PROCEEDINGS OF SCIENCE

Using the ELK Stack for CASTOR Application Logging at RAL

Rob Appleyard[1]
Science and Technology Facilities Council, Scientific Computing Department
STFC Rutherford Appleton Laboratory, Harwell Oxford, OX110QX, UK
E-mail: Rob.Appleyard@stfc.ac.uk

James Adams
Science and Technology Facilities Council, Scientific Computing Department
STFC Rutherford Appleton Laboratory, Harwell Oxford, OX110QX, UK
E-mail: James.Adams@stfc.ac.uk

Logging of application and system information is crucial to running any service. Whether it is looking for trends in use, investigating specific problems or analysing security incidents, logs are an essential source of information for any service manager or system administrator. However, distributed applications produce large volumes of logging information from a number of disparate, but related, daemons running on an increasing number of machines. In this scenario, log analysis with traditional command line tools such as *awk* and *grep* is hugely inefficient. In addition, presenting the results of analysis using tools like spreadsheets is a time consuming process for a busy administrator. At STFC, the Scientific Computing Department has adopted the ELK stack (www.elastic.co) for collating of logs from the mass storage system. The ELK stack is built from three open source components – Elasticsearch, Logstash, and Kibana. It is currently gathering information from about 300 machines at the Rutherford Appleton Laboratory covering both the WLCG Tier 1 and local facilities storage systems. In this presentation, we discuss the evolution of Elasticsearch within STFC and present specific use cases where it has clear advantages over traditional methods. We also compare it with the previous logging system developed at CERN (DLF) and show that the ELK stack is a more generically useful toolset. Finally, we demonstrate its usefulness for looking at both trends and specific event analysis.

International Symposium on Grids and Clouds (ISGC) 2015
15 -20 March 2015,
Academia Sinica, Taipei, Taiwan

[1]Speaker

PoS(ISGC2015)027

1. Introduction

CASTOR[1] is a distributed hierarchical storage management system developed at CERN for managing physics data at petabyte-scale. It is used by the Scientific Computing Department (SCD) at the STFC Rutherford Appleton Laboratory (RAL) to manage disk and tape data for the UK's WLCG[2] Tier 1 data centre ('the Tier 1') as part of the GridPP[3] project. CASTOR is a database-centric system – a central Oracle database is used to manage all transactions, and every one of these interactions is logged.

RAL's CASTOR instance runs at large-scale – current storage capacities 11PB of disk and 13PB of tape. The disk storage capacity is currently distributed across 161 storage nodes, and 25 additional nodes are used for management and access to CASTOR instances (RAL has 4 CASTOR instances serving WLCG users – one each for the three largest LHC experiments, and one split between ALICE and other users). Other services run as part of RAL's Tier 1, most significantly a 550-node Condor batch farm. The Scientific Computing Department's RAL data centre also hosts many non-Tier 1 services, such as the 'Emerald' GPGPU cluster and the 'JASMIN/CEMS' climate and earth system science infrastructure.

As will be discussed in the following sections, we consider the ability to collect and conveniently search through the logs produced by CASTOR to be a requirement for running the system at this scale. To this end, we have adopted the ELK (Elasticsearch, Logstash and Kibana)[4] software stack to collect CASTOR's logs and enable administrators to search through the logs and identify points of interest. In the following sections, we will discuss the evolution of CASTOR log analysis at RAL, from the starting point several years ago to the present day situation.

2. CASTOR's logging

CASTOR's logging is very verbose. As previously noted, every database transaction is logged, and the mean size of a log message is 460 bytes. This logging is therefore coming from the application itself, rather than the OS daemons. CASTOR itself has a variety of different daemons that perform different tasks in the system. The messages from these daemons come through at a high rate - the RAL CASTOR instances currently produce around 70,000,000 log events per day, from over 300 source nodes. This totals up to around 32GB/day of logging data.

Below, in Figure 1, is a graph from our Ganglia instance illustrating the rate of local log generation on RAL's most verbose node, which is one of the management nodes dedicated to the ATLAS experiment. Typically, this node generates around 2GB of logs per day, but during the highest-volume 24h period, over 10GB of logging information was generated by the system. There is some contribution here from system logging, but the scale of this is insignificant (a few MBs) - the dominant part of the data is made up of CASTOR application logs.

