

CAS query system

Version 1.0.3

William O'Mullane, Aug 4, 2003

The original idea here was to have batch and quick access to CAS. It now appears that allowing temporary tables means we may have 3 modes of access. These would be

1. Quick Access (no registration)
2. Registered Quick Access (registration required allow temp tables)
3. Batch Access (registration required allow temp tables).

The Job control described below for this could also be used for Condor Monitoring. Below where User Screens are mentioned these describe user applications (initially web pages). Where WebServices are mentioned take this to mean SOAP services available via http.

WebServices

This system is underpinned by WebServices. We should pass the WebServicesID to the web service for authentication of each transaction. In any aspx pages we can cache the WebServicesID in the users session thus making this seamless to website users i.e. they just log in once and as long as they have a session (standard timeout 20 mins) we will always get the WebServicesID from the session.

In this way writing custom clients even from outside may be done easily.

There should also be an anonymous method, probably by having a special anon user with, which will not allow use of MyDB.

Privileges

A simple privileges by token system shall be used in the system. Currently we envisage only four privilege tokens. Tokens will be assigned to user accounts

- Query - user may run queries (all)
- Admin - Admin user may do admin functions such as create users
- Group - User may create a group and invite other users to join
- MYDB - User may have a MYDB for local storage (all)

User Registration

Registration would be required for Batch Access and for Quick access using temp tables. The current quick access without registration should be kept – this is used for SDSSQA etc.

We need a registration screen where we take

- User Name
- Email Address
- Password (optional)

This information needs to be stored in a Database (lets call this table Users).

When the user is created we shall concatenate the userid and password and hash them to gain a Long number. This shall be stored in the User Record and must be unique. If it is not unique we can keep incrementing it until it is unique. This number will form the WebServices token for identifying this user. This token will always be accepted by WebServices thus allowing direct access without the need to log in.

User Screens

Need screen for creating an account.

We need a screen for updating user information.

We need a login screen (this means we start using sessions in our aspx pages).

Users should be able to enter either their userid or email address and password to login.

The WebServicesID should be shown to the user upon creation of the account.

Need a forgotten password screen which mails the password to the user. We could do something more sophisticated with questions etc. but this is probably sufficient for now.

WebServices

We need an anonymous user who can create accounts.

The following WebServices will be needed (wsid= WebServicesId, User is a Structure containing the user information):

- User createAccount(long wsid,string UserId,string Name,string Email,string Password) – create account , send email to user, and to the administrator
- Long getWebServiceId(string userid, string password) – userid could be the email address or userid. Return the WebServicesId.
- String forgotPassword(long wsid,string email) – email password for the given email address to the given email address, if it matches one in the Users table.
- Bool updateAccount(long wsid, User user);
- User[] findUser(wsid,string predicate) – where clause for searching for a user or users.
- String getUniqueWebServiceId(long id) – check if id is unique if not add numbers on the end until it is unique and return it.

User Table Schema

Name	Varchar(80)	The persons real name
Email	Varchar(80)	Persons email address
Password	Varchar(20)	Optional password
WebServicesId	Int64	Unique identifier for this user for web services, automatically generated when account is created
Privileges	Varchar(200)	Comma separated list of privileges for this user. Currently these would be: UserAdmin JobControl MyDb

		If this is null then the user may submit queries and that's it.
TimeCreated	DateTime	Automatic date of creation
Created by	Int64	WebServicesId of creator.
Userid	Char(32)	Userid chosen by user (unique)
SendEmail	Int	ForAdminOnly=0, OnBatchFailure=1, OnBatchCompletion=2,

Job Tracking

All jobs (Quick and batch). should be recorded in the Jobs Table. As for all other services mentioned here this needs to have a WebServices interface. As such it could also be used to keep track of CondorJobs.

For a job we need to store several pieces of data including:

- Task Name (provided by submitter)
- Time of submission
- Time of start of job
- Time of end of job
- Input query
- The modified query (if modified)

The complete suggested schema is shown below.

User screens

On the job screen we should show the duration of the job if it is finished

We would need to be able to give the user some status like whether his job(s) is executing or where it is in the queue.

