Correlating Log Messages for System Diagnostics

Presented by
David Dillow

Raghul Gunasekaran, David Dillow, Galen Shipman, Don Maxwell, Jason Hill
Oak Ridge Leadership Computing Facility

Byung H.Park, Al Geist
Computer Science Research Group

Introduction

- **Challenges analyzing large-scale system logs**
  - Large volume of data (occur in bursts)
  - Quickly overwhelmed with data
  - More of a failure log than an event log
  - Logging systems can drop messages under load
  - Redundant information
  - Need a comprehensive understanding of the compute infrastructure
  - Knowledge of the current system state

Introduction

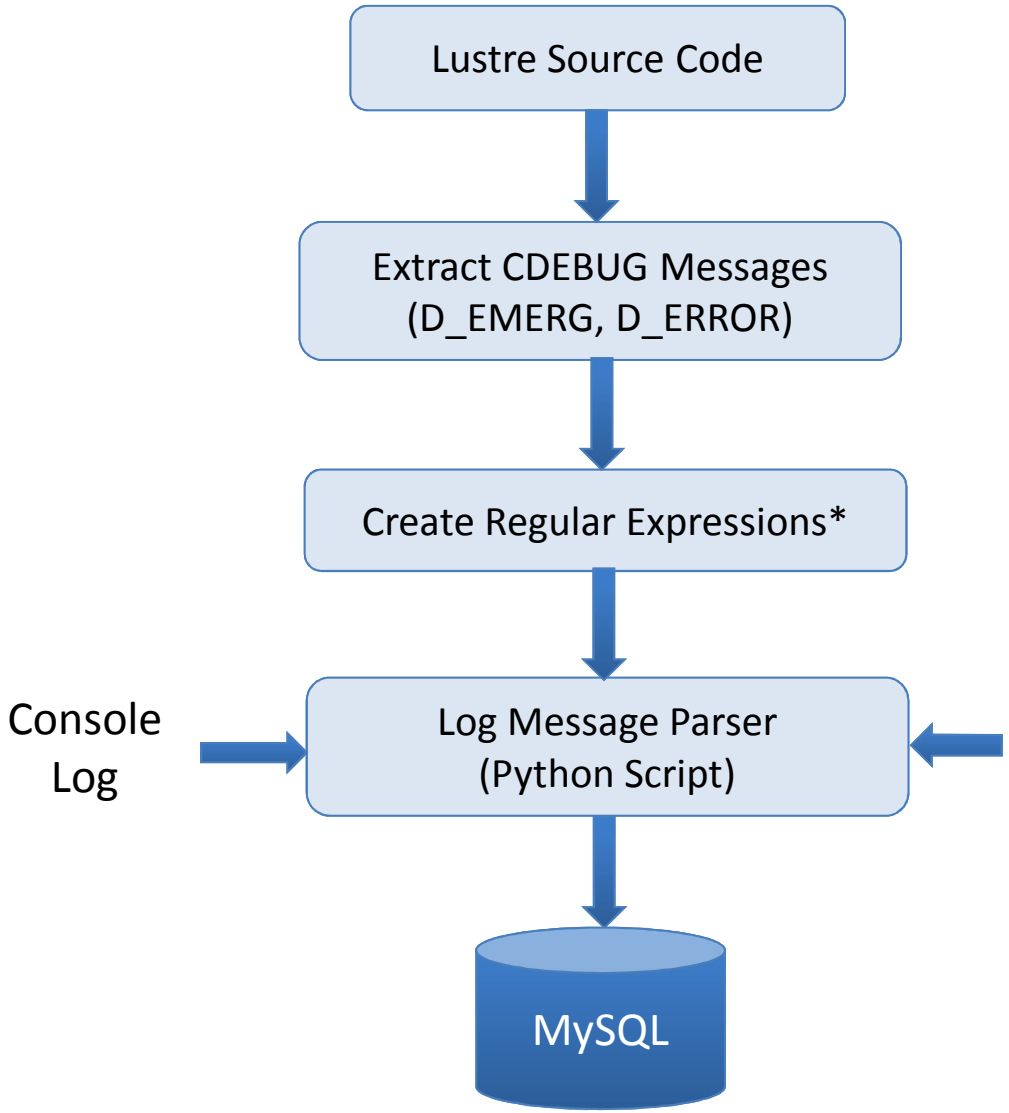
Console Log Messages

- Highly unstructured log messages
  - printk statements within the kernel
- Message structure changes for new releases

Our Approach

- Pre-process Log Messages
  - Parsed in a machine-readable format
  - Add attributes that simplify interpretation
- Cluster Log Messages
  - Reduce redundant information
- Time-series analysis of clustered log messages
  - Failure trends for application and hardware

Log Pre-processing



XT5 Node Info

Node	NID	IP Address	Type
C1-7c2s7n1	4861	xx.xx.xx.33	cnode
C4-7c0s1n3	2215	xx.xx.xx.106	rtr
C24-0c0s4n0	17939	xx.xx.xx.157	login
C24-0c0s4n3	17936	xx.xx.xx.160	batch

