Failure Prediction in Large-scale Computing Systems via Log Mining

Anwesha Das

27th September 2018

Today's Talk

- NEC Labs (NGLA: Next Generation Log Analytics)
  - LogMine – CIKM'16
  - LogLens – ICDCS'18 (Industry Track), uses LogMine
- My work: Node Failures on HPC platform (Cray Supercomputers)
  - Aarohi – Online Failure Prediction
  - RCA – Root Cause Analysis of Compute Node Failures

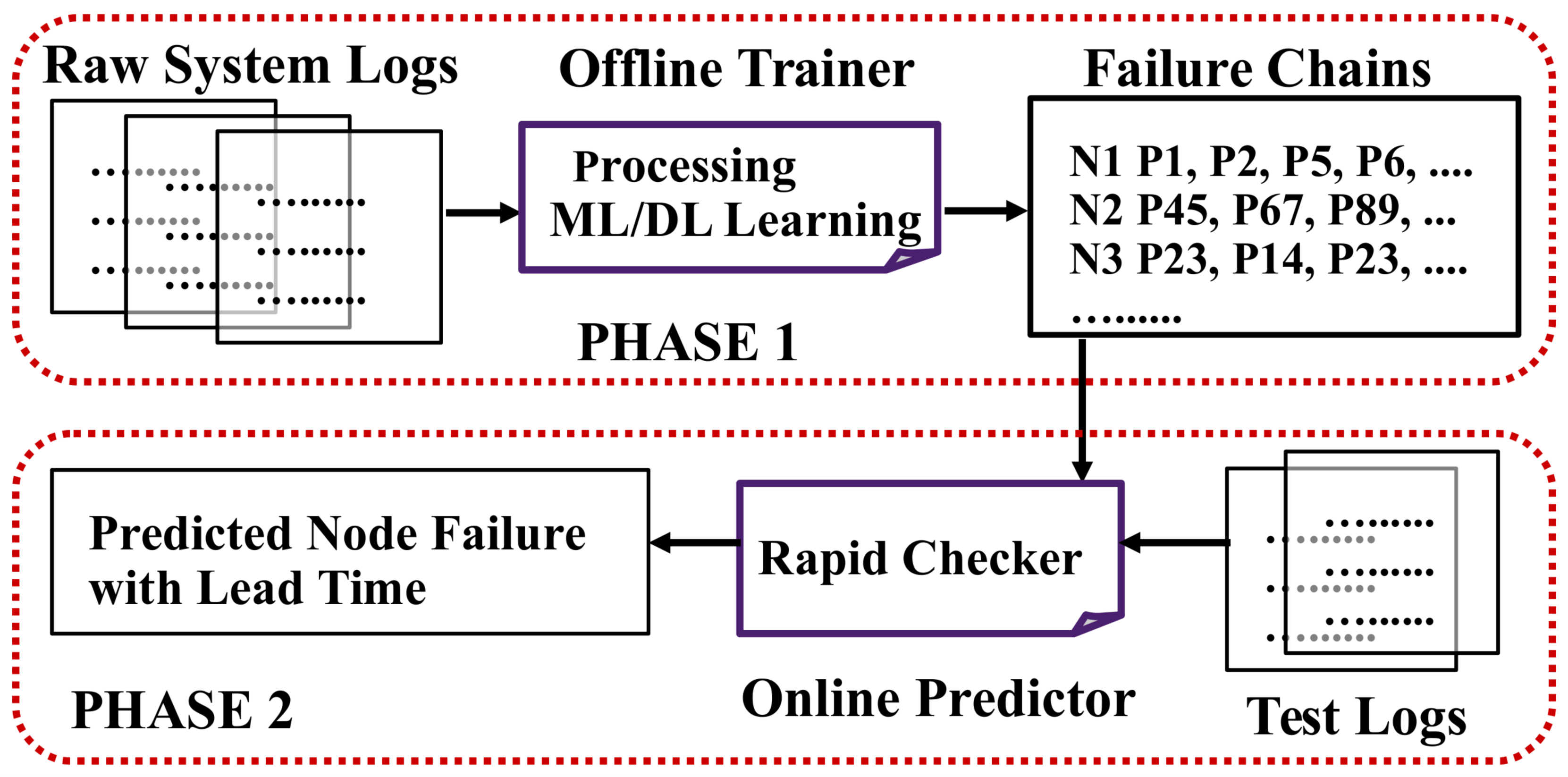
Research Problem 1

- Online Failure Prediction from Heterogeneous Logs
  - Large – Scale Systems, Fast log parsing (Tokenization)
  - Quick inference during testing
  - Can we contribute an efficient automated framework for **proactive** fault tolerance in HPC? (before the failed component stops responding)

- Impediments:
  - Require **low inference time**
  - Effective lead time → **sufficient for proactive** actions ?
  - Low **inaccuracies** (False Positive and False Negative Rates), else contributions not worthwhile
  - Generality, Cross – System Portability ?

