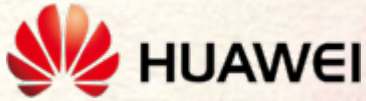
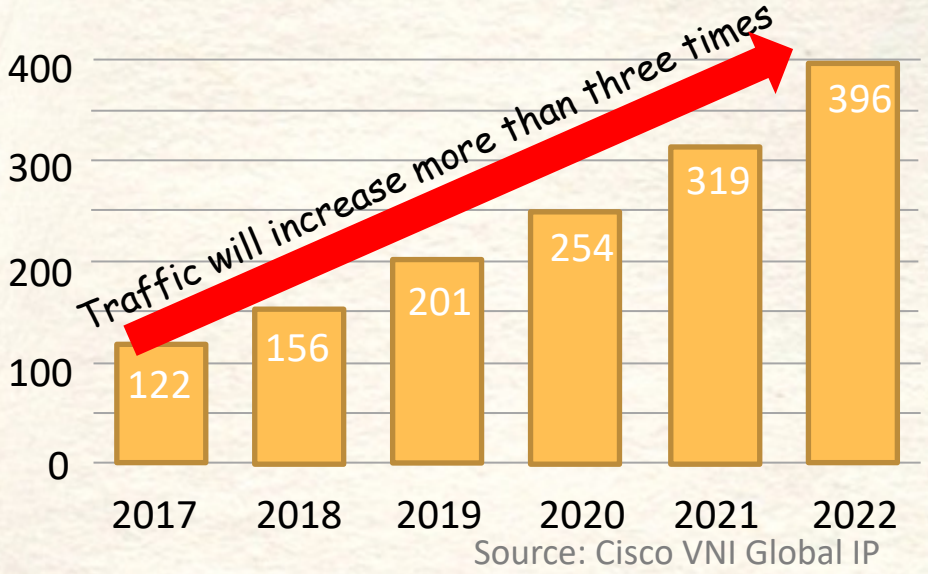
LogParse: Making Log Parsing Adaptive through Word Classification

Weibin Meng, Ying Liu, Federico Zaiter, Shenglin Zhang, Yihao Chen, Yuzhe Zhang
Yichen Zhu, En Wang, Ruizhi Zhang, Shimin Tao, Dian Yang, Rong Zhou, Dan Pei



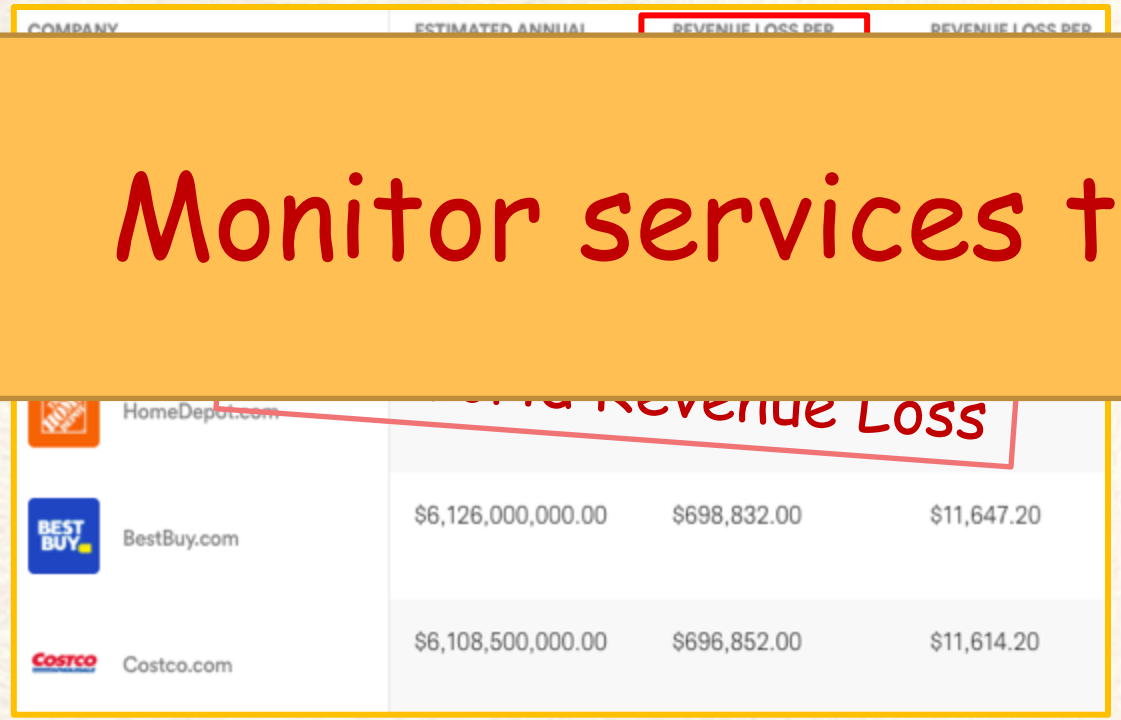
Background

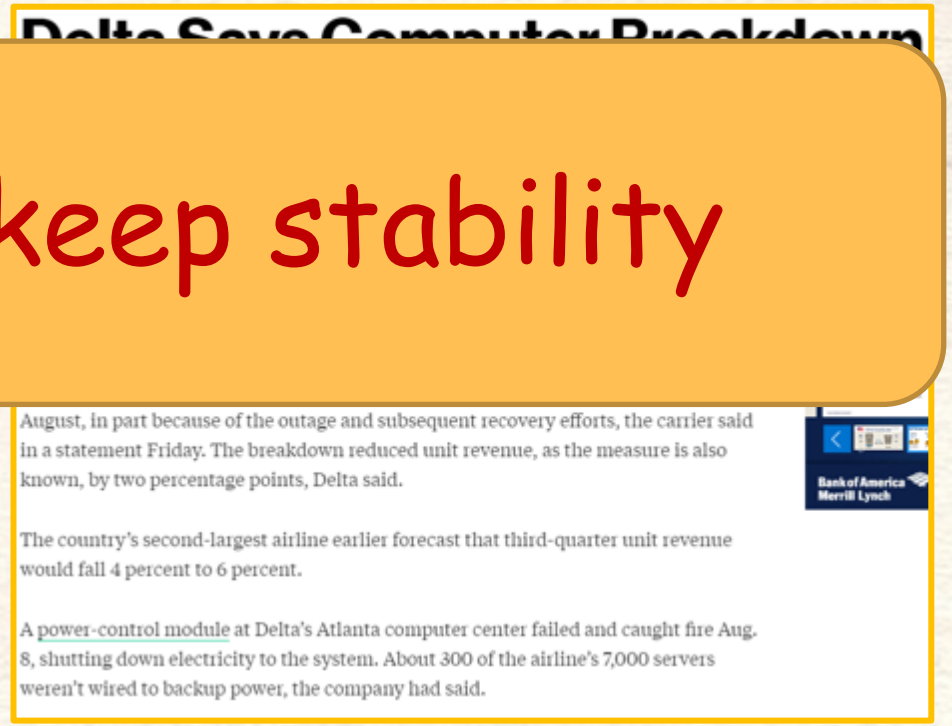
- Internet provide various types of services
- The traffic is growing rapidly.



Background

- Stability of services are becoming more and more important.





