

Mining Invariants from Logs for System Problem Detection

Jian-Guang LOU, Qiang FU

Software Analytics

Microsoft Research Asia

Background

Many systems produce log messages for trouble-shooting.

- Logs usually record important system actions or events for trouble shooting, and can reflect the execution paths of a program
- Logs are used to detect problems:
 - Work flow errors -- there are errors occur in the execution paths;
 - Low performance problems -- the execution time or the loop number is much larger than normal cases.

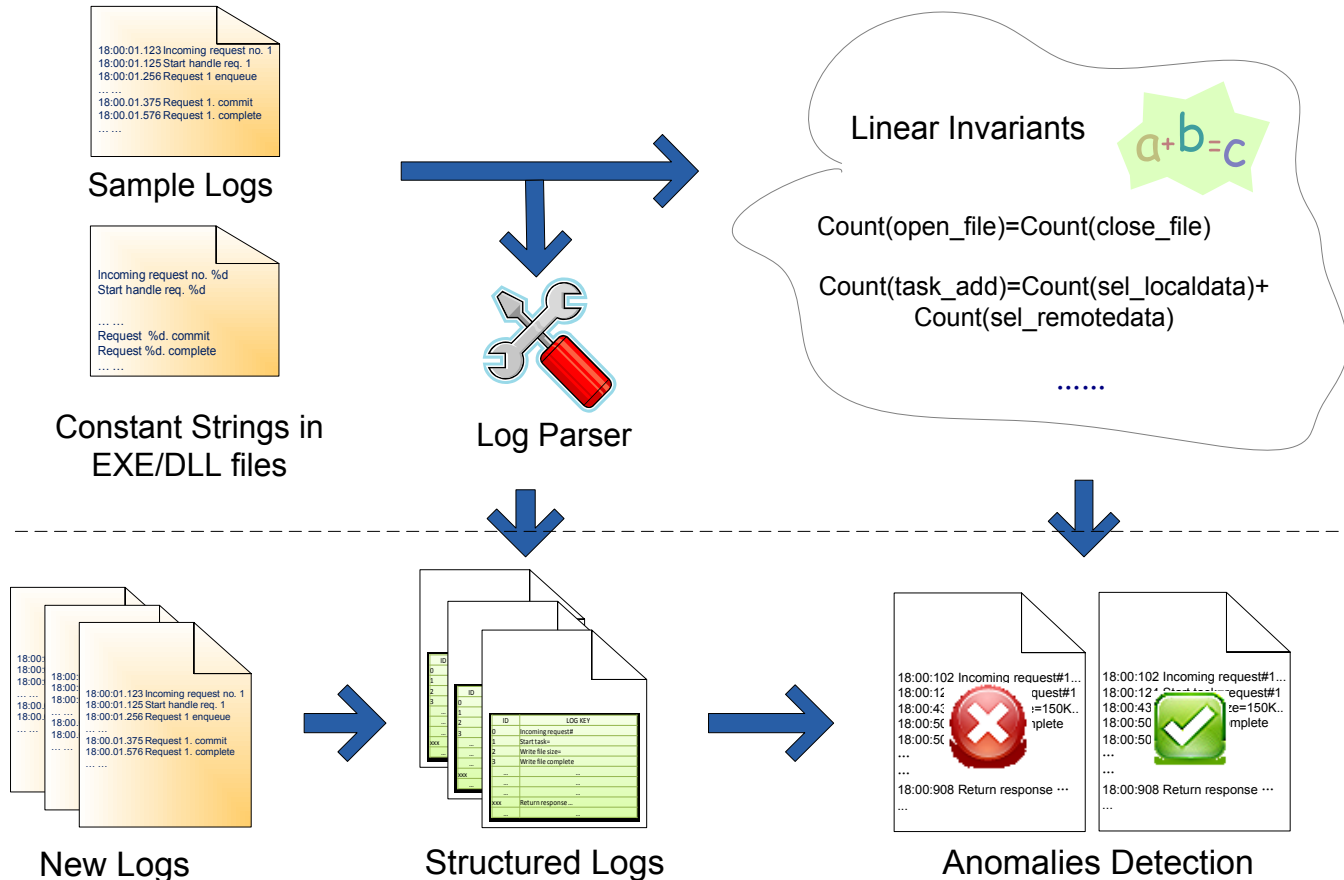
Problems

- Manually inspecting logs is not feasible
 - Large scale of system
 - High complexity of system
- Traditional rule/keyword based log analysis tools:
 - Heavily depend on the knowledge of operators
 - Difficult to keep rules updated when components are frequently revised or upgraded

Automatic Log Analysis

- Statistical model based methods:
 - treat a log sequence as a feature vector
 - [Xu et al. 2009]: Mine console logs for large-scale system problem detection based on PCA analysis.
 - [Mirgorodskiy et al. 2006]: Use string edit distance to categorize logs and detect anomalies.
- Behavior model based methods:
 - view a log sequence as a program work flow
 - [Tan et al. 2008]: Learn and visualize control flow models from Hadoop logs based on some pre-defined log tokens.
 - [Cotroneo et al. 2007]: Derive work flow models for a Java VM.

