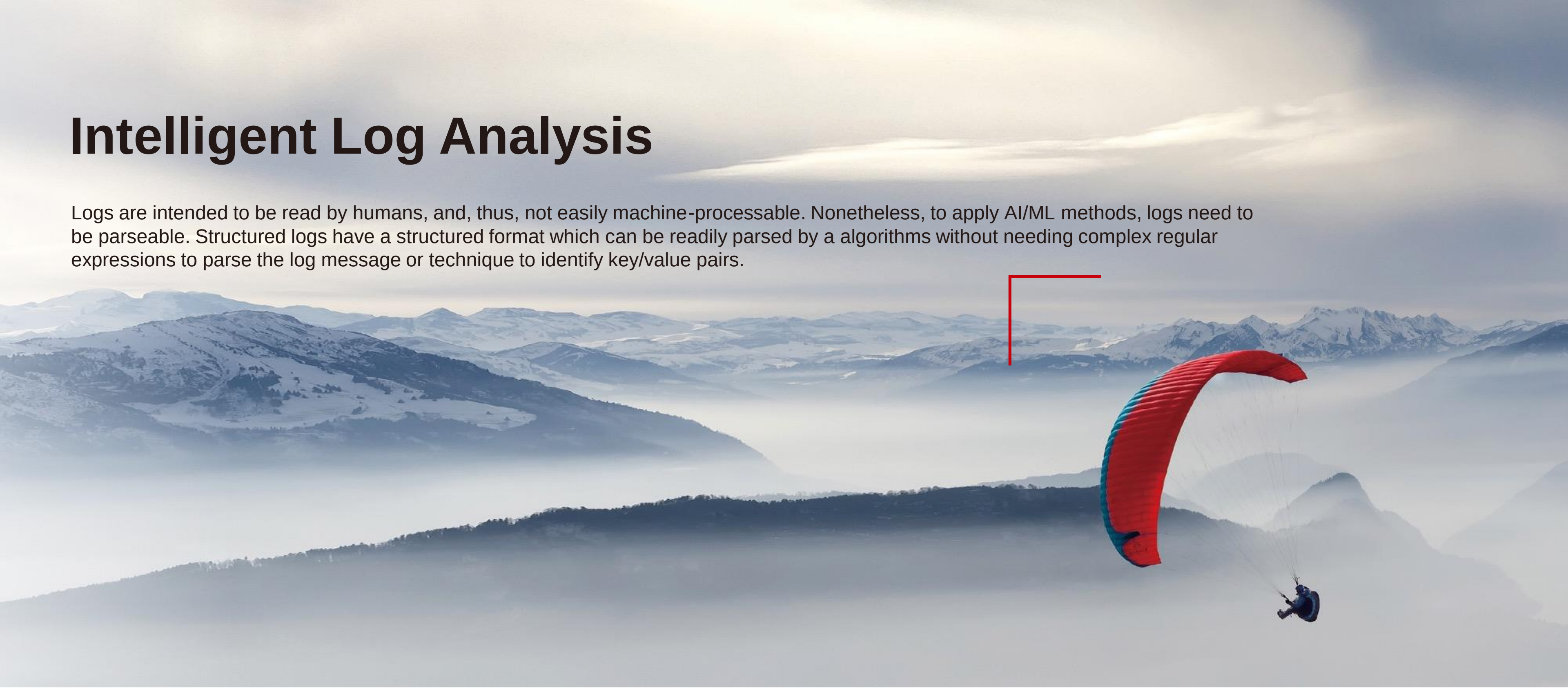
Intelligent Log Analysis

Logs are intended to be read by humans, and, thus, not easily machine-processable. Nonetheless, to apply AI/ML methods, logs need to be parseable. Structured logs have a structured format which can be readily parsed by a algorithms without needing complex regular expressions to parse the log message or technique to identify key/value pairs.



Lecture at Technical University of Berlin

Jorge Cardoso
Chief Engineer for Hyperscale AIOps
Munich Research Center

2020.12.23

Technische Universität Berlin

HUAWEI

Intelligent Structured Log Analysis
Description

Background. Application logs were originally introduced to assist developers to debug software systems. Logs enabled to troubleshooting systems by offering a view into the states that are reached and traversed while systems are in operation.

Problem. In the past, humans have been part of the process of interpreting logs. The contents of logs were specified (often using printf), read and understood by humans. Technological advances and large-scale system complexity have dramatically increased the volume of logs generated. This volume compels to adopt AI/ML methods for processing. Unfortunately, existing log formats were designed to be human understandable and not AI/ML parseable. For example, while a human can easily understand the meaning of the log message: “Apr 25 14:01:12 user Throughput exceed 20Gbps and 7Mpps in 35% of last 15 minutes, above the time threshold 10%!”, its parsing and understanding by AI/ML methods is expensive, complex and possibly inaccurate since it is difficult to distinguish and identify domain knowledge, properties, values, taxonomies and categorical data. In other words, extracting feature vectors from logs is a nontrivial, however critical, procedure that exerts influence on the efficacy of AI/ML algorithms.

