

# Listening to your Cluster with LoGS

James E. Prewett

The Center for High Performance Computing at UNM (HPC@UNM)

## 1 Introduction

Large systems are now being built from smaller systems. GNU/Linux clusters have gone from a fad at a couple of academic institutions to being the some of the largest and fastest computers in the world. Similarly designed systems have become the bread and butter of more traditional super-computing vendors. Cluster computing is now big business. All of the computers listed in the TOP500 List for November 2003 are clusters[1]! Further, smaller clusters are now being purchased (and often administered) by individual research groups; clusters are accessible computing power.

One of the problems with this new machine design is that the machines are much more complex to maintain, monitor, and administer. Instead of having to care for one instance of an operating system, we now have to look after N instances with N times as many components that can fail!

One way to make the administration of these machines bearable is to carefully analyze system logs with a real-time analysis tool. Unfortunately, the available free-software tools are lacking in many ways when events must be correlated across a potentially very large number of machines. Another issue with log analysis for larger clusters is that the volume of log data can be quite large. Free tools such as Logsurfer[2] and SWATCH[3] can be very inefficient when finding interesting messages due to the organization of their ruleset.

LoGS is a log analysis engine that attempts to address many of the issues with maintaining cluster machines. LoGS has a dynamic ruleset, is able to look for one or more messages before triggering an action, and has a powerful programming language used for configuration and extension. With proper rule-set construction, LoGS is a very efficient analysis engine.

LoGS is a programmable log analysis engine available under the terms of the GNU General Public License (GPL)<sup>1</sup>. LoGS aims to be a very flexible tool for identifying and responding to interesting messages or groups of messages.

LoGS is written entirely in Common Lisp<sup>2</sup> and utilizes the CLOS object system. Common Lisp was chosen for its flexibility (as Paul Graham points out, the entire language is always available and code is expressed as a series of expressions[4]), for having an excellent (and fast![5]) regular expression engine, and because nesting data structures with lisp is very simple and intuitive. Common Lisp is also the programming language for creating the LoGS ruleset and for extending LoGS.

---

<sup>1</sup> LoGS has not yet been released, but will be soon.

<sup>2</sup> Currently, LoGS is only known to work with CMUCL Common Lisp.

LoGS development is currently being driven by the need to administer GNU/Linux based clusters more efficiently and to create detailed reports demonstrating the amount of usage, etc., for my clusters. However, LoGS is intended to be useful for a variety of log analysis tasks. Development is currently focused on making LoGS useful for analyzing log files from cluster machines, computing grids, and large enterprise log servers. Some work is also being done to make LoGS more useful for analyzing Apache httpd log files (which should also be useful for Globus Toolkit 3 analysis as they are using the Apache log formats).

This work was based off of my earlier work using Logsurfer as my log analysis tool[6]. Many of the examples shown here are very similar to those shown in that work. However, I believe the examples are simpler, more powerful, and more easily readable than my attempts with other log analysis packages. Presented here is an introduction to using LoGS to do real-world cluster logfile analysis.

## 2 LoGS Components

LoGS consists of five different components[7]: messages, rules, rulesets, actions, and contexts. These components are similar to their counterparts in the Logsurfer program with the addition of the ruleset. Messages are a single piece of input from a data source (such as a single line from a syslog file). Rules match messages and trigger actions based on the messages that are matched. Contexts gather incoming messages and are in many respects similar to rules except that they can match multiple messages. Actions are used to respond to an event and are triggered when a rule matches a message or a context is closed.

### 2.1 LoGS Rules

Rules in the LoGS system match messages and trigger actions based upon those matches. A rule consists of seven optional components, a match function, a no-match function, a delete-rule function, a no-delete-rule function, an absolute timeout, a continue value (named “continuep”), and a list of actions. Note that while all seven of these fields are optional, it is generally useful to specify at least a match function. Since LoGS is written in Common Lisp, it takes advantage of the language for its configuration. The functions that make up a rule may be any Common Lisp function (provided it accepts the correct number and types of arguments).

The match function is used to determine which messages the rule matches, the no-match function determines exceptions to this. The delete-rule function is used to determine if a rule, once it is determined to match a message, should be deleted after executing its actions, the no-delete-rule function specifies exceptions to this. The timeout value specifies an absolute time in seconds after which a rule should be deleted. The continuep value specifies whether to test subsequent rules in the ruleset if this rule matches. Finally, the actions list is a list of Common Lisp functions.

