Operational Analytics of HDFS Logs with Anomaly Detection

Abhitosh - *abhitosh2024@iisc.ac.in*

Amit Nitin Joshi - *amitj@iisc.ac.in*

Naik Raghavendra Narottam - *nraghavendra@iisc.ac.in*

Vignesh S - *vigneshs@iisc.ac.in*

Problem statement

- Derive long term and real-time Operational Insights using Application Logs
  - System Health
  - System Performance
  - Anomalies

- Challenges
  - Scale and Volume of
  - Velocity in scaled environment

Features

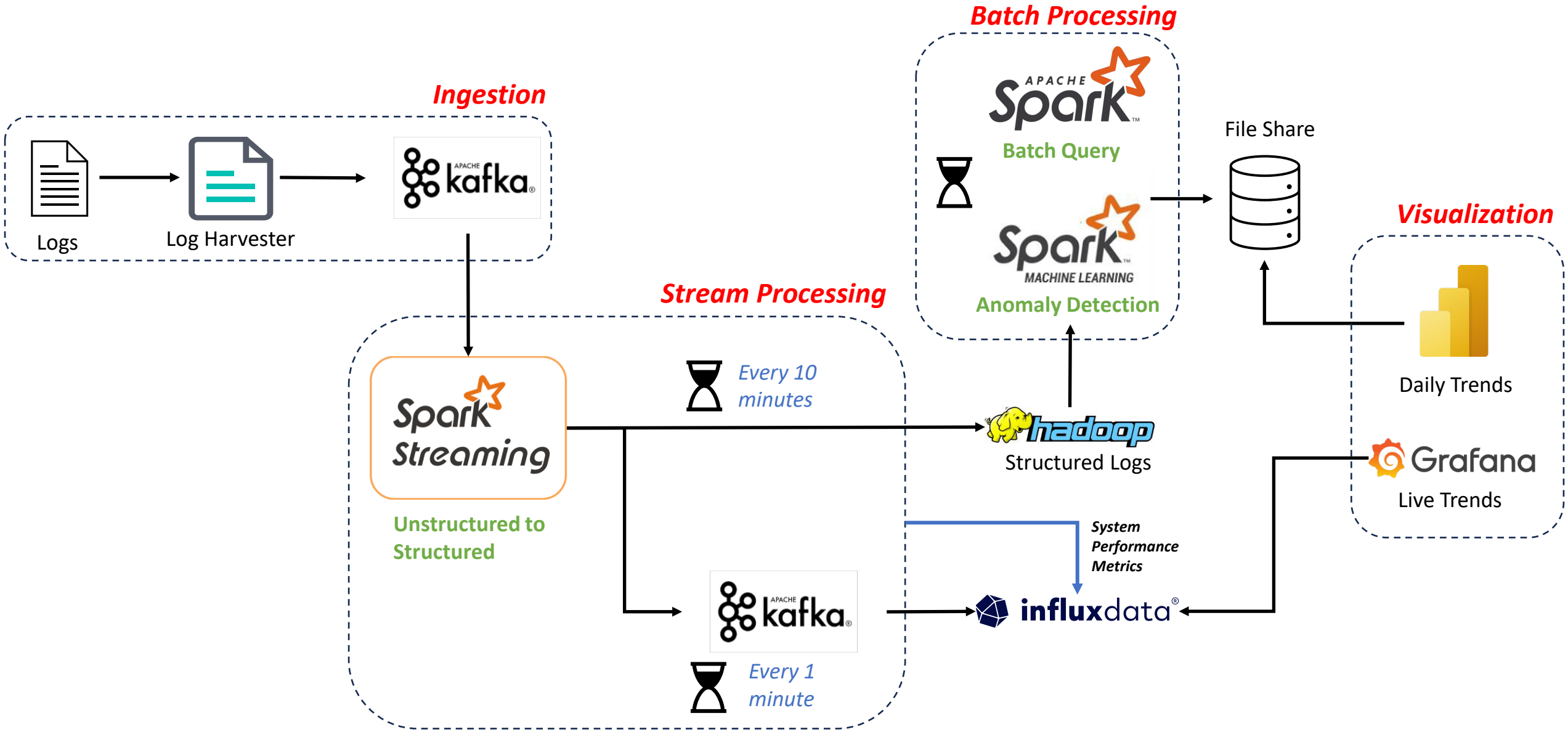
- Operations dashboard
  - Real time insights
  - Daily insights
  - Blocks with anomalies
- Non Functional
  - Optimal throughput and latency