We should allow the option of emailing the user when the job is finished (but not make it mandatory).

We should allow the user to see status of multiple batch jobs and cancel them if he so wishes (hence password).

We should allow the user to clear finished jobs from the screen. i.e. (select * from jobs where status = finished).

When submitting a task we should allow the user to name the task .

We should allow a task to be resubmitted which will create a new job.

WebServices

The following WebServices will be needed (wsid= WebServiceId):

For Cas we would need a simple submission service such as :

- Long SubmitJob (long wsid,string query,string taskname,int estimate, int autocomplete) – this uses wsid to get the userid and creates the job entry sets the TimeSubmit and the information provided. Autocomplete means if the job fails at any given que bump it to the next queue.
- Void CancelJob(long wsid, long jobid) – cancel this job(or at least flag for cancellation).
- Int StatusJob(long wsid, long jobid) – status of this job

- Job[] FindJob(long wsid, string predicate) – will need to be able to list jobs in some general maner. E.g predicate may be “jobid = 1233”. We would restrict this to returning only jobs belonging to this wsid.

These are Job Control web services for internal use.

- Long CreateJob (long wsid,string query,string modifiedquery,string taskname,int estimate, bool autocomplete) – this uses wsid to get the userid and creates the job entry sets the TimeSubmit and the information provided. Autocomplete means if the job fails at any given que bump it to the next queue.
- StartJob(long wsid,long jobid,string hosted,string ouputloc) – update the job entry and set the TimeStart, changes the status to started and puts in the HostIp.
- FailJob(long wsid,long jobid,string message) – updates status to failed , TimeEnd and message.
- CancelJob(long wsid,long jobid,string message) – updates status to canceled , TimeEnd and message.
- FinishedJob(long wsid,long jobid,string message,long rows) – update the status to finished, update message and rows sets TimeEnd.
- Job[] FindJob(long wsid, string predicate) – will need to be able to list jobs in some general maner. E.g. for Condor predicate may be “TaskName like ‘PhotoZ’ and Status = 0” while for the Cas Medium Access we may have “Estimate > 1.5 and Estimate < 60 and Status =0”. Although it may be more efficient to parameterize these queries.
- Long resubmitJob(long wsid, long jobid, int estimate) – if the job is finished or failed (status > 1) submit a new job with the same task name etc., a new job id will be assigned . Use the new estimate
- Long resubmitJob(long wsid,long jobid) – as above but use the next highest estimate i.e. bump the job to the next queue.
- Long resubmitTask(long wsid, string taskname, int estimate) – as for resubmit job but search for the last task from this wsid with this taskname.

Jobs Table Schema

JobId	Int64	Unique, auto generated Identifier
WebServicesId	Int64	Foreign key to users atble
TaskName	VarChar(80)	A name for the task, supplied by the submitter
OutputLoc	VarChar(200)	Mydb.table for CAS, file name for condor
Status	Int(1)	(ready=0, started=1, pendingCancel=2, canceled=3,failed=4, finished=5)
HostIp	char(15)	Machine which actually did the job
TimeSubmit	DateTime	AutoGenerated time when this record was created.
TimeStart	DateTime	Time the job started
TimeEnd	DateTime	Time thejob finished
Rows	Int64	Number of rows returned
Message	Varchar(200)	Error or status message.
Query	Varchar(2000)	The original query
ModifiedQuery	Varchar(2000)	Modified query if it was modified by the system.
Estimate	Int32	Estimated num minutes for job (will be used by CAS

		to make the multiple queues)
AutoComplete	int	If the job fails in the queue it was next to send it to the next priority queue until it finishes or fails.
SendEmail	Int	ForAdminOnly=0, OnBatchFailure=1, OnBatchCompletion=2,

Time Estimates/ Multiple Queues

We shall allow a time estimate to be submitted with a job. This will be used to decide which Queue a job will go to. Initially we may have 4 queues namely:

- 1.5 minute (quick access queue)
- 20 minute (medium access queue)
- 60 minute (long access queue)
- >60 minute (overnight queue)

If I submit a job with 15 minute estimate it would go to the 20 minute queue. If a Job does not finish within the max time of the queue it goes to it will be failed and it will need to be resubmitted with a higher time estimate. In the case of registered users we should provide an extra service, which will try the first specified queue and if it fails try the next and then the next until the task finishes. This is a case of automatically resubmitting the task with higher estimates. This must be optional, as some users will want their job to fail if it takes too long in the quick queue. In this way all jobs may be submitted to the same interface with estimates of 1.5 minutes or less being treated as Quick Access.

