



# Detecting Large-Scale System Problems by Mining Console Logs

Wei Xu\* Ling Huang<sup>†</sup>

Armando Fox\* David Patterson\* Michael Jordan\*

\*UC Berkeley

<sup>†</sup> Intel Labs Berkeley

# Why console logs?

- Detecting problems in large scale Internet services often requires detailed instrumentation
- Instrumentation can be costly to insert & maintain
  - High code churn
  - Often combine open-source building blocks that are not all instrumented
- Can we use console logs in lieu of instrumentation?
  - + Easy for developer, so nearly all software has them
  - Imperfect: not originally intended for instrumentation



Parse  
Detect



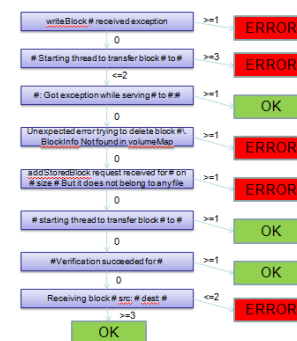
```

Block Index (from 1)
blk_09409127041117410090
blk_559437526778770626
blk_4793343704465889935
blk_1589623562745254692
blk_4675567517221500599
blk_4510506055782028641
blk_7791459630923742010
blk_6518953162876851443
blk_2342980358925460265
blk_7243640621214951509
blk_1078842198851715009
blk_6850648019882791636
blk_3487431885974825349
blk_5541402214405611694
blk_357805717325649806
blk_6719194466846743828
blk_186955280298731284
blk_3090708128539938366
blk_6537172909640915994
blk_19167650158843573
blk_575096239701914177
blk_798188546227669579
blk_1437957991444103370
    
```

```

3896 blk_559437526778770...
3897
unique:2 3896
manual label:1
comments: writeBlock received exception java.io.IOException
blk_559437526778770626
2 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0
ABNORMAL
dn_10_251_107_49 | 080722 172537 7609 INFO dfs.DataNode$DataXceiver: Receiving bloc
nn_10_251_210_161 | 080722 172537 29 INFO dfs.FSNamesystem: BLOCK* NameSystem
dn_10_251_193_205 | 080722 172539 7644 INFO dfs.DataNode$DataXceiver: Receiving blo
dn_10_251_107_49 | 080722 172541 7609 INFO dfs.DataNode$DataXceiver: writeBlock blk,
=====
abnormal
writeBlock received exception java.io.IOEX
    
```