Monitor services to keep stability

Service logs

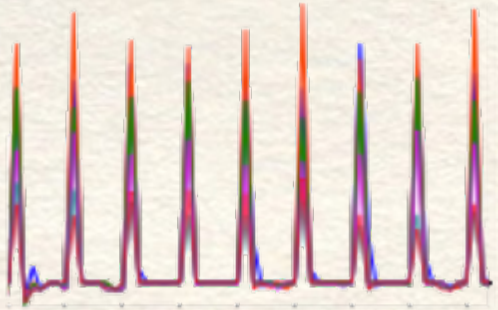
- Logs are the most valuable data for service management
  - Logs record a vast range of events (7*24) of services
  - Every service generates logs

Types	Timestamps	Detailed messages
Switch	Jul 10 19:03:03	Interface te-1/1/59, changed state to down
Supercomputer	Jun 4 6:45:50	RAS KERNEL INFO 87 L3 EDRAM error dcr 0x0157 detected and corrected over 27362 seconds
HDFS	Jun 8 13:42:26	INFO dfs.DataNode PacketResponder: PacketResponder 1 for block blk_-160899968791[illegible]862906 terminating
Router	Jul 11 11:05:07	Neighbour(rid:10.231.0.43, addr:10.231.39.61) on vlan23, changed state from Exchange to Loading

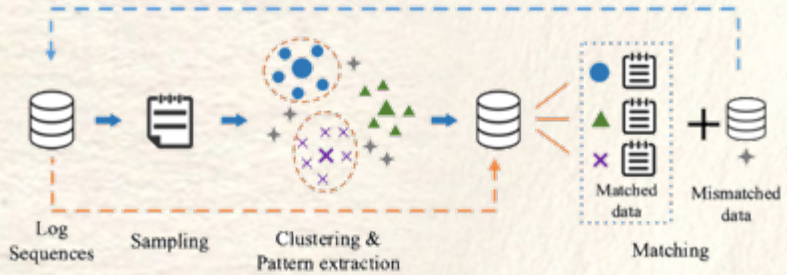
Unstructured logs

Log parsing

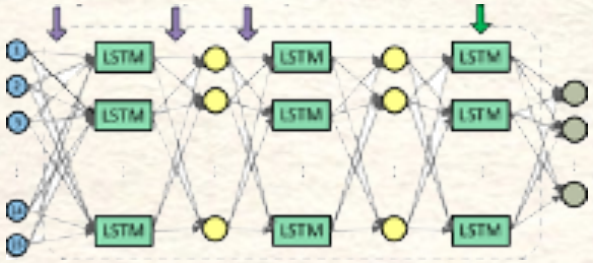
- Log analysis → Log-based service management
- Log analysis contains two steps[1]:
  - Log Parsing and Log Mining
- Log parsing effects the performance of log analysis



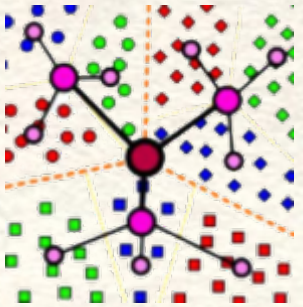
Performance monitoring
[INFOCOM'19]



Problem Identifying
[FSE'18]



Anomaly detection
[IJCAI'19]

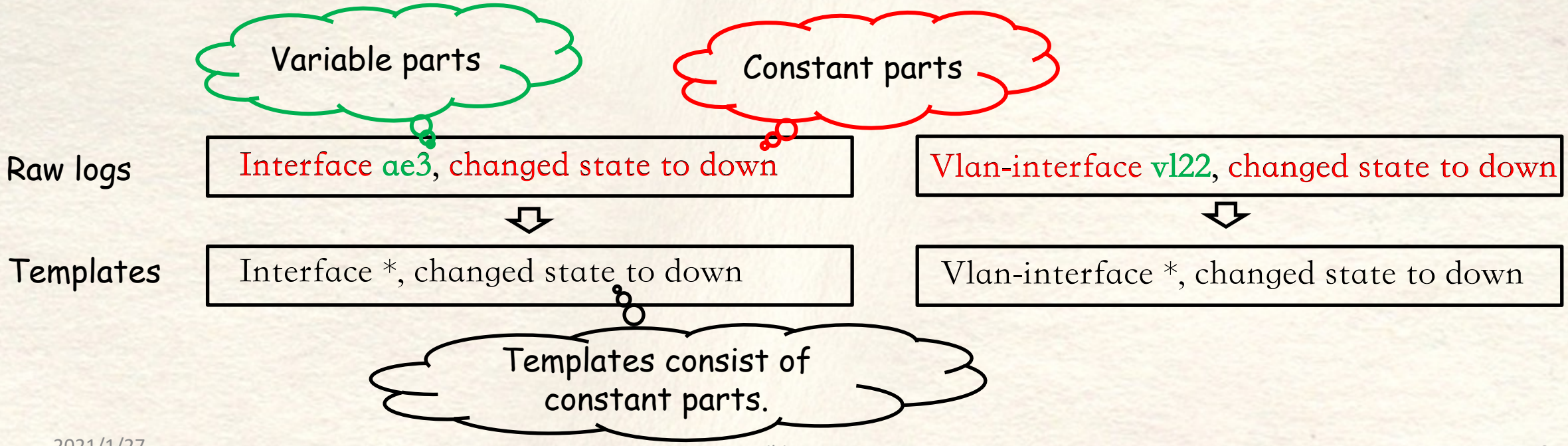


Failure prediction
[SIGMETRICS'18]

[1] Pinjia He, Jieming Zhu, et al. An Evaluation Study on Log Parsing and Its Use in Log Mining. DSN'16

Log parsing

- An unstructured log is "printf"ed by services
- The goal of log parsing is to distinguish between constant part and variable part.



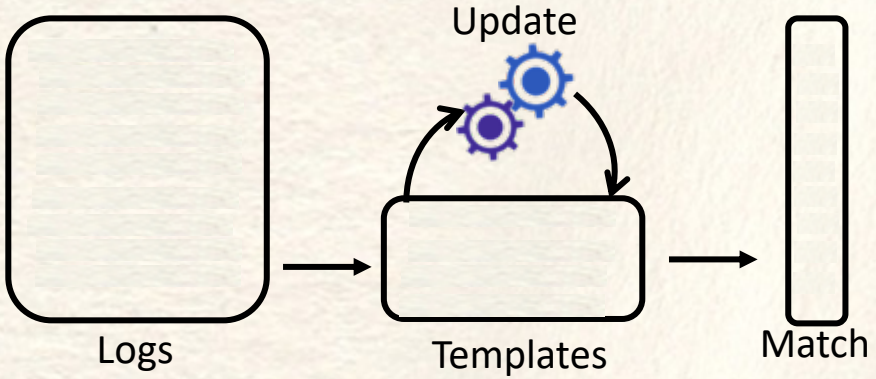
Adaptiveness

- Adaptiveness is important for log parsing
  - Goal: match any types of logs
- Intra-service adaptiveness ✖
- Cross-service adaptiveness ✖

Traditional log paring approaches or don't support intra-service adaptiveness, or do not support cross-service adaptiveness, or both.

Intra-service Adaptiveness

- Intra-service adaptiveness
  - Software/firmware upgrades can generate new types of logs
  - New logs cannot match any existing templates



Historical logs:
L_1. Interface ae3, changed state to down
L_2. Vlan-interface vl22, changed state to down
L_3. Interface ae3, changed state to up
L_4. Interface ae1, changed state to down

Real-time logs:
L_5. Interface ae1, changed state to up
L_6. Vlan-interface vl22, changed state to up

Template extraction:
T_1. Interface *, changed state to down
T_2. Vlan-interface *, changed state to down
T_3. Interface *, changed state to up

Template update:
T_4. ? ? ?

Template match:
L_1->T_1 ,ae3 L_2->T_2 ,vl22 L_3->T_3 , ae3
L_4->T_1,ae1 L_5->T_3 ,ae1 L_6-> ? ? ?

