Applications for a Site-oriented Analytics Platform

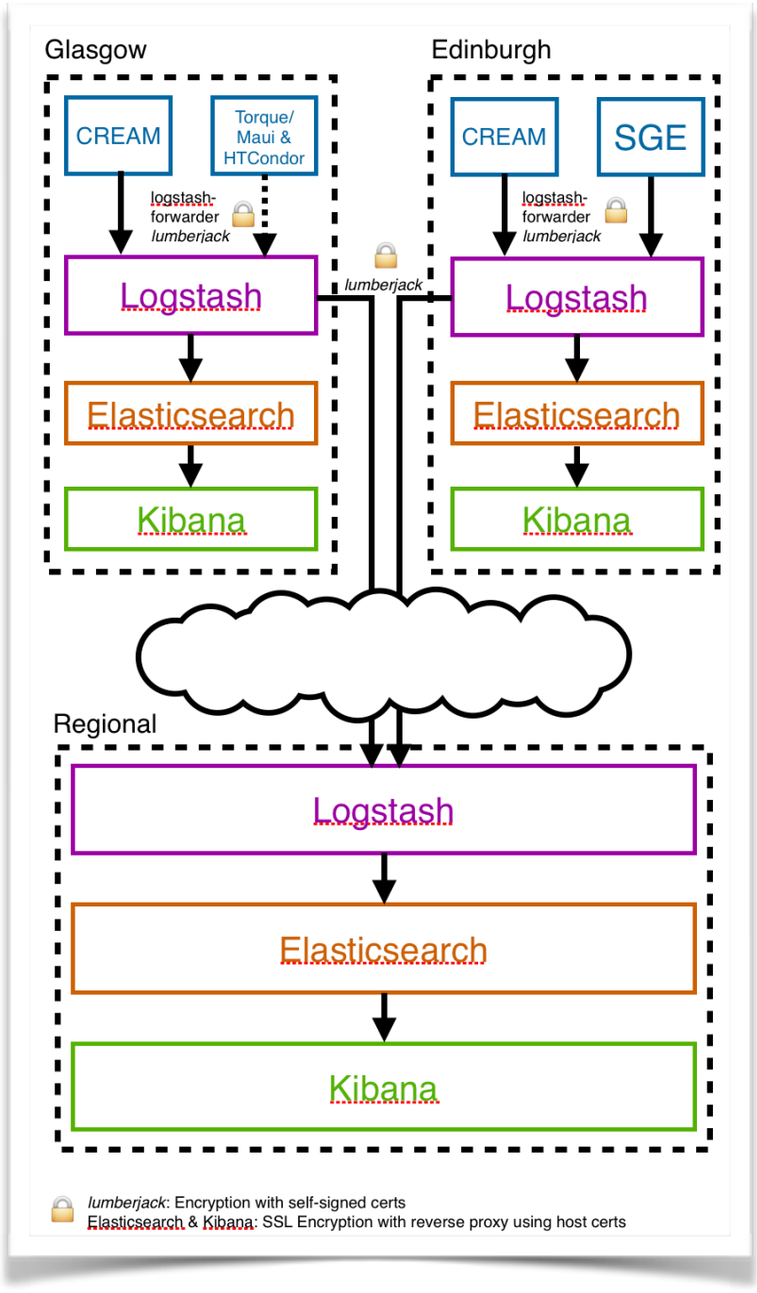
David Crooks and Andrew Washbrook
University of Edinburgh
UK HEP SYSMAN Meeting
2nd June 2015

Introduction

- Once the data is harmonised and stored in site and regional Elasticsearch instances it is possible to apply detailed analytics to gain further insight into site operations

- The ELK platform makes analytics easier to perform when processing unstructured multiple data sources

- The potential of such a platform is shown by two example applications built using the model described in the previous presentation



Process flow from hosts to Regional Logstash Instance

Application 1 - Motivation

```
01 Jun 2015 14:15:29,632 INFO
org.glite.ce.creamapi.jobmanagement.cmdexecutor.AbstractJobExecutor - JOB CREAM349229006
STATUS CHANGED: RUNNING => DONE-OK [failureReason=reason=0] [exitCode=W]
[localUser=pilotatlas03] [workerNode=eddie544] [delegationId=1433124246.563805]
01 Jun 2015 14:16:00,708 INFO
org.glite.ce.creamapi.jobmanagement.cmdexecutor.AbstractJobExecutor - JOB CREAM198870909
STATUS CHANGED: REGISTERED => PENDING [localUser=prdatlas027]
[delegationId=1431487673.580182]
```

