ATLAS Database Project

DISTRIBUTED DATABASE SERVICES CLIENT

Authors:	**Yulia Shapiro, Alexandre Vaniachine (editor), Torre Wenaus**
Date:	**July 8, 2004**
Project:	**ATLAS Database**
Activity:	**11. Distributed Database Services**
Document link:	**http://atlas.web.cern.ch/Atlas/GROUPS/DATABASE/project/services/client.pdf**

Abstract: This document defines the database client library software layer for distributed database services access in ATLAS Database Project. The Project plan prioritizes rationalization and cleanup of how server specification is done in applications which access database servers. The client library implements a consistent strategy for database server access. The Distributed Database Services client library serves as a unique layer for enforcing policies, following rules, establish best practices and encode logic to deliver efficient, secure and reliable database connectivity to applications in a heterogeneous distributed database services environment. This document collects requirements, outlines architecture and the workplan. The implementation responsibilities are also discussed.

Document Log

Issue	Date	Comment	Author
0-3	30/06/04	Initial version	A. Vaniachine
1-0	03/07/04	Architectural diagram added	A. Vaniachine
1-1	03/07/04	Incorporated feedback	A. Vaniachine
1-2	03/07/04	Added WBS items	A. Vaniachine
1-3	06/07/04	Expanded requirements	A. Vaniachine
1-4	08/07/04	Added executive summary	A. Vaniachine

Document Change Record

Issue	Item	Reason for Change

CONTENT

1. EXECUTIVE SUMMARY

In ATLAS Computing Model various applications require access to the data resident in the relational databases. Examples of these are databases for detector production, detector installation, survey data, detector geometry, online run bookkeeping, run conditions, online and offline calibrations and alignments, offline processing configuration and bookkeeping. In a current phase of ATLAS Software Development process applications communicate with various relational databases in a traditional approach (Figure 1).

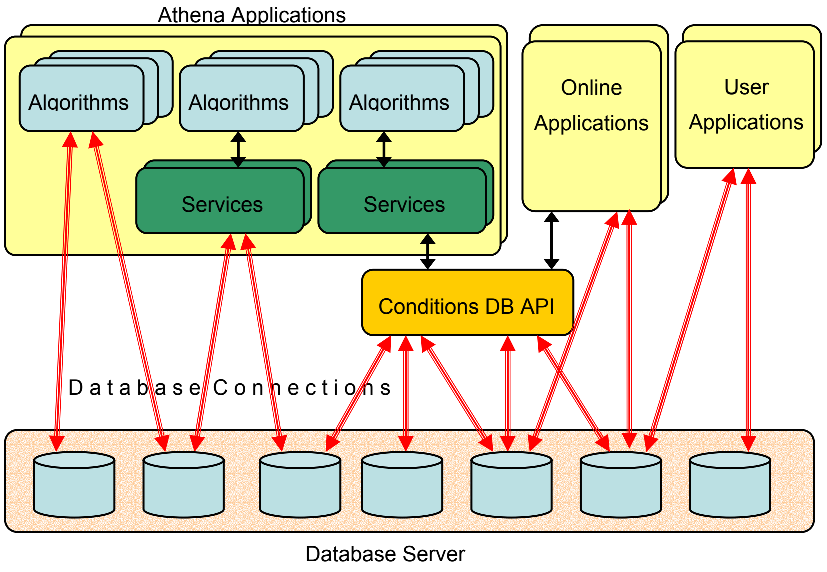


Figure 1. Data workflow in a traditional approach

The ATLAS Database Project is responsible for ensuring the integration and operation of the full distributed database and data management infrastructure of ATLAS [1]. The Distributed Database Services area of the Project is responsible for design, implementation, integration, validation, operation and monitoring of database services. To achieve the integration goal the Distributed Database Services client library serves as a unique layer for enforcing policies, following rules, establish best practices and encode logic to deliver efficient, secure and reliable database connectivity to applications in a heterogeneous distributed database services environment (Figure 2).