Our Basic Idea

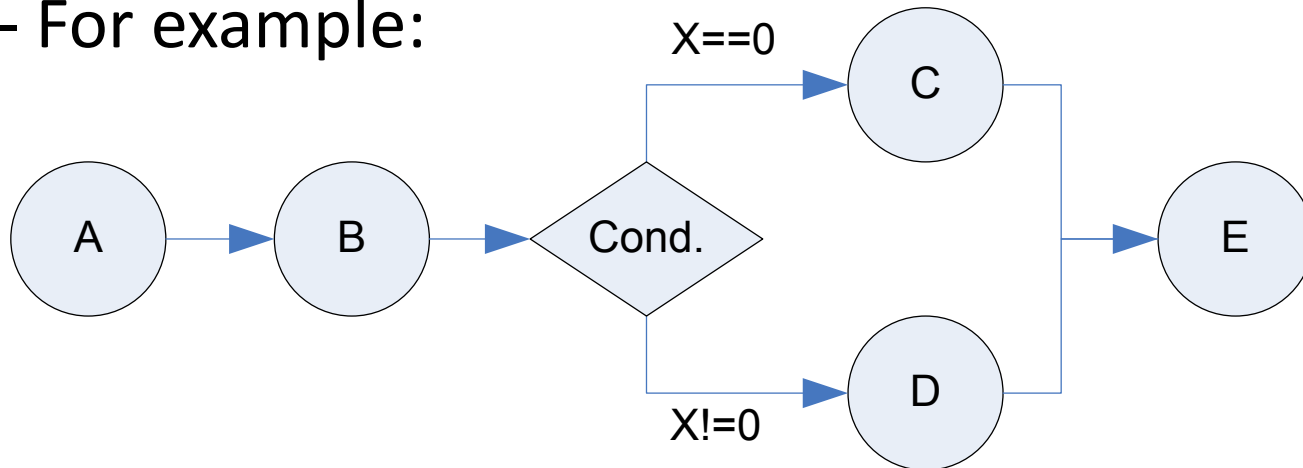


Normal behaviors can be learned from logs, and then be used to detect anomalies.

Linear Program Invariant

- A predicate always holds the same value under different normal executions.

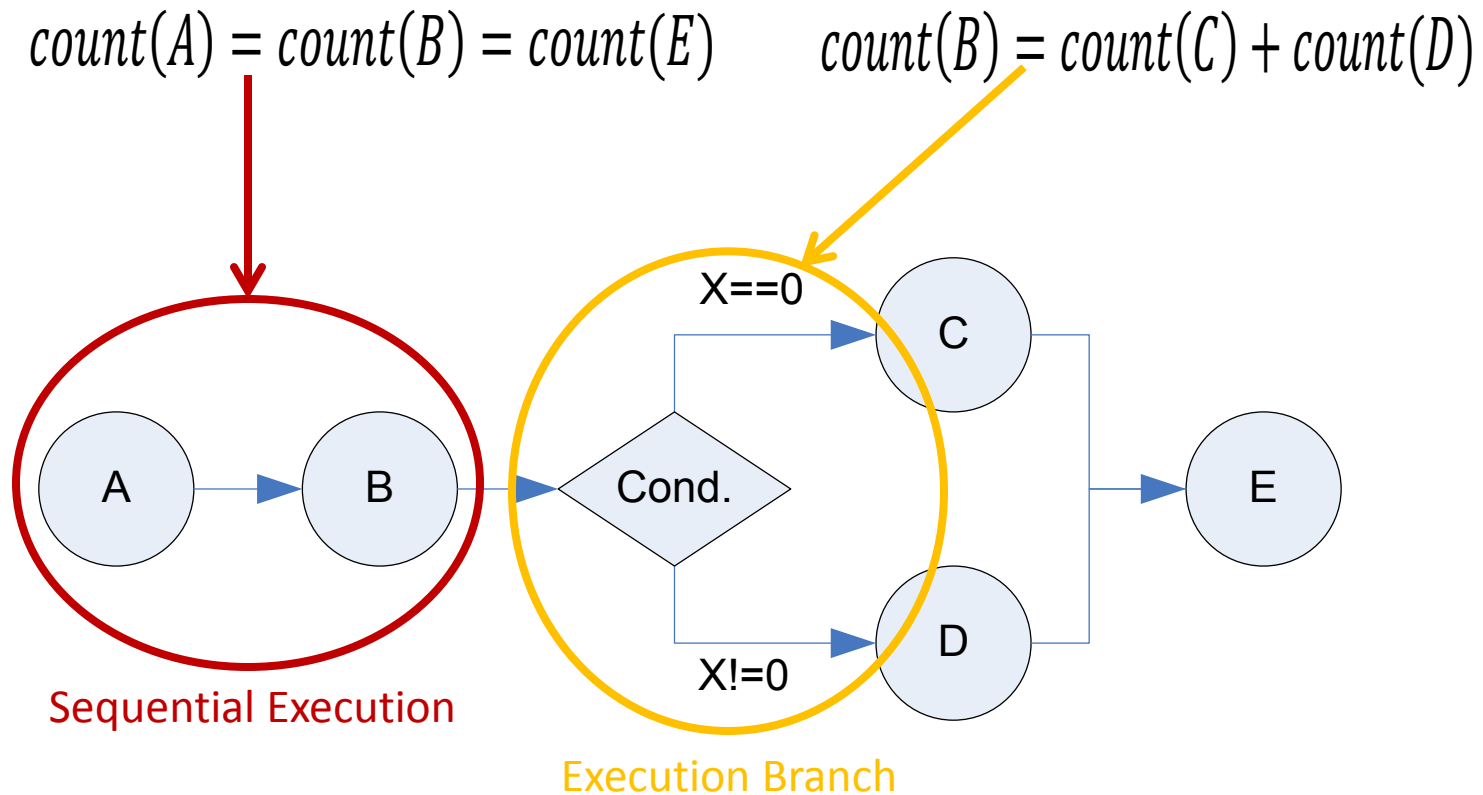
– For example:



$$\text{count}(A) = \text{count}(B) = \text{count}(E)$$

$$\text{count}(B) = \text{count}(C) + \text{count}(D)$$

Invariant and Execution Path



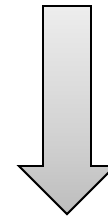
Linear invariants reflect the properties of execution path.

Invariant Violation and Anomaly(1)

- A violation of invariant often indicates a system problem.



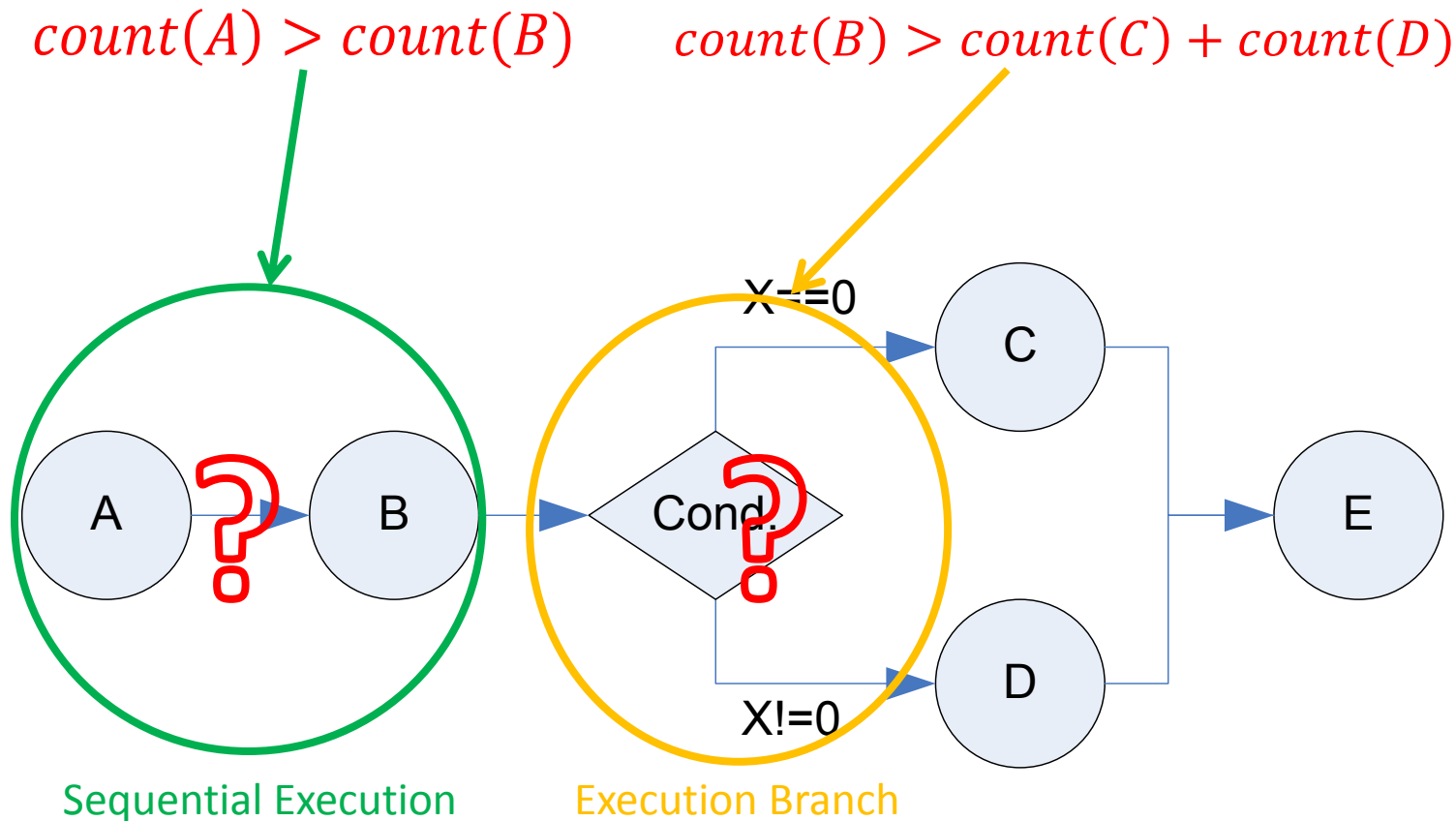
count(Enter) \neq count(Leave)