Design Goals

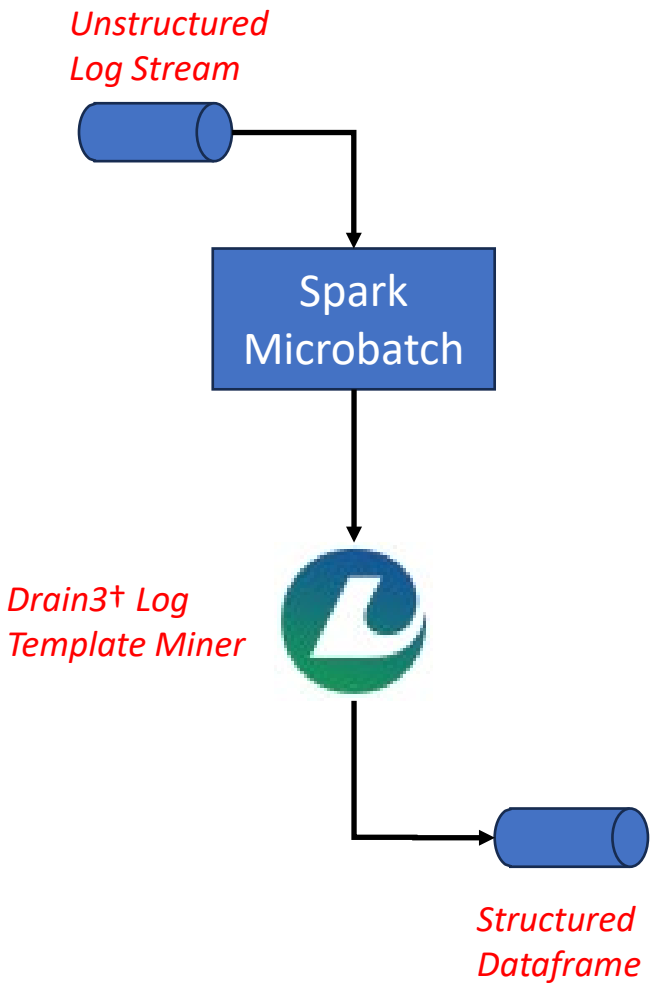
- **Distributed Data Parallel Processing**
  - Efficiently handle large-scale log data by splitting tasks across multiple nodes
  - Enable real-time monitoring and faster issue detection during high data volumes
  - Enhance fault tolerance with redundancy and data replication to prevent data loss

- **Visualization of System KPIs**
  - Present key metrics (failure rates) via interactive dashboards
  - Track system performance and detect errors, malfunctions, or bottlenecks
  - Visualize real-time data using charts
  - Enable quicker anomaly detection and informed decision-making
  - Improve operational monitoring, system stability, and performance optimization