**A simple Rule:** The simple rule shown in Figure 1 prints out every message that it sees. This rule demonstrates a couple of things about LoGS. First, rules are represented

as CLOS objects within LoGS, so they can be created using the standard `make-instance` macro to create them. Second, the match function specified is `(lambda (message) t)`. This match function is very simple, it takes a message as its argument and always returns true. Most log analysis packages force you to use regular expressions which can be both less clear and less efficient in circumstances such as this. It has been suggested that other techniques such as Bayesian statistical analysis[8] or even simpler techniques such as simple string length may be useful for determining which messages are interesting. Finally, this rule is very useful if placed so that it is the last rule in the ruleset. In this way the rule can be used to report on all of the messages that were missed by the rest of the ruleset.

```
; create a rule that prints every message
(make-instance
 'rule
 :match (lambda (message) t) ; match every message
 :actions
 (list
  ;; print out the message
  (lambda (message matches sub-matches)
    (format t "~A%" (message message))))))
```

**Fig. 1.** A LoGS rule that prints every line.

When used in conjunction with a relatively complete LoGS configuration, this rule can be used to tell when something anomalous has occurred. Take for example the messages left by the exploit of an input validation vulnerability in `rpc.statd` as published by the CERT Coordination Center® in CERT® Advisory CA-2000-17[9]. The messages recorded as a byproduct of the attack are many bytes of (primarily) non-printable characters, mostly consisting of the code for the exploit itself. These lines are difficult to match explicitly unless you've previously seen them or large pieces of them verbatim. By having a failsafe rule such as this as the last in the ruleset, these messages will be noticed.

**LoGS as a filter:** The majority of the messages recorded in system logs report normal operation of various components. By specifying what “normal operation” looks like to LoGS, we can filter out all of this data. This technique has been referred to as “artificial ignorance”[10]. The rule presented in Figure 2 filters out one example of an innocuous message.

**Using LoGS to alert you upon detection of a hardware failure:** LoGS may be used to identify and, if possible, correct problems found within a cluster or network. Automated identification of these problems allows them to be repaired as quickly as possible.

```

; create a rule that ignores messages indicating
; the correct number of processors being detected
(make-instance
 'rule
 :match (lambda (message)
          (cl-ppcre::scan "1.* kernel: Processors: 2"
                        (message message)))
 :actions (list #'ignore))

```

**Fig. 2.** An example of filtering out uninteresting messages.

In this role, the log analysis tool becomes a “virtual system administrator”. Clusters are becoming much larger. Red Storm<sup>3</sup> will have approximately 90,000 components[11]. Because clusters consist of many more components than traditional computing platforms, it is more likely that one or more of the components will fail and, therefore, the more valuable these virtual administrators become[12].

In Figure 3, LoGS is used to detect a machine reporting the wrong number of processors. When it detects this problem, it removes the machine from the schedulable nodes and reports the error. Note that the messages that report the correct number of processors have already been filtered out by the rule in Figure 2. It is often useful to create rules, such as this one, that catch specific differences from the norm.

Also note that this rule has multiple actions in the action list. Actions in the action list are run sequentially when a rule matches a message. Most log analysis tools force you to write multiple rules if you need to execute multiple actions.

## 2.2 LoGS Rulesets

LoGS has an object-oriented notion of rulesets as well. Rulesets may be modified by actions; rules may be inserted or deleted and the entire ruleset object may even be deleted! Rulesets are a sub-class of the rule class and share many of their properties.

Unlike other log analysis applications, rulesets in LoGS may be nested inside of other rulesets (arbitrarily deep). Because rulesets share many of the features of a rule, such as having a match function, rulesets can match messages. Unlike rules, when a ruleset matches a message, the rules inside of the ruleset are checked against the message. This kind of functionality allows rulesets to be organized in a tree like fashion, greatly reducing the number of comparisons that must be made to determine if a given message is interesting. My working ruleset, consisting of 93 rules, is divided up into 17 separate rulesets. With this ruleset, the most matches that are ever attempted are 23. Tools such as Logsurfer and SWATCH would require 93 matches to be attempted!

**Using a dynamic ruleset to limit alerts:** Because rulesets are dynamic in LoGS, they can be used to modify LoGS behavior in response to various messages. For example,

<sup>3</sup> Red Storm is a new massively parallel processing supercomputer being installed in 2004 at Sandia National Laboratories.

```
(make-instance
 'rule
 :match
 (lambda (message)
  (cl-ppcre::scan-to-strings
   "(1.*) kernel: Processors: ([0-9]+)"
   (message message)))
 :actions
 (list
  (lambda (message matches sub-matches)
   (remove-node-from-queue (aref sub-matches 0)))
  (lambda (message matches sub-matches)
   (exec
    "/usr/adm/bin/alert"
    (format
     () "Wrong number of processors: ~A detected on node: ~A"
     (aref sub-matches 1)
     (aref sub-matches 0))))))
```

**Fig. 3.** A rule for removing nodes that boot without the correct number of processors from the queue and alerting an administrator.

it is often desirable to limit the number of alerts that will be triggered by a particular rule. If a rule is to send an e-mail to the administrator prompting her to fix an issue that represents a minor annoyance as opposed to a serious problem, it is generally best to limit the number of e-mail messages that are sent.

