MOODLE MONITORING BEST PRACTICES

Jonathan Barber, Rodolfo Matos, Susana Leitão

University of Porto (PORTUGAL)

Abstract

Learning Management Systems (LMSs) provide their greatest value when they are unobtrusive and don't distract users from their education. Two aspects govern the obtrusively of an LMS: the user interface and the system's performance. If the interface is logically designed and intuitive then users will quickly learn it, if the LMS is performant and responds quickly to user actions, then they will remain focused on their activities. Here we discuss metrics that describe the behaviour of a installation and can aid in troubleshooting performance related issue. We focus on the Moodle LMS running on the Linux operating system (although the metrics are applicable to other LMSs and other operating systems) and examine all of the layers of the applications stack, from the storage system through the operating system to the database and web server and into the LMS environment before finishing at the network system. We also illustrate the use of monitoring tools to display the real time metrics and their historical values which can be used to show usage trends and for capacity planning.

Keywords: Moodle, Linux, monitoring, security, prevention, auditing.

1 INTRODUCTION

Learning is an activity best performed without distractions, classically in the hushed environs of a library or the quiet classroom. The electronic environment of a Learning Management System (LMS) is no different, and they perform best when they don't distract students from learning. Besides the quality of teaching material within the LMS, two principal aspects govern how obtrusive an LMS is; the user interface (UI), and the speed with which the LMS responds. If the LMS has a poorly constructed UI which requires significant effort to navigate, or if the LMS is slow to respond to user interaction, then there is a risk the student's concentration will be broken and their learning performance harmed.

The speed with which an LMS responds can be measured through a metric called latency. Latency is the length of time between a user action (e.g. clicking on a link) and the corresponding response is received (being shown the page the link refers to). Lower latency is generally considered to be better, and has been demonstrated to lead to increased user satisfaction in commercial web environments [1].

Both the UI and performance of web based LMSs can be monitored so that issues can be identified and addressed. Continuous monitoring can detect abnormal situations and allow the LMS administrator to take remediative action. Additionally, retention of past monitoring metrics gives a basis for evidence based capacity planning in the future.

The UI can be monitored through user testing and analytics software (such as Google Analytics) and are not the focus of this paper. Latency measurements are most accurately reported by browser tools such as Mozilla's FireBug, but can be found in the LMS web server logs when each request and corresponding response has a timestamped entry. However, these latency measurements are the total time required to service a request and don't provide enough information to determine why a LMS is "slow". We illustrate how to determine why an LMS might be "slow" by following the lifecycle of a user's request to the Moodle LMS installed on the Linux operating system, highlight key metrics that contribute towards LMS latency and provide troubleshooting tools to help LMS administrators improve the productivity of their students by reducing the latency of their LMS.

Finally we describe tools and methods that provide continuous monitoring of an LMS environment and allow historical comparisons, capacity planning and raise alerts if problems are detected.

The metrics we describe are based in our experiences of administering the University of Porto's Moodle installation, which has 40,000 enrolled and active users.

2 THE LIFECYCLE OF A REQUEST

In this section we will document how a request from a user is handled by a LMS and we will describe tools and metrics that describe the state of the LMS and operating system in each step in the request's journey through its lifecycle.

The request lifecycle described in the following subsections is based on the Moodle LMS installed on a Linux OS with a MySQL database and the Apache web server (httpd) with mod_php. The exact versions of Moodle, Linux, MySQL and httpd are not relevant and the description is applicable to other LMSs, web servers and databases (although some of the details may differ).

2.1 How A Browser Interacts With A Web Based LMS

The user enters an web address (e.g. "http://example.com/moodle") in their browser address bar and hits "Go". Their browser looks up the host in the address ("example.com") and sends the request for the address to that host. The host runs a web server which is listening for browser requests, when the web server receives the request it passes it to the LMS software which examines the address and composes and sends the response. The response is normally a HTML document that can contain addresses of other resources in the LMS. When the browser receives the HTML document it reads it and makes any additional requests for other resources (such as images) that are required to fully display the page to the user. The time between the user hitting "Go" and the page being displayed is the latency for the whole page, which is composed of the latency for each of the resources that the page requires. Because the browser doesn't know the resources required to display for a requested address, the latency for a whole page is greater than the individual resources.

