RAD Lab
UC Berkeley

# Detecting Large-Scale System Problems by Mining Console Logs


Wei Xu* Ling Huang†

Armando Fox* David Patterson* Michael Jordan*

*UC Berkeley † Intel Labs Berkeley

# Why console logs?

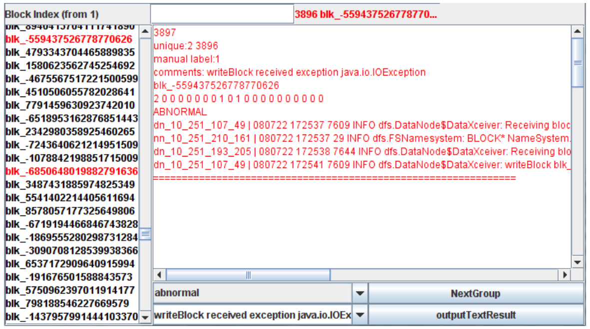
- Detecting problems in large scale Internet services often requires detailed instrumentation
- Instrumentation can be costly to insert & maintain
  - High code churn
  - Often combine open-source building blocks that are not all instrumented
- Can we use console logs in lieu of instrumentation?

+ Easy for developer, so nearly all software has them

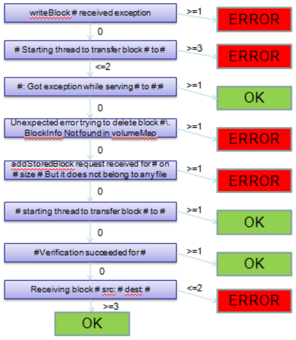
– Imperfect: not originally intended for instrumentation