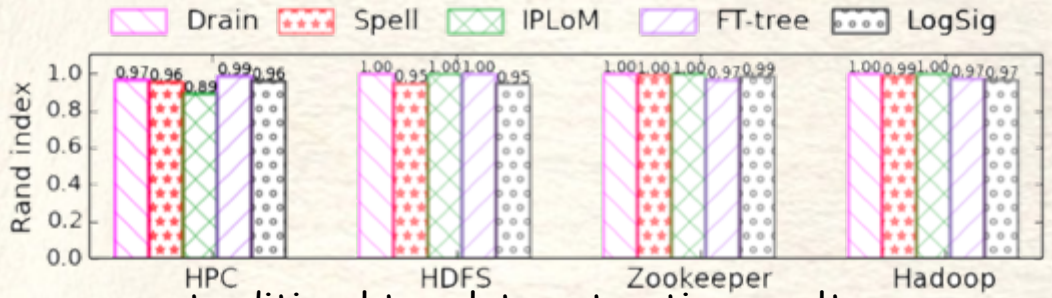
Intra-service Adaptiveness

- Traditional log parsing methods:
  - Drain (ICWS'17), FT-tree (IWQoS'18) which claimed to support template update
  - LogSig (CIKM'11), Spell (ICDM'16), IPLoM (KDD'09) don't support template update

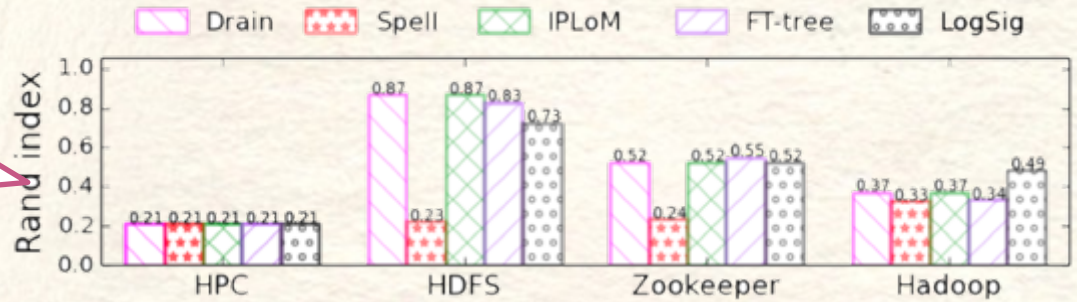
Historical logs:
L_1. Interface ae3, changed state to down
L_2. Vlan-interface vl22, changed state to down
L_3. Interface ae3, changed state to up
L_4. Interface ae1, changed state to down
Real-time logs:
L_5. Interface ae1, changed state to up
L_6. Vlan-interface vl22, changed state to up

Template extraction:
T_1. Interface *, char
T_2. Vlan-interface *
T_3. Interface *, char
Template update:
T_4. ? ? ?
Template match:
L_1->T_1,ae3 L_2->T_2,vl22 L_3->T_3, ae3
L_4->T_1,ae1 L_5->T_3,ae1 L_6-> ? ? ?

When face new types of logs, all traditional methods achieve low accuracies



traditional template extraction results
when all the logs are used for training

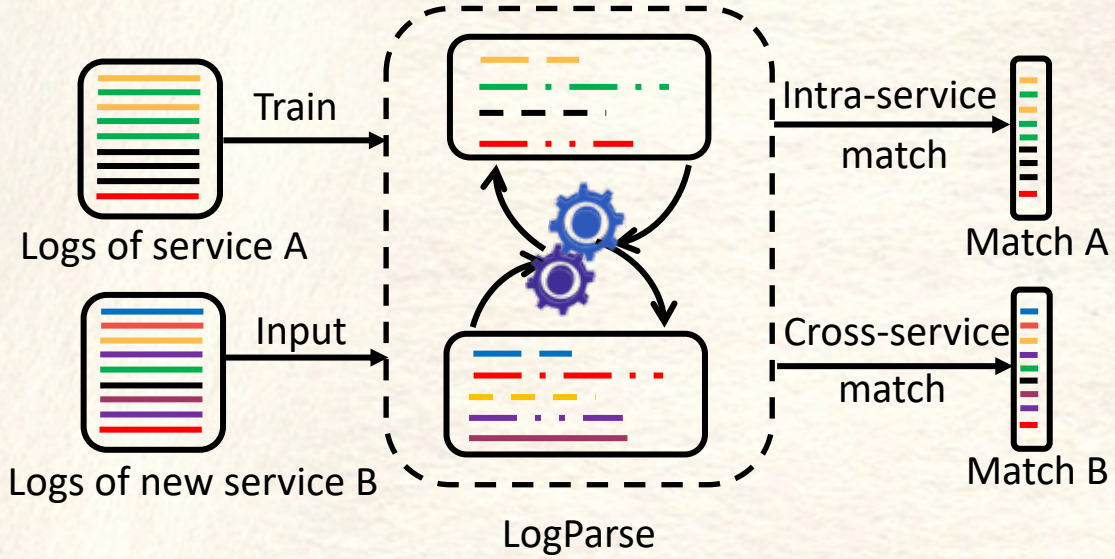


traditional template extraction results
when only 10% of logs are used for training

Cross-service Adaptiveness

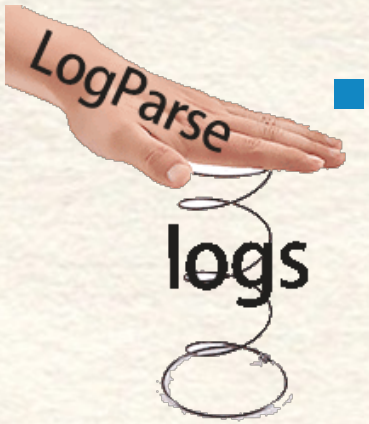
- Observation:
  - No enough historical logs when a brand new service goes online
- Aim:
  - A model trained by service A is also suitable for service B

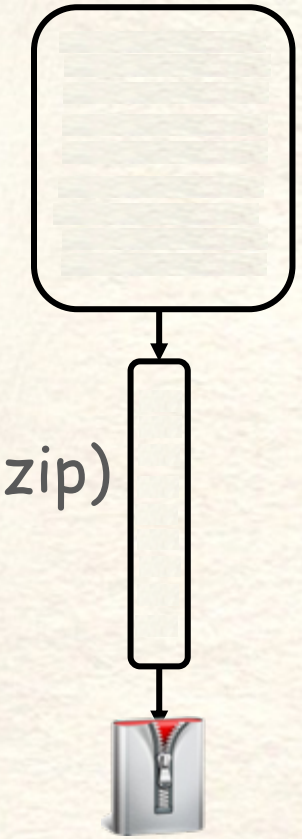
- Cross-service adaptive is for models rather than template sets.
- Templates are generated by trained model.



Log compression

- Log parsing without adaptiveness limit many log analysis applications
  - Requires a corresponding template for any given log
- Log Compression (An application)
  - Short-term storage (real-time queries)
    - Template index + Variables
  - Long-term storage
    - LogParse + Traditional method (e.g., zip)





Historical logs:
L_1. Interface ae3, changed state to down
L_2. Vlan-interface vl22, changed state to down
L_3. Interface ae3, changed state to up
L_4. Interface ae1, changed state to down
Real-time logs:
L_5. Interface ae1, changed state to up
L_6. Vlan-interface vl22, changed state to up

Short term storage by LogParse:
L_1->T_1 ,ae3
L_2->T_2 ,vl22
L_3->T_3 , ae3
L_4->T_1,ae1
L_5->T_3 ,ae1
L_6->T_4 ,vl22

Long-term compression

Idea

- Observation:
  - Operators usually distinguish variables based on features of words