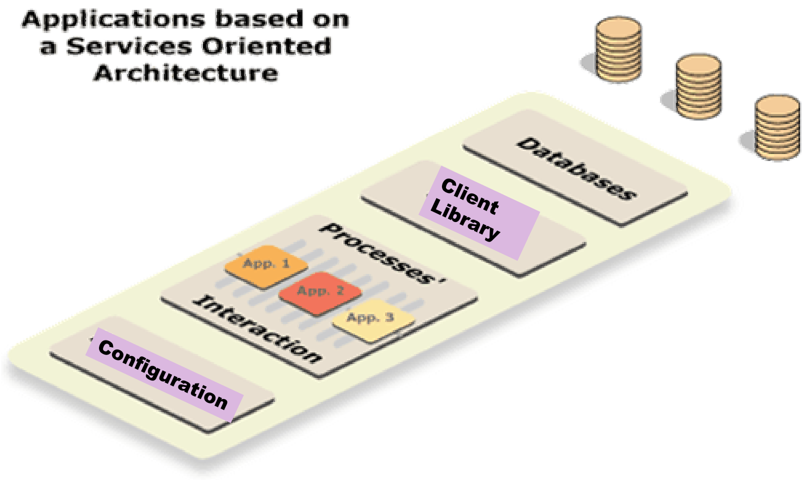


Figure 2. Integration of workflow though the client library services layer

2. REQUIREMENTS FOR DATABASE COMMUNICATIONS MANAGEMENT

2.1. PRIORITY REQUIREMENTS

In a production environment of ATLAS Data Challenges and Combined Testbeam efforts applications have accessed relational databases for various reasons. We have learned of various new database connectivity requirements that have to be satisfied in different software domains. A representative sample of these requirements is listed below.

2.1.1. Connection Indirection Layer

The distributed database servers' configuration in ATLAS is not static. To shield applications (deployed world-wide) from changes in the distributed database servers' configuration an extra layer of indirection (e.g., logical-to-physical database server mapping) is required.

ATLAS Database Project Plan document [1] prioritizes the rationalization and cleanup of how server specification is done in jobs which access database servers. It should be easy for a user to configure a job without detailed knowledge of server location. This is also important for supporting server failover to backups.

2.1.2. Access Model Uniform across Domains

As a result of the inherent differences in the online and the offline Computing Models and their corresponding database access modes we have encountered difficulties during database access in offline applications using the libraries developed specifically for the online environment. In short, the online Computing Model assumes a secure local environment behind the closed firewalls, the offline Computing Model has to provide access to distributed databases deployed across the wide area network worldwide. In addition, in the online environment a limited (less then a dozen) number of clients access the database concurrently for writing, the offline Computing Model has to provide access to thousands of concurrent client applications accessing the database for reading. A model uniform across both domains is required for robust database access.

2.1.3. Database Connection Management

To scale powerful Athena/Gaudi on-demand data access architecture [2] (designed primarily for a file-based data access) for the use case of ATLAS Data Challenges with thousands of application instances concurrently accessing database-resident data an effective database connection management solution is required.

2.2. DISTRIBUTED DATABASE SERVICES CLIENT

To satisfy the above and other requirements a new software component has to be developed. To converge on a consistent strategy for database server access this component – the Distributed Database Services Client library – will encapsulate on the client side the database connectivity management. This will be a single place for enforcing policies, implementing rules, establish best practices and encode logic to deliver efficient, secure and reliable database connectivity to applications.

ATLAS Computing Model requires that access to the data should be transparent and efficient [3]. In addition to file-based event data, ATLAS data processing applications require access to large amounts of valuable non-event data (detector conditions, calibrations, etc.) stored in relational databases.

To provide efficient, robust and secure applications access to database-resident data interaction the Distributed Database Services client library functionalities should satisfy additional requirements listed below.

2.3. SCALABILITY REQUIREMENTS

2.3.1. Connection Pooling

To streamline overhead form making a new database connection for each application request to for a database-resident data a connection pooling is required. The client library should keep a set of concurrent database connections in a 'persistent' connections pool. For each Athena service request for a database connection the pre-established database connection is activated and dispatched to the service. The library should support heterogeneous databases.