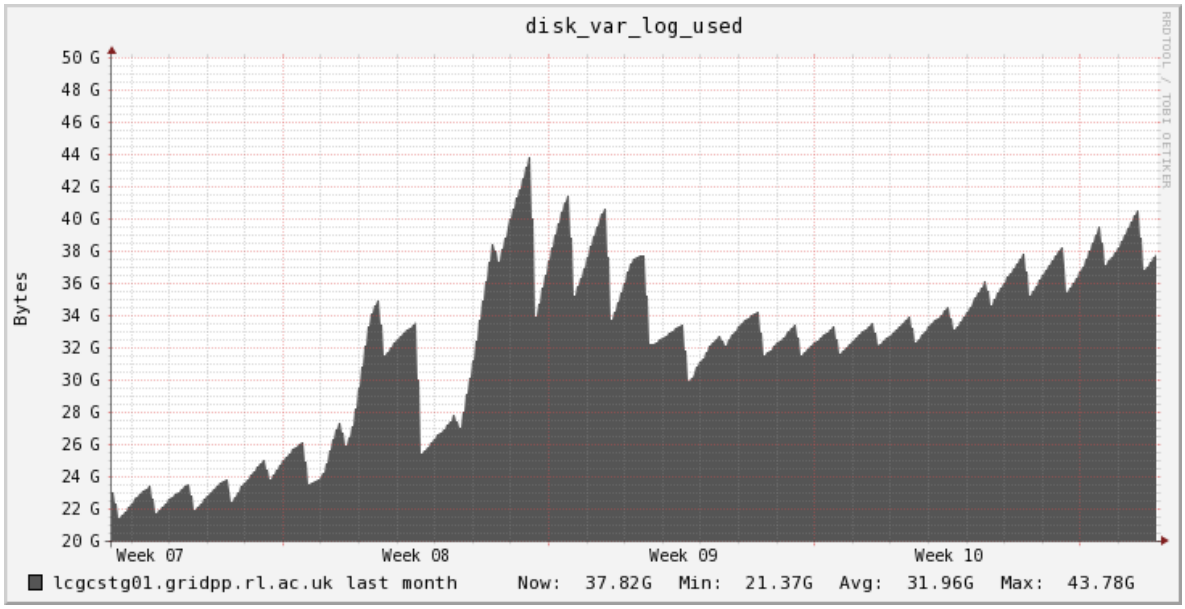

Fig 1: Graph showing the data stored on the /var/log partition on one of the management nodes for RAL's ATLAS CASTOR instance. This is the most verbose node in the system, and the daily rate is roughly 2GB/day. Note that the 'peaking' behaviour is due to the daily logrotation that occurs at 1315.

The format of CASTOR's logging is regular. There is a preamble consisting of metadata – a timestamp, the source node, and the source daemon. This is followed by a series of key-value pairs, containing the actual message. The message can contain a variety of data, but a typical message might have the severity of the message, some message text, and the IDs of transactions and files that the message concerns. This format is common to all the various CASTOR daemons, and the various ID values of entities mentioned by the log messages are shared between different daemons (CASTOR transactions are stored in the central Oracle database). Usually, each field is referred to by daemons by a name common to all of them, but this is not completely consistent – there is some variation, often in parts of the system that are under less active development. A sample of CASTOR log messages from RAL's preproduction instance is shown below in figure 2.

The shared ID values are a useful property for administrators. If one wishes to search for everything relating to a file with a given ID, the same ID will appear, referred to in the same way, in all the various logfiles. Thus the question can be asked – why, given the sharing of IDs, is it necessary to have a specialised search system? CASTOR runs on nodes that use a Red Hat-derived Linux operating system, and so there are a wide variety of powerful command line text search tools available, such as *grep*, *awk* and *sed*. These utilities can easily parse even very large text files in reasonable timescales, and allow broad freedom to process data as needed.

```
2015-02-27T10:35:11.321427+00:00 ccse08 stagerd[15180]: LVL=Info TID=15207 MSG="Request
processed" REQID=9f1786a7-7abd-4e25-85c2-e16020721ae7 NSHOSTNAME=cprens.ads.rl.ac.uk
NSFILEID=35103020 SUBREQID=0eb01a56-f662-7fa8-e053-87b7f682af1f Type="StagePutRequest"
Filename="/castor/preprod.ral/preprodDisk/rob/stress/file-10M-preprod-4-1-10M.22288"
Username="rvv47345" Groupname="esc" SvcClass="preprodDisk" ProcessingTime=0.035222
2015-02-27T10:35:11.328425+00:00 ccse08 stagerd[15180]: LVL=Info TID=15199 MSG="Request
processed" REQID=4ac7af54-1bd4-4235-8670-b3c392c8ee5a NSHOSTNAME=cprens.ads.rl.ac.uk
NSFILEID=35103021 SUBREQID=0eb5b121-7fba-7ffd-e053-87b7f682c682 Type="StagePutRequest"
Filename="/castor/preprod.ral/preprodDisk/rob/stress/file-10M-preprod-2-1-10M.24497"
Username="rvv47345" Groupname="esc" SvcClass="preprodDisk" ProcessingTime=0.033232
2015-02-27T10:35:11.329387+00:00 ccse08 stagerd[15180]: LVL=Info TID=15236 MSG="Invoking
prepareForMigration" REQID=51b2697b-3d20-481d-9a8f-9ec5f3de10d5
NSHOSTNAME=cprens.ads.rl.ac.uk NSFILEID=35102867 ChkSumType="AD" ChkSumValue="9600001"
SubReqId=1049923615 FileSize=10485760
2015-02-27T10:35:11.333296+00:00 ccse08 stagerd[15180]: LVL=Info TID=15219 MSG="Request
processed" REQID=fc0099bf-6437-4beb-8566-be18ff363536 NSHOSTNAME=cprens.ads.rl.ac.uk
NSFILEID=35102845 SUBREQID=0ee353a9-6a4b-0160-e053-87b7f6822b2c
Type="StagePrepareToGetRequest"
Filename="/castor/preprod.ral/preprodDisk/rob/stress/file-10M-preprod-4-2-10M.11851"
Username="rvv47345" Groupname="esc" SvcClass="preprodDisk" ProcessingTime=0.031011
2015-02-27T10:35:11.334272+00:00 ccse08 stagerd[15180]: LVL=Info TID=15221 MSG="Request
processed" REQID=11144718-9d98-4c5d-abeb-a7ab3a2fc903 NSHOSTNAME=cprens.ads.rl.ac.uk
NSFILEID=35102822 SUBREQID=0ee353a5-f73f-0162-e053-87b7f682d2a2
Type="StagePrepareToGetRequest"
Filename="/castor/preprod.ral/preprodDisk/rob/stress/file-10M-preprod-5-1-10M.9095"
Username="rvv47345" Groupname="esc" SvcClass="preprodDisk" ProcessingTime=0.029075
2015-02-27T10:35:11.343329+00:00 ccse08 stagerd[15180]: LVL=Info TID=15218 MSG="Request
processed" REQID=08336579-c2a6-4b55-b8af-75e5cb39d244 NSHOSTNAME=cprens.ads.rl.ac.uk
NSFILEID=35102813 SUBREQID=0ee353a5-fd6e-014c-e053-87b7f6822851
Type="StagePrepareToGetRequest"
Filename="/castor/preprod.ral/preprodDisk/rob/stress/file-10M-preprod-3-2-10M.29584"
Username="rvv47345" Groupname="esc" SvcClass="preprodDisk" ProcessingTime=0.034123
2015-02-27T10:35:11.353199+00:00 ccse08 stagerd[15180]: LVL=Info TID=15207 MSG="Request
processed" REQID=17a9c0c7-df94-4242-8a65-7fb1830c628a NSHOSTNAME=cprens.ads.rl.ac.uk
NSFILEID=35103022 SUBREQID=0eb01a56-f663-7fa8-e053-87b7f682af1f Type="StagePutRequest"
Filename="/castor/preprod.ral/preprodDisk/rob/stress/file-10M-preprod-2-1-10M.25135"
Username="rvv47345" Groupname="esc" SvcClass="preprodDisk" ProcessingTime=0.028000
2015-02-27T10:35:11.362366+00:00 ccse08 stagerd[15180]: LVL=Info TID=15199 MSG="Request
processed" REQID=2389d471-da5e-45fa-9e38-bdb2d5697b1b NSHOSTNAME=cprens.ads.rl.ac.uk
NSFILEID=35103023 SUBREQID=0eb5b121-7fbb-7ffd-e053-87b7f682c682 Type="StagePutRequest"
Filename="/castor/preprod.ral/preprodDisk/rob/stress/file-10M-preprod-4-1-10M.30892"
Username="rvv47345" Groupname="esc" SvcClass="preprodDisk" ProcessingTime=0.030504
2015-02-27T10:35:11.365454+00:00 ccse08 stagerd[15180]: LVL=Info TID=15235 MSG="Invoking
getUpdateDone" REQID=e81cc3ad-9a3b-4575-b78b-f813f4ac4da5 NSHOSTNAME=cprens.ads.rl.ac.uk
NSFILEID=35102813 SubReqId=1049923603
```

