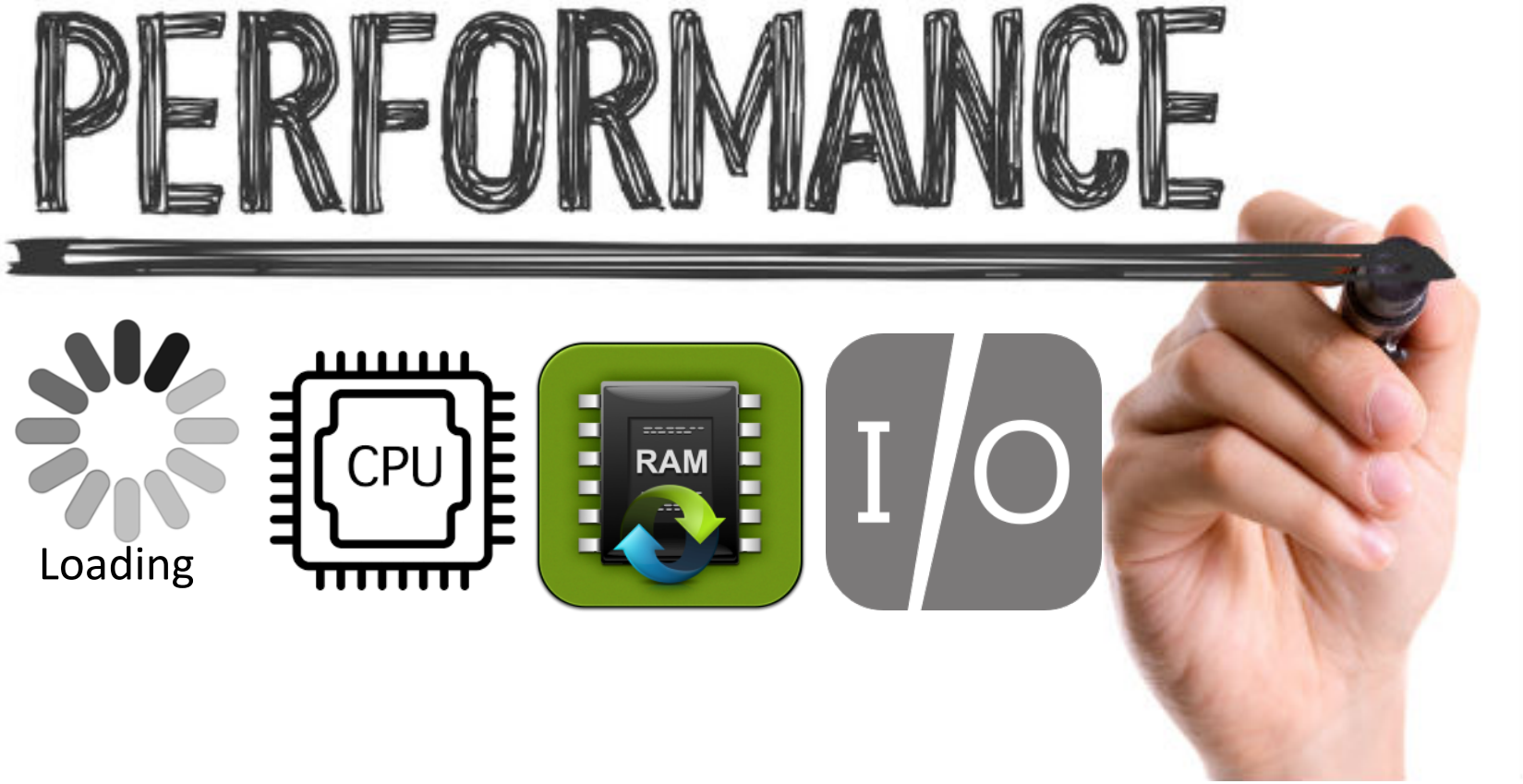
Advancing State-of-the-art Log Analytics Infrastructures

Weiyi Shang

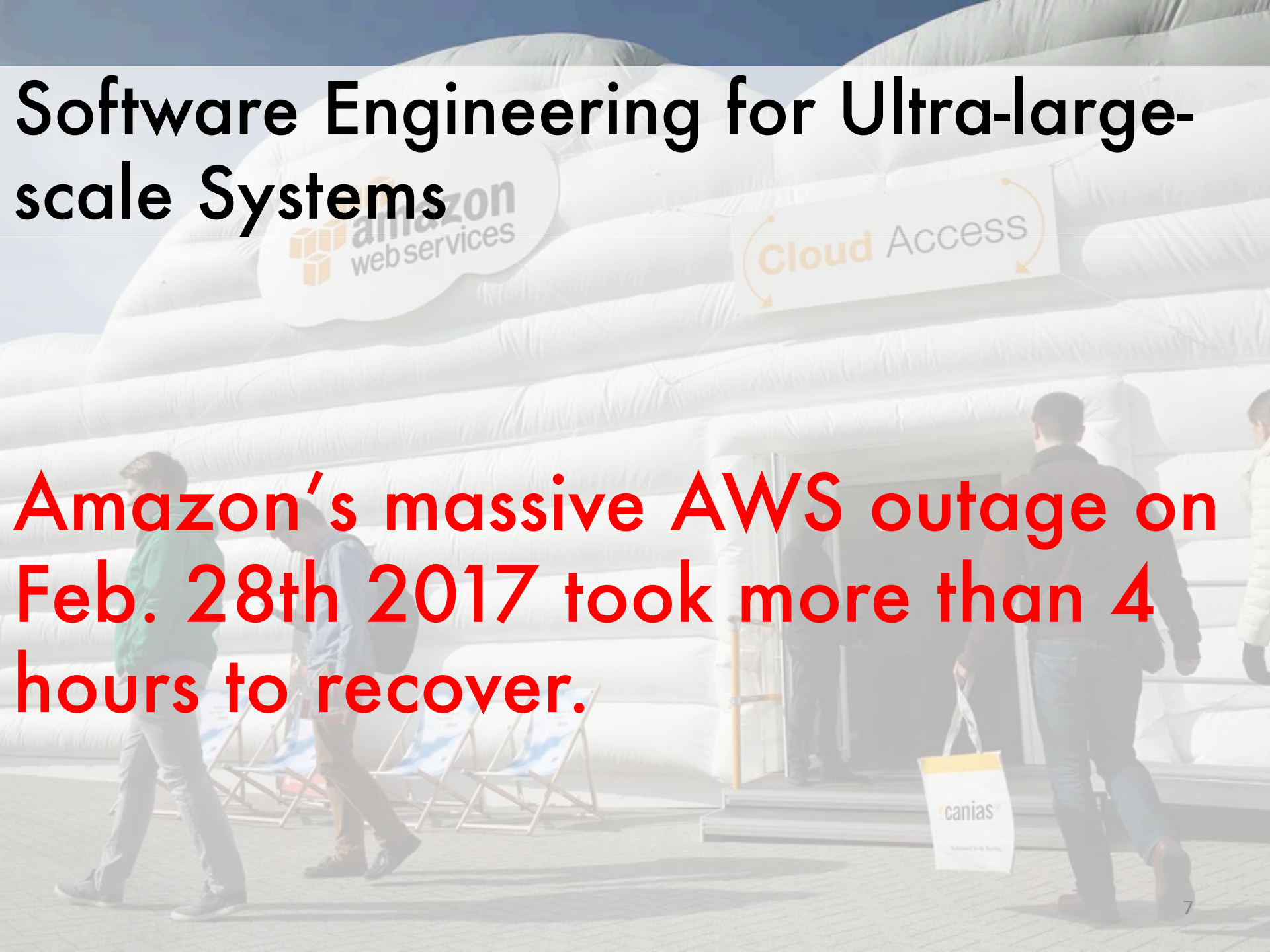
http://users.encs.concordia.ca/~shang/

UNIVERSITÉ Concordia UNIVERSITY

Software system failures are often due to performance issues rather than functional bugs

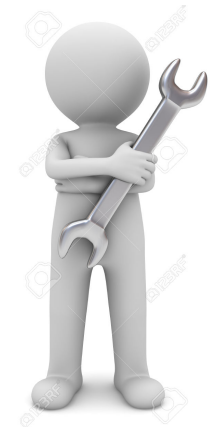


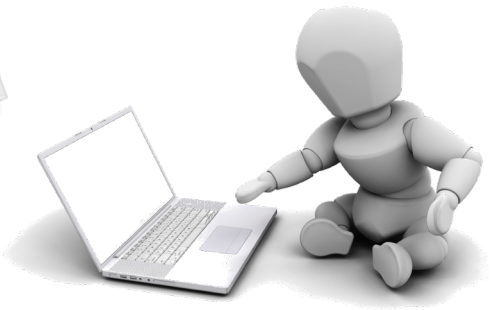
Software Engineering for Ultra-large-scale Systems



Amazon's massive AWS outage on Feb. 28th 2017 took more than 4 hours to recover.

Logs are one of the only resources of information during load testing





Operator

Developer

"The Bone of the System"
[ICSE SEIP 2016]

Microsoft Research

Logs record valuable information during system execution

```
start_task() {
    try{
    Task t=new Task();
    …
    t.start();

    }
    catch(Exception e){

    }
}
```

Add logging statements

```
void
usage (char *name)
{
  printf ("usage:\n");
  printf ("%s -a [-c file",
name);
#ifdef LOFI
  printf ("[-g] [-G] ");
#endif
  printf ("[-p what] [-r]
[-u file [type]]");
#ifdef LOFI
  printf (" [-w love] [-w
eweld] [-z size] ");
#endif
```

Add logging statements

```
void
usage (char *name)
{
  printf ("usage:\n");
  printf ("%s -a [-c file",
name);
#ifdef LOFI
  printf ("[-g] [-G] ");
#endif
  printf ("[-p what] [-r]
[-u file [type]]");
#ifdef LOFI
  printf (" [-w love] [-w
eweld] [-z size] ");
#endif
```

Release

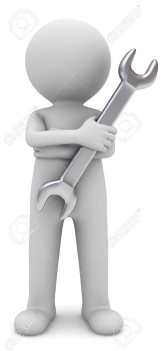
Add logging statements

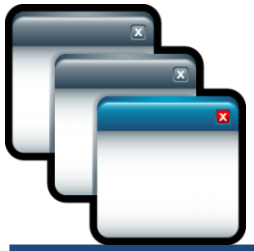
```
void
usage (char *name)
{
  printf ("usage:\n");
  printf ("%s -a [-c file",
name);
#ifdef LOFI
  printf ("[-g] [-G] ");
#endif
  printf ("[-p what] [-r]
[-u file [type]]");
#ifdef LOFI
  printf (" [-w love] [-w
eweld] [-z size] ");
#endif
```

Release

Produce during load tests

Operators develop Log Processing Apps to understand load testing results





Log Processing Apps

grep "Error starting Task"

Time = 1000, Task A started

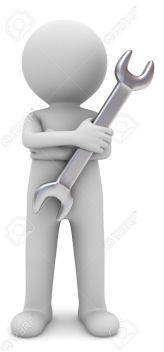
…

…

Time = 3000, Error starting Task C

…

Log Processing Apps



Add logging statements

Release

Produce during load tests

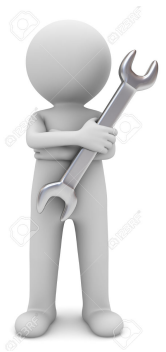
Perl

RegEx [^...]

Analyze

Load related issues

Log Processing Apps



Make logging decisions

Release

Produce during load tests

Analyze

Perl

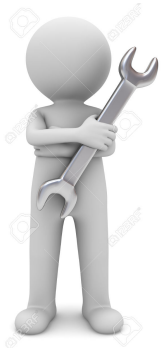
RegEx
[^...]

Make logging decisions

Release

Load related issues

Log Processing Apps

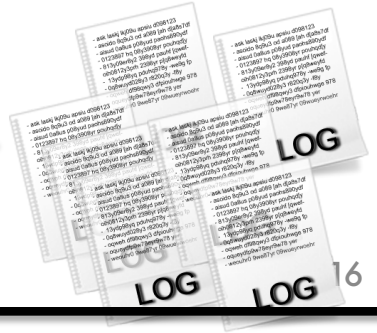


How to analyze logs

Perl

RegEx [^...]

Report



Produce at run-time

Analyzing testing results using logs

Small sample data and pseudo cloud

Big data and real-life cloud

hadoop

BDA platform

hadoop

BDA platform

Execution sequences

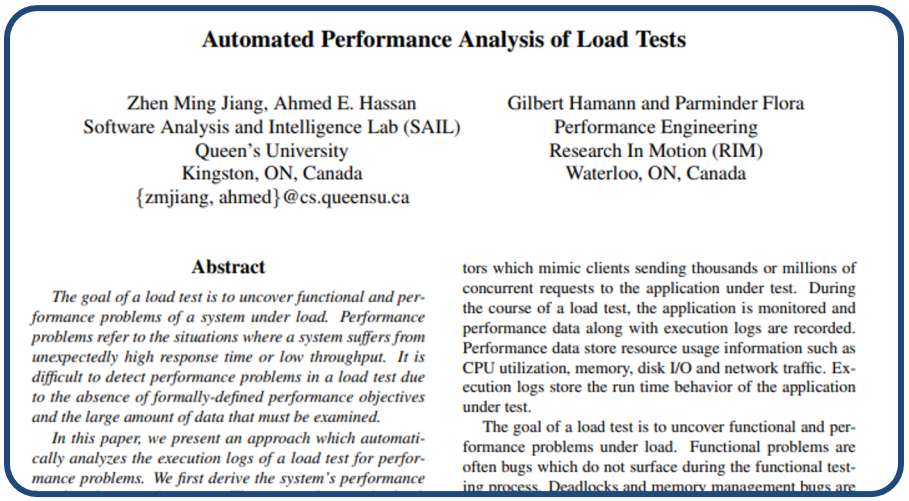
Execution sequence delta

Execution sequences

Abstracting logs → Linking logs → Simplifying sequences

[ICSE 2013]

Logs are widely used in log tests



Automated Performance Analysis of Load Tests

Zhen Ming Jiang, Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
Queen's University
Kingston, ON, Canada
{zmjiang, ahmed}@cs.queensu.ca

Gilbert Hamann and Parminder Flora
Performance Engineering
Research In Motion (RIM)
Waterloo, ON, Canada

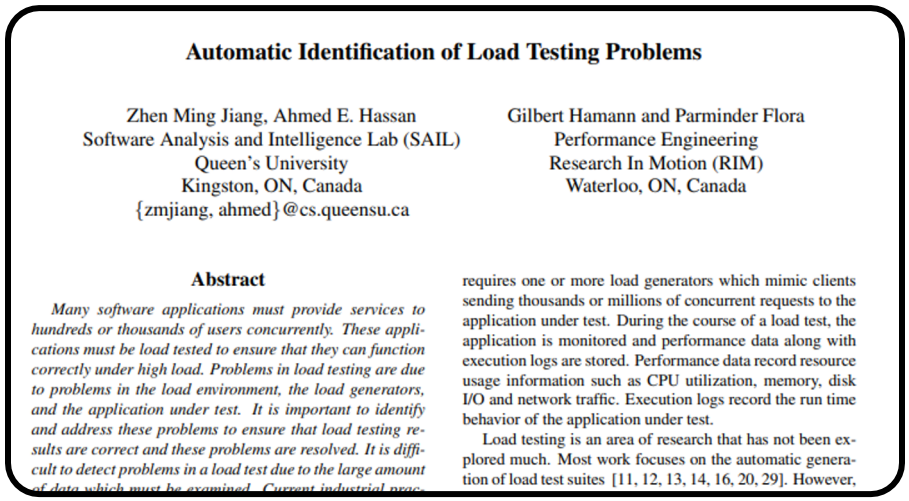
Abstract

The goal of a load test is to uncover functional and performance problems of a system under load. Performance problems refer to the situations where a system suffers from unexpectedly high response time or low throughput. It is difficult to detect performance problems in a load test due to the absence of formally-defined performance objectives and the large amount of data that must be examined.

In this paper, we present an approach which automatically analyzes the execution logs of a load test for performance problems. We first derive the system's performance

tors which mimic clients sending thousands or millions of concurrent requests to the application under test. During the course of a load test, the application is monitored and performance data along with execution logs are recorded. Performance data store resource usage information such as CPU utilization, memory, disk I/O and network traffic. Execution logs store the run time behavior of the application under test.