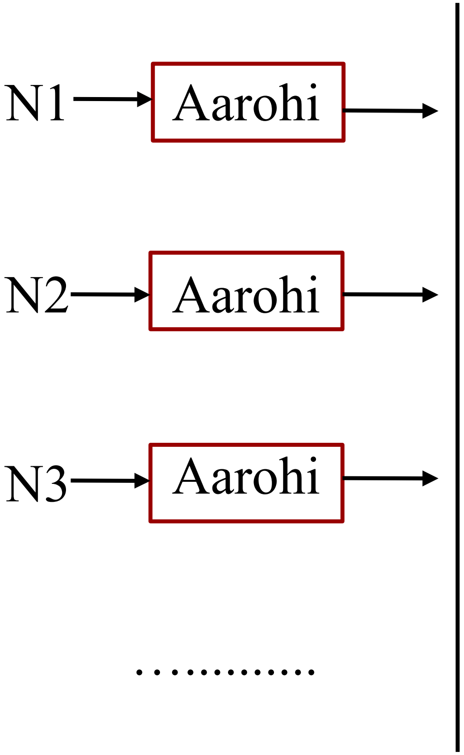
Aarohi

- Phase 1: TBP, Desh, Phase 2: Simple (no novelty)
- Phase 2: Aarohi, output of Phase 1 prerequisite (no novelty)



Aarohi

- Real–time inference, process 1 log message at a time (phrase)
- RE/CFG based compilation for failure prediction
- **Node–specific** Failure Prediction

