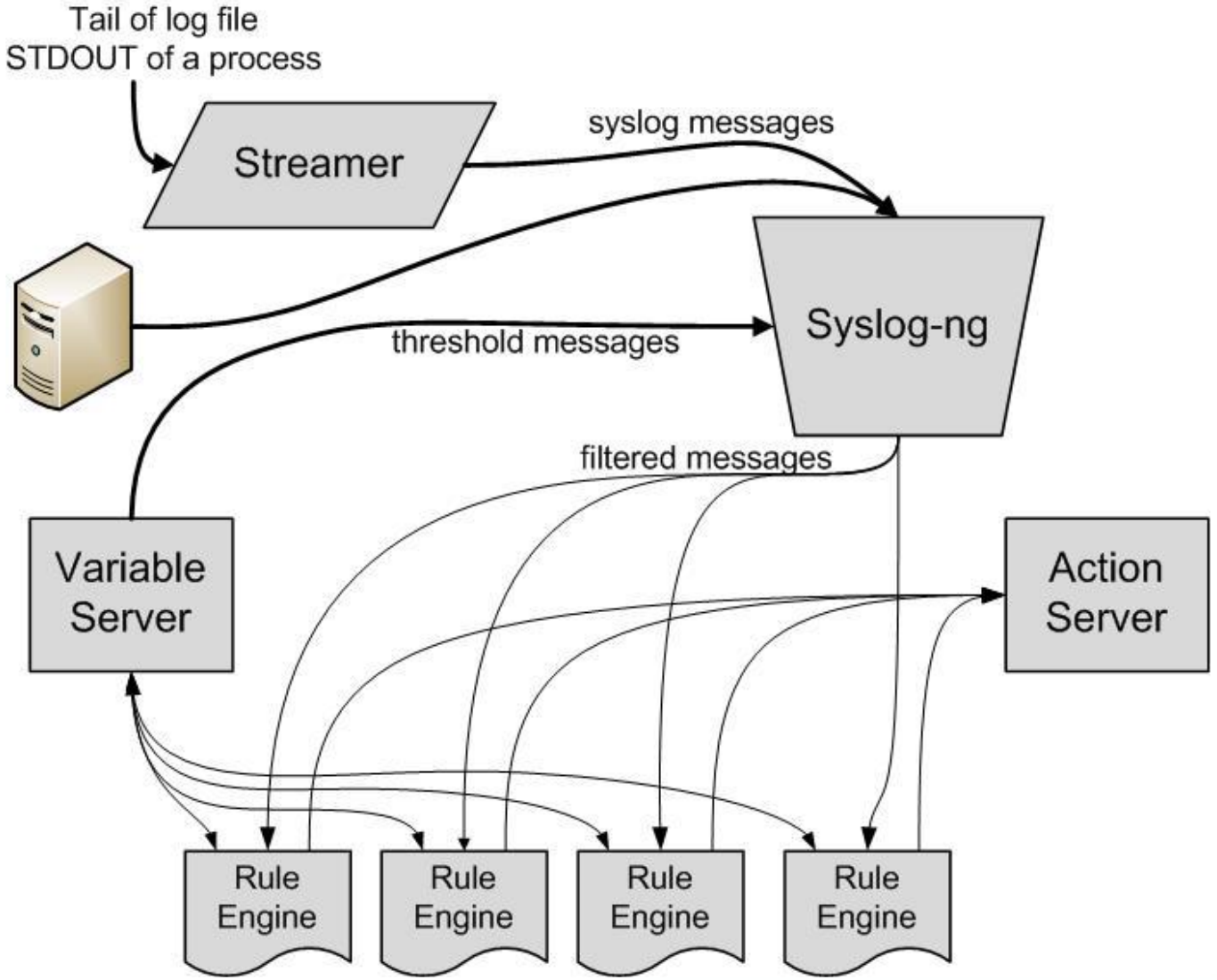
Log Analysis and Event Correlation Using Variable Temporal Event Correlator (VTEC)

Paul Krizak | November 11, 2010

Background

- Swatch was being used until 2006, didn't scale past a few thousand systems, lacked sophisticated event correlation features.
- Project goals:
  - Scale to 10's of GB/day of log data
  - Take advantage of multiple processors
  - Correlate events in real-time (no batch processing)
  - Correlation rules must be easy to read, modify, create
- SEC, Splunk evaluated
  - SEC: rules too difficult to read/learn/modify
  - Splunk: indexer did not scale (v1.0)

