XCG Accounting

Author: Vanamala Venkataswamy
Date: 05-14-2012

Prerequisites:

1. Create a central database, we use MySQL (installed and maintained by CS department)
2. Create the following tables in the Database table xcgaccoutingrecords, table xcgbescontainers, table xcgcredentials, table xcgareccredmap and table xcgcommandlines (Information on the table columns is found in Appendix A)
3. Link the database into grid name space
4. Two usernames to access this central DB, one has write privileges and the other has read-only privileges. Username with write privileges will be used to run the accounting tool and username with read-only privileges will be used to get web GUI statistics.
5. Web server Eg. webserver provided by CS department to be able to talk to Central database

There are several pieces to collecting and processing job accounting data for the XCG.

Background:

Raw job accounting data is kept on the home container of the BES that runs the job. It will stay there forever, unless someone signals that the container can delete accounting information up to a specified accounting record ID (we call this "committing" the records). We use a grid command line tool named "*accounting"* to collect accounting records, put the collected data into a database (currently this is the department's MySQL database), and to commit all records collected on the container so that the container can delete them from its local database. In order to support our on-demand online accounting graphs, the raw accounting data collected from the containers must be processed into a format that the graphing pages can use. So, overall, the collection process has 2 parts:

a) collect raw data from containers and store in raw accounting database tables

b) process the data to update tables supporting accounting graphs

Accounting Database:

The permanent home for XCG accounting data is a set of tables in the "vcgr" database on CS department's MySQL database server. We maintain 2 accounts for this database:

vcgr_mmm2a (password: "****") which has full rights and must be used for collecting records and doing any kind of updates.

xcg_ro (password: "****") which is a read only account. This is used for the web accounting graphs web site and should be the only one given out more publicly.

The raw accounting data is placed into 5 related tables by the accounting collection tool. In order to make the data easier to process for usage graphs, a stored procedure named procPopulateXCGJobInfo crunches all of the data in the 5 raw accounting data tables and stores them in 2 derived "tmp" tables for use by the accounting graph php pages on the web site.

The vcgr database currently contains 15 tables, only about half related to the new way of doing accounting.

Accounting related tables set by accounting collection tool:

- xcgaccoutingrecords – each row holds the raw accounting information for a single XCG job.
- xcgbescontainers – each row holds information about a BES container that has had

accounting data collected for it. The key for the BES record in this table is used to match accounting records in the xcgaccoutingrecords table to the BES container it ran on. Records in this table are matched during the accounting record collection process based on the BES's EPI, so re-rolling a BES will cause a new BES entry to appear in this table. The besmachinename field in this table is populated by the machine's IP address when it is first created and this is used by our accounting graphs as the name of the BES. However, the besmachinename field can be manually updated to put in a more human friendly name and to tie together records from two BES instances that have served the same purpose. This is something I do periodically to make the usage graphs more readable and informative.

- xcgcredentials – contains a list of every individual credential used by any job. Multiple jobs using the same credential will share an entry in this table. Since the relationship between jobs (xcgaccountingrecords) and credentials (xcgcredentials) is many-to-many, the xcgareccredmap table provides the relationship mapping between them. The credentialtype field is set to NULL for new entries, but can be set to values "Client", "Service", "User" or "Group" manually. I occasionally manually edit this field to set the proper designation for new entries.
- xcgareccredmap – associative table between xcgaccountingrecords and xcgcredentials.
- xcgcommandlines – contains each portion of a job's commandline – one entry for argument (including the executable argument). The accounting tool did not work properly at first and only recorded the last argument for jobs. This was fixed sometime after initial rollout of the new accounting infrastructure.

Denormalized accounting data for usage graphs:

In order to easily support reasonably fast creation of a wide range of usage graphs, we use a stored procedure to create a couple of tables to store pre-processed denormalized information about jobs.

- tmpXCGJobInfo – this table contains 1 row per job with pretty much everything in it that we can run a report against. This includes our best guess the job's "owner" in human friendly terms, the bes container's name, various run time information, and information about the day, week, month, and year of execution.
- tmpXCGUsers – this table is an intermediate table used by the stored procedure that creates the tmpXCGJobInfo table. It really can be deleted as soon as the stored procedure finishes – not sure why it isn't…

The denormalization process

The denormalization process deletes and re-creates the tmpXCGJobInfo and tmpXCGUsers tables from data in the raw accounting tables. Denormalization is done by running the procPopulateXCGJobInfo stored procedure.