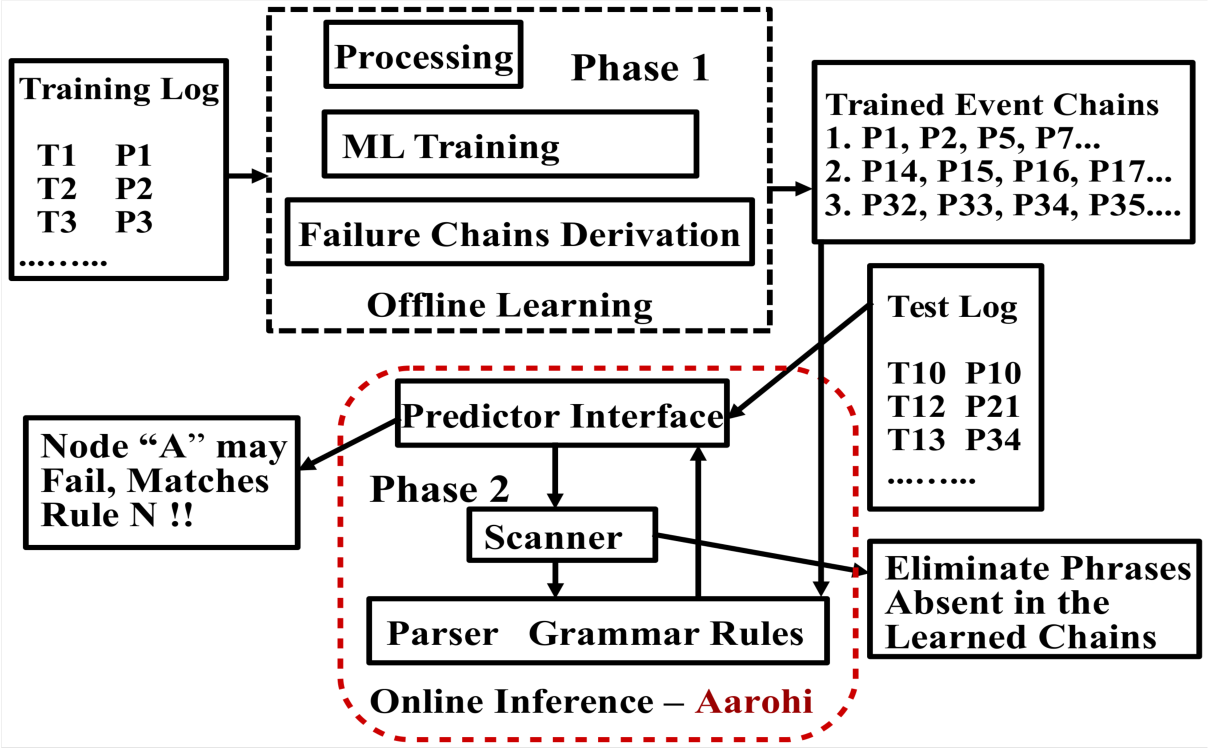


N1 → Aarohi →

N2 → Aarohi →

N3 → Aarohi →

....



Training Log

T1 P1
T2 P2
T3 P3
....

Processing

Phase 1

ML Training

Failure Chains Derivation

Offline Learning

Trained Event Chains
1. P1, P2, P5, P7...
2. P14, P15, P16, P17...
3. P32, P33, P34, P35....

Test Log

T10 P10
T12 P21
T13 P34
....

Predictor Interface

Phase 2

Scanner

Parser Grammar Rules

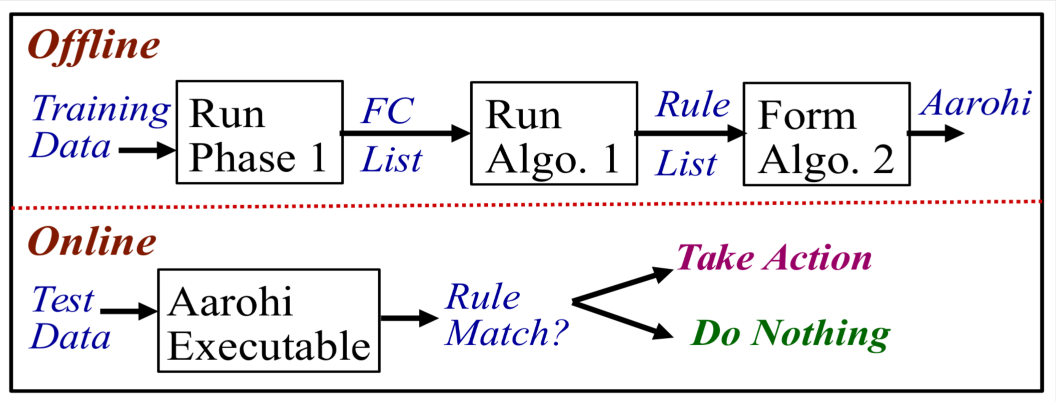
Online Inference – Aarohi

Node "A" may Fail, Matches Rule N !!