The goal of a load test is to uncover functional and performance problems under load. Functional problems are often bugs which do not surface during the functional testing process. Deadlocks and memory management bugs are



Automatic Identification of Load Testing Problems

Zhen Ming Jiang, Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
Queen's University
Kingston, ON, Canada
{zmjiang, ahmed}@cs.queensu.ca

Gilbert Hamann and Parminder Flora
Performance Engineering
Research In Motion (RIM)
Waterloo, ON, Canada

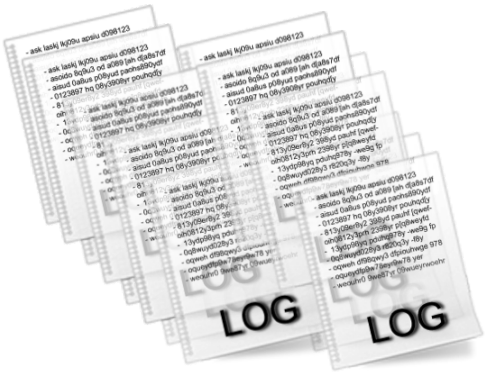
Abstract

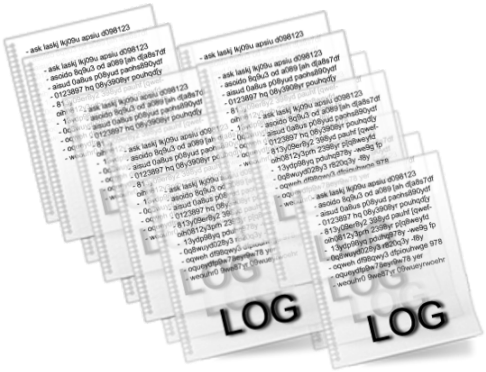
Many software applications must provide services to hundreds or thousands of users concurrently. These applications must be load tested to ensure that they can function correctly under high load. Problems in load testing are due to problems in the load environment, the load generators, and the application under test. It is important to identify and address these problems to ensure that load testing results are correct and these problems are resolved. It is difficult to detect problems in a load test due to the large amount of data which must be examined. Current industrial prac-

requires one or more load generators which mimic clients sending thousands or millions of concurrent requests to the application under test. During the course of a load test, the application is monitored and performance data along with execution logs are stored. Performance data record resource usage information such as CPU utilization, memory, disk I/O and network traffic. Execution logs record the run time behavior of the application under test.

Load testing is an area of research that has not been explored much. Most work focuses on the automatic generation of load test suites [11, 12, 13, 14, 16, 20, 29]. However,

Logs are widely used for various downstream tasks of load tests





Failure Diagnosis
[TSE 2021]

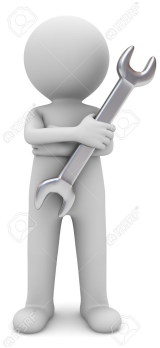
Load test generation
[ASE 2019]

System Comprehension
[Locke et al. TSE 2021]

Software logs are crucial for a variety of downstream tasks in practice of load tests

System issues

Log processing apps



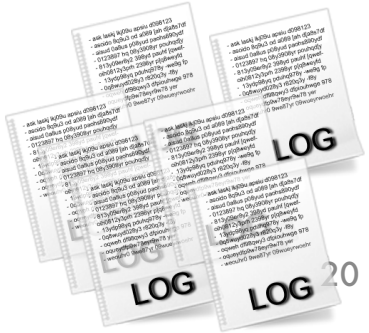
Perl

RegEx [^...]

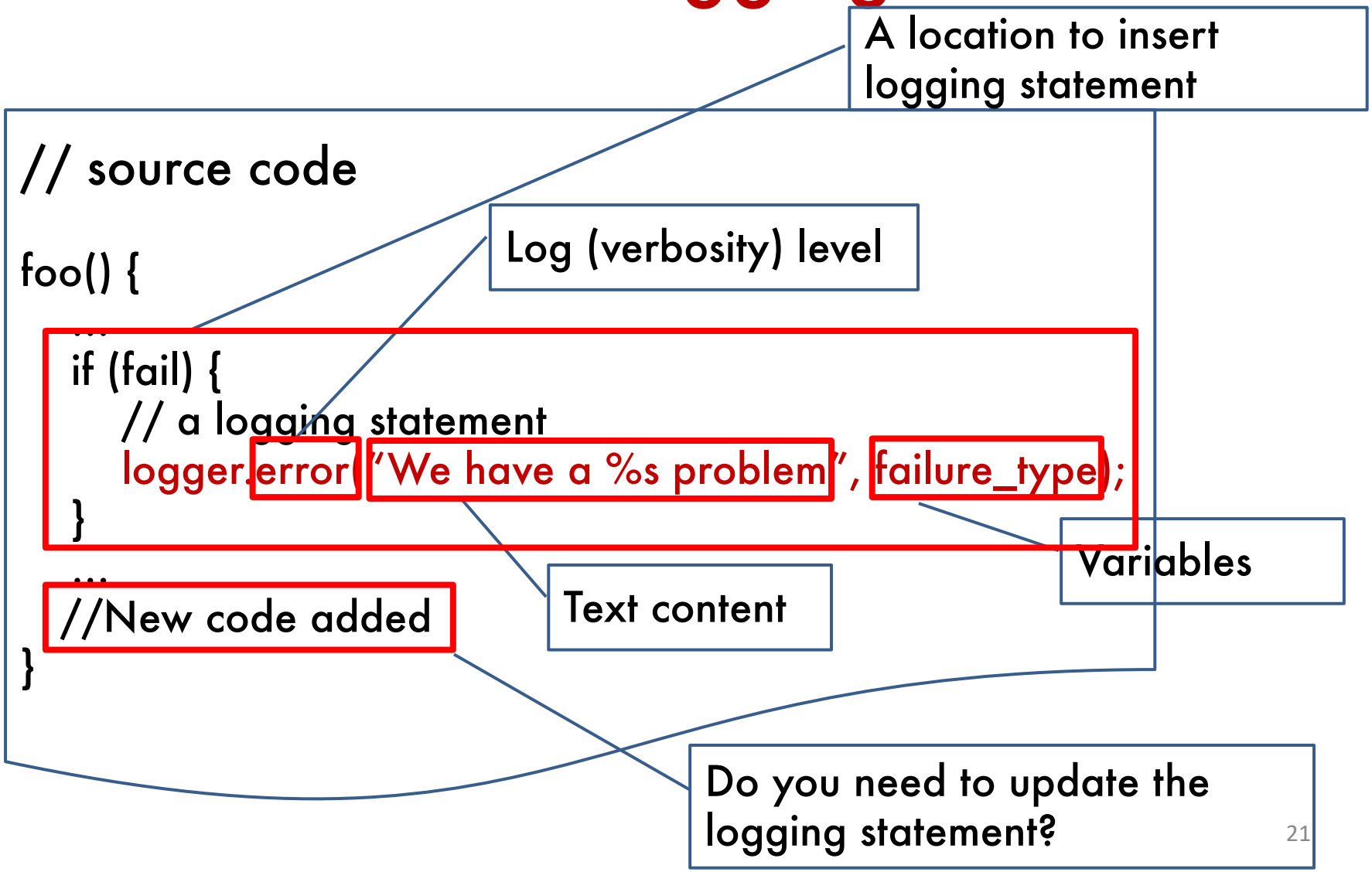
Report

How to make logging statements?

Produce at run-time



An example of a logging statement and logging decisions



Proactively suggesting the generation of logging statements

```
// source code

foo() {
    ...
    if (fail) {
        // a logging statement
        logger.error("We have a %s problem",
failure_type);
    }
    ...
  //New code added
}
```

Where to log?

What to log?

Which level to log?

When to update log?

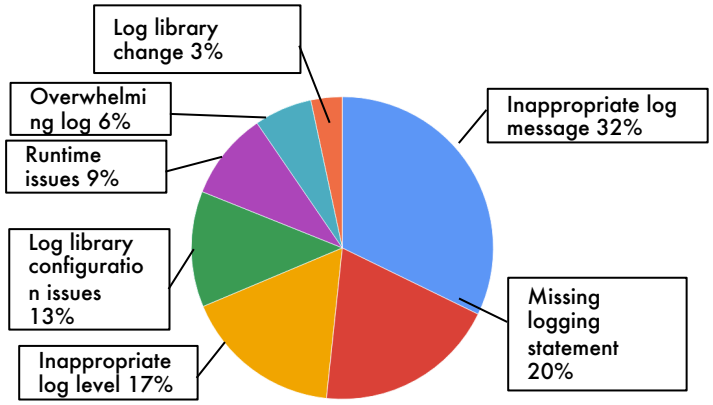
Heng Li

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

[EMSE 2017, 2018, 2019]
[ASE 2020, ICSE 2021, SANER 2022]

Zhenhao Li

Detecting the anti-patterns/bugs in logging statements



Verbosity level checker

Verbosity level guard checker

Typo checker

Duplicate Logging Statements:
Two or more logging statements that have identical static text messages.

Exception catch block Checker

Mehran Hassani

[EMSE 2018, ICSE 2019, TSE 2021]

System issues

Log processing apps

Ma log de

How to make logging statements?

How to analyze logs

Perl

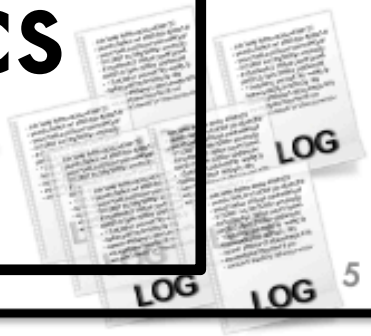
RegEx [^...]

Report

Re

Logging analytics infrastructure

Produce at run-time



System issues

Log processing apps

Mak
log
de

VMware vCenter Log Insight

- Smart Parsing
- Advanced Indexing

logstash

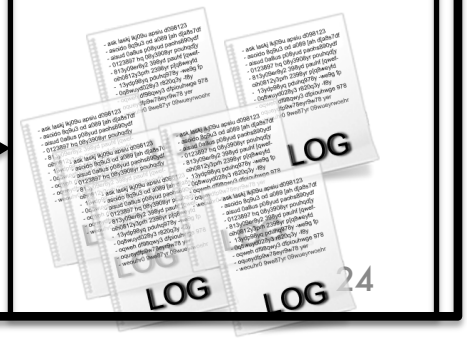
ELK Stack

splunk>

graylog

Release

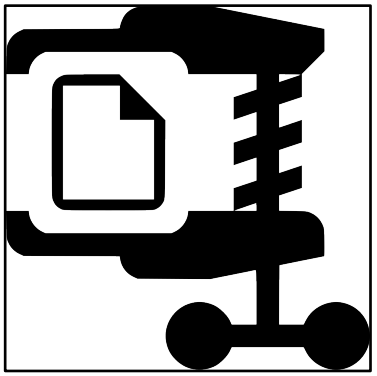
Produce at run-time



System issues

Log processing apps

Mak
log
de



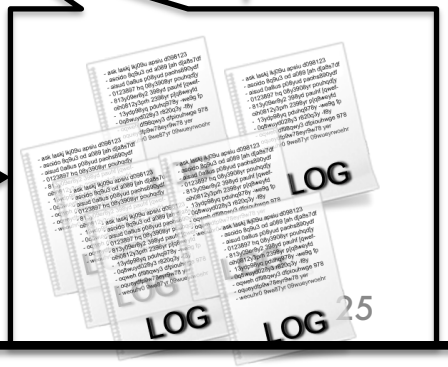
Compression

Raw log (Unstructured)	17/06/09 20:11:11 INFO storage.BlockManager: Found block *rdd_42_20* locally
Parsed log (Structured)	**Timestamp**: 17/06/09 20:11:11; **Level**: INFO **Logger**: storage.BlockManager **Static template**: Found block <*> locally **Dynamic variable(s)**: *rdd_42_20*

Parsing

Release

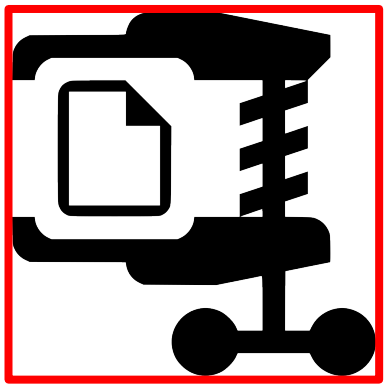
Produce at run-time



System issues

Log processing apps

Mak
log
de



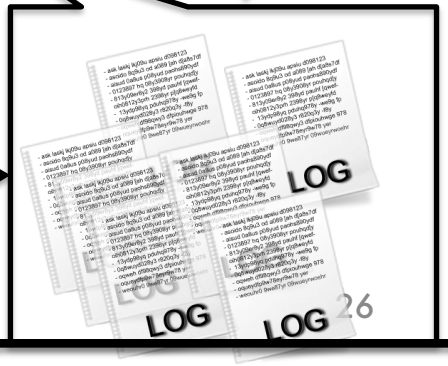
Compression

Raw log (Unstructured)	17/06/09 20:11:11 INFO storage.BlockManager: Found block *rdd_42_20* locally
Parsed log (Structured)	**Timestamp**: 17/06/09 20:11:11; **Level**: INFO **Logger**: storage.BlockManager **Static template**: Found block <*> locally **Dynamic variable(s)**: *rdd_42_20*