Sample CREAM CE logs

- Logs from **CREAM CEs** were chosen as an example application to highlight consolidated middleware monitoring
- Common service used on both candidate Grid sites
- Here the test case was focussed on harvesting the **state changes** of jobs
- Useful in looking for correlations in job activity between the two sites
  - Job submission, execution and completion throughput
  - Error spikes

Log Forwarding

```
{
  "network": {
    "servers": [ "SITESERVER:5043" ],
    "ssl certificate": "/etc/logstash/logstash.cert",
    "ssl ca": "/etc/logstash/logstash.cert",
    "timeout": 15
  },
  "files": [
    {
      "paths": [
         "/var/log/cream/glite-ce-cream.log"
      ],
      "fields": { "type": "cream" }
    }
  ]
}
```

List of downstream servers listening for messages

Host-based SSL to authenticate downstream server

Single paths and globs can be used

Dictionary of fields to annotate each event

logstash-forwarder.conf on CE host

Site Logstash Configuration (1)

```
input {
   lumberjack {
      port => 5043
      ssl_certificate => "/etc/logstash/logstash.cert"
      ssl_key => "/etc/logstash/logstash.key"
   }
}
```

Listens for new messages from a Cream CE instance

Logstash configuration (input)

Logstash output sent to multiple output streams simultaneously

```
output {
   lumberjack {
      hosts => 'REGIONALSERVER'
      port => 5044
      ssl_certificate => '/etc/logstash/logstash-regional.cert'
   }
   stdout { codec => rubydebug }
   elasticsearch { host => localhost }
   file {
      path => "/var/log/logstash/debug/cetest-%{+YYYY-MM-dd}.txt"
   }
}
```

Logstash configuration (output)

Site Logstash Configuration (2)

```
filter {
   ..
   if [type] == "cream" {
      grok {
         patterns_dir => "/etc/logstash/conf.d/patterns"
         match => [ "message" , "%{CREAM_TIMESTAMP:cream_timestamp} %{WORD:cream_priority}
%{NOTSPACE:cream_logger} - %{GREEDYDATA:cream_message}"]
         ..
         add_field => [ "site_logstash", "SITESERVER" ]
       }
       mutate {
         replace => [ "message", "SITESERVER syslog %{message}" ]
       }
       if [cream_logger] ==
"org.glite.ce.creamapi.jobmanagement.cmdexecutor.AbstractJobExecutor" {
         if [cream_message] !~ "STATUS CHANGED" {
            drop { }
         }
         mutate {
            gsub => [ "message" , "\s\[localUser=[a-z].*" , " " ]
            gsub => [ "cream_message" , "\s\[localUser=[a-z].*" , " " ]
         }
         grok {
            patterns_dir => "/etc/logstash/conf.d/patterns"
            match => [ "cream_message" , "JOB %{CREAM_ID:cream_id} STATUS CHANGED: %
{CREAM_STATUS:cream_initial_status} => %{CREAM_STATUS:cream_final_status} %
{GREEDYDATA:dump}" ]
         }
      }
  ..
}
```