Eliminate Phrases Absent in the Learned Chains

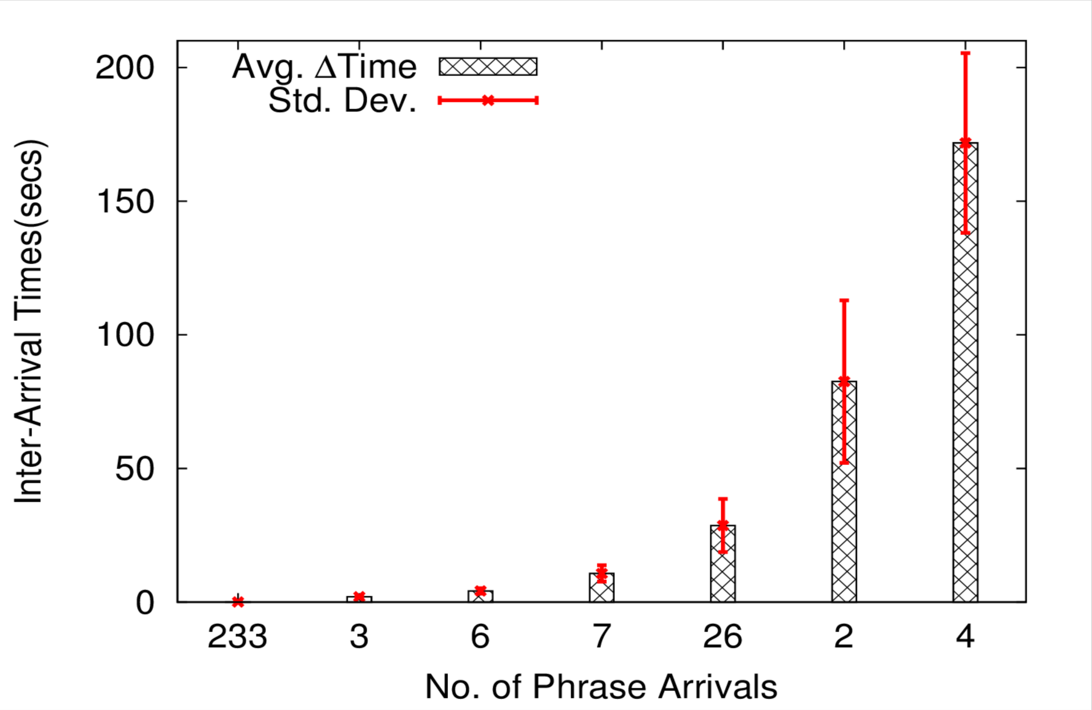
Aarohi

- Failure Chain (FC) to Grammar Rules (Algorithm 1, *Offline*)
  - Tokenization (Raw Log → Template → Token)
  - FC–based Rule Formulation, Single chain rules → LALR(1) Grammar
- Parser Formation (Algorithm 2, *Offline*)
  - Scanner → Skip Token, Return Token + Arrival Time
  - Parser → Parse log, Rule Check, Error handling semantics
  - Track checked rule + current token, abort if ΔT > threshold
- Test data with Aarohi Executable (*Online*)



Time Differences

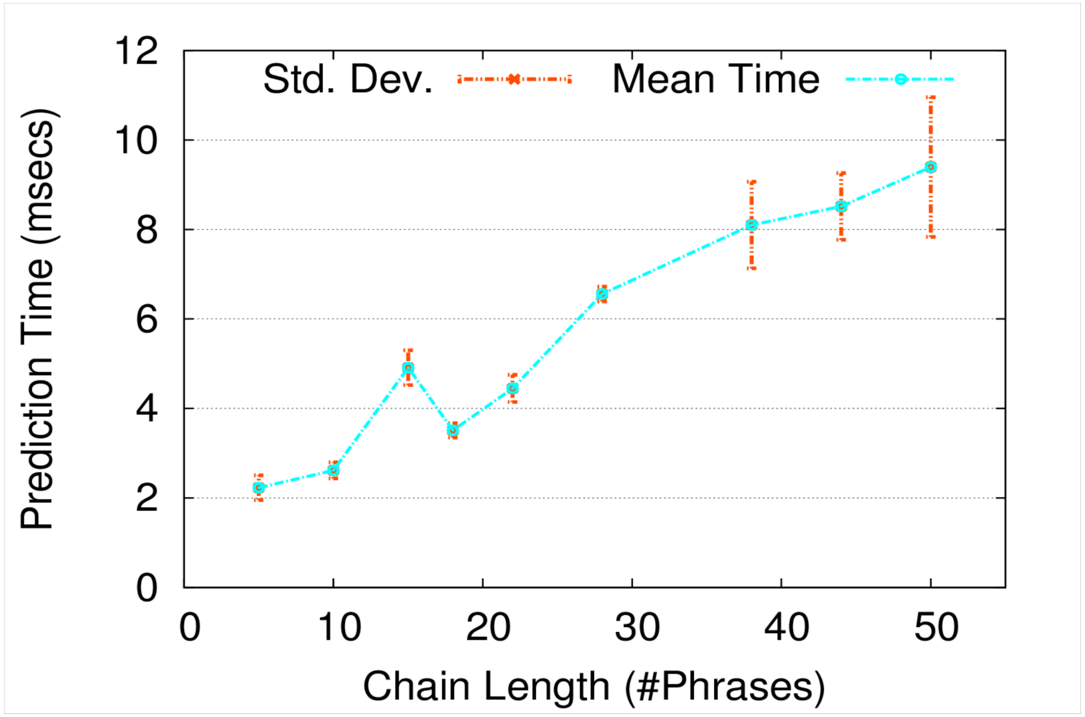
How distant are consecutive phrases from one another ?



