Software Usage Logging Using Domain Name Service (DNS)

by Richard C. Angelini, Brian M. Rapp, and Howard M. Kash

ARL-TR-6104

September 2012

Approved for public release; distribution is unlimited.
NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer’s or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.
Software Usage Logging Using Domain Name Service (DNS)

Richard C. Angelini, Brian M. Rapp, and Howard M. Kash
Computational and Information Sciences Directorate, ARL

Approved for public release; distribution is unlimited.
The Data Analysis and Assessment Center at the U.S. Army Research Laboratory required a method to capture software utilization data from both client workstations and high-performance computing (HPC) resources. As part of an overall effort to understand utilization across all machines in the U.S. Department of Defense HPC Modernization Program, we looked to develop a method that would allow for accurate, centralized logging of application usage from a wide variety of geographically dispersed systems. This report presents a method of logging software utilization data by means of domain name system queries.
INTENTIONALLY LEFT BLANK.
1. Introduction

The Data Analysis and Assessment Center (DAAC) at the U.S. Army Research Laboratory (ARL) required a method to capture software utilization data from both client workstations and high-performance computing (HPC) resources. As part of an overall effort to understand utilization across all machines in the U.S. Department of Defense (DOD) High-Performance Computing Modernization Program (HPCMP), we looked to develop a method that would allow for accurate, centralized logging of application usage from a wide variety of geographically dispersed systems. We have long felt that an accurate accounting of how certain data-analysis tools (VisIt, ParaView, and EnSight) are utilized has been under reported due the nature of capturing usage data in a client-server environment. Existing monitoring facilities on the HPC systems are not able to properly capture all of the data required to effectively analyze this utilization; and there is virtually no method to capture utilization data from the client-side Windows, Linux, and OSX workstations. An effort was initiated by the ARL Computational Science Environment (CSE) team which successfully utilized a webserver-based method of capturing usage data within ARL, but this method is not practical for gathering data from across the HPCMP due to numerous network security policies. The goal of this new initiative was to develop and implement a solution unencumbered by network security policies that would allow client workstations and HPC systems to log usage data associated with various data analysis packages to a centralized repository where the data could be easily monitored and analyzed. The results of this utilization analysis will assist HPCMP and DAAC personnel in justifying how resources should be allocated for application support, which customers are using a particular software application, and how that application is using computational resources on the various HPC systems.

2. Methodology

The goal of the data collection was to obtain as much information as possible from as many disparate systems as possible without having special client software or connectivity requirements. The original CSE method used a local web server for collecting logs. Client systems would post utilization data to the web server where it would be analyzed. While this method worked fine for a local area network, it was not as attractive for broader implementation due to the requirement for a publicly accessible web server and HPC systems (particularly cluster compute nodes) not having access outside of the local network. However, most systems will have access to the local/site domain name service (DNS) server for resolving hostnames and internet protocol (IP) addresses. The local DNS server in turn will have external access in order
to perform queries for hostnames and IP addresses that are outside of the local domain. Therefore, encoding utilization data into a DNS query can be used to get the data to a central location for analysis.

2.1 Domain Name System

DNS is the means by which computers translate a human-readable name like “www.example.com” into an IP address like “192.0.43.10” and vice-versa. When the need for such a translation arises, the computer sends a query to its local DNS server (called a caching resolver), which will then forward the query to another DNS server that is authoritative for the “example.com” domain. That authoritative server provides the answer to the query and can also log a record of the transaction to a local file. If we control one of these authoritative name servers, then we can use the following method to capture encoded messages.

2.2 DNS Configuration/Logging Subdomain

We start by creating a subdomain specifically for collecting utilization data. The purpose of creating a subdomain is so that the encoded queries can be directed to a specific server, which could be one of the existing authoritative servers or a server dedicated to the data collection process. Otherwise, the query could go to any of a number of servers that are authoritative for the domain since there are typically three or more servers that are authoritative for a domain for redundancy. This would require access to all of the authoritative servers for the domain and the data would have to be collected from all of the servers and aggregated. It also makes it easier to parse out the encoded queries since they have a unique domain. We will use logging.example.com as the subdomain for illustrative purposes in this report and Berkeley Internet Name Domain (BIND) DNS server syntax. The delegation in the parent (example.com) zone would look like this:

```
logging IN NS ns1.example.com,
```

where ns1.example.com is the DNS server where the queries will be collected and analyzed.