Architecture



Unstructured to Structured

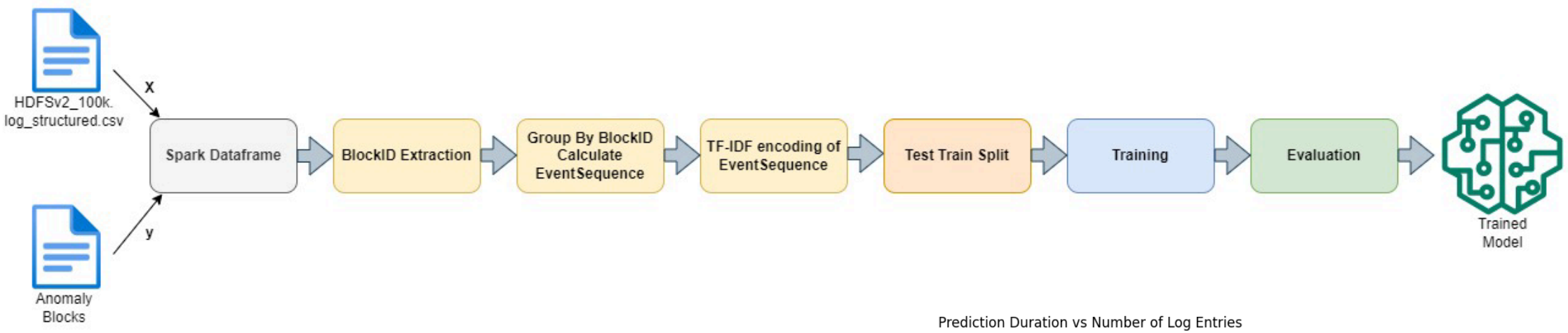



```
2015-08-21 11:16:08,958 INFO org.apache.hadoop.hdfs.server.datanode.DirectoryScanner: Periodic Directory Tree Verification scan starting at 1440132307958 with interval 21
2015-08-21 11:16:08,959 INFO org.apache.hadoop.hdfs.server.datanode.VolumeScanner: VolumeScanner(/tmp/hadoop-hdfs/dfs/data, DS-7d357a5f-01e9-4fb1-8d89-47b0c77e215e): fini
2015-08-21 11:16:08,960 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Block pool BP-108841162-10.10.34.11-1440074360971 (Datanode Uuid null) service to mesos-mast
2015-08-21 11:16:08,972 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Block pool Block pool BP-108841162-10.10.34.11-1440074360971 (Datanode Uuid null) service to
2015-08-21 11:16:08,972 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: For namenode mesos-master-1/10.10.34.11:9000 using DELETEREPORT_INTERVAL of 300000 msec  BLO
2015-08-21 11:16:09,036 INFO org.apache.hadoop.hdfs.server.datanode.VolumeScanner: VolumeScanner(/tmp/hadoop-hdfs/dfs/data, DS-7d357a5f-01e9-4fb1-8d89-47b0c77e215e): no s
2015-08-21 11:16:09,056 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Namenode Block pool BP-108841162-10.10.34.11-1440074360971 (Datanode Uuid e659dd9a-5d49-41bb
2015-08-21 11:16:09,056 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Acknowledging ACTIVE Namenode Block pool BP-108841162-10.10.34.11-1440074360971 (Datanode Uu
2015-08-21 11:16:09,107 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Successfully sent block report 0xdbc5da04566f,  containing 1 storage report(s), of which we
2015-08-21 11:16:09,108 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Got finalize command for block pool BP-108841162-10.10.34.11-1440074360971
2015-08-21 11:16:44,610 WARN org.apache.hadoop.hdfs.server.datanode.DataNode: IOException in offerService
```