Fig 2: Sample set of log messages produced by the 'stagerd' daemon on RAL's preproduction CASTOR instance while stress tests were being run. The time between the first and last message is 0.044s

The problem with using UNIX command line tools is that, as previously noted, CASTOR is not running only on one system. It is a distributed system, with over 300 nodes that perform varying functions. Using *grep* and *awk* to search through archived log messages on a single node is not an unreasonable task. Doing so when one has hundreds of potential locations in which key information may be located becomes inconvenient very quickly for administrators.

We therefore need some system for collecting CASTOR's log messages into a single location, to make them more convenient to analyse. This system should provide the user with

some means of querying the dataset thus created, so administrators can search for key pieces of information easily. Also, given that the combined set of logs comprehensively describes the activity of the system, making this dataset searchable also opens the possibility of using it to visualise trends in CASTOR's behaviour. Exploiting this potential would be highly desirable.

3. DLF

When RAL installed CASTOR, a log storage and analysis system was already extant. It was a system called the Distributed Logging Facility (DLF)[5]. It was developed at CERN specifically for use with CASTOR, and was based on an Oracle database. Logging information was sent using rsyslog to one of three collector nodes, which used a daemon called *logprocessord* to collect all the log information and send it to the DB.

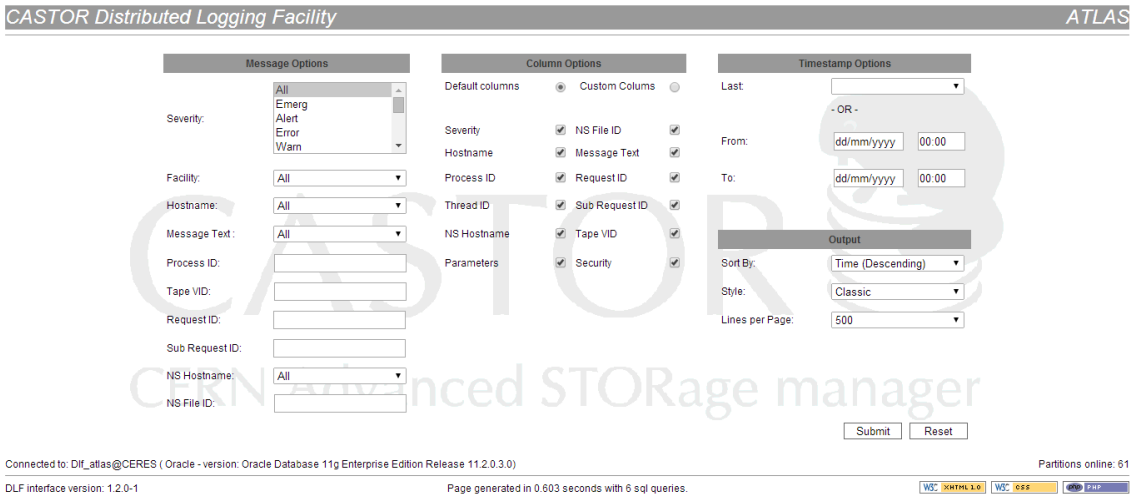

Fig 3: The query input page for RAL's DLF instance.