An example of the logging.example.com zone file that would be on ns1.example.com is shown in figure 1. The optional wildcard record (‘*’) can be used so that all queries ending with “logging.example.com” will return a specific IP address, such as “1.1.1.1.” This could be used by the client logging script to verify that the query was successfully received and, if not, try a different method or print a notice to the user. It has a low, 5-s time-to-live (TTL) to prevent long-term caching.

---

Figure 1. Sample logging.example.com zone file.

The BIND server on ns1.example.com must also be configured to do query logging, such as with the following logging stanza in named.conf:

```
logging {
  channel query_log {
    file "/var/log/query.log";
    severity info;
    print-category no;
    print-severity yes;
    print-time yes;
  };
  category queries { query_log; };
};
```

In some situations, such as a cluster system where back-end computer nodes do not have direct network access in order to send DNS queries to the local resolver, it may be necessary to set up a DNS forwarder. On a cluster node that has access to both the internal cluster network and the external network (a login node, for example), install the current BIND RPM or compile the latest version of BIND from source. Configure the named.conf file as follows so that it only listens for queries on the internal network, and forwards them to the local resolver(s) on the external network. In this example, 172.16.100.0/24 is the internal network and 172.16.100.100 is the internal address of the login node on which BIND is running; the site resolvers on the external network are 192.168.2.10 and 192.168.3.10.

```
$TTL 3600
@    IN      SOA   ns1.example.com. hostmaster.example.com. ( 2012051101 ; Serial No. 3600 ; Refresh 1200 ; Retry 1209600 ; Expire 5 ) ; Negative Cache TTL

IN      NS      ns1.example.com.
*
5      IN      A      1.1.1.1
```
options {
    listen-on { 172.16.100.100; };  
    allow-query {172.16.100.0/24; };  
    allow-query-cache {172.16.100.0/24; };  
    allow-recursion {172.16.100.0/24; };  
    forwarders { 192.168.2.10; 192.168.3.10; };  
    forward only;  
    max-ncache-ttl 1;  
};

On each of the back-end compute nodes, the resolver configuration file (e.g./etc/resolv.conf) would list the address of the forwarder as the nameserver:

    nameserver  172.16.100.100

2.3 Query Strings

The various information fields to be transmitted must be encoded into a single string which will be presented to the DNS server as a fully qualified domain name for translation. If periods are not present in the data, they can be used to separate the fields just as ordinary DNS addresses use them to separate subdomains. If more than one field contains periods, creative use of different field delimiters will be necessary. Adding a timestamp field to the string prevents DNS caching from interfering with our request to the authoritative name server since each query, unless sent with the exact same data within a one second interval, will be unique.

2.4 Logging Scripts

As part of the launching script for the software to be logged, we will add additional code to send our DNS query. The “nslookup” command, which is available on virtually every operating system, can be used to generate just such a query. Other common commands, such as ping or dig, or a custom C or Perl module could also be used. The logging script can take the fields to be logged as arguments. In a Windows batch file, the commands can resemble this:

```
set COMPUTER=%COMPUTERNAME:=%
set TIMESTR=%TIME:.=%
nslookup -nosearch %COMPUTER.%2.%3.%TIMESTR%.logging.example.com
```

The UNIX shell script commands look similar:

```
HOST=`/bin/hostname -s`
TIMESTR=`/bin/date +%s`
nslookup -nosearch $HOST.$1.$2.$3.$TIMESTR.logging.example.com
```

To log the program name and two more arguments with one of these scripts, we provide three arguments separated by spaces:

```
logging-script example-program argument2 argument3
```
2.5 Parsing Script