Figure 4 shows a rule that alerts an administrator to the existence of a problem. It then modifies the ruleset so that a new rule, that does nothing when it matches a message, will intercept notifications of the problem.

**Using state to identify problems:** One problem with other log analysis tools is that there is little or no support for maintaining any sort of state. This is not a problem with LoGS. Rules may maintain state as a lexical closure around one or more rules. Rules may also modify the local or global environment.

Because we can maintain state, we can get more accurate results and be able to specify our intention more clearly. Figure 5 shows how LoGS rules can maintain state to find the CRC32 compensator exploit[13].

### 2.3 LoGS Actions

Actions define what is to be done with a particular message or context of messages (See section 2.4 for details on contexts). An action can be triggered either by a rule matching a message, a ruleset matching a message, or by a context exceeding one of its limits. An action can be any Common Lisp function that accepts three arguments, namely:

1. The whole message

```

(make-instance
  'rule
  :match
  (lambda (message)
    (cl-ppcre::scan-to-strings
      (as-concatenated-string
        "(1.*) sshd \\[[0-9]+\\] lastlog_get_entry : Error "
        "reading from "
        "/var/log/lastlog : No such file or directory")
      (message message)))

  :actions
  (list

    (lambda (message matches sub-matches)
      (exec
        "/usr/adm/bin/alert"
        (format () "lastlog does not exist on ~A"
          (aref sub-matches 0))))

    (lambda (message matches sub-matches)
      (rule-before
        (make-instance
          'rule

          :match
          (lambda (message)
            (cl-ppcre::scan-to-strings
              (as-concatenated-string
                "(1.*) sshd \\[[0-9]+\\] lastlog_get_entry : Error "
                "reading from "
                "/var/log/lastlog : No such file or directory"))

            :timeout (+ (get-universal-time) 43200)

            :actions ())))))

```

**Fig. 4.** A rule to alert an administrator of a problem, then modify the ruleset so the problem will not be reported for 12 hours.

```

(let ((compensation ())
      (corrupted-check-bytes ())
      (timeout-before-auth ()))
  (make-instance
    'rule
    :match (lambda (message)
              (cl-ppcre::scan-to-strings
                (as-concatenated-string
                 "(crc32 compensation attack: network attack|"
                 "Corrupted check bytes on input|"
                 "Timeout before authentication)"))
              (message message)))
    :actions
    (list
      (lambda (message matches sub-matches)
        (progn
          (cond ((equal (aref sub-matches 0)
                        "crc32 compensation attack: network attack")
                 (setf compensation t))
                ((equal (aref sub-matches 0)
                        "Corrupted check bytes on input")
                 (setf corrupted-check-bytes t))
                ((equal (aref sub-matches 0)
                        "Timeout before authentication")
                 (setf timeout-before-auth t)))
          (if (and
                compensation
                corrupted-check-bytes
                timeout-before-auth)
              (exec "/var/adm/bin/ssh_attack_report.sh"))))))))

```

**Fig. 5.** A rule to identify when OpenSSH's CRC32 compensator has been exploited

2. The portion of the message that caused the match (assuming that a regular expression was used).
3. Any sub-matches found (again, assuming that a regular expression was used)

Because actions are Common Lisp functions, they can be customized to the end user and are very powerful.

## 2.4 LoGS Contexts

Contextual information is key when analyzing a log file. Log analysis programs capable of making use of contextual information are able to detect and respond to more signatures of unusual behavior[14] than those that cannot. LoGS uses Contexts to gather and store related messages for potential use at a later time. This way the related messages may be analyzed as a group rather than one at a time.

Figure 6 shows LoGS rules that gather all of the log messages recorded from any of the machines involved in a PBSPro job. When the job is finished a report is generated from the messages that were gathered.

## 3 Conclusion

LoGS is a powerful, dynamic, extendible log analysis tool that is useful for monitoring clusters, grids, and other large machines or groups of machines. LoGS is currently being used by The Center for High Performance Computing at The University of New Mexico to monitor its largest computational cluster, LosLobos (a 256 node, GNU/Linux cluster) as well as other log files such as web logs and e-mail logs.