Visualize



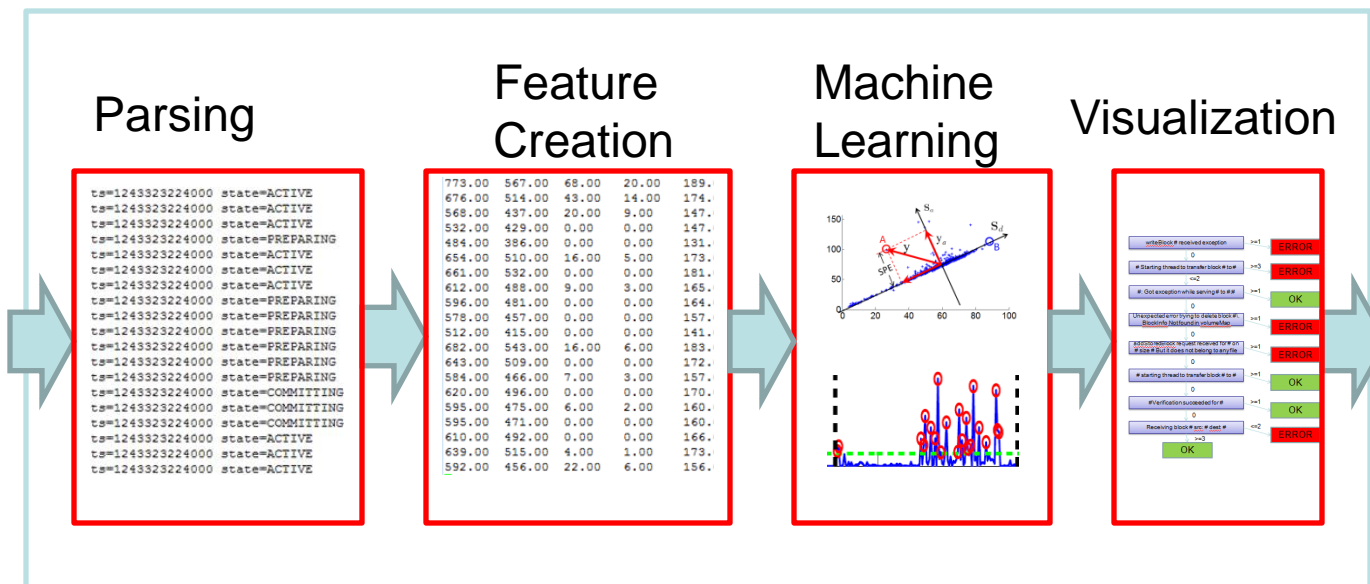
200 nodes,  
>24 million lines of logs

Abnormal log segments

A single page visualization

- Fully automatic process without any manual input

# Our approach and contribution



- A general methodology for processing console logs automatically
- Validation on two real systems

# Key insights for analyzing logs

- The log contains the necessary information to create features
  - Identifiers
  - State variables
  - Correlations among messages

receiving blk\_1  
received blk\_1

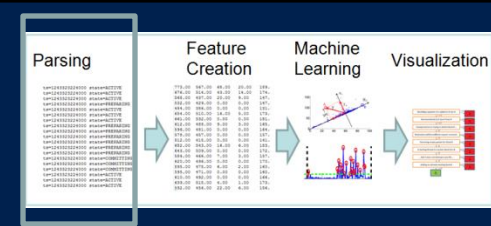
NORMAL

receiving blk\_2

ERROR

- Console logs are inherently structured
  - Determined by log printing statement

# Step 1: Parsing



- Free text → semi-structured text
- Basic ideas

Receiving block blk\_1

Log.info("Receiving block " + blockId);

Receiving block (.\*) [blockId]

Type:

Receiving block

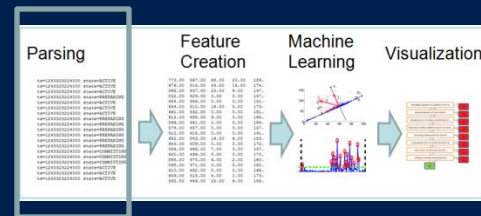
Variables:

blockId(String)=blk\_1

- Non-trivial in object oriented languages
  - Needs type inference on the entire source tree
- Highly accurate parsing results



# Step 2: Feature creation - Message count vector



- Identifiers are widely used in logs
  - Variables that identify objects manipulated by the program
  - file names, object keys, user ids
- Grouping by identifiers
  - Similar to execution traces
- Identifiers can be discovered automatically

receiving blk\_1  
receiving blk\_2  
receiving blk\_1  
received blk\_2  
received blk\_1  
received blk\_1  
receiving blk\_2



# Feature creation – Message count vector example

- Numerical representation of these “traces”
  - Similar to *bag of words* model in information retrieval

