Popularity framework for monitoring user workload

Angelos Molfetas\textsuperscript{1,2}, Mario Lassnig\textsuperscript{2}, Vincent Garonne\textsuperscript{2}, Graeme Stewart\textsuperscript{2}, Martin Barisits\textsuperscript{3}, Thomas Beermann\textsuperscript{1}, Gancho Dimitrov\textsuperscript{1}

\textsuperscript{1}University of Sydney, Sydney, Australia
\textsuperscript{2}CERN, Geneva, Switzerland
\textsuperscript{3}Vienna University of Technology, Vienna, Austria
\textsuperscript{4}Bergische Universität Wuppertal, Wuppertal, Germany

E-mail: angelos.molfetas@cern.ch

Abstract. This paper describes a monitoring framework for large scale data management systems with frequent data access. This framework allows large data management systems to generate meaningful information from collected tracing data and to be queried on demand for specific user usage patterns in respect to source and destination locations, period intervals, and other searchable parameters. The feasibility of such a system at the petabyte scale is demonstrated by describing the implementation and operational experience of a real world management information system for the ATLAS experiment employing the proposed framework. Our observations suggest that the proposed user monitoring framework is capable of scaling to meet the needs of very large data management systems.

1. Introduction

Effectively managing wide area distributed systems with large storage facilities, such as scientific data grids, typically requires recording large amounts of tracing and logging data in persistent storage. This data can be mined to provide monitoring information that can be used to identify interesting trends in data usage and facilitate the operational management of these distributed systems. The timely provision of this data in petabyte scale systems with high levels of data access requires the deployment of data structures that are able to scale to large numbers of events occurring on these systems and preprocessing methods that facilitate fast queries for extracting meaningful data usage information.

This paper presents a framework, which we will refer to as the popularity framework, for large data grids by collecting and processing large amounts of user data access events into data structures that can then be efficiently queried to provide information on data access. A data management system that employs this framework should be able to store all data access and movement operations occurring on the data grid, and provide a management information system that can be queried to provide information on data access that occurred on the data grid using search parameters, such as date and time of access, source, destination, user requesting the data access, and the type of application that accessed the data. This work builds on previous work [1] to provide a system which can supply information on data access with a daily or hourly granularity.

This paper presents a real world example where the proposed framework was used as the basis for a Management Information System (MIS) for the ATLAS high energy physics experiment[2]. This case study provides an ideal test bed for the proposed framework as the ATLAS experiment...
generates petabytes of data, has very high levels of data access, and uses a globally distributed data grid.

The paper continues by describing the popularity framework (section 2), specifically describing a method to collect data access traces (section 2.1) and an improved aggregation framework that provides user monitoring functionality (section 2.2). Following this, the paper illustrates a real world implementation of the popularity framework (section 3) to address uses cases for the ATLAS experiment, before concluding (section 4).

2. The popularity framework

The popularity framework encompasses three components: the tracer, which collects information about data access events occurring on the grid; a tracer aggregator, which creates summaries from the traces; and a query system, which searches through the summaries.

2.1. Tracer

A record of every data access event occurring on the grid creates a corresponding tracer entry. This includes data transactions that directly read, import into the grid, export out from the grid, replicate to other sites, and move data within a site (for example from central storage to a worker node). This should be done by using a lightweight submission agent API or CLI, referred to in this paper as a tracer agent. A tracer agent should be used by all grid applications to submit records, called tracer entries, describing file data access events they have enacted.

The event information that should be captured by each tracer entry is largely dependent on specific characteristics of the data grid and for what uses cases it is used for. Generally, the tracer should have the basic attributes relevant to a data grid, that is source site, destination site, time stamp the operation began, time stamp when the operation ended, and the application and user initiating the event.

These traces are collected by a front end, such as a web or an Active MQ server, and inserted into appropriate structured storage (for example, a table in relational database). Obviously, the tracer system will not record transactions occurring by end users by non grid tools. For example, a user doing a network read from within a site through a NFS mount.

The structured storage should employ a simplistic data structure to store the tracers, particularly if the data grid has a high rate of data access events. By simplistic we mean that B-tree or bitmap indexes should not be used. The traces should be partitioned on storage by the event’s starting time stamp. The ideal partition interval is likely to be daily one, but the size of the interval is largely dependent on the average number of traces received in a day.

2.2. Tracer aggregation

Omitting the use of indexes reduces the space taken by the traces on storage and removes the concurrency from updating the index leafs blocks, but it will also increase the time taken to query certain collection of traces. This problem is alleviated through the adoption of aggregation tables which the popularity framework introduces, which is described in this section.

