel errors resulting in an rts panic: total_fatal_rts_panic_count = 3983

Latest fatal kernel error resulting in an rts panic:

Date
2006-01-03

TABLE IV: Latest date of fatal kernel error resulting in an rts panic

A. Extras results

Task 3: How many fatal log entries that occurred on a Monday resulted from a "machine check interrupt"?

Number of Fatal Logs
38

TABLE V: Number of Fatal Logs

Task 9: What are the top 5 most frequently occurring dates in the log?

Date	Date Count
2005-07-09	381827
2005-06-14	381561
2005-12-01	271341
2005-11-03	200937
2005-07-23	200654

TABLE VI: Date and Date Count

V. Conclusions

This project report outlines the systematic approach taken for analyzing the extensive BGL log dataset using the Apache Spark framework within a Jupyter Notebook environment. The primary objective of the analysis is to gain insights from the log data generated by the BlueGene/L supercomputer at Lawrence Livermore National Labs. The methodology includes data acquisition, preparation, and computation tasks using PySpark and SparkSQL using Specific design patterns, such as aggregation and filtering, are applied to answer the posed questions. The methodology showcased in this project report highlights the advantages of using Apache Spark, especially its in-memory data processing capability, integration with various components, and support for large-scale data processing. PySpark and SparkSQL are used for

analysis tasks. The process includes data transformation, column casting, parsing timestamps, and data aggregation using Spark framework. This project report presents several key tasks with corresponding algorithms, such as identifying fatal log entries in a specific month, calculating average seconds for specific events, determining the most frequently occurring hours, finding nodes with the smallest number of events, and identifying the latest fatal kernel error date. In the results section, the outcomes of various analysis tasks are presented. These include the number of fatal log entries in September resulting from a major internal error, the average number of seconds for re-synch state events on each day of the week, the top hours of log occurrences, nodes with the smallest number of APPUNAV events, and the latest date of a fatal kernel error resulting in an rts panic with some extra insights gained while analyzing log data set. This paper provides a comprehensive overview of the approach used to analyze the BGL log dataset, highlighting the effective use of Apache Spark and design patterns for scalable and efficient data computation tasks. The results section demonstrates the practical application of the methodology to answer specific analytical questions related to the log data.

REFERENCES

[1] S. 2023, "Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025." [Online]. Available: https://www.statista.com/statistics/871513/worldwide-data-created

[2] D. Talia, "A view of programming scalable data analysis: from clouds to exascale," *Journal of Cloud Computing*, vol. 8, no. 1, p. 4, 2019.

[3] L. Belcastro, R. Cantini, F. Marozzo, A. Orsino, D. Talia, and P. Trunfio, "Programming big data analysis: principles and solutions," *Journal of Big Data*, vol. 9, no. 1, pp. 1–50, 2022.

[4] W. Yoo, M. Koo, Y. Cao, A. Sim, P. Nugent, and K. Wu, *Performance analysis tool for HPC and big data applications on scientific clusters*. Springer, 2016.

[5] N. R. Adiga, G. Almási, G. S. Almasi, Y. Aridor, R. Barik, D. Beece, R. Bellofatto, G. Bhanot, R. Bickford, M. Blumrich *et al.*, "An overview of the bluegene/l supercomputer," in *SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. IEEE, 2002, pp. 60–60.

[6] J.-W. Park, X. Huang, and C.-H. Lee, "Analyzing and predicting job failures from hpc system log," *The Journal of Supercomputing*, pp. 1–28, 2023.

[7] B. H. Park, S. Hukerikar, R. Adamson, and C. Engelmann, "Big data meets hpc log analytics: Scalable approach to understanding systems at extreme scale," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 758–765.

[8] N. Ahmed, A. L. Barczak, T. Susnjak, and M. A. Rashid, "A comprehensive performance analysis of apache hadoop and apache spark for large scale data sets using hibench," *Journal of Big Data*, vol. 7, no. 1, pp. 1–18, 2020.

[9] A. B. Patel, M. Birla, and U. Nair, "Addressing big data problem using hadoop and map reduce," in *2012 Nirma University International Conference on Engineering (NUiCONE)*. IEEE, 2012, pp. 1–5.

[10] T. Tekdogan and A. Cakmak, "Benchmarking apache spark and hadoop mapreduce on big data classification," in *Proceedings of the 2021 5th International Conference on Cloud and Big Data Computing*, 2021, pp. 15–20.

[11] B. H. Park, Y. Hui, S. Boehm, R. A. Ashraf, C. Layton, and C. Engelmann, "A big data analytics framework for hpc log data: Three case studies using the titan supercomputer log," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 2018, pp. 571–579.

[12] M. Dwyer, N. Kaff, J. Cohen, and M. Frauenhoffer, "Leveraging comprehensive data analysis to inform parallel hpc workloads," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 3960–3967.

[13] F. Yin and F. Shi, "A comparative survey of big data computing and hpc: From a parallel programming model to a cluster architecture," *International Journal of Parallel Programming*, vol. 50, no. 1, pp. 27–64, 2022.

[14] C. by LogPAI, "Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics," Sep. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.3227177

[15] Sparkdoc, "spark.apache.org." [Online]. Available: https://spark.apache.org/docs/latest/cluster-overview.html