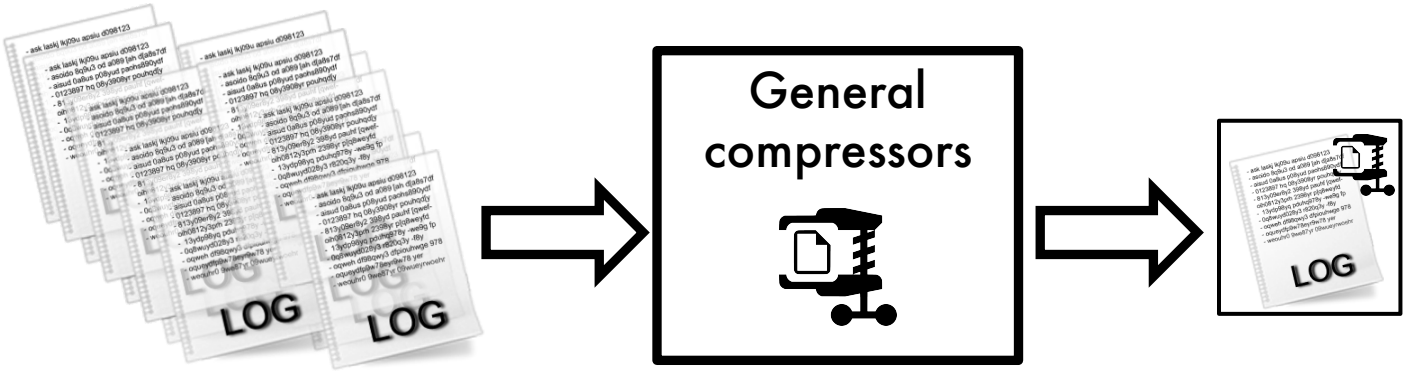
Parsing

Release

Produce at run-time



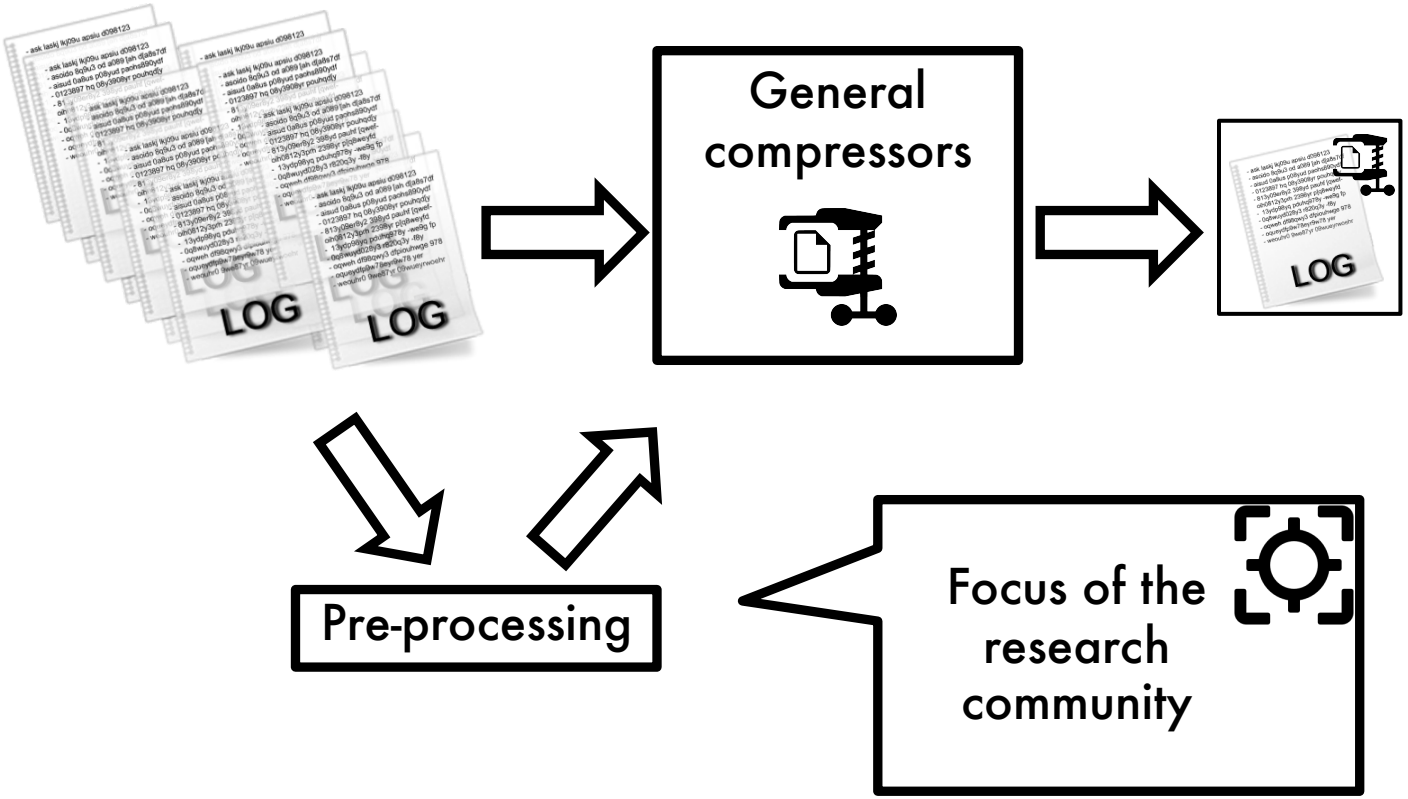
Leveraging general compressors on log data



Cannot achieve optimal performance!

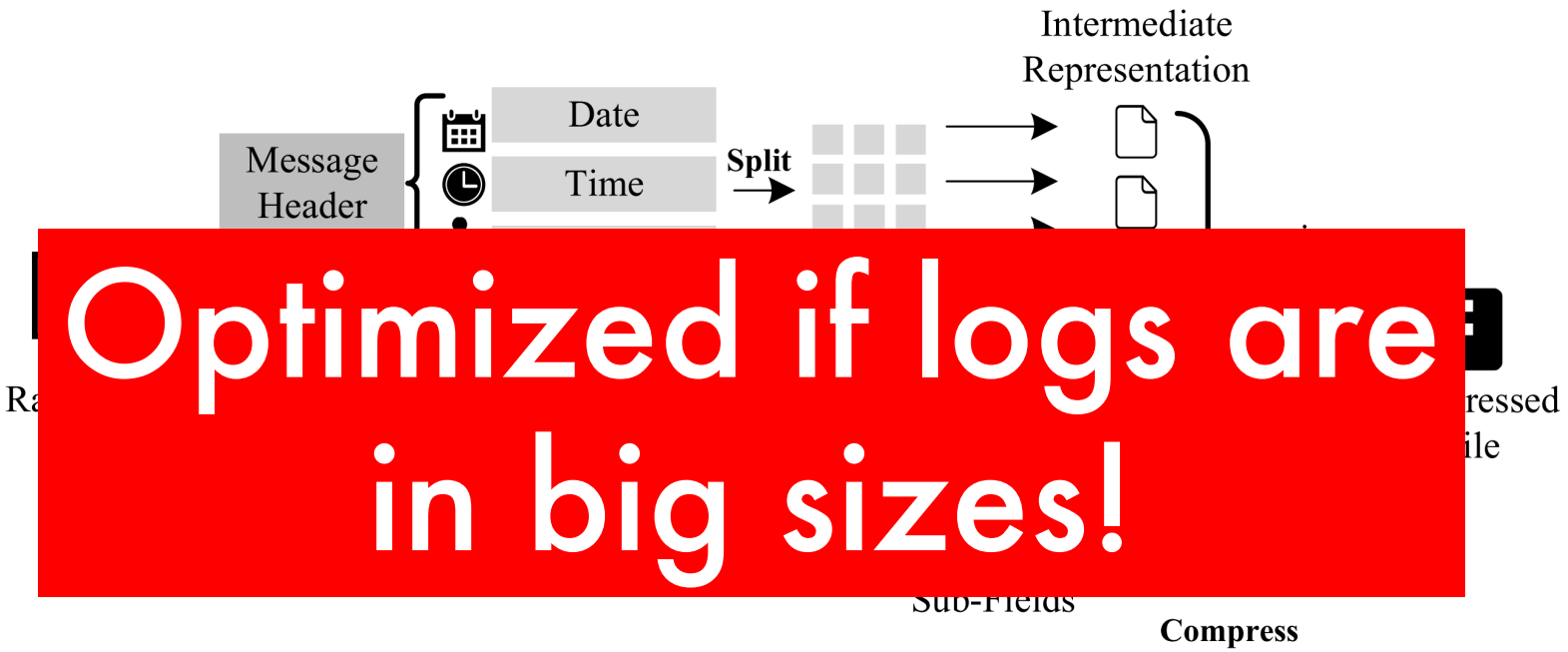
[EMSE 2020]

Leveraging general compressors on log data



[EMSE 2020]

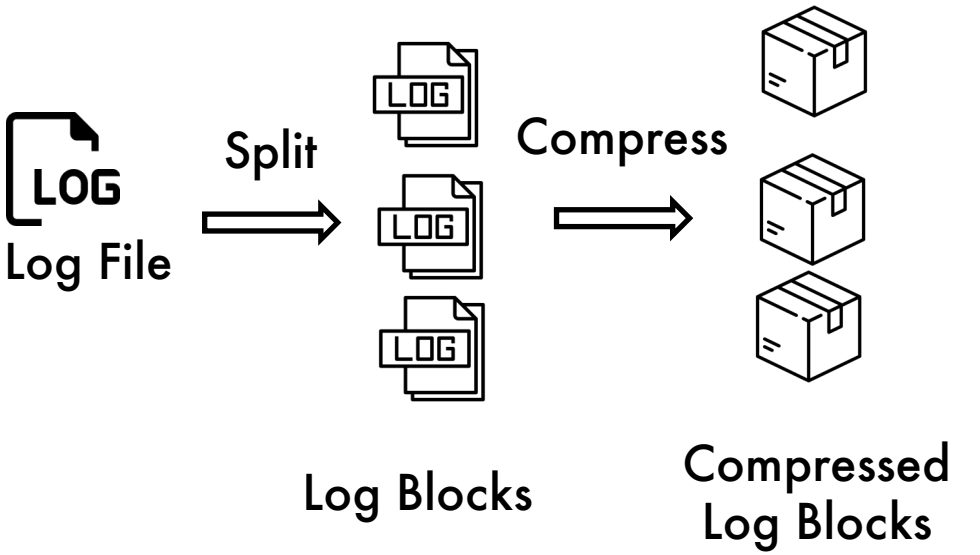
Logzip: Extracting Hidden Structures via Iterative Clustering for Log Compression



Optimized if logs are in big sizes!

Ref: Liu, Jinyang, et al. "Logzip: Extracting Hidden Structures via Iterative Clustering for Log Compression." *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019.

Logs are typically stored in small blocks



Apache LOG4J 2 ™ LOGBack The Generic, Reliable, Fast & Flexible Logging Framework log4net

Time/Size-based log rolling

ELK Stack

16KB/60KB

splunk>

128KB

DATADOG

256KB

HUMIO

384KB ~ 1024KB

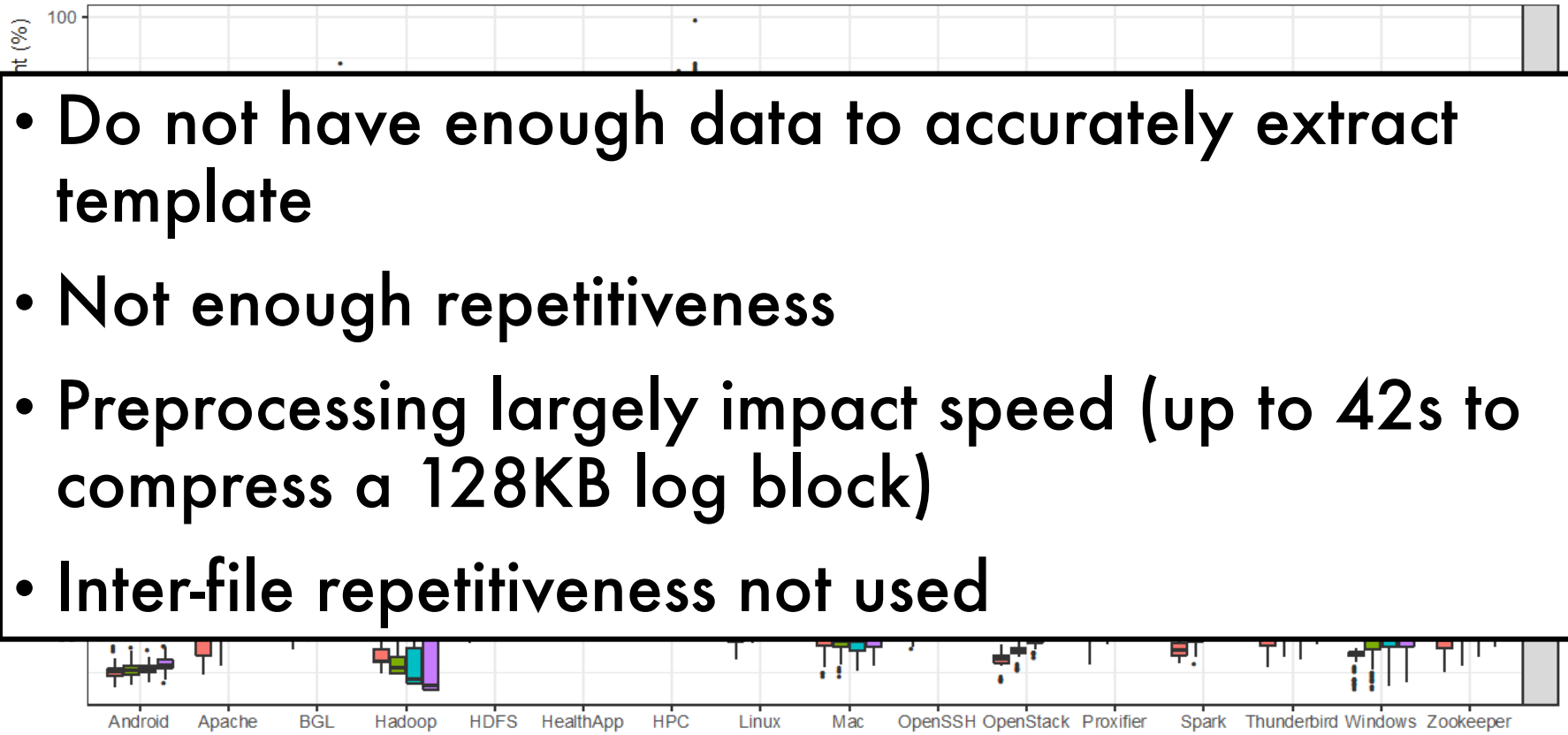
sumo logic

64KB

NGINX

64KB

Logzip does not perform well on small log blocks.



- Do not have enough data to accurately extract template
- Not enough repetitiveness
- Preprocessing largely impact speed (up to 42s to compress a 128KB log block)
- Inter-file repetitiveness not used

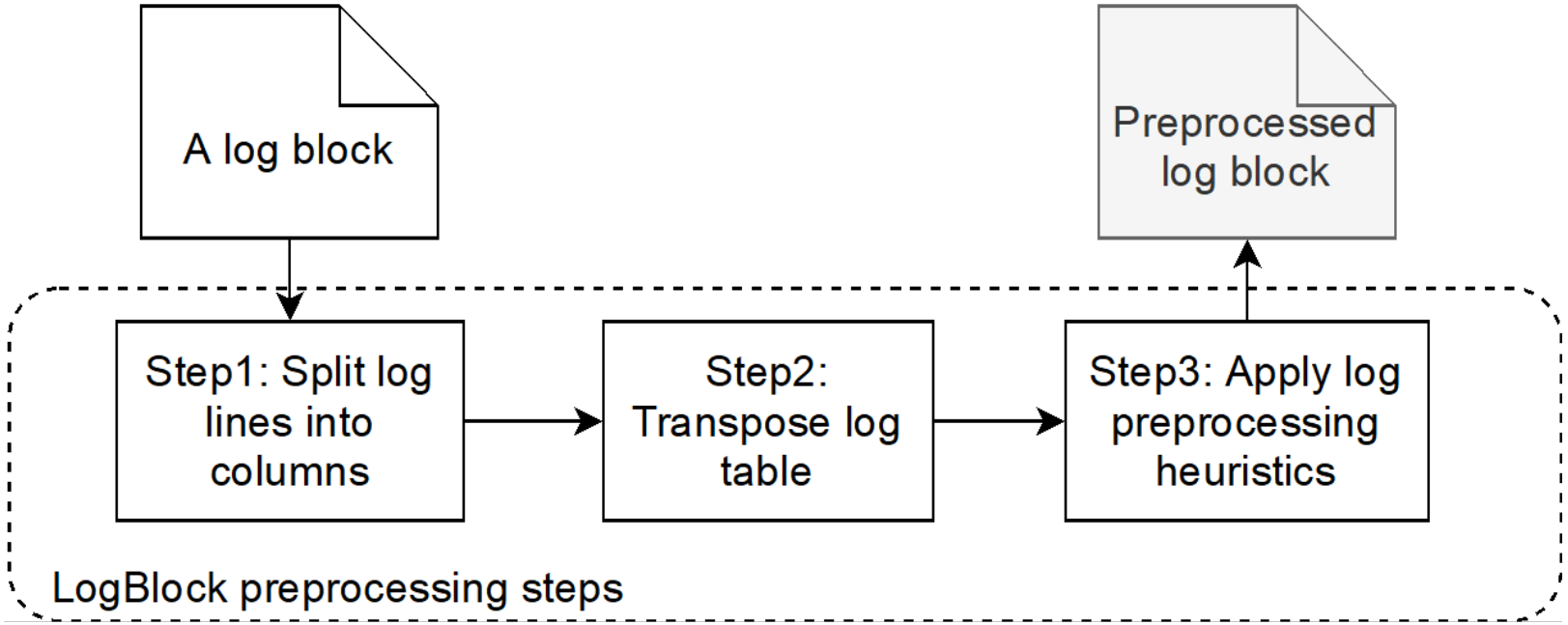
The compression ratios of Logzip are 4% to 98% (by a median of 63%) of the compression ratio without it.

Initial investigation on log data

We observe 4 types of repetitiveness from the non-content part of our selected log data.

- **T1: Identical tokens:** Tokens with the same information (e.g., Year component).
  - H1: Extract identical tokens: Extract the identical token and its number of occurrences.
- **T2: Similar numeric tokens:** Long & numeric tokens (e.g., Timestamp).
  - H2: Delta encoding for numbers: Save the delta between the current token and its prior token (first token preserved).
- **T3: Repetitive tokens**: Few tokens repeating a lot. (e.g., Log level)
  - H3: Build dictionary for repetitive tokens: Build a dictionary for each identical token and replace tokens with their indexes.
- **T4: Tokens with common prefix string:** Tokens start with the same information (e.g., Module).
  - H4: Extract common prefix string: Save the prefix string and store the remaining part of each token.