Architecture



Rule Engines

- Accept filtered log data from syslog-ng on STDIN
- Interacts with variable server and action server using Perl object interfaces

```
# Instantiate objects to communicate with the servers
my $vs = new VariableInterface();
my $as = new ActionInterface();

# Setup thresholds
$vs->set_incrementer_threshold("name", ">=", 10);

# Process log data
while(<STDIN>) {
    my %msg = parse($_);
    # Event correlation stuff
}
```

Temporal Variables

- Scalar
  - Stores an arbitrary data value that has a finite lifetime
  - Returns “0” when timed out
  - Good for alarms, temporary data storage

```
# Send an e-mail, but don’t send another one for an hour
if($vs->get_scalar(“sent_mail”) == 0) {
    send_mail();
                    #    name       timeout
    $vs->set_scalar(“sent_mail”, “+3600”);
}

# Set an alarm (scalar + threshold); wake us up in 60 seconds
if($msg{message} =~ /my important trigger/) {
    $alarm = $vs->set_alarm(60, “some_event”);
}
if($msg{message} =~ /TIMEOUT: SCALAR some_event/) {
    take_action();
}
```

Temporal Variables

- Incrementer
  - Counts the number of events over a window of time
  - Used to detect rates of events over time
  - Timeout period lets you adjust how far into the past you want to count data
  - Increment amount is generally set to 1

```
# Detect rate of EXT3 errors on www over 10 second period.
if($msg{from_host} eq "www" and $msg{message} =~ /EXT3 Error/i) {
    # No instantiation, just create as needed at any time.
    #                        name      increment  timeout
    $vs->set_incrementer("www_ext3",    1,       "+10");
    if($vs->get_incrementer("www_ext3") > 10) {
        # More than 10 EXT3 errors/10 sec => 60/min or 1/sec
        notify_ops_staff();
    }
}
```

Temporal Variables

- List
  - Collects incrementers into one structure using keys
  - Aggregates data across similar incrementers

```
# Detect NFS problems
If($msg{message} =~ /NFS server (\w+) not responding/i) {
    my $nfs_server = $1;
    # The increment amount in a list is hard-coded to 1
    #             name   key_to_increment  timeout
    $vs->set_list("nfs", $nfs_server,       "+60");
    # fetch the number of unique keys in the "nfs" list
    if($vs->get_list_keys("nfs") > 50) {
        # More than 50 hosts (keys) have reported that $nfs_server
        # is down (key has not timed out) in the last 60 seconds.
        notify_ops_staff();
    }
}
# You can also query the entire list - this would return the total
# number of "not responding" messages in the last 60 sec:
$vs->get_list_all("nfs")
```

Variable and Action Servers

- Variable server
  - Hosts variables in a common namespace
    - Rule engines can share data
    - Rule engines do not have to maintain state
  - Injects messages into log stream when threshold conditions are met (as defined by the rule engines)
- Action server
  - Queues jobs that can alert or correct problems detected by rule engines
  - Jobs can run at a specific time, or “now+time”

Some Rule Engines at AMD

- Failed hardware
  - Counts ECC/EXT3 errors and alerts when threshold exceeded
- NFS file server checks
  - Monitors for "NFS server xxx not responding" and alerts when a large number of unique hosts are reporting problems
- Interactive load monitor collator
  - Aggregates periodic load data from interactive servers and sends out collated reports to users
- Reboot loop detection
  - Alerts support staff if hosts are stuck in a reboot loop

Designing for Performance

- Current performance
  - Handling ~1000 msg/sec (~10GB/day) with 21 rule engines and four local disk logs on a four-vCPU VM
- Multi-threading
  - Variable and action servers use Perl threads to distribute workload
  - Rule engines, syslog-ng are all in separate processes
- Bottlenecks
  - Syslog-ng can be a bottleneck if many match() rules are used (regular expression engine)
  - Incrementer calculation routine in is O(n); does not scale well with frequent events (>200 events/sec)

Challenges

- Variable server performance (due to incrementers)
  - New O(log n) routine is being tested
- Feedback from actions
  - Currently mostly “fire and forget”
    - You can (awkwardly) have actions use things like `logger(1)` to inject status data back into the log stream

Lessons Learned

- Designing for scalability from the ground up is crucial
  - Take advantage of multi-core with threads and multiple processes
  - RAM is cheap – use in-memory data structures instead of disk-based databases
  - Watch out for algorithms that are O(n) or worse
- Abstraction is a powerful tool when correlating events
  - Abstracting rate data into a simple incrementer/list query
  - Breaking up complex correlations onto multiple engines
- Never underestimate the power of familiarity
  - For a system that must be "programmed" – stick with a familiar language, so that your customers/colleagues will actually use it!

Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

AMD
The future is fusion

Backup Slides

Paul Krizak | November 11, 2010

Animation: Incrementer