Approach. A new approach proposes to move away from unstructured logs into structured logs since their format can be readily parsed by AI/ML algorithms without requiring complex techniques to identify feature vectors. Structured logs are typically written in a format such as JSON, CSV, XML, and RDF that can be easily parsed and processed. For example, log records containing I/O request rate, request length, queue size and other properties can be exploited to build models for storage throughput and latency. As another example, the use of feature vectors to identify system states provides a new way to discover relationships between components by looking at uncommon states which are correlated when failures occur.

Results. Structured log analysis provides a novel event infrastructure which will enable to make additional progress in the fields of usage analysis, performance modelling, anomaly detection, failure diagnosis and security.

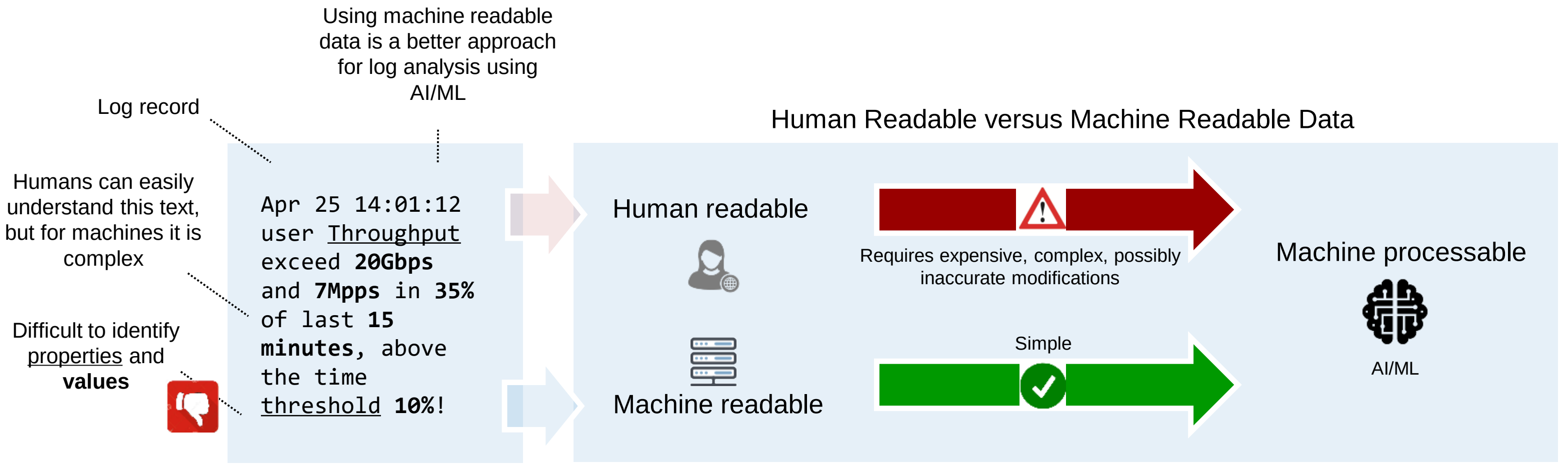
Log Analysis
Problem Definition

Take Away

- Since traditional **log** protocols (e.g., syslog, RFC 5424) where mainly developed to be human readable, their processing by AI/ML techniques is complex

Machine Readable Data: A format that can be easily processed by a computer

- Marked up human-readable
  - HTML, microformats, RDFa, etc.
- Intended for machine processing
  - CSV, XML, JSON, RDF, etc.



[1] tools.ietf.org/html/rfc5424

Log Analysis
Applications

Usage analysis [1]
- User behaviour analysis (e.g., Twitter [1]), log-based metrics counting (e.g., Google Cloud [32]), and workload modeling (e.g., Microsoft [33])
- Requires log parsing

Performance modeling
- Facebook [3] uses logs for performance improvements
- Requires log parsing

Anomaly detection
- PCA [18], invariant mining [34], and deep learning [10]
- Requires log parsing

Duplicate issue identification
- System issues (e.g., disk/network errors) often recur or can be reported by different users, leading to many duplicate issues
- Microsoft has reported some studies [11], [35], [36] on this task
- Requires log parsing

Failure diagnosis
- Recent progress [4], [37] has been made to automate root cause analysis based on machine learning techniques
- Requires log parsing

