

# Deploying LoGS to analyze console logs on an IBM JS20

James E. Prewett

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

## 1 Introduction

In early 2005, The Center for High Performance Computing at The University of New Mexico acquired an IBM JS20[2] that has been given the name “Ristra”. The hardware consists of 96 blades, each with two 1.6 GHz PowerPC 970 processors and 4 GBs of RAM. The blades are housed in 7 IBM BladeCenter chassis. Myrinet is used as the high-speed, low-latency, interconnect. Included in the machine’s components is a management node which is an IBM x335 system.

This system uses the Xcat cluster management software from IBM[3]. The system was configured so that a management node would collect all of the system logs as well as all of the output to the console of the individual blades. Unfortunately, this logging of the blades’ console output put quite a heavy load on the system, especially the disk. It was our goal to also monitor scheduler logs on this machine by mounting the PBS MOM log directory via NFS to each of the blades. It seemed this would be quite problematic if we could not reduce the load on the administrative node.

Under certain conditions, the console logs were growing especially quickly. Sometimes the flood of messages indicated an error with the hardware or software on the blades themselves, or with that used to gather and store the console output from the blades. In other cases, the output seemed to indicate normal operation of the monitoring infrastructure of the machine. In either case, the output was rather verbose and was being written to disk. This was putting a heavy load on the disk sub-system.

In order to solve the problem presented by these log files, we decided to replace the log files (in `/var/log/consol/`) on disk with FIFO files that could be monitored by a log analysis tool. We decided to use LoGS as the tool to monitor these log FIFO files as it is capable of finding important messages and reacting to them. LoGS was then used to filter out innocuous messages, store important messages to files on disk, and react to certain conditions that it could repair without human intervention.

## 2 LoGS

LoGS is a log analysis tool, similar in many ways to Logsurfer[1], developed by the author to aid in monitoring cluster log files[4]. LoGS is a rule-based tool that allows for the ruleset to be modified while the program is running, this is used, amongst other things, to help to reduce the volume of the log data written to disk. LoGS is written entirely in Common Lisp<sup>1</sup>. Lisp is also the language used for both configuration and extension of LoGS.

<sup>1</sup> Currently LoGS is only supported on CMU Common Lisp. Ports to other Lisps are nearly done, but are a low development priority.

LoGS is particularly well suited for the task of monitoring log files from cluster machines. It is designed so that rulesets are essentially tree-shaped, allowing them to be more efficient than “flat” rulesets such as those available in Logsurfer. The messages a rule matches are specified by predicate functions which are compiled to machine code when running under CMU Common Lisp, allowing for fast execution. LoGS is also useful in this scenario as actions, called in response to a rule matching a message, are also Common Lisp functions allowing us an essentially limitless variety of responses to messages.

### 2.1 New LoGS extensions

In order for LoGS to be useful in monitoring these log files, LoGS had to be extended in a couple of ways<sup>2</sup>. First, LoGS had to be modified so that it is capable of monitoring multiple input files at a time. Because the log files in this case were FIFO files, LoGS had to be extended to handle these files specially<sup>3</sup>. Finally, because these log files do not indicate the host from which the log message originates, LoGS had to be extended to keep track of the name of the log file from which each incoming message originated<sup>4</sup>.

## 3 Ristra’s console logs

The log files in question consist of all of the output to the console of the blades of the system. These messages are collected via the “Serial over LAN” (SOL) functionality built into the machine. They also contain informational messages about the state of the machine, including the state of the SOL sub-system.

On this machine, we’ve found that at times the SOL sub-system needs to be reset. This is indicated this message: “SOL is not ready”. When this occurs, these messages are generated at the rate of approximately 150 messages per second per machine on which the SOL sub-system is inactive.

In this situation, we want to have an indication that the SOL sub-system had a problem, but simultaneously not be flooded by these messages. We use LoGS to write these messages to a special log file, and to filter out subsequent messages indicating the problem for a period of five minutes. This way, we will reduce the volume of log messages from approximately 45,000 messages in a five minute period down to only a single message that is recorded (written to disk) during that same period. Also, we use LoGS to react to this by:

- first determining the BladeCenter chassis that is having the problem and
- then running a program designed to reset the SOL sub-system on the BladeCenter chassis.