2.2 Request/Response Lifecycle

The lifecycle of a request and response is illustrated in Figure 1 and the major actions in the figure are as follows:

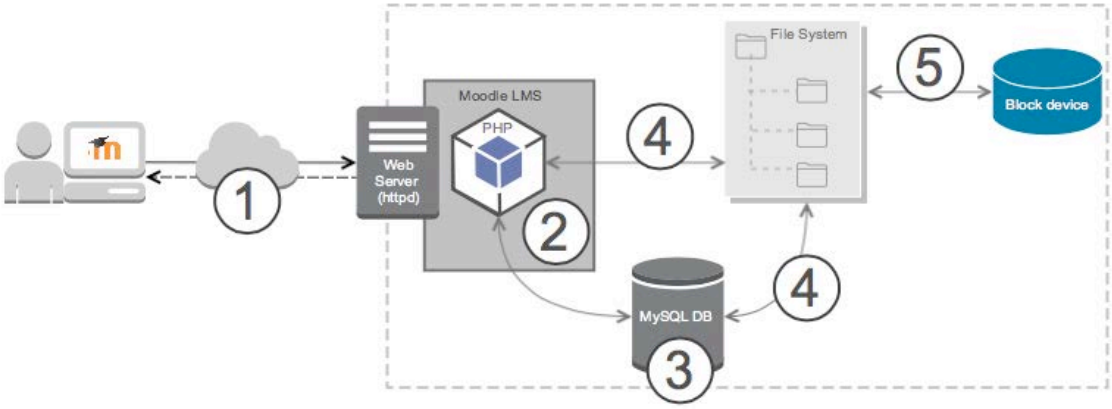

Fig. 1: User request and its lifecycle.

1. The user request is sent across the network to the web server
2. The web server receives the request and passes it to Moodle, Moodle then parses the request and determines the actions it needs to take to fulfill the request - normally this involves creating a HTML document from the result of many MySQL queries
3. MySQL parses the Moodle queries and searches its database (DB)
4. The MySQL DB files and static artifacts (e.g images) are accessed via a file system
5. The file system is stored on a block device such as a hard drive

We will now examine each of these stages with the tools that report metrics that describe their state.

2.2.1 Request Crosses Network

Before the LMS can respond to requests, those requests must first be transmitted over the network to the LMS and this introduces the first source of latency. It is difficult to evaluate the network performance as it's not normally possible to monitor the steps between the client and the LMS.

However, it is possible to examine the status of the network on the server. `iftop` shows the amount of bandwidth consumed in real time broken down by host. This allows you to see if there are particular remote hosts that are consuming a lot of network capacity. "`ip -s link`" shows the statistics for the network interfaces which includes errors. "`netstat`" and "`ss`" show the network connections that the server has and the state they are in: the number of these connections will tend to change dramatically when a problem exists and either the connections will drop to a low number (because the server isn't being contacted by many clients) or reach extremely high levels (because the server is unable to respond and close the connections or many more clients are trying to connect to the server than normal).

2.2.2 Web server and Moodle

2.2.2.1 Web server state

Once the request reaches the web server, the server passes the request to Moodle.The status of Apache can be monitored in real time with the "`apachetop`" command [2].

The general health of the system can be monitored with the `top` command and observing the load, CPU and RAM utilisation in real time.

```
top  -  13:53:17  up  85  days,   8:13,   1  user,   load  average:  0.14,  0.24,  0.19
Tasks:   645   total,     1   running,   644   sleeping,     0   stopped,     0   zombie
Cpu(s):   0.2%us,   0.1%sy,   0.0%ni,  99.7%id,   0.0%wa,   0.0%hi,   0.0%si,   0.0%st
Mem:    32865296k   total,   32307672k   used,     557624k   free,     212632k   buffers
Swap:    8388600k   total,     334936k   used,    8053664k   free,    26779832k   cached

 PID   USER         PR    NI    VIRT    RES    SHR   S   %CPU   %MEM      TIME+    COMMAND

4126   apache       20     0    724m    90m    64m   S    3.7    0.3      0:55.83   httpd

21366   apache       20     0    735m   119m    89m   S    1.0    0.4      2:22.02   httpd

18417   apache       20     0    722m    90m    64m   S    0.7    0.3      0:41.19   httpd

 109 root      20   0     0    0    0 S  0.3  0.0   2:28.40 events/10
(...)
```

The `ps` command allows you to list the processes running on the host. It can display a large amount of information about the processes, but the simple metric of the number of running httpd processes is often useful on its own. This number should correlate the number of active LMS users, so if you see an unexpectedly large or small number of processes it could indicate a problem:

```
[root@moodleas1 ~]# ps -C httpd -o cmd h | uniq -c
    65 /usr/sbin/httpd
```

It can be helpful to run "`ps auxf`" which reports all of the running processes on the host - this gives a good overview of the whole system which can tell you if something is wrong if you know what the "normal" state looks like.