Challenge
- Transform unstructured logs into structured logs

Existing Approaches
- Handcrafted regular expressions or grok patterns [2] to extract event templates and key parameters: *time-consuming* and *error-prone*
- Automated log parsing using data-driven such as SLCT, LogCluster, IPLoM, LKE Spell, Drain: *approximated* and *computationally expensive*

```
/*  A logging code snippet extracted from:
    hadoop/hdfs/server/datanode/BlockReceiver.java */

LOG.info("Received block " + block + " of size "
    + block.getNumBytes() + " from " + inAddr);
```

Log Message

```
2015-10-18 18:05:29,570 INFO dfs.DataNode$PacketResponder: Received
block blk_-562725280853087685 of size 67108864 from /10.251.91.84
```

Structured Log

TIMESTAMP	2015-10-18 18:05:29,570
LEVEL	INFO
COMPONENT	dfs.DataNode$PacketResponder
EVENT TEMPLATE	Received block <*> of size <*> from /<*>
PARAMETERS	["blk_-562725280853087685", "67108864", "10.251.91.84"]

Fig. 1. An Illustrative Example of Log Parsing

[1] Tools and Benchmarks for Automated Log Parsing, Jieming Zhu, et al.
[2] https://logz.io/blog/logstash-grok

Structured Logging
Overview

Take Away
- Generate machine readable log files to supported advanced analytics for anomaly detection and root-cause analysis

Limitations of Unstructured Logging
- Complex parsing to extract properties and variables
- Requires template mining (e.g., using Drain algorithm)

Structured Logging
- Logs are written in a structured format (e.g., JSON) that can be easily parsed and processed using AI/ML

Technical Benefits
- *Log processing.* key/value pairs, explicit numbers vs strings, and support for nested data structures enable structured logs to be easily processed by AI/ML
- *System monitoring*. Easily generate charts to analyze behavior
- *Data anonymization*. Easily change log records to hidden confidential information
- *Log searching.* Enables to easily search and correlate log messages
  - With structured logging: clientid: 12345
  - With unstructured logging: SELECT text FROM logs WHERE text LIKE ”Customer %"

When a log record contains only a string message, it is called unstructured: complex to process by AI/ML methods

Unstructured Logging

```
log.Debug(“Response time customer ” + 1234 + ‘ ‘ + 55 + ‘ms’)
```

```
DEBUG 2017-01-27 – Response time customer 1234 55ms
```

Parsing is done using, e.g., SLCT, LogCluster, IPLoM, LKE Spell, Drain, etc.

Structured Logging

```
log.Debug(“Response time customer ”, new {clientid=1234}, {rt={d…:55, unit: “ms”}))
```

```
DEBUG 2017-01-27 – Response time customer {"clientid": 1234}
                                          {”rt": {‘duration’:55, ”unit": ‘ms’}
```

```
{ "time": "2010-01-01 12:34:56.0000",
  “msg”: “Response time customer ”,
  "clientid ": ”1234",
  ”rt": {
    “duration”: 55,
    “unit”: “ms”}
}
```

Serialize the entire message and additional metadata as JSON

Structured Logging
Implementation

Take Away
- Structured logging libraries exist for most languages. Google also provides a solution
  - https://cloud.google.com/logging/docs/structured-logging

Python
- Structured logging can be easily implemented in Python or use a specialized library such as structlog [1]

Simple implementation in Python [2]

```
import json
import logging

class StructuredMessage(object):
    def __init__(self, message, **kwargs):
        self.message = message
        self.kwargs = kwargs

    def __str__(self):
        return '{%s: %s}' % (self.message, json.dumps(self.kwargs))

m = StructuredMessage   # optional, to improve readability

logging.basicConfig(level=logging.INFO, format='%(message)s')
logging.info(m('message 1', foo='bar', bar='baz', num=123, fnum=123.456))
```

Which results in following log

```
{"message 1": {"fnum": 123.456, "num": 123, "bar": "baz", "foo": "bar"}}
```

Using package structlog [1]

```
import structlog

log = structlog.get_logger()
try:
    raise ValueError("This is the exception message.")
except ValueError:
    log.exception("This is the log message.")
```

Which results in following log

```
{"event": "This is the log message.",
"exception": "Traceback (most recent call last)",
"File": "/usr/bin/decision_tree.py",
"Line": 27,
"Module": "__main__" ,
"Exception": "ValueError("This is the exception message.")"}
```

Note: approaches such as LogAdvisor [3] can help developers to write log statements

[1] https://github.com/hynek/structlog/
[2] https://docs.python.org/2/howto/logging-cookbook.html#implementing-structured-logging
[3] Learning to Log: Helping Developers Make Informed Logging Decisions -- LogAdvisor -- Jieming Zhu

Log Parsing
Problem Definition

1. Import messages from log storage
- Access database, e.g., from ELK

2. Preprocessing
- Parse content to extract **timestamp**, pid, log level and **variable** part
- **Parse timestamps and fix them**
- **Squash Python stack traces**