Receiving blk\_1

Receiving blk\_1

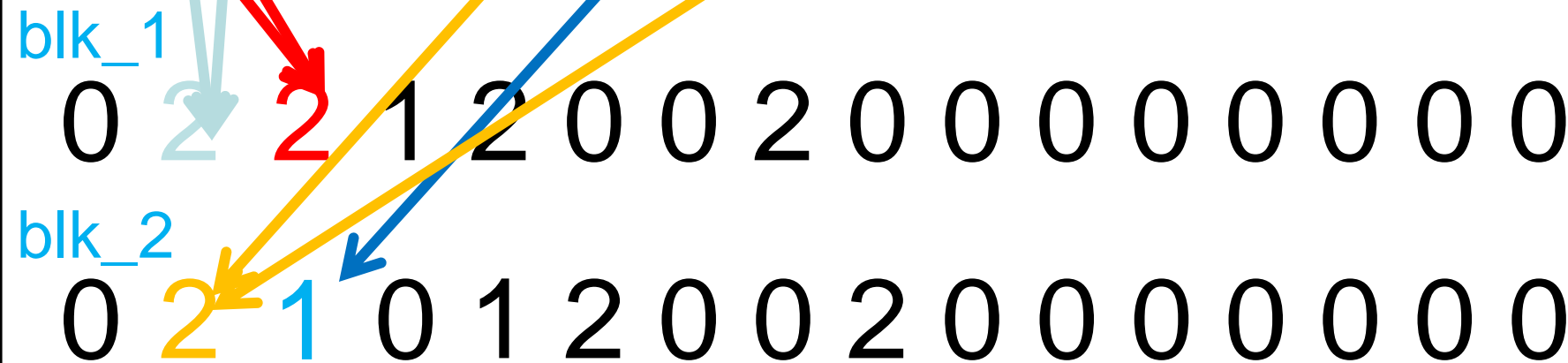
Received blk\_1

Received blk\_1

Receiving blk\_2

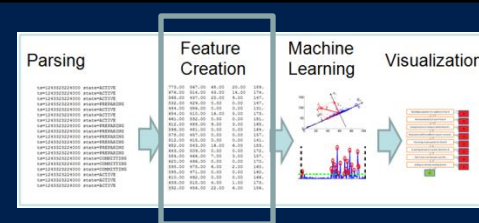
Received blk\_2

Receiving blk\_2

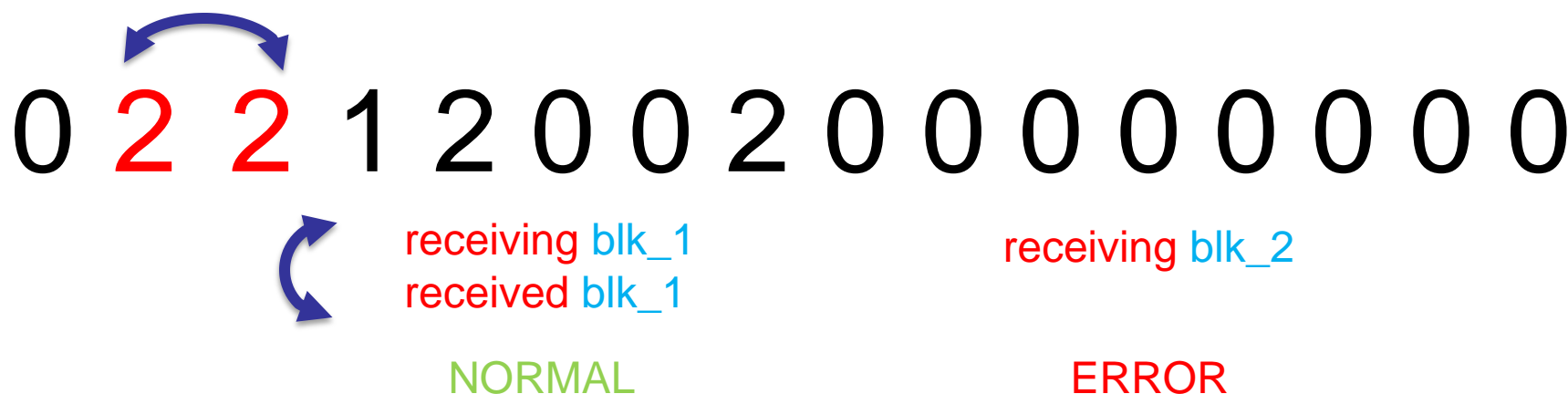




# Step 3: Machine learning – PCA anomaly detection



- Most of the vectors are normal
- Detecting abnormal vectors
  - Principal Component Analysis (PCA) based detection
  - PCA captures normal patterns in these vectors
- Based on *correlations* among dimensions of the vectors



# Evaluation setup

- Experiment on Amazon's EC2 cloud
  - 203 nodes x 48 hours
  - Running standard map-reduce jobs
  - ~24 million lines of console logs
  - ~575,000 HDFS blocks
- 575,000 vectors
- ~ 680 distinct ones
- Manually labeled each distinct cases
  - Normal/abnormal
  - Tried to learn why it is abnormal
  - For evaluation only

