The ATLAS DDM Tracer monitoring framework

Dongsong ZANG\textsuperscript{1}, Vincent GARONNE\textsuperscript{2}, Martin BARISITS\textsuperscript{2}, Mario LASSNIG\textsuperscript{2}, Graeme Andrew STEWART\textsuperscript{2}, Angelos MOLFETAS\textsuperscript{2}, Thomas BEERMANN\textsuperscript{2}  
for the ATLAS Collaboration

\textsuperscript{1}Institute of High Energy Physics, Beijing, China  
\textsuperscript{2}CERN, Geneva, Switzerland

E-mail: zangds@ihep.ac.cn

Abstract. The DDM Tracer monitoring framework is aimed to trace and monitor the ATLAS file operations on the Worldwide LHC Computing Grid. The volume of traces has increased significantly since the framework was put in production in 2009. Now there are about 5 million trace messages every day and peaks can be near 250Hz, with peak rates continuing to climb, which gives the current structure a big challenge. Analysis of large datasets based on on-demand queries to the relational database management system (RDBMS), i.e. Oracle, can be problematic, and have a significant effect on the database's performance. Consequently, we have investigated some new high availability technologies like messaging infrastructure, specifically ActiveMQ, and key-value stores. The advantages of key-value store technology are that they are distributed and have high scalability; also their write performances are usually much better than RDBMS, all of which are very useful for the Tracer monitoring framework. Indexes and distributed counters have been also tested to improve query performance and provided almost real time results. In this paper, the design principles, architecture and main characteristics of Tracer monitoring framework will be described and examples of its usage will be presented.

1. Introduction
The ATLAS Distributed Data Management (DDM) \cite{1} system manages 75PB data distributed over about 130 sites in nearly 30 countries around the world on the Worldwide LHC Computing Grid (WLCG). Each site may have different local environment such as network settings and Massive Storage System (MSS). The tracer monitoring framework records the detailed information about each file operations. These trace messages are used to monitor the complicated system, find the bottleneck, schedule the data transferring and improve the system performance.

Basically, the DDM system provides some user functionalities such as dq2-get, dq2-put and dq2-ls. dq2-get downloads files from datasets stored at remote grid sites; dq2-put stores datasets in the distributed system; dq2-ls allow users to search for data in the distributed system. We take dq2-get for example to show how users all around the world use the DDM system to get data from remote sites. When a user types something like dq2-get dataset.*, he would get dataset.1, dataset.2 and dataset.3. The DDM system will launch a separate thread for each dataset, because they may be hosted in different sites. The same is true for each file in a dataset, because the files may be in different Massive Storage System, hence need different access methods. At the end of each file request, a trace message will be generated and sent to a server.
The trace message will include some information like local site, remote site, user identification. Also it will log several timestamps about the requests. As described in figure 1, the request’s start time(sT), catalogue request time(cT), relative start time(rT), transfer start time(tT), validation start time(vT) and end time(eT) will be recorded.

The volume of traces has increased significantly since the framework was put in production in 2009. Now there are about 200 thousands trace messages every hour, as shown in figure 2, and peaks can be near 250Hz, as shown in figure 3, with peak rates continuing to climb, which gives the current structure a big challenge.

2. Tracer monitoring framework

2.1. Design principles

*Easy to deploy.* The DDM software client is deployed at the sites over the WLCG around the world. Each site on the grid may have different network and computing environment. So the monitoring framework should be easy to deploy, has little affection to the application.

*Reliable.* There should not be single point of failure in the framework. Original messages should not be lost before they are correctly computed.

*Scalable.* The number of files accessed on the grid is growing very fast. So each part of the architecture should be scalable to satisfy the system need.

2.2. The architecture

The architecture of the framework is shown in figure 4. It includes the generating, computing and querying of data stream. The data stream concept means data takes the form of continuous data streams rather than finite stored data sets, and clients require long-running continuous queries as opposed to one-time queries. In this case the data stream composed of infinite ordered messages, each
message contains some key:value couples. The clients are integrated in the application and send trace messages to the message queue (use ActiveMQ[2]) by STOMP[3] protocol. The messages are then consumed by the process elements cluster. The framework provides some query APIs to support real time queries.