Design of our preprocessing approach: LogBlock



We do not perform extra information reduction steps to log content part for compression performance concern.

[TSE 2021]

LogBlock's preprocessing example

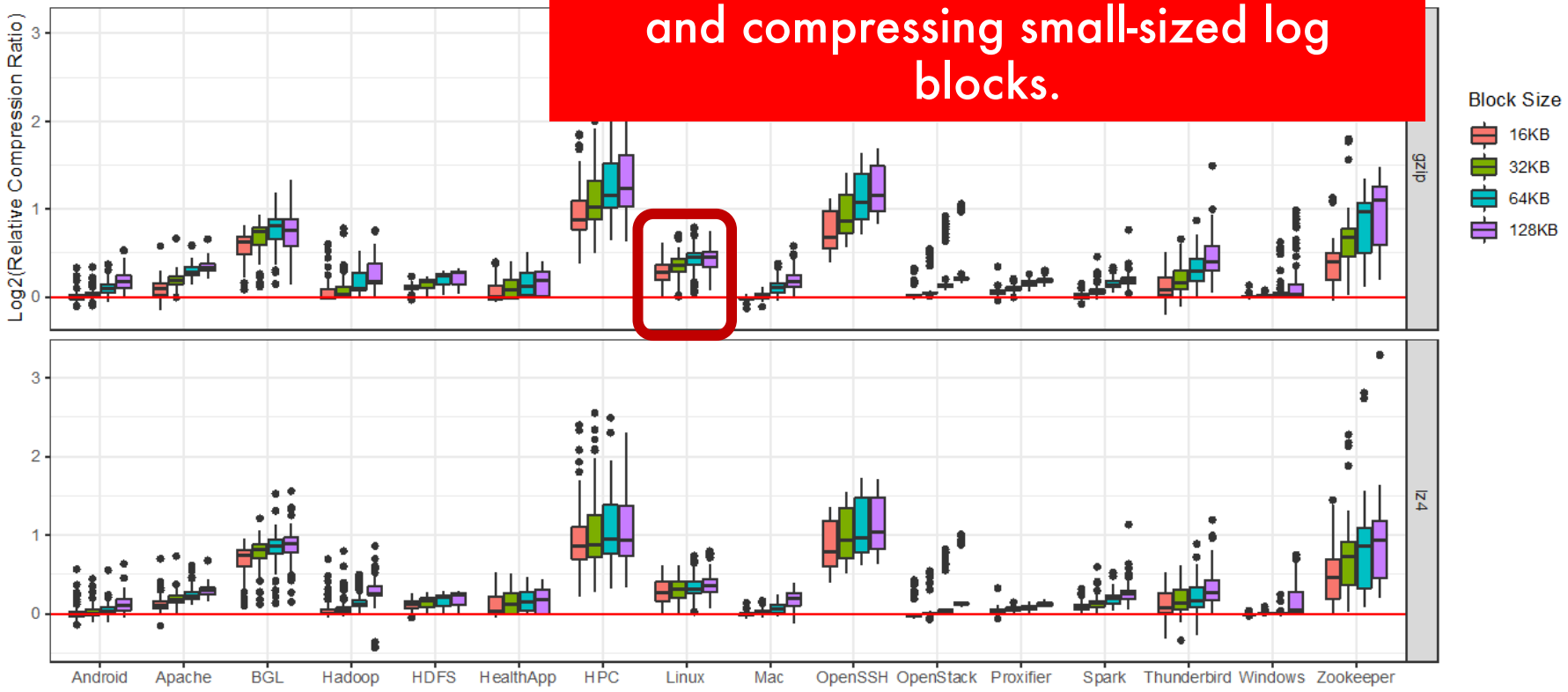
Line	Year	Log level	Log content
1	2015	WARN	Send worker leaving thread
2	2015	INFO	Received connection request 1
3	2015	WARN	Send worker leaving thread
4	2015	WARN	Interrupted while waiting message
5	2015	INFO	Received connection request 3

Transpose

Line	1	2	3	4	5
Year	2015	2015	2015	2015	2015
Log level	WARN	INFO	WARN	WARN	INFO
Log content	Send worker leaving thread	Received connection request 1	Send worker leaving thread	Interrupted while waiting message	Received connection request 3

LogBlock improves the compression ratio on small log blocks

Our approach is 31.0 to 50.1 times faster than Logzip in preprocessing and compressing small-sized log blocks.

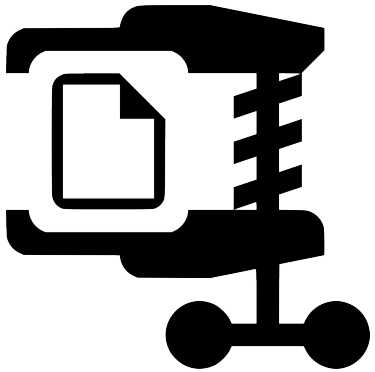


Our approach improves the compression ratio by a median of 5%, 9%, 15% and 21% on 16KB, 32KB, 64KB, and 128KB blocks in comparison to compression without any preprocessing.

System issues

Log processing apps

Mak
log
de



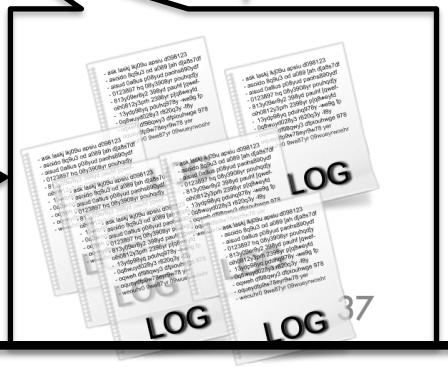
Compression

Raw log (Unstructured)	17/06/09 20:11:11 INFO storage.BlockManager: Found block *rdd_42_20* locally
Parsed log (Structured)	**Timestamp**: 17/06/09 20:11:11; **Level**: INFO **Logger**: storage.BlockManager **Static template**: Found block <*> locally **Dynamic variable(s)**: *rdd_42_20*

Parsing

Release

Produce at run-time



Log Parsing

logInfo("Found block $blockId locally")

Generate

17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_20 locally

Contain

Timestamp: 17/06/09 20:11:11; **Level**: INFO
Logger: storage.BlockManager
Static template: Found block <*> locally
Dynamic variable(s): rdd_42_20

Automated log parsing suffers from low efficiency

Spell: Streaming Parsing of System Event Logs

Min Du, Feifei Li
School of Computing, University of Utah
mind@cs.utah.edu, lifeifei@cs.utah.edu

Drain: An Online Log Parsing Approach with Fixed Depth Tree

Pinjia He*, Jieming Zhu*, Zibin Zheng†, and Michael R. Lyu*
*Computer Science and Engineering Department, The Chinese University of Hong Kong, China
{pjhe, jmzhu, lyu}@cse.cuhk.edu.hk
†Key Laboratory of Machine Intelligence and Advanced Computing (Sun Yat-sen University), Ministry of Education
School of Data and Computer Science, Sun Yat-sen University, China
zhzibin@mail.sysu.edu.cn

An automated approach for abstracting execution logs to execution events

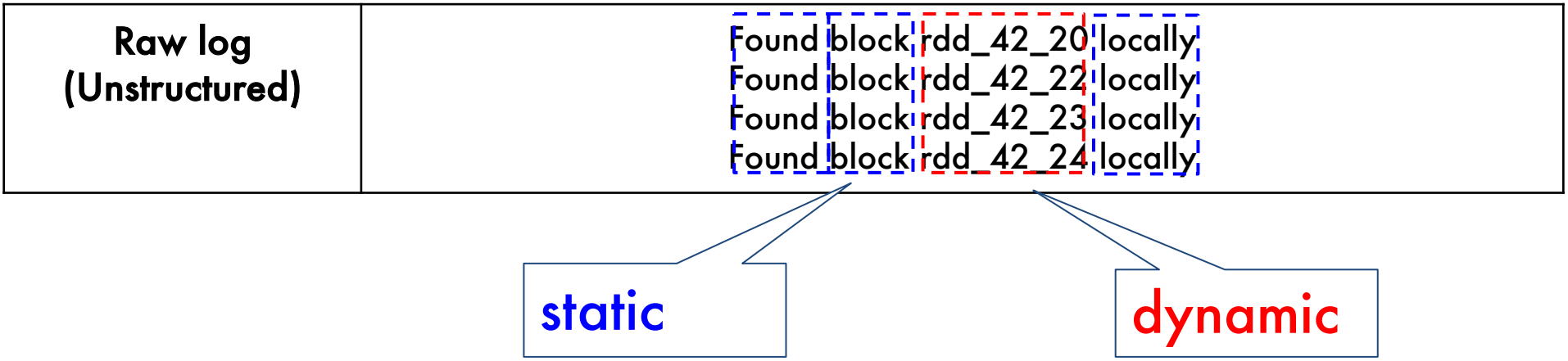
Zhen Ming Jiang[1, *, †], Ahmed E. Hassan[1], Gilbert Hamann[2] and Parminder Flora[2]

Efficiency is an important concern for log parsing

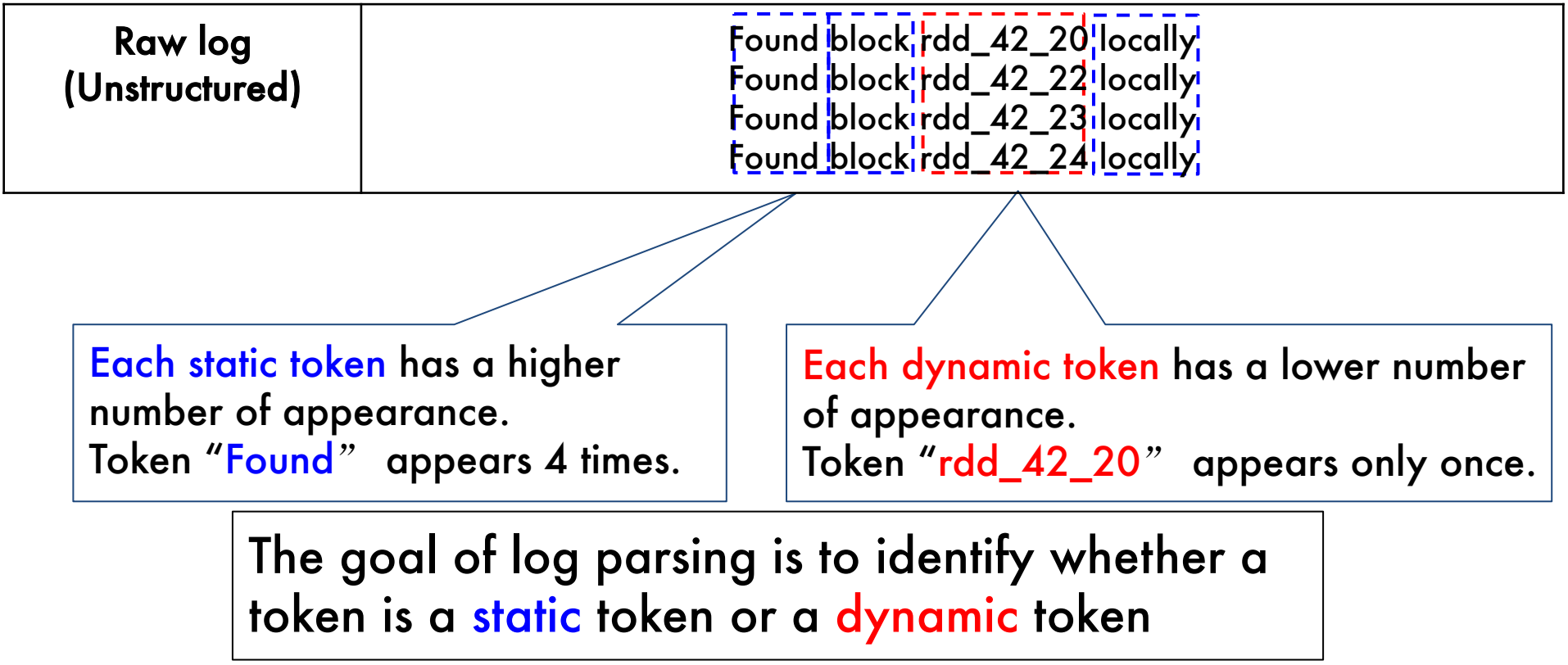
The main task of log parsing

Raw log (Unstructured)	Found block rdd_42_20 locally Found block rdd_42_22 locally Found block rdd_42_23 locally Found block rdd_42_24 locally

The main task of log parsing

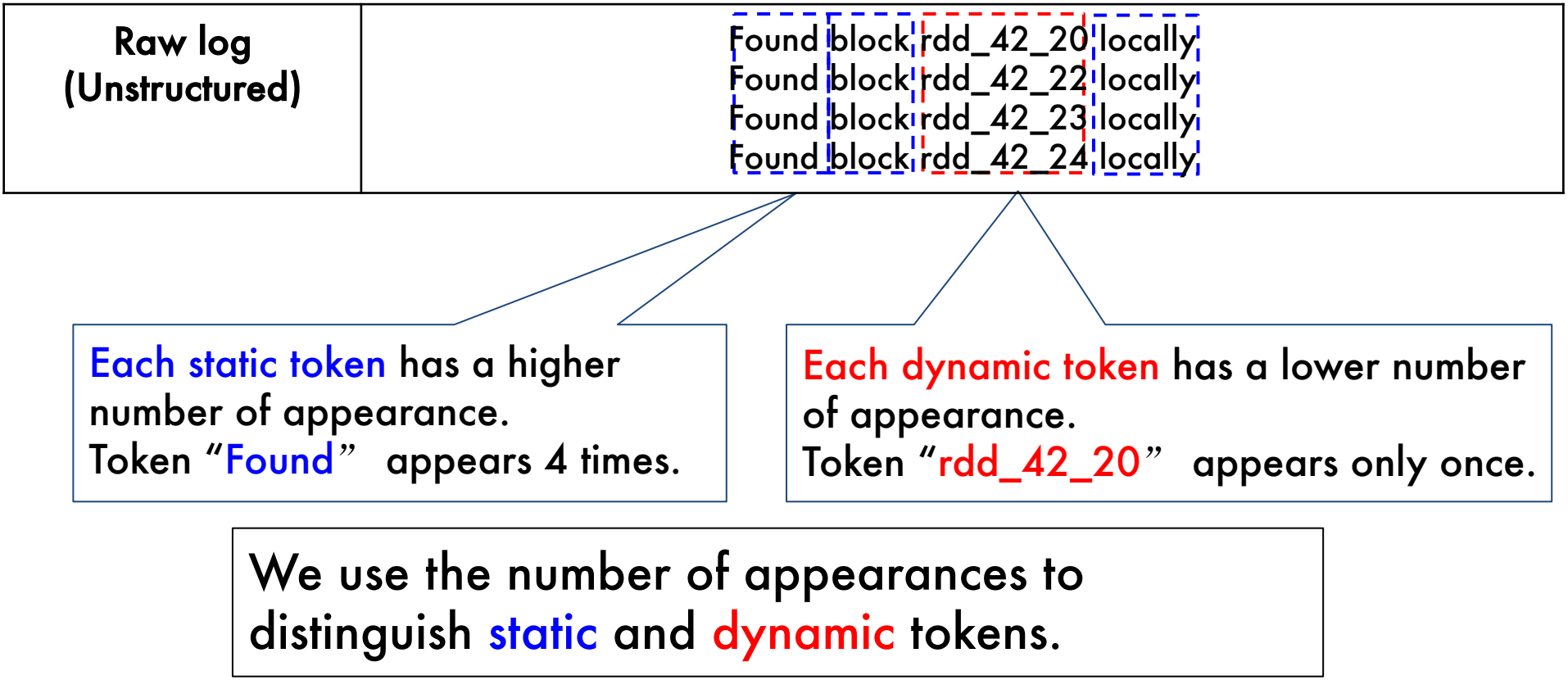


The main idea of Logram



[TSE 2020]

The main idea of Logram



[TSE 2020]

The main idea of Logram

Raw log (Unstructured)	Expecting attribute name [0x800f080d - CBS_E_MANIFEST_INVALID_ITEM] Failed to get next element [0x800f080d - CBS_E_MANIFEST_INVALID_ITEM]

A dynamic token may also appear frequently.