LineId	Date	Time	ms	Level	Component	Content	EventId	EventTemplate	ParameterList
1	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773204_2032380 10.10.34.33:50010 10.10.34.11:50010 10.10.34.16:50010	14d49f3f	BLOCK* addToInvalidates: <*> <*> <*> 10.10.34.16:50010	['blk_1075773204_2032380', '10.10.34.33:50010 10.10.34.11:50010']
2	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773230_2032406 10.10.34.23:50010 10.10.34.29:50010 10.10.34.30:50010	8d0306e3	BLOCK* addToInvalidates: <*> <*> 10.10.34.29:50010 <*>	['blk_1075773230_2032406 10.10.34.23:50010', '10.10.34.30:50010']
3	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773211_2032387 10.10.34.35:50010 10.10.34.26:50010 10.10.34.32:50010	4e47f9f1	BLOCK* addToInvalidates: <*> 10.10.34.35:50010 <*> <*>	['blk_1075773211_2032387', '10.10.34.26:50010 10.10.34.32:50010']
4	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773221_2032397 10.10.34.31:50010 10.10.34.12:50010 10.10.34.16:50010	14d49f3f	BLOCK* addToInvalidates: <*> <*> <*> 10.10.34.16:50010	['blk_1075773221_2032397', '10.10.34.31:50010 10.10.34.12:50010']
5	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773198_2032374 10.10.34.37:50010 10.10.34.23:50010 10.10.34.34:50010	67542852	BLOCK* addToInvalidates: <*> <*> 10.10.34.23:50010 <*>	['blk_1075773198_2032374 10.10.34.37:50010', '10.10.34.34:50010']
6	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773201_2032377 10.10.34.38:50010 10.10.34.13:50010 10.10.34.18:50010	09227ed6	BLOCK* addToInvalidates: <*> <*> 10.10.34.13:50010 <*>	['blk_1075773201_2032377 10.10.34.38:50010', '10.10.34.18:50010']
7	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773231_2032407 10.10.34.22:50010 10.10.34.19:50010 10.10.34.13:50010	951e4756	BLOCK* addToInvalidates: <*> 10.10.34.22:50010 <*> <*>	['blk_1075773231_2032407', '10.10.34.19:50010 10.10.34.13:50010']
8	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773212_2032388 10.10.34.36:50010 10.10.34.30:50010 10.10.34.20:50010	2f85e538	BLOCK* addToInvalidates: <*> 10.10.34.36:50010 <*> <*>	['blk_1075773212_2032388', '10.10.34.30:50010 10.10.34.20:50010']
9	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773217_2032393 10.10.34.19:50010 10.10.34.17:50010 10.10.34.40:50010	97ddbce0	BLOCK* addToInvalidates: <*> 10.10.34.19:50010 <*> <*>	['blk_1075773217_2032393', '10.10.34.17:50010 10.10.34.40:50010']
10	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773200_2032376 10.10.34.35:50010 10.10.34.37:50010 10.10.34.15:50010	4e47f9f1	BLOCK* addToInvalidates: <*> 10.10.34.35:50010 <*> <*>	['blk_1075773200_2032376', '10.10.34.37:50010 10.10.34.15:50010']
11	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773220_2032396 10.10.34.32:50010 10.10.34.22:50010 10.10.34.39:50010	e619d482	BLOCK* addToInvalidates: <*> <*> 10.10.34.22:50010 <*>	['blk_1075773220_2032396 10.10.34.32:50010', '10.10.34.39:50010']
12	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773228_2032404 10.10.34.22:50010 10.10.34.21:50010 10.10.34.11:50010	951e4756	BLOCK* addToInvalidates: <*> 10.10.34.22:50010 <*> <*>	['blk_1075773228_2032404', '10.10.34.21:50010 10.10.34.11:50010']
13	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773223_2032399 10.10.34.24:50010 10.10.34.36:50010 10.10.34.27:50010	dfb8e476	BLOCK* addToInvalidates: <*> <*> <*> 10.10.34.27:50010	['blk_1075773223_2032399', '10.10.34.24:50010 10.10.34.36:50010']
14	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773224_2032400 10.10.34.36:50010 10.10.34.17:50010 10.10.34.37:50010	2f85e538	BLOCK* addToInvalidates: <*> 10.10.34.36:50010 <*> <*>	['blk_1075773224_2032400', '10.10.34.17:50010 10.10.34.37:50010']
15	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773209_2032385 10.10.34.31:50010 10.10.34.29:50010 10.10.34.30:50010	8d0306e3	BLOCK* addToInvalidates: <*> <*> 10.10.34.29:50010 <*>	['blk_1075773209_2032385 10.10.34.31:50010', '10.10.34.30:50010']
16	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773235_2032411 10.10.34.15:50010 10.10.34.39:50010 10.10.34.37:50010	22a69001	BLOCK* addToInvalidates: <*> <*> 10.10.34.39:50010 <*>	['blk_1075773235_2032411 10.10.34.15:50010', '10.10.34.37:50010']
17	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773232_2032408 10.10.34.21:50010 10.10.34.23:50010 10.10.34.34:50010	67542852	BLOCK* addToInvalidates: <*> <*> 10.10.34.23:50010 <*>	['blk_1075773232_2032408 10.10.34.21:50010', '10.10.34.34:50010']
18	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773214_2032390 10.10.34.30:50010 10.10.34.21:50010 10.10.34.11:50010	0f802519	BLOCK* addToInvalidates: <*> 10.10.34.30:50010 <*> <*>	['blk_1075773214_2032390', '10.10.34.21:50010 10.10.34.11:50010']
19	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773226_2032402 10.10.34.19:50010 10.10.34.30:50010 10.10.34.23:50010	97ddbce0	BLOCK* addToInvalidates: <*> 10.10.34.19:50010 <*> <*>	['blk_1075773226_2032402', '10.10.34.30:50010 10.10.34.23:50010']
20	10/23/2016	5:03:35	680	INFO	BlockStateChange	BLOCK* addToInvalidates: blk_1075773216_2032392 10.10.34.33:50010 10.10.34.15:50010 10.10.34.35:50010	ce4decb0	BLOCK* addToInvalidates: <*> <*> <*> 10.10.34.35:50010	['blk_1075773216_2032392', '10.10.34.33:50010 10.10.34.15:50010']