**Problem
on Critical Section Operations**

Invariant Violation and Anomaly(2)

- Violated invariants often give diagnosis cues.



Formulation of Invariant

- A linear invariant can be presented as a linear equation:

$$a_0 + a_1x_1 + a_2x_2 + \cdots + a_mx_m = 0$$

where x_i is the message count of message i .

- Given a set of logs, we have

$$\mathbf{X}\theta = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1m} \\ 1 & x_{21} & x_{22} & \ddots & x_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \theta = 0$$

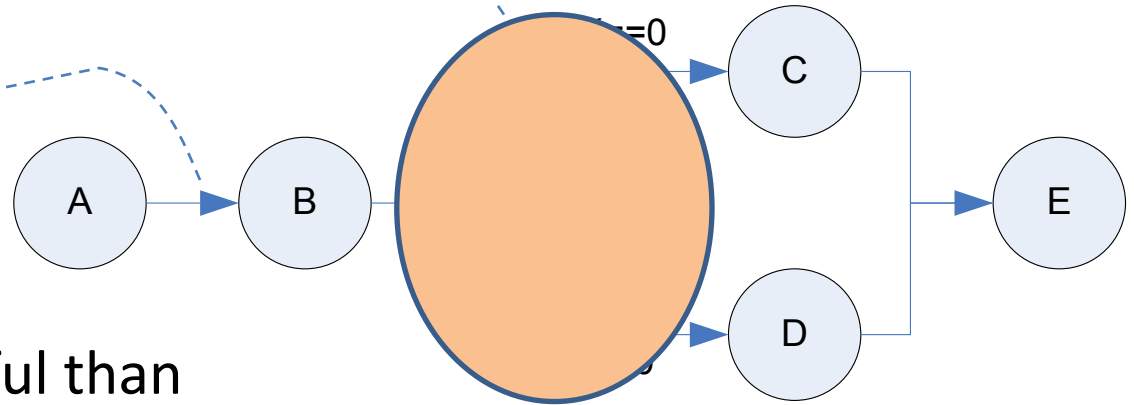
where $\theta = [a_0, a_1, a_2, \cdots, a_m]^T$

What Is A Meaningful Invariant?

-- Sparse Non-zero Coefficients

$$c(B) = c(C) + c(D)$$

$$c(A) = c(B)$$



are more meaningful than

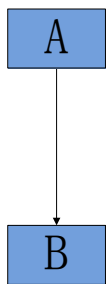
$$c(A) + 3c(B) - 2c(E) - 2c(C) - 2c(D) = 0$$

Any vector in the Null Space of X is an invariant;
Only sparse invariants are interested.

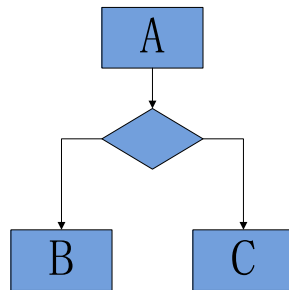
What Is A Meaningful Invariant?