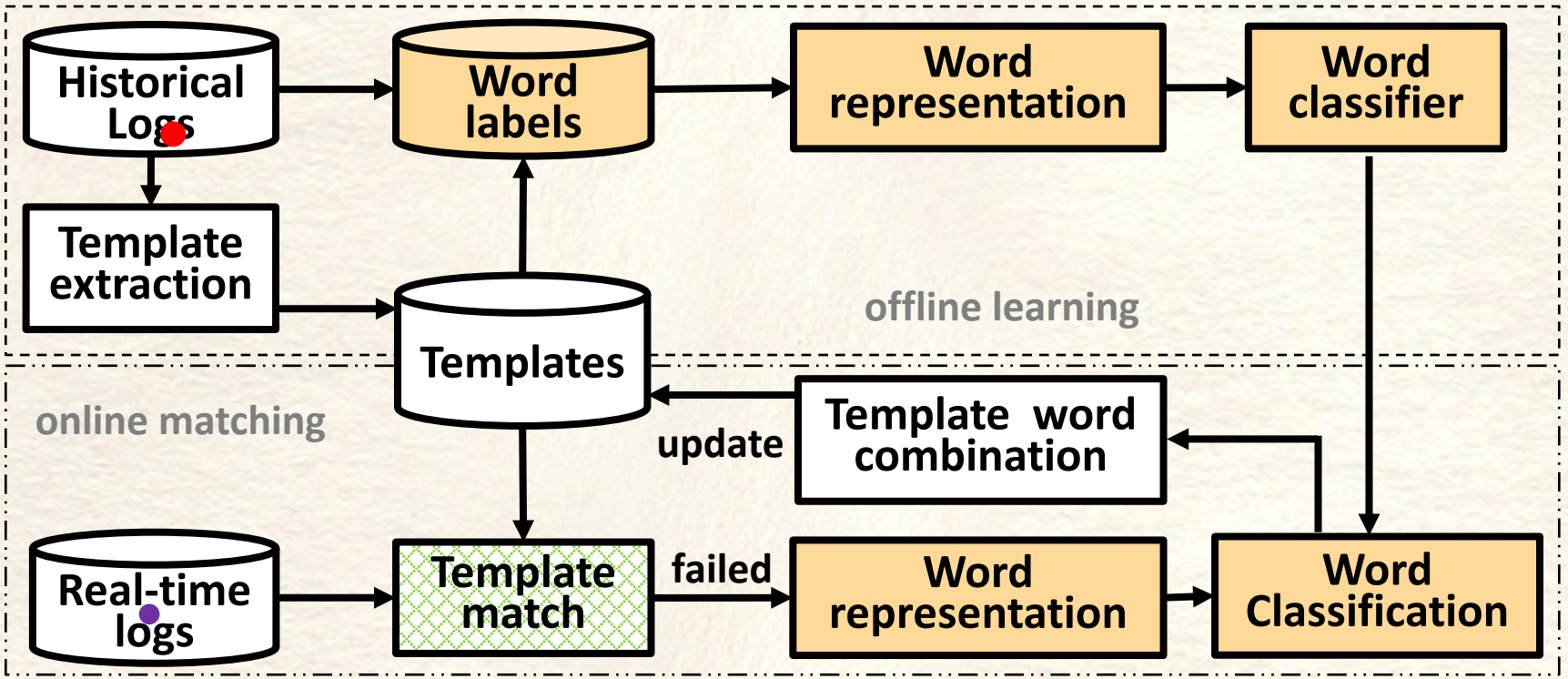
Mixed characters and numbers are usually variables

Historical logs:
L_1. Interface ae3, changed state to down
L_2. Vlan-interface vl22, changed state to down
L_3. Interface ae3, changed state to up
L_4. Interface ae1, changed state to down
Real-time logs:
L_5. Interface ae1, changed state to up
L_6. Vlan-interface vl22, changed state to up

letters are usually template words

A log parsing problem → A word classification problem

LogParse Workflow

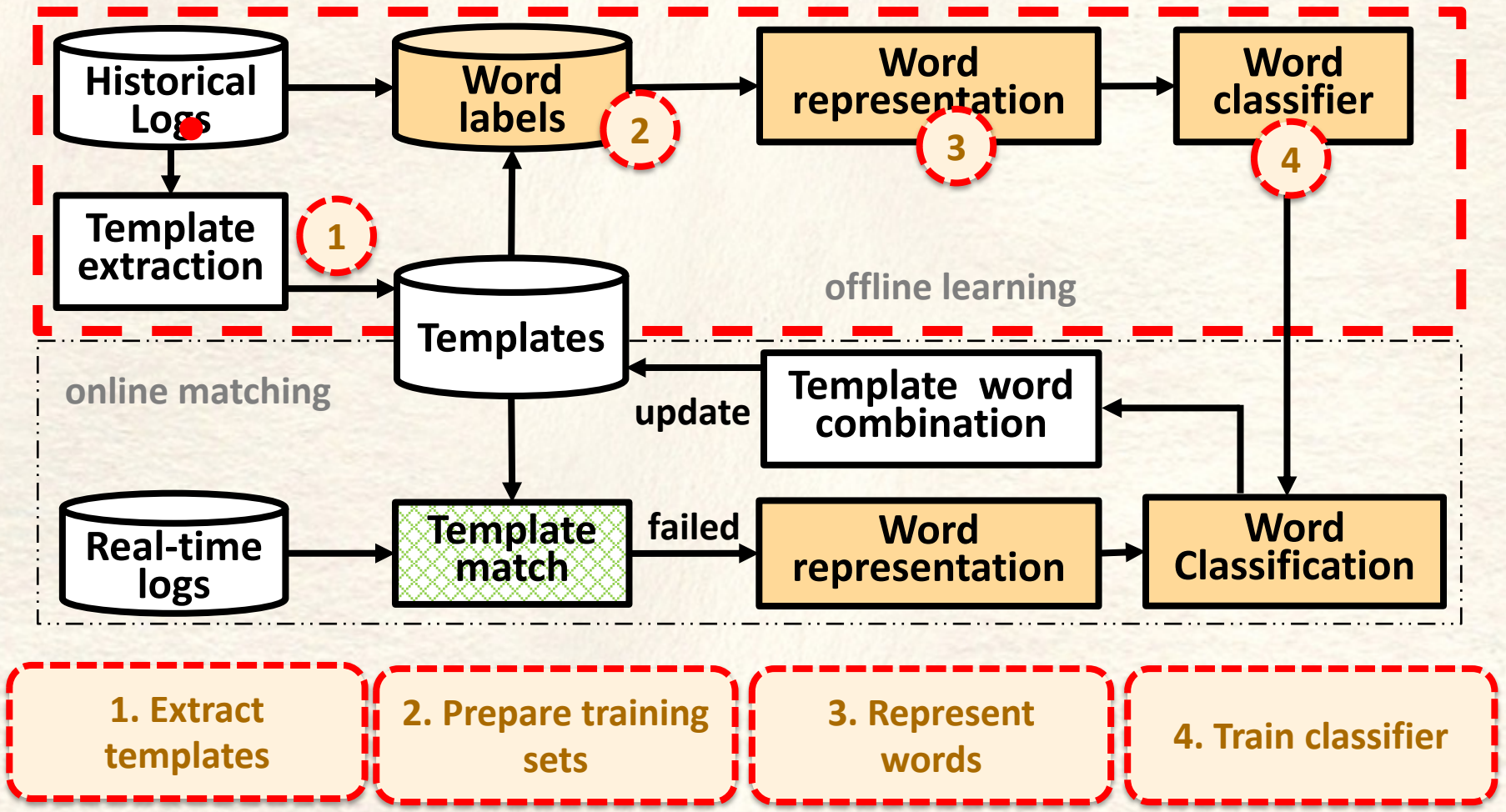


- Offline learning:
  - Prepare training word sets and train word classifier
- Online log parsing:
  - Match logs and update template sets