<sup>2</sup> LoGS is still in the early stages of development and is considered to be alpha software.

<sup>3</sup> By default, FIFO files are opened in “blocking” mode. Since input could be coming in from any of these files, the FIFOs have to be opened “non-blocking”.

<sup>4</sup> The name of the log file is then used to determine the host that the message is from.

In this case, the Xcat command “mpareset” is called with the appropriate BladeCenter as its argument in order to reset the SOL sub-system.

Since the output to the console is also collected, informational messages about the state of the machine that may not otherwise be visible are written to the log files as well. For these messages, we use LoGS to filter out messages indicating proper behavior of the machine. After filtering out these “uninteresting” messages, we use LoGS to alert system administrators to problems with the system.

For the console logs, we take the approach that any message that we are not explicitly filtering out is an interesting message. In this way, important messages that we have never seen before will be saved. This is done with the default rule in the LoGS ruleset which catches any messages not dealt with by other rules in the ruleset. This rule is illustrated in Figure 1.

#### 4 Filtering out innocuous messages from the log stream

Many of the messages that are recorded in these log files are merely informational. They often are the contents of the console output from interaction between the management software and the blades. This includes many lines of login interactions, etc.

These particular messages are not interesting to us. So, we tell LoGS to discard them, rather than storing them. Figure 2 illustrates one of these rules used to remove these messages from what is to be saved to disk.

#### 5 Responding to SOL failures

The SOL failures represent a real problem both due to the volume of the log messages generated and to the fact that remote console access is not available to system administrators when this occurs. From what we have seen so far, SOL failures can fall into one of three categories:

- SOL is unavailable on a single system,
- SOL is unavailable on all of the systems in a particular chassis, and
- SOL has been disabled on a single system.

We decided to allow LoGS to attempt to diagnose and automatically correct problems that can be resolved without disrupting users’ use of the system.

From our experience, the first class of failures can usually be repaired by simply rebooting the system in question<sup>5</sup>. The second class of failures requires that the SOL sub-system on the BladeCenter chassis be reset. The third failure requires that SOL be re-enabled for the system in question on the BladeCenter chassis. In order to be able to respond to the second class of failures, LoGS is instructed to run specific Xcat commands that are designed to repair the problem. In the third case, LoGS must call an expect script in order to interact with the BladeCenter chassis directly (via its telnet interface) and re-enable the SOL functionality.

---

<sup>5</sup> Obviously, rebooting a blade could potentially disrupt a compute job, so LoGS is not allowed to correct this problem.

With the first class of failures, we do not want LoGS to automatically react to the situation. The node in question may be in use by a compute job and we do not want to interrupt work being done on the system. However, it is important that administrators be made aware of the problem so that it may be repaired at their convenient. Figure 3 illustrates the rule used find these messages and to report them to the administrators.

The second and third classes of failures can be safely responded to without interrupting service on the machine by resetting the SOL sub-system on a chassis or by re-enabling the SOL for a particular node. The rule in Figure 4 illustrates a rule to respond to the SOL failures by resetting the SOL sub-system. Likewise, the rule in Figure 5 illustrates a rule to respond to SOL being disabled on a single blade.

## 6 Responding to disk errors

One way in which console logs can be especially useful is to discover problems with disk drives. Many of the indications of errors on the disks on the blades of this machine were logged only to the system console.

In the case of these failures, we want to gather the messages indicating problems with the particular device. If we see many messages of this sort, then certainly, we have a problem and a report should be sent to a system administrator to arrange to repair the disk. If this is a transient error, we merely want to record the messages to a log file.

Figure 6 illustrates a rule which will detect disk errors. Once an error has been detected, a rule will be created to add subsequent messages related to this disk to a context. If the context accumulates a certain number of messages before its timeout, then the contents of the context will be mailed to a system administrator.