3. Template Mining
- Reconstruct log templates from message content
- Collect variables and their values

Complex and Expensive

4. Record's time-series
- Group records by templates and create time-series

5. Classification
- Classify time-series as permanent, periodic or isolated

TABLE II
SUMMARY OF AUTOMATED LOG PARSING TOOLS

Log Parser	Year	Technique	Mode	Efficiency	Coverage	Preprocessing	Open Source	Industrial Use
SLCT	2003	Frequent pattern mining	Offline	High	×	×	✓	×
AEL	2008	Heuristics	Offline	High	✓	✓	×	✓
IPLoM	2012	Iterative partitioning	Offline	High	✓	×	×	×
LKE	2009	Clustering	Offline	Low	✓	✓	×	✓
LFA	2010	Frequent pattern mining	Offline	High	✓	×	×	×
LogSig	2011	Clustering	Offline	Medium	✓	×	×	×
SHISO	2013	Clustering	Online	High	✓	×	×	×
LogCluster	2015	Frequent pattern mining	Offline	High	×	×	✓	✓
LenMa	2016	Clustering	Online	Medium	✓	×	✓	×
LogMine	2016	Clusteri						
Spell	2016	Longest common						
Drain	2017	Parsing t						
MoLFI	2018	Evolutionary al						

For more than 15 years many log parsers have been proposed. Nonetheless, parsing is still expensive (time), complex, and not always yields a high accuracy

```
2019-07-10T15:23:52.264 18550 ERROR oslo.messaging._drivers.impl_rabbit
[-] [fa5b6584-eb05-4a8a-bce2-356a66a218cb] AMQP server on
192.168.5.151:5672 is unreachable: timed out. Trying again in 8
seconds.: timeout: timed out
```

Timestamp: `2019-07-10T15:23:52.264`
Content: `AMQP server on 192.168.5.151:5672 is unreachable: timed out. Trying again in 8 seconds.: timeout: timed out`

`AMQP server on @VAR1 is unreachable: Trying again in @VAR2 seconds.`

`AMQP server on @VAR1 is unreachable: Trying again in @VAR2 seconds.`
`AMQP server on @VAR1 is unreachable: Trying again in @VAR2 seconds.: RecoverableConnectionError: @VAR3`

Summary: `AMQP server is unreachable`
Keywords: `AMQP`
Rank: 55

Log Parsing
Problem Definition

Limitations

- Accuracy for complex raw logs (e.g., HPC) is relatively low: 81%

TABLE II: Parsing Accuracy of Log Parsing Methods (Raw/Preprocessed)

	BGL	HPC	HDFS	Zookeeer	Proxifier
SLCT	**0.61/0.94**	0.81/0.86	0.86/0.93	0.92/0.92	0.89/-
IPLoM	0.99/0.99	0.64/0.64	0.99/1.00	0.94/0.90	0.90/-
LKE	0.67/0.70	0.17/0.17	**0.57/0.96**	0.78/0.82	0.81/-
LogSig	**0.26/0.98**	0.77/0.87	0.91/0.93	0.96/0.99	0.84/-

Generating these events requires sophisticated algorithms

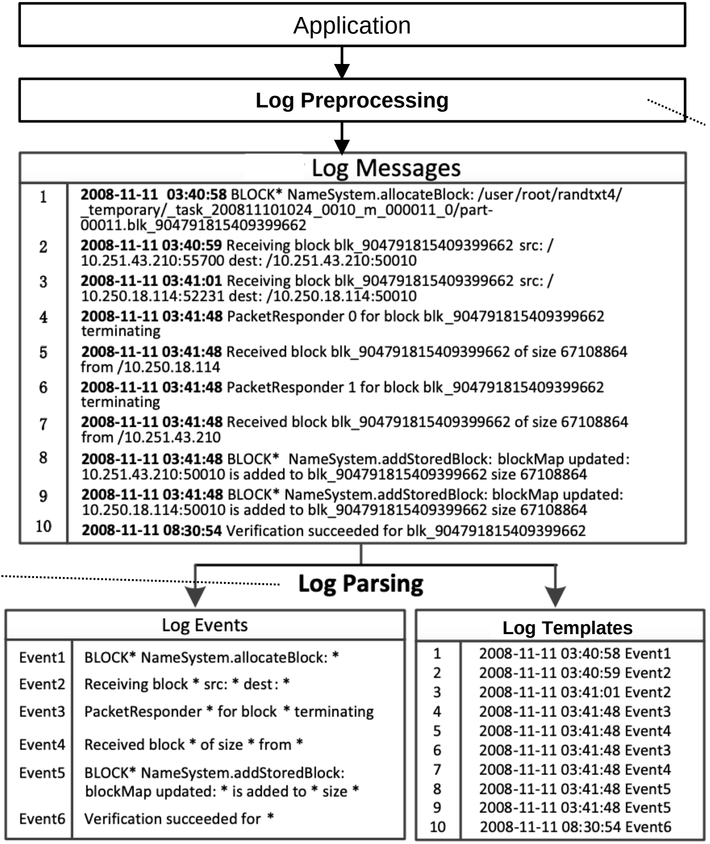
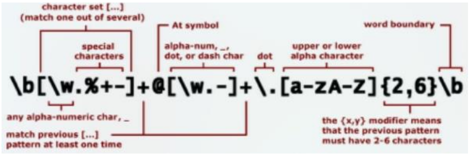


