Table of Contents

Storage & Aggregation Review

This document is the output of the Storage & Aggregation working group: Marian, Eddie and Julia.

It presents an overview of the current approaches in aggregation and storage in WLCG monitoring applications (https://twiki.cern.ch/twiki/bin/view/LCG/InitialArchitecture). Since both storage and aggregation can have wide interpretation, in this summary we refer to storage as any component or technology used as a persistent store for monitoring/accounting data, e.g. RDMS, files; aggregation as any algorithm or component computing aggregates that requires significant processing or storage resources, e.g. PL/SQL procedures implementing specific aggregation algorithm would qualify; select * from group like queries would not (simple aggregation SQL queries often used in web interfaces are not covered).

As requested in the WLCG monitoring consolidation meeting 18 July 2013, we attempt to answer the following questions regarding storage and aggregation:

- Are we doing things consistently for all of our applications?
- If we are doing things in multiple ways, is there a good reason for it?
- Are there any new technologies that would help us here?
- How long would it take to change?
- How would that impact the other layers?

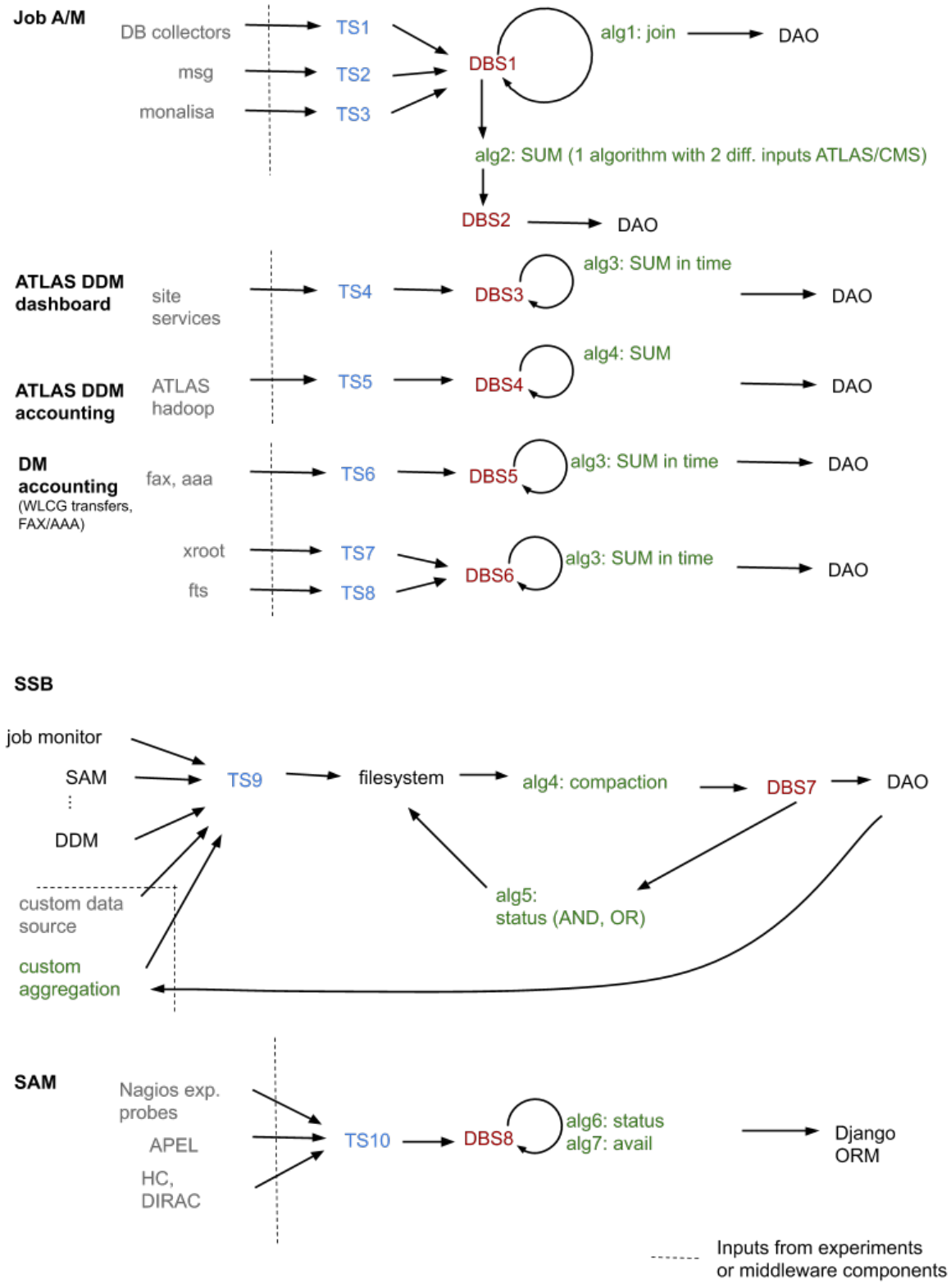
Are we doing things consistently for all of our applications?