2.3.2. Connection Fallout

To assure that Athena services follow the established best practices and place connections back into the pool the service can not submit a new connection request until the previous connection is returned to the pool.

2.3.3. Connections On-demand

The library should take full advantage of the Athena/Gaudi on-demand data access architecture. To avoid making unnecessary connections to databases sources a "lazy initialization" is required. The connection should not be created in the application initialization phase. To conserve resources the connection should be created transparently the first time data is needed.

2.3.4. Connection Timeouts

According to the ATLAS Computing Model a distributed chaotic resource usage should be supported. In a resulting use case of multiple application jobs starting simultaneously on one cluster a configurable timeout value is required for new connections. To balance performance and resource usage the idle connection should be destroyed after a configurable timeout period. Both timeout values should be configurable.

2.3.5. Connection Retries

Retries are proven to be helpful in case when the database server is just busy with other clients, which is becoming a requirement for a use case of multiple application jobs starting simultaneously on one cluster. A configurable maximum number of connection attempts is required.

2.3.6. Connection Failover

Another improvement in the database connectivity robustness will be provided by transparent application failover logic (when the number of the specified retries failed, try to connect to the replica server) and automatic updates of the initial configuration of database replicas.

2.3.7. Load Balancing

The library will reuse POOL catalogue infrastructure (suggested by Dirk Duellmann) to provide logical-to-physical database server mapping, such that physical connections are forwarded to replica or failover databases without impacting the client. A configurable replica database server catalogue will be used for connection load balancing. Support for different load balancing strategies is required.

2.4. SECURITY REQUIREMENTS

2.4.1. Client Passwords

The library will hide database connection credentials, which is required for application code exposure on the web. Initially the credentials repository component of the library will secure the client clear-text passwords required for database authorization.

2.4.2. Client Certificates

According to the ATLAS Database Project plan for support of the grid based access to event and non-event data the client library will be grid-enabled. It will be capable of forwarding the grid certificate proxy used by application to grid-enabled database servers or grid-enabling layers surrounding database.

2.5. SUPPORT REQUIREMENTS

2.5.1. Client Logging

The Athena application issues uncoordinated requests for various database services: IOVDB, ConditionsDB, etc. To analyse, monitor and debug the application sequence of database queries a configurable level of client-side logging is required.

In addition, the library should provide client logging capability on the server-side. For every client database connection the library will insert log records into database logging tables. Such logging has been shown to be instrumental for debugging the reported client problems.

2.5.2. Performance Monitoring

A related to logging issue is performance monitoring both on the application and on the database server side. Capture of query execution timing and server load information is required.

2.5.3. Error Reporting

Another required feature is a centralized place for client connection errors reporting. In case of a complete failure the client library will dump the debugging information required for error reporting and follow-up problem resolution.

2.6. LIBRARY REQUIREMENTS

2.6.1. External Dependencies

The client library should be lightweight: no heavier or more complex then necessary, with minimal external dependencies. For the library to be an important tool in online these are necessary requirements (but attractive also in other contexts).

2.6.2. Backward Compatibility

To ease the process of ATLAS software migration to the new library the compatibility interface is required. The shallow compatibility library will intercept all Athena application calls for database services and service them through the new library. In that way the required changes in the available Athena services code will be minimized delivering autonomy in the applications' development and maintenance.

2.6.3. Software Assurance

For assurance of each of the requirements implemented a corresponding execution tool is required to test the service function. To assure concurrent interoperability of service functions a comprehensive homogeneous tool suite is required.

2.6.4. Documentation

An up-to-date user guide and a comprehensive documentation for core developers should accompany the software development process.