The main idea of Logram

Raw log (Unstructured)	Expecting attribute name [0x800f080d - CBS_E_MANIFEST_INVALID_ITEM] Failed to get next element [0x800f080d - CBS_E_MANIFEST_INVALID_ITEM]

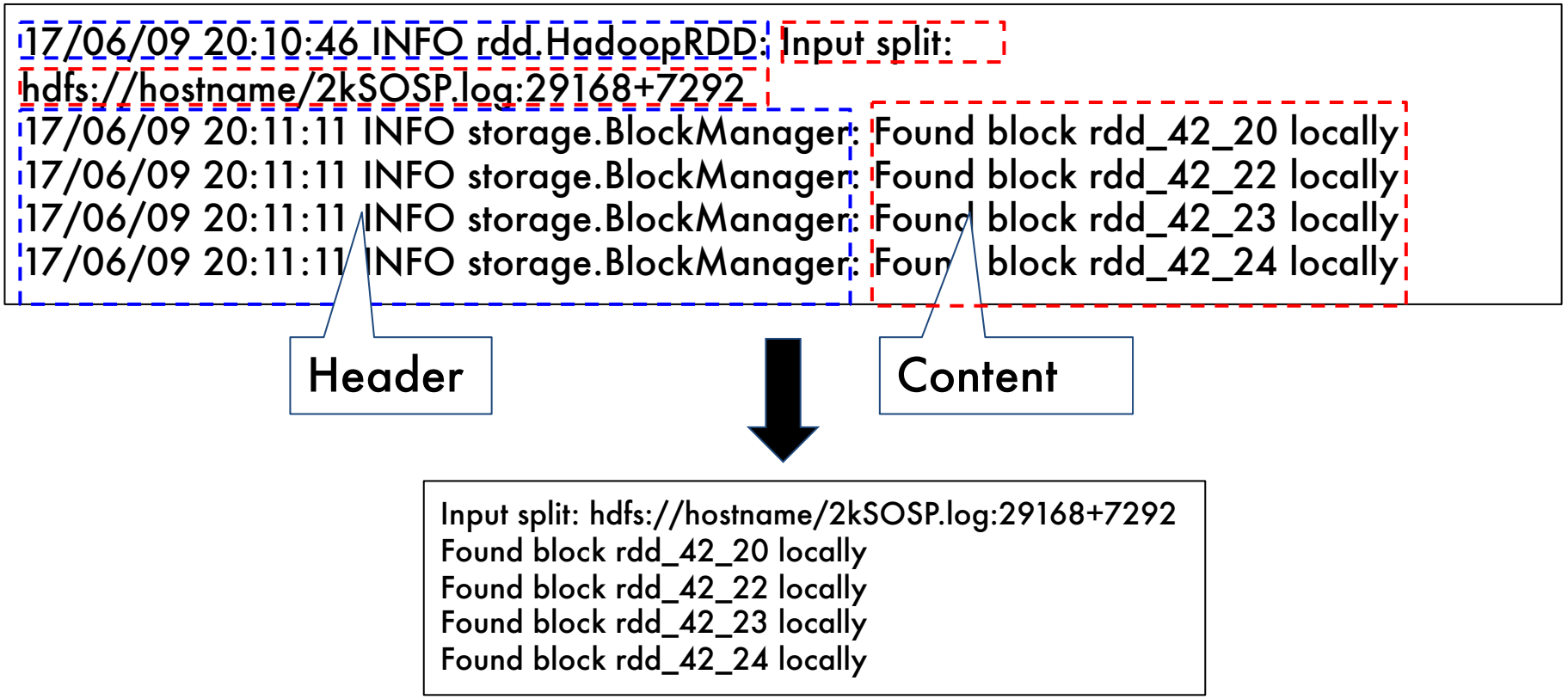
If we consider 3-grams instead of individual token, each 3-gram only appear once.

A dynamic token may also appear frequently.

Step 1: Dictionary setup for n-grams

```
17/06/09 20:10:46 INFO rdd.HadoopRDD: Input split:
hdfs://hostname/2kSOSP.log:29168+7292
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_20 locally
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_22 locally
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_23 locally
17/06/09 20:11:11 INFO storage.BlockManager: Found block rdd_42_24 locally
```

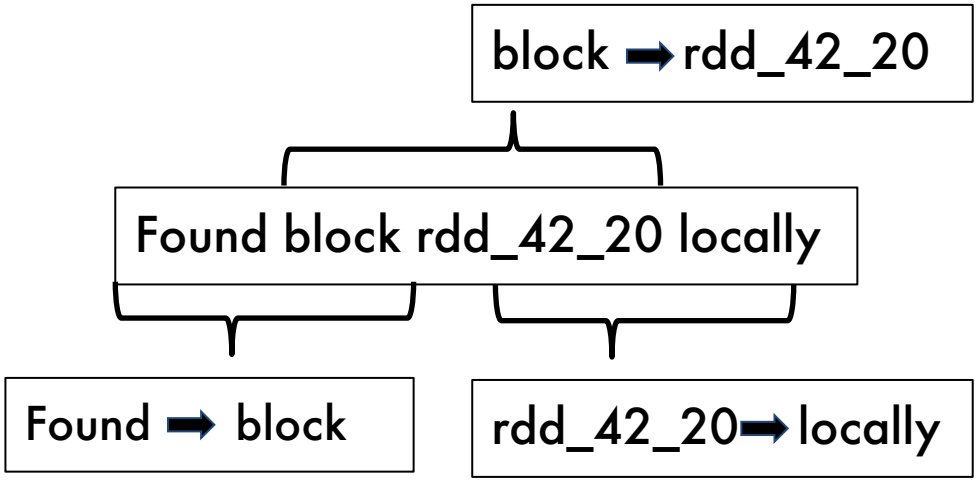
Step 1: Dictionary setup for n-grams



Step 1: Dictionary setup for n-grams

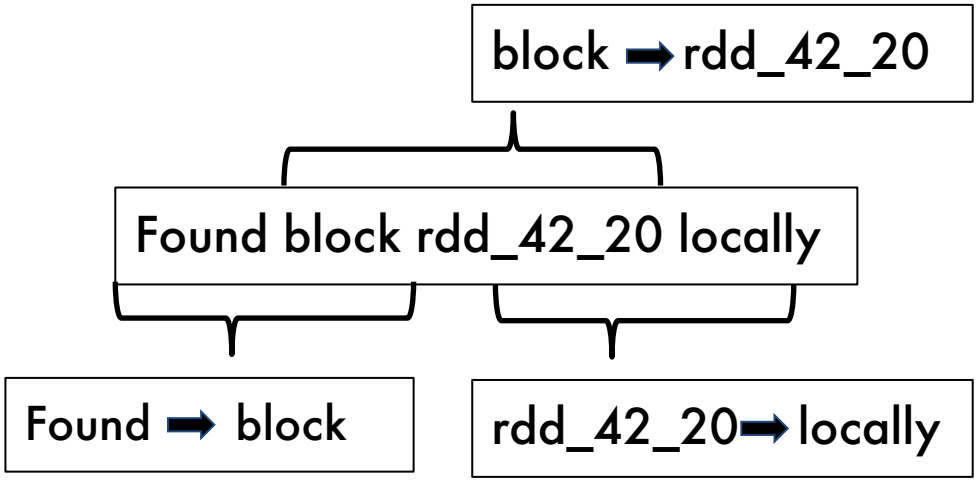
Found block rdd_42_20 locally

Step 1: Dictionary setup for n-grams

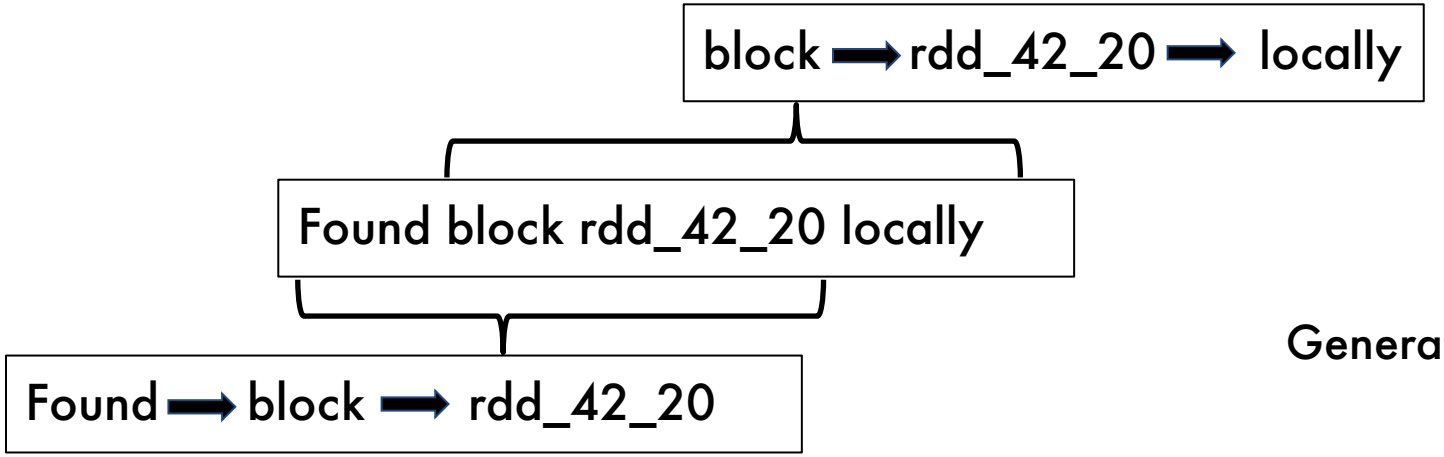


Generate 2-gram

Step 1: Dictionary setup for n-grams



Generate 2-gram



Generate 3-gram

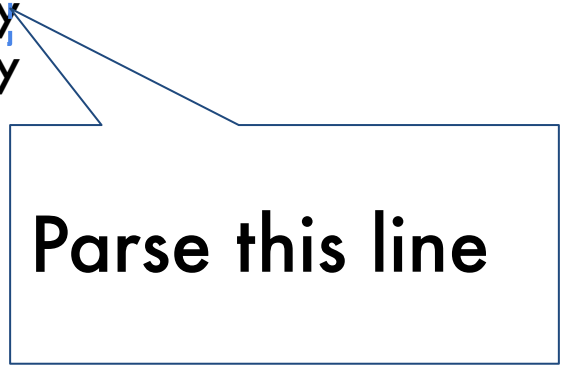
Step 1: Dictionary setup for n-grams

3-grams	#
Input ➡ split: ➡ hdfs://hostname/2kSOSP.log:21876+7292	1
split: ➡ hdfs://hostname/2kSOSP.log:21876+7292 ➡ Input	1
hdfs://hostname/2kSOSP.log:21876+7292 ➡ Input ➡ split:	1
…	
split: ➡ hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found	1
hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found ➡ block	1
Found ➡ block ➡ rdd_42_20	1
block ➡ rdd_42_20 ➡ locally	1
rdd_42_20 ➡ locally ➡ Found	1
locally ➡ Found ➡ block	3
…	

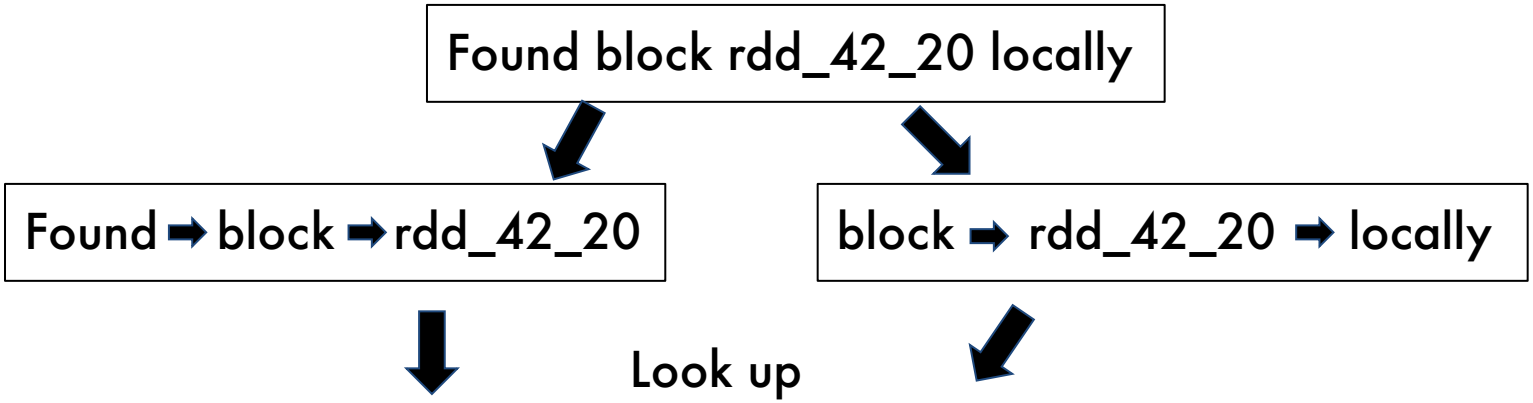
2-grams	#
Input ➡ split:	5
split: ➡ hdfs://hostname/2kSOSP.log:21876+7292	1
hdfs://hostname/2kSOSP.log:21876+7292 ➡ Input	1
…	
hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found	1
Found ➡ block	4
block ➡ rdd_42_20	1
rdd_42_20 ➡ locally	1
locally ➡ Found	4
…	

Step 2: Parsing logs with n-gram dictionaries

Input split: hdfs://hostname/2kSOSP.log:29168+7292
Found block rdd_42_20 locally
Found block rdd_42_22 locally

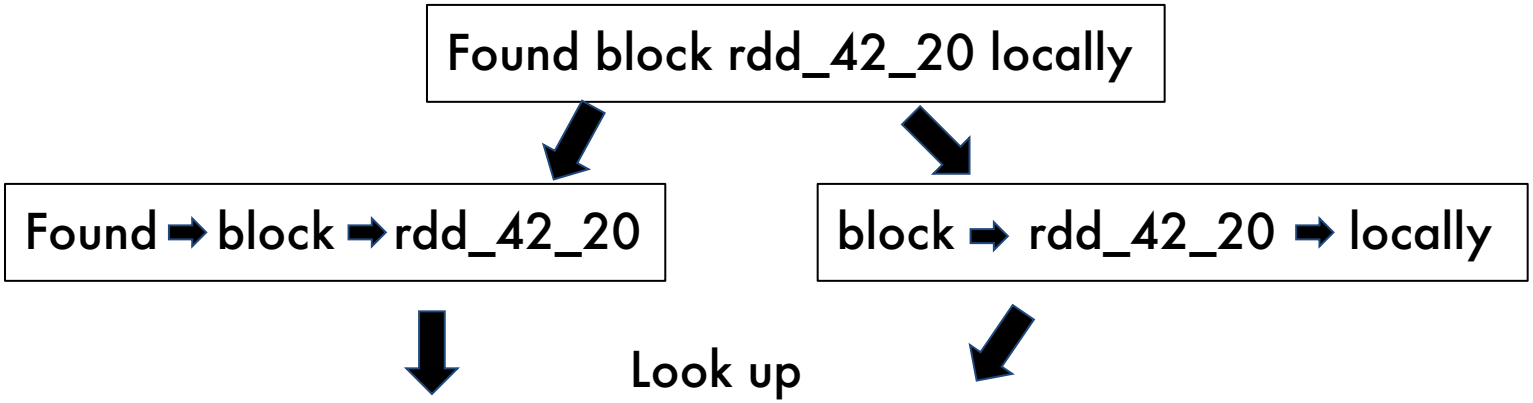


Step 2: Parsing logs with n-gram dictionaries



3-grams	# appearance
split: ➡ hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found	1
hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found ➡ block	1
Found ➡ block ➡ rdd_42_20	1
block ➡ rdd_42_20 ➡ locally	1
rdd_42_20 ➡ locally ➡ Found	1
locally ➡ Found ➡ block	5

Step 2: Parsing logs with n-gram dictionaries



3-grams	# appearance
split: ➡ hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found	1
hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found ➡ block	1
Found ➡ block ➡ rdd_42_20	1
block ➡ rdd_42_20 ➡ locally	1
rdd_42_20 ➡ locally ➡ Found	1
locally ➡ Found ➡ block	5

These two 3-grams may contain dynamic values since their appearances are only 1.