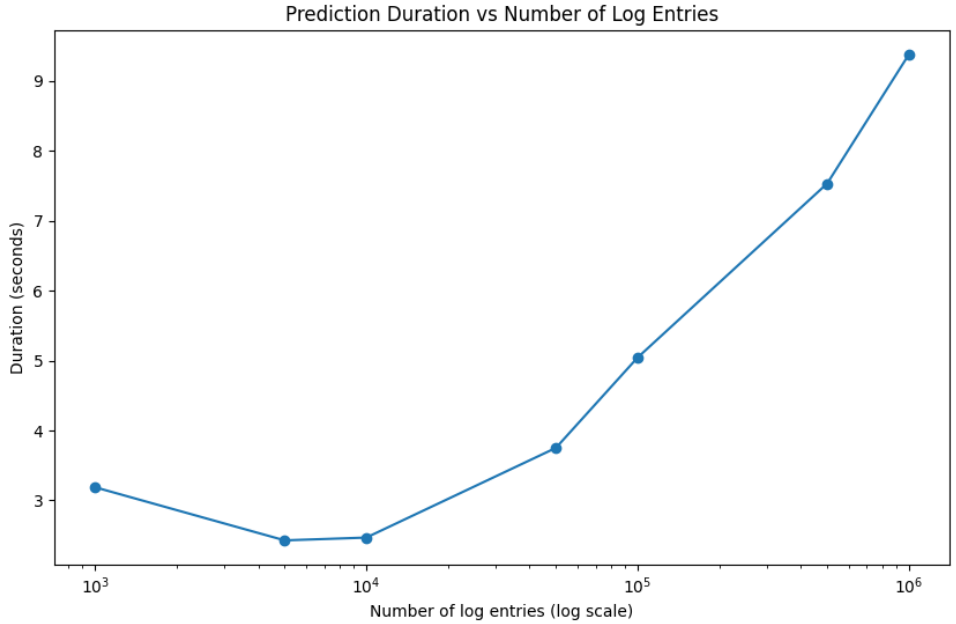
† https://github.com/logpai/Drain3

Anomaly Detection



Name	Value
Training duration	106.27 s
Accuracy	0.7723
Precision	0.9714
Recall	0.5151
F1 Score	0.6733
ROC-AUC	0.7723

