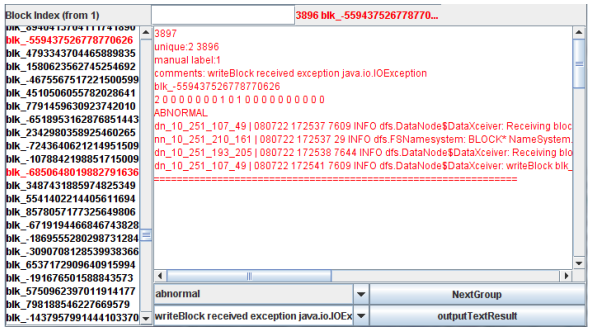
Mining Console Logs for Large-Scale System Problem Detection

Wei Xu Ling Huang
Armando Fox David Patterson Michael Jordan

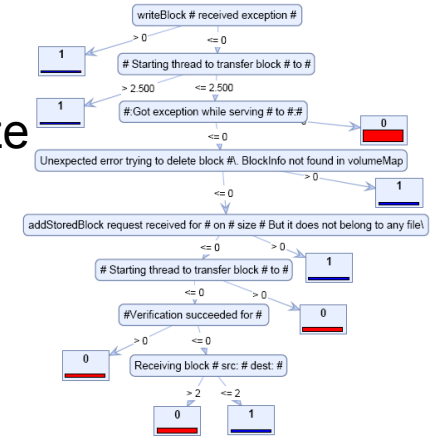
TexPoint fonts used in EMF

Make sense out of console logs

Extract
Detect



Visualize



200 nodes,
>24 million lines of logs

abnormal log segments

A single decision tree
to visualize system behavior

Motivation - useful but ignored

- Console logs are useful
  - In almost every software system
  - Hand-picked information by developers
  - Expressive, convenient to use
  - Especially in large scale Internet services
    - Open source code + in-house development
    - Continuously changing system
- But they are ignored
  - Console logs are intended for a single developer
  - Assumption: log writer == log reader
  - Today many developers => massive textual logs

Console logs are ignored because they are hard to read

Human

– Verbose
– Awkward language
– Different levels of implementation details

HODIE NATUS EST RADICI FRATER*

today unto the Root a brother is born.

Machine

– Highly unstructured, looks like free text

* "that crazy Multics error message in Latin."
http://www.multicians.org/hodie-natus-est.html

Problem: Don't know what to look for!

Goal and key observations

- Discover the most interesting log messages without any prior input
- Recover log structure from source code analysis
  - Console logs were intrinsically structured
    - Determined by log printing statement
  - Constant strings = markers of message structure
  - Source code is usually available
- Message groups (and correlations among messages) more likely to reveal problems
  - Many ways to group related log messages
  - i.e. not just by time

Approach - Extract and mine structured information

Source Code / Program binary

Msg Template:
Creating file (.*)
[filename][string]

… …
Creating file mydata
Wrote file mydata, size 23453674
Creating file junkfile
Backing up file mydata to 10.0.0.1
Done bk-up file mydata, status=OK
… …

Msg Type
1
2
1
3
4

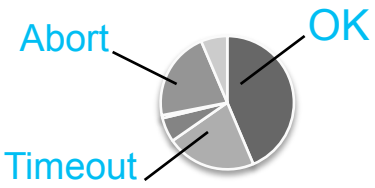
Step 1: Extract Structures

Message Type Variables

Step 2: Create Features

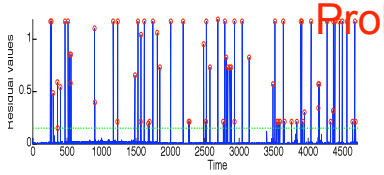
mydata → 1 1 1 1 0 0 0 0 0
junkfile → 1 1 0 1 0 0 0 2 0
… 1 1 1 1 0 0 0 0 0