Custom patterns created for clarity

Append information required for the regional logstash

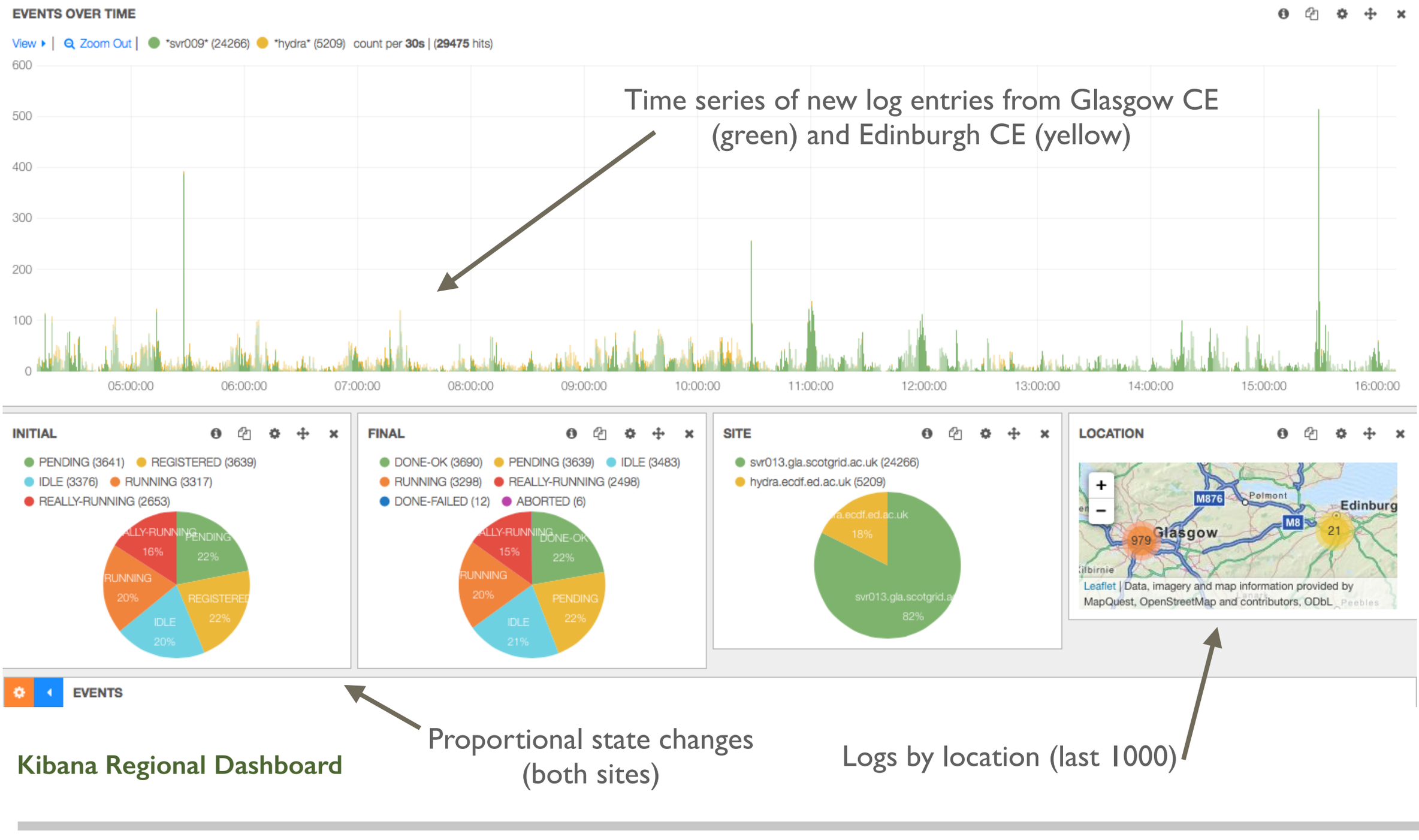
Select Cream logs and then the specific logger used in this test

Mutate action to redact user information

Logstash configuration (filter)

Kibana Regional View

- Filtered Cream CE state change logs sent to regional ELK instance at Glasgow
- Consolidated data can be viewed from Kibana dashboard



Kibana Regional Dashboard

Application 1 - Future Work

Application Extensions

- CREAM CE state change collection across two sites running for some time
- Expand logstash filter to include all useful Cream CE logging data
- Collect data from other volunteer sites into regional instance
- Enhance features on Kibana site and regional dashboards
- Abstract CE data collection to incorporate logs from other CE types (e.g. ARC)

Similar Applications

- Similar data collection and filtering logic can be applied to other site middleware:
  - Storage interface transaction logs
  - Worker node extensions (CVMFS logs and selected syslog messages)
  - ARGUS Authentication and Authorisation logs
  - BDII logging
  - APEL logging
  - Perfsonar data

Application 2 - Motivation

- This application demonstrates how useful **time-series** data can be derived from **event-based** data stored in Elastic search
- Illustrate how trending data can be extracted from local resource management system (LRMS) job accounting records
- ***Note:*** we can probably get the same application from the source files without the ELK platform - but will be less trivial to do so
- Show how can easily extend results to a suitable visualisation tool

Process diagram for the derivation of metrics and their aggregation for use in time series visualisation

Logstash Configuration (1)

```
input {
   file {
      path => "PATH/TO/GE_ACCOUNTING_FILE"
      type => "accounting"
   }
}
```

Logstash configuration (input)

Logstash was configured to parse **Gridengine** accounting record file entries for each job completed on the Edinburgh site compute resources

```
output {
   if !("_grokparsefailure" in [tags]) {
      if [type] == "accounting" {
         elasticsearch {
            protocol => "http"
            host => "localhost"
            index => "ge-accounting"
            index_type => "job_record"
         }
      }
   }
}
```

Logstash configuration (output)

Does not send to Elasticsearch if accounting file entry cannot be parsed correctly