# PCA detection results

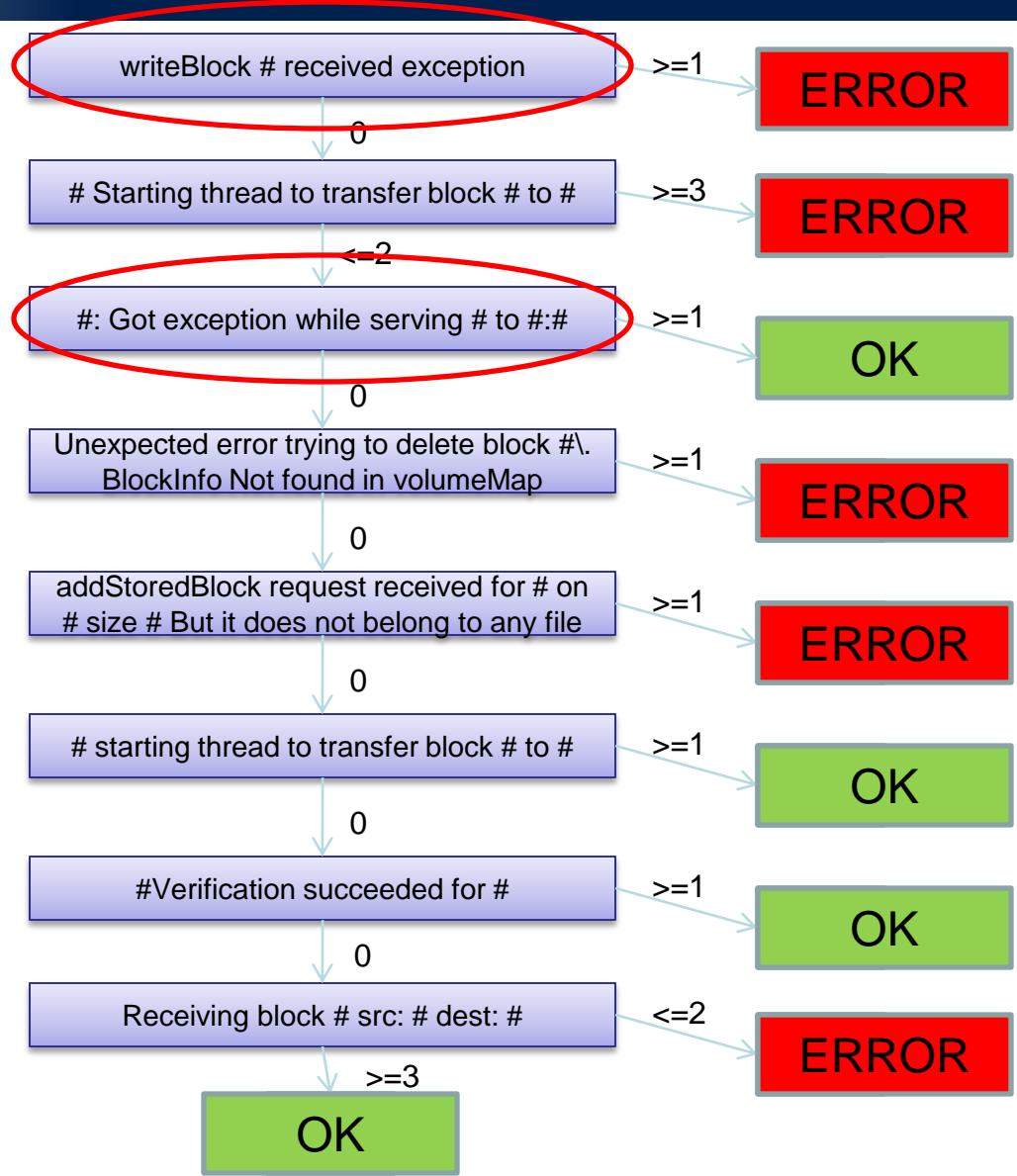
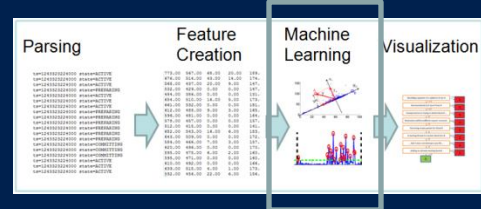
Anomaly Description		Actual	Detected
1	Forgot to update namenode for deleted block	4297	4297
2	Write block exception then client give up	3225	3225
3	Failed at beginning, no block written	2950	2950
4	Over-replicate-immediately-deleted	2809	2788
5	Received block that does not belong to any file	1240	1228
6	Redundant addStoredBlock request received	953	953
7	Trying to delete a block, but the block no longer exists on data node	724	650
8	Empty packet for block	476	476
9	Exception in receiveBlock for block	89	89
10	PendingReplicationMonitor timed out	45	45
11	Other anomalies	108	107
<b>Total anomalies</b>		<b>16916</b>	<b>16808</b>
<b>Normal blocks</b>		<b>558223</b>	