Changes in the hosts memory and CPU usage can be viewed with the "`vmstat`" command. The argument 10 makes the command reported the vaues every 10 seconds:

```
[root@moodleas1 ~]# vmstat 10
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
2  0 334952 594364 209136 26740116    0    0   151    99    0    0  1  0 98  0  0
1  0 334952 590032 209160 26742136    0    0     2   124 6960 25352  1  3 96  0  0
1  0 334952 585648 209176 26744160    0    0     3   132 6322 24247  0  3 97  0  0
(...)
```

The "`r`" and "`b`" columns report the number of processes running and blocked, large "`b`"'s tend to be a bad sign as it indicates a problem is stopping processes from executing.

Of particular interest are the swap-in (“si”) and swap-out (“so”) columns. These represent data being moved to and from physical memory (RAM) to the swap disk, and large values mean that the machine is “swapping” data between RAM and the swap disk - this is normally because the machine is running out of RAM. As the swap disk is much slower than RAM, large values in these columns mean that performance will be terrible. Related to this is the “swpd” column, which shows the amount of data stored on the swap disk. Having a non-zero value here is not a bad sign in of itself, as Linux is aggressive at moving unused data in RAM to the swap disk.

The IO columns “bi” and “bo” give an indication of how much data is being read and written to the block devices (such as hard drives) on the system. Lower values mean that little data is being accessed from disk, which can mean that the system is idle, or that a large amount of data is being read from the cache (the size of which is reported in the “cache” column).

The CPU columns report the percentage of time the CPUs in the system spent executing user code (“us”), system or kernel code (“sy”), waiting for data from disk devices (“wa”) or doing nothing (“id”). Large values in “wa” are normally bad here, as it indicates the system is waiting to read or write data to hard drives and can’t do anything until this is complete. Complementing these tools is the “sar” command, which runs every 10 minutes and captures many of the metrics displayed by the tools above. Because of the number of metrics it captures, it would require too much space to describe fully here, but we recommend that you read the documentation [3] and enable it.

2.2.2.2 Moodle state

The LMS application can reveal important information that’s hidden at the operating system level as it’s specific to the LMS environment. The amount of information that’s available and how easy it is to access will vary depending on the LMS you use. For example, we have found the following metrics to be useful for Moodle, but exposing some of them required writing custom database queries:

- The number of logged in users
- The number of active users - these are users who are logged and have accessed the LMS recently. They can be counted globally and on a per course basis
- How many enrolled users could be taking quizzes, and of those how many are active (this can be counted globally and per quiz)
- Usage statistics for various activities and resources (such as forums, chats, submissions) to understand the context in which they are used
- How many backups and restores are being performed

2.2.3 MySQL handles the Moodle queries

LMSs depending heavily on databases to store structured information about their users and all of the features the LMS provide. Therefore problems in the database can lead to poor performance in the LMS. Unfortunately, databases are themselves complicated pieces of software that can require great expertise to troubleshoot. Therefore, this section is a necessarily brief overview of a large subject and is focused on MySQL. Other databases will have their own methods for troubleshooting and reporting metrics. To examine the database status, first you must connect through a client such as the command line tool mysql.

A simple way of seeing the MySQL database state is through the command “show global status”. In particular the values for Threads_connected and Threads_running variables are interesting as they respectively report the number of connections to the server and the number of queries running. Assuming the database serves only Moodle, Threads_connected should be 2 - as one connection is required for Moodle, and another connection is needed for the client reading the global status. Threads_running should correlate with the number of clients connected to the LMS - as a busy server should have more running queries.

So called “slow queries” are defined as queries that take more than *N* seconds to run, and are a problem because they can block shorter queries from executing. *N* is defined by the MySQL variable long_query_time and is 10 seconds by default. Because small numbers of slow queries can have disproportionate effects, it’s important to enable slow query logging and monitor the slow query log [4].

One source of slow queries is when the buffer for the database tables is too small to hold all the information needed by the database. This is called the InnoDB buffer pool and its size is controlled by

the variable `innodb_buffer_pool`. If your buffer pool is too small then data in the database will not be held in RAM and will have to be read repeatedly from the relatively slow block devices. It's possible to view the statistics of the buffer pool through the global status and the variables starting with `'Innodb_buffer_pool_pages'`.