# Result preview

Parse
Detect




Visualize



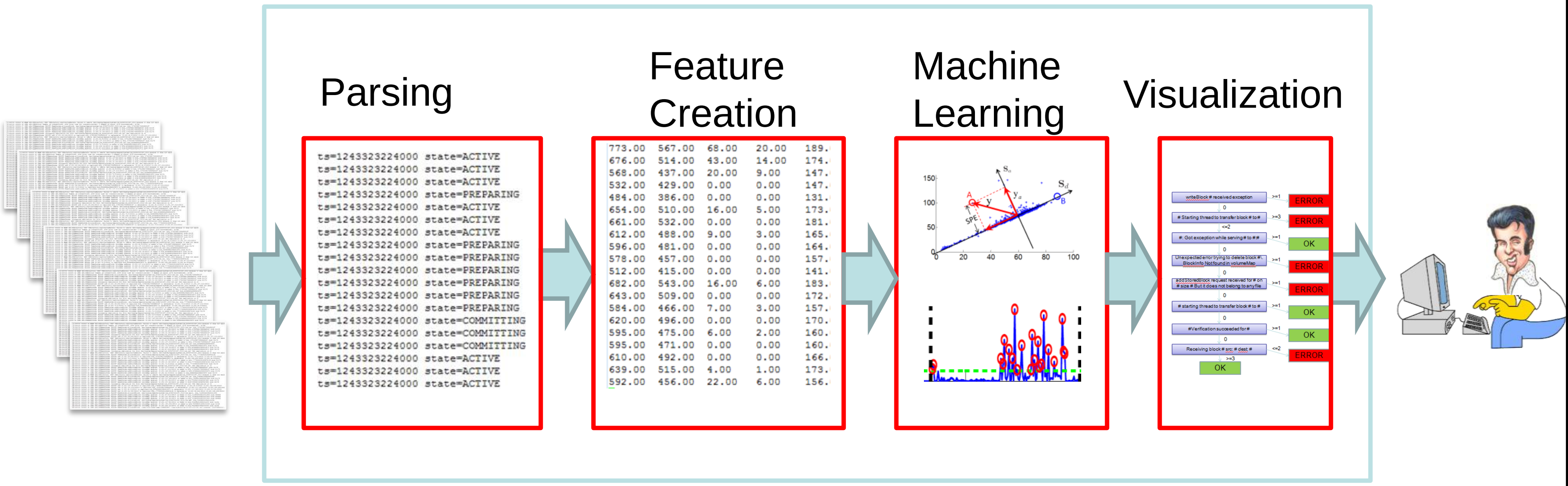

200 nodes,
>24 million lines of logs

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

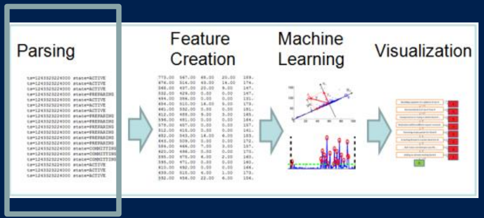
NORMAL

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

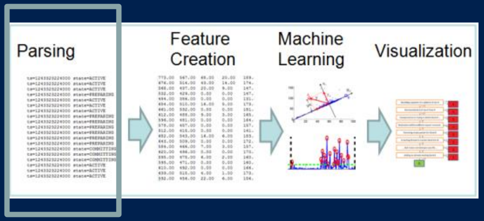
Log.info("Receiving block " + blockId);

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

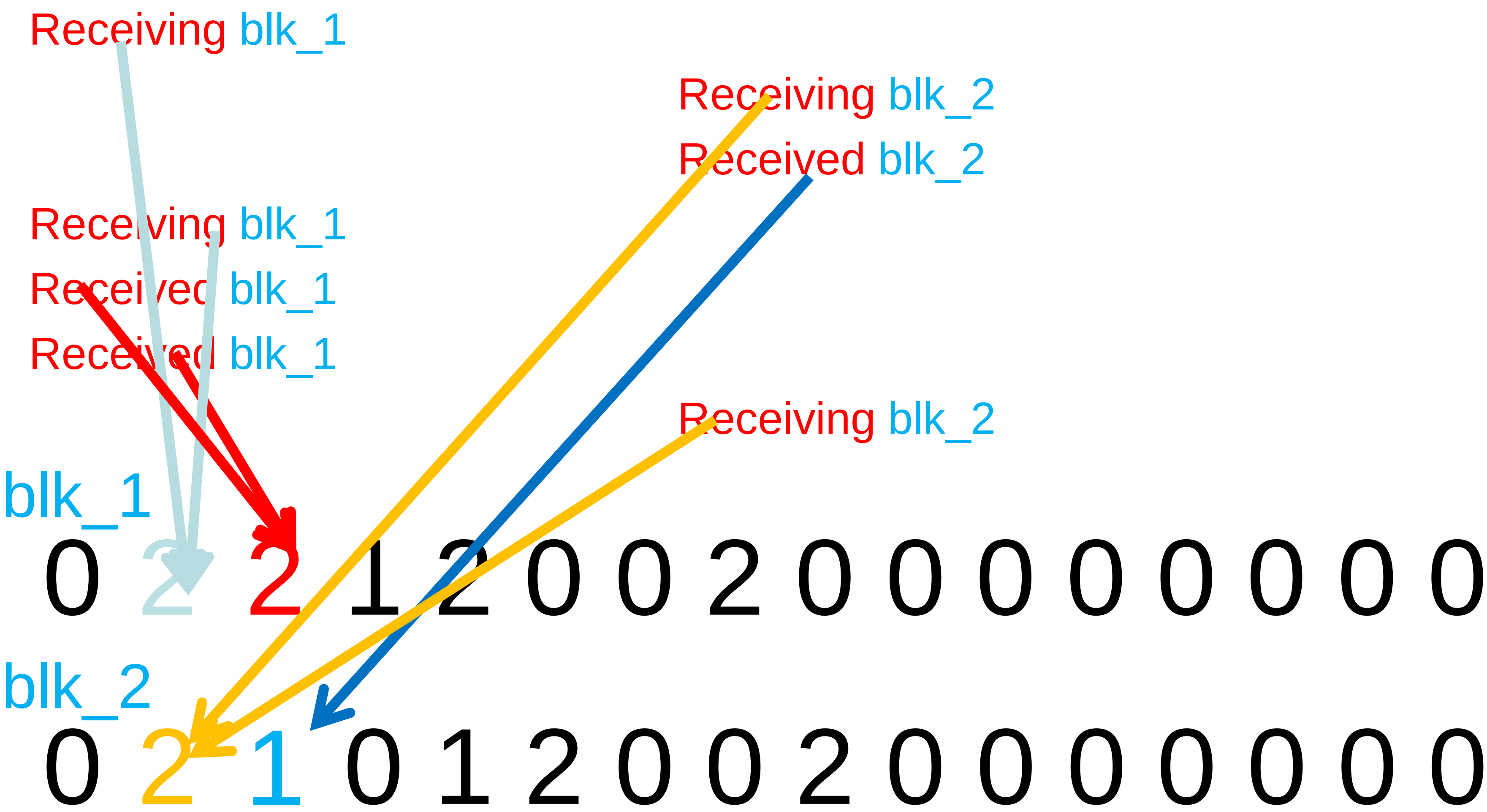



- Identifiers are widely used in logs
  - Variables that identify objects manipulated by the program
  - file names, object keys, user ids
- Grouping by identifiers
  - Similar to execution traces
- Identifiers can be discovered automatically