The way in which LoGS is designed allows it to be very efficient, a feature that is required in a cluster environment. LoGS also allows one to express very complex rules and actions that can even modify LoGS itself while it is running!

Because of the features shown here, I believe that LoGS is an excellent choice for cluster administrators who would like to get more out of their log analysis efforts.

```

; create a context for a PBS job
(make-instance
 'rule
 :match
 (lambda (message)
  (cl-ppcre::scan-to-strings
   (as-concatenated-string
    "[0-9]+.1102;Job Run at request of "
    "Scheduler@1102.alliance.unm.edu on hosts (.*)"
    (message message)))
 :actions
 (list
  (lambda (message matches sub-matches)
   (make-instance
    'context
    :name (format () "Job: ~A" (aref sub-matches 0))
    :match (lambda (message)
             (cl-ppcre::scan-to-strings
              (reduce-to-regexp
               (aref sub-matches 1))
              (message message))))))))))

; report on that job when it is finished
(make-instance
 'rule
 :match
 (lambda (message matches sub-matches)
  (cl-ppcre::scan-to-strings
   "Job;([0-9]+).1102;Obit received"
   (message message)))
 :actions
 (list
  (lambda (message matches sub-matches)
   (summarize-job-context
    (get-context
     (format () "Job: ~A"
              (aref sub-matches 0))))))))))

```

**Fig. 6.** A rule to report on all of the log messages from the machines in a PBS job.

## References

1. Top 500 List for November 2003, Nov 2003. Available from World Wide Web: <http://www.top500.org/lists/2003/11/>.
2. DFN-CERT Zentrum für sichere Netzdienste GmbH. DFN-CERT: Logsurfer Homepage, Dec 2000. Available from World Wide Web: <http://www.cert.dfn.de/eng/logsurf/>.
3. Todd Adkins. SWATCH: The Simple WATCHer of Logfiles, 2004. Available from World Wide Web: <http://swatch.sourceforge.net/>.
4. Paul Graham. What made Lisp Different, May 2002. Available from World Wide Web: <http://www.paulgraham.com/diff.html>.
5. Edi Weitz. CL-PPCRE Benchmarking, December 2003. Available from World Wide Web: <http://www.weitz.de/cl-ppcre/#bench>.
6. James E. Prewett. Analyzing cluster log files using Logsurfer, June 2003. Available from World Wide Web: [http://www.linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF0%3/A06-Prewett\\_J.pdf](http://www.linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF0%3/A06-Prewett_J.pdf).
7. James E. Prewett. LoGS, January 2004. Available from World Wide Web: <http://www.hpc.unm.edu/~download/LoGS.pdf>.
8. Pinchuan Ma. Log Analysis-Based Intrusion Detection via Unsupervised Learning, 2003. Available from World Wide Web: <http://www.inf.ed.ac.uk/publications/thesis/online/IM030059.pdf>.
9. CERT/CC. CERT Advisory CA-2000-17 Input Validation Problem in rpc.statd, September 2000. Available from World Wide Web: <http://www.cert.org/advisories/CA-2000-17.html>.
10. Marcus J. Ranum. artificial Ignorance: How-to Guide, September 1997. Available from World Wide Web: <http://archives.neohapsis.com/archives/nfr-wizards/1997/09/0098.html>.
11. Suzanne M. Kelly: Distinguished Member of Technical Staff, Sandia National Laboratories. Private Correspondence, February 2004.
12. T. Roney, A. Bailey, and J. Fullop. Cluster Monitoring at NCSA, June 2001. Available from World Wide Web: <http://www.ncsa.uiuc.edu/Divisions/CC/systems/LCI-Cluster-Monitor.html>.
13. David A. Dittrich. Analysis of SSH crc32 compensation attack detector exploit, November 2001. Available from World Wide Web: <http://staff.washington.edu/dittrich/misc/ssh-analysis.txt>.
14. Sasha. Holistic Approaches to Attack Detection, August 2001. Available from World Wide Web: <http://www.phrack.org/show.php?p=57&a=11>.

## 4 Acknowledgments

I would like to thank the following people (listed in no particular order) and organizations for all of their help and/or support for my log analysis research including the LoGS project:

- The Center for High Performance Computing at The University of New Mexico
- Sandia National Laboratories
- Dr. Robert A. Ballance
- Dr. Barney Maccabe
- Nancy Gough
- John P. Rouillard
- David Worth
- Christopher Jordan
- Sheryl Hurley
- James Laros
- My wonderful wife, Jeannette B. Zion