Tuning parameters such as the `innodb_buffer_pool` is complex, and care must be taken when changing these values. Many guidelines exist on the internet and it is best to experiment with them in a test environment before implementing them in production. We have found the following documents to be useful in the past [5],[6],[7],[8]

In addition, we found the Percona Toolkit [9] and Monitoring Plugins [10] to be very useful in administering MySQL.

2.2.4 MySQL And Static Artifacts Stored On A Filesystem

The file system doesn't offer many metrics to monitor, the main one are how much space is being used by the filesystem cache, which is visible with the "`vmstat`" command. Advanced information of how much space is being used by different parts of the file system cache can be seen with the "`vmstat -m`" command, which displays information about the the various kernel memory structures, although this is not normally needed.

2.2.5 The file system is stored on block device(s)

The file system is stored on one or more block devices, which are normally disk drives. You can see which file systems are stored on which block devices using the "`mount`" command. The performance of block devices are the main determinant of how quickly data can be accessed, and generally determine the speed of a system if CPU and RAM are not bottlenecks. Block device activity can be monitored with the command "`iostat -kx 10`". This continuously monitors all block devices on the system and every 10 seconds displays the changes since the previous report was generated with data volumes in kilobytes:

```
[root@moodleas1 httpd]# iostat -kx 10
Linux 2.6.32-431.23.3.el6.x86_64 (moodleas1.up.pt)     20-01-2015     _x86_64_    (24 CPU)


avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          1,38    0,02    0,31    0,24    0,00   98,06


Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await
 svctm  %util
sdb               0,00     0,00   17,84    3,44   847,32   559,09   132,19     0,06    3,05
  1,01   2,15
sdc               0,00     0,00    0,34    0,00     1,38     0,00     8,08     0,00    0,00
  0,00   0,00
sda               1,33    23,66   11,70    8,95   230,04   130,51    34,92     0,03    1,40
  0,95   1,96
sdd               0,00     0,00    0,30    0,00     1,19     0,00     8,09     0,00    0,00
  0,00   0,00
sde               0,00     0,00   17,84    3,39   847,00   556,99   132,29     0,06    3,06
  1,02   2,16
sdf               0,00     0,00   17,84    3,47   847,48   559,42   132,04     0,06    3,04
  1,01   2,14
sdg               0,00     0,00    0,30    0,00     1,21     0,00     8,09     0,00    0,00
  0,00   0,00
sdh               0,00     0,00    0,30    0,00     1,20     0,00     8,09     0,00    0,00
  0,00   0,00
sdi               0,00     0,00   17,84    3,41   847,20   555,66   132,07     0,06    3,05
  1,01   2,15
dm-0              1,19    11,36   71,36   13,70  3388,99  2231,17   132,15     0,27    3,15
  0,75   6,37
dm-1              0,00     0,00   72,52   25,07  3388,89  2231,17   115,18     0,42    4,32
  0,65   6,35
```

The report starts with a summary of CPU utilization (as given by `top` and `vmstat`) and then for each block device (the "`Device`" column) lists a series of metrics. The column to examine first is "`%util`" which shows the percentage of CPU time for which the device was busy. A value of 100% means that

the device was constantly busy and so is likely a source of slowness on the system. The "await" column then tells you how long read and write requests took to service - larger times means the device was busier and will lead to higher latency responses from the LMS. The other columns report the rate of reads and writes in operations per second ("rrqm/s", "wrqm/s", "r/s", and "w/s") and the amount of data that's read and written in kilobytes per second ("rkB/s", "wkB/s").

3 AUDITING AND ANALYSIS

Section 2 described tools to display metrics at a point in time. To know whether these metrics are abnormal (for example "is the current level of CPU utilization normal"?) requires implicit historical knowledge of the system by an administrator who has experience of how the "normal" system appears. However, this dependency on fallible human memory can be augmented through historical information that exists in various locations on the operating system and the LMS, and extended through the use of monitoring solutions.

3.1 LMS (Moodle) Status

Historical information specific to the LMS is generally available within the LMSs database. This information can be used beyond the simple metrics described in Section 2.2.2.2 to analyse user behaviour within restricted environments such as exams and quizzes - as we have previously demonstrated with Moodle and our MoodleWatcher proctor project [11][12]. Additionally, we have used the data to create dashboards displaying information useful to our LMS administrators, such as calendar views of quizzes. This provides an easy way to see when critical activities are taking place, therefore ensuring that support is available without exam proctors having to request it and avoiding maintenance work during these periods.