Fig. 1: Overview of Log Parsing

Use regular expressions to identify special fields, e.g., IP, port, IDs



Related Work
Facebook

(Huawei Cloud has Cloud Auto Remediation (CAR) which is rule-base)

Remediation at Facebook
- *FB Auto Remediation* (FBAR) [1, 2] (2011)
- Detect and react to failures
  - migrating services, rebooting, reimaging or off-lining the machine for manual repair

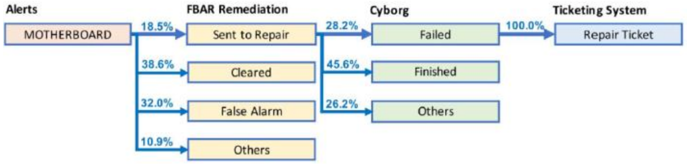


Fig. 2. Percentage of failures ending in each branch from the previous stage.

Benefits of FBAR service
- Developed and maintained by 2 engineers
- Doing the work of 200 system administrators
- Manages more >50% Facebook infrastructure

100x efficiency improvement

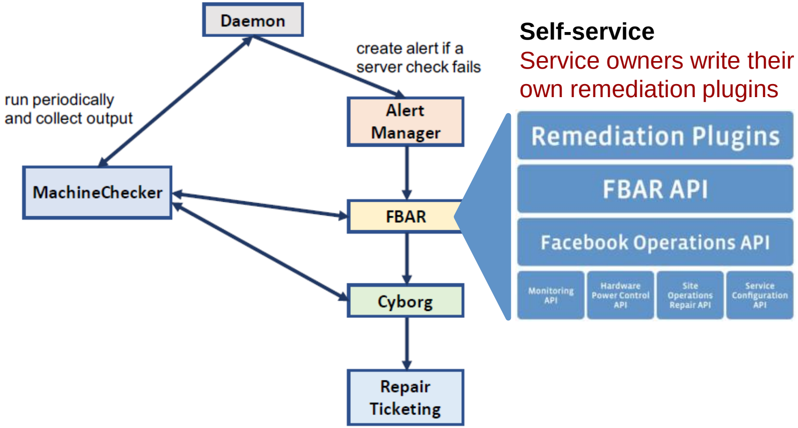


Fig. 1. The hardware failure detection and remediation flow.

Overall Procedure

1. **MachineChecker** [4]: runs checks periodically (ping, ssh, NIC speed, SMART checks)
2. **FBAR**: Once a check fails, an alert is created with failure information
3. **Service owners**: write customized remediation for symptoms and set proper rate limits for remediation to make sure there are always sufficient servers run
4. **Cyborg**: When none of remedia passes case to Cyborg [4] repai
5. **Human involvement**. Cyborg e such as firmware updates. If fix fails: create repair ticket for a technician
6. **Recommendation**. Use ML NLP fastText [3] to learn from repair ticket/logs and recommend repair actions

FBAR uses a simple form of templating to identify states using log files

[1] A. Power, "Making facebook self-healing," https://www.facebook.com/notes/facebook-engineering/making-facebook-selfhealing/10150275248698920/, 2011.
[2] R. Komorn, "Making facebook self-healing: Automating proactive rack maintenance," https://code.facebook.com/posts/629906427171799/making-facebook-self-healing-automating-proactiverack-maintenance/, 2016.
[3] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," arXiv:1607.01759, 2016.

Related Work
Facebook

Anomaly Detection
- Many false alarms
  - Alarms when servers are under heavy load
- Transient failures
  - Run CPU, MEM, and network benchmarks to create loads to reproduce transient failures (CoreMark, iperf3, …)
- Techniques
  - Time series, Holt-Winters, exponential, and Gaussian mixture models
- Thresholds
  - Pre-defined static thresholds
  - Data points outside predictive thresholds
  - Learned from the time series using machine learning

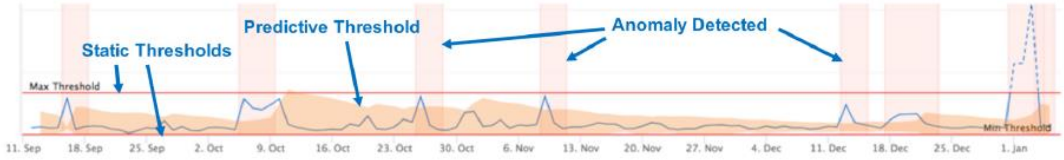


Fig. 3. Server reboot rate for a specific service.