![Tracer monitoring architecture](image)

**Figure 4. Tracer monitoring architecture**

2.3. The sequence of trace messages

In a data stream system, the message order is an important issue, for example, if the users want to compute the messages during some specific time. As the client is distributed all around the world, with different network and system configurations, the timestamp from the client in the messages can’t be used to sort the sequence. In this framework, we use the time when a message is received by the queue server, ignoring the networking time difference.

2.4. Distributed computing of the data stream

![Distributed computing of tracer stream](image)

**Figure 5. distributed computing of data stream**

The parallel computing of the data stream needs proper schedule, high performance aggregation operation, data persistence and fault tolerant support. The scheduling of the message is in a pub-sub way: as shown in figure 5, the Process Element(PE) first subscribes to a message queue and then starts polling messages from the queue. The computed results will be pushed to the aggregator to perform a reduce operation.

When subscribing to a queue using the STOMP protocol, the ack mode[4] is set to ‘client-individual’, and the ActiveMQ extension ‘activemq.prefetchSize’[5] is used. This makes sure that no messages are lost before they are correctly computed. When a message is computed correctly, an acknowledgement is sent back to the queue server. If the number of unacknowledged messages reaches ‘activemq.prefetchSize’, no new messages will be available to this PE.
To simplify the logic of Process Element, the aggregating operations (such as count, sum, avg, max, min) uses an increment counter. The Process Element will cache and compute the messages in a specific time range, and then send a key:increment tuple to the aggregator to increase the value by value += increment. This increment model has two benefits: one is that users can get the newest query result. For example, when doing an one-day statistics, one can get the result until now without having to wait for the whole day’s data. Another benefit is that the Process Element doesn’t have to cache lots of data, and since no persistence is needed, the result can be recomputed from the queued messages.

The disadvantage of the increment model is that the aggregator needs a very high update performance since there may be many compute metrics and the update interval is short. Also, the consistence problem should be considered when there are many Process Elements. In this framework, we use column-based, multi-replica, no-sql database - Cassandra as the aggregator. Cassandra’s distributed counters[6] use hash function to map update operations to the whole cluster, and get very high performance while keep the consistence of the final result.

3. Evaluation
The tracer monitoring architecture is first evaluated in a testing environment for performance and stability, and then deployed in the DDM software stack, i.e. DQ2, on the real production WLCG grid.

3.1. ActiveMQ & Cassandra write performance
As the two very important components in the architecture, we have tested the write performance of the message queue, i.e. ActiveMQ, and tuple aggregator, i.e. Cassandra with a tool called multi-mechanize [7].

![Figure 6. ActiveMQ write performance](image1)

![Figure 7. Cassandra write performance](image2)

Figure 6 shows that a single ActiveMQ server is capable of 1000HZ trace message insertion rate, with each message about 1MB. Each insertion of a message establishes a new connection. In production, two servers are used for both the load balance and failover reasons. Figure 7 shows that the 9-node Cassandra cluster is capable of 10,000HZ insertion rate with each message about 1MB. That is much better than a relational database like Oracle can do.

According to the test result, both ActiveMQ and Cassandra have enough write performance to support the tracer monitoring framework.

3.2. Data query performance
Trace messages over a month with about 34 gigabytes and 90,578,231 rows were used for the query test. A same query was tested in four different ways.

According to the test result in table 1, the traditional ‘store and query’ method can’t satisfy the real-time aggregation query requirement, both for Oracle and Cassandra. A Cassandra column family was used to build index to make the query faster, the result improved very much, but still not optimal.

| Queried on                     | Oracle in testing environment | Oracle in production | Cassandra cluster | Cassandra with index | Cassandra distributed counter |
|-------------------------------|-------------------------------|----------------------|-------------------|----------------------|-------------------------------|
| Query time                    | 10.4 minutes                  | > 5 hours            | 28.3 minutes      | 10 seconds           | < 0.1 second                 |
Note that the Oracle in production used by ATLAS holds all the data and support other applications, while other tests were made in an isolated environment.