OLCF System Information

IB Router Info

IP Address List

File System Server info

IP Address	Type
xx.xx.xx.100	oss
xx.xx.xx.200	mgs
xx.xx.xx.211	mds

* Done manually for individual messages

Log Pre-processing

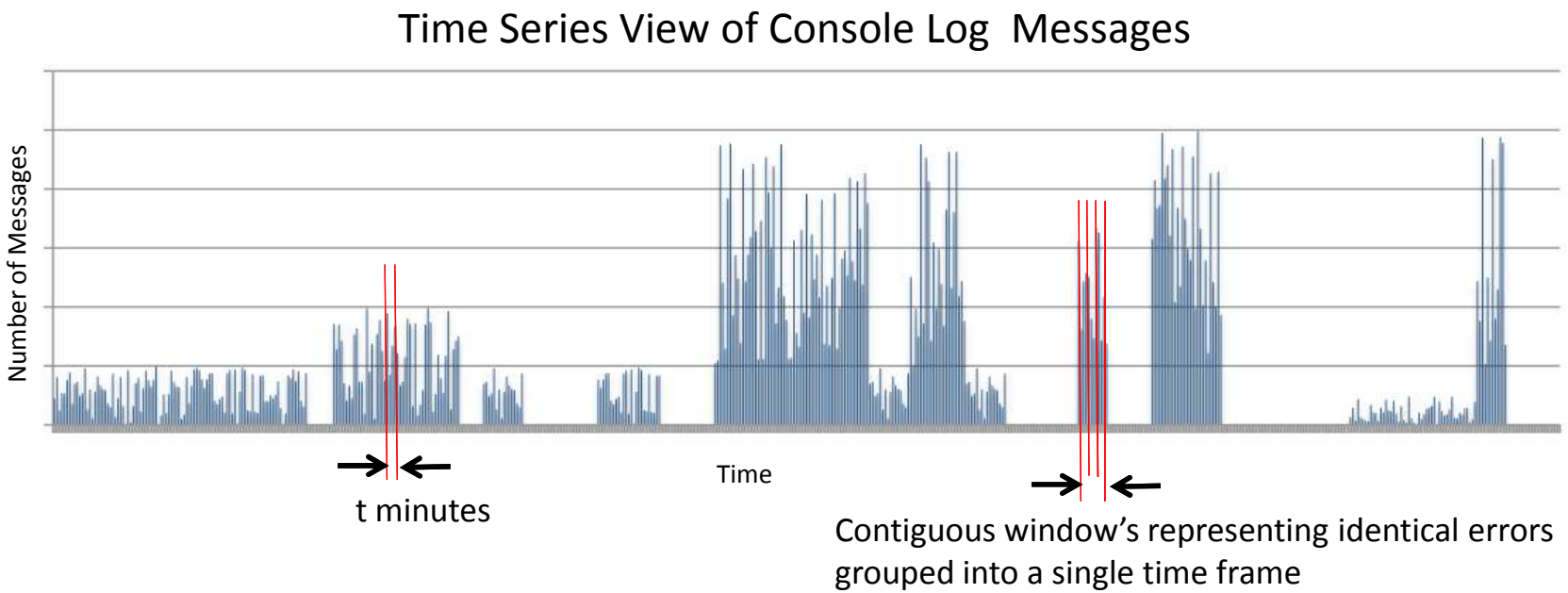
[2010-01-13 07:22:05][c16-3c1s4n0]LustreError: 16149:0:(ptllnd_peer.c:903:kptllnd_peer check_bucket()) Could not send to 12345-15235@ptl1 after 250s (sent 293s ago); check Portals for possible issues

Timestamp 2011-01-13 07:22:05
sourceID c16-3c1s4n0 sourceNID 1948 sourcetype cnode
processID 16149 module lnet
modulefile ptllnd_peer.c modulefunction kptllnd_peer_check_bucket()
target1 18235 target1type rtr
errormsg check Portals for possible issues

[2010-01-24 12:04:02][c10-4c1s0n1]LustreError: 25704:0:(events.c:55:request_out_callback()) @@@ type 4, status -5 req@ffff8101f428b800 x1872401/t0 o4->widow1-OST0008 UUID@xx.xx.xx.105@o2ib: 6/4 lens 384/480 e 0 to 1 dl 1264353000 ref 3 fl Rpc:/0/0 rc 0/0

Timestamp 2010-01-24 12:04:02
sourceID c10-4c1s0n1 sourceNID 8609 sourcetype cnode
processID 25704 module ptl
modulefile events.c modulefunction request_out_callback()
target1 widow1-OST0008 target1type ost
target2 xx.xx.xx.105 target2type oss
errormsg operation 4 failed errorcode 5

Clustering Event Logs



Outline of our clustering approach

- Analyze log messages within a time window
- Group contiguous windows representing identical errors
- Check for patterns across the time windows