2.2.1. Original aggregation method

The popularity framework was first introduced to provide a querable system providing data access information with daily granularity [1]. This reduces the number of traces by aggregating tracer entries into summaries using the following parameters:

1. Logical collections of files, that is grouping files into logical blocks that have a common semantic meaning (for example, they all came from the same work flow instance)
2. Period (typically per day), that is grouping all data access traces that occurred in a period, for a given data block and set of parameters
3. Other tracer parameters (for example, user, source site, and destination site)
This aggregation occurs once a day and is stored into structured storage with corresponding indexes and uniqueness constraint insuring no duplicate entries for a logical collection of files with a given set of tracer parameters exist for the same period.

2.2.2. Two level aggregation method
One of the limitations of the original aggregation method described in section 2.2.1 is that the generated summaries have insufficient granularity for queries that looked for short term data access patterns. For example, if the aggregation period is a day, and we wanted to query for the most active users in the last two hours it is not possible to do this by querying the daily summaries, and we instead have to query the tracer collection directly. For this reason we introduce a two step aggregation process, where we generate short term summaries (for instance hourly summaries) then long term summaries (for example daily summaries) from the short term summaries.

2.3. Querying a popularity system
Almost all queries should be done against the tracer aggregations, as they will result in scanning fewer rows and may benefit from accessing indexes. Only long running analytics queries should be done against the tracers where it is necessary to ascertain patterns occurring at the file level, particularly ones that need to identify patterns occurring at the file level, or queries requiring granularities under an hour. For example, calculating the daily average and maximum number of file level events occurring on the data grid per second (for example, as shown in figure 4) would be require accessing the tracers directly.

3. ATLAS popularity system
This section describes a real world implementation of an MIS employing the popularity framework. The section explains how the ATLAS popularity system evolved to cope with the increasing number of transactions, and to consequently illustrate that the popularity framework exhibits very good scalability.

3.1. The ATLAS experiment
The ATLAS experiment at the Large Hadron Collider [2] seeks to extend the frontiers of high energy physics by observing particle collisions at very high energies and luminosity [3]. The ATLAS experiment produces about 15 Petabytes per year. Approximately 5 Petabytes are directly produced by the experiment’s detector[4]. The remaining 10 Petabytes are generated by the experiment from monte carlo simulations and derived data originating from workflows that process the data recorded by the detector. These numbers account for multiple replicas on the grid, and as of time of writing, a total of approximately 87 Petabytes is being stored by the ATLAS experiment. The evolution of the total storage and number of files over time for the ATLAS experiment are visualised in figures 1 and 2.

Experimental data is disseminated to various sites around the world on the Worldwide LHC Computing Grid (WLCG) [5] and centrally managed in order to insure that the data is available to be analysed by all institutions that are part of the ATLAS collaboration. This data is stored on storage resources which are pledged by the collaborating institutes to the ATLAS experiment.

The ATLAS data model organises information in logical collections of files called datasets [6], which are the basic unit of operation. This means that when data is transferred on the ATLAS grid, collections of files are scheduled for transfer according to what datasets they belong to.

To facilitate the management of experimental data on the grid, the experiment’s Distributed Data Management (DDM) system incorporates an MIS that uses the popularity framework to provide experimental data usage information to facilitate operational activities, particularly the optimisation of pledged storage resources.
3.2. Use cases
The popularity system was made to answer particular questions from DDM operations staff, administrators, and users, regarding data access. These types of queries always have a time constraint. An example of such a question is “what are the most popular datasets accessed in the last 30 days”. Popularity here is defined by the number of file accesses that the dataset received in the specified time period.

Furthermore, ATLAS dataset names include metadata information in the dataset name. For this reason, often the searches are constrained also by dataset name, for example, “what are the most popular datasets accessed in the last 30 days matching dataset name wild card search data10_7TeV*.ESD*”.

For the process of optimising the use of pledged storage resources, it is necessary to ascertain what unpopular datasets are present at a given site, which can be deleted to free more space for new data. Therefore, it is necessary for the popularity system to be able to provide a list of unpopular dataset replicas at given site, and whether there are any policies mandating the presence of these datasets.

Grid applications in the ATLAS experiment submit traces to a web service (a cluster of Apache web servers) every time a file is accessed or moved synchronously on the grid. The traces are hash tables which are read by a python process on the Apache web servers and inserted into tracer table in a Oracle enterprise relational database. Originally, data access information was directly extracted from this table. However, with time it became apparent that for the system to scale with an increasing number of traces being generated (see figure 3), it was necessary to introduce an aggregation process that generated daily summaries.