-- Integer Coefficients

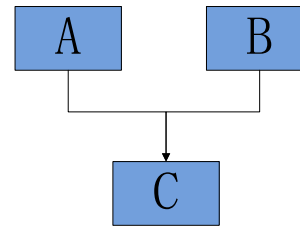
Elementary work flow structures can be interpreted by integer invariants.



Sequential



Branch



Join

... ..

Integer invariants are easy to be understood by human operators.

Problem Statement

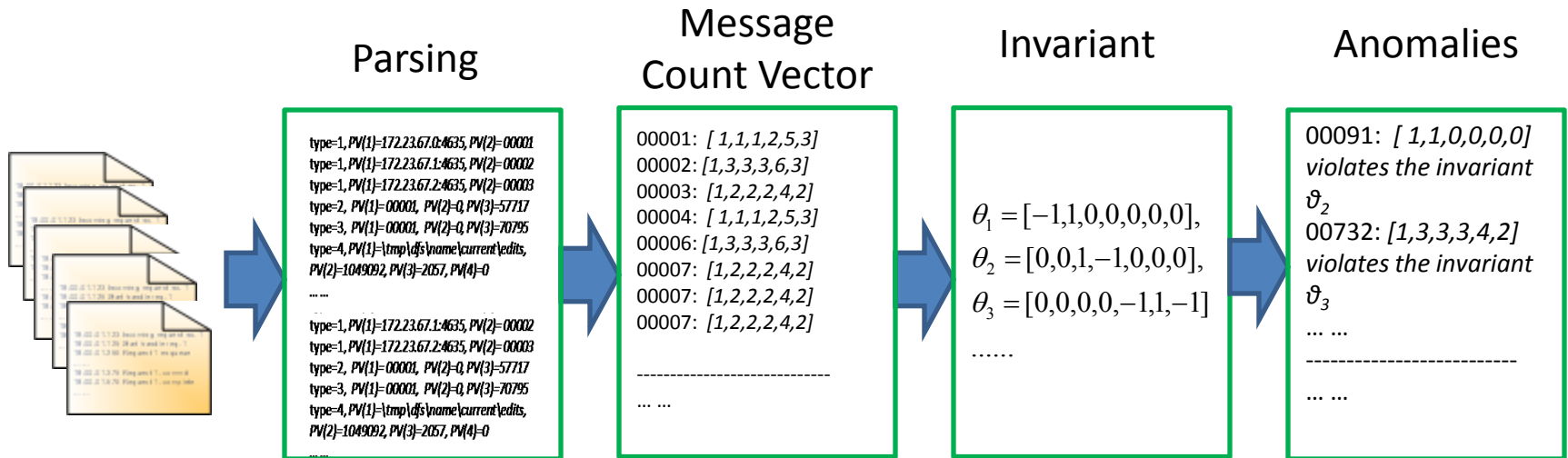
- Due to noise pollution, mining invariants is to find integer sparse solutions of regression.

$$\mathbf{X}\theta = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1m} \\ 1 & x_{21} & x_{22} & \ddots & x_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \theta = 0 \quad \longrightarrow \quad \mathit{arg\ min} \|\mathbf{X}\theta\|_0$$

– Challenges:

- A typical integer sparse regulation problem (NP-Hard)
- Traditional method is to relax 0-norm to 1-norm. However, it cannot guarantee to find all invariants.

Learning Invariant Overview

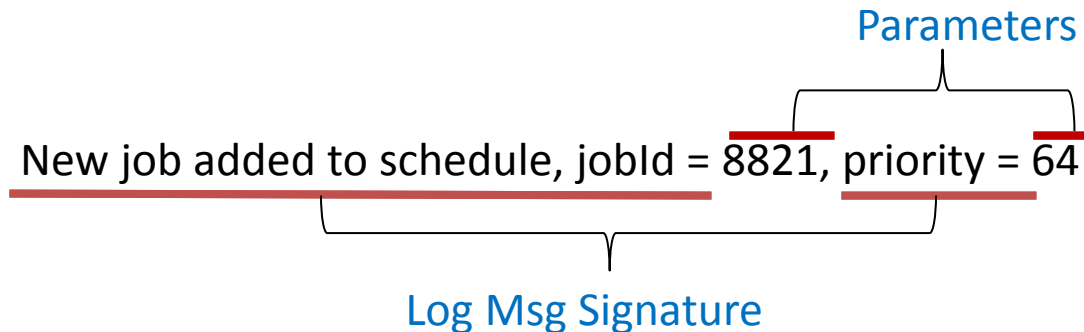


Four Steps:

Log parsing, Message Grouping and Counting,
Search Invariants, and Anomaly Detection