Logstash Configuration (2)

```
filter {
   if [type] == "accounting" {
      grok {
         match => [ "message", "%{WORD:name}:%{WORD:hostname}:%
{WORD:group}:%{WORD:owner}:%{DATA:job_name}:%{NUMBER:job_number}:..]
      }
      mutate {
         convert => [ "arid", "integer" ]
         convert => [ "cpu", "float" ]
         ..
      }
      if [submission_time] and [start_time] {
         ruby {
            code => "event['queue_time'] = event['start_time'].to_i -
event['submission_time'].to_i"
         }
      }
      ..
      if [submission_time] {
         ruby {
             code => 'require "date"; event["submission_time"]  =
DateTime.strptime(event["submission_time"], "%s").iso8601'
         }
      }
}
```

Colon separated field match

Conversion to appropriate types to standardise further data manipulation

Derived job metadata stored in Elasticsearch in step with the original data source to avoid unnecessary recalculation

Date fields converted to a consistent format to allow time-based searches through the Elasticsearch API

Logstash configuration (filter)

Analytics Example

- Python script then used to produce analytics data by retrieving job metadata through the **Elasticsearch-py** API

```
results = es.search(body=
   {"query": {
      "filtered": {
         "filter": {
            "bool": {
               "must": [
                  {"range":
                     {“start_time": {"gte": start_time, "lt": end_time}}
                  },
                  {"exists": {"field": "submission_time"}},
                  {"exists": {"field": "start_time"}},
                  {"exists": {"field": "queue_time"}},
                  {"query": {"match":  {"group": groupname}}},
                  {"query": {"match":  {"slots": cores}}}
                  ]}}}},
      "size": 0,
      "aggs": {
         “queue_avg”: {“avg": {"field": "queue_time"}},
         “script_avg”: {"avg":
            {"script": "doc['start_time'].value -
doc['submission_time'].value"}}}},
      size=0,
      index=INDEX)
```

Query for previous 4, 12 and 48 hours

Query for each set of CPU resources allocated (single core, multi-core)

Query for each Virtual Organisation (VO) with access to site resources (and for each university group with access to ECDF)

Elasticsearch query using Elasticsearch-py API

Graphite Integration

- Analytics script scheduled to run every 10 minutes
- **Average queue time** for different VOs and resources sent to **Graphite**
- Graphite was considered best option for storing derived time series data
- Allows **Grafana** to be used to visualise time-series data
  - Similar in scope to the Kibana interface

```
metricvaluestr = "%.2f" % queue_time
metrictimestr = str(time.mktime(end_date.timetuple()))
message =  metricname + " " + metricvaluestr + " " + metrictimestr + "\n"
if (dographite):
  sock = socket.socket()
  sock.connect((CARBON_SERVER, CARBON_PORT))
  sock.sendall(message)
  sock.close()
```