Oracle in testing use version 11g with configurations described below:

Hardware type: Intel(R) Xeon(R) CPU L5640 @ 2.27GHz (2/12), 48290 MB (16387 MB)
Storage: ASM and 8Gbps dual-ported HBAs, 2 storage arrays, 24 SAS disks in total, NAS on 10GigE also available

Using the data stream concept, the tracer monitoring framework computes the trace messages in real-time, and store the result for final or intermediate queries. This kind of query is very fast, independently of whether it is a simple query or a very complicated query. The message queue servers, the process elements cluster and the Cassandra distributed counters are all scalable with high performance, making the whole framework scalable too.

4. Examples of usage
The tracer monitoring framework is used by many applications or services.

4.1. Real time monitoring & data usages analysis
This is the most direct usage of the tracer monitoring framework. The real time monitoring can monitor the status of the DDM system, report errors and failed transfers, show the activities between each site.

The analysis of the history data can give some useful statistics, for example the data type percentage of the 'get' operation of files from the grid or the distribution of the users over the sites who issued the file 'get' operation, as shown in figure 8. Also metrics over time as shown in figure 9.

Figure 8. File operation statistics on dq2-get of Jun, 2011

Figure 9. File size requested per hour in the last 2 days
4.2. System simulation & work load prediction
Like other applications in the WLCG grid, the DDM application is very complicated. The impact of changing the data placement policies or network topology should be well considered before taking place. The DDM simulation project is aimed to evaluate the system under different conditions based on the trace messages.

The work load prediction can be very useful to improve the DDM system performance. For example, the number of concurrent requests can affect the performance of Massive Storage System dramatically. Reference [8] describes two models: last-7 and SARIMA to predict the file operations.

4.3. Popularity & dynamic deletion of unpopular replica
To know dynamically which files are “hot” and which are not is very useful for the data placement and deletion service. The popularity service described in reference [9] uses the recorded trace messages to analyse the popular and unpopular files, and support dynamic replica reduction.

5. Conclusion
This paper describes a tracer monitoring framework for ATLAS DDM application on the WLCG grid. It uses STOMP protocol and ActiveMQ to send and queue the trace messages, a process elements cluster to compute the data stream and Cassandra increment counters to aggregate the final result. Tests show that this framework has high performance and scalability. Several examples are showed to illustrate the usefulness of this framework.

Future work involves developing more query APIs, dynamically changing the DDM policies based on the tracer monitoring framework.

References
[1] M. Branco, et al, Managing ATLAS data on a petabyte-scale with DQ2, J. Phys.: Conf. Ser. 119 062017(2008).
[2] ActiveMQ[EB/OL], http://activemq.apache.org/
[3] The Simple Text Oriented Messaging Protocol[EB/OL], http://stomp.codehaus.org/Protocol/
[4] STOMP client frames: SUBSCRIBE ack Header[EB/OL], http://stomp.github.com/stomp-specification-1.1.html#SUBSCRIBE_ack_Header
[5] ActiveMQ extensions to Stomp[EB/OL], http://activemq.apache.org/stomp.html
[6] Multi-Mechanize[EB/OL], http://testutils.org/multi-mechanize/
[7] Cassandra increment counters[EB/OL], https://issues.apache.org/jira/browse/CASSANDRA-1072
[8] M. Lassnig, T. Fahringer, V. Garonne, A. Molfetas, and M. Branco, Stream monitoring in large-scale distributed concealed environments, in the 5th e-Science conference, Oxford, Dec 9-11, IEEE Computer Society (2009).
[9] A. Molfetas, F. B. Megino, A. Tykhonov, et al, Popularity framework to process dataset traces and its application on dynamic replica reduction in the ATLAS experiment, 2011 J. Phys.: Conf. Ser. 331 062018