香港中文大學
The Chinese University of Hong Kong

Automated Runtime Data Analysis for System Reliability Management

HE, Pinjia

Supervisor: Prof. Michael R. Lyu

2018/02/08

Modern systems are serving many aspects of our daily life

Popular modern systems

Search engine

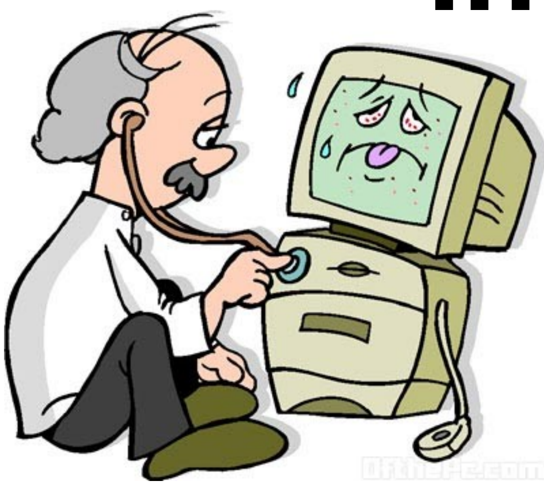
Cloud services

Online chatting

Office software

And many others…

System reliability is very **important**



Real-World Revenue Loss

[Statistics from: https://techcrunch.com/2017/02/28/amazon-aws-s3-outage-is-breaking-things-for-a-lot-of-websites-and-apps/]

Real-World Revenue Loss

Lloyd's Estimates the Impact of a U.S. Cloud Outage at $19 Billion

By: Sean Michael Kerner | January 24, 2018

A joint research report from insurance provider Lloyd's of London and the American Institutes for Research (AIR), looks at the potential costs related to a major public cloud outage in the U.S.

As organizations around the world increasingly rely on the cloud, the impact of a public cloud failure is something that insurance companies are now concerned about. A 67-page report released on Jan. 23 from Lloyd's of London and AIR Worldwide provides some insight and estimates on the potential losses from a major cloud services outage—and the numbers are large.

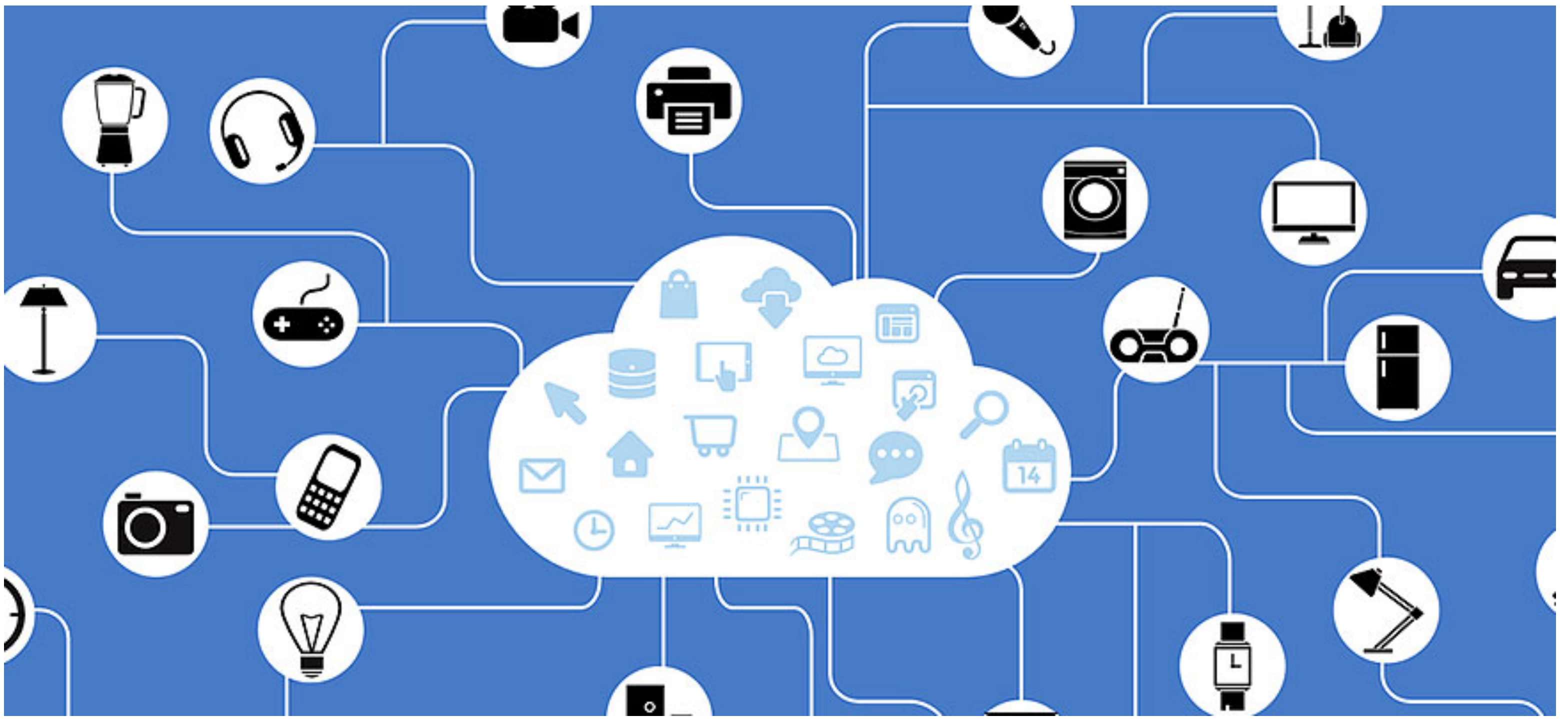
According to the report, a cyber-incident that impacted the operations of one of the top three public cloud providers in the U.S. for three to six days, could result in total losses of up to $19 billion. Of those loses, only $1.1 to $3.5 billion would be insured, leaving organizations left to cover the rest of the costs.

[Source: http://www.eweek.com/cloud/lloyd-s-estimates-the-impact-of-a-u.s.-cloud-outage-at-19-billion]

Reliability management of modern systems **is important,**

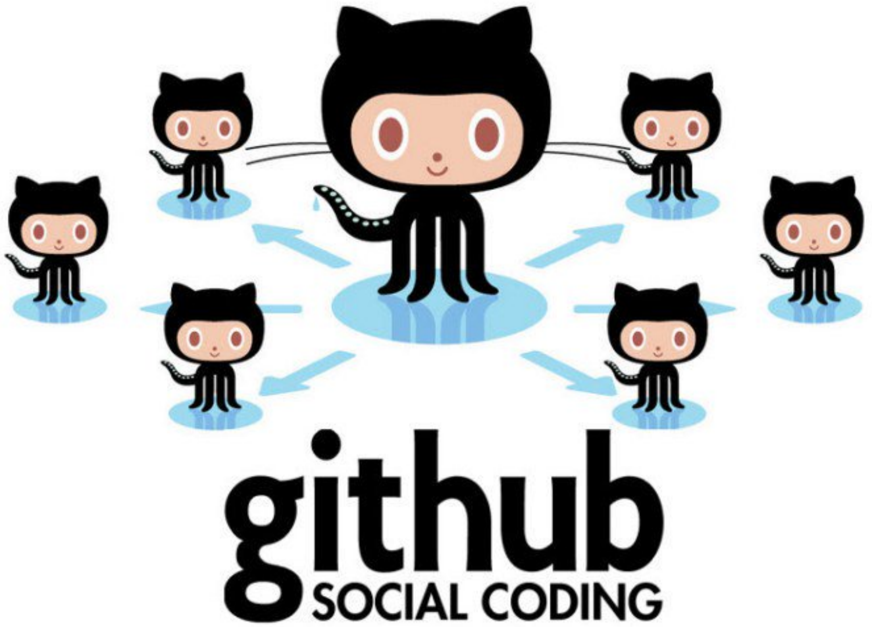
but challenging

Modern systems are becoming **large-scale in size**



[Image from: http://www.lancaster.ac.uk/scc/research/distributed-systems/]

Modern systems are **complex in structure**

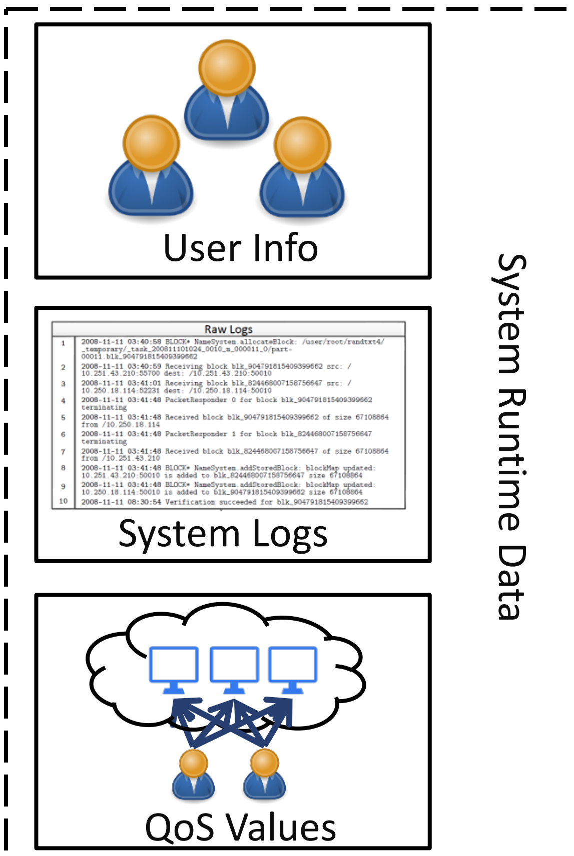


[Image from: http://www.smashingbuzz.com/2015/01/ultimate-github-features]

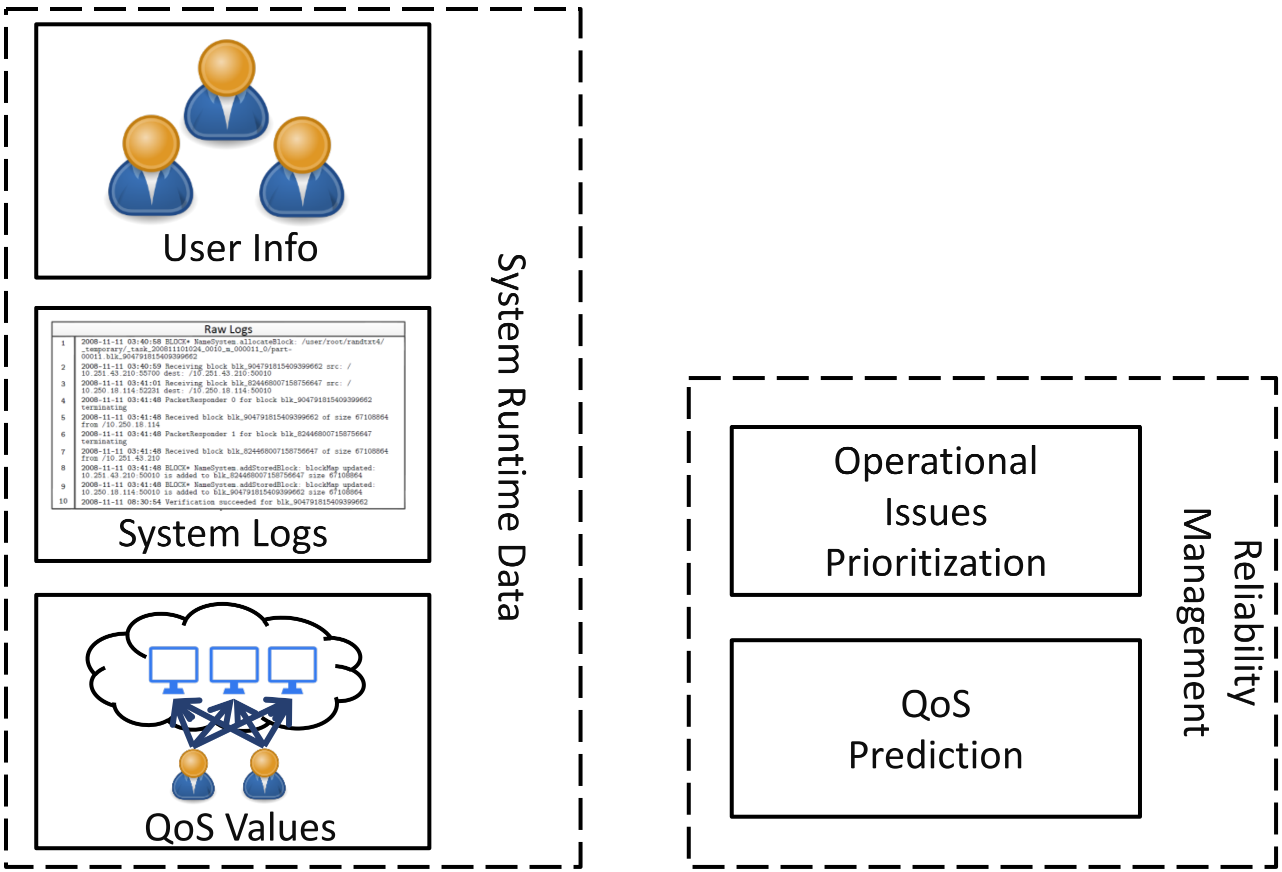
Traditional engineering techniques
are often not sufficient