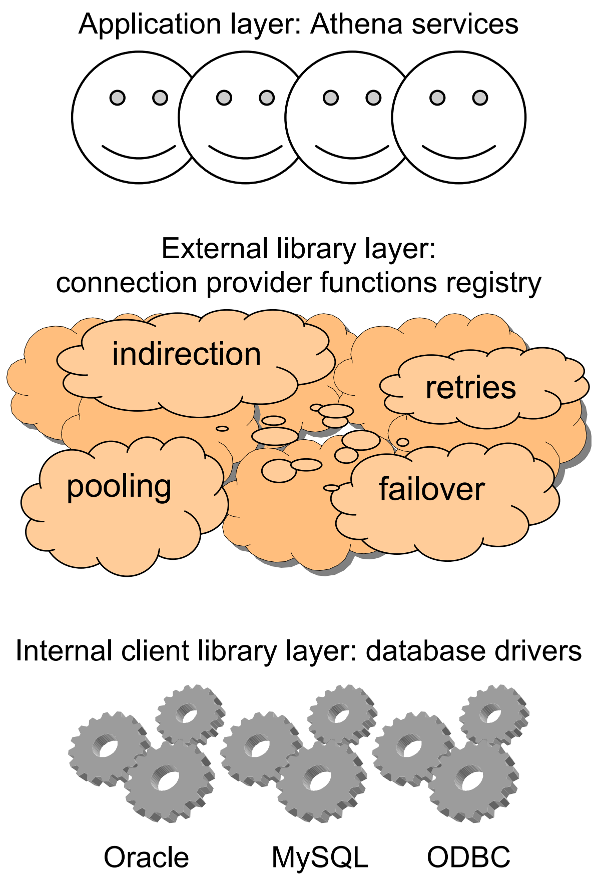
3. EVENT-DRIVEN SERVICE-ORIENTED ARCHITECTURE

3.1. DESIGN RATIONALE

To service the application requests for database-resident data in the Athena/Gaudi on-demand data access architecture an event-driven system is required. The preferred way to satisfy ATLAS Athena application framework requirements will be to develop a service, e.g., the `DatabaseMgrSvc` that will communicate with an underlying Distributed Database Services client library. That library in turn will supply the execution of requested services. The rationale for such design is to make possible reuse of the library by other ATLAS software domains, e.g., in online. The design based on reusability and interaction also makes the library easy to use in the common LHC project on Distributed Database Deployment.

3.2. ARCHITECTURAL PLATFORM

To avoid over-design and integrate various requirements in one library we chose a service-oriented platform to capture a complete workflow of the event-driven communications in unique layer. The client library repository registers interoperable service functions. Each autonomic service function satisfies a particular well-defined self-contained requirement of Section 1 that does not depend on the context or state of other functions. A loosely coupled service functions communicate with each other via data passing. Synchronous interaction of two or more service functions can execute the requested database connection action. The architecture hides the complexity of heterogeneous database technologies by separating the library into external and internal layers (Figure 3).


Application layer: Athena services
External library layer:
connection provider functions registry
indirection
retries
pooling
failover
Internal client library layer: database drivers
Oracle
MySQL
ODBC
Figure 3: Multi-layer service-oriented client architecture

3.3. LAYER DECOMPOSITION

The external event-driven layer encapsulates generic logic of database connections rules that gives services to clients. The external layer provides management of database drivers, database connections and Connections/Servers lists. To satisfy the requirement of heterogeneous database connectivity the internal library layer is composed of a number of technology specific database drivers providing support for MySQL and Oracle initially and ODBC driver later. The internal layer executes actual connection opening according to the required driver dispatch.

Through emphasis on a practical approach, and by avoiding too much abstraction and flexibility at early stage we concentrate on solving actual problems of ATLAS operations workflow.

4. WORKPLAN AND MILESTONES

4.1. RESPONSIBILITIES

Yulia Shapiro agreed to be the main developer for Distributed Database Services client library.

4.2. PRIORITIES

Our plan is to deliver a basic tool early and incrementally add functionality.

The fist priority will be reuse of the POOL indirection mechanism to handle the logical-to-physical database server mapping.

We believe that it would be reasonable to offer ‘simulated’ support for logical database connections, offering clients the use of logical connection names but internally just mapping them to physical names locally within the connection manager. This would allow early, centralized management of database connections, albeit without the full flexibility of the eventual database, but eliminating hard-wired connection strings in Athena jobOptions, client configuration files, ConditiondsDB folder attributes and such.