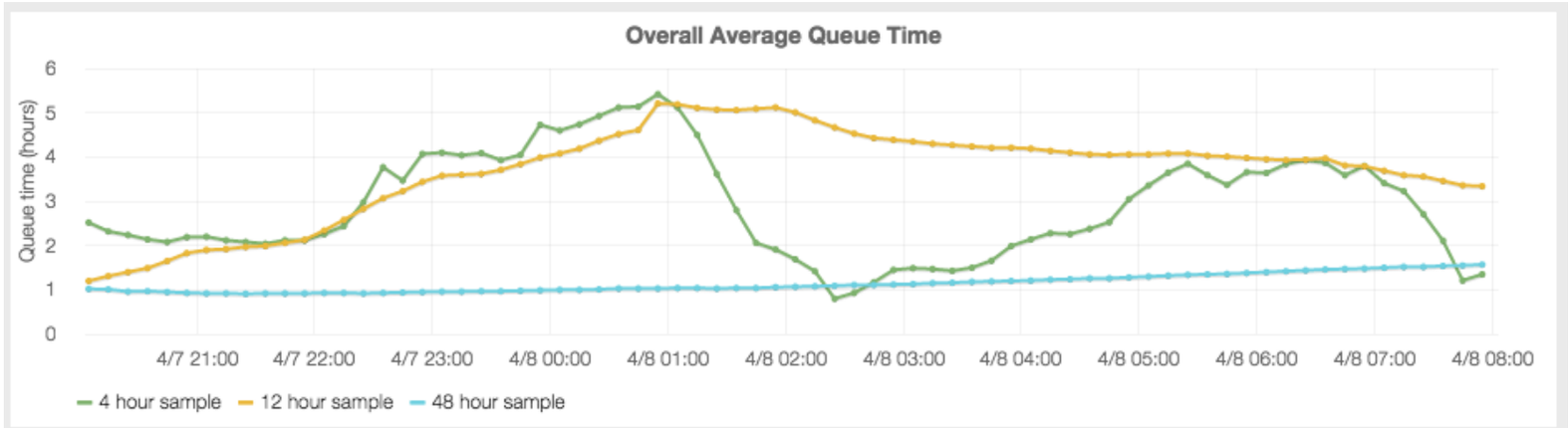
Data export to Graphite through Python

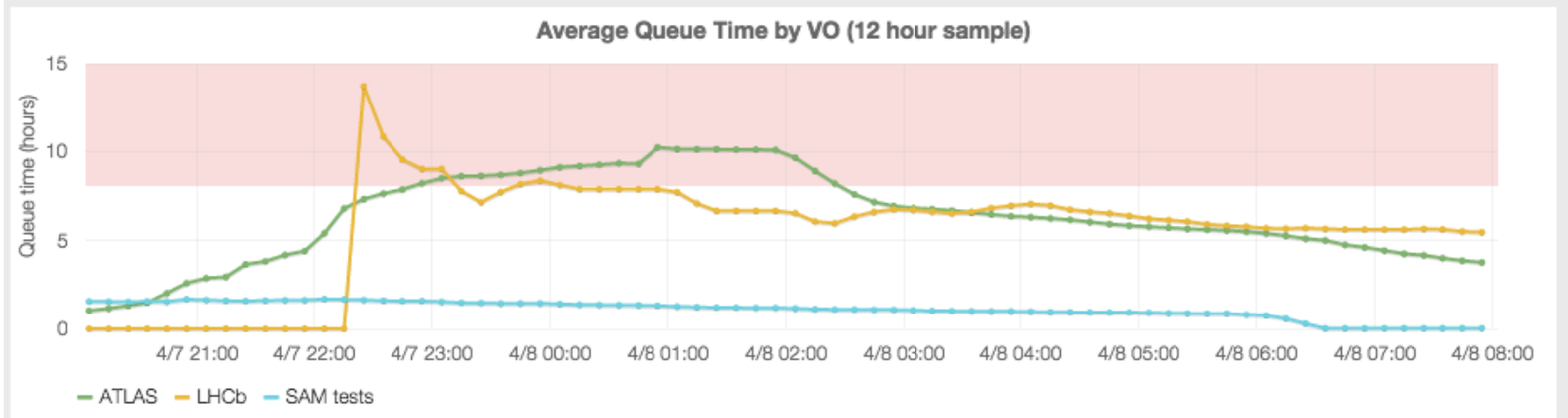
Grafana Example View

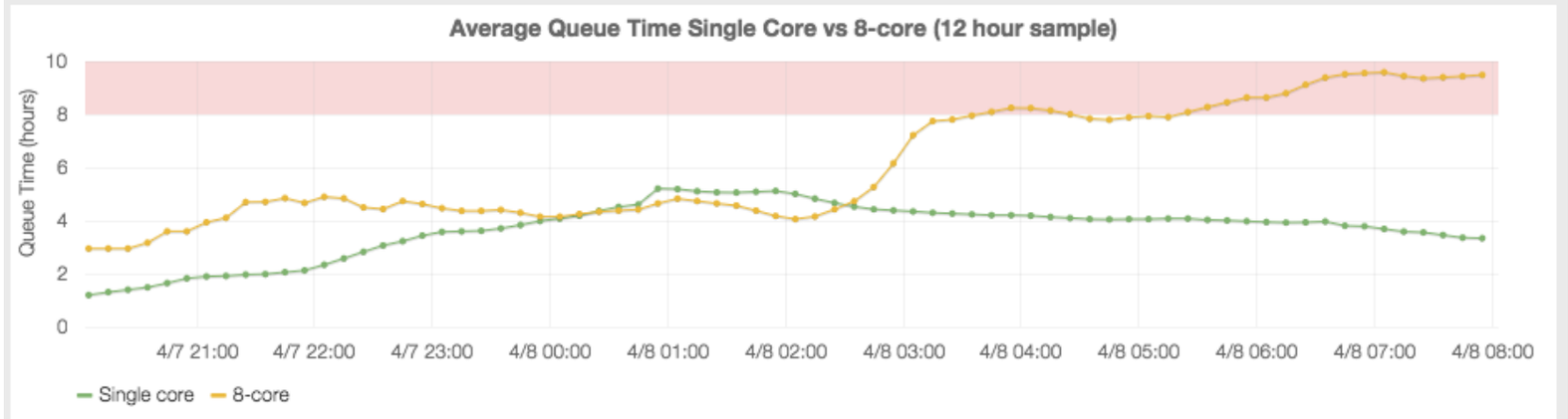
Rolling averages of overall job queue times for three chosen sampling periods indicate the transient and long term trends in scheduling conditions