Automated runtime data analysis is in need

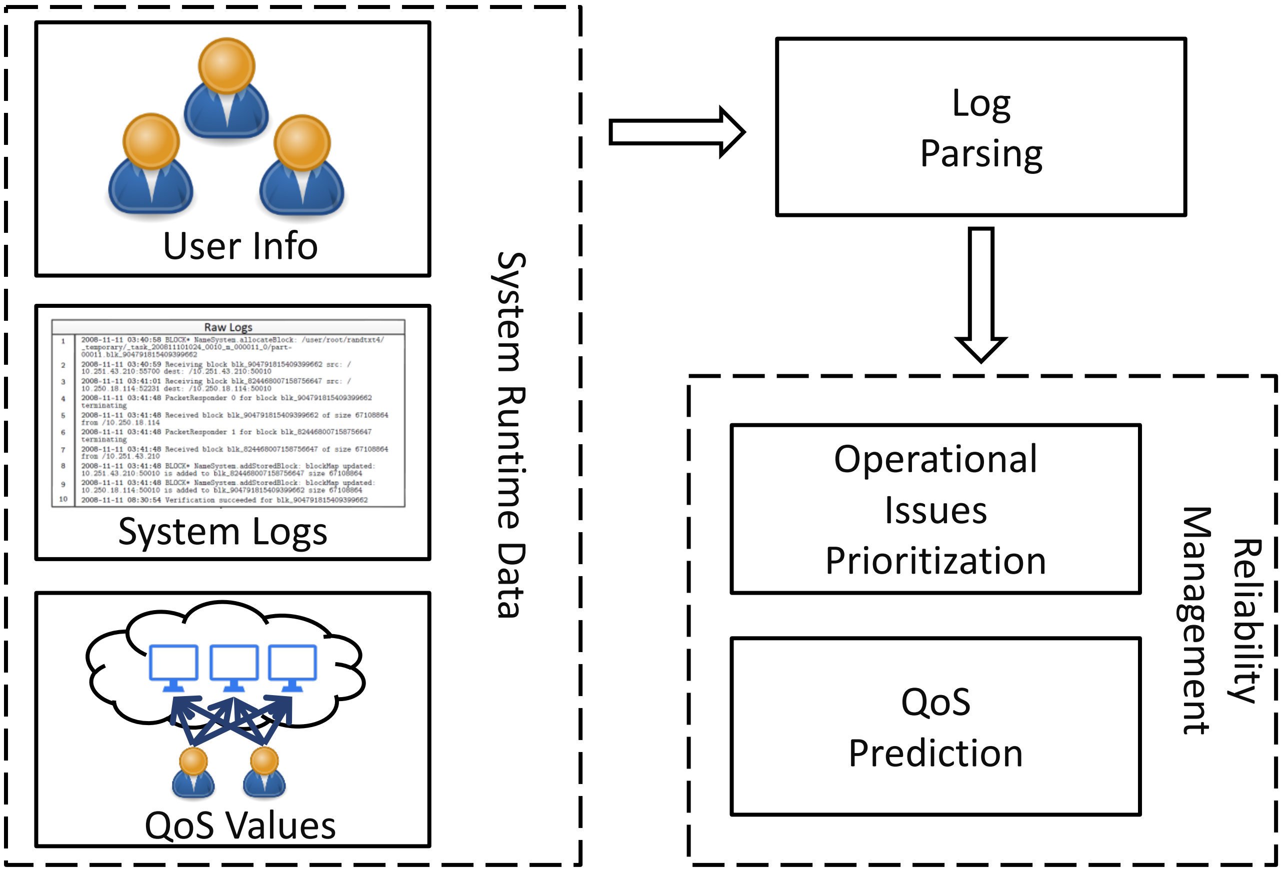
Automated runtime data analysis
for system reliability management



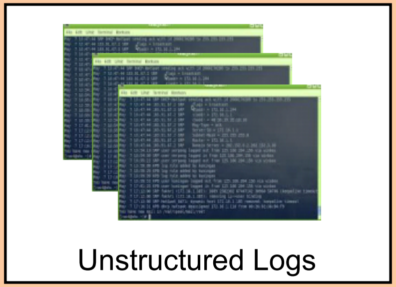
Automated runtime data analysis
for system reliability management



Automated runtime data analysis
for system reliability management



Thesis contributions

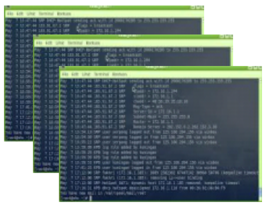


Evaluation study on log parsing [DSN’16] (**Chapter 3**)

Parallel log parsing [TDSC’17] (**Chapter 4**)

Online log parsing [ICWS’17] (**Chapter 5**)

Thesis contributions



Unstructured Logs

Evaluation study on log parsing [DSN'16] (**Chapter 3**)

- Reviews and evaluate four representative log parsers
- A case study of the effectiveness of log parsers on log mining
- Six findings and open-source toolkit

Parallel log parsing [TDSC'17] (**Chapter 4**)

- The first parallel log parsing framework
- Specially-designed heuristic rules and clustering algorithm
- Evaluate on real-world data and large-scale synthetic data
- Open-source

Online log parsing [ICWS'17] (**Chapter 5**)

- Online parser Drain via fixed depth tree
- 51.85% ~ 81.47% efficiency improvement with comparable accuracy
- Open-source

Thesis contributions

1 : (Event 1, Event 2, Event 3)
2 : (Event 4, Event 1, Event 3)
3 : (Event 1, Event 2)
4 : (Event 3, Event 3, Event 6)
……
Log Event Sequences

→ **Operational issues prioritization** (**Chapter 6**)

– An operational issue prioritization framework POI
– Coarse-grained clustering and fine-grained clustering
– Novel weighting method Inverse Cardinality (IC)

QoS of service 1: (q_{11}, q_{12},…, q_{1m})
QoS of service 2: (q_{21}, q_{22}, …, q_{2m})
QoS of service 3: (q_{31}, q_{32}, …, q_{3m})
QoS of service 4: (q_{41}, q_{42}, …, q_{4m})
……
QoS Values

→ **QoS prediction** [ICWS'14] (**Chapter 7**)

– Hierarchical matrix factorization model
– Location of both users and services

Outline

- **Topic 1: Evaluation study** on log parsing
- **Topic 2: Parallel log parsing** for large-scale log data
- **Topic 3: Online log parsing** via fixed depth tree
- **Conclusion and future work**

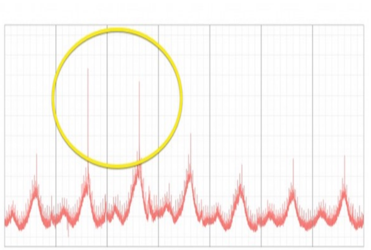
Outline

- **Topic 1: Evaluation study** on log parsing
- **Topic 2: Parallel log parsing** for large-scale log data
- **Topic 3: Online log parsing** via fixed depth tree
- Conclusion and future work

Logs are widely-employed to enhance the system reliability by

log analysis

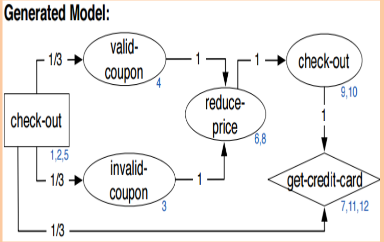
Log Analysis



Anomaly Detection

Detecting largescale system problems by mining console logs [SOSP'09]

Log Clustering based Problem Identification for Online Service Systems [ICSE'16]



Program Verification

Leveraging existing instrumentation to automatically infer invariant-constrained models [FSE'11]

Assisting developers of big data analytics applications when deploying on hadoop clouds [ICSE'13]

Performance Monitoring

Structured comparative analysis of systems logs to diagnose performance problems [NSDI'12]

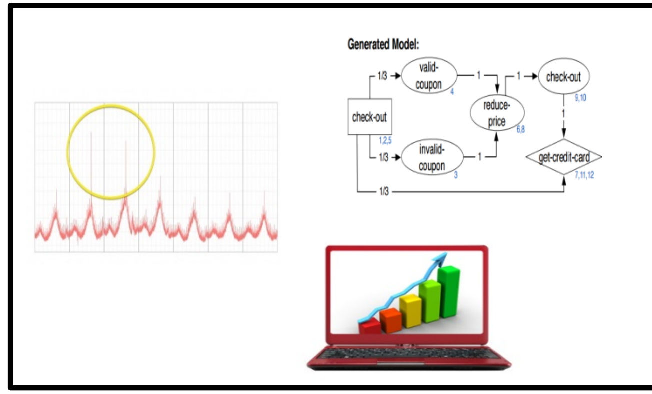
Be conservative: enhancing failure diagnosis with proactive logging [OSDI'12]

Log Analysis contains two steps:

Log Parsing and **Log Mining**

Log Analysis: log parsing & log mining

	Raw Log Messages
1	**2008-11-11 03:40:58** BLOCK* NameSystem.allocateBlock: /user/root/randtxt4/_temporary/_task_200811101024_0010_m_000011_0/part-00011.blk_904791815409399662
2	**2008-11-11 03:40:59** Receiving block blk_904791815409399662 src: /10.251.43.210:55700 dest: /10.251.43.210:50010
3	**2008-11-11 03:41:01** Receiving block blk_904791815409399662 src: /10.250.18.114:52231 dest: /10.250.18.114:50010
4	**2008-11-11 03:41:48** PacketResponder 0 for block blk_904791815409399662 terminating
5	**2008-11-11 03:41:48** Received block blk_904791815409399662 of size 67108864 from /10.250.18.114
6	**2008-11-11 03:41:48** PacketResponder 1 for block blk_904791815409399662 terminating
7	**2008-11-11 03:41:48** Received block blk_904791815409399662 of size 67108864 from /10.251.43.210
8	**2008-11-11 03:41:48** BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.43.210:50010 is added to blk_904791815409399662 size 67108864
9	**2008-11-11 03:41:48** BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.18.114:50010 is added to blk_904791815409399662 size 67108864
10	**2008-11-11 08:30:54** Verification succeeded for blk_904791815409399662



Log Mining

Log Parsing

Log Events	
Event1	BLOCK* NameSystem.allocateBlock: *
Event2	Receiving block * src: * dest: *
Event3	PacketResponder * for block * terminating
Event4	Received block * of size * from *
Event5	BLOCK* NameSystem.addStoredBlock: blockMap updated: * is added to * size *
Event6	Verification succeeded for *

Strutured Logs		
1	blk_904791815409399662	Event1
2	blk_904791815409399662	Event2
3	blk_904791815409399662	Event2
4	blk_904791815409399662	Event3
5	blk_904791815409399662	Event4
6	blk_904791815409399662	Event3
7	blk_904791815409399662	Event4
8	blk_904791815409399662	Event5
9	blk_904791815409399662	Event5
10	blk_904791815409399662	Event6

Log Parsing Example

Raw Log

2008-11-11 03:41:48 Received block blk_90 src: /10.251.30.6 dest: /10.251.30.6: of size 67108864

```
LOG.info("Received block " + block +
        " src: " + remoteAddress +
        " dest: " + localAddress +
        " of size " + block.getNumBytes());
```

Log Parsing

Structured Log

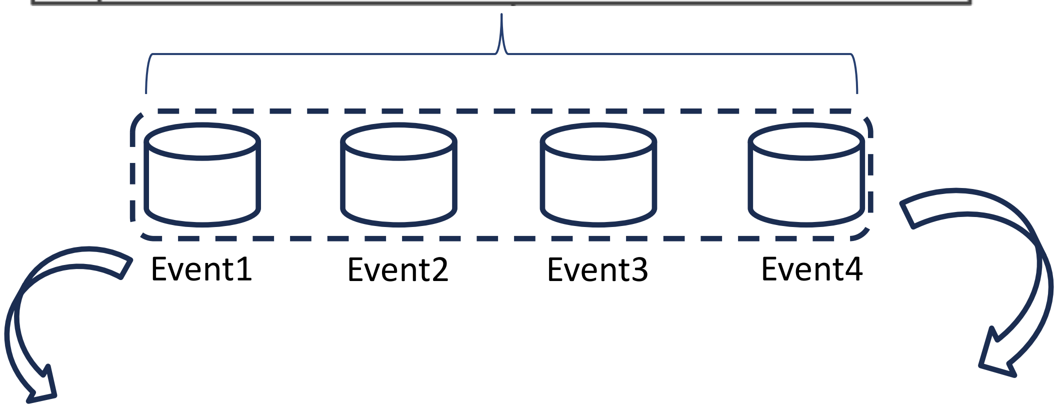
blk_90 -> Received block * src: * dest: * of size *

The goal of log parsing is to distinguish between constant part and variable part from the log contents.