Model Performance Metrics



Inference Performance Metrics

Development Setup

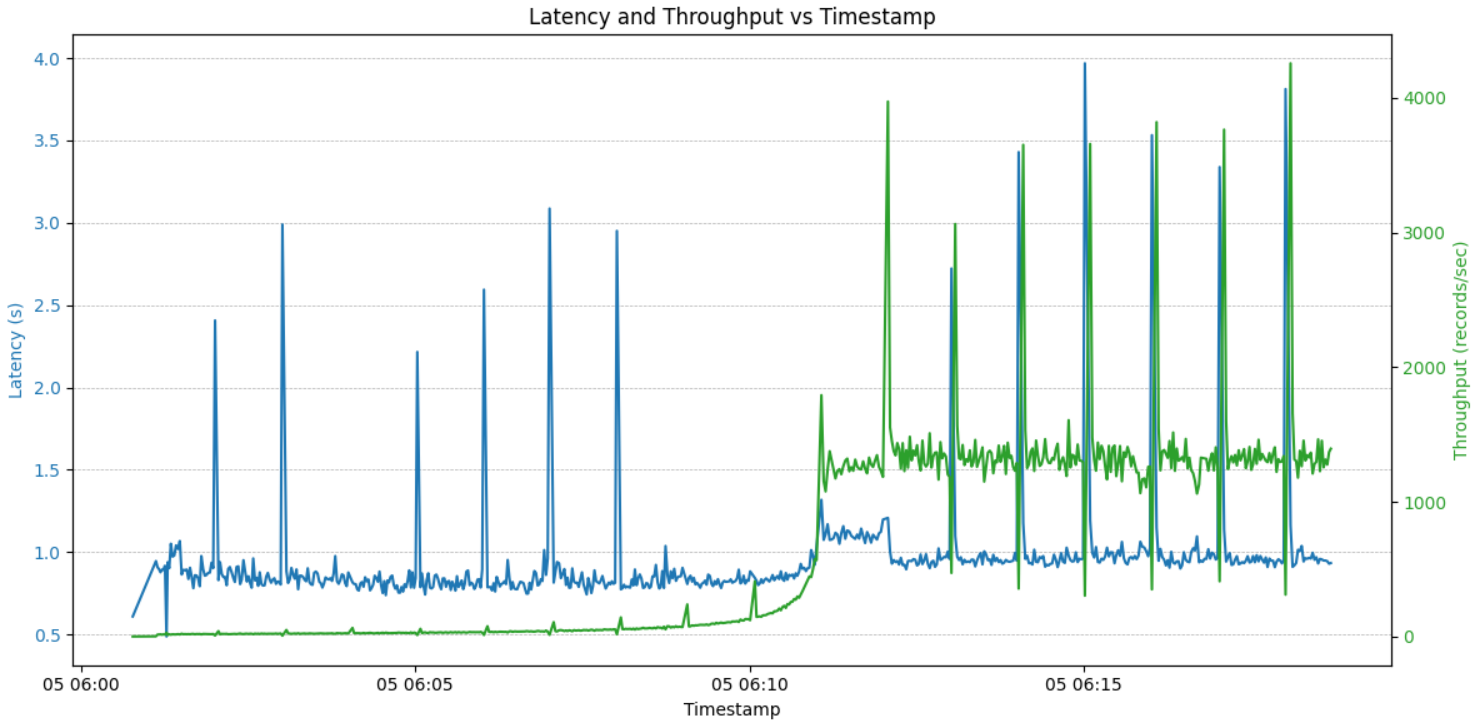
Evaluation: Throughput and latency

Experiment:

- Gradually increase input rate
- Measure latency of log parsing
- Measure throughput of the system

Observations:

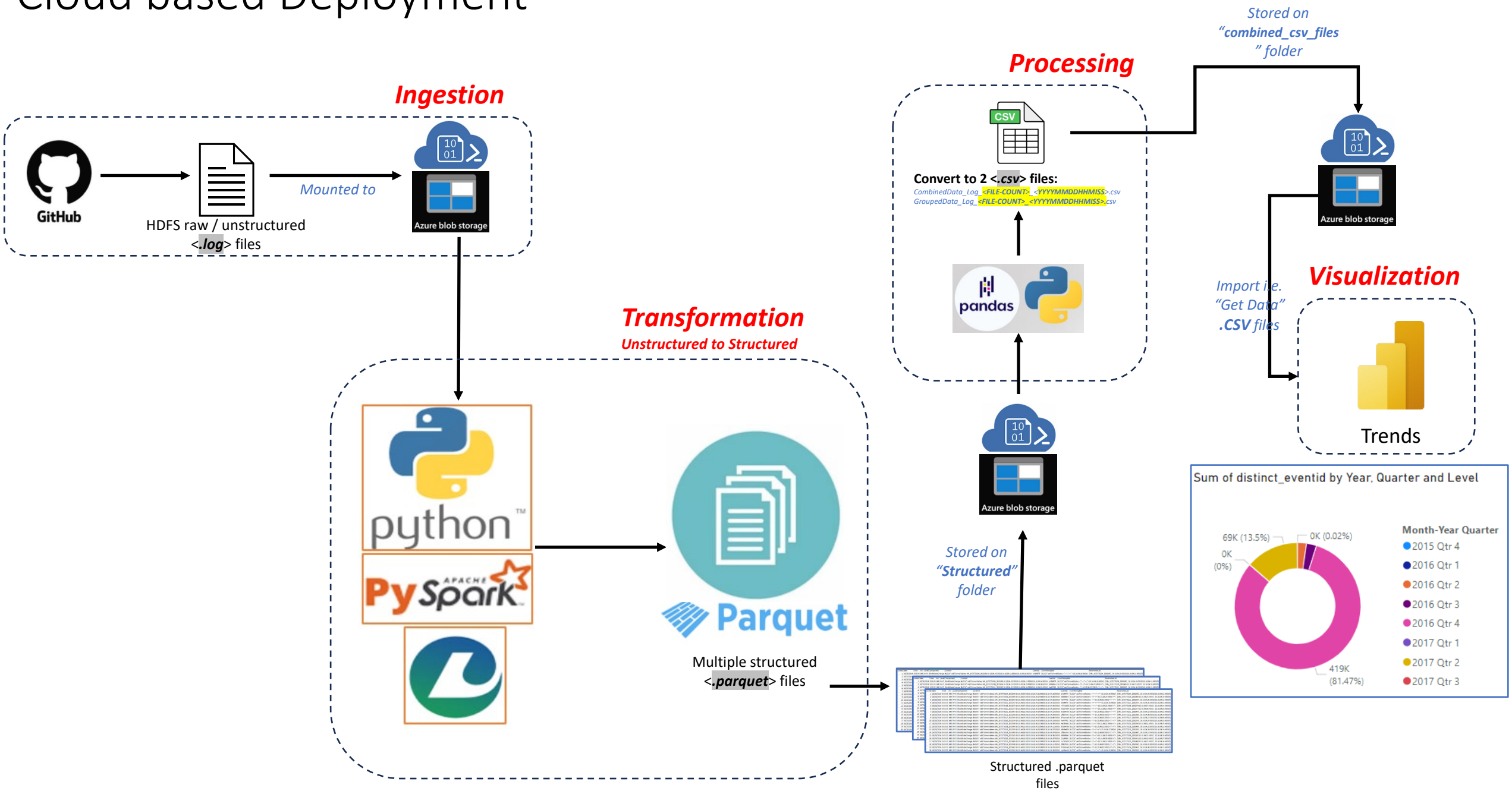
- No significant impact on latency
- Throughput saturates after a certain point



References and Further Reading

- 1. Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, Michael R. Lyu. “Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics.” (“GitHub - logpai/loghub: A large collection of system log datasets for ...”) (“GitHub - logpai/loghub: A large collection of system log datasets for ...”) IEEE International Symposium on Software Reliability Engineering (ISSRE), 2023.
- 2. Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. (“LogAnMeta: Log Anomaly Detection Using Meta Learning - ResearchGate”) “Drain: An Online Log Parsing Approach with Fixed Depth Tree, Proceedings of the 24th International Conference on Web Services (ICWS), 2017.” (“logparser/logparser/Drain/README.md at main - GitHub”)
- 3. Shilin He, Jieming Zhu, Pinjia He, Michael R. Lyu. Experience Report: System Log Analysis for Anomaly Detection, *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2016. (“GitHub - logpai/loghub: A large collection of system log datasets for ...”) 中文版本 *ISSRE Most Influential Paper*
- 4. Use open source Drain3 log-template mining project to monitor for network outages
- 5. Fingerprinting the Datacenter: Automated Classification of Performance Crises, by Peter Bodík, Moises Goldszmidt, Armando Fox, Hans Andersen. *Microsoft*
- 6. loghub
- 7. Drain3
- 8. loglizer