## 7 LoGS performance

Our administrative machine is going to have several important responsibilities, including log analysis, so the log analysis program cannot consume too many resources. In the experiments we ran on Ristra, LoGS generally was not noticeable in the output from “top”, but at times would consume up to around 25% of the CPU. LoGS was seen to also use up to around 128 MB of memory, but only under a very high load of messages, at most times memory usage was around 10 MB. With the relatively complete ruleset described in this paper, LoGS was able to process over 70,000 incoming messages a second from a total of 97 different files!

## 8 Conclusion

LoGS can be used to aid in the administration and maintenance of cluster style system such as the IBM JS20. These sorts of systems require flexible log analysis tools in order to cope with the unique problems posed by the log files of these machines. Further, we ask that the tool be able to automatically respond to certain types of system failures.

The volume of the log data before the logs were replaced by FIFO files that were monitored by LoGS was at times greater than the amount that could be written to disk.

This caused the management node of the cluster to become rather slow and unresponsive. By utilizing LoGS in this way, the administrative node is now able to be more reactive to users as well as being able to serve more purposes for the cluster. For example, now that the IO load has been significantly reduced, this machine serves directories for the nodes to store their PBS MOM logs so that they may be analyzed in a manner quite similar to that described here for the console logs.

We have also shown how LoGS can be used to automate certain cluster administration tasks. In general, this allows for problems to be dealt with much more rapidly than waiting for a human to correct the situation. Also, the administrators are now freed up to address other issues.

## References

1. DFN-CERT: Logsurfer Homepage. <http://www.cert.dfn.de/eng/logsurf/>, Jan 2005.
2. IBM eServer BladeCenter. <http://www-1.ibm.com/servers/eserver/bladecenter/>, Jan 2005.
3. Egan Ford. [xcat.org](http://www.xcat.org/). <http://www.xcat.org/>, Jan 2005.
4. James E. Prewett. Listening to your Cluster with LoGS. Proceedings of the Linux Clusters: The HPC Revolution 2004, April 2004.

```
(make-instance
  'rule
  :match
  ;; match all messages
  (lambda (message)
    (declare (ignore message))
    t)
  :actions
  (list
   ;; write message to /var/log/LoGS/missed/<node name>
   (file-write
    (format () "/var/log/LoGS/missed/~A"
              (car
               (last
                (CL-PPCRE:SPLIT
                 "/"
                 (from-file message))))))))))
```

**Fig. 1.** A LoGS rule to catch messages not explicitly dealt with.

```

;; a rule to filter out "username: USERID" messages
(filter
  (lambda (message)
    (cl-ppcre::scan
      "^username: USERID"
      (message message))))

```

**Fig. 2.** A LoGS rule to filter out login prompt messages.

```

;; match a single "SQL is not ready" message
;; create a rule to ignore further messages from
;; this host for 5 minutes
(make-instance
  'rule
  :match
  (lambda (message)
    (multiple-value-bind (match-start match-end)
      (cl-ppcre::scan "SQL is not ready" (message message))
      (when match-start (values t `((from-file ,(from-file ←
        message)))))))
  :continuep t
  :actions
  (list (file-write
        (format () "/var/log/LoGS/important/logfile"
          (car (last (CL-PPCRE:SPLIT "/" (from-file ←
            message))))))
        ;; ignore further messages from this host for 5 minutes
        (lambda (message)
          (rule-before
            (make-instance
              'rule
              :timeout (+ *now* (* INTERNAL-TIME-UNITS-PER-SECOND←
                300))
              :environment env ;; grab environment from return
              ;; of match function
              :match
              (lambda (message)
                (and (equal from-file (from-file message))
                  (cl-ppcre::scan "SQL is not ready" (message ←
                    message))))))))))