Log ... ring

Raw Log Messages

1	**2008-11-11 03:40:58** BLOCK* NameSystem.allocateBlock: /user/root/randtxt4/_temporary/_task_200811101024_0010_m_000011_0/part-00011.blk_904791815409399662
2	**2008-11-11 03:40:59** Receiving block blk_904791815409399662 src: /10.251.43.210:55700 dest: /10.251.43.210:50010
3	**2008-11-11 03:41:01** Receiving block blk_904791815409399662 src: /10.250.18.114:52231 dest: /10.250.18.114:50010
4	**2008-11-11 03:41:48** PacketResponder 0 for block blk_904791815409399662 terminating
5	**2008-11-11 03:41:48** Received block blk_904791815409399662 of size 67108864 from /10.250.18.114
6	**2008-11-11 03:41:48** PacketResponder 1 for block blk_904791815409399662 terminating
7	**2008-11-11 03:41:48** Received block blk_904791815409399662 of size 67108864 from /10.251.43.210
8	**2008-11-11 03:41:48** BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.43.210:50010 is added to blk_904791815409399662 size 67108864
9	**2008-11-11 03:41:48** BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.18.114:50010 is added to blk_904791815409399662 size 67108864
10	**2008-11-11 08:30:54** Verification succeeded for blk_904791815409399662



Log Events

Event1	BLOCK* NameSystem.allocateBlock: *
Event2	Receiving block * src: * dest: *
Event3	PacketResponder * for block * terminating
Event4	Received block * of size * from *
Event5	BLOCK* NameSystem.addStoredBlock: blockMap updated: * is added to * size *
Event6	Verification succeeded for *

Strutured Logs

1	blk_904791815409399662	Event1
2	blk_904791815409399662	Event2
3	blk_904791815409399662	Event2
4	blk_904791815409399662	Event3
5	blk_904791815409399662	Event4
6	blk_904791815409399662	Event3
7	blk_904791815409399662	Event4
8	blk_904791815409399662	Event5
9	blk_904791815409399662	Event5
10	blk_904791815409399662	Event6

Manual maintenance of log event is difficult, even with the help of regular expression

- **The volume of log is growing rapidly.** (e.g., 50 GB/h [Mi TPDS'13])

- **Developer may not understand the logging purpose.** (open source components [Xu SOSP'09])

- **Log statements in modern systems update frequently.** (e.g., hundreds of new statements a month [Xu PhD Thesis'10])

Automated log parsing

is highly in demand

State-of-the-art Log Parsing Methods

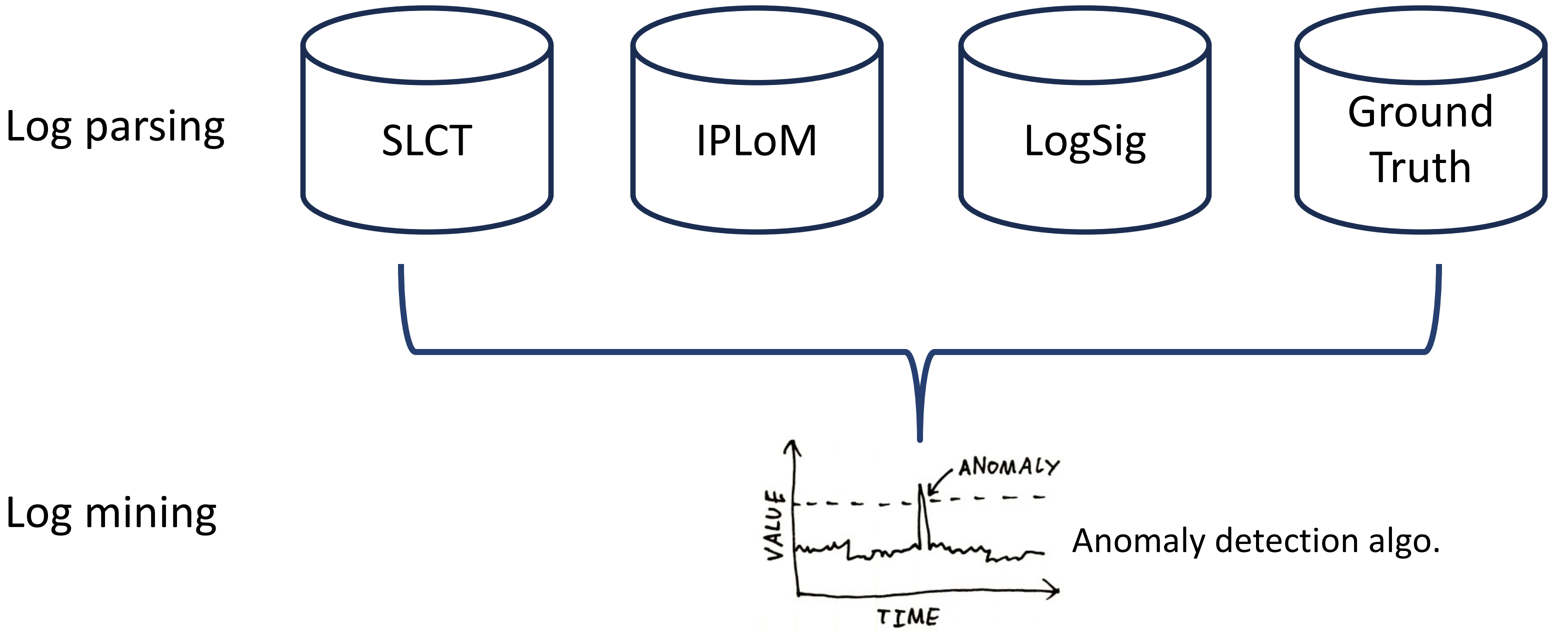
- **SLCT:** Simple Logfile Clustering To

Heuristic Rules

- **IPLoM:** Iterative Partitioning Log Mining [KDD’09, TKDE’12]

Clustering Algorithms

- **LKE:** Log Key Extraction [ICDM’09
- **LogSig:** Log Signature Extraction [CIKM’11]



Will the performance of log parsers affect the anomaly detection results?

- Case study on real-world anomaly detection task [SOSP'09]

- 11,175,629 HDFS logs
- 575,061 HDFS blocks
- **16,838** anomalies

Accuracy Metric

- Parsing accuracy: **F-measure** of clustering algorithm

- **F-measure** = 2 * Precision * Recall / (Precision + Recall)

- Precision = TP/(TP+FP)   Recall = TP/(TP+FN)

- TP: assigns two logs with the same log event to the same cluster
- TN: assigns two logs with different log events to different clusters
- FP: assigns two logs with different log events to the same cluster

	Parsing Accuracy	Reported Anomaly	Detected Anomaly	False Alarm
SLCT	0.83	18,450	10,935 (64%)	7,515 (40%)
LogSig	0.87	11,091	10,678 (63%)	413 (3.7%)
IPLoM	0.99	10,998	10,720 (63%)	278 (2.5%)
Ground truth	1.00	11,473	11,195 (66%)	278 (2.4%)

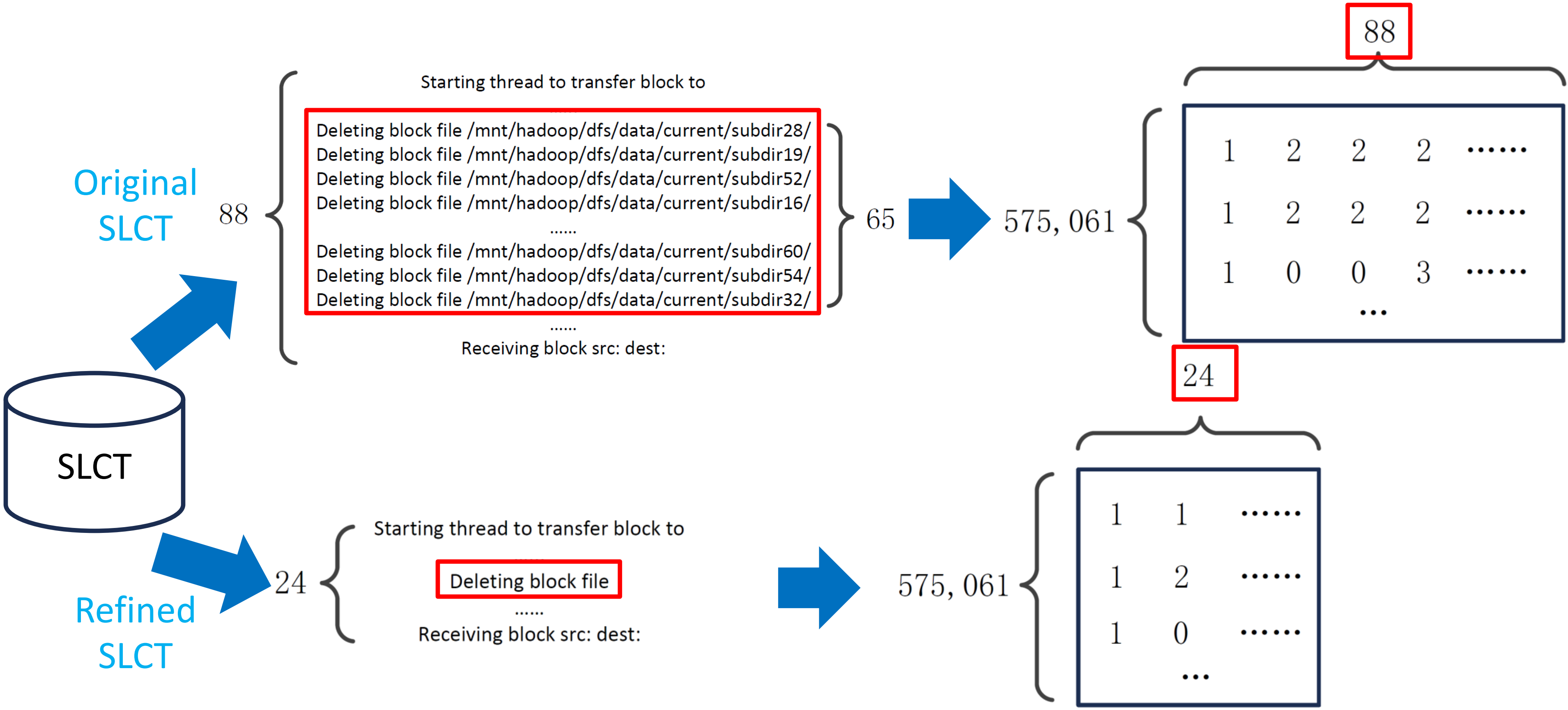
- **Parsing Accuracy**: F-measure\
- **Report Anomaly**: #anomalies reported
- **Detected Anomaly**: #true anomalies detected
- **False Alarm**: #wrongly detected anomalies

	Parsing Accuracy	Reported Anomaly	Detected Anomaly	False Alarm
SLCT	0.83	18,450	10,935 (64%)	7,515 (40%)
LogSig	0.87	11,091	10,678 (63%)	413 (3.7%)
IPLoM	0.99	10,998	10,720 (63%)	278 (2.5%)
Ground truth	1.00	11,473	11,195 (66%)	278 (2.4%)

Finding: Log parsing is important because log mining is effective only when the parsing accuracy is high enough.

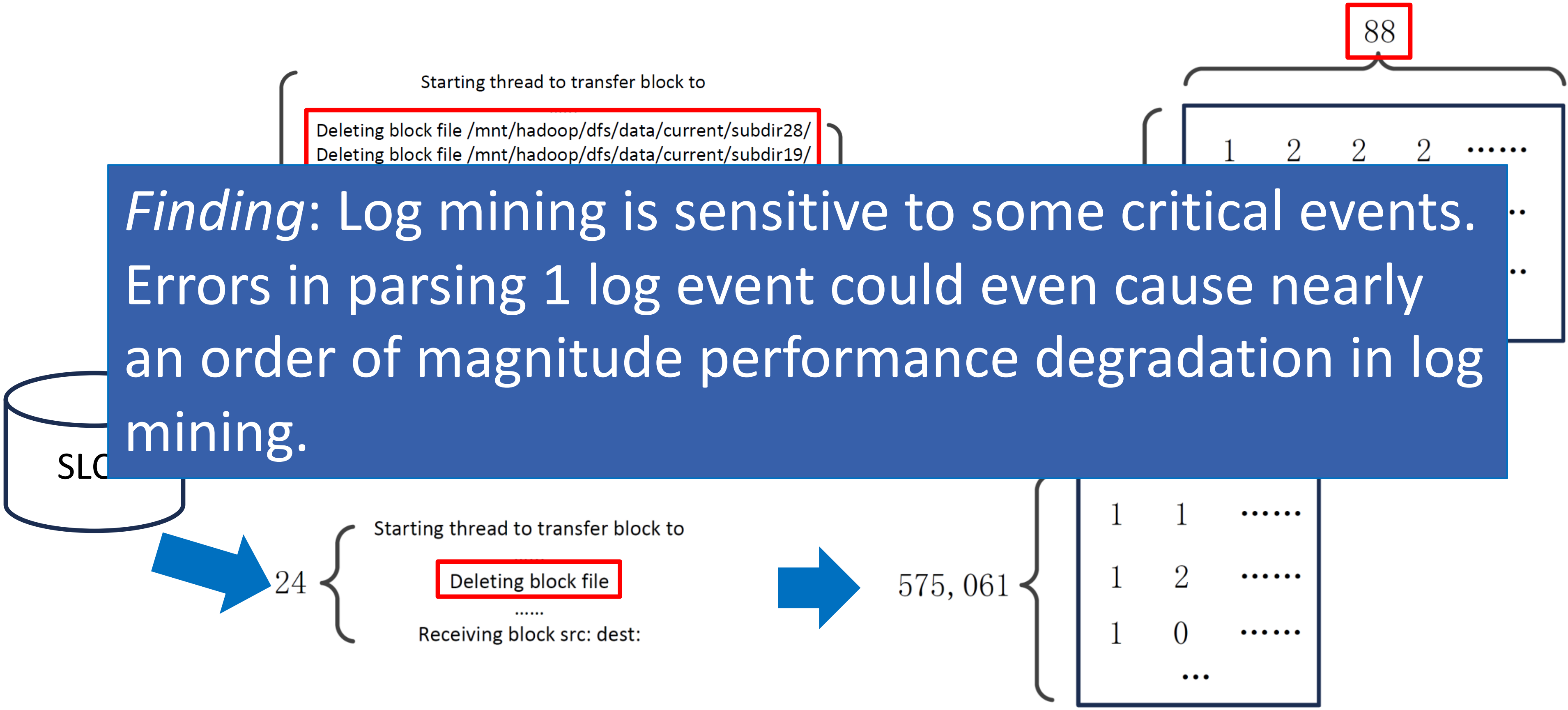
	Parsing Accuracy	Reported Anomaly	Detected Anomaly	False Alarm
SLCT	0.83	18,450	10,935 (64%)	7,515 (40%)
LogSig	0.87	11,091	10,678 (63%)	413 (3.7%)
IPLoM	0.99	10,998	10,720 (63%)	278 (2.5%)
Ground truth	1.00	11,473	11,195 (66%)	278 (2.4%)