Clustering Event Logs

Identifying a Time window

- For a set time frame, initially one minute
  - A set of one-to-one mapping between *sourceType* and *target1Type*
  - Should be one-third of the original log messages
  - If not, increase time window by one minute, max 10 minutes
  - Regenerate mapping and check for one-third criteria
- The one-third requirement is based on our observation that most often events in the log are represented by three consecutive log messages
- The ten minute limit, is based on a Lustre timeout setting which is currently 600 seconds for OLCF systems.
- The ten minute window also helps identify periodic messages from lost RPCs

Clustering Event Logs

Clustering within a time frame

- Summary of the log
  - Clustered as *sourceType, target1Type, module* and *errormsg*
- Isolating single points of failure
  - For a specific *sourceType* – list of distinct source NID's and error messages
  - Similarly for target1Type.
  - The NID list helps identify problem
    - If a single NID entry – single point of failure (Hardware anomalies)
    - List of NID's – identify the commonalities in the next stage
- Target Mapping
  - Clustering *target1* to *target2* (helps identify OSS/RAID controller failures)
- Associate NID list to Application information
  - Identify what App's were running on the nodes.

    Example: In most observed cases, nodes complaining run the same application.

Clustering Event Logs

Clustering across time frame

- Grouping Time windows
  - Time windows should be contiguous
  - Identical Error Messages
- Correlating Messages across time windows
  - Check for identical error messages across time widows
    - Then, compare NID list across time windows - look for any common NIDs
  - w.r.t. Application
    - Identify Application running on the list of NIDs
    - Compare with past 10 runs of the application

Results

Correlating logs

- **30 minutes from system boot a burst of over 3000 messages clustered as 7 messages**

SourceType	Target1type	Module	ErrorMsg
cnode	ost	ptl	Connection lost
cnode	mdt	ptl	Connection lost
cnode	ost	ptl	Connection restored
cnode	oss	ptl	Request Timed Out
cnode	mdt	ptl	Connection restored
cnode	mgs	ptl	Connection restored
cnode	mgs	ptl	Connection lost

A set of Compute nodes lost connection with OSS, MDS and MGS, and re-established connections. No commonalities across the compute nodes and no application was running.

Results

Correlating logs

- 10 Minutes prior to the burst of messages, we observed this Lustre Info message in the log.
- A particular router had a connection race problem with OSS

SourceNID	SourceType	Target1type	Module	Errormsg
1947	rtr	oss	ptl	Conn race

- We saw a total of 19 messages in the log.
- Through our clustering approach we were able to present information in a condensed format.
- Time Series Analysis & Domain Knowledge helped attribute transient errors to recent connection loss messages.

Results

Hardware Anomalies

- During one of the system event 23,000 log messages were generated within three minutes. These clustered into two groups.

SourceType	Target1Type	ErrorMsg
cnode	oss	Request Timed Out
cnode	ost	400 failed
cnode	ost	Connection lost
cnode	ost	Connection restored to service

SourceType	Target1	Target1Type	ErrorMsg
batch	c14-0c0s6n0	rtr	PTL_NAL_FAILED(4)
svc	c14-0c0s6n0	rtr	PTL_NAL_FAILED(4)
login	c14-0c0s6n0	rtr	PTL_NAL_FAILED(4)
cnode	c14-0c0s6n0	rtr	PTL_NAL_FAILED(4)

Results

Hardware Anomalies

- This was followed by 2 million log messages over the next 3 hours

SourceType	Target1	Target1Type	ErrorMsg
cnode	c14-0c0s6n0	rtr	check portals
batch	c14-0c0s6n0	rtr	Timing out
login	c14-0c0s6n0	rtr	Could not get credits for

- The first set of log messages identify a specific router was at fault.
- ~4 min from the first set of messages the router had a kernel panic
- Isolated a this failure from a deluge of log messages
- Can similarly isolate OSS or RAID controller failures, where the log would complain on the OSTs
- Could be used to identify new failures where the actual event may not be currently monitored.

Results

Application Patterns

- Correlate error messages to application runs
- To identify potential anomalous application behavior
- Remove messages which are caused by well-known problems
  - HW failures
  - Out of Memory errors
  - Segmentation faults
  - Wall time exhaustion

Results

Application Patterns - Example 1

- User reserves large number of compute nodes
- Multiple aprun's in parallel on a smaller set of compute nodes – time limit of 1800 seconds.
- More than 50% of jobs terminated within a few seconds.
- Observed for more than 2 months, user did not complain
- Lustre error messages: PTL_NAL_FAILED(4) followed by MDS_CLOSE failed.
- Followed by a SIGKILL
- Appears to be caused by the application

Results

Application Patterns – Example 2

- In a three month observation period for only three weeks an application caused File System errors.
- Lustre error message: OST_STATFS failure
- Application did not quit, but caused the same file system error.
- User acknowledged they were trying new libraries during that time period.
- Appears to be related to this application's activity.

Results

- A limitation is that we were analyzing a few months of old data
- Changing system software complicates root cause analysis
- However, we show that correlation exists between Application behavior and Lustre error messages
- Analyzing in near real-time would increase our works impact
  - Determine source of faults and address them quickly
  - Identify application usage patterns that expose faults
  - Improved feedback to the user community

Questions ?