For the batch system then we can run a couple of queues for Long Jobs and Medium jobs hopefully giving good return times on medium jobs. Hence the way the queues are handled may vary in each case.

Temp Databases (MyDb)

Each registered user should have a Temporary Database created upon registration. We must allow for the temp databases to be created on multiple machines. This database shall have an initial size of 10MB and a max size of 100MB (needs to be checked – how many rows can we put in 100MB).

The database name shall be stored in the users Users Record.

In a query the user should always refer to MyDB to reference his DB. The query processor will have to swizzle this into the correct name by looking it up in the users table.

In the case of MyDb we need to allow fairly unrestricted access e.g:

- Insert
- Drop
- Create
- Select into
- Functions
- Stored Procedures

e.g. Drop mydb.table; should be allowed while drop anything else should not.

Basically the user running these queries needs DB Owner privileges on all MyDbs and DbReader on the SDSS Dbs. Personally I think this is safe enough but we will try to parse the SQL for extra security.

We need maintenance screens to view, delete and modify limits on databases. Initially we can use Enterprise manager for this.

Initially we should allow users to clean up after themselves. Ultimately we should write a system which checks usage of DBs and deletes older ones, which are not used (mailing the user first of course) as space is required. Least Frequently used policy in effect.

We should limit the number of MyDBs to 100 initially – this implies however limiting the number of registered users to 100 also.

Group Access

Any user should be allowed to create a group and invite other users to the group.

The invited user would have to accept to be a member of the group.

Users in a group could then publish a table to his/her groups there by giving Read access to the table to all members of the groups to which he/she belongs.

Users would then access these group tables with a group_ prefix and the userid e.g.

Select * from GROUP_USERID.TNAME

This could be extended with permissions later if it is seen as necessary.

User Screens

Need screen for creating a group and selecting users to invite. (option of sending invite mail)

Need a screen for removing users from a group .

Need a screen or service (could just have a link in an email) to accept membership of a group.

WebServices

We need an anonymous user who can create accounts.

The following WebServices will be needed (wsid= WebServicesId, User is a Structure containing the user information):

- long createGroup(long wsid,string GroupName,string, Description,string Password) – create group return the id.
- void addUsersToGroup(int64 webservicesid , string[] userid,long gid) – userid could be the email or userid. Gig is the id of the group to add the users to. Only of the webservicesid matches that of the group owner.
- void removeUsersFromGroup(int64 webservicesid, string[] userid,long gid) – userid could be the email or userid. Gig is the id of the group to remove the users from. Does not actually remove them just marks them removed with the status flag. Only of the webservicesid matches that of the group owner.
- string acceptMembership(int64 webservicesid, long gid) – change status of member row to accepted if such a row exists and the status was invited. Return a nice message.
- Void publishTable(int64 wsid, string tableName) – publish the users table to the group – entry in GroupTables

- UnPublishTable(int64 wsid, string tableName)

Group Table Schema

Name	Varchar(80)	GroupName
GID	long	Id automatically assigned by system
Owner	Int64	WebServicesid of creator (integrity constraint to user)
Description	VarChar(200)	Some description provided by creator