Recommendations
- Use ML to learn from closed repair tickets
- Recommend repair actions
  - Based on how similar tickets have been closed
  - Use raw text logs as features
  - Predict the repair actions
- Evaluation (undiagnosed tickets)
  - Recommend up to 5 repair actions for undiagnosed ticket
  - Correct repair actions: 50% to 80%

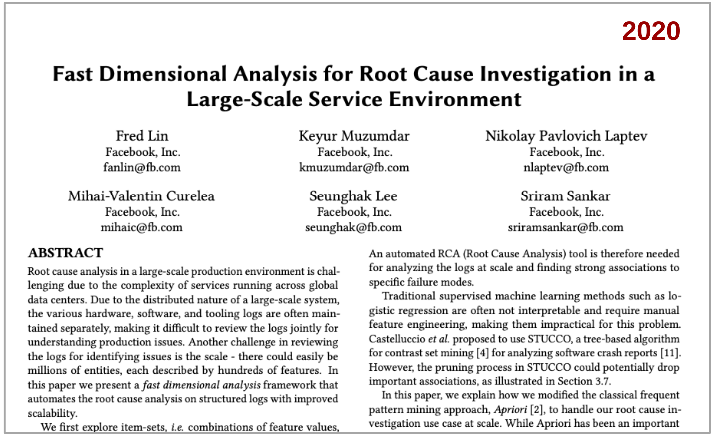
Procedure

- Identify failure states
- Create context for failure
  - Abstract logs, metrics, traces, events, …
- State explosion problem
  - Aggregates states according to patterns
- Map state patterns to recovery scripts
- Establish relationships between patterns’ elements and scripts actions
- Recommendation
  - For a given failure, find the matching script
- Feedback loop from technitiaon

Related Work
Facebook

Real-time Automated RCA (Root Cause Analysis)

- Used in production at Facebook
- Analyzes structured logs to find failure modes associations
  – *Software* service logs and *Hardware* telemetry
- Examples
  – Identify a specific combination of hardware and software configurations correlated to bad reboots
  – Identify characteristics of a software job that are correlated to exceptions
- System architecture
  – Structured logs are pushed to Scuba in-memory db
  – Scuba use dimensional RCA
  – For long-term analytics, logs are stored in HDFS, queried by Hive and Presto



2020

Fast Dimensional Analysis for Root Cause Investigation in a Large-Scale Service Environment

Fred Lin, Facebook, Inc., fanlin@fb.com
Keyur Muzumdar, Facebook, Inc., kmuzumdar@fb.com
Nikolay Pavlovich Laptev, Facebook, Inc., nlaptev@fb.com
Mihai-Valentin Curelea, Facebook, Inc., mihaic@fb.com
Seunghak Lee, Facebook, Inc., seunghak@fb.com
Sriram Sankar, Facebook, Inc., sriramsankar@fb.com

ABSTRACT

Root cause analysis in a large-scale production environment is challenging due to the complexity of services running across global data centers. Due to the distributed nature of a large-scale system, the various hardware, software, and tooling logs are often maintained separately, making it difficult to review the logs jointly for understanding production issues. Another challenge in reviewing the logs for identifying issues is the scale - there could easily be millions of entities, each described by hundreds of features. In this paper we present a *fast dimensional analysis* framework that automates the root cause analysis on structured logs with improved scalability.

We first explore item-sets, *i.e.* combinations of feature values,

An automated RCA (Root Cause Analysis) tool is therefore needed for analyzing the logs at scale and finding strong associations to specific failure modes.

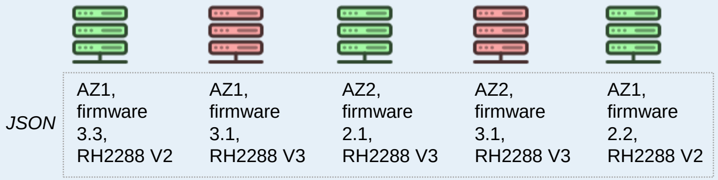
Traditional supervised machine learning methods such as logistic regression are often not interpretable and require manual feature engineering, making them impractical for this problem. Castelluccio *et al.* proposed to use STUCCO, a tree-based algorithm for contrast set mining [4] for analyzing software crash reports [11]. However, the pruning process in STUCCO could potentially drop important associations, as illustrated in Section 3.7.