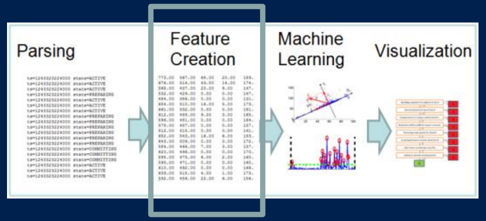
receiving blk_1
receiving blk_2
receiving blk_1
received blk_2
received blk_1
received blk_1
receiving blk_2

# Feature creation – Message count vector example

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



# 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

0 2 2 1 2 0 0 2 0 0 0 0 0 0 0 0

receiving blk_1
received blk_1

NORMAL

receiving blk_2

ERROR

# 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

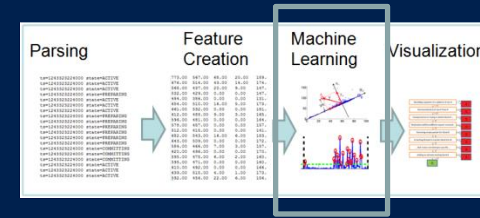
| | Anomaly Description | Actual | Detected |
|---|---|---|---|
| 1 | Forgot to update namenode for deleted block | 4297 | 4297 |
| 2 | Write block exception then client give up | 3225 | 3225 |
| 3 | Failed at beginning, no block written | 2950 | 2950 |
| 4 | Over-replicate-immediately-deleted | 2809 | 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 |
| | **Total anomalies** | **16916** | **16808** |
| | **Normal blocks** | **558223** | |