Cloud based Deployment



MS-Azure (DFS) Architecture

Architecture Components

- **Log Storage:**
  - Use Azure Blob Storage to store raw HDFS logs (.log files).
  - Logs are ingested periodically from on-premises or other cloud locations.
- **Log Parsing:**
  - Drain Parser (integrated in Azure Databricks) is used to parse unstructured logs into structured log templates and extract metadata.
  - Parsed output is saved as .parquet files in Blob Storage.
- **Data Processing:** *Azure Databricks (Spark/PySpark)*
  - Aggregates and transforms the .parquet files.
  - Generates summarized .csv outputs for visualization.
  - Performs batch-based or stream-based log processing.
- **Analytics and Visualization:**
  - Azure Blob Storage serves as the data source for Power BI.Power BI visualizes aggregated data (e.g., event trends, latency, error distribution).
- **Workflow Automation:**
  - Azure Data Factory or Azure Logic Apps is used for orchestrating data ingestion, processing, and export pipelines.

Reasons for Failure

- **Resource Constraints:**
  - Free-tier clusters have limited memory and compute resources. Parsing an 800 MB file and generating 2000 small .parquet files can quickly exhaust the available memory or disk I/O capacity.
- **Excessive File Partitioning:**
  - This can overwhelm the cluster's metadata handling and slow down processing due to excessive partitioning.
- **Drain Parser Complexity:**
  - Drain parsers often involve recursive algorithms, which can be computationally expensive. Processing large files can cause memory overflows in constrained environments.
- **Driver Overload:**
  - If the driver node processes excessive data or metadata, it may crash due to insufficient memory or compute power.

Explain: Drain Parser

The Drain Parser is a tree-based log parsing algorithm designed to parse unstructured log messages into structured formats efficiently. It works by identifying patterns in log messages and extracting structured templates and parameters.

Key Features:

- Tree-Based Approach: Drain builds a hierarchical tree with nodes that represent log message patterns. Each path in the tree corresponds to a structured log template.
- Parameters Extraction: It separates dynamic parameters (e.g., timestamps, IP addresses, etc.) from static text in log messages.
- Similarity Threshold (st): Controls how similar a new log message must be to an existing template to fit into the same structure.
- Depth (depth): Limits the depth of the tree for log grouping.

Workflow of Drain Parser:

- Input: Raw unstructured log messages.
- Log Template Extraction: Static parts of the logs are grouped into templates, while dynamic parts are parameterized.

Code Explanation

The code is a pipeline that processes unstructured log files, parses them into a structured format using the Drain log parser, and then saves the results as a structured .parquet file for further analysis.

The provided code:

- *Reads Unstructured Logs:*
  - *Reads log files stored in the DBFS mount (…:/mnt/...).*
  - *Uses Spark to parallelize* log processing.

- Parses Logs Using Drain Parser:
  - Each partition of the RDD is processed by the process_file_train function.
  - Drain Parser takes the log format, regular expressions (regex), and similarity threshold to parse the logs.
- Processes Multiple Files:
  - Logs from multiple files are processed and combined into a unified DataFrame (*aggregated_df*).
- Writes Output to Parquet:
  - The final structured DataFrame is written as a *.parquet* file to …/mnt/hdfsv2logs/structured.
- The structured format includes:

Column Name	Data-Type	Description
Date	String	The date extracted from the log message.
Time	String	The time extracted from the log message.
ms	Integer	Milliseconds extracted from the log message.
Level	String	Log severity level (e.g., INFO, ERROR, WARN).
Component	String	Log component/module generating the message.
Content	String	The dynamic content of the log message.
EventTemplate	String	Template representing the static structure of the log message.
EventId	Integer	A unique identifier assigned to each log template by Drain Parser.