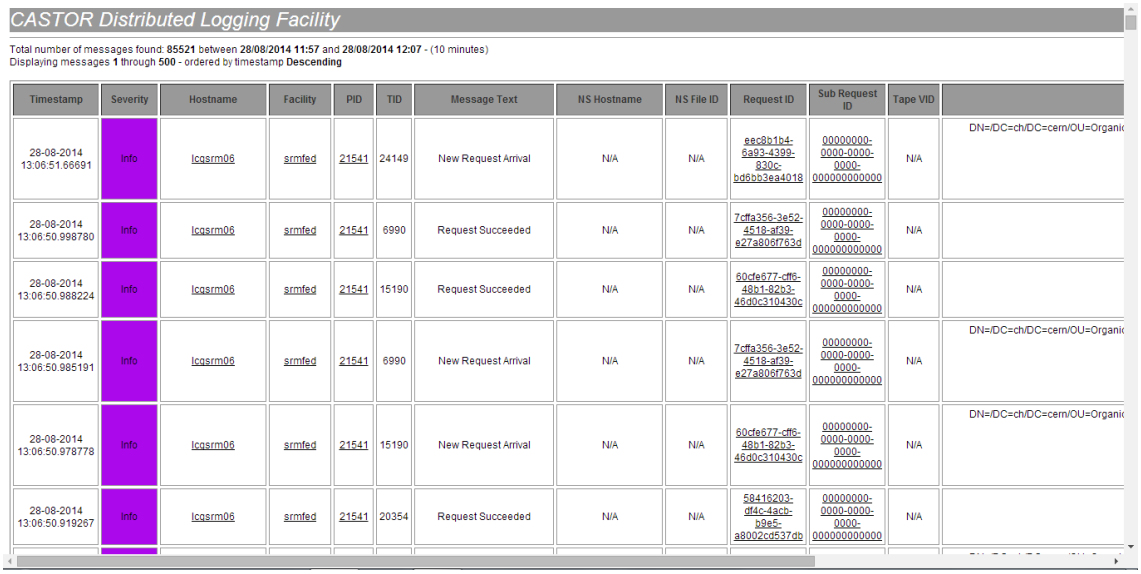

Fig 4: A sample results page for RAL's DLF instance.

DLF's key strength was the quality of its query definition. Because it was a bespoke solution developed specifically for CASTOR, the developers could anticipate the fields that would be of most interest to the user and make them conveniently available for searching. Users could also trigger new queries by clicking on certain properties in the results, such as the NSFILEID (the ID assigned to a file by the CASTOR Name Server, used by various different daemons).

However, as the CASTOR system came under heavy load during LHC run one, some shortcomings of DLF became apparent. The search system proved not to scale well to the levels required by the verbosity of CASTOR logging output. Even with separate indices for each CASTOR instance, some queries were taking over 1 hour to return any data at all. These performance issues were found to be due scaling issues in the software, rather than the hardware. Unsurprisingly, the most computationally difficult queries, such as those spanning large time periods, were the most problematic, especially since DLF would return no results until it the entire query had finished running.

This level of performance is unacceptable for production usage. Performance issues with DLF were not unique to RAL – CERN were using the same system, and were seeing many of the same problems. Developers at CERN noted the problem and designed their own system to replace DLF.

4. The CERN Hadoop Solution

As noted above, DLF's performance limitations were not unknown at CERN. CERN's approach to solving the problem was to build a bespoke software stack based on Apache Hadoop and related products.

Messages produced by CASTOR daemons were passed through a bespoke producer script (written in Python) named 'simple-log-producer', which tokenised the log messages and forward them on to a message broker running ActiveMQ or Apollo. The data from the broker was then sent to a variety of consumers (also bespoke Python). The final destinations supplied by these consumers included:

- Hadoop – The raw log events are stored as text files in HDFS for potential data mining.
- HBase – Using similar filters to DLF, messages relating to transfer events are stored in HBase tables and exposed via a web dashboard.
- Metrics Analysis Engine – Messages relating to the performance of CASTOR as well as counts of events are stored in a MySQL database to provide graphs on a web dashboard.

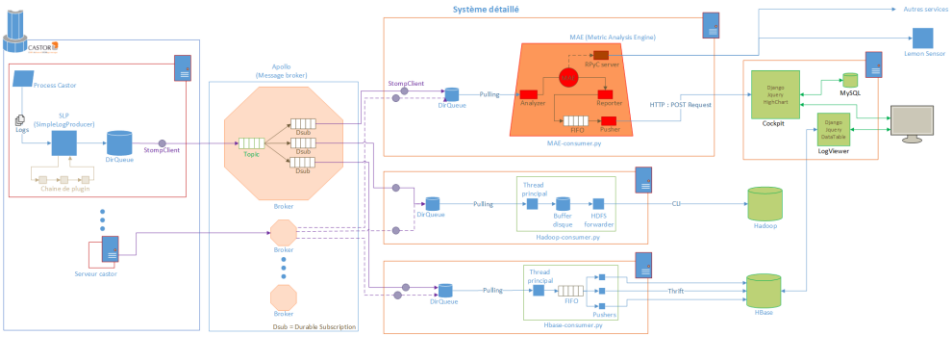

Fig 5: Architecture of CERN's monitoring infrastructure[6]

The initial approach at RAL was to simply reproduce the CERN infrastructure, on the grounds that it was known to work and seemed like a definite upgrade to the DLF. However, this strategy proved to be much more complex to implement that originally expected – the system was designed to be built on top of CERN's Agile Infrastructure Platform[7], which STFC has no analogue to.

A considerable amount of time was expended trying to assemble a working version of the system at RAL, and various different adaptations were attempted. The Apollo broker proved difficult to get started, and time was also spent on an attempt to eliminate the broker and replace it with a single node that used rsyslog to forward messages onto the various consumers, and then reproducing the code that performs the tokenisation functionality of simple-log-producer within the producers. This system was nicknamed 'Smooshed-log-producer'.