In this table, we summarise API, statistics and implementation technologies used for the key aspects of storage and aggregation by the various WLCG monitoring applications.

App	Storage					Aggregation		
	API IN	Type/Impl.	Size in total	Stats	API OUT	Type/Impl.	Stats	API
Job Monitoring	DB Collectors + Messaging for ATLAS MonALISA for CMS	REL/ORACLE	1520GB (including Accounting) for ATLAS 1573GB (including Accounting) for CMS	ATLAS: ~10 to 11 million rows per day for job status updates considering a job's life cycle CMS: 400K rows per day		MERGE / Oracle Procedure for ATLAS INSERT or UPDATE / Python collector for CMS	~1.1 million rows per day for ATLAS ~ 400K rows per day for CMS	DAO H (serialis JSON, CSV)
Job Accounting		REL/ORACLE	Same as above	~1.1 million rows for ATLAS per day 400K for CMS per day		SUM, AVG, .. / DBMS scheduled	Hourly, daily & 5 min snapshot (pending/running jobs) ~65K for the daily summarised statistics	DAO H (serialis JSON, CSV)
	HTTP Post	REL/ORACLE	1787GB					

ATLAS DDM Dashboard				Keeping only last month's stats		SUM in time / ORACLE PL/SQL	10 min & 24 hours bin	DAO H JSON serialis
ATLAS DDM Accounting	HTTP Get Collector	REL/ORACLE	-	1.6 million rows per day		SUM /Python Collector for initial aggregation and Oracle procedure for final aggregation	75K per day	DAO H (serialis JSON, CSV)
DM accounting (WLCG Transfers Monitoring)	Messaging AMQ	REL/ORACLE	Keeping RAW data for 3 months - 242 million rows	2.6 million rows per day		SUM in time / ORACLE PL/SQL	153K per day 10 min & 24 hours bin	DAO H JSON serialis
DM accounting (FAX & AAA Monitoring)	Messaging AMQ	REL/ORACLE	Keeping RAW data for 3 months - will be indefinitely really soon - 202 million rows	2.2 million rows per day		SUM in time / ORACLE PL/SQL	? per day 10 min (13 million rows) & 24 hours bin (4.5 million rows)	DAO H JSON serialis
SSB (also SiteView)	Collector HTTP Get	Files on file system	300MB daily and we keep data for up to a month	2 million rows per day		Weighted AVG, compaction, compression / Python	Bins depending on status change	DAO A HTTP J serialis
SAM	Messaging	REL/ORACLE	1.2 TB - keeping indefinitely	1 million rows per day	API accessing RAW data	STATUS, AVAIL / ORACLE PL/SQL and Python	Status changes (25k/day status) Avail hourly, daily, weekly, monthly (132k/day site avail)	Django ORM H (JSON, serialis
Google Earth		KML files						

The following two diagrams show the different aggregations used together with their inputs and output (diagrams outline a simplified data flow for each application; legend: inputs from experiments or middleware components or probes - grey; TS (blue) - transfer schema; DBS (red) - database schema; DAO - Data Abstraction Object; aggregation algorithms are shown in green via alg#: brief description of algorithm)