	Parsing Accuracy	Reported Anomaly	Detected Anomaly	False Alarm
SLCT	0.83	18,450	10,935 (64%)	7,515 (40%)
LogSig	0.87	11,091	10,678 (63%)	413 (3.7%)



Parsing Accuracy	Reported Anomaly	Detected Anomaly	False Alarm
0.91	11,539	10,746 (64%)	793 (6.8%)

	Parsing Accuracy	Reported Anomaly	Detected Anomaly	False Alarm
SLCT	0.83	18,450	10,935 (64%)	7,515 (40%)
LogSig	0.87	11,091	10,678 (63%)	413 (3.7%)



Finding: Log mining is sensitive to some critical events. Errors in parsing 1 log event could even cause nearly an order of magnitude performance degradation in log mining.

Parsing Accuracy	Reported Anomaly	Detected Anomaly	False Alarm
0.91	11,539	10,746 (64%)	793 (6.8%)

Outline

- **Topic 1: Evaluation study** on log parsing
- **Topic 2: Parallel log parsing** for large-scale log data
- **Topic 3: Online log parsing** via fixed depth tree
- Conclusion and future work

Why we need parallel log parsers?

Motivations & Contributions

Weakness of existing parsers

- Existing log parsers do **not consistently** obtain **high accuracy** on all datasets.

- When logs grow to a **large scale**, existing parsers **fail to complete** in reasonable time.

Motivations & Contributions

Weakness of existing parsers

Parallel log parser: POP

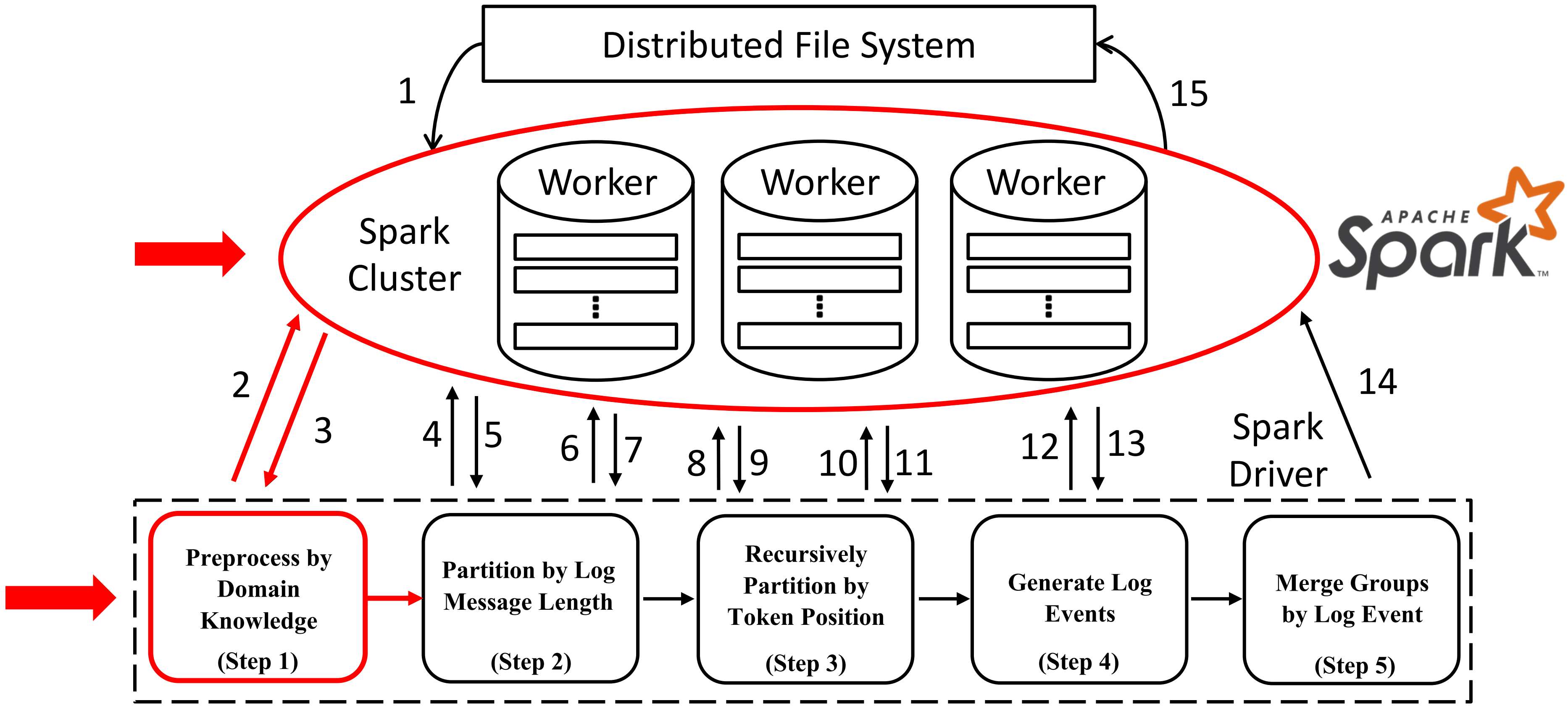
- Existing log parsers do **not consistently** obtain **high accuracy** on all datasets.

POP achieves the highest parsing accuracy on all datasets.

- When logs grow to a **large scale**, existing parsers **fail to complete** in reasonable time.

POP can handle 200m HDFS logs in 7 mins, while the state-of-the-art needs more than half an hour.