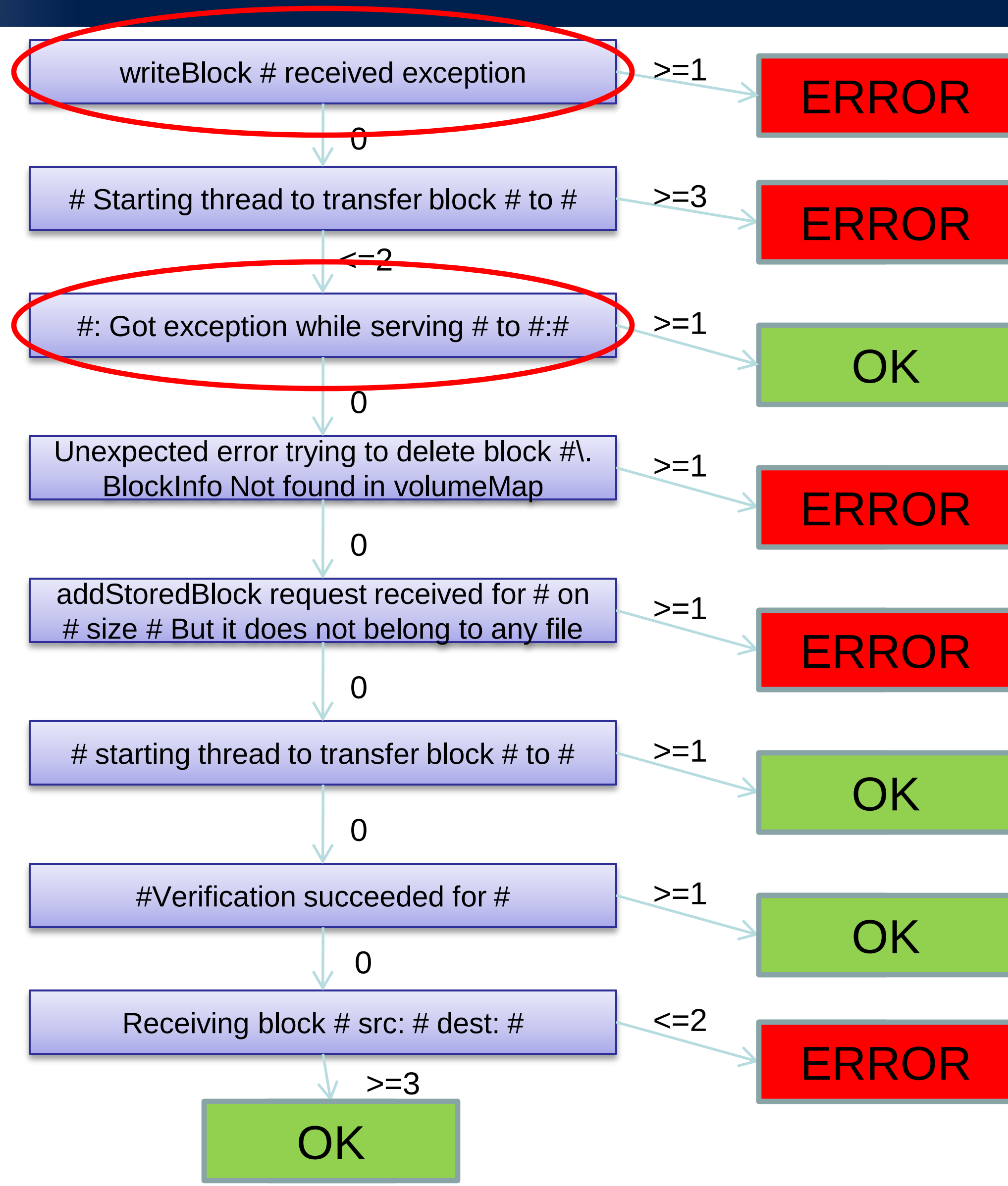
**False Positives**

| | Description | False Positives |
|---|---|---|
| 1 | Normal background migration | 1397 |
| 2 | Multiple replica ( for task / jobdesc files ) | 349 |
| | **Total** | **1746** |

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

# Step 4: Visualizing results with decision tree



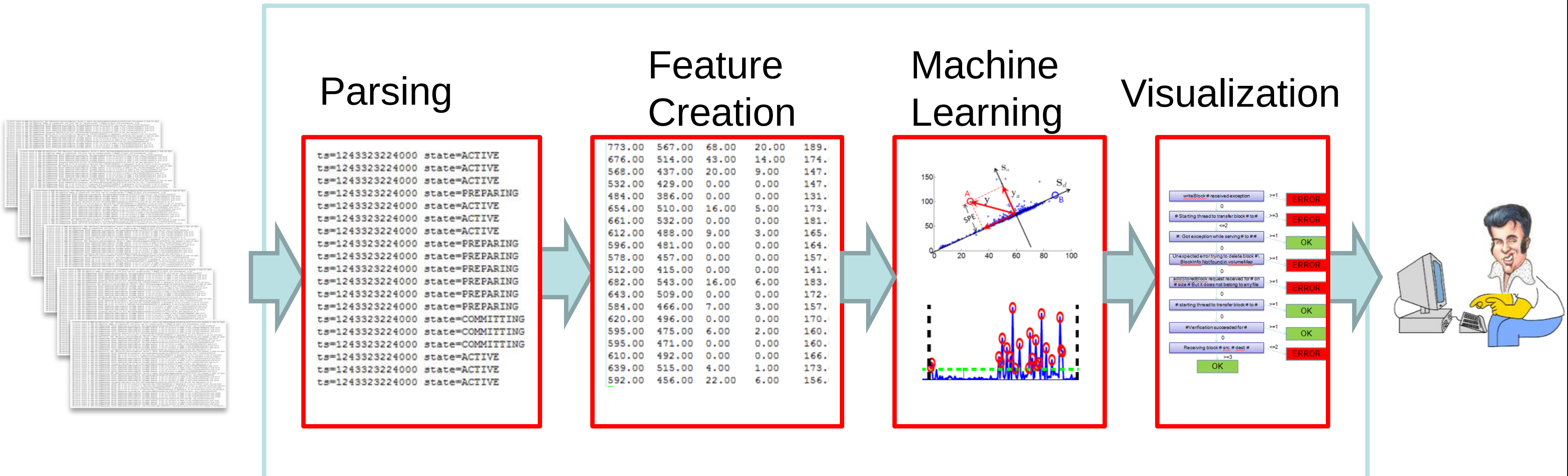



# Future work

- Parsing
  - Extract templates from program binaries
  - Support more languages

- Feature creation and machine learning
  - Allow online detection
  - Cross application/layers logs

# Summary




http://www.cs.berkeley.edu/~xuw/

Wei Xu <xuw@cs.berkeley.edu>