An adaptive framework for online log parsing

Toolkit: https://github.com/WeibinMeng/LogParse

Offline Learning

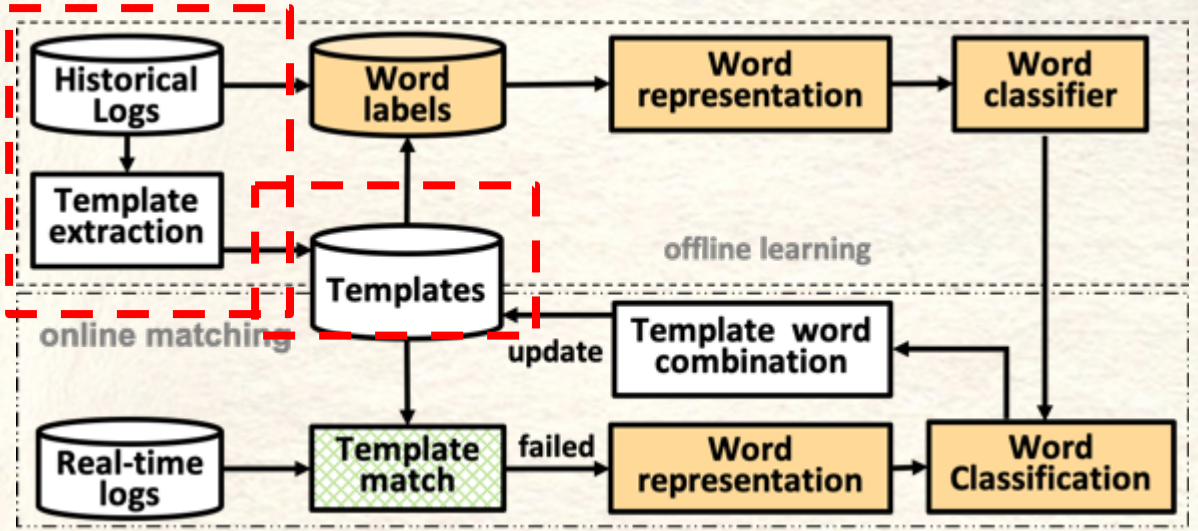


Template extraction

- Extract templates by traditional log parsing methods
  - Generate accurate templates (in offline stage)
  - Unsupervised methods
- Use the results as the initial template set

Rawlogs:
L_1. Interface ae3, changed state to down
L_2. Vlan-interface vlan22, changed state to down
L_3. Interface ae3, changed state to up.
L_4. Interface ae1, changed state to down

Templates:
T_1. Interface *, changed state to down
T_2. Vlan-interface *, changed state to down
T_3. Vlan-interface *, changed state to up



Prepare training sets

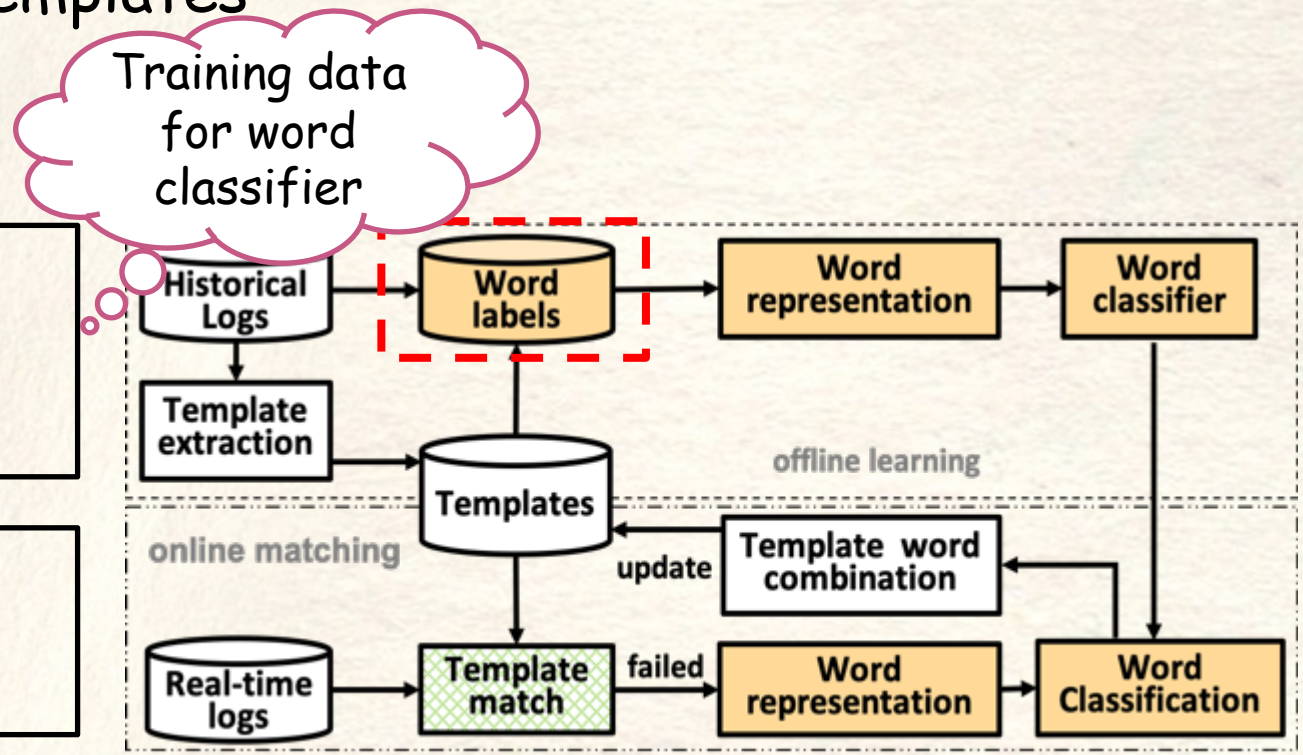
- Distinguish variable/template words
  - variable words: words in logs but not in templates
  - template words: words in templates

Rawlogs:
L_1. Interface ae3, changed state to down
L_2. Vlan-interface vlan22, changed state to down
L_3. Interface ae3, changed state to up.
L_4. Interface ae1, changed state to down

Templates:
T_1. Interface *, changed state to down
T_2. Vlan-interface *, changed state to down
T_3. Vlan-interface *, changed state to up

Template words:

Variable words:



Word representation

- Machine learning algorithms require structured data
- Present each word by using a character-level count vector
  - The set of characters is fixed -> fixed dimensionality
    - e.g., 128 characters in ASCII

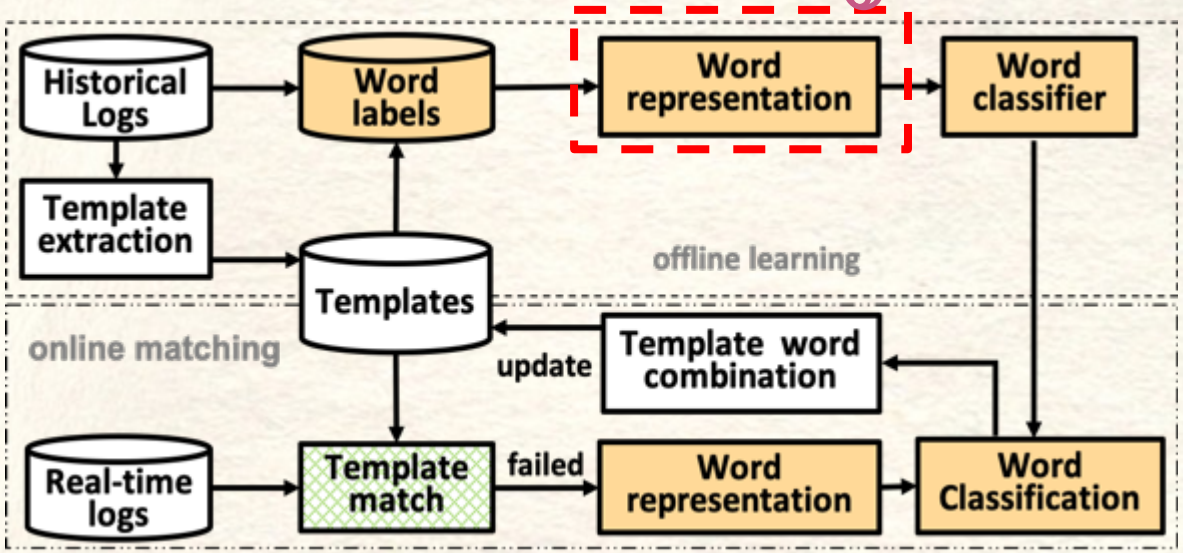
We can represent any word even unseen words