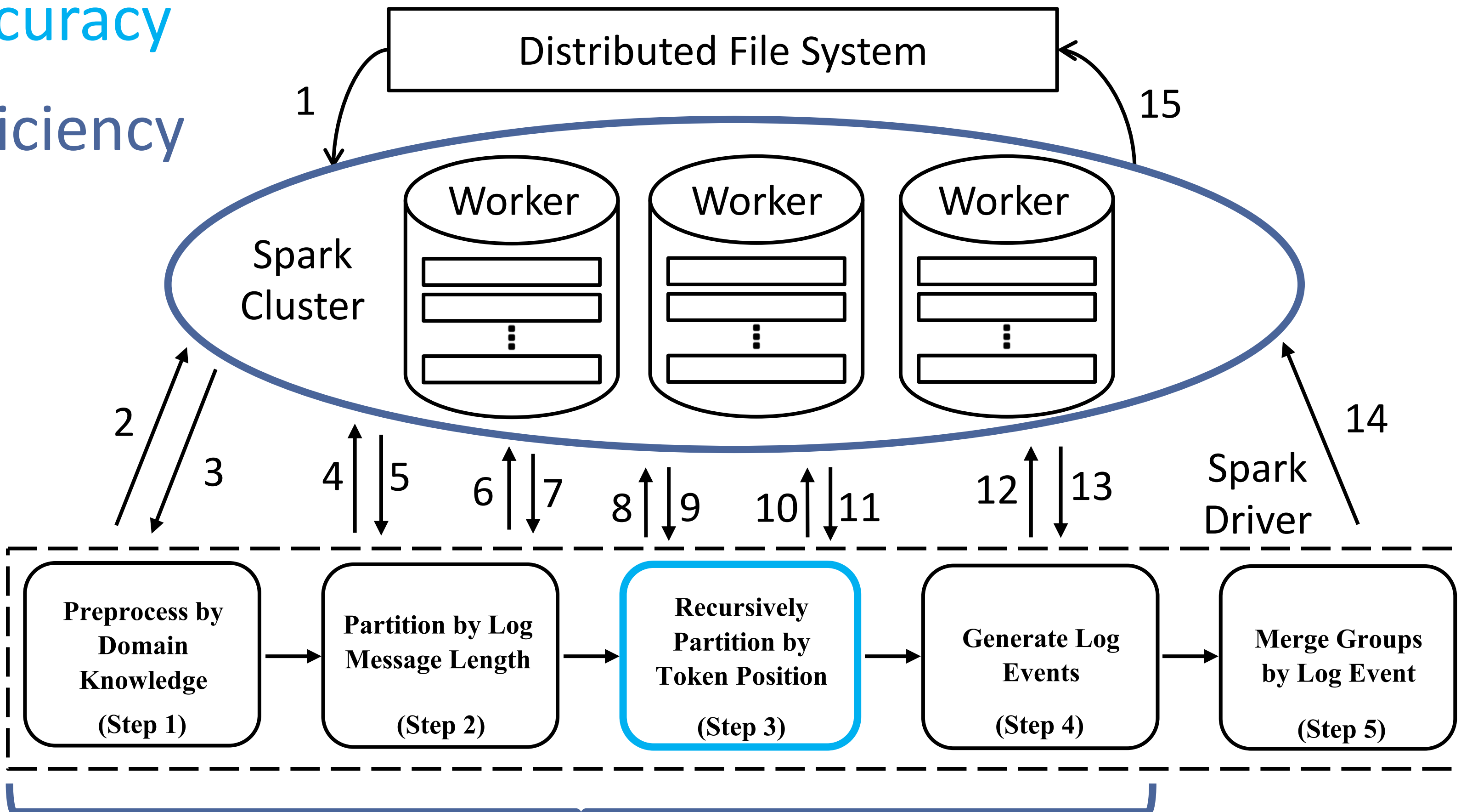
Framework of POP



Novelty of POP

Accuracy

Efficiency



Step 1: Preprocess by domain knowledge

- **Prune** variable parts according to simple regular expressions

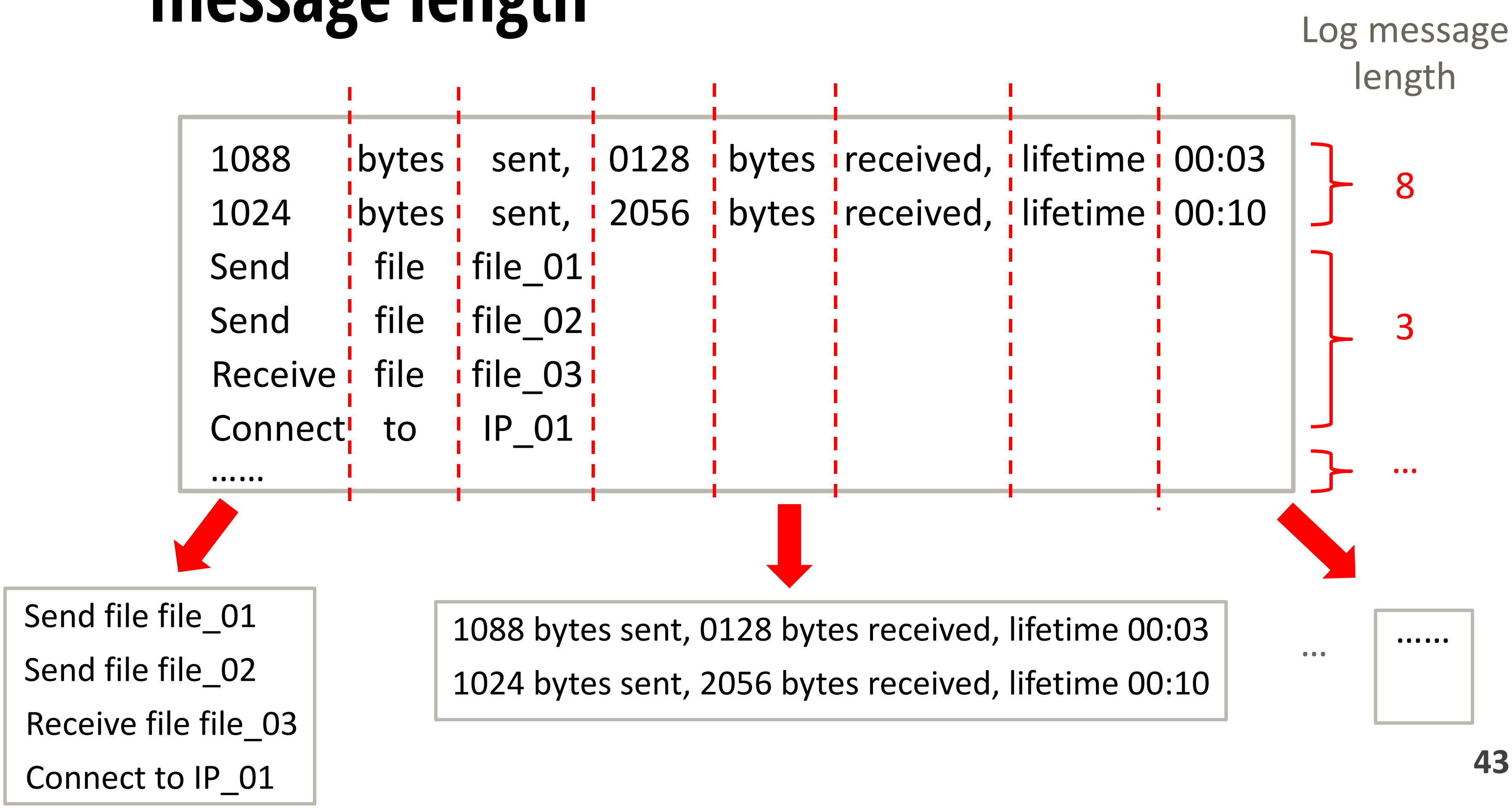
blk_[0-9]+

Received block blk_904791815409399662 of size 67108864 from /10.251.43.210

Received block of size 67108864 from /10.251.43.210

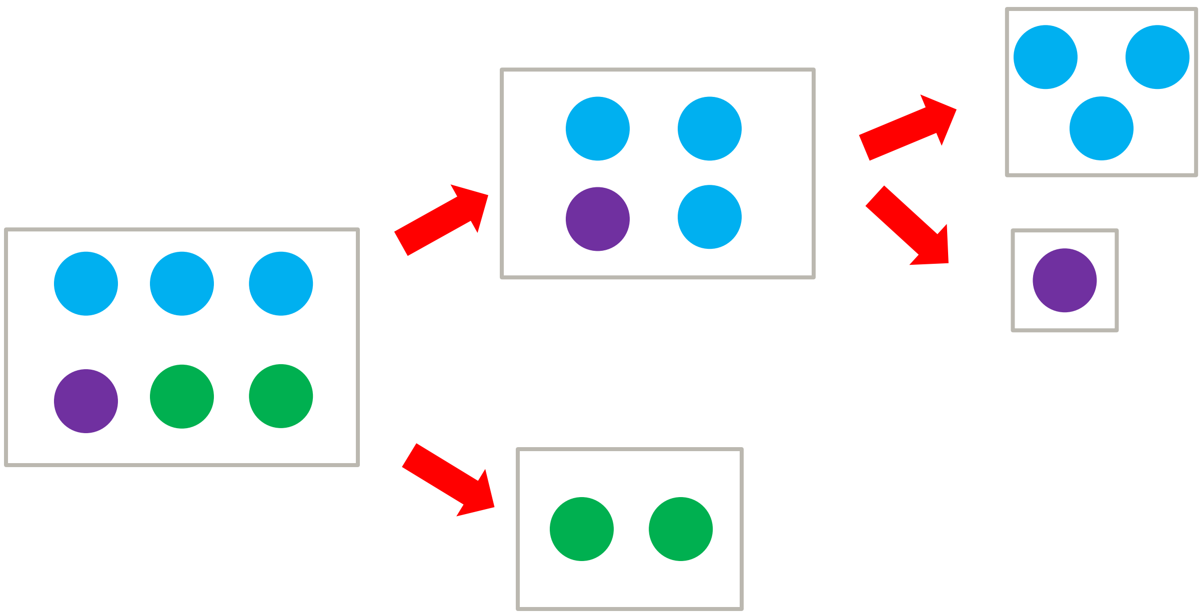
Step 2: Partition by log message length

- Partition logs into different groups based on **log message length**



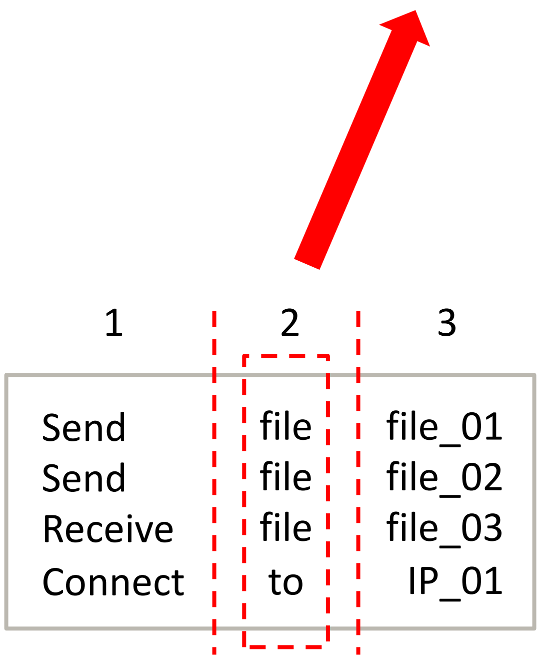
Step 3: Recursively partition by token position

- Find **split token position**
- **Recursively** partition a log group
- Stop until all log groups are **complete groups**



Step 3: Recursively partition by token position

- Find **split token position**

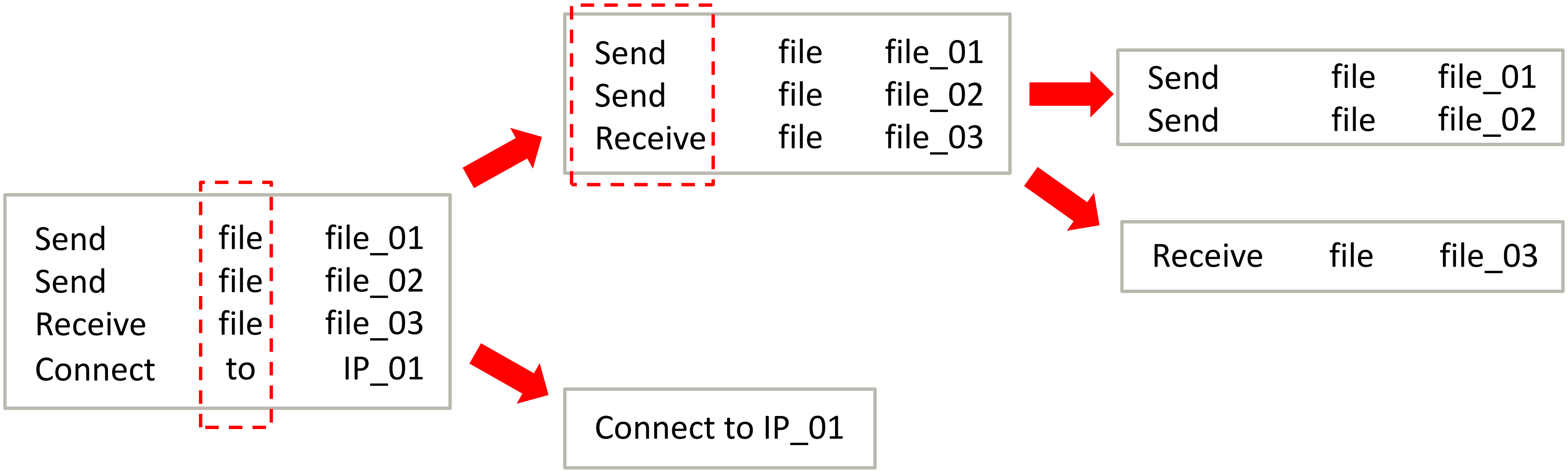


1: {Send, Receive, Connect}
2: {file, to}
3: {file_01, file_02, file_03, IP_01}

1: 3 distinct tokens
2: 2 distinct tokens
3: 4 distinct tokens

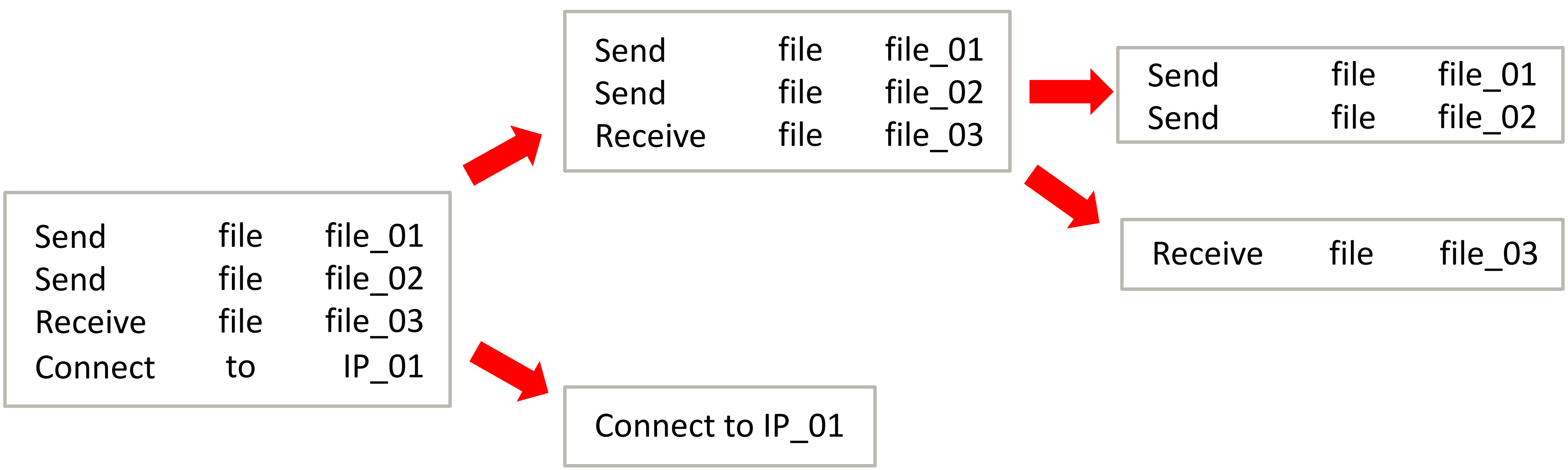
Step 3: Recursively partition by token position

- Find **split token position**
- **Recursively** partition a log group



Step 3: Recursively partition by token position

- Find **split token position**
- **Recursively** partition a log group
- Stop until all log groups are **complete groups**



Step 3: Recursively partition by token position

- Stop until all log groups are **complete groups**

$$\text{Group goodness} = \frac{\#token\ position\ with\ one\ distinct\ token}{log\ message\ length}$$

file
file
file

Send	file	file_01
Send	file	file_02
Receive	file	file_03

$\frac{1}{3} \approx 0.33$

Send	file	file_01
Send	file	file_02

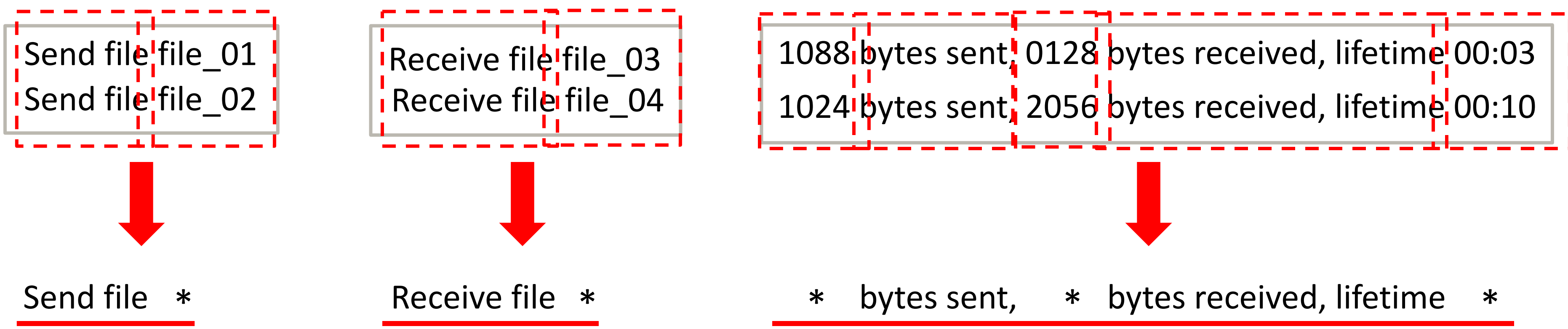
$\frac{2}{3} \approx 0.66$

Complete group

- Compare with a group threshold **gs** (e.g., 0.5)

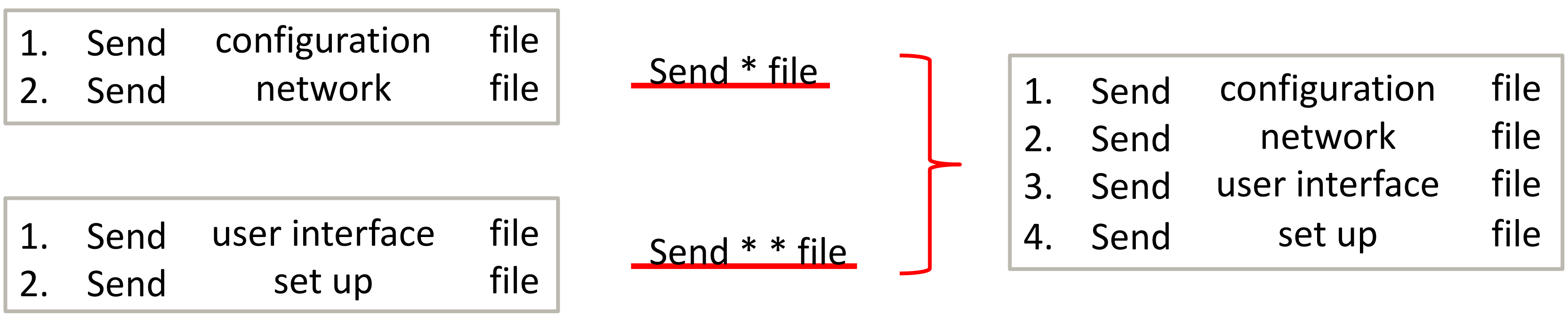
Step 4: Generate log events

- Inspect the tokens in each **token position** of each log, calculate the number of **distinct tokens**