This was finally abandoned following a review meeting which determined that the CERN solution was too top-heavy to fit STFC's requirements. This meeting noted that the problem being worked on was far from unique in the computing world. Conveniently and safely storing logging information from a distributed service and making it searchable for later reference is something that anyone running such a service is likely to want to be able to do, and so it would be surprising if there were not a suitable off-the-shelf solution available.

Based on this reasoning, a brief search was conducted for suitable alternatives, and we chose to try a one-day experiment of installing an Elasticsearch-based cluster, fed by Logstash, to see if our goals could be accomplished without the quantity of work that seemed to be required to make the CERN solution operational. This experiment was successful – our prototype using the ELK stack was deployed within an afternoon.

5. The ELK Stack

The ELK stack is made up of three pieces of software:

- Elasticsearch – a document oriented database and search engine (based on Apache Lucene[8]).
- Logstash – a message processing pipeline (input → filter → output).
- Kibana – a web-based interactive data analysis portal.

These are deployed as a chain - we send log events from hosts over UDP to Logstash, these are processed and tokenised, then forwarded to Elasticsearch. Kibana is then used to provide web-based dashboards based on the information in Elasticsearch.

6. Implementation of ELK at RAL

6.1 Hardware

Our initial Elasticsearch cluster was composed of ten retired worker nodes, originally acquired as part of the Tier 1's 2008-2009 purchasing round, each with the following specification:

- Two quad-core CPUs (no hyper-threading)
- 16GB RAM
- Single 500GB SATA disk

These have recently been replaced four new dedicated nodes, each with the following specification:

- Two six-core CPUs (24 threads with hyper-threading)
- 128GB RAM
- 4TB RAID5 array of SAS disks

Our first cluster allowed us to store up to forty-eight days of logs from all CASTOR instances, however due to the lack of RAM in the nodes, searching any further back than three days was very slow. The replacement cluster has been designed to allow for fast long-term analysis.

Should, in the future, this hardware prove inadequate (perhaps because of more widespread usage of Elasticsearch at RAL), then from an extension perspective, Elasticsearch is appropriately named, as its architecture makes it very easy to expand. It is largely a case of installing the software on new hardware and pointing it to the cluster, although this scaling cannot continue indefinitely - architectural work is very necessary behind the scenes when scaling to high levels.

The precise configuration of our Elasticsearch cluster has required some careful thought, as the scale at which our cluster operates at a level beyond the published HOWTOs at the time of writing. RAL is certainly not alone in operating the ELK stack at this scale, but as far as the writers are aware, none of the other users appear to have made public details of their configuration. The authors are happy to discuss their experiences if asked.

6.2 Software Adaptations

Messages are sent from the source nodes to the Logstash forwarder (a virtual machine hosted on our production infrastructure) over UDP, and the forwarder node is then responsible for onward transport to the Elasticsearch cluster. The choice of UDP transport was made for reasons of convenience – the logging configuration on the source nodes is only slightly adapted from the Tier 1's central logging system, which uses rsyslog, and UDP and TCP are the simplest protocols to send from rsyslog. TCP sending is not used to avoid making the Logstash forwarder a potential single point of failure for the Tier 1 – the TCP protocol guarantees delivery, so if it were in use and the destination node failed, unsent messages would pile up at the sources, with potentially problematic consequences.

On the other hand, the UDP protocol does not guarantee message delivery, so there is an obvious risk of losing messages between the source nodes and the Logstash forwarder. The message loss rate has not yet been quantified, but we have not observed large holes in our log records when compared to the log files stored on local nodes. A future development goal for the system is to evaluate alternatives to UDP for message forwarding.

In general, the log messages provided by CASTOR are easy for Logstash to process, however there are several small issues we discovered after we started trying to use the logs. The first problem we encountered was simply inconsistent capitalisation of field names in the key-value part of the messages – unfortunately, different daemons use different conventions for the same field. As Elasticsearch is case sensitive to field names it treats different capitalisations as different fields, making tracing of events much more difficult.

To rectify this we developed a substitution ruleset for logstash to correct for these (and other) inconsistencies in field naming, a small sample is shown below.

```
rename => [ "castor_FileName", "castor_Filename" ]

rename => [ "castor_FileSize", "castor_Filesize" ]

rename => [ "castor_fileSize", "castor_Filesize" ]

rename => [ "castor_checksumValue", "castor_ChecksumValue" ]

rename => [ "castor_ReqId", "castor_REQID" ]

rename => [ "castor_SubReqId", "castor_SUBREQID" ]

rename => [ "castor_CastorReqId", "castor_REQID" ]

rename => [ "castor_CastorRequestId", "castor_REQID" ]

rename => [ "castor_svcClassId", "castor_SvcClass" ]

rename => [ "castor_svcClassId", "castor_SvcClass" ]

rename => [ "castor_Pivilege", "castor_Privilege" ]
```

We also developed a set of rules to ensure that fields that should contain numerical values are stored as such, this then allows the values of these fields to be analysed by Kibana.

Another configuration choice made to support convenient search was the choice of Lucene index length – Lucene sorts data into indices and for logging data accumulated over time the obvious choice is to have one index for each fixed time period. To match our log-rotation schedule, we use the Logstash default of one index per day. Results are returned by Elasticsearch queries index-by-index, so multi-day queries will return the most recent day's results, followed by the preceding days in sequence. This is a useful feature – unlike DLF, the user sees the results of their query being constructed as Elasticsearch progresses through the search. They can also see that the search engine has not simply hung, which is a problem that DLF's long search times frequently caused users to suspect.