## False Positives

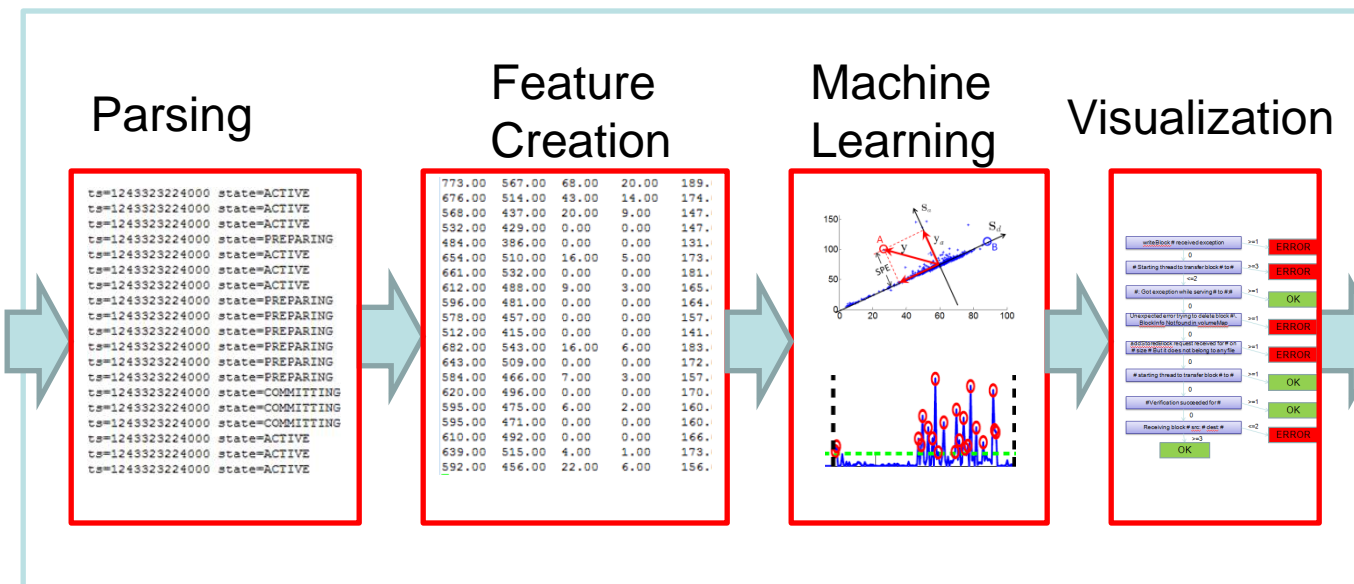
Description	False Positives
1 Normal background migration	1397
2 Multiple replica ( for task / jobdesc files )	349
<b>Total</b>	<b>1746</b>

How can we make the results easy for operators to understand?

# Step 4: Visualizing results with decision tree



- Parsing
  - Extract templates from program binaries
  - Support more languages
- Feature creation and machine learning
  - Allow online detection
  - Cross application/layers logs



<http://www.cs.berkeley.edu/~xuw/>  
 Wei Xu <xuw@cs.berkeley.edu>