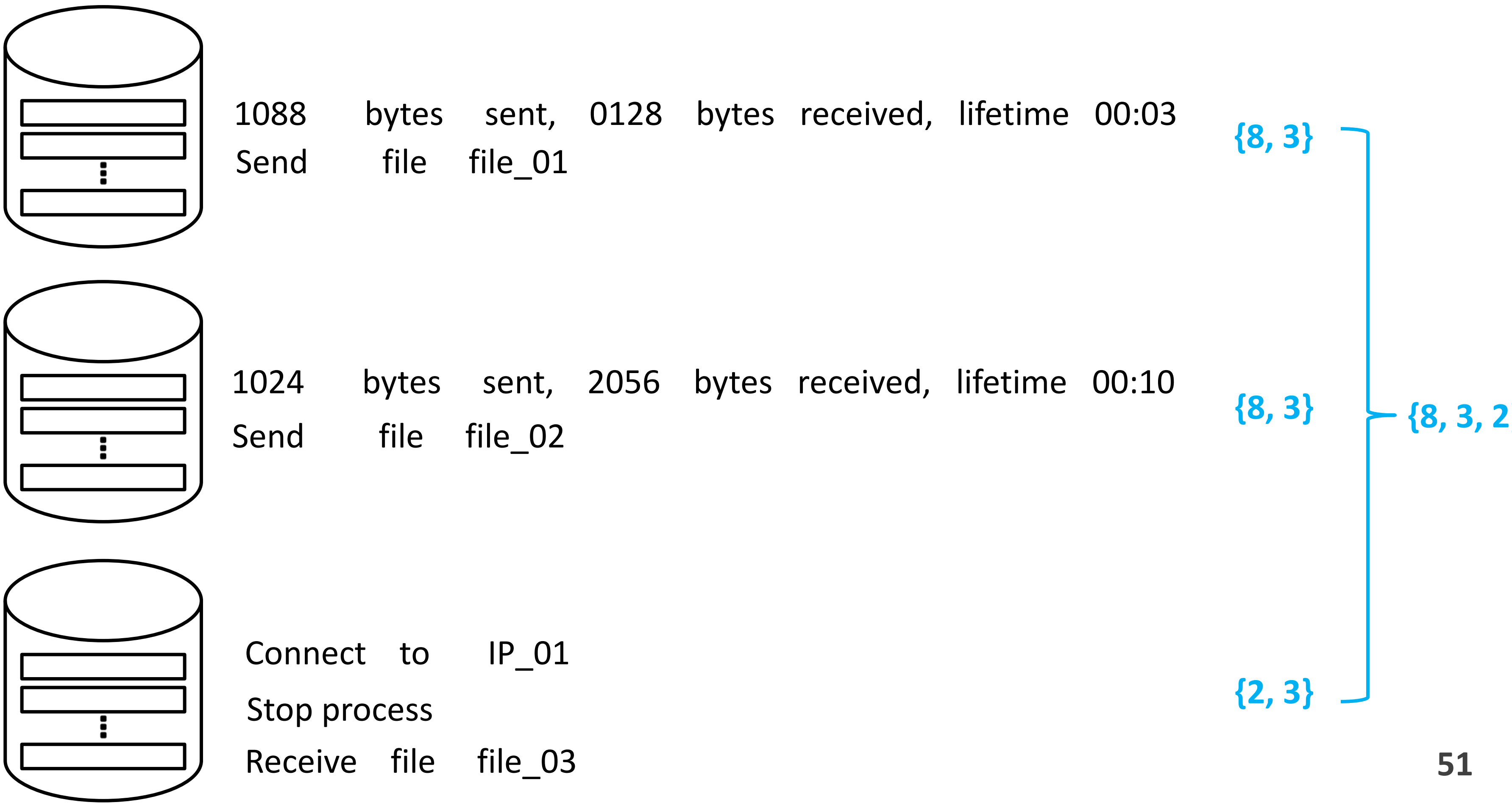
Step 5: Merge groups by log event

- **Hierarchical clustering** on **log events**, instead of log messages



Parallelization

- Log message lengths in Step 2



Data sets

- Data set (supercomputer, distributed system, standalone software)

System	Description	#Logs	Length	#Events	
BGL	BlueGene/L Supercomputer	4,747,963	10∼102	376	[DSN'07]
HPC	High Performance Cluster (Los Alamos)	433,490	6∼104	105	[TKDE'12]
Proxifier	Proxy Client	10,108	10∼27	8	
HDFS	Hadoop File System	11,175,629	8∼29	29	[SOSP'09]
Zookeeper	Distributed System Coordinator	74,380	8∼27	80	

- Randomly select 2,000 logs from each data set

- **Accuracy results:**

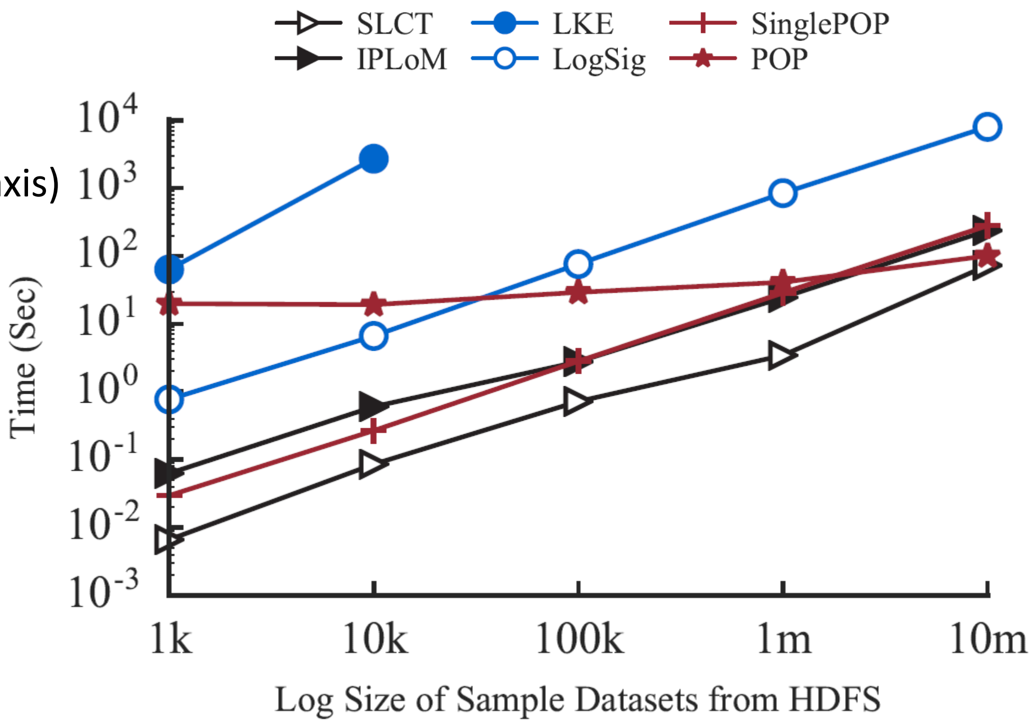
	BGL	HPC	HDFS	Zookeeper	Proxifier
SLCT	0.94	0.86	0.93	0.92	0.89
IPLoM	0.99	0.64	1.00	0.90	0.90
LKE	0.70	0.17	0.96	0.82	0.81
LogSig	0.98	0.87	0.93	0.99	0.84
POP	0.99	0.95	1.00	0.99	1.00

- Evaluate the running time of log parsing methods on all data sets by **varying the number** of raw logs.

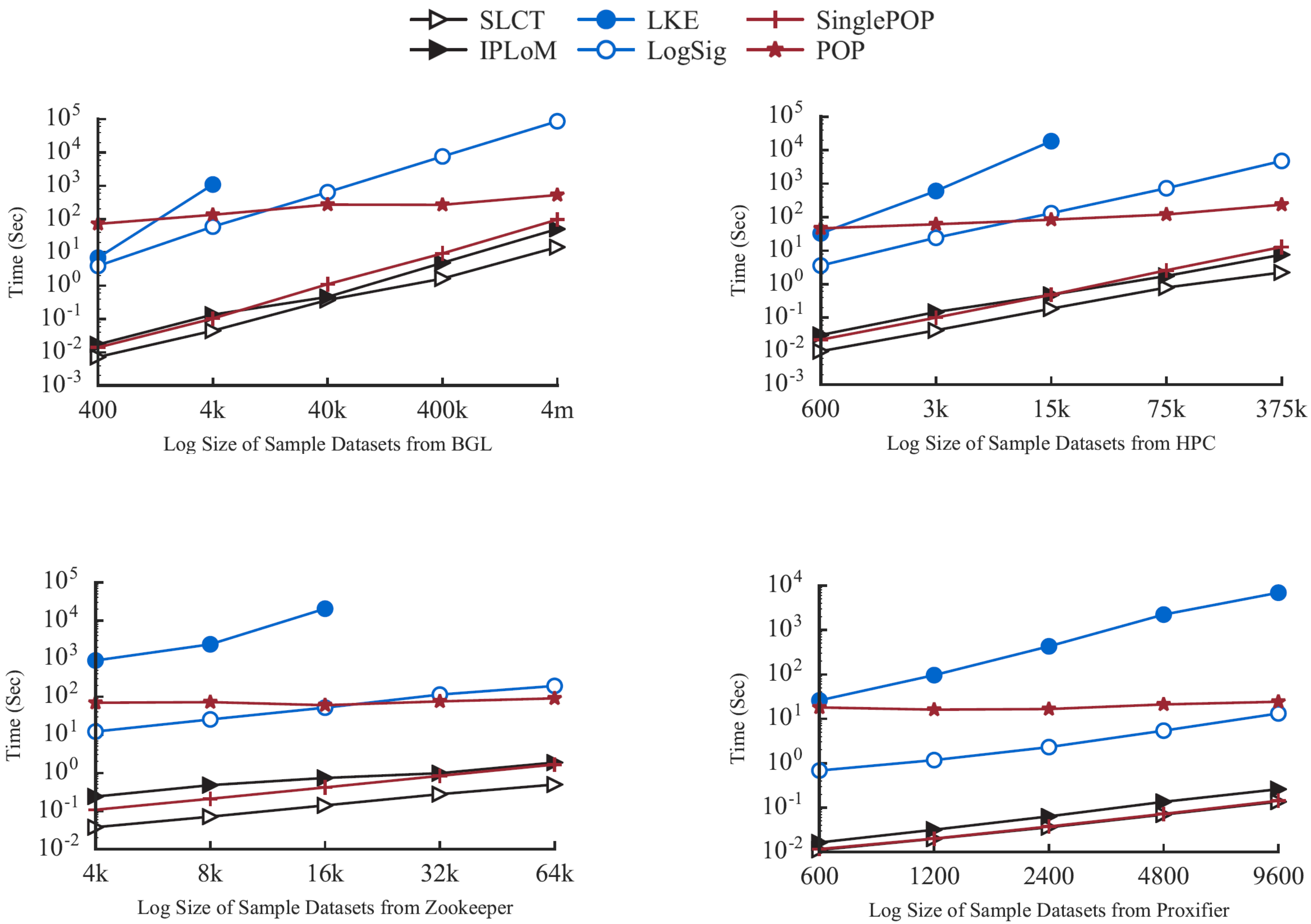
BGL	400	4k	40k	400k	4m
HPC	600	3k	15k	75k	375k
HDFS	1k	10k	100k	1m	10m
Zookeeper	4k	8k	16k	32k	64k
Proxifier	600	1200	2400	4800	9600