6.3 Software Development

Despite the strong integration between Elasticsearch and Kibana, the latter is not a requirement for querying the search cluster. Elasticsearch exposes an interface through which Lucene queries can be sent directly to the search cluster. This functionality was used to develop an experimental command-line tool for running simple queries, as an alternative to Kibana. This tool sends queries directly into the Elasticsearch cluster using Lucene query syntax.

We also commissioned a project to build a tool on top of Elasticsearch to provide DLF-like CASTOR integration. This was done by a work experience student. The project produced a working prototype, but after evaluation it was decided that developing and supporting this tool further was not a good time investment when compared to educating users in how to get the information they were looking for from Kibana.[2]

7. User experience in Kibana

RAL users interact with our implemenation of the ELK stack through the Kibana web interface. This is hosted on a local private web server, accessible only within RAL. We have configured several custom dashboards for different CASTOR use cases. Examples of these include:

- A generic CASTOR dashboard showing the volume of messages from CASTOR from the last 12 hours and and a sample of the most recent.
- A heatmap showing the number of client connections to the CASTOR SRM nodes[3] from each country
- A dashboard that shows the top 10 most 'talkative' nodes in CASTOR over time and the proportion of messages from each one.
- A display that shows the average wait time for requests in the CASTOR stager, sorted by VO.

[2] This is available at https://github.com/stfc/castor-logging
[3] The SRM nodes are responsible for Storage Resource Manager[9] (SRM) protocol access into CASTOR.

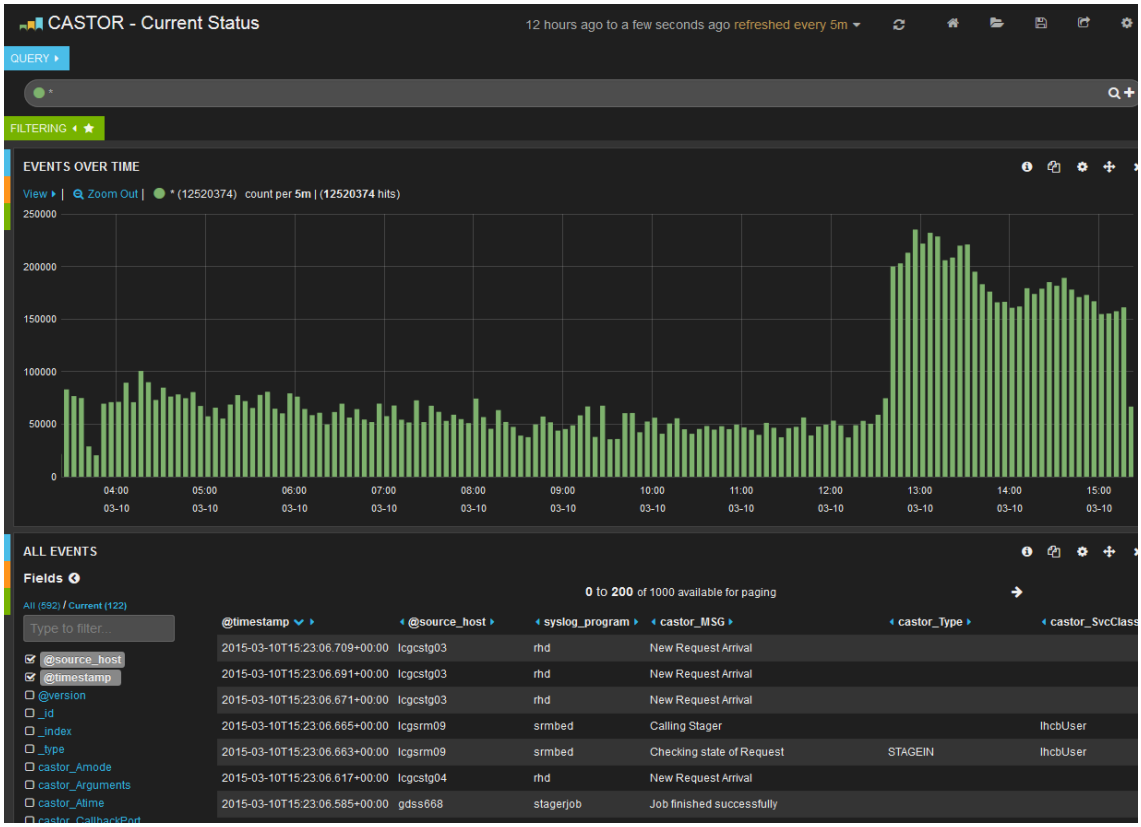

Fig 6: The default dashboard for Kibana at RAL as of 2015-03-10. The graph shows the raw quantity of log messages received by Elasticsearch over time.