Step 2: Parsing logs with n-gram dictionaries

3-grams	# appearance
split: ➡ hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found	1
hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found ➡ block	1
Found ➡ block ➡ rdd_42_20	1
block ➡ rdd_42_20 ➡ locally	1
rdd_42_20 ➡ locally ➡ Found	1
locally ➡ Found ➡ block	5

Look up

2-grams	# appearance
hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found	1
Found ➡ block	4
block ➡ rdd_42_20	1
rdd_42_20 ➡ locally	1
locally ➡ Found	4

Step 2: Parsing logs with n-gram dictionaries

3-grams	# appearance
split: ➡ hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found	1
hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found ➡ block	1
Found ➡ block ➡ rdd_42_20	1
block ➡ rdd_42_20 ➡ locally	1
rdd_42_20 ➡ locally ➡ Found	1
locally ➡ Found ➡ block	5

Look up

2-grams	# appearance
hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found	1
Found ➡ block	4
block ➡ rdd_42_20	1
rdd_42_20 ➡ locally	1
locally ➡ Found	4

This 2-gram contains only static tokens.

Step 2: Parsing logs with n-gram dictionaries

3-grams	# appearance
split: ➡ hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found	1
hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found ➡ block	1
Found ➡ block ➡ rdd_42_20	1
block ➡ rdd_42_20 ➡ locally	1
rdd_42_20 ➡ locally ➡ Found	1
locally ➡ Found ➡ block	5

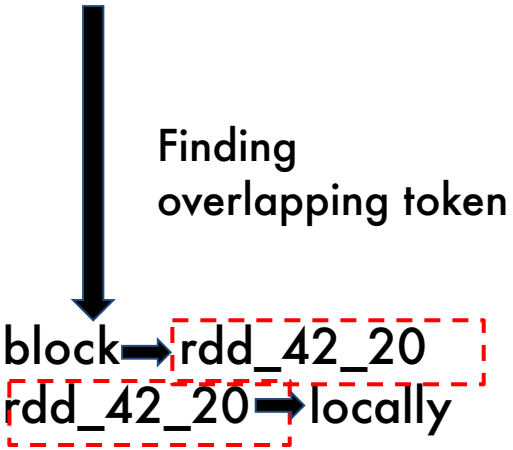
Look up

2-grams	# appearance
hdfs://hostname/2kSOSP.log:29168+7292 ➡ Found	1
Found ➡ block	4
block ➡ rdd_42_20	1
rdd_42_20 ➡ locally	1
locally ➡ Found	4

These 2-grams may contain dynamic tokens.

Step 2: Parsing logs with n-gram dictionaries

2-grams	# appearance
hdfs://hostname/2kSOSP.log:29168+7292 ➡Found	1
Found ➡block	4
block ➡rdd_42_20	1
rdd_42_20 ➡locally	1
locally ➡ Found	4



Step 2: Parsing logs with n-gram dictionaries

2-grams	# appearance
hdfs://hostname/2kSOSP.log:29168+7292 ➡Found	1
Found ➡block	4
block ➡rdd_42_20	1
rdd_42_20 ➡locally	1
locally ➡ Found	4

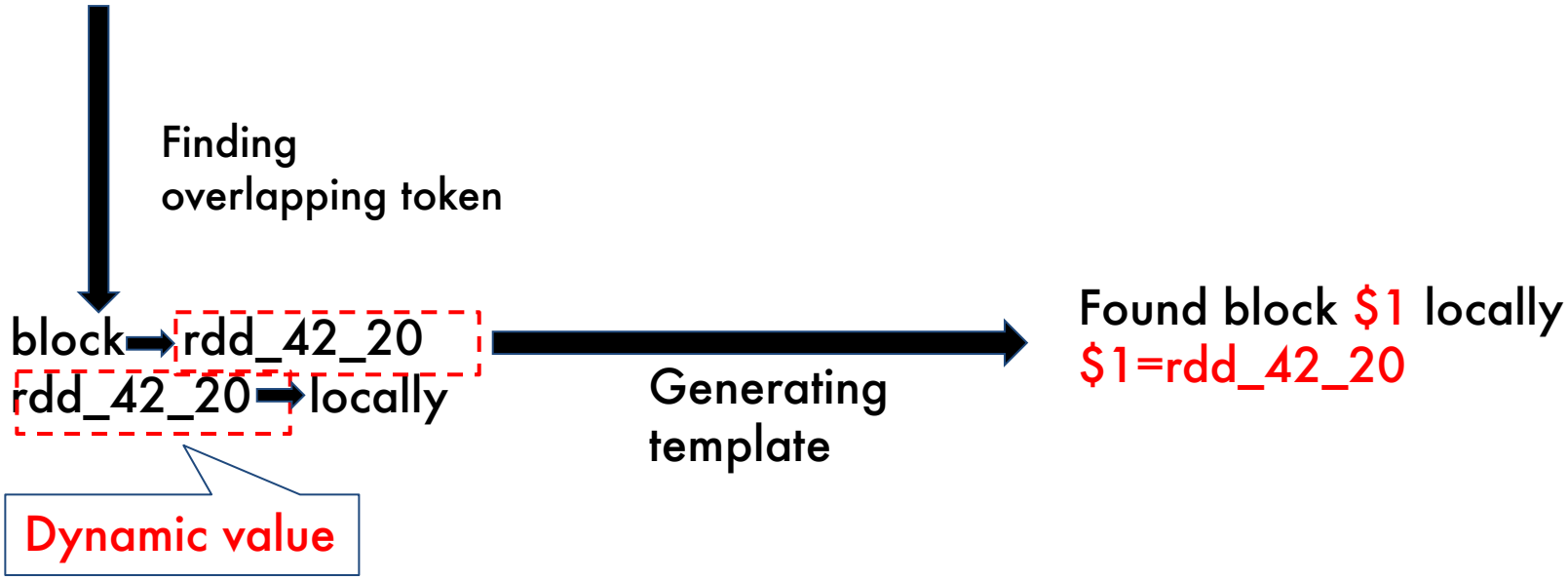
Finding
overlapping token

block ➡rdd_42_20
rdd_42_20 ➡locally

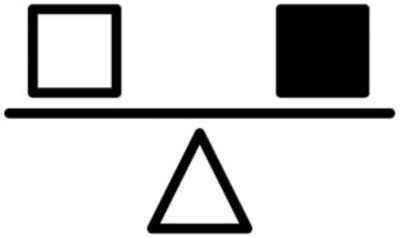
Dynamic value

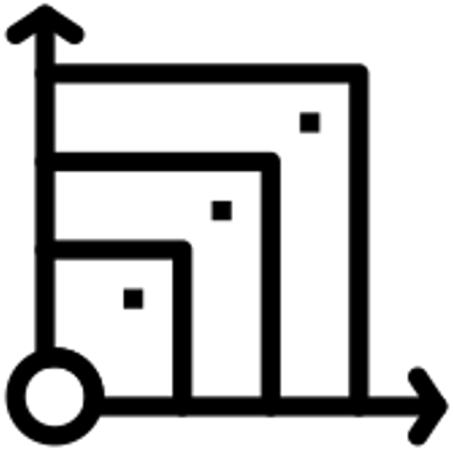
Step 2: Parsing logs with n-gram dictionaries

2-grams	# appearance
hdfs://hostname/2kSOSP.log:29168+7292 ➡Found	1
Found ➡block	4
block ➡rdd_42_20	1
rdd_42_20 ➡locally	1
locally ➡ Found	4

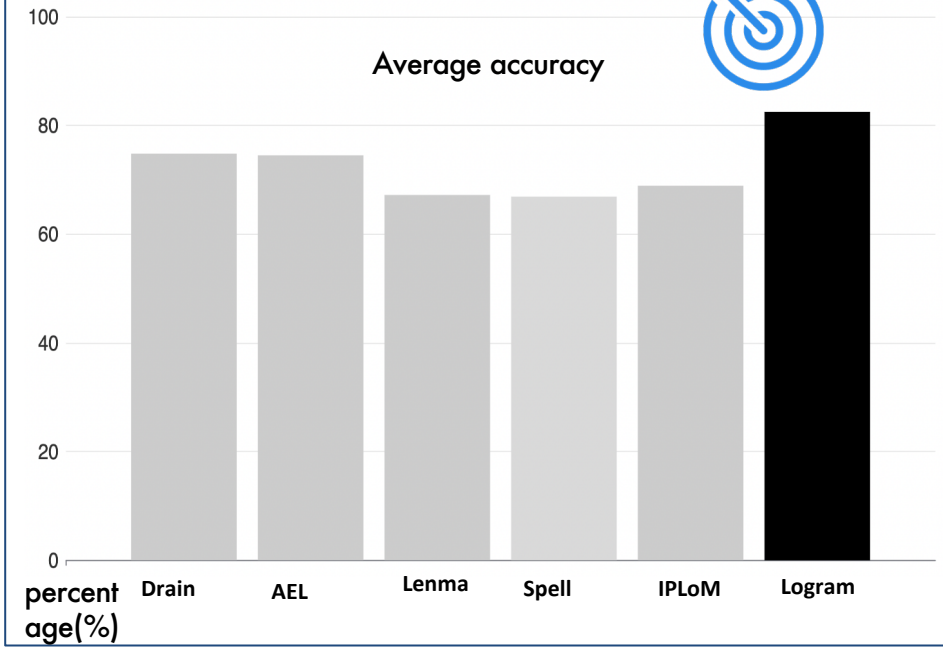


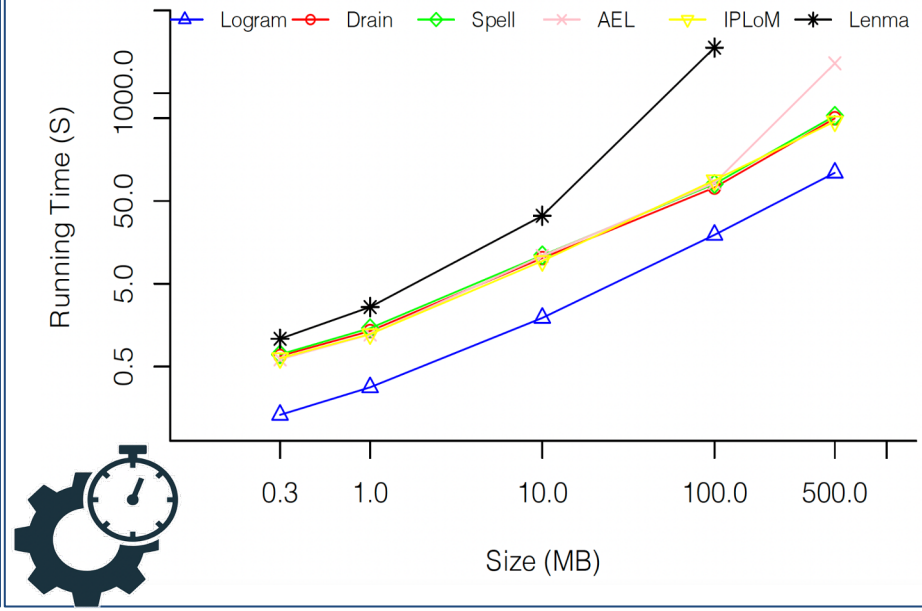
Evaluation

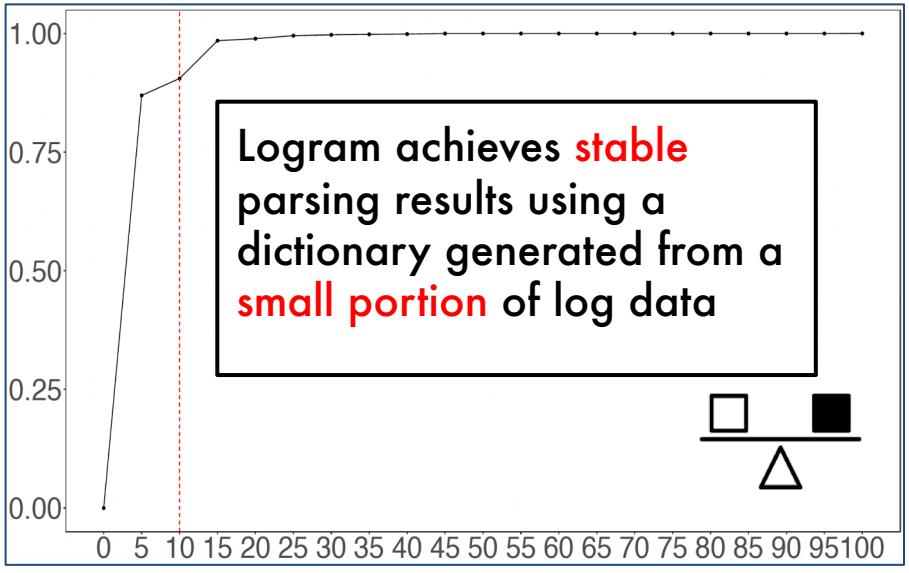




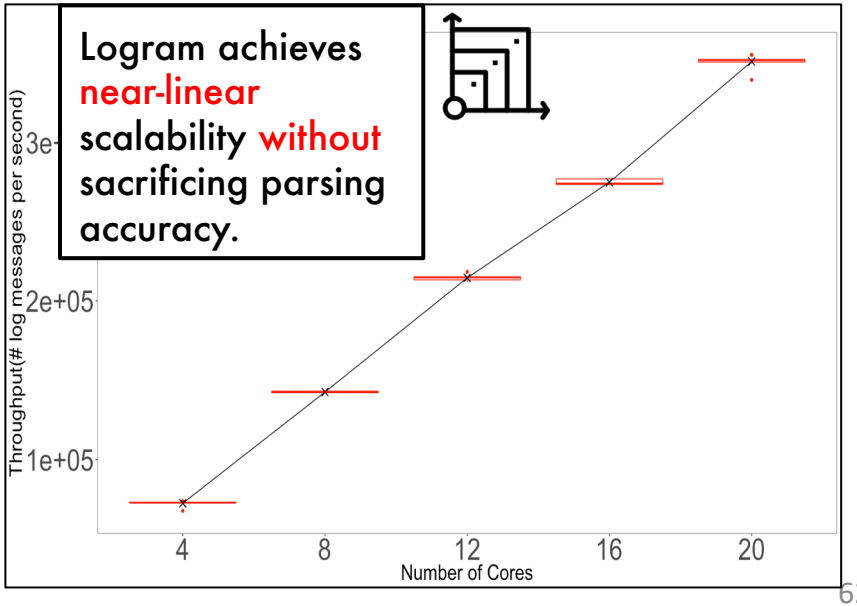
Accuracy Efficiency Stabilisation Scalability





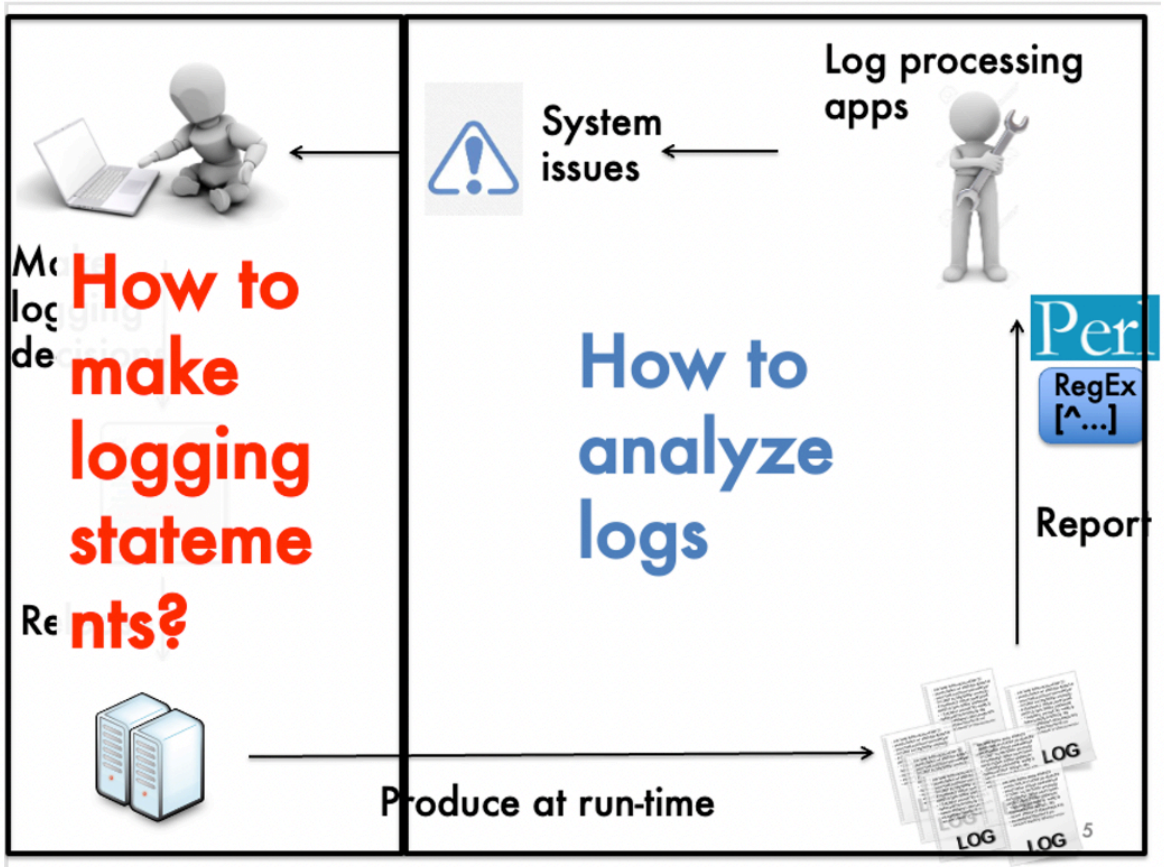


Logram achieves stable parsing results using a dictionary generated from a small portion of log data



Logram achieves near-linear scalability without sacrificing parsing accuracy.