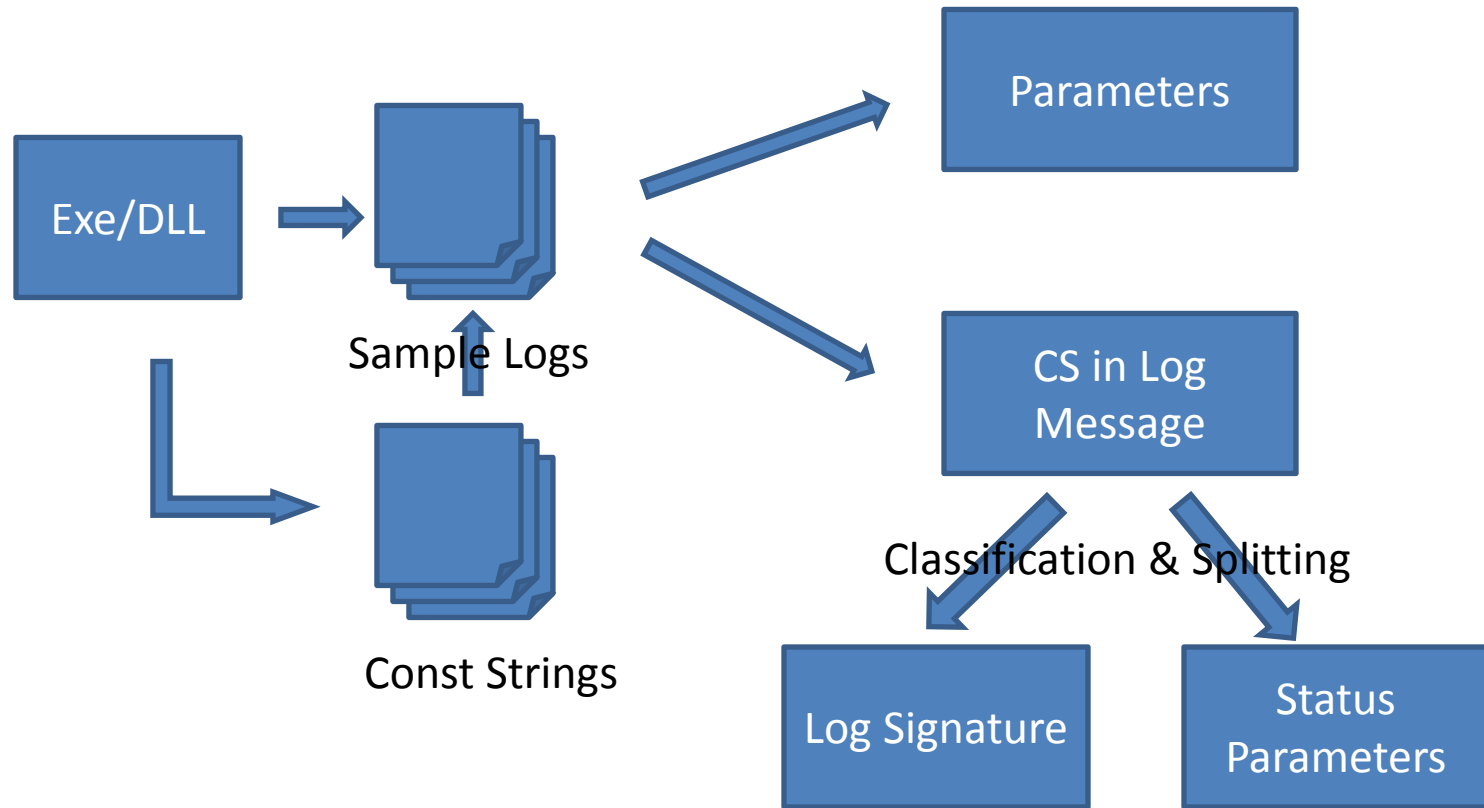
Step 1: Log Parsing

- Goal: Free text logs → structured logs



- Basic idea:
 - Log messages of the same message type usually have a high similarity.
 - Words of log message signatures are often embedded as constant strings in DLL/EXE files (e.g. symbols).

Parsing: Our Solution



Limitation:

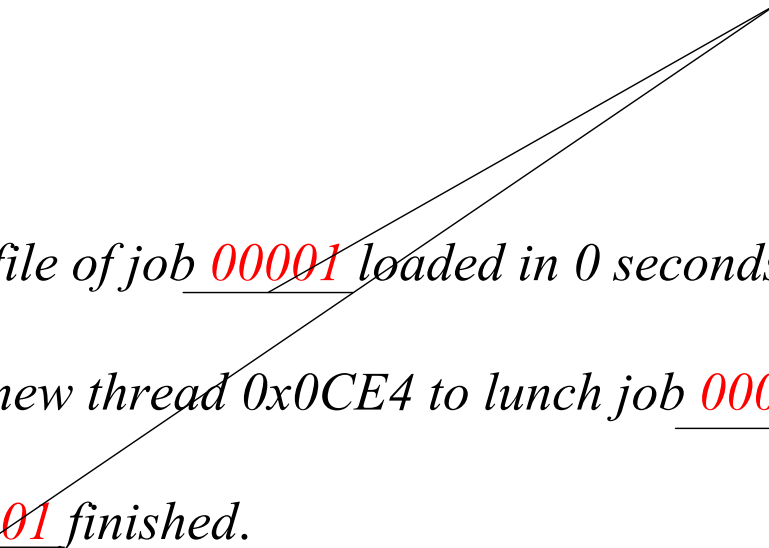
Coverage depends on the sample logs.

Step 2: Message Grouping

-- Cogenetic Parameters

- Cogenetic parameters: parameters record the value of the same system variable.

Job_ID



18:51:05.767 *Image file of job 00001 loaded in 0 seconds, size 57717.*

18:51:06.048 ...

18:51:06.329 *Start a new thread 0x0CE4 to lunch job 00001.*

18:51:06.501 ...

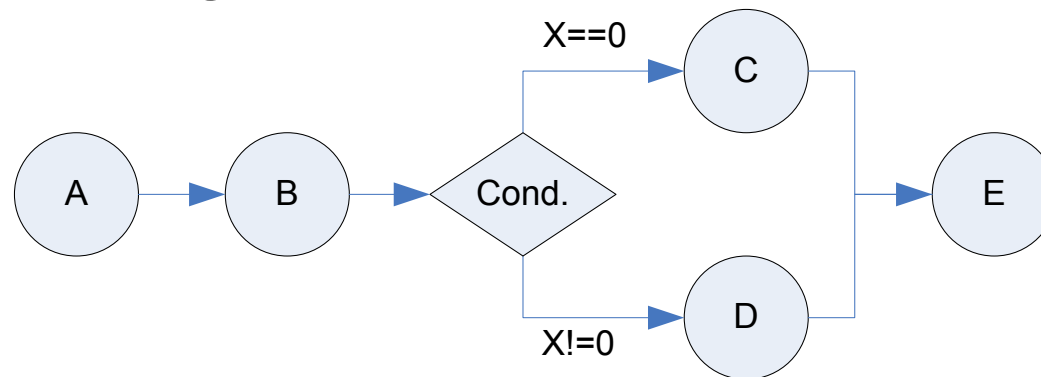
18:51:06.658 *Job 00001 finished.*

18:51:06.673 *Image file of job 00002 loaded in 0 seconds, size 70795.*

...

Detecting Cogenetic Parameter Groups

- Two parameters are cogenetic, if
 - (1) they have the same value set(e.g. $A=B$),
 - (2) one parameter's value set is a sub-set of the other's (e.g. $B \supseteq C$),
 - (3) there is a mid-parameter satisfying (2) with these two parameters. (e.g. $B \supseteq C \ \&\& \ B \supseteq D$)



- Each cogenetic parameter group corresponds a system variable.

Results of Parameter Grouping

- Testing on Hadoop logs, we detect the following meaningful program variables:
 - Map/Reduce Task ID, Map/Reduce Task Attempt ID, Block ID, and JVM ID, Storage ID, IP address and port, and write data size of task shuffling.
- Testing on CloudDB, we found program variables:
 - request ID, CASNode ID, replica operation ID, ...

Message Grouping

- Variables that identify objects are often recorded in logs, which can be used to group messages.
 - e.g. Request ID, task ID, ...

18:51:05.767 *Image file of **job 00001** loaded ...*

18:51:06.048 ...

18:51:06.329 *Start a new thread 0x0CE4 to lunch **job 00001**.*

18:51:06.433 *Image file of **job 00002** loaded in 0.seconds, size 70795.*

18:51:06.501 ...

18:51:06.629 *Start a new thread 0x0DE5 to lunch **job 00002**.*

18:51:06.701 ...

18:51:06.758 ***Job 00001** finished.*

...

18:51:06.927 ***Job 00002** finished.*

...

Message Count Vector

- Count number for each message type in a group.

```

18:51:06.048 ...
18:51:06.433 Image file of job 00002 loaded in
0 seconds, size 70795.
18:51:06.501 ...
18:51:06.629 Start a new thread 0x0DE5 to
lunch job 00002.
18:51:06.701 ...
...
18:51:06.927 Job 00002 finished.
...

18:51:05.767 Image file of job 00001 loaded ...
...
18:51:06.329 Start a new thread 0x0CE4 to
lunch job 00001.
...
...
...
...
18:51:06.758 Job 00001 finished.
...
...
...