GroupMembers Table Schema

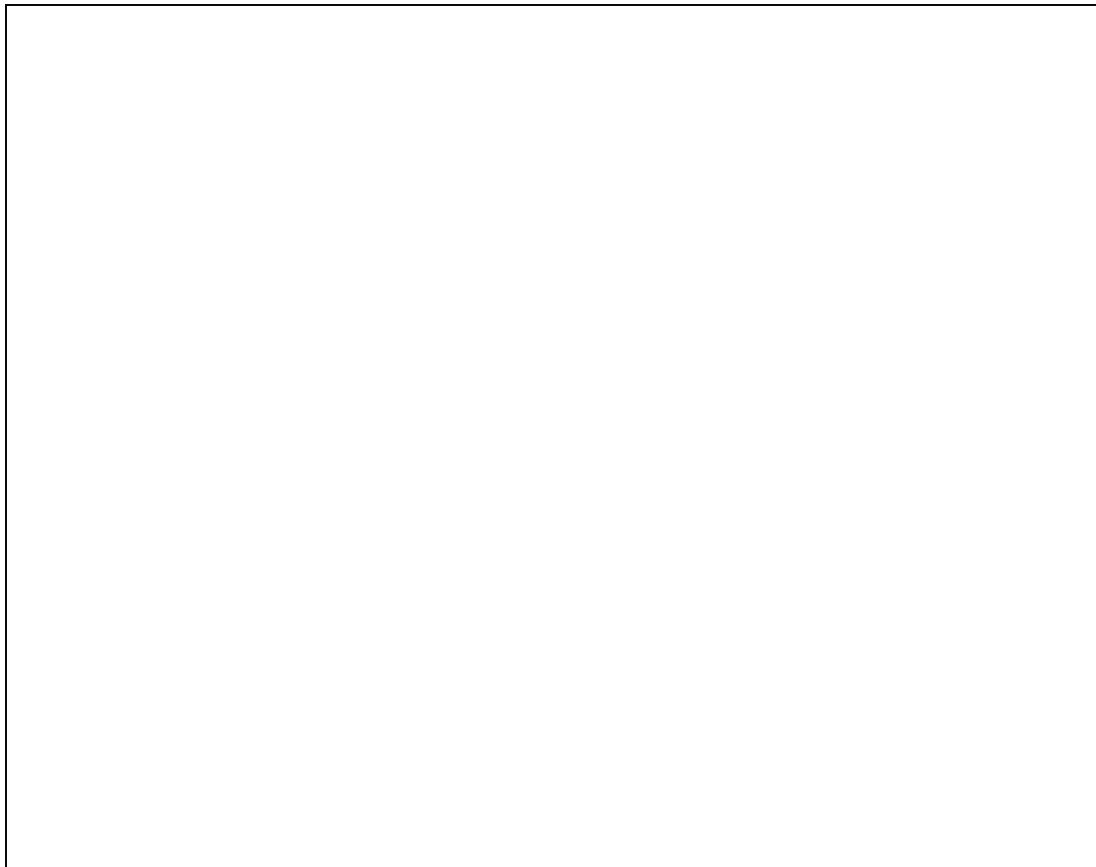
Name	Varchar(80)	GroupName
GID	long	Id automatically assigned by system
WebServicesId	Int64	Members WebServicesId
Status	Int	0=removed,1=invited,2=accepted

GroupTables Table Schema

wsid	long	Id of table owner
tableName	VarChar(80)	Name of published table

Job Execution

We foresee quick access being synchronous and longer jobs being performed asynchronously. This is presented in Nolan's diagram below. And expanded in the following sections.



Quick Access

If the user is logged in allow use of MyDB.

We should try the query for 90 seconds with 1K row limit.

If the query fails to complete within the limitations we should forward the user to the batch page.

We consider that CasAccess

1. do the query rewrite
2. creates an entry in Jobs for the Query
3. finds a machine to execute the job on
4. executes it for the limit
5. returns the result if it finishes
6. resubmits to the next priority queue if it fails and it is flagged to do so.

This is basically what is currently in place. What needs to be decided is which queries are allowed to run in this mode. Currently of course 1 hour queries are accepted.

Batch Access

We probably need to have a dedicated machine for batch access so it will not interfere with the quick access.

Again in batch mode we will probably not want to run more than a couple of queries simultaneously to avoid high overheads of Context switch. So we will need some minimum scheduler/load balancer. This may probably be achieved using .NET's pooling but we need to look into it.