In this paper, we explain how we modified the classical frequent pattern mining approach, *Apriori* [2], to handle our root cause investigation use case at scale. While Apriori has been an important

Anomalous Hardware and Software Configurations

- Find groups of servers that failed to reboot due to a configuration problem. Root cause analysis
  – Labeled servers that did not rebooted as positive and the rest as negative
  – Create dataset with the labels and with >20 service attributes, e.g., model, services, firmware, kernel
  – Identify attributes correlated with positive/negative labels
  – Root cause: *{firmware version, component model, server model}*

Why some servers have bad reboots?
What is the root cause?



Hardware/software properties

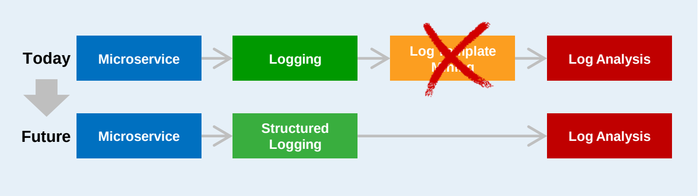
Related Work
Commercial Solutions

Platform	Query, filter, Visual.	Template Mining	Log Comparison	Anomaly Detection	Log Correlation	Incident Detection	RCA	Uniqueness
Coralogix	ELK	Loggregation		Error spike anomalies		Flow anomalies		
SumoLogic	Yes	LogReduce	LogCompare					
DataDog	Yes	LogPatterns (video)			Link logs and traces	Transactions (using ID)		
SolarWinds Loggly	Yes			Anomaly detection				Security
Logz.io	ELK	LogPatterns (clustering, video)						Security
Rollbar	Yes	Grouping/Fingerprinting						
Logentries	Yes	RegEx Named Capture Groups		Anomaly Alert (rule-based)				SQL-based query language (LEQL)
Oracle Log Analytics	Yes	Log Comparison (clustering)		Anomaly Detection (OD)	Time-series correlation			
LogicMonitor	Yes						RCA (topology & metrics)	
Elastic.co (ELK)	Yes			Outlier detection (using counts)				
Sematext	ELK							Security, 75+ integration

Structured Logging
Use Cases and PoCs

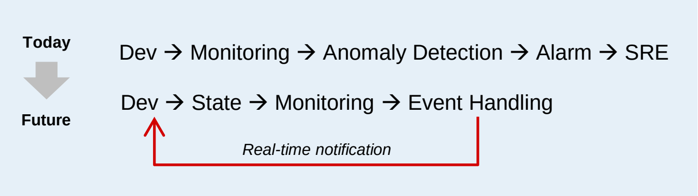
Use Case 1: Cloud Log Analysis PoC

- *Pain point.* Log analysis for CLS requires the use of CPU intensive log parsing techniques (e.g., regex and Drain [1])
- *Technical benefit*. Adoption of structured logging will lower CPU processing
- *Business value*. <u>Performance improvement</u> and <u>lower computational resources</u>



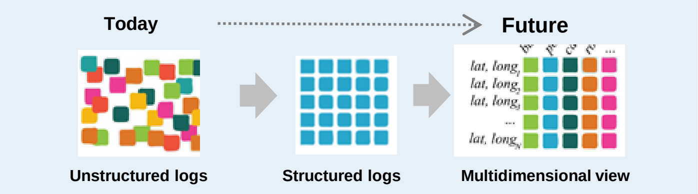
Use Case 2: Real-time Error Tracking PoC

- *Pain point.* DevOps cannot be notified easily when their code reaches a specific faulty state in (pre)production
- *Technical benefit*. Real-time close-loop system from Dev, to code execution, to Ops, and to Dev (Dev → Ops → Dev)
- *Business value*. <u>Real-time error monitoring and debugging of services</u>



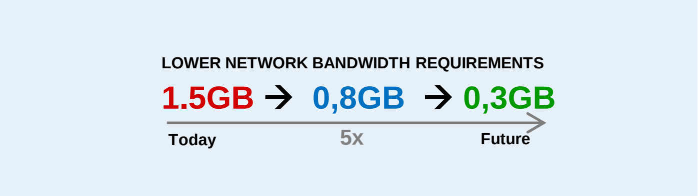
Use Case 3: Multidimensional Root-Cause Mining PoC

- *Pain point.* It is currently difficult to use logs from CLS for root-cause analysis
- *Technical benefit*. Structured logs enable to extract correlation rules from logs to conduct root-cause analysis
- *Business value*. <u>Lower troubleshooting time for service failures by 80%</u>



Use Case 4: Cloud Log Bandwidth Reduction PoC

- *Pain point.* Transfer of logs from services to CLS uses a high network bandwidth
- *Technical benefit*. Structured logs and *protobuf* enable up to 5x bandwidth savings
- *Business value*. <u>Network bandwidth savings</u> 5x