- **Efficiency experiments (real-world datasets):**

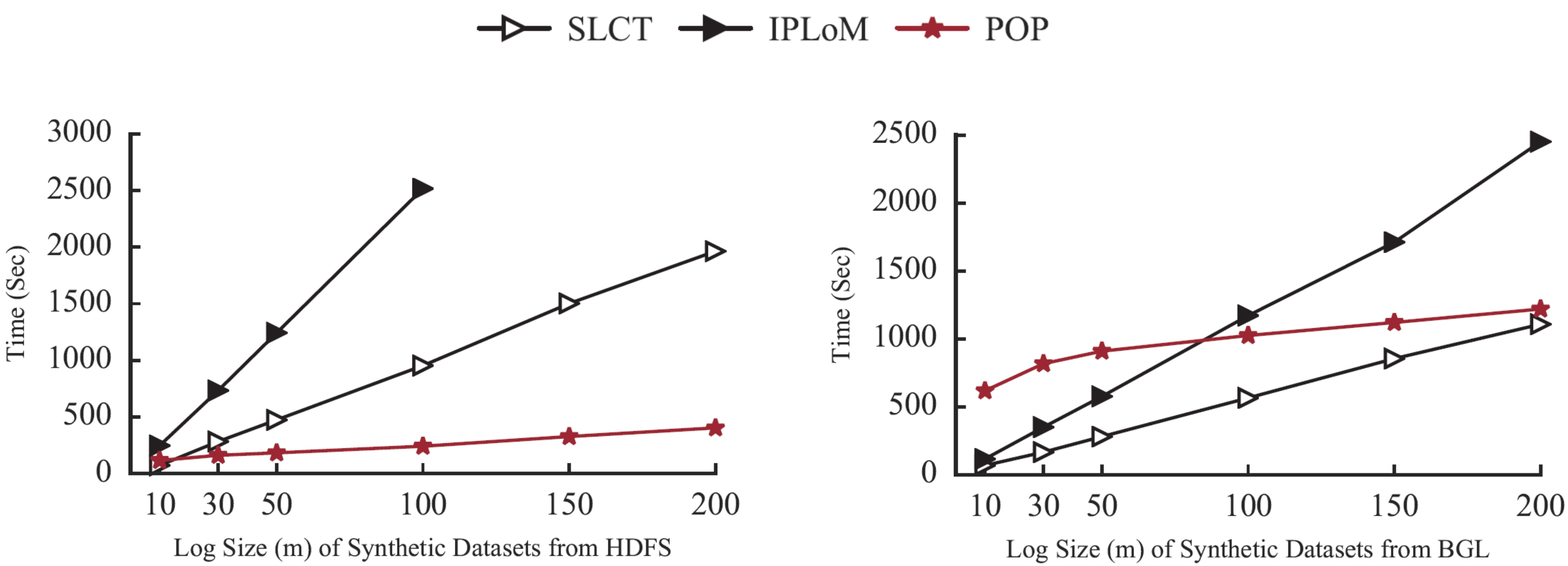
- Running time (y-axis)
- Slope



RQ1: Accuracy RQ2: Efficiency



- **Efficiency experiments (synthetic datasets):**



Outline

- **Topic 1: Evaluation study** on log parsing
- **Topic 2: Parallel log parsing** for large-scale log data
- **Topic 3: Online log parsing** via fixed depth tree
- Conclusion and future work

Why we need online log parsers?

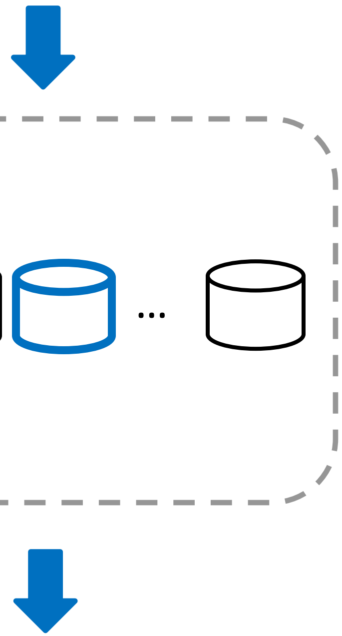
Motivations

- **Offline log parsers**
  - Log event changes
- **Modern system structure**
  - Log collection works in a streaming manner

An **online log parser** is in demand

Framework of Online Parser

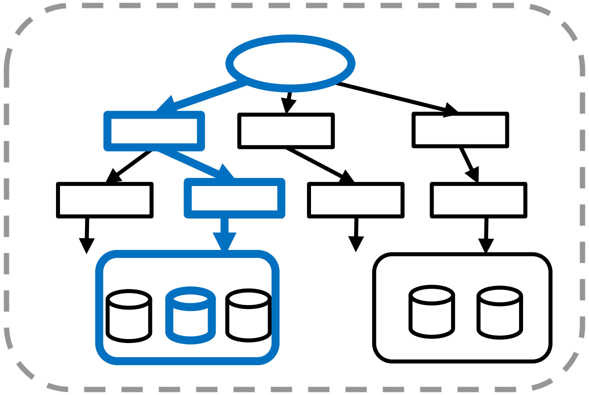
2008-11-11 03:41:48 Received block blk_90 of size 67108864 from /10.250.18.114



blk_90 -> Received block * of size * from *

Framework of Drain

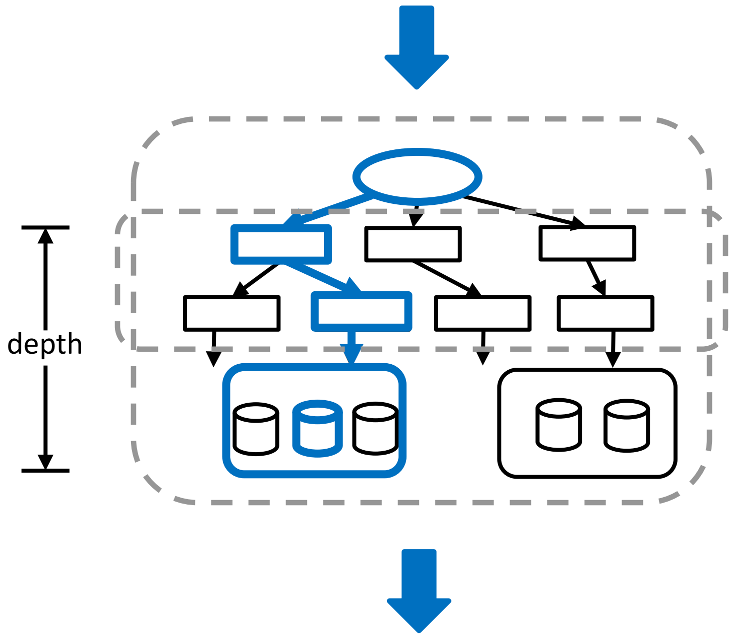
2008-11-11 03:41:48 Received block blk_90 of size 67108864 from /10.250.18.114



blk_90 -> Received block * of size * from *

Novelty of Drain

2008-11-11 03:41:48 Received block blk_90 of size 67108864 from /10.250.18.114

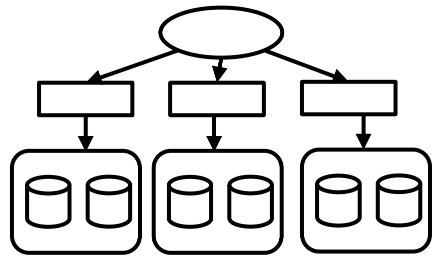


- Encodes heuristic rules in the tree to conduct pre-clustering.
- The depth can be fixed by a pre-defined parameter depth.

blk_90 -> Received block * of size * from *

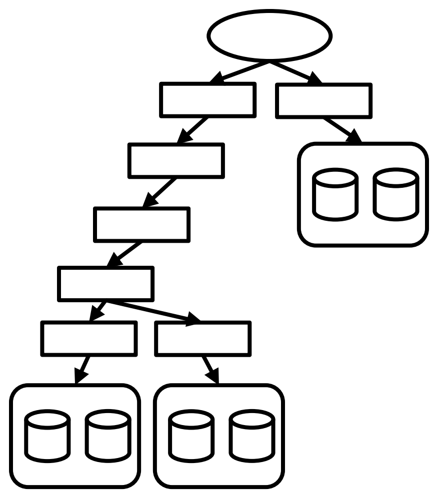
Depth

Under-parsed



Send	file	file_01
Send	file	file_01
Receive	file	file_02

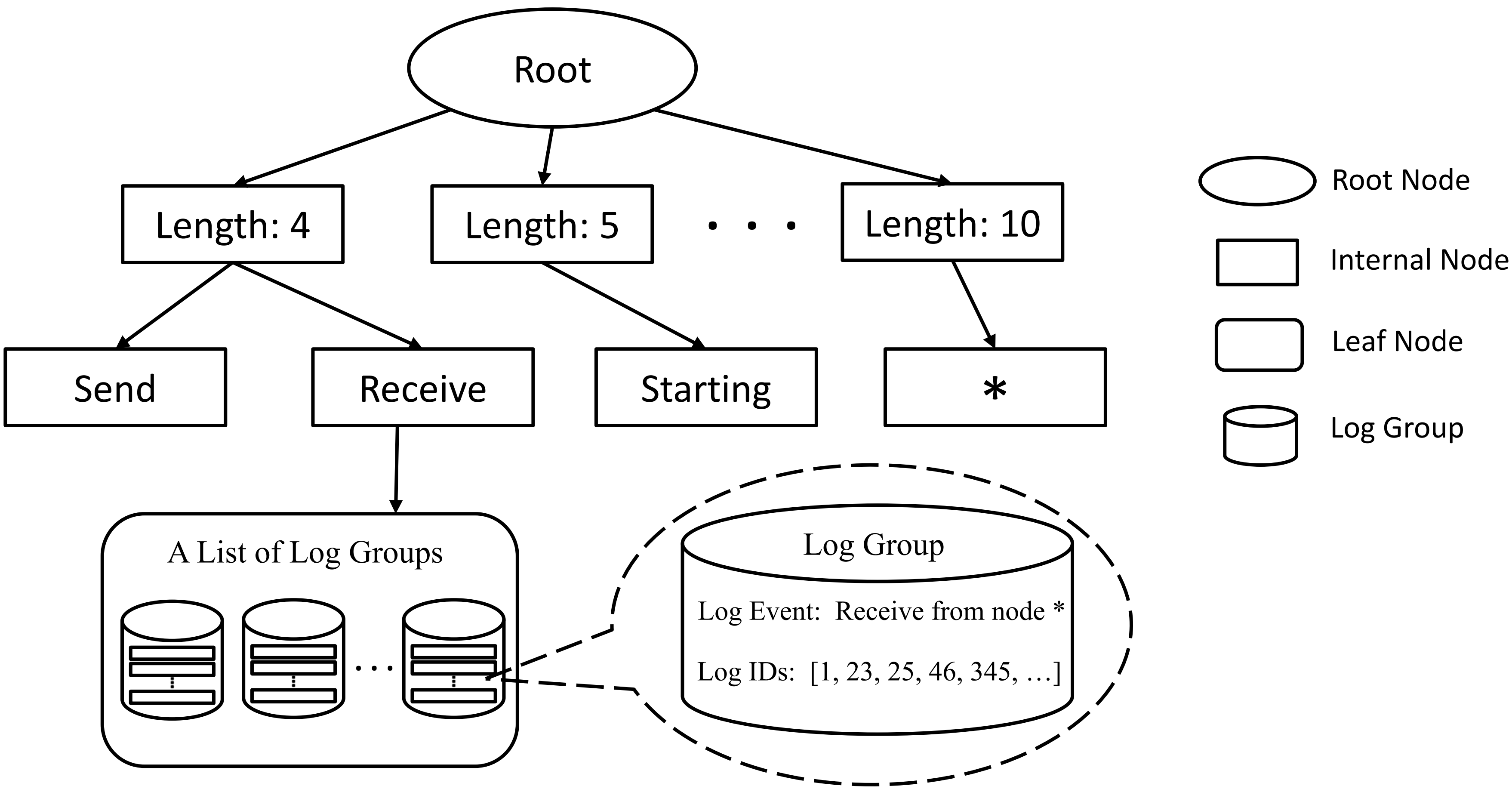
Over-parsed



Send	file	run.py

Send	file	boot.py

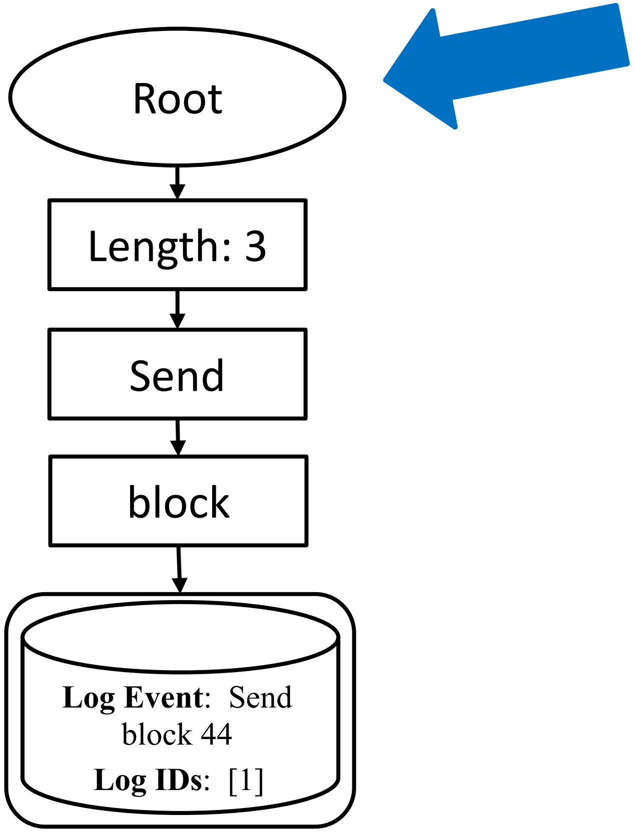
Framework of Drain



Fixed depth tree (depth=3)

Update of Drain

Receive file 01



Fixed depth tree (depth=4)