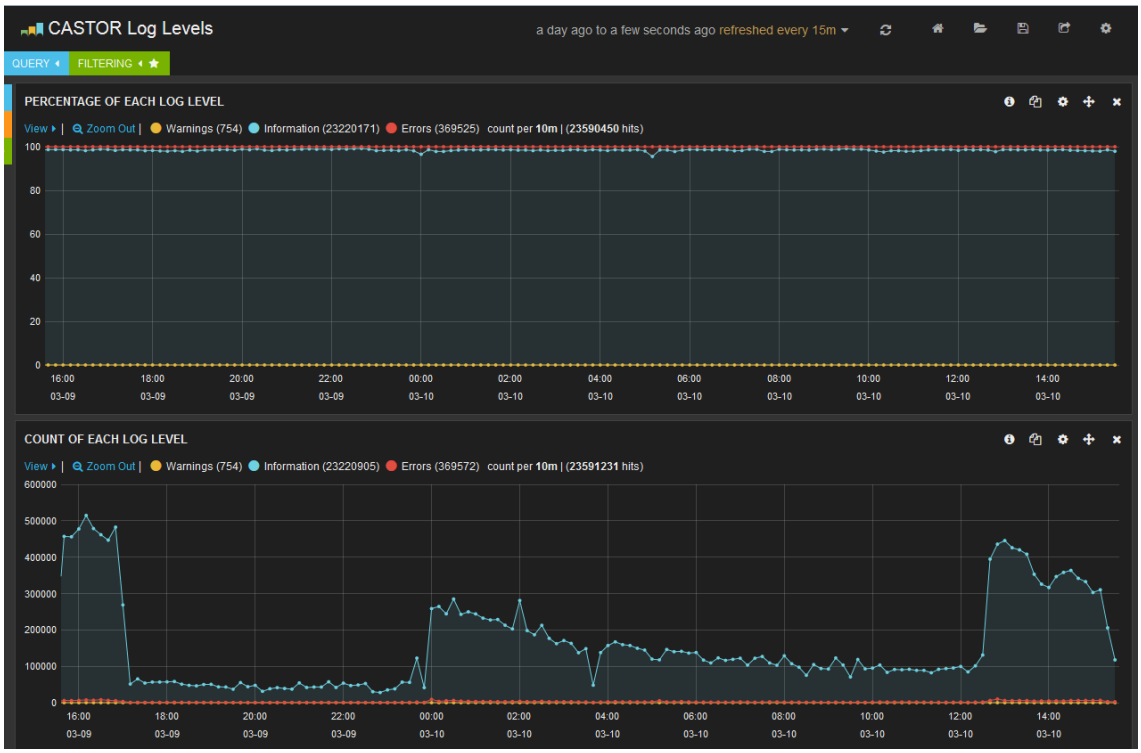

Fig 7: A Kibana dashboard that shows the proportion of log messages of each severity level that are received by Elasticsearch over time.

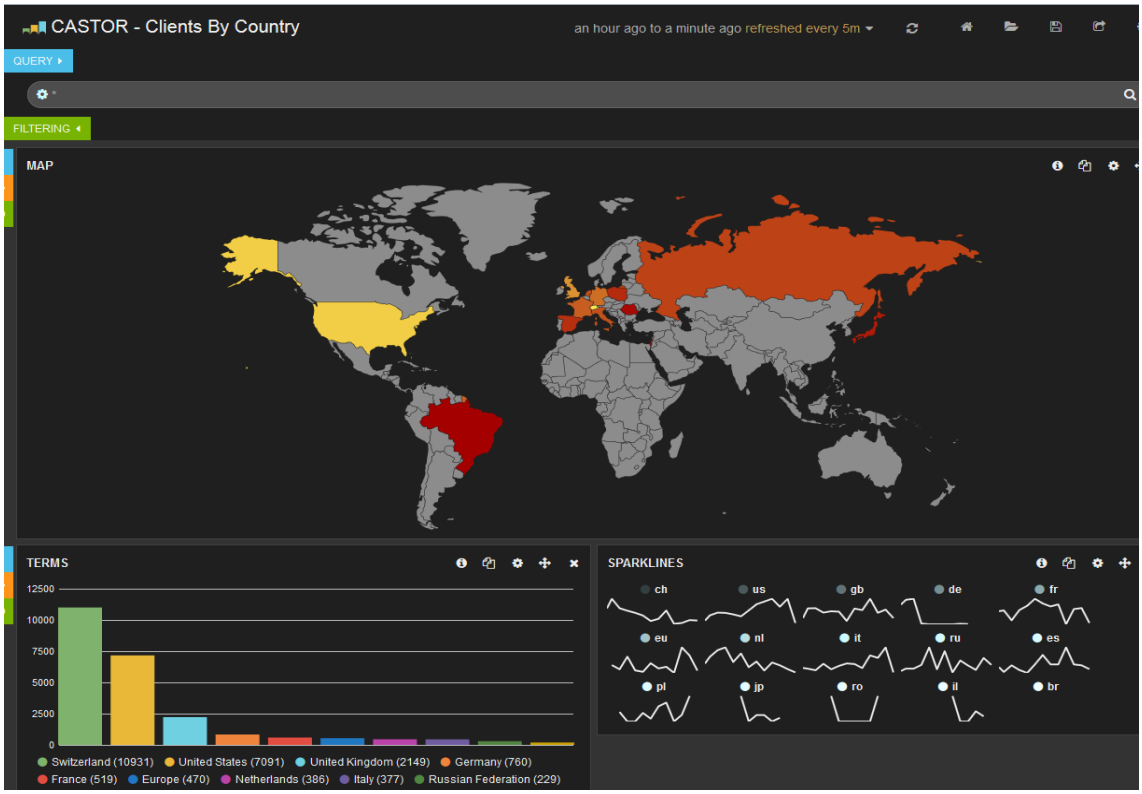

Fig 8: A Kibana dashboard that shows the geographical origin of requests into the CASTOR SRM for the last hour. Note that RAL's local batch farm is excluded, as it would swamp all other sources.