```

...	Image loaded	Job lunch	Job finished	...
Log group 1	1	1	1	...
Log group 2	1	1	1	...

Step 3: Search Invariant

- A hypotheses and testing framework:
 - Try any combination of non-zero coefficients to construct invariant candidates,
 - Then validate whether they fit the log data.
- Computational cost $O\left(\sum_{i=1}^m c_m^i\right)$
 - ◆ How to reduce computational cost based on properties of log analysis?

Reduce Computational Cost(1)

- Divide and conquer: (message grouping)
 - Log messages often form some groups. There are strong correlations in the same group, and no obvious inter-group correlation.

$$o\left(\sum_{i=1}^m c_m^i\right) \rightarrow o\left(\sum_{i=1}^{m_1} c_{m_1}^i\right) + o\left(\sum_{i=1}^{m_2} c_{m_2}^i\right) + \dots$$

- Limit number of non-zero coefficients :
 - In most systems, the dimension of row space of \mathbf{X} is often very small (e.g. $r \leq 5$), the number of non-zero coefficients is very small [Xu2009].

$$o\left(\sum_{i=1}^m c_m^i\right) \rightarrow o\left(\sum_{i=1}^r c_m^i\right)$$

Reduce Computational Cost(2)

- Early termination:
 - According to linear algebra, there is at most k independent invariants, k is the dimension of X 's Null Space. We can early terminate the search process once we obtain k independent invariants.
- Skipping strategy:
 - A linear combination of invariants is also a valid invariant. We can skip the searching on the combinations of obtained invariants.

Results of Computational Cost Reduction

Our reduction strategy largely reduces the computational cost, especially when the dimension is large.

Message group of related object identifier	Full search space size	Our search space size
Hadoop logs with MapTask ID	128	37
Hadoop logs with ReduceTask ID	8	6
Hadoop logs with MapTask Attempt ID	268435456	3310
Hadoop logs with ReduceTask Attempt ID	33554432	730
Hadoop logs with JVM ID	128	16

Results of Discovered Invariants on Hadoop Log

Based on manually checking, we find all discovered invariants are reasonable, no false positive.

Message groups of related object identifiers	Invariants (≤ 3 coef.)	Invariants (≥ 4 coef.)
Hadoop logs with MapTask ID	3	0
Hadoop logs with ReduceTask ID	1	0
Hadoop logs with MapTask Attempt ID	21	3
Hadoop logs with ReduceTask Attempt ID	17	0
Hadoop logs with Data Block ID	9	0
Hadoop logs with JVM ID	5	0
Hadoop Logs with Storage ID	3	0
Logs with IP/port	4	0
Logs with task write packet size	1	0

Note: we also find invariants for CloudDB logs, but no ground truth for evaluation.

Results of Anomaly Detection on Hadoop Log

We compare our results with PCA based algorithm [Xu09]. Our method not only detects anomalies, but also gives the cue why they are abnormal.

Anomaly Description	PCA based Method	Our Method
Tasks fail due to heart beat lost.	397	779
A killed task continued to be in RUNNING state in both the JobTracker and that TaskTracker for ever	730	1133
Ask more than one node to replicate the same block to a single node simultaneously	26	26
Write a block already existed	25	25
Task JVM hang	204	204
Swap a JVM, but mark it as unknown.	87	87
Swap a JVM, and delete it immediately	211	211
Try to delete a data block when it is opened by a client	3	6
JVM inconsistent state	73	416
The pollForTaskWithClosedJob call from a Jobtracker to a task tracker times out when a job completes.	3	3

False Positive Description	PCA Method	Our Method
Killed speculative tasks	585	1777
Job cleanup and job setup tasks	323	778
The data block replica of Java execution file	56	0
Unknown Reason	499	0

A Detail Case

-- Anomaly of “a task JVM hang”

$c(\text{JVM spawned}) \neq c(\text{JVM exited})$



a JVM spawned
but did not exit



JVM was hung.

$c(\text{JVM spawned}) = c(\text{given task})$



A task was given
after a JVM spawned



JVM got hung
after it was assigned a task



Results on Production Programs

- CloudDB logs:
 - 266 invariants are learned from CASNode level logs;
 - 9 anomalies are detected in one test set and confirmed by a tester.
- SharePoint logs:
 - Detected anomalies are checked by developers;
 - About 78% anomalies are real errors.

Summary

- We proposed an automatic anomaly detection technique by mining linear invariants from logs including:
 - A method to automatically identify a set of parameters that correspond to the same program variable;
 - A method to discover sparse integer linear invariants from logs;
 - A method to detect anomalies based on invariants and gives intuitive cues for diagnosis.

THANKS!