Templates words:
Interface changed state to Vlan-interface down up

Variables words:
ae3, ae1, vlan22

Character count vector:

a - z, A - Z,	0 - 10,	symbol	label
[1,	,2,	4, ,x ,0 ,0 ⋯,	x, x, x] template
[x,	,x,	x, ,x ,1 ,2 ⋯,	x, x, x] variable
[x,	,x,	x, ,x ,0, 0 ⋯,	x, 1, x] template
[1,	,2,	4, ,x ,0 ,0 ⋯,	x, x, x] template
[x,	,x,	x, ,x ,1 ,2 ⋯,	x, x, x] variable
[x,	,x,	x, ,x ,0, 0 ⋯,	x, 1, x] template
[x,	,x,	x, ,x ,1 ,2 ⋯,	x, x, x] variable
[x,	,x,	x, ,x ,0, 0 ⋯,	x, 1, x] template
…			



Word classifier

- Train supervised machine learning classifier
  - E.g., SVM, Random forest.
- The whole framework of LogParse is unsupervised
  - We used unsupervised methods to generated training set

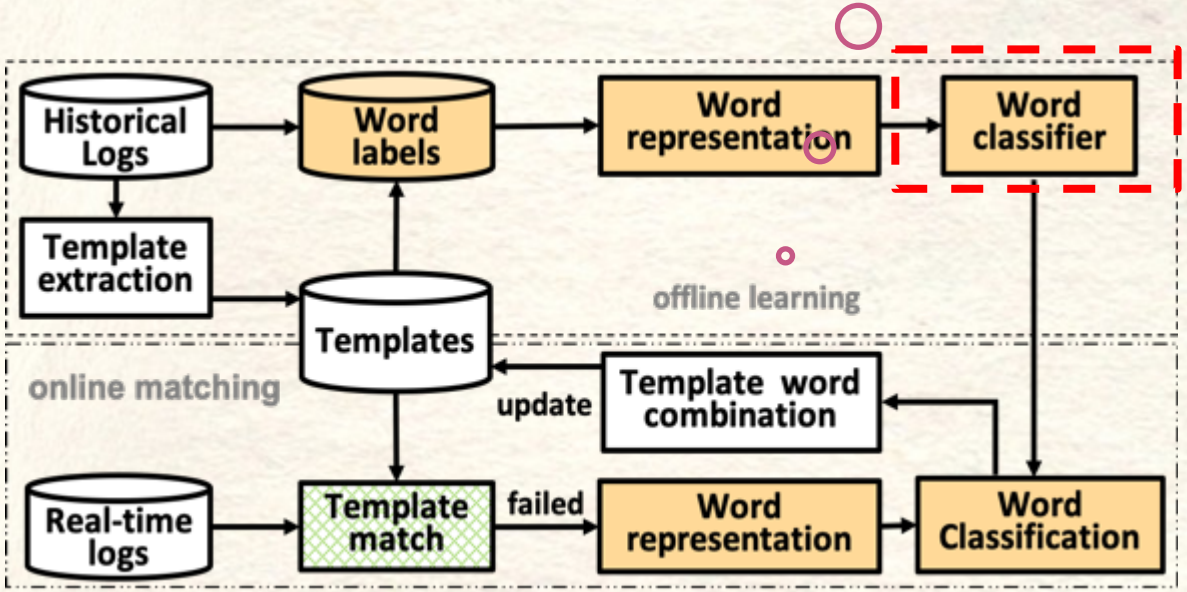
The whole framework is still unsupervised

Character count vector:

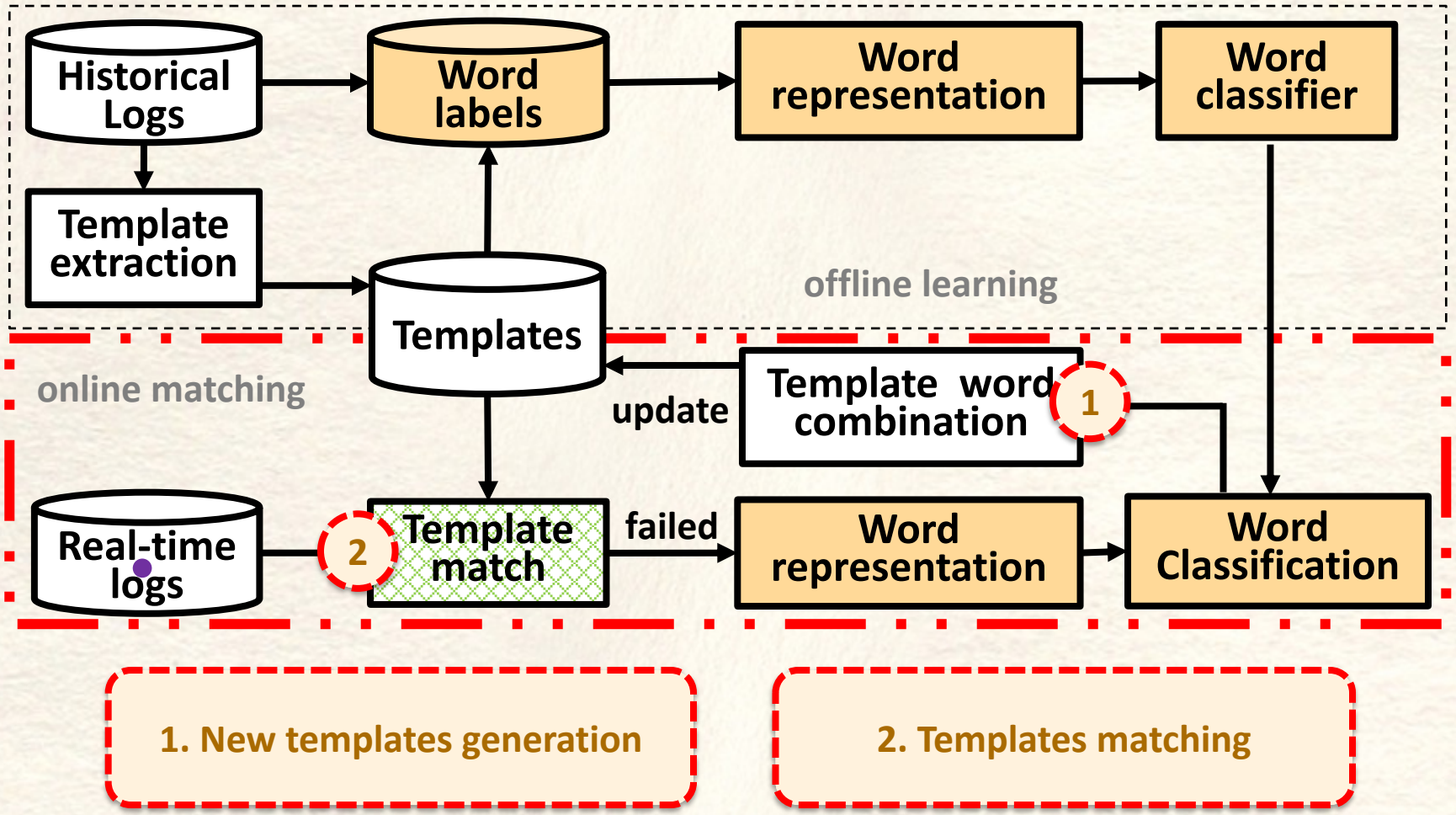
a - z, A - Z,	0 - 10,	symbol	label
[1,	,2,	4, ,x ,0 ,0 ⋯, x, x, x]	template
[x,	,x,	x, ,x ,1 ,2 ⋯, x, x, x]	variable
[x,	,x,	x, ,x ,0, 0 ⋯, x, 1, x]	template
[1,	,2,	4, ,x ,0 ,0 ⋯, x, x, x]	template
[x,	,x,	x, ,x ,1 ,2 ⋯, x, x, x]	variable
[x,	,x,	x, ,x ,0, 0 ⋯, x, 1, x]	template
[x,	,x,	x, ,x ,1 ,2 ⋯, x, x, x]	variable
[x,	,x,	x, ,x ,0, 0 ⋯, x, 1, x]	template
…			