All batch queries should be of the form "Select into mydb.X" hence batch output would always be to a designated table in MyDb. Hence cleanup is a problem for the user.

A user may see partial result by submitting a QuickAccess job like "select count(*) mydb.answer with (noLock)"

All batch jobs will be submitted through CasAccess which will

1. do the query rewrite
2. create an entry in Jobs for the Query (and the re written query).
3. returns the job id.

Batch Execution

We will need a windows service, probably per batch queue, which will query the Jobs table to find jobs to execute and execute them on an appropriate machine.

Servers Table Schema

We need a server config table so we can find an appropriate database to run a query on. This would allow us to configure a cluster of servers and run queues on

ConnectionString	Varchar(200)	SQL Connection string for the server possible with Pooling set to lower than 100.
Queue	int	length of Queue allowed i.e. 1.5, 60
DataBase	Varchar(200)	Which Db is it i.e. DR1, BEST, DR2

MyDb Table Extract

We would then need to provide a MyDbTableExtract service which would return a table from MyDB as VoTable/Fits etc. In this way we could isolate I/O across the net to the user from I/O on the Database. We would not need to worry about cleaning up temp space as the user would have to do that in their own MyDb. We do not have to be so carefull about Temp Files which are created as theymay always be recreated from the MYDB.

WebServices

This implies a webservises like:

- Void ExtractFitsAsDime(long wsid, string tableName) – attach fits file as dime attachment to SOAP Message.
- VOTable ExtractVOTable(long wsid, string tableName);
- CASStream ExtractStream(long wsid, string tableName);
- DataSet ExtractDataSet(long wsid, string tableName);

The above are good for small tables we also need some utilities and a delayed way of creating larger files.

- String[] listOutputTypes() – list which are valid output file types
- String[] listMydbTables(long wsid) – list tables in the users mydb.
- Long RequestFileOutput(long wsid, string tablename, string outputtype) request generation of the file for the table in mydab. The file url will be put in the job record upon completion. This returns instantly with the jobid.

CAS Access library & load balance

The library written by Nolan and Tamas would be underneath these layers.

CAS Access needs to be modified to accept the WebServicesID which it can use to lookup the users MyDb if it needs to do so.

CAS Access need to substitute all oocurences of MyDb with the users actual db name which must be looked up from the Database.

CAS Access needs to allow certain keywords in the case of MyDb (as in Temp Database)

CAS Access should be modified to do simple load balancing for the QuickAccess queries. This can be done by having multiple connection strings configured (in the Servers Table) and doing a rough estimate on the load of each machine perhaps using Perf monitor. Simply using the Pooling parameters may be sufficient for this. All jobs should be logged in the jobs tables not just batch jobs. Hence CAS Access would need to log the start and end of each job and perhaps the machine it was sent to. This can be done used

The Ferris Wheel For All Sky Queries

This is a longer term goal.

We have spoken for some time about batching queries together to run in a sort of Ferris wheel. The notion here being to allow batches of queries to access the data sequentially using some block/bucket as the unit of access.

This could be taken a step further for distribution if we distributed the sky over several nodes and allow parallelisation of the query – some buckets could be on different machines. This could be grid implementation of course using services under Globus – or simply using WebServices (probably easier).

Split The Query

Regardless of distribution the first task in this system would be to rewrite the query as a set of queries restricted to some bucket..

Is it good enough to run several of these queries one after the other or do we need to combine them in some way ? The latter being very difficult but probably what we need to do for best performance.

The way this was approached for GAIA was to make the Algorithms Data Driven, a process then sequentially accessed the data and passed it to all waiting algorithms.

I guess we need something like that here.

Database Organisation

The Database would need to be organized such that data in a bucket was contiguous and lead to a sequential access.

Execution

Some execution framework will be required although it should not be difficult.. All Queries will be split and appropriate parts queued for appropriate buckets. The Processor will go through all Buckets, Sequentially Access the data and apply the queries. SQLServer allows piggy back queries to be made in a similar access pattern.