2023



Logging analytics infrastructure


How to make logging stateme nts?

Logs are widely used for various downstream tasks of load tests

Failure Diagnosis [TSE 2021]

Load test generation [ASE 2019]

System Comprehension [Locke et al. TSE 2021]

Software logs are crucial for a variety of downstream tasks in practice of load tests

Analysis-aware (or even load-test driven) logging decision support

Logs are widely used for various downstream tasks of load tests

Failure Diagnosis [TSE 2021]

Load test generation [ASE 2019]

System Comprehension [Locke et al. TSE 2021]

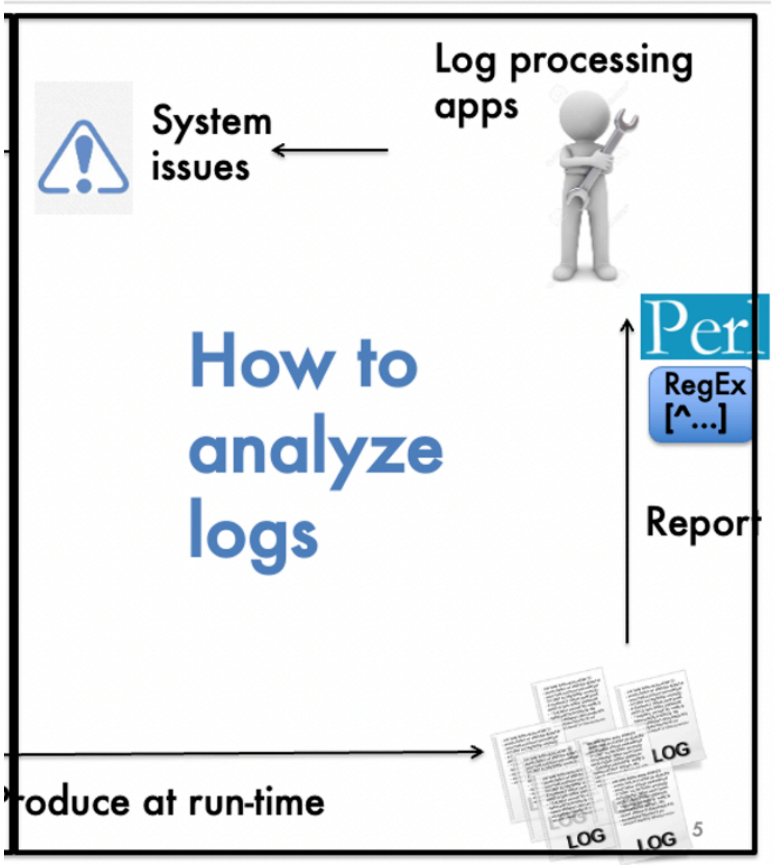
Software logs are crucial for a variety of downstream tasks in practice of load tests

A Survey on Automated Log Analysis for Reliability Engineering

SHILIN HE, Microsoft Research
PINJIA HE, Department of Computer Science, ETH Zurich
ZHUANGBIN CHEN, TIANYI YANG, YUXIN SU, and MICHAEL R. LYU, Department of Computer Science and Engineering, The Chinese University of Hong Kong

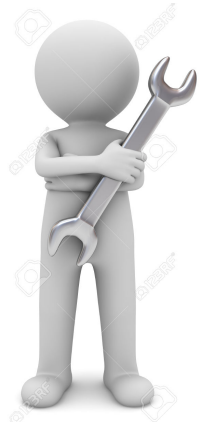
Logs are semi-structured text generated by logging statements in software source code. In recent decades, software logs have become imperative in the reliability assurance mechanism of many software systems because they are often the only data available that record software runtime information. As modern software is evolving into a large scale, the volume of logs has increased rapidly. To enable effective and efficient usage of modern software logs in reliability engineering, a number of studies have been conducted on automated log analysis. This survey presents a detailed overview of automated log analysis research, including how to automate and assist the writing of logging statements, how to compress logs, how to parse logs into structured event templates, and how to employ logs to detect anomalies, predict failures, and facilitate diagnosis. Additionally, we survey work that releases open-source toolkits and datasets. Based on the discussion of the recent advances, we present several promising future directions toward real-world and next-generation automated log analysis.

CCS Concepts: • **Software and its engineering → Software maintenance tools**; **Software creation and management**.



Very successful research area

Focusing on the ease of practitioners' adoption

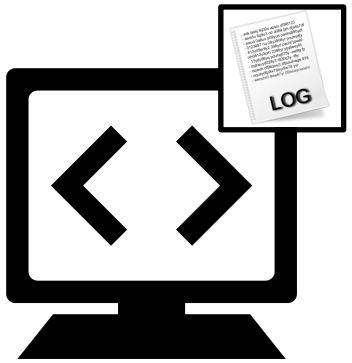


Limited generalized toolsets in practice

Goal: Making log analytics as easy/low-cost as possible

Domain specific language for Log analytics

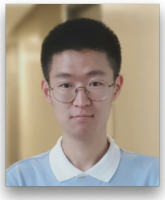
Log analytics as services



Accessible log analytics during development

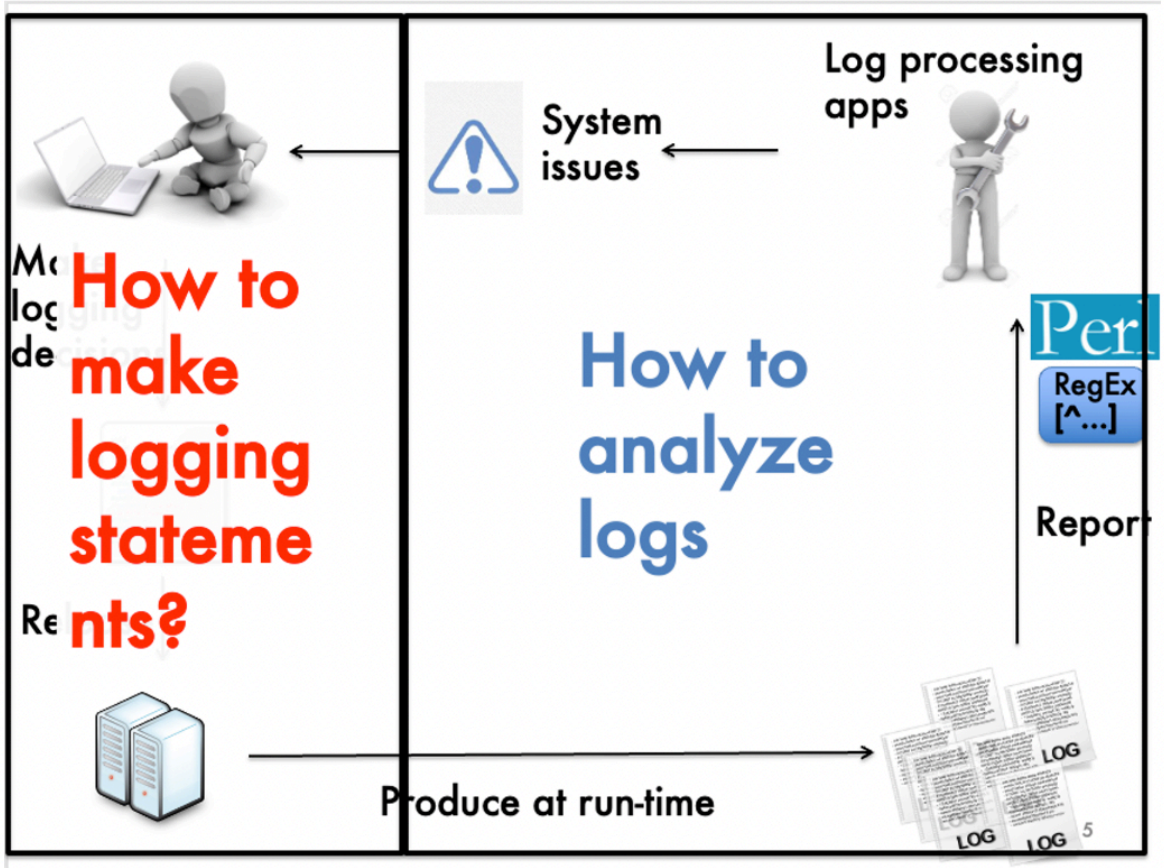
Logging analytics infrastructure

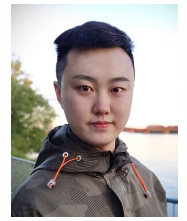
Heng Li



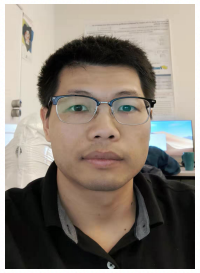
Zhenhao Li

Mehran Hassani





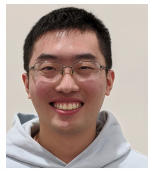
Lizhi Liao



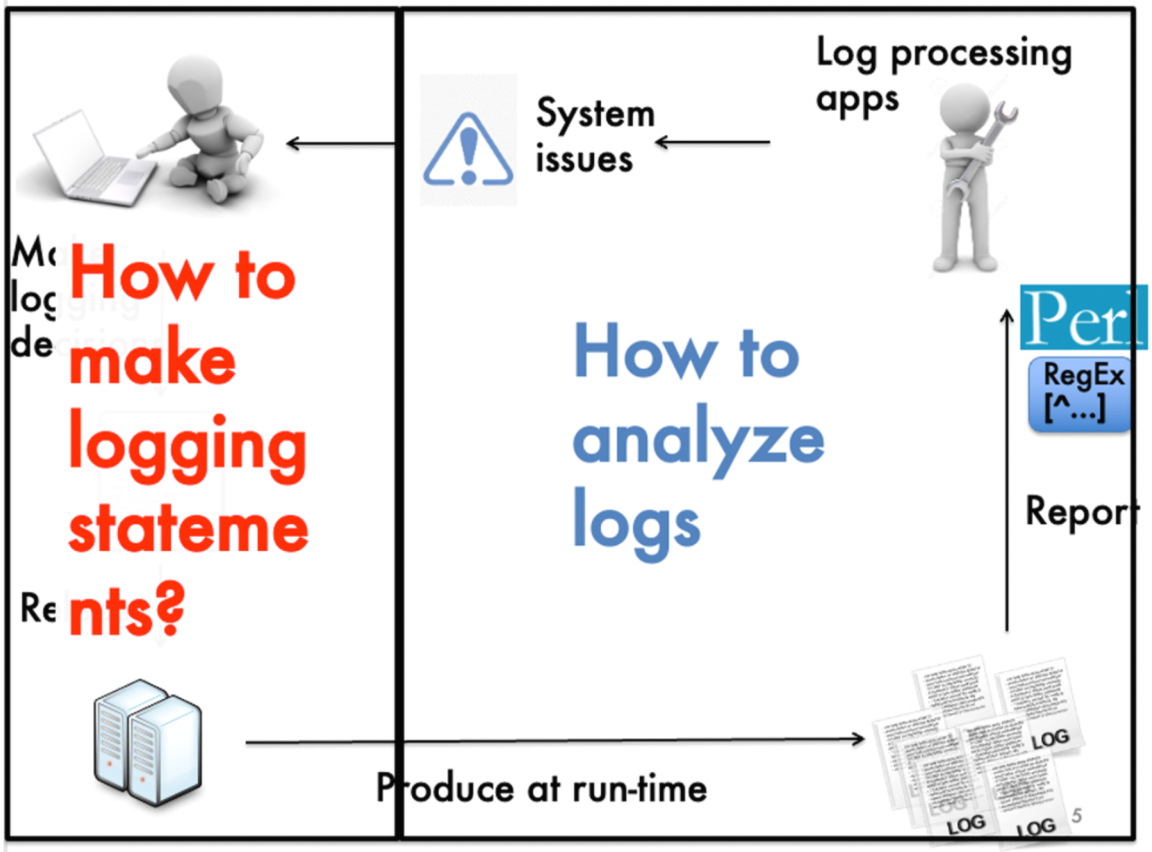
Jinfu Chen

Kundi Yao

Logging analytics infrastructure



Hetong Dai



Logging analytics infrastructure

System issues

Log processing apps

Mak
log
de

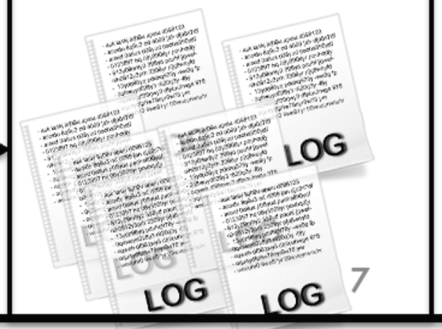
Compression

Raw log (Unstructured)	17/06/09 20:11:11 INFO storage.BlockManager: Found block *rdd_42_20* locally
Parsed log (Structured)	**Timestamp**: 17/06/09 20:11:11; **Level**: INFO **Logger**: storage.BlockManager **Static template**: Found block <*> locally **Dynamic variable(s)**: *rdd_42_20*