Machine learning

Word classifier



Online log parsing



New templates generation

- Steps:
  - Classify each word by the trained word classifier.
  - Construct a new template by combining all template words

Historical logs:
L_1. Interface ae3, changed state to down
L_2. Vlan-interface vl22, changed state to down
L_3. Interface ae3, changed state to up
L_4. Interface ae1, changed state to down
Online logs (new type):
L_5. Vlan-interface vl22, changed state to up

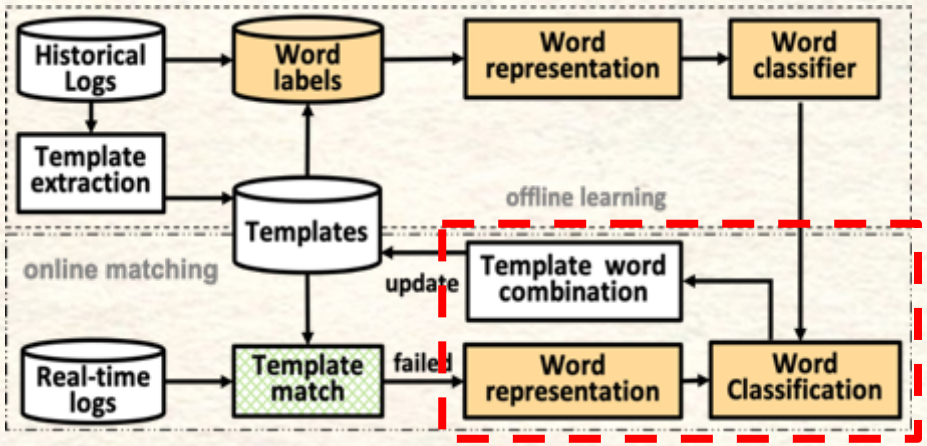
Template words

variable words

Template set:
T_1. Interface *, changed state to down
T_2. Vlan-interface *, changed state to down
T_3. Interface *, changed state to up
T_4. *,

Template match:
L_1->T_1 ,ae3 L_2->T_2 ,vl22
L_3->T_3 , ae3 L_4->T_1,ae1

log parsing problem → word classification problem



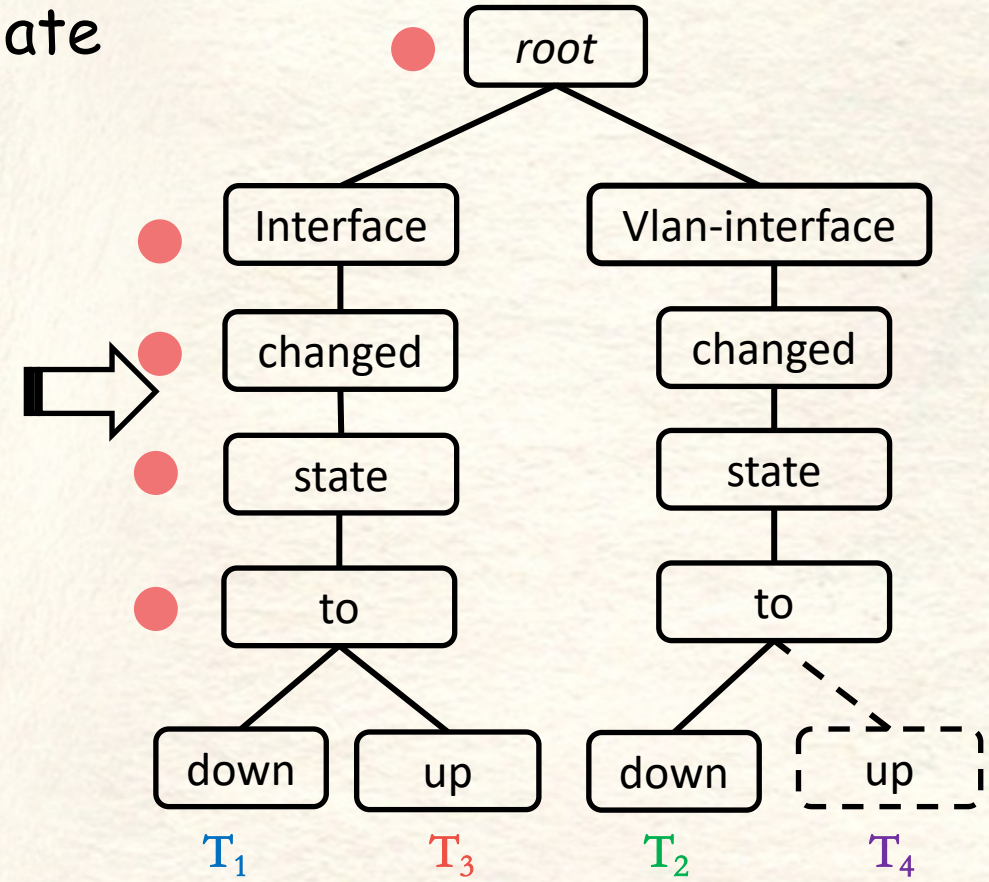
Template matching

- Build a prefix-tree for template matching
- Each root-to-leaf path is a template

Templates:
T_1. Interface *, changed state to down
T_2. Vlan-interface *, changed state to down
T_3. Interface *, changed state to up

Template update:
T_4. Vlan-interface *, changed state to up

Online log:
Interface ae3, changed state to down



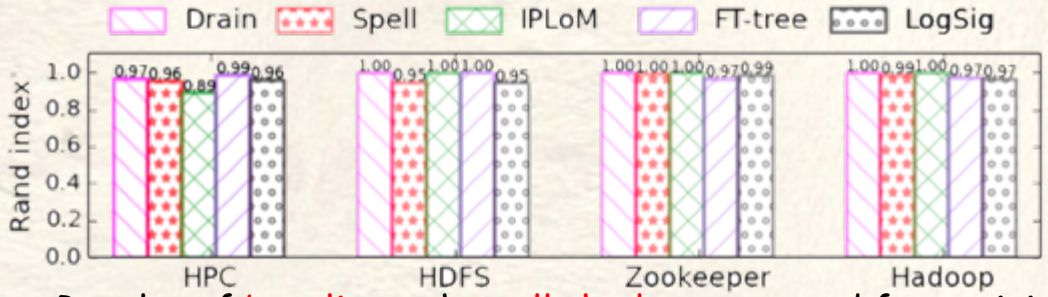
Datasets & Baselines

- Datasets:

Datasets	Description	# of logs
HPC	High performance cluster	433,489
HDFS	Hadoop distributed file system	11,175,629
ZooKeeper	ZooKeeper service	74,380
Hadoop	Hadoop MapReduce job	394,308

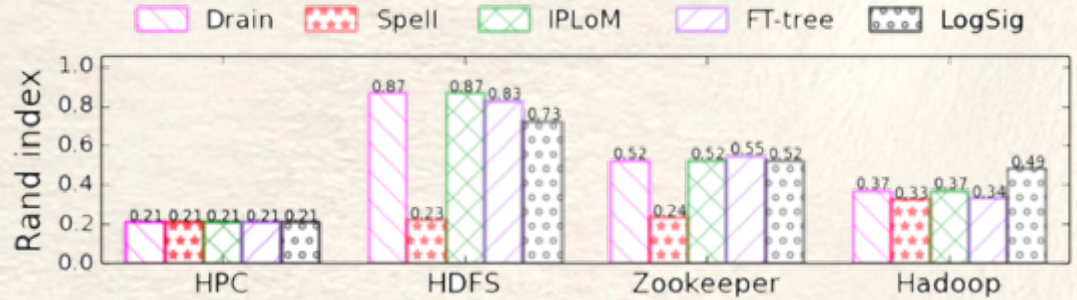
- Baselines:
  - Drain (ICWS'17), FT-tree (IWQoS'18) which claimed to support template update
  - LogSig (CIKM'11), Spell (ICDM'16), IPLoM (KDD'09) don't support template update