3.2 Log files

The operating system log files (typically found under the /var/log directory) are a rich source of information. The web server Apache log is particularly useful as it contains a record of every request and response, its utility is enhanced if it is configured to log the length of time responses take with the LogFormat [13] directive %D [14]. This is the best way to monitor the latency of responses, as the latency of every response is captured and can be analysed with a tool such as GoAccess [15]. An example of the configuration required to capture the response time is to first create a log format called "latency" in the apache2.conf file:

```
LogFormat "%h %l %u %t \"%r\" %>s %O (%D)" latency
```

and then to use this format to log to the latency.log file:

```
CustomLog ${APACHE_LOG_DIR}/latency.log latency
```

which creates the following example log lines:

```
192.168.3.1 - - [20/Jan/2015:14:08:19 +0000] "GET /pix/moodlelogo.png HTTP/1.1" 200 2734 (10605)
192.168.3.1 - - [20/Jan/2015:14:08:20 +0000] "GET /theme/clean/pix/favicon.ico HTTP/1.1" 200 1449 (219)
(...)
```

Here it can be seen that it took 10605 microseconds to serve the moodlelogo.png compared to 219 microseconds for the favicon.ico

3.3 Changing files

We have found it useful to monitor whether files associated with the LMS are changing. Such files includes those uploaded by the LMS users (such as images and PDFs) and the LMS source code and configuration, which could be changed by the LMS administrators for reasons such as site customization. The contents of a file can be summarized through its checksum, which can be found for all files in a directory hierarchy with the following script:

```
#!/bin/sh
# Takes an argument a list of paths to directories and checksums all of the files within them
find $* -type f -print0 | xargs -0 md5sum
```

3.4 Monitoring Tools

Monitoring tools take the metrics we have described in Section 2 and 3, and continuously record their values. These can then be used as a dashboard to view the current status of the LMS or to analyse historical trends in the LMS. This allows you to, for example, determine whether the number of users is growing and estimate what impact this might have on the server CPU utilization and therefore guide future hardware purchases.

Examples of open source monitoring tools are:

- Nagios [16]
- Zabbix [17]
- Zenoss [18]

In addition to the recording of metrics, these systems can be configured to send alerts to the LMS administrator when a problem is detected such as when CPU utilization is higher than expected or the LMS can't be contacted. They also allow multiple systems to be monitored, so if the LMS and database servers are on different hosts, the data from the systems can be correlated. In addition, probes can be placed remotely in the network to monitor access to the LMS from different physical locations. This can be useful when an academic institute is not on a single campus and you wish to be notified if other campuses have trouble accessing the LMS.

4 CONCLUSIONS

We have described just a small set of the monitoring possibilities that are available. Although monitoring is a wide ranging topic, which can require a depth of knowledge, it is critical to providing a high quality service and should be used on a daily basis and not only when planning a new platform. Monitoring allows both the detection and resolution of problems and the proactive prevention of problems caused by increased usage of the LMS. Historical records of metrics provide an evidential basis for capacity planning and resource allocation.

REFERENCES

[1] http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html

[2] https://github.com/JeremyJones/Apachetop

[3] http://sebastien.godard.pagesperso-orange.fr/man_sar.html

[4] http://dev.mysql.com/doc/refman/5.1/en/slow-query-log.html

[5] http://dev.mysql.com/doc/refman/5.6/en/optimize-overview.html

[6] https://docs.moodle.org/22/en/Performance_recommendations#MySQL_performance

[7] http://www.singlehop.com/blog/analyzing-mysql-performance-and-bottlenecks/

[8] http://www.sitepoint.com/optimizing-mysql-bottlenecks/

[9] http://www.percona.com/software/percona-toolkit

[10] http://www.percona.com/software/percona-monitoring-plugins

[11] Matos, R., Torrão S., Vieira T. (2012) "Moodlewatcher: Detection and prevention of fraud when using Moodle quizzes" INTED2012 Abstracts CD (ISBN: 978-84-615-5562-8), INTED2012 Proceedings CD (ISBN: 978-84-615-5563-5).

[12] Matos, R., Torrão S., Vieira T., Carvalho F. (2012) "Moodlewatcher: One year experience of detecting and preventing fraud when using Moodle quizzes" EDULEARN12 Abstracts CD (ISBN: 978-84-695-3176-1), EDULEARN12 Proceedings CD (ISBN: 978-84-695-3491-5).

[13] http://httpd.apache.org/docs/2.2/mod/mod_log_config.html#logformat

[14] http://httpd.apache.org/docs/2.2/mod/mod_log_config.html#formats

[15] http://goaccess.io/

[16] http://www.nagios.org/

[17] http://www.zabbix.com/

[18] http://www.zenoss.org/