Message Count Vector



Status Ratio

Step 3: Mining Features with ML

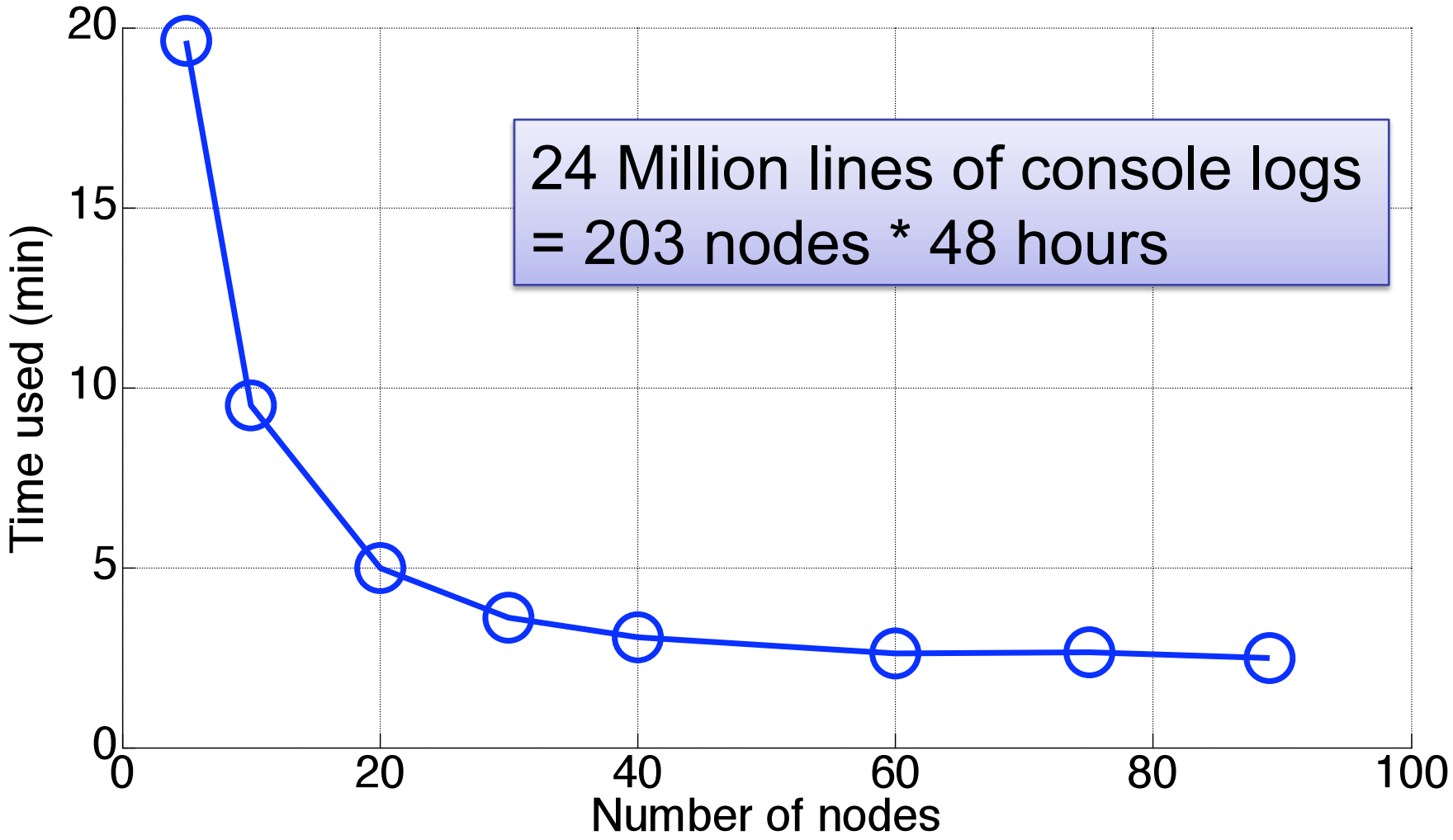


PCA Anomaly Detection

Case studies

- Surveyed a number of software apps
  - Linux kernel, DHCP, OpenSSH …
  - Apache, MySQL, Jetty, Lucene …
  - Hadoop, Cassandra, Nutch …
- In this talk – Hadoop file system
  - Per block operation logging (usually ignored)
- Experiment on EC2 cloud
  - 203 nodes * 48 hours
  - ~ 300 TB HDFS data (550,000 blocks)
  - ~24 million lines of console logs

Step 1: Log parsing
Scale log parsing with map-reduce



Step 2: Feature - Message count vector

- Try to capture per operation behavior
- Find *identifiers* == message variables that
  - Have ***many*** distinct values
  - Appear in multiple message types
  - Reported many times
- Group these messages by identifiers => *message group*
- Count of distinct message types in each group
  - Similar to *Bag of word*s model in IR
- One message count vector per identifier value

Message count vector example

datanode_r16 | Receiving block blk_100 src: … dest:...