[1] https://github.com/logpai/logparser

Use Case 3
Multidimensional Root-Cause Mining

Goal. Reduce troubleshooting time of access logs

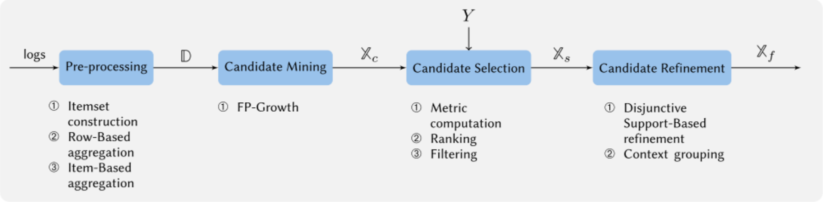
Business value. Save 1h-3h of debugging depending on which information is found manually

Problem. Logs often contains the explanation of failures via the correlation of a failure with its root causes. Unfortunately, it is typically difficult to manually find these correlations

Approach. Association rules can be used to find the rules that correlate how symptoms and root-cause occur together in log records

For example, the rule {error_500} ⇒ {az=north_china, host= 169.206.0.23} indicate the root cause of the status code error_500 is caused by host 169.206.0.23 located in availability zone north_china

Pipeline of the LogRule algorithm. Blue boxes are algorithm phases, numbered items below are the consecutive steps taken in each phase. First, structured logs are pre-processed to construct a transactional database D. FP-Growth is then applied to generate a list of candidate explanations Xc, which are verified and selected via statistical correlation with the input failure pattern Y. Remaining explanations Xs are then refined and organized to provide a final set of explanations Xf



P. Notaro, S. Haeri, J. Cardoso and M. Gerndt, "LogRule: Efficient Structured Log Mining for Root Cause Analysis," in IEEE Transactions on Network and Service Management, 2023

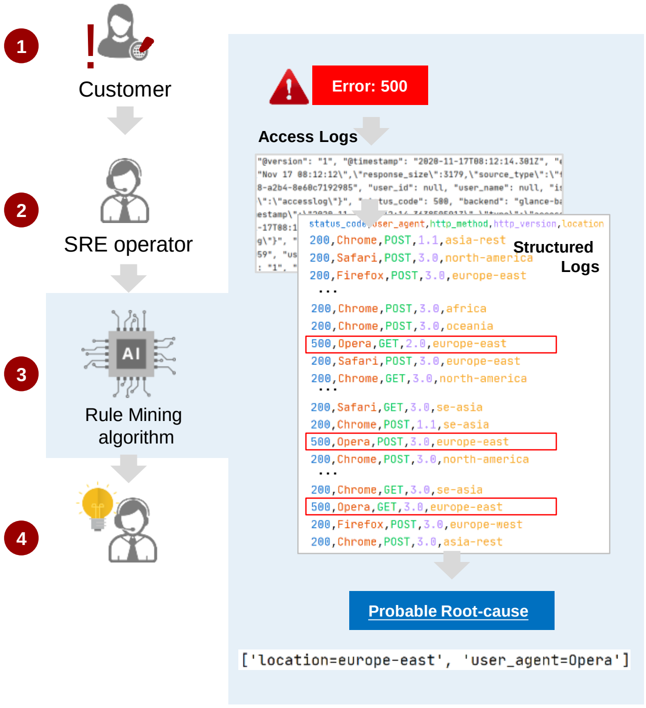


Fig. Results running PoC for Use Case #3

Practical Implementation

Title: Structured Log Analysis in a Web Applications

Description:
- Choose a programming language and a web framework (e.g., Python with Flask, Java with Spring Boot).
- Implement structured logging in the application using a logging library that supports structured data (e.g., Python's structlog, Java's Logback).
- Ensure that logs include contextual information such as timestamps, log levels, user IDs, request IDs, and other relevant metadata.
- Create log entries for different levels (debug, info, warning, error) during various operations within the application (e.g., user login, data retrieval, error handling)
- Based on the paper "LogRule: Efficient Structured Log Mining for Root Cause Analysis” Implement a basic algorithms for root-cause analysis using logs

Deliverables
- Source code of the application with structured log analysis algorithm implemented.
- A report explaining the choices made for logging library, structure of log messages, use cases, and examples of RCA.

References

- Fast dimensional analysis for root cause analysis at scale, Facebook
- Sentry.io, rollbar.com, Raygun, Airbrake, Bugsnag, OverOps
- JSON databases: MongoDB, CouchDB, ElasticSearch, MySQL, PosgreSQL

- Structured logging at Google Cloud
- https://cloud.google.com/logging/docs/structured-logging

- Key foundations of logging
- https://tersesystems.com/blog/2020/03/10/a-taxonomy-of-logging/

- Traditional Logging versus structured logging
- https://softwareengineering.stackexchange.com/questions/312197/benefits-of-structured-logging-vs-basic-logging

- Why Structured logging is important
- https://stackify.com/what-is-structured-logging-and-why-developers-need-it/