- ✓ 93% of the phrase inter-arrival times ≤ 4 mins (helps define timeout)
- ✓ 6.7% outliers, ΔT ≥ 20 mins (*high variance*, not shown)
- ✓ More than 77% of the phrases have ΔT ≤ 1 sec (micro/milli secs)

Results

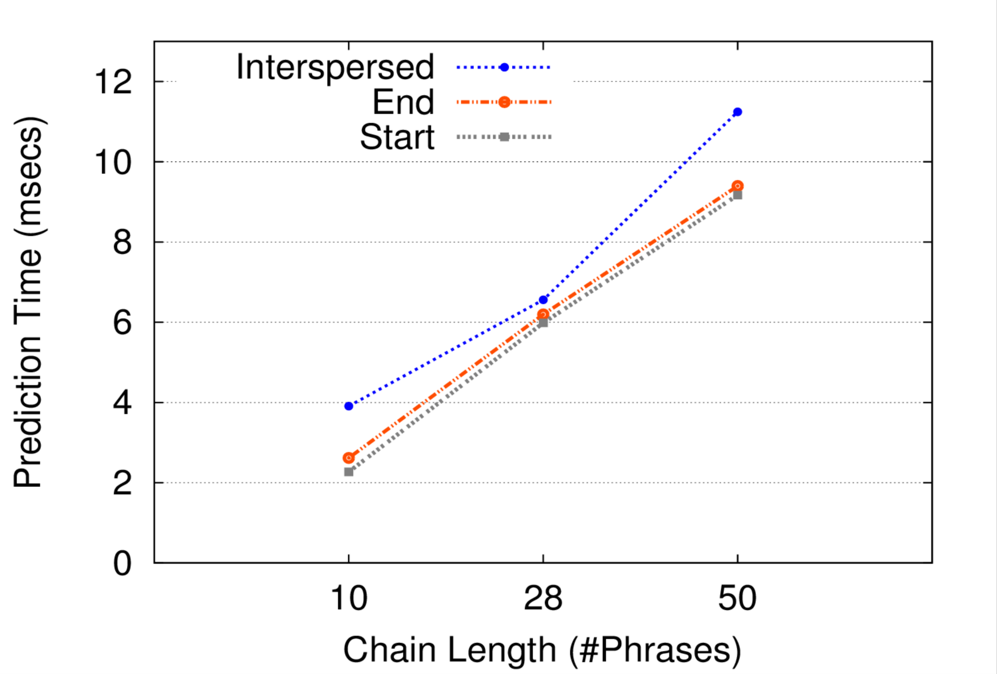
How high are the inference times with different chain lengths?



- ✓ Inference Time < 10 msecs for chain length ≤ 50
- ✓ Contains benign + FC-related phrases in the test log
- ✓ Std. Deviation ≤ ±1.56 msecs

Results

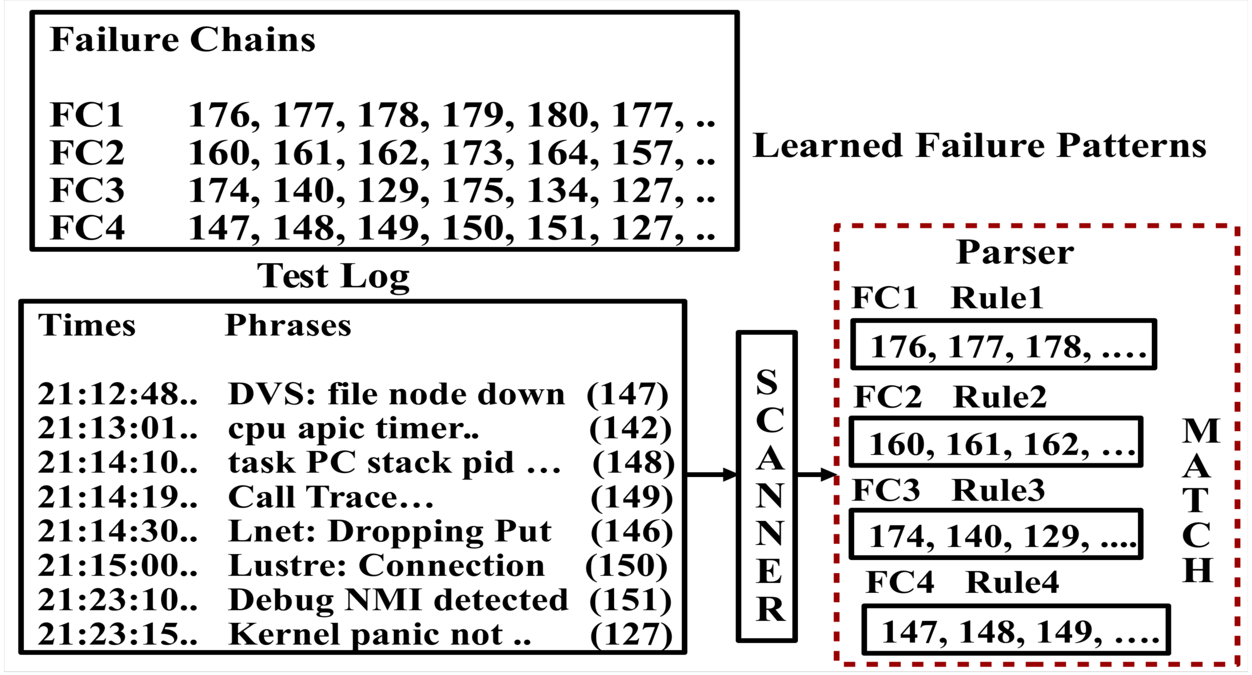
Does the prediction time fluctuate based on the location of benign phrase concentration (start/end or interspersed) in between FCs ?