In summary, WLCG monitoring applications are now storing data three different areas of data: raw data with short lifetime; aggregated data with longer term lifetime (sometime indefinite) and metadata like e.g. topology, profiles, configurations, etc. Overall data flow of WLCG monitoring is using ten different transport schemas for transferring raw data that are in turn stored in eight different database schemas and processed with at least seven different aggregation algorithms (additional aggregation is also performed outside of the system by the experiments, cf. SSB; most of the aggregations need additional metadata for processing). Currently Oracle and classical files are used as the core storage technology. Most of the aggregations are written in PL/SQL, but few of them are also implemented in python. Existing aggregation algorithms can be

classified into three different categories: data cleansing and filtering; computation of throughputs or summary statistics - accounting (SUM, SUM in time, AVG, etc.) and site status and availability computations. In terms of throughput the highest load is seen in Job monitoring application with 10-11 millions updates per day. Some of the applications require more than 1 TB of storage for raw data (largest database is ATLAS DDM Dashboard with 1.7 TB, followed by Job monitoring and SAM). Finally, storage and aggregation layer also contains data abstraction API, which currently adopts Data Access Object (DAO) pattern, however different implementations are used between Dashboards applications and SAM (relies on Django ORM).

If we are doing things in multiple ways, is there a good reason for it?

Currently, single technology is used to store and aggregate data (Oracle PL/SQL with few exceptions - SSB/SAM use external python scripts to process data). Different applications have different data flows and usually do not share anything in common apart from underlying technology (or programming language).

The following reasons were identified justifying the differences:

- Historical as some of the applications evolved independently for a number of years
- High diversity of inputs that correspond to many different sources of data, but also differences between technologies adopted by different experiments
- Different processing requirements, ranging from near real-time use cases needed to support experiments operations to accounting and reporting that are only used on weekly or monthly basis

However wherever possible we should try to converge on the common schema. One possibility for example are SSB, MRS and ACE, which contain metrics (as a function of time) assigned to set of instances. Instances are either independent or have very simple hierarchical dependencies.

Are there any new technologies that would help us here?

In storage, there are following areas (types) of storages:

- Key/value stores (memcache, redis)
- Aggregate(Document)/Columnar stores (Mongo, HBase, Cassandra, ElasticSearch, SOLR, RIAKsearch, ThriftDB)
- Scalable filesystems (HDFS)
- Relational DBs
- Graph DB (Neo4j)
- Round-robin DBs (Carbon/Graphite, RRD)

It's important to note that different storage types offer different capabilities, which makes them suitable for different types of storage/aggregation problems. In our view different storages can be seen as alternatives that can run in parallel with existing solutions (not as replacements) to improve a specific storage or processing task in the application. While some of the recent technologies are promising huge benefits, there are areas where relational databases are still the best option (e.g. catalogues, many relations and no prior knowledge on what queries will be used). For our use cases (storage of the time-series data and its subsequent aggregation), existing solutions in aggregated/columnar stores and scalable filesystems can be evaluated to understand if higher scalability and robustness can bring significant benefit to the current design, development and operations of the WLCG monitoring applications.

In aggregation (data analytics) we see two main areas of work:

- Event processing frameworks/libraries, e.g. Riemann, SPARK

- Full-text search engines with customizable aggregation functions - ElasticSearch (Lucene), RIAKsearch, ThriftDB

For our use cases evaluating at least one from each area would be beneficial to understand if some of the existing aggregations can be performed in a different way, re-using work already performed within the open source projects, while at the same time gaining experience and insight on how these projects are used to monitor computing infrastructures.

In data abstraction API different implementations are used currently in Dashboard applications and SAM. Converging to a single solution and evaluating many existing open source projects implementing DAO pattern would be certainly beneficial.

In order to help with adoption of the new technologies and approaches it's important to first decouple the existing application components into well separated areas (or layers) that can be easily replaced with alternative projects or technologies.

How long would it take to change?

The actual time is very difficult to estimate as this will likely depend on each application and the time needed to accomplish the following tasks:

- Decoupling the existing components of the applications into areas or layers that can be independently changed without affecting the rest of the system.
- Evaluation of the new technologies, prototyping and gaining experience in deploying and operating them
- Convergence into a common strategy on adopting and supporting technologies to be used by different groups within IT

How would that impact the other layers?

Assuming decoupled components are in place, the most affected layers would be those connecting to output APIs (DAO), i.e. visualization layer or third party systems. With decoupled aggregation and storage the actual impact would be low as the APIs can be kept stable while changes are performed within the layer (however performance and usability might vary depending on the underlying technology).

This topic: LCG > StorageAggregationReview
Topic revision: r13 - 2013-08-02 - JuliaAndreeva