namenode_r10 | allocateBlock: blk_100

namenode_r10 | allocateBlock: blk_200

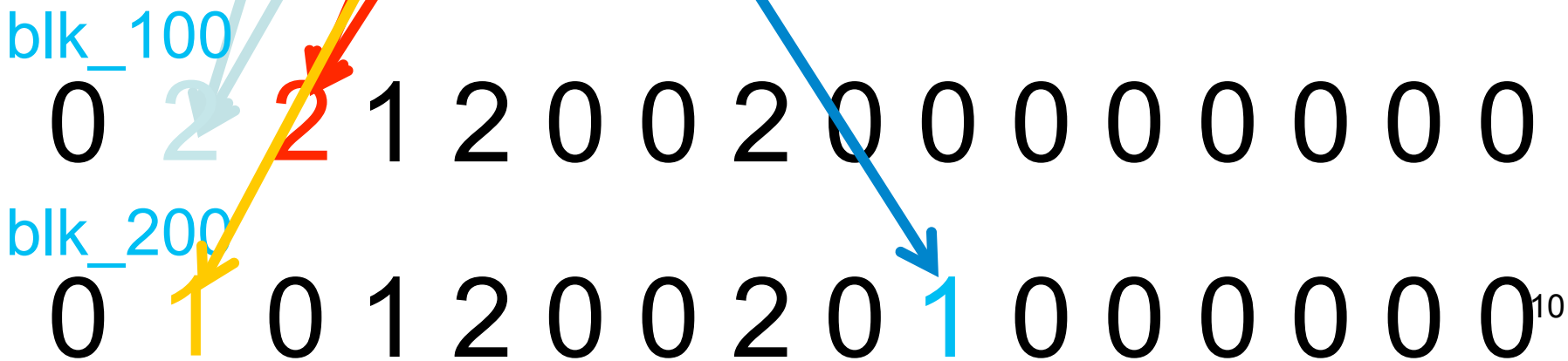
datanode_r16 | Receiving block blk_200 src: … dest:...

datanode_r14 | Receiving block blk_100 src: …dest:…

datanode_r16 | Received block blk_100 of size 49486737 from …

datanode_r14 | Received block blk_100 of size 49486737 from …

datanode_r16 | Error Receiving block blk_200 of size 49486737 from …



Step 3: Mining PCA detection and improvement

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

55,000 vectors, one per block

- What to find abnormal block operations
- Dimensions highly correlated
  - Unusual correlations indicate abnormal execution paths
  - PCA separates normal pattern from abnormal, making anomalies easy to detect
- Feature construction analogous to “bag of word’’ model in IR
  - Applying tf/idf + cosine similarity significantly improves results

PCA detection results

Seq	Event Description	Actual	RAW	TF-IDF
1	Forgot to update namenode for deleted block	4297	475	4297
2	Write block exception then client give up	3225	3225	3225
3	Failed at beginning, no block written	2950	2950	2950
4	Over-replicate-immediately-deleted	2809	2803	2788
5	Received block that does not belong to any file	1240	20	1228
6	Redundant addStoredBlock request received	953	33	953
7	Trying to delete a block, but the block no longer exists on data node	724	18	650
8	Empty packet for block	476	476	476
9	Exception in receiveBlock for block	89	89	89
10	PendingReplicationMonitor timed out	45	37	45
11	Other anomalies	108	91	107
	Total	**16916**	**10217**	**16808**

Seq	Description	RAW	TF-IDF
1	Normal background migration	1399	1397
2	Multiple replica (for task / jobdesc files)	372	349
	Total	**1771**	**1746**

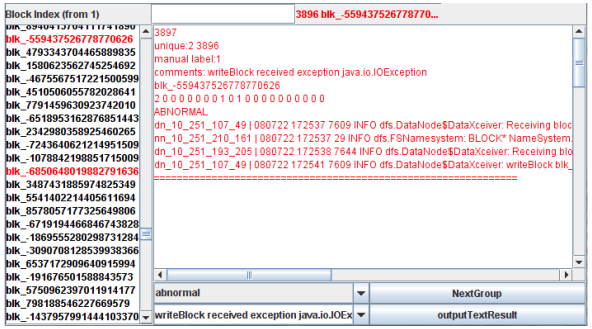
Explaining detection results with decision tree

Future Work

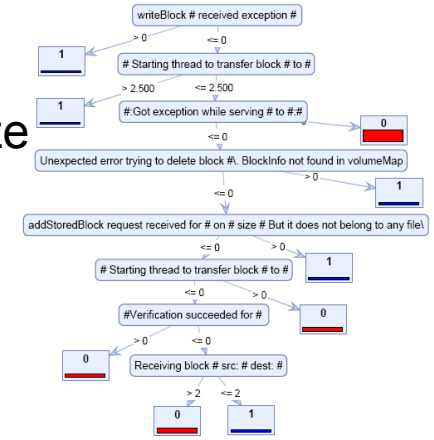
- More production logs (can you help?)
- Machine learning
  - Cross application logs
  - More features (esp. console log specific features)
  - Multiple sources learning
  - Allow operator feedback (semi-supervised learning)
  - Allow online detection
- System
  - Support C programs + Linux binary (or data driven..)
  - Make open source project
- Suggestions?

Summary

Extract
Detect



Visualize



200 nodes,
>24 million lines of logs

abnormal log segments

A single decision tree
to visualize system behavior

Backup slides