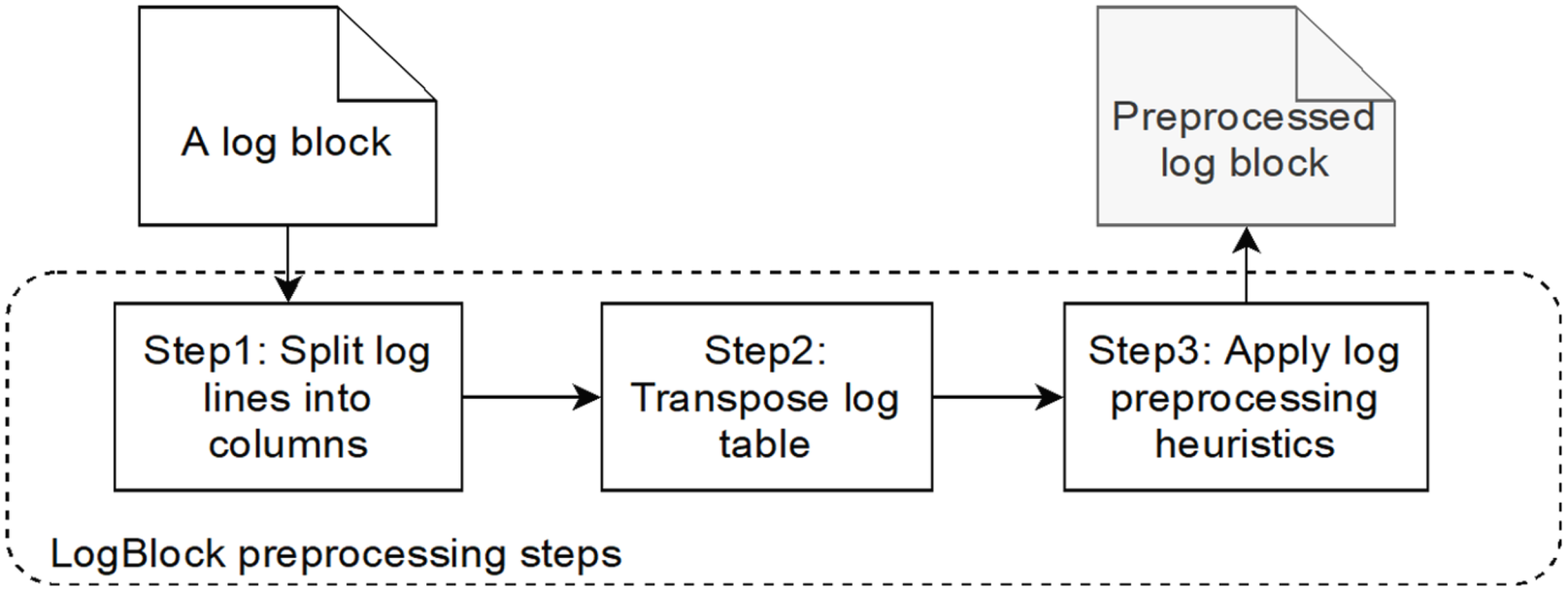
Parsing

Release

Produce at run-time

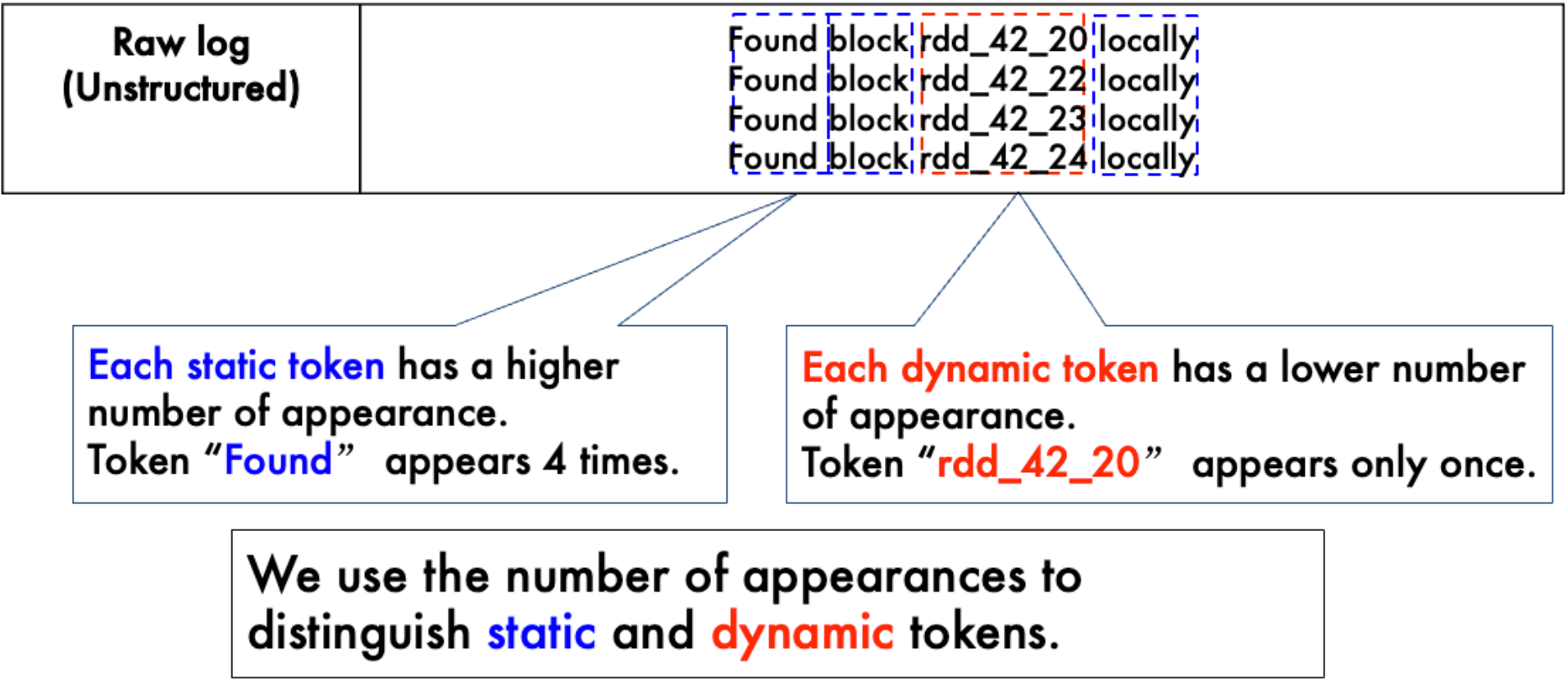


Design of our preprocessing approach: LogBlock

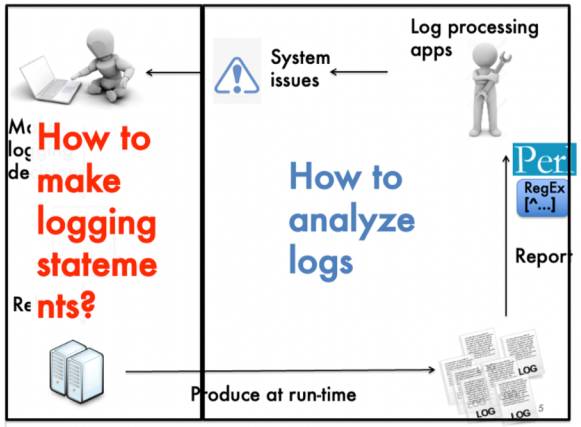


We do not perform extra information reduction steps to log content part for compression performance concern.

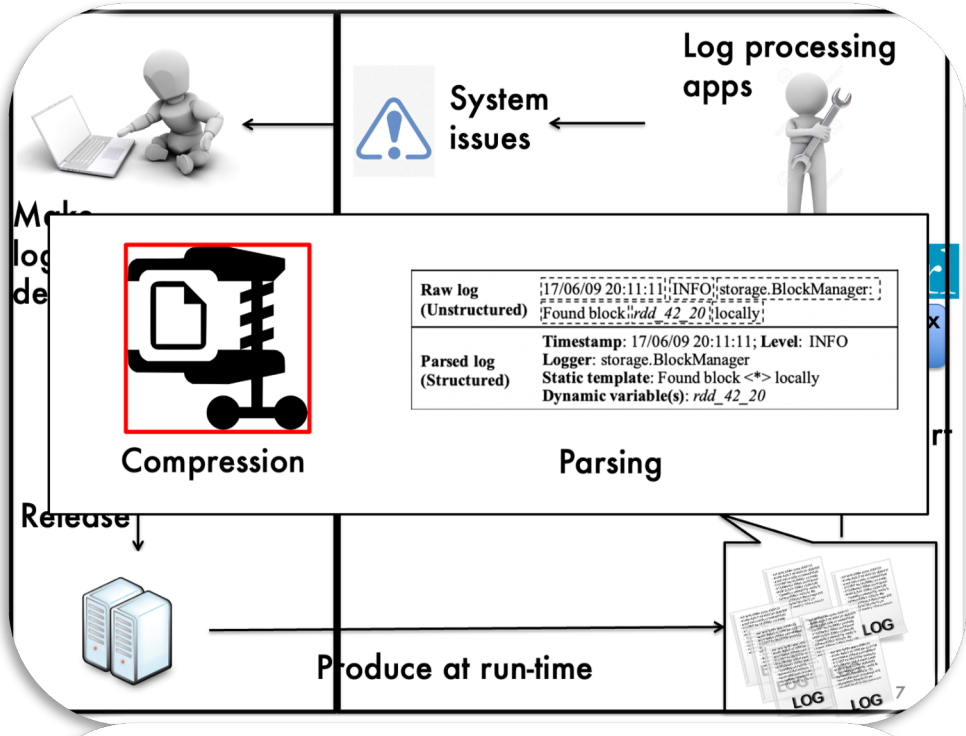
The main idea of Logram



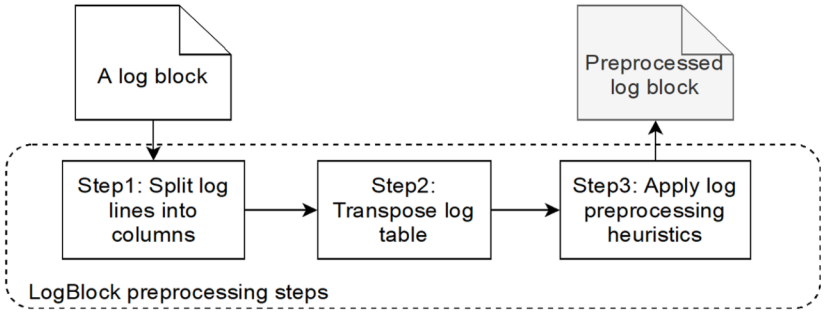
[TSE 2020]



Logging analytics infrastructure



Design of our preprocessing approach: LogBlock



We do not perform extra information reduction steps to log content part for compression performance concern.

The main idea of Logram

Raw log (Unstructured)	Found block rdd_42_20 locally Found block rdd_42_22 locally Found block rdd_42_23 locally Found block rdd_42_24 locally

Each static token has a higher number of appearance.
Token "Found" appears 4 times.

Each dynamic token has a lower number of appearance.
Token "rdd_42_20" appears only once.

We use the number of appearances to distinguish static and dynamic tokens.

2020]

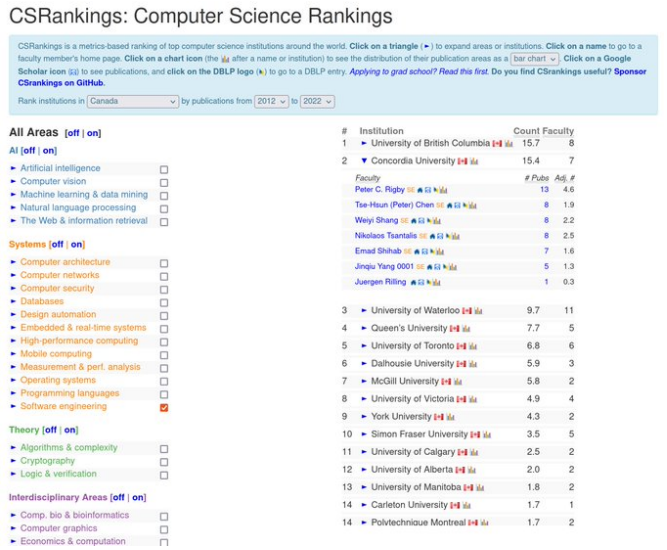
COME and JOIN US







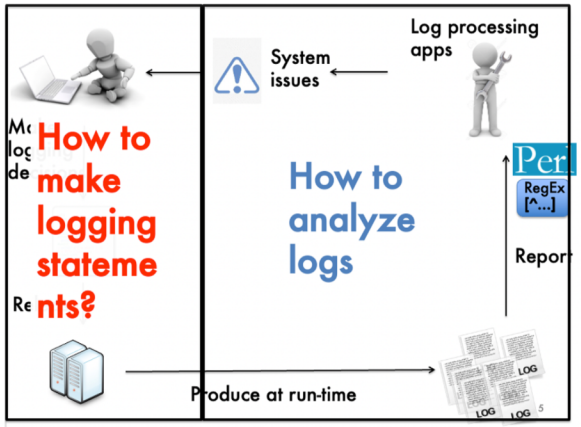


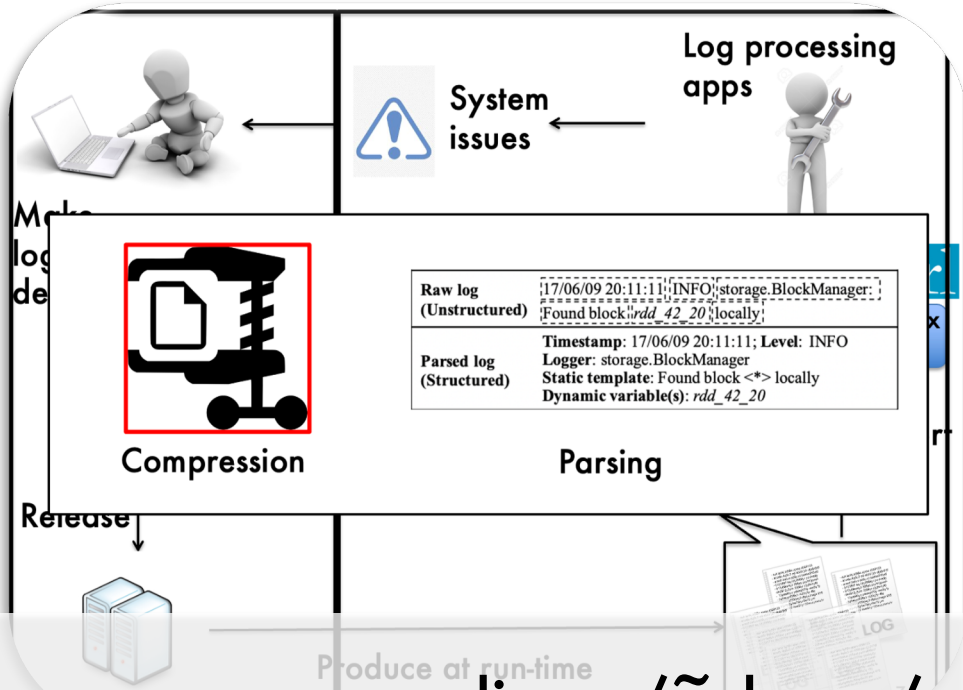


CS ranking on Software Engineering:
2nd in Canada, 6th in North America
and 9th in the world

We are hiring

Master's and PhD with full scholarship
PostDoc with special title
and multiple faculties positions

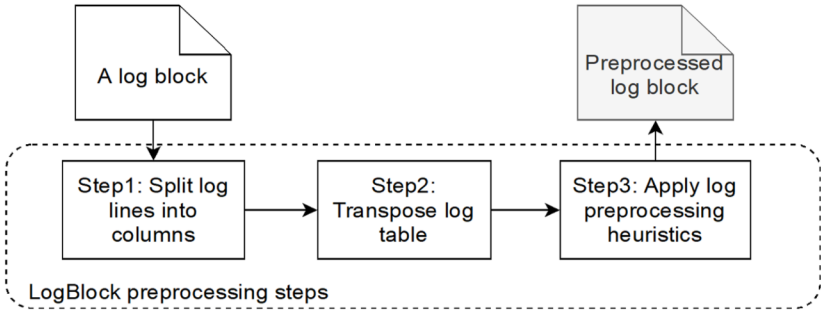




Logging analytics infrastructure

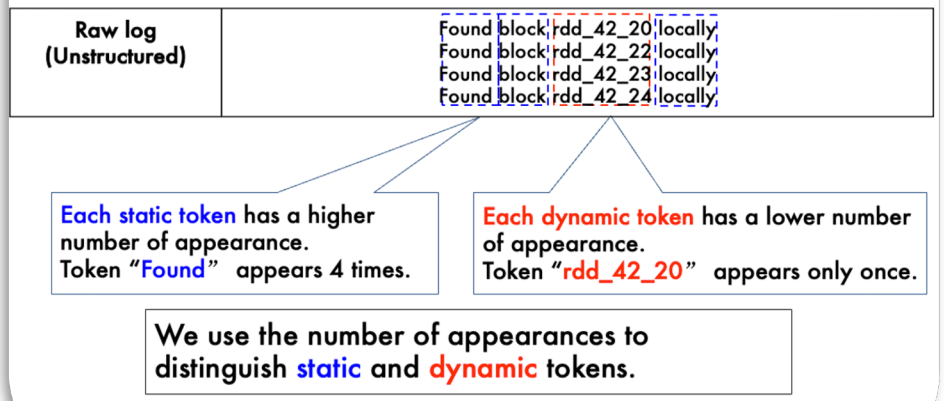
Weiyi Shang http://users.encs.concordia.ca/~shang/

Design of our preprocessing approach: LogBlock



We do not perform extra information reduction steps to log content part for compression performance concern.

The main idea of Logram



2020]