- ✓ Start/End concentrated non-FC phrases → similar prediction times
- ✓ Alternate interleaved phrases interspersed in between → higher prediction times

Factors currently being addressed

- Inference time, does not include the tokenization time (inefficiently done)
- Single instance Parser, No Simultaneous Multiple Rule Checks
  - Phrase Inter-twining exists, but presence of an entire FC between two phrases is rare (absent) for nodes (but theoretically possible)
  - Log Timestamp versus System Time, handling in practice ?



Failure Chains

FC1	176, 177, 178, 179, 180, 177, ..
FC2	160, 161, 162, 173, 164, 157, ..
FC3	174, 140, 129, 175, 134, 127, ..
FC4	147, 148, 149, 150, 151, 127, ..

Learned Failure Patterns

Test Log

```
Times          Phrases

21:12:48..   DVS: file node down   (147)
21:13:01..   cpu apic timer..      (142)
21:14:10..   task PC stack pid …   (148)
21:14:19..   Call Trace…           (149)
21:14:30..   Lnet: Dropping Put    (146)
21:15:00..   Lustre: Connection    (150)
21:23:10..   Debug NMI detected    (151)
21:23:15..   Kernel panic not ..   (127)
```

Parser

FC1 Rule1

176, 177, 178, ….

FC2 Rule2

160, 161, 162, …

FC3 Rule3

174, 140, 129, ….

FC4 Rule4

147, 148, 149, ….

Factors currently being addressed

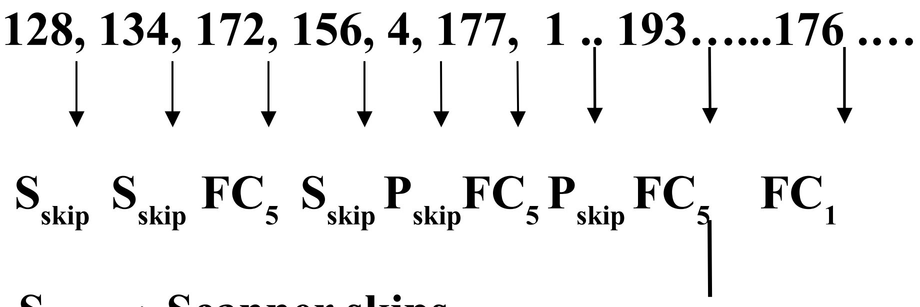
- FC1: {176 177 178 179 180 137}, FC2: {172 177 178 193 137} Single Chain Rule

S→(176 C 137) | (172 C 137), C→(B 179 180) | (B 193), B→(177 178) LALR(1) Rule

LALR (1) evaluation results

- Raw log tokenization via parser rules
  - **Lustre: 29289:0:(obd_config.c:1127:class_config_llog_handler()) Skipped 1 previous similar message → Lustre_*_skipped_* → P200**
  - Add it to the inference time