4.3. MILESTONES

4.3.1. Indirection

On a later timescale the library will be actually using a POOL catalogue-derived logical name database for the logical-to-physical database mapping. Servers list (indirection mechanism) will provide logical-to-physical mapping between the database server and its replicas.

4.3.2. Pooling

The next milestone will include the functionality of the service class `DBPoolConManager` that will create and manage different connection Pools. This service holds connection pool of connections previously open by application. Connections list will provide mapping to connections known to application. In case connection to specific database is needed, the request will be supplied to connections list including database driver and user credentials, verify that this type of connection exist and return message to connections registry, from which handle will be returned to a new or already existing connection.

We will provide two connection modes: in one mode this Pool will have open connections to database ready, in a second mode each method `getConnection()` try to open connection using specific `database, user, password`. The second method is preferred for holding specific pool of open connections, since most of `user/password/database` credentials are created for groups. In that use case the `getConnection()` method will dispatch handle to already open connection. This Pool will be created using parameters `DB_URI, user, password, max_conn`.

Class `DBPoolConnections` will provide methods to connect to DB using open Pool connection provided by `DBPoolConManager`, in case there are no connections available because it outnumbers `max_conn`, it will retry for specified `time_out` period (specified for each `getConnection()`) and then try to connect to replica servers, using e.g., the `DatabaseServersCatalog.xml` file to resolve indirection. After successful completion, a handle to open connection will be returned to pool connections vector.

4.4. INTEGRATION

At a later stage the library will adopt the preferred technology for the client-database interaction that will be selected in the course of the ATLAS Distributed Database Services project. This technology will address the question of how clients should access databases. We will have to figure out what the right approach is. Current ATLAS two-tier architecture – a direct use of database client software is one option but it brings complications particularly with Oracle (complexity, license). People are experimenting with SOAP but encountering performance degradation. Several experiments and projects reported various benefits from insertion of additional software layers between the client application and the database server. Jack Cranshaw proposed recently general four-tier architecture to optimize the client interaction with the database. Evidence of a four-tier approach benefits was presented in a recent talk by Lee Leuking. In this particular implementation Tomcat/Apache servers physically close to the database handle the direct database interactions, and clients access data via http requests to this server, receiving the data packaged by the server essentially as a blob with only a thin HTML/XML wrapper, apparently greatly reducing the web services performance hit. A further nice feature is that the client interaction goes not directly to the Tomcat server but to an HTTP caching proxy (e.g. squid) which can sit close to the client (e.g. on a gateway node), and of which there can be many. This gives scalability, and automatic caching (for reuse) of the data that is actually used locally (and only the data that is used locally).

4.5. WBS AND COMPLETION MILESTONE

The overall completion milestone is in time for the Data Challenge 3 ATLAS software release. We will plan a series of the progressive release dates before that. For actual dates we are in the process of coordinating these with the Distributed Database Deployment project workplan in preparation. A table below presents the work breakdown items from the ATLAS Database Project WBS.

Distributed Database Services Client WBS

WBS	Name
1.3.11.5	Distributed database services client
1.3.11.5.1	Architecture design
1.3.11.5.2	Connection indirection
1.3.11.5.3	Connection retries and timeout
1.3.11.5.4	Connection pooling
1.3.11.5.5	Failover and load balancing
1.3.11.5.6	Advanced connection management features

5. REFERENCED DOCUMENTS

[1] ATLAS Database Project Plan.	Editors: Richard Hawkings, Torre Wenaus http://atlas.web.cern.ch/Atlas/GROUPS/DATABASE/project/mgmt/AtlasDBProjectPlan.pdf
[2] ATLAS Common Framework	http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/General/Documentation/AthenaDeveloperGuide-8.0.0-draft.pdf
[3] ATLAS Computing Model	R. Jones, D Malon, D Quarrie, T Wenus, D Barberis, G Poulard, S Jarp, R Hawking, R Dobinson http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/computing-model/comp-model-dec03.doc