The UNIX scripting language Perl is effective at matching regular expressions, so we use it to search the DNS query logs for the data generated by our logging scripts. The File::Tail extension allows our Perl script to read continuously from the DNS query log, and resume reading when the file is rotated to conserve disk space. Our output file is a list containing only the data fields of interest, and it can be readable by those who have no need-to-know for the rest of the DNS query logs. An example parser is shown here:

```perl
#!/usr/bin/perl
use File::Tail;       # Matija Grabnar's Tail.pm
use strict;

my $LOGGING_DOMAIN = "logging.example.com";
my $querylog = "/usr/local/domain/var/log/query.log";
my $logfile = "/tmp/logging.logfile";

tie *INPUT,"File::Tail",{name=>$querylog,debug=>0,
  interval=>1,maxinterval=>5,reset_tail=>100,
  tail=>0, ignore_nonexistent=>1,
  adjustafter=>10, errmode=>"return"};

open (OUTPUT,">$logfile") or die "Can't append to $logfile: $!";
select((select(OUTPUT), $| = 1)[0]);    # Auto-flush writes to OUTPUT

while (<INPUT>)
{
  if ($_ =~ /(^S+) (\S+) info: client .+ query: ([0-9a-zA-Z-\/_]+)\.(\[0-9a-zA-Z-\/_]+)\.(\[0-9a-zA-Z-\/_]+)\.(\[0-9a-zA-Z-\/_]+)\.(\[0-9a-zA-Z-\/_]+)\.(\[0-9a-zA-Z-\/_]+)/) {
    print OUTPUT "$1 $2 $3 $4 $5 $6
";
  }
}

2.6 Benefits

Because DNS resolution is an important part of the operating system, the information we need can be transmitted even if ordinary web traffic is blocked. We do not require the overhead of a web server, or subject ourselves to the various vulnerabilities that it can entail. The logging scripts are fairly simple, and can be run on both Windows and UNIX operating systems. The operating system resolvers and DNS servers also have built-in timeout and retry mechanism which makes the delivery of the queries very reliable.

2.7 Wiring the Applications

Ensight, ParaView, and VisIt are client-server applications supported by the DAAC for use by HPCMP customers for interactive data analysis. We had an understanding that these packages were being regularly used by the computational researchers but really didn’t have an
understanding of how often the packages were used, or the resources used to perform the data analysis. Each of these packages starts up very differently, and exactly how to implement the software logging was not a trivial task. It was the goal of the DAAC to capture application startup parameters from the client workstation, and the HPC resources from server side of the connection.

For the Ensight client, we modified the existing EnSight startup scripts on Linux (ensight100.launcher, ensight100.client) by adding the command line:

```bash
logging-script ensight100_client LINUX 10.0.1a
```

Similarly, for the Mac OSX implementation of Ensight, we modified the same scripts with this command line:

```bash
logging-script ensight100_client MAC 10.0.1a
```

ParaView does not start up with a shell script and we did not want to modify the source code, and in this case, we were able to create a Unix alias for the paraview command via the paraview module. The alias that we built looked like this:

```bash
alias paraview /usr/cta/paraview/utils/logging-script paraview LINUX 3.14.1;
/usr/cta/paraview/3.14.1/bin/paraview
```

VisIt has a significantly more complicated startup procedure and it was not immediately obvious how or where to implement the logging capabilities in this package. DAAC staff determined that a startup script called “frontendlauncher” needed to be modified to appropriately capture utilization. The code added to the frontendlauncher looked like this:

```bash
# ARL CHANGES
if ($visitargs[0] eq "-viewer") {
    @loggercmd = ("/usr/cta/visit/utils/logging-script" , "Visit" , "LINUX" , $ver );
    print join(" ", @loggercmd) . "\n";
    system @loggercmd;
}
# END ARL CHANGES
@visitcmd = ("${visitdir}/bin/internallauncher", @visitargs);
exec @visitcmd or die "Can't execute visit launcher script: $!\n"
```

On the server side, the logging script is implemented in the Ensight startup script that submits the portable batch system (PBS) job. At that time, we have the appropriate information to pass to the logger and we know that we can perform an nslookup command from the login node where the script resides:

```bash
if (-e $CEI_HOME/local/logging-script) then
    set cei_version=$passed_path:t
    $CEI_HOME/local/logging-script ensight_launcher_${cei_version} ${num_nodes} ${num_procs}
endif
```
ParaView implements the logging script using a very similar methodology within the server-side startup script:

```bash
if (-e ${PV_HOME}/utils/logging-script) then
  ${PV_HOME}/utils/logging-script US_paraview_${VERSION} ${NODES} ${PROCS}
endif
```

Adding the logging-script functionality to the server side of VisIt was particularly tricky given the complex nature of the job launching functionality. Job launching on the server side is managed through a 3500-line Perl script that is quite convoluted. The code added to the internal launcher is shown here:

```perl
@loggercmd = ("/usr/cta/visit/utils/logging-script" , "visit_server_${ver}", ${nodes} , ${procs});
system @loggercmd;
```

### 2.8 Results

This logging methodology was implemented during the month of May 2012. After that first month of logging software usage based on this new methodology, we were able to quickly characterize utilization at a level of detail that was previously not available. This detail allows the DAAC to begin to analyze how the software is being utilized, the client platforms being used to initialize the software, and how the new HPCMP utility servers are being incorporated into the daily work flow of the computational researchers.

Total number of log entries: 1221.

Total number of unique hosts reporting results (client workstations and HPC systems): 65.

Total number of (ParaView/EnSight/VisIt) HPC job launchers started: 241.

- Ensight – 149
- ParaView – 71
- VisIt – 21

Total number of jobs started on the various utility servers: 223 (some of these are where the app clients were started on a U.S. login node).

- ARL – 152
- ERDC – 14
- AFRL – 31
- NAVO – 22
- MAUI – 4
ARL client workstations:

- Hits from Linux Workstations – 536
- Hits from Mac Workstations – 334
- Hits from Windows Workstations – 0 (no apps wired on Windows)
- Ensight client startups – 205
- Paraview client startups – 226
- Visit client startups – 260 (260 startups, but only 21 hpc launches)
- Cubit startups – 131

3. Conclusion

The initial implementation of a logging utility based on DNS queries to capture utilization of specific client-server applications has been successful. Logging of carefully crafted DNS queries has proven to be a workable substitute for a web-based solution when the data to be collected consists of a few simple character strings. We have been able to generate the log entries that we feel are required to adequately understand the nature of how these applications are being utilized. We can now easily count how many times particular software package was used, who used that package, and how that application used resources on the HPC systems.

One piece of information that became immediately apparent as we started browsing the logging results was how customers were using resources on the HPCMP utility servers. These resources are made available to HPC customers specifically for pre- and post-processing of results generated on the larger computational systems, and users are limited to four nodes and 16 processors per node, for a total of 64 processors. By looking at the logging results, we were able to determine that customers seemed to always ask for the maximum amount of resources (four nodes and 16 processors/node) presumably without regard to the amount of data to be analyzed; however, in this case, more is not necessarily better. A wide variety of factors (number of files, number of timesteps, size of computation grid, etc) impacts the performance of these client-server applications as they are distributed across multiple processors, and it is possible to request too many processors and actually decrease interactive performance. That is, the application will spend more time trying to distribute the data and pass information amongst the allocated processors rather than actually performing data analysis. Once we became aware of this tendency, we were able to use this as an educational opportunity for our customers and provide them with additional information on how to make best use of the computation resources available for their use. While this was not an anticipated result, this example demonstrates the value of having a centralized logging facility that allows the DAAC to monitor and understand how the applications are being utilized.
<table>
<thead>
<tr>
<th>NO. OF COPIES</th>
<th>ORGANIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEFENSE TECHNICAL INFORMATION CTR (PDF only) DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FORT BELVOIR VA 22060-6218</td>
</tr>
<tr>
<td>1</td>
<td>DIRECTOR US ARMY RESEARCH LAB IMAL HRA 2800 POWDER MILL RD ADELPHI MD 20783-1197</td>
</tr>
<tr>
<td>1</td>
<td>DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL 2800 POWDER MILL RD ADELPHI MD 20783-1197</td>
</tr>
<tr>
<td>NO. OF COPIES</td>
<td>ORGANIZATION</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
</tr>
</tbody>
</table>
| 1            | LOCKHEED-MARTIN INFO SYS AND GLOBAL SER  
J P MARTIN  
M J MOTSKO  
939-I BEARDS HILL RD PMB 203  
ABERDEEN MD 21001 |
| 7            | DIR USARL  
RDRL CIH C  
K T KIRK  
RDRL CIH S  
B M RAPP (2 CPS)  
H M KASH (2 CPS)  
R C ANGELINI (2 CPS) |

ABERDEEN PROVING GROUND