Test data stream



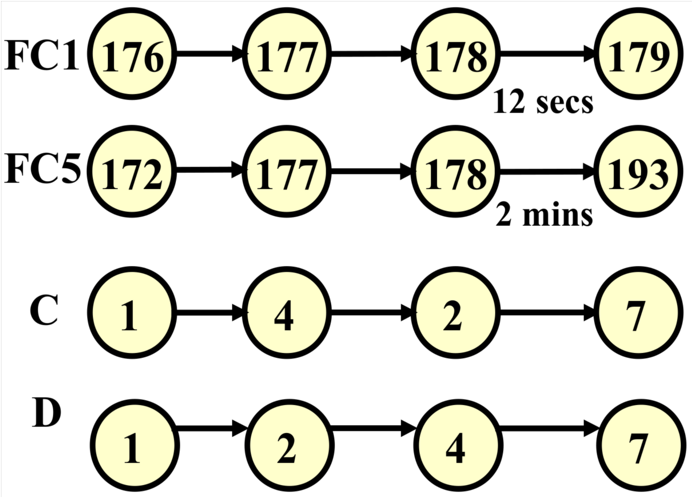
128, 134, 172, 156, 4, 177, 1 .. 193……176 ….

S_{skip} S_{skip} FC_5 S_{skip} P_{skip} FC_5 P_{skip} FC_5 FC_1

S_{skip} **→ Scanner skips**

P_{skip} **→ Parser skips**

FC5 Match



Research Problem 2

- *How* do nodes fail?
  - Understand external environmental influences on compute nodes
  - Underlying inter–node correlations (beyond spatial/temporal characteristics)
  - Investigated limited view of *isolated* node failures (high-level causes)

  Goal: Have better clarity of the global view through *holistic analysis* ?
- Current state–of–the–art:
  - Studies on node–specific events in isolation (external impact unaccounted)
  - Failures studied on different layers (application/hardware) or components (interconnect/GPU) in isolation (uncorrelated)
  - Spatial or temporal characterization in terms of *manifested* node failures
- How faults propagate causing nodes to fail?
  - ➔ Facilitate better failure handling (reactive/proactive) for *sustained resilience*

Research Problem 2

- **Impediments:**
  - Missing SEDC data, detailed application logs unavailable (only job scheduler related)
  - Transient faults (absent in logs, missing data due to logging discrepancy or intangible impact ?), hard to decipher
  - Distinguish fail-slow (functional but degraded mode) versus fail-stop?
  - Further inputs may be required from operators for validation !!
- **Solution Design** (finer to coarser)
  - Backtrack from node-specific failure logs to blade→chassis→cabinet
  - Correlate controller/environment/event logs around the same time-frame
  - Cascading impact? Lead time enhancements? FP Rate degrades?

Not interesting: ***High Level Categorization (layer or component), Internal vs. External causes, Node Failure characterization*** **(already done)**

Case Studies

1 week log – 6 node failures

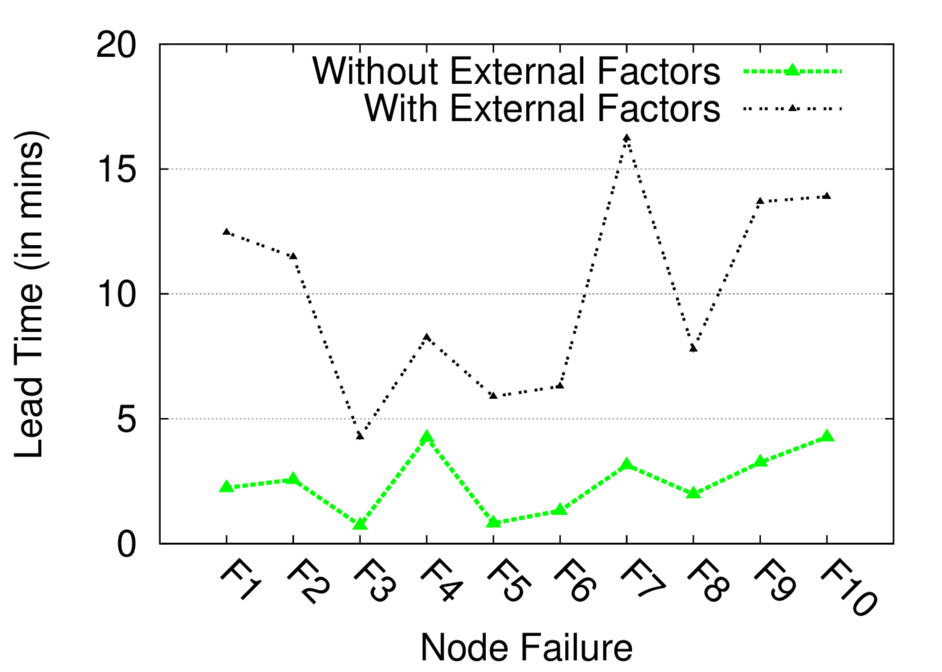
- 1st, 4th & 6th days – 1 failure/day, a) ***App-caused*** (out of memory/killed process → kernel-oops), b) ***App-triggered*** Kernel-oops (unable to handle kernel paging request), c) H/W errors, critical MCEs
- 2nd Day – 3 failures, Neither temporally nor spatially close (3 separate groups & cabinets, at 4 am, 12.38 pm & 3.21 pm) but same pattern (H/W error, processor corruptions → MCEs → Kernel-oops)