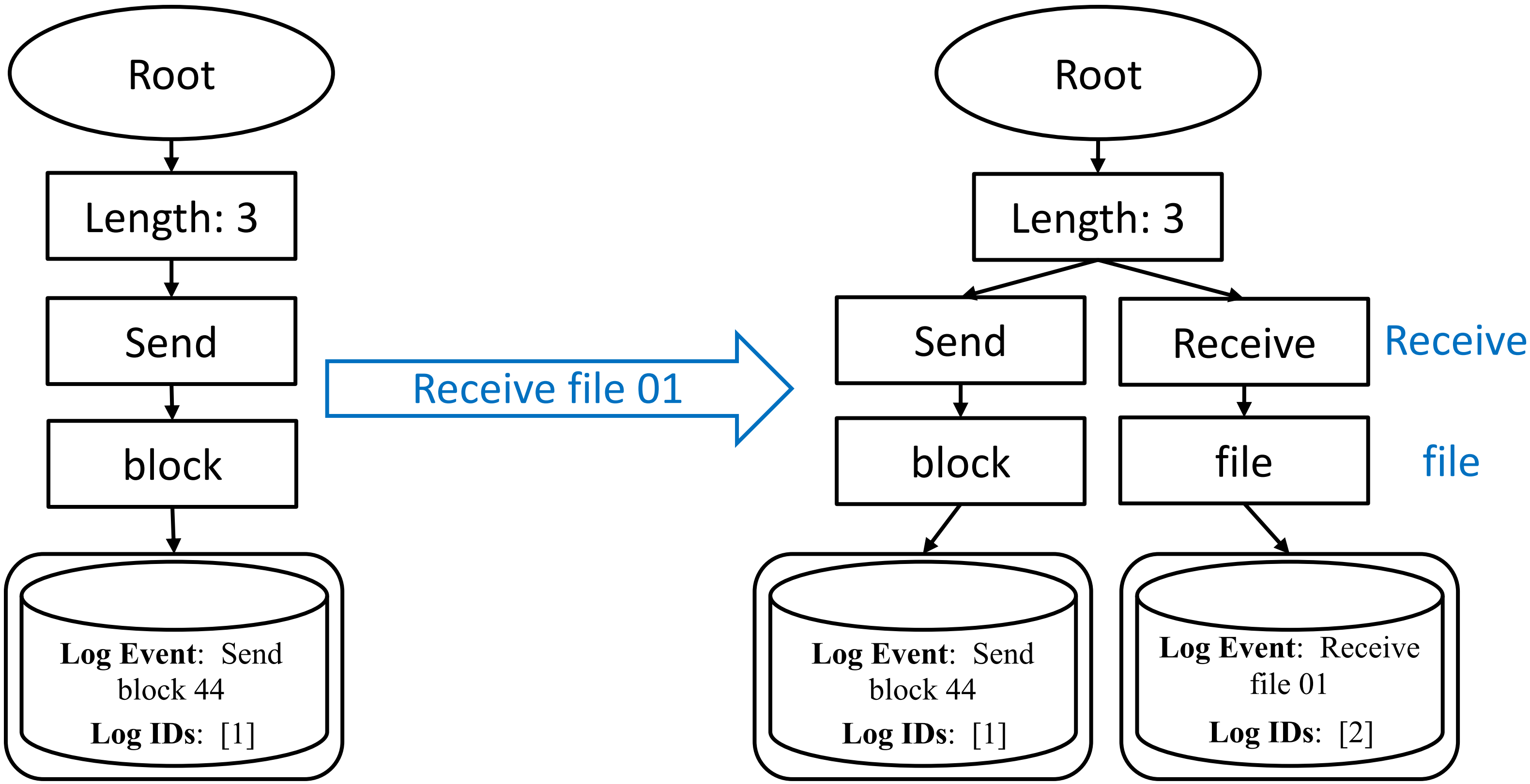
Length is 3

First token is Receive

NOT match!

Need to update the tree.

Update of Drain



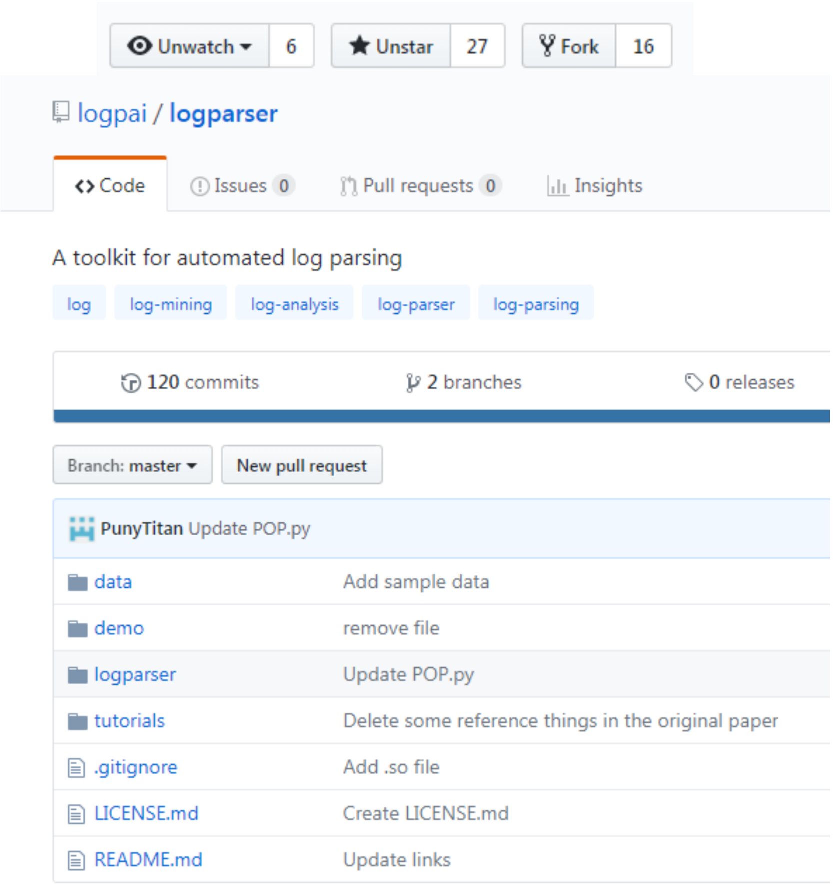
Fixed depth tree (depth=4)

- **Accuracy results:**

	BGL	HPC	HDFS	Zookeeper	Proxifier
Offline Log Parsers					
LKE	0.67	0.17	0.57	0.78	0.85
IPLoM	0.99	0.65	0.99	0.99	0.85
Online Log Parsers					
SHISO	0.87	0.53	0.93	0.68	0.85
Spell	0.98	0.82	0.87	0.99	**0.87**
Drain	**0.99**	**0.84**	**0.99**	**0.99**	0.84

- **Efficiency experiments:**

	BGL	HPC	HDFS	Zookeeper	Proxifier
Offline Log Parsers					
LKE	N/A	N/A	N/A	N/A	8888.49
IPLoM	140.57	12.74	333.03	2.17	0.38
Online Log Parsers					
SHISO	10964.55	582.14	6649.23	87.61	8.41
Spell	447.14	47.28	676.45	5.27	0.87
Drain	115.96	8.76	325.7	1.81	0.27
Improvement	**74.07%**	**81.47%**	**51.85%**	**65.65%**	**68.97%**



Parsers

If you are not familiar with log parser, please check the Principles of Parsers
The codes are here.

- SLCT (Simple Logfile Clustering Tool): A Data Clustering Algorithm for Mining Patterns from Event Logs (SLCT is wrapped around on the C source code provided by the author.)
- IPLoM (Iterative Partitioning Log Mining): A Lightweight Algorithm for Message Type Extraction in System Application Logs
- LKE (Log Key Extraction): Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis
- LogSig: LogSig: Generating System Events from Raw Textual Logs
- Drain: Drain: An Online Log Parsing Approach with Fixed Depth Tree
- POP: a parallel log parsing method optimized on top of Spark.

Data

In data, there are 5 datasets for you to play with. Each dataset contains several text files.

- rawlog.log: The raw log messages with ID. "ID\tword1 word2 word3"
- template[0-9]+: The log messages belong to a certain template.
- templates: The text of templates.

Quick Start

Input: A raw log file. Each line of the file follows "ID\tword1 word2 word3"
Output: Two parts. One is splitted log messages (only contains log ID) in different text files. The other is the ***templates*** file which contains all templates.

Examples: Before running the examples, please copy the parser source file to the same directory.

- Example1: This file is a simple example to demonstrate the usage of LogSig. The usage of other log parsers is similar.
- Example2: This file is to demonstrate the usage of POP.
- Example3: This file is used to evaluate the performance of LogSig. It iterates 10 times and record several important

Parsers are open source on **github.com/logpai/logparser**

Outline

- **Topic 1: Evaluation study** on log parsing
- **Topic 2: Parallel log parsing** for large-scale log data
- **Topic 3: Online log parsing** via fixed depth tree
- Conclusion and future work

Conclusion

Contributions

- **Evaluation study of log parsing**
  - Six insightful findings and an open-source log parsing toolkit
- **Parallel log parsing** for large-scale log data
  - A parallel log parsing framework POP
- **Online log parsing**
  - An online log parsing method Drain based on fixed depth tree

Conclusion

Contributions

- **Evaluation study of log parsing**
  - Six insightful findings and an open-source log parsing toolkit
- **Parallel log parsing**
  - A parallel log parsing framework POP built on top of Spark
- **Online log parsing**
  - An online log parsing method Drain based on fixed depth tree
- **Operational issues prioritization**
  - An operational issues prioritization method POI via hierarchical clustering
- **Location-based QoS prediction**

Future work

Parameter-free Online log parser

- An online log parser that automatically tunes the parameters

Publications (1)

Journal

1. Pinjia He, Jieming Zhu, Shilin He, Jian Li, Michael R. Lyu. Towards Automated Log Parsing for Large-Scale Log Data Analysis. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 14 pages, accepted, 2017.
2. Jieming Zhu, Pinjia He, Zibin Zheng, Michael R. Lyu. Online QoS Prediction for Runtime Service Adaptation via Adaptive Matrix Factorization. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, Volume 28, Issue 10, pages 2911-2924, 2017.

Conference

1. Pinjia He. An End-to-end Log Management Framework for Distributed Systems. *The 36th International Symposium on Reliable Distributed Systems (SRDS)*, pages 266-267, 2017.
2. Jieming Zhu, Pinjia He, Zibin Zheng, Michael R. Lyu. CARP: Context-Aware Reliability Prediction of Black-Box Web Services. *The 24th International Conference on Web Service (ICWS)*, pages 17-24, 2017.
3. Pinjia He, Jieming Zhu, Zibin Zheng, Michael R. Lyu. Drain: An Online Log Parsing Approach with Fixed Depth Tree. *The 24th International Conference on Web Service (ICWS)*, pages 33-40, 2017.
4. Jian Li, Pinjia He, Jieming Zhu, Michael R. Lyu. Software Defect Prediction via Convolutional Neural Network. *The International Conference on Software Quality, Reliability and Security (QRS)*, pages 318-328, 2017.

Publications (2)

5. Pinjia He, Jieming Zhu, Shilin He, Jian Li, Michael R. Lyu. An Evaluation Study on Log Parsing and Its Use in Log Mining. *The 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 654-661, 2016.

6. Shilin He, Jieming Zhu, Pinjia He, Michael R. Lyu. Experience Report: System Log Analysis for Anomaly Detection Factorization. *The 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207-218, 2016.

7. Cuiyun Gao, Baoxiang Wang, Pinjia He, Jieming Zhu, Yangfan Zhou, Michael R. Lyu. PAID: Prioritizing App Issues for Developers by Tracking User Reviews Over Versions. *The 26th International Symposium on Software Reliability Engineering (ISSRE)*, pages 35-45, 2015.

8. Jieming Zhu, Pinjia He, Qiang Fu, Hongyu Zhang, Michael R. Lyu, Dongmei Zhang. Learning to Log: Helping Developers Make Informed Logging Decisions. *The 37th International Conference on Software Engineering (ICSE)*, pages 415-425, 2015.

9. Jieming Zhu, Pinjia He, Zibin Zheng, Michael R. Lyu. A Privacy-Preserving QoS Prediction Framework for Web Service Recommendation. *The 22nd International Conference on Web Service (ICWS)*, pages 241-248, 2015.

10. Pinjia He, Jieming Zhu, Zibin Zheng, Jianlong Xu, Michael R. Lyu. Location-Based Hierarchical Matrix Factorization for Web Service Recommendation. *The 21st International Conference on Web Service (ICWS)*, pages 297-304, 2014.

Publications (3)

11. Jieming Zhu, Pinjia He, Zibin Zheng, Michael R. Lyu. Towards Online, Accurate, and Scalable QoS Prediction for Run time Service Adaptation. *The 34th International Conference on Distributed Computing Systems (ICDCS)*, pages 318-327, 2014.
12. Tong Zhao, Junjie Hu, Pinjia He, Hang Fan, Michael R. Lyu, Irwin King. Exploiting Homophily-based Implicit Social Network to Improve Recommendation Performance. *The International Joint Conference on Neural Networks (IJCNN)*, pages 2539-2547, 2014.
13. Pinjia He, Jieming Zhu, Jianlong Xu, Michael R. Lyu. A Hierarchical Matrix Factorization Approach for Location-Based Web Service QoS Prediction. *The International Workshop on Internet-based Virtual Computing Environment (iVCE)*, pages 290-295, 2014.

Thank you!

Q&A