```

**Fig. 3.** A rule to notice single-blade SQL failures and ignore subsequent messages for a period of five minutes.

```

(make-instance
 'rule
 :match
 (lambda (message)
  (multiple-value-bind (match-start match-end)
    (cl-ppcre::scan "SOL is not ready" (message message))
    (when match-start (values t `((from-file ,(from-file ←
      message)))))))
 :actions
 (list
  ;; make a context to see if other nodes on this chassis are ←
  having
  ;; the same problem and a rule to add messages to that ←
  context
  (lambda (message)
    (ensure-context
     ;; once we have seen all 14, signal problem
     :max-lines (- 14 1)
     :timeout (+ *now* (* INTERNAL-TIME-UNITS-PER-SECOND 60))
     :name (format () "problems on BladeCenter: ~A"
      (gethash
       (car (last (CL-PPCRE:SPLIT "/" (from-file ←
        message))))
       *NODE-TO-CHASSIS-HASH*))
     :actions
     (list
      (lambda (context)
        (when (< 13 (ecount context))
          (extensions:run-program "/opt/xcat/bin/mpareset"
           `((gethash
            (car (last (CL-PPCRE:SPLIT
             "/" (from-file (aref (data context)←
              0))))
            *NODE-TO-CHASSIS-HASH*)))))
          ;; add this log to the context
          (lambda (message)
            (add-to-context
             (format () "problems on BladeCenter: ~A"
              (gethash (car (last (CL-PPCRE:SPLIT "/" (from-file ←
                message))))
                *NODE-TO-CHASSIS-HASH*))
             message))))
      message))))

```

**Fig. 4.** A rule to gather all “SOL not ready” messages from a chassis into a context. If all blades in the chassis report a failure, “mpareset” is called to correct the situation.

```

;; match "SOL on blade is not enabled" messages
;; write them to /var/log/LOGS/important/<hostname>
;; create new rule to cause "SOL on blade is not enabled" ←
    messages to be
;; ignored for the next 5 minutes (to avoid floods to the ←
    log files).
;; call an expect script to re-enable SOL if possible.
(make-instance
 'rule
 :match
 (lambda (message)
  (multiple-value-bind (match-start match-end)
    (cl-ppcre::scan "SOL on blade is not enabled" (message ←
    message))
    (when match-start (values t `((from-file ,(from-file ←
    message)))))))
 :actions
 (list (file-write
  ;; write it to the log file
  (format () "/var/log/LOGS/important/~A"
    (car (last (CL-PPCRE:SPLIT "/" (from-file ←
    message))))))
  ;; ignore further messages from this host for 5 minutes
  (lambda (message)
    (rule-before (suppress
      (lambda (message)
        (and (equal from-file (from-file ←
        message))
          (cl-ppcre::scan "SOL on blade is ←
          not enabled"
            (message message ←
            ))))
          300
          :environment env)))
  ;; run the solenable script to re-enable SOL
  (lambda (message)
    (extensions:run-program
     "/opt/xcat/bin/solenable"
     `((, (car (last (CL-PPCRE:SPLIT "/" (from-file message ←
     )))))))))

```

**Fig. 5.** A rule to find systems with SOL disabled and then call a script to re-enable SOL.

```

;; notice disk errors
;; gather them for 30 seconds and mail them all to an admin
(make-instance
 'rule
 :match
 (lambda (message)
  (multiple-value-bind (match sub-matches)
    (cl-ppcre::scan-to-strings "Buffer I/O error on device↵
                               (\\w+), .*"
                               (message message))
    (when match (values T `((device ,(aref sub-matches 0))))↵
    ))
 :actions
 (list
  (lambda (message)
   (rule-before
    (make-instance
     'rule
     :timeout (+ *now* (* INTERNAL-TIME-UNITS-PER-SECOND 30))
     :environment env
     :match
     (lambda (message)
      (when (cl-ppcre::scan-to-strings (cadar env) (message ↵
                                       message))
        (values t env)))
     :actions
     (list
      (lambda (message)
       (add-to-context
        (format () "disk errors on host ~A" (from-file ↵
                                             message))
        message))))))
    (lambda (message)
     (ensure-context
      :name (format () "disk errors on host ~A"
                    (from-file message))
      :max-lines 1024 ; not likely to be exceeded
      :timeout (+ *now*
                  (* INTERNAL-TIME-UNITS-PER-SECOND
                    30))
      :actions
      (list
       (lambda (context)
        (mail context "systems@hpc.unm.edu" (name context))))↵
      ))
  ))

```

**Fig. 6.** A rule to gather messages indicating disk errors and mail them to an administrator.