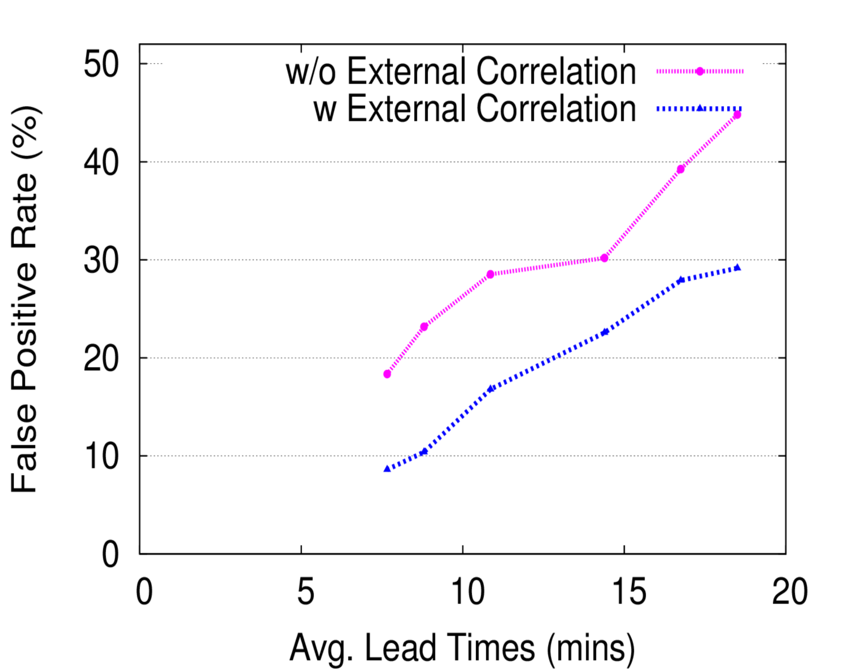
External Factors:

- 1st Day: No early indications around that time frame (purely app-caused)
- Day 2, 4 (Blade: Aries link error, get_die_temp_threshold/cannot get CPU Tjmax but not close to the failure time)
- 6th Day: This node had several early indicators of ***ec_hw_errors***, link errors for > 1 hour (fail-slow, degraded but functional component?)

Results

By how much can the lead times improve considering the external impact ?





- ~5 times increase in lead times with external factors accounted (2 to 12 mins)
- FP rate do not degrade with subsystem correlations (18.35% to 8.58%)
- Fan speed, Temperature threshold violations common but ***not main culprit*** of several node failures (not shown)

Root Cause Diagnosis

- **Internal causes (console/message/consumer)**
  - Do not have early symptoms in controller/SEDC logs
  - Lead time enhancements not possible (subject to further studies)
  - App-related (App → Resource constraints → Kernel oops → Failure)
- **External causes (controller/SEDC/event)**
  - Lead time enhancements feasible based on early symptoms

How much do the past findings hold?

1. 39% fail-slow hardware faults caused by external factors (FAST'18)
2. S/W causes 20% failures but contribute to 53% system downtime,
   H/W causes 42% failures but contribute to 23% repair time
   (261 days logs, 3.7 TB data of Blue Waters Petascale) (DSN'14)
3. App-caused congestion, Lane degrades/link failures, Bursty n/w throttling (DSN'18)
4. SWOs→Lustre FS, Failover methods (Interconnect/FS) (DSN'14, TPDS'17)

Plans Ahead

- Continue work on RCA
  - Measurement-driven, automating seems impractical
  - Lead time characterization necessary (not much extra log based timely correlation feasible)
  - How to quantify power implications?
- On the horizon
  - Real-time Streaming Logs (unlike archived logs)
  - Deployment in a Production Cluster
  - Demonstrate Feasibility Through Practice
    - Trigger Proactive/Reactive Actions during Lead Time ?
    - Assess performance trade-off ?