The average queue time for sample VOs

Overall average queue time trends for jobs requesting different CPU resources (1-core versus 8-core)







Rolling average queue times displayed in Grafana

Ongoing Work

Application Extensions

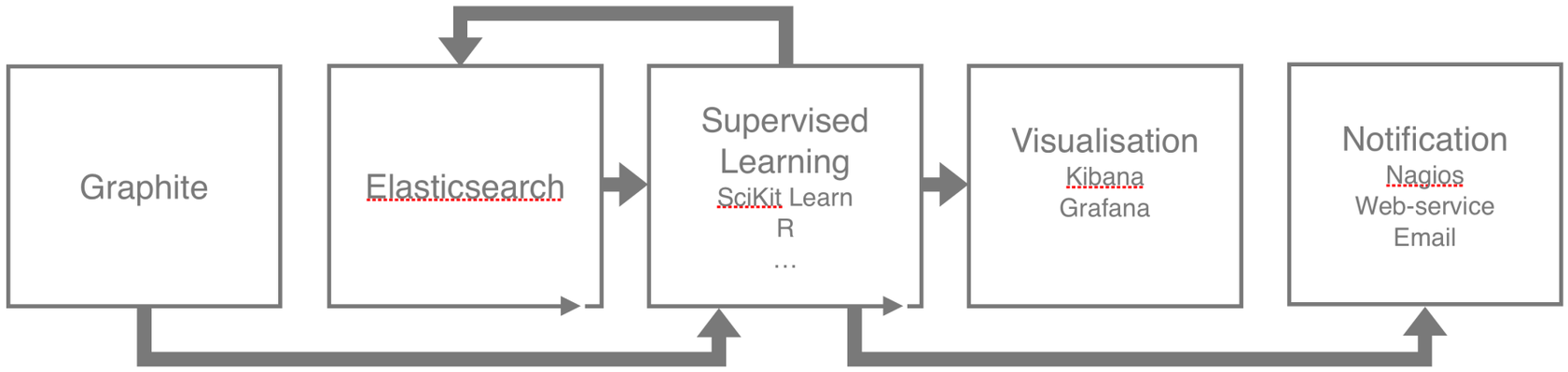
- Now looking into expanding analytics scope from Gridengine data sources
- **qstat** parsing of queued and running jobs will give a more representative **live** view of scheduling conditions
- Inclusion of new derived metrics such as **job throughput** and **rolling resource share**
- Some challenges in the integration of multiple data sources

Expansion to Regional View

- Analytics data is retrieved through the Elasticsearch API and **not** at the data source
- Therefore we can reuse the same analytics scripts for LRMS implementations at other computing sites
- Only need to make changes to the Logstash configuration
- Same application can also be run on a much larger dataset at the regional instance
- Using Condor implementation at Glasgow as second example data source

Future Directions

- Best practices need to be considered before we move to a production-level solution
  - The resiliency of dataset storage in Elasticsearch
  - Scaling limitations of data ingestion and transport
  - Extensive treatment of the security implications with sharing site data



Process diagram for further analytics applications

Further Applications

- Automatic notification of error conditions to other monitoring frameworks
- Threshold monitoring for issue pre-emption (e.g. removal of a worker node)
- Harnessing supervised machine learning techniques to automatically identify emerging trends and spurious activities
- Per-job system resource utilisation profiling on worker nodes

Conclusions

- It has been shown that the construction of a site-oriented analytics platform for WLCG computing sites is feasible by the use of emerging open-source analytics tools such as Elasticsearch, Logstash and Kibana

- Minimal extensions are required to generate a multi-tiered distributed solution which enabled relevant data to be forwarded to a regional instance to allow common trends to be identified across Grid computing sites

- Further data analytics by the use of third party tools and customised scripts acting on the primary or derived datasets

- We will now look towards developing this work in collaboration with HEP and WLCG communities with the aim of potential deployment at other participating Grid computing sites

- We would like to open this up for further collaboration to develop a common framework. **Please let us know if you are interested!**