A number of cases where specific users placed very high loads on specific sites, by initiating large numbers of concurrent downloads, meant that it was necessary to provide functionality where user activity could be monitored. In order to address this use, it was necessary to introduce aggregation at the hourly level.

3.3. The DDM Tracer system
All traces are inserted into a Oracle heap table. As these traces are submitted by client applications, there is no guarantee that each tracer is unique. For this reason, there is no primary key constraint on this table.

Originally all queries regarding data access were against the tracer table, which had two
secondary indexes to match the two dominant query search constraints (described in section 3.2), one on the event’s time stamp and the another on the dataset name. Even with the assistance of the two secondary indexes, queries became very slow on the tracer table. This was largely due to large range scans that many queries generated resulting into large numbers of random reads.

The size of range scans could not be further reduced due to two reasons. Firstly, many queries had very broad search parameters resulting to scanning millions of rows. Secondly, more constrained searches use lower cardinality attributes (for example, searches constrained by destination site), and consequently do not benefit from b-tree indexes. Furthermore, due to the very large number of insertions on this table, bitmap indexes are not appropriate as update contention of the index block may occur[7].

The ATLAS system has been able to scale with an increasing number of tracer events received over time (see figure 3). Furthermore, the ATLAS DDM tracer system has been able to cope with spikes of over 1200 tracer insertions per second (see figure 4).

![Data access events](image1)

**Figure 3.** Traces received per month.

![Data access events per second](image2)

**Figure 4.** Tracers received per second.

### 3.4. One level aggregation of traces

The popularity framework was introduced to alleviate the problems of querying a very large tracer table described in section 3.3. This was done by aggregating the traces into larger blocks. The main operational unit for data in the ATLAS experiment is the *dataset*, which is a logical collection of files. The traces were aggregated into larger blocks by dataset name, the daily period they belong in, and other transactional parameters like source and destination site.

Adopting this approach provided a much smaller table that could be queried to ascertain the popularity of ATLAS datasets. A uniqueness constraint was placed on the table as any duplicate tracer entries will be aggregated into single entries.

### 3.5. Two level aggregation of traces

Following the successful implementation of the popularity framework it became necessary to monitor user interactions with grid storage elements. This use case came about, because certain users were running grid applications that were requesting very large amounts of data from particular end points, which significantly effected the storage infrastructure at the site.

The daily aggregations described in section 3.4 cannot be used as user monitoring requires a finer granularity. They cannot be used to identify changes in user patterns occurring in the short term, that is within the last few hours. For this reason an hourly aggregation table was introduced which aggregated the traces, and the daily aggregation process was modified to aggregate hourly summaries instead of raw traces.
Since long term queries use the daily table, there is no need to maintain a history of all hourly summaries. Once a day all hourly summaries that are older than 60 days are removed from the system. A summary of the tables required for popularity can be seen in Table 1.

<table>
<thead>
<tr>
<th>Table</th>
<th>Size</th>
<th>Lifetime</th>
<th>Insertion freq</th>
<th>Oldest entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traces</td>
<td>3.3 billion</td>
<td>Forever</td>
<td>Real Time</td>
<td>14-05-2008</td>
</tr>
<tr>
<td>Hourly Report</td>
<td>7.2 million</td>
<td>60 days</td>
<td>Once an hour</td>
<td>24-02-2012</td>
</tr>
<tr>
<td>Daily Report</td>
<td>27 million</td>
<td>Forever</td>
<td>Once a day</td>
<td>17-10-2009</td>
</tr>
</tbody>
</table>

3.6. Long running analysis of the tracers

It is useful to perform long running queries operating at the file level, as well as data mining operations on the tracer table. These operations take a considerable time to run and can strain the database and impact Online Transactional Processing (OLTP).

The tracers are replicated in an eventually consistent manner to a separate Oracle instance using Oracle Active Dataguard[8] and to a Hadoop cluster[9]. This allows long running analysis to be performed using SQL queries and map reduced jobs, without impacting on OLTP workload.

4. Conclusion

This paper presents a framework for a management information system that is able to be queried for data access information with three different levels of granularities.

The framework insures scalability by employing a simple data structure that employs only a period-based partitioning scheme to organise the tracers on persistent storage without using any indexes. These traces are then be aggregated into much smaller hourly and/or daily summaries, which can be easily indexed. This approach allows a data management system to capture large amounts of information regarding data access events at very high rates, while allowing the system to be query very efficiently.

We have presented a real world implementation of this system for the ATLAS experiment that is able to provide information at real time, hourly, and daily granularities. The system has been shown to scale with increasing number of data access events in the ATLAS experiment.

References