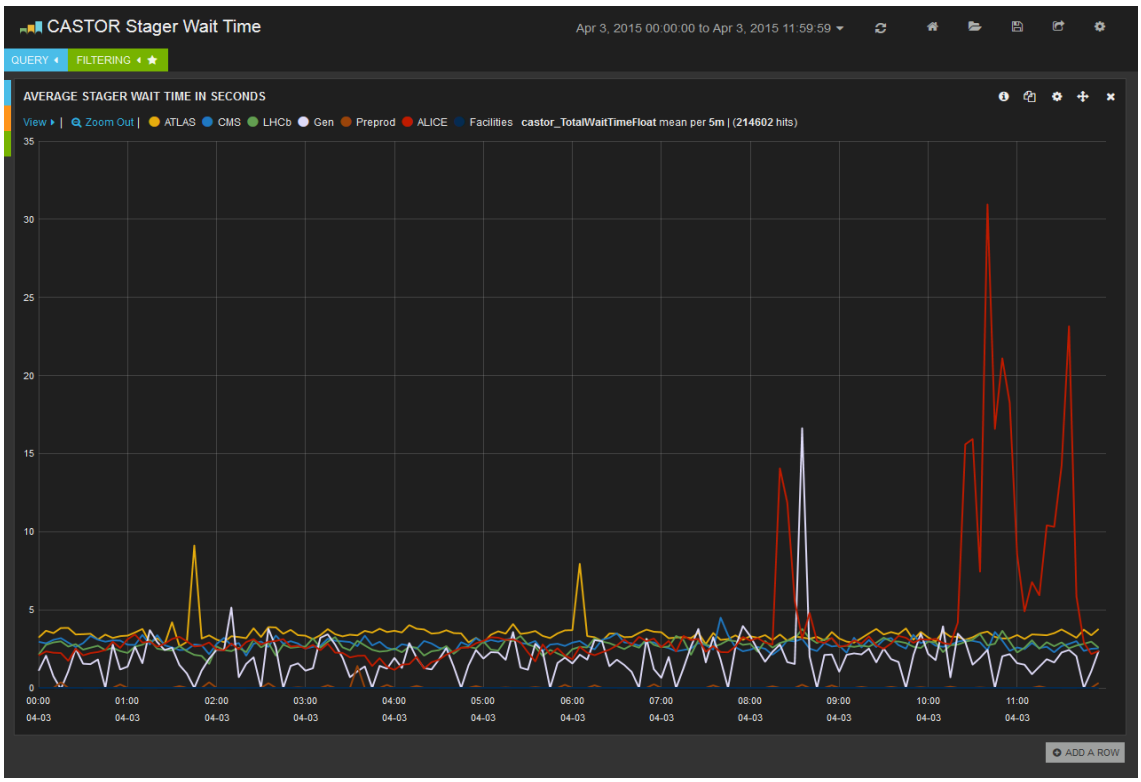

Fig 9: A Kibana dashboard that shows the average wait time for requests into CASTOR's stagers, sorted by VO.

These dashboards are saved within Elasticsearch and are available to all users. Users can also define and save their own dashboards as needed.

While we are satisfied with the quality of Kibana, the user experience is not flawless. Queries are entered into Kibana's search box using Lucene query syntax, and so to make full use of Kibana, some knowledge of this is required. It has some idiosyncrasies, indeed the most common problem encountered by RAL users was using the wrong type of quotation marks to denote a search string that contains a space – Lucene will only accept double-quotes, and single quotes are ignored.

8. Evaluation

We believe that the project to improve RAL's CASTOR log search capabilities has been a success. Elasticsearch improves on the naive approach of using command line tools to process individual log tools through both convenience and speed, and it improves over DLF in that the query speed is quick enough to no longer a concern for users, and the range of possible queries is increased.

In addition, the Kibana web interface provides a new capability for the Tier 1. None of the previous log analysis tools provide visualisation analysis capabilities, and this has already proved useful for tracking load on the system.

9. The Future

The ELK stack is now the standard way of searching CASTOR's logs at RAL. However, the four new dedicated nodes mentioned above were only installed at the end of March, and so we are still exploring their full capabilities. While the dominant user of SCD's ELK stack is expected to continue to be CASTOR, other Tier 1 or SCD services could easily send their logging to the system. In a similar vein, it could also be used to store and analyse system logging data. These would be stored in a separate set of indices to the CASTOR logging data.

Another possibility is to support Elasticsearch as a service usable by local RAL users for data analysis. The Tier 1 came to the ELK stack because it fitted our requirements for storing CASTOR's logs. However, there are plenty of other use cases for a powerful search engine. Users with large datasets could feed it into a hived-off part of the Elasticsearch cluster and use it for search and analysis purposes. SCD is currently at an early stage of looking into these possibilities.

10. Conclusion

This paper gives an account of the evolution of application log storage and analysis for the CASTOR system at the RAL Tier 1 site. It discusses the requirements imposed by the CASTOR system, and discusses past solutions, including the 'DLF' system in use previously. It gives an account of the implementation of the ELK stack at RAL and shows how the ELK stack can be

used to provide customised dashboards for easy visualisation. Finally, it discusses future plans for usage of the ELK stack at RAL.

References

[1] J.P. Baud, et al. *CASTOR status and evolution*, arXiv preprint cs/0305047

[2] http://wlcg.web.cern.ch

[3] D. Britton, et al. *GridPP: the UK grid for particle physics*. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 367.1897 (2009): 2447-2457

[4] https://www.elastic.co

[5] T. Rekatsinas, et al. *CASTOR end-to-end monitoring*, Journal of Physics: Conference Series (Vol 219, No. 4, p. 042052). IOP Publishing

[6] Castorwww.web.cern.ch/castorwww/presentations/2013 – "Overall schema of the monitoring infrastructure"

[7] Presti, G. Lo et al. *Streamlining CASTOR to manage the LHC data torrent* Journal of Physics: Conference Series. Vol. 513. No. 4. IOP Publishing, 2014.

[8] Bialecki, Andrzej, Robert Muir, and Grant Ingersoll. *Apache Lucene 4*, SIGIR 2012 workshop on open source information retrieval, 2012

[9] Abadie, Lana, et al. *Storage Resource Managers*. No. CERN-IT-Note-2008-010.2007.