Evaluation on Intra-service adaptiveness



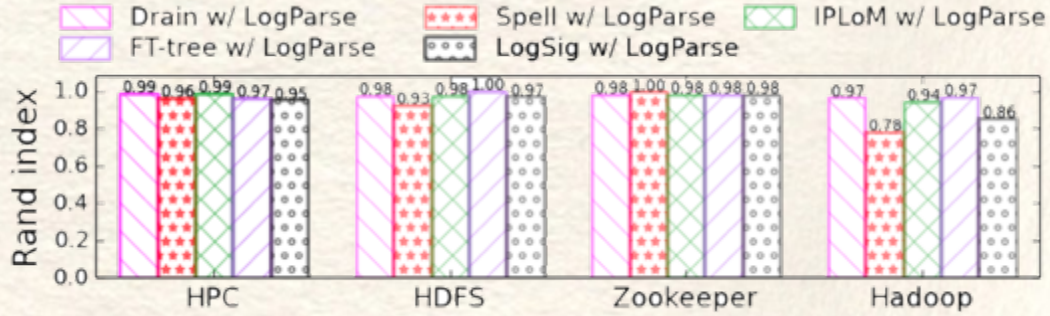
Results of baselines when all the logs are used for training

All baselines perform good in offline stage



Results of baselines when only 10% of logs are used for training

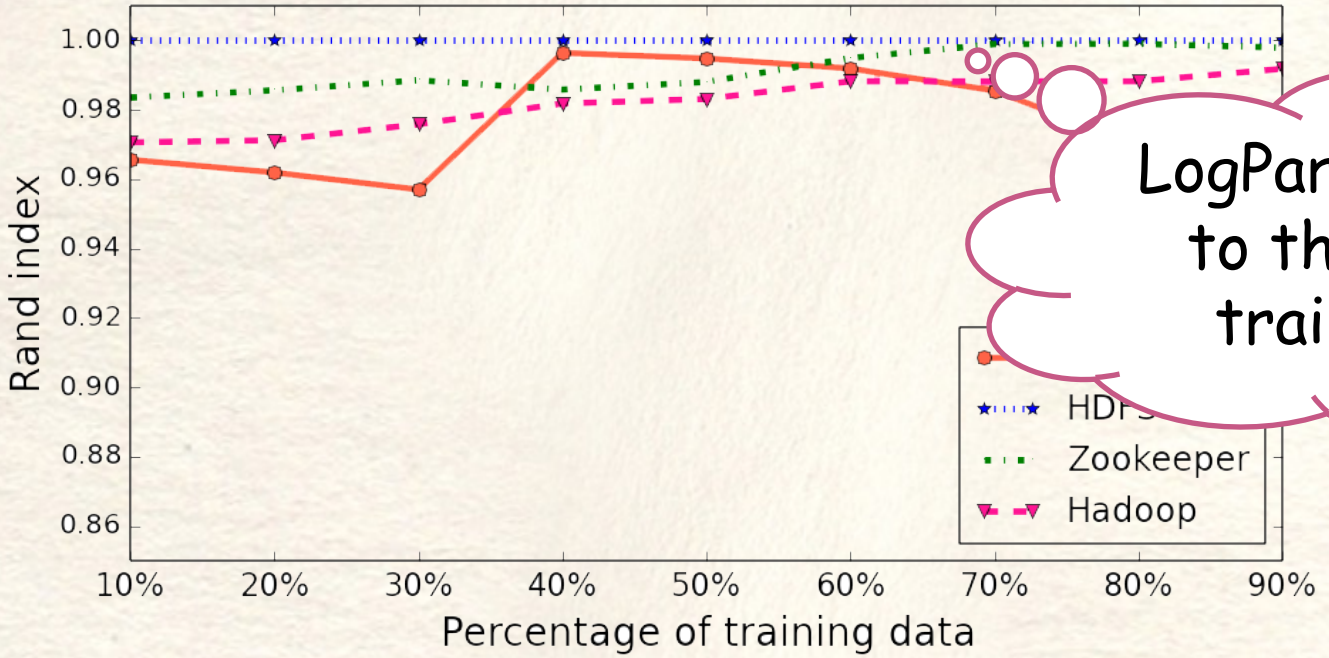
All baselines perform bad for online matching and update



Results of LogParse when only 10% of logs are used for training

Accuracy of LogParse is even higher than baselines trained by all logs.

Evaluation on stability



LogParse is stable to the scale of training data

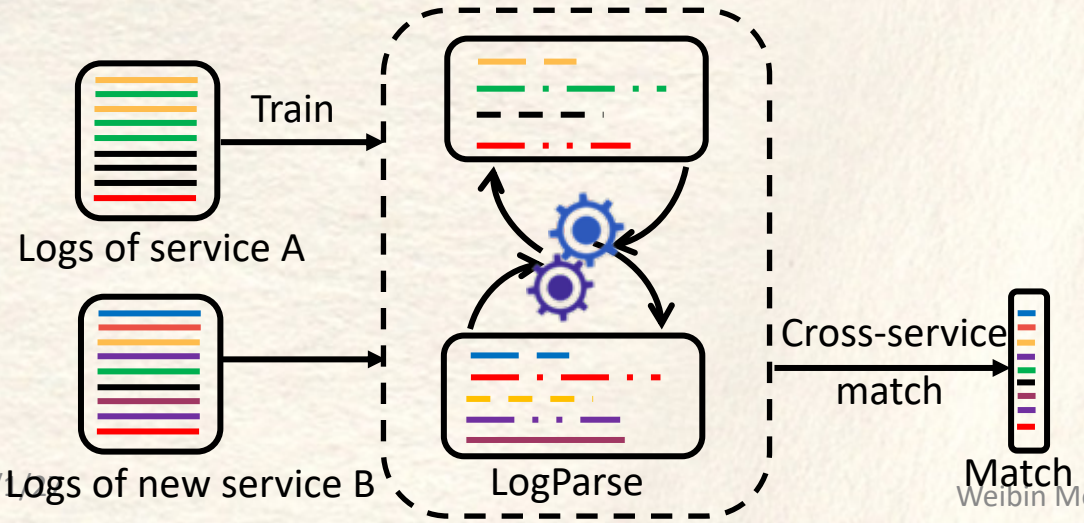
template accuracy as the percentage of training data increases from 10% to 90%

Evaluation on cross-service adaptive

Training data (service A)	Testing data (service B)			
	HPC	HDFS	Zookeeper	Hadoop
HPC	-	0.983	0.999	0.923
HDFS	0.982	-	0.993	0.974
ZooKeeper	0.993	1.0	-	0.937
Hadoop	0.983	0.999	0.999	-

Trained by

Match to



On average, LogParse achieves a cross-service accuracy of 0.980

Evaluation on compression

Type	Method	HPC	HDFS	Zookeeper	Hadoop	Average
Shore-term storage	LogParse	13.0%	14.0%	19.6%	4.6%	12.8
	bzip	9.6%	17.4%	9.7%	6.4%	10.8%
	7zip	9.7%	18.1%	9.4%	5.9%	10.8%
	zip	11.4%	20.9%	10.1%	7.2%	12.4%
Long-term storage	LogParse+bzip	1.4%	2.1%	2.4%	1.2%	1.8%
	LogParse+7zip	2.3%	2.6%	[illegible]	[illegible]	[illegible]
	LogParse+zip	2.2%	2.6%	[illegible]	[illegible]	[illegible]

LogParse is helpful to log compression

Conclusion

LogParse, an adaptive log parsing method

- Intra-service
- Cross-service

Log compression, an application of LogParse

- Assign template for any given log

An open-source toolkit

THANKS

Q&A

mwb16@mails.tsinghua.edu.cn

Toolkit: https://github.com/WeibinMeng/LogParse