Notes:

- Besides denormalizing the data so that there is single row per job, it also tries to figure out which user "owns" the job and stores it's best guess in the username field of the tmpXCGJobInfo table. This field is used by the usage graphs to display/filter who ran a job. The algorithm for doing so is imperfect, but must be understood to properly understand the usage graph behavior. A job's owner is determined as the credential with the lowest cid (credential id) associated with the job that is designated as a "User" (it may also require that the credential has the string X509AuthnPortType in it). It then assumes that the credential is in the format of those we mint for ordinary XCg users and extracts the owner's name from the CN

field of the credential. This process will only work if several conditions are met:
  - 1) The job has at least one credential minted as a normal user by XCG in the usual format.
  - 2) The credential has been manually marked as a "User" credential in the xcgcredentials table
- Users that use a different type of credential (e.g. username/password or other outside credential), were run only by admin (different cred format) or have not had their entry in the xcgcredentials table manually updated to type "User" will be labeled as owned by "unknown".

Process:

1) Collect raw data.
   a. The grid tool "accounting" is used to collect raw accounting data and store it in the CS department database. The tool takes several options/arguments:
      a.i. –collect: tells the tools to do the actual data collection. NOTE: unless you specify the "--no-commit" flag, the --collect flag will commit all accounting records successfully collected from the target container(s).
      a.ii. --no-commit: optional flag to stop tool from committing (i.e. erasing) records from the target container(s).
      a.iii. --recursive: allows you to specify a directory of containers and the tools will recursively go through all entries in directory and collect from each one.
      a.iv. --max-threads=<# threads>: allows the collection to be multi-threaded
      a.v. <source container> | <source container directory>: which container (directory) to collect. Note that the tool
      a.vi. <target database>: connect information for database that will store collected data.
   b. Typical use (as admin – other users will not have permission to collect or commit accounting data from most/all containers):
      b.i. accounting --collect --recursive /containers /accounting/CS-MySQL
      b.ii. Notes:
         b.ii.1. The tool uses grid containers, not BES containers as targets. Even though the accounting records do identify which actual BES container the job was associated with, the tools collect all of the data for all BES's it contains at once.
         b.ii.2. We can use the directory /containers recursively because we try to maintain /containers such that it has all of our useful containers in it and no other extraneous entries. This helps simplify the process significantly.
         b.ii.3. We use the RNS path /accounting/CS-MySQL as the target database. Mark set up this RNS entry with the proper connection information for the vcgr database on the department server to help the collection process. The tool can handle extracting the connection info from the EPR contained by the RNS entry.
         b.ii.4. You will be prompted for the password for the vcgr_mmm2a account on the CS department database server.
         b.ii.5. You will see the tool collecting data from each container in turn. Note exceptions – there are sometimes entries of down containers left in the /containers directory or there are sometimes unexpected problems.

2) Process the data for the online graphs
    a. Log into the department's PHP MySQL web front end. Use a browser and go to . At the login screen, use the vcgr_mmm2a account and password to login.
    b. Go to the vcgr database. Just click on the vcgr database entry on the left side of the screen
    c. Get an SQL window. Click the SQL tab and text area will appear where you can type in SQL statements.
    d. Run the stored procedure procPopulateXCGJobInfo (procudure definition found in Appendix A). In the SQL text area type "call procPopulateXCGJobInfo" and click the Go button. It will run for several minutes and then the screen will go blank when it is done (that's MySQL for you). The procPopulateXCGJobInfo stored procedure will erase the old tmpXCHJobInfo and tmpXCGUsers tables and will repopulate them after crunching whatever data is in the raw accounting tables. GUI statistics can be looked up here http://vcgr.cs.virginia.edu/XCG/2.0/stats/usage_graphs/

To link a Database into Grid name space, use the following command
mint-epr --link=/accounting/XCG-DB jdbc:mysql://mysql.cs.virginia.edu/vcgr?user=vcgr_mmm2a

Appendix A

USERS

Field	Type	Null	Key	Default	Extra
id	bigint(20)	NO	PRI	*NULL*	auto_increment
name	varchar(128)	NO		*NULL*	
title	varchar(128)	YES		*NULL*	
department	varchar(128)	YES		*NULL*	
organization	varchar(128)	YES		*NULL*	
country	varchar(64)	NO		*NULL*	
email	varchar(128)	NO		*NULL*	
username	varchar(64)	NO		*NULL*	
password	varchar(128)	NO		*NULL*	
idppath	varchar(256)	NO		*NULL*	
homedir	varchar(256)	NO		*NULL*	

XCGCREDENTIALS

Field	Type	Null	Key	Default	Extra
cid	bigint(20)	NO	PRI	*NULL*	auto_increment
credential	blob	NO		*NULL*	
credentialtype	varchar(16)	YES		*NULL*	
credentialdesc	varchar(512)	NO		*NULL*	
credentialhash	int(11)	NO		*NULL*	

Field	Type	Null	Key	Default	Extra
timeadded	timestamp	NO		CURRENT_TIMESTAMP	

GENIIJOBLOG

Field	Type	Null	Key	Default	Extra
eventid	bigint(20)	NO	PRI	*NULL*	auto_increment
eventtype	varchar(64)	NO		*NULL*	
identities	varchar(1024)	NO		*NULL*	
eventtime	datetime	NO	MUL	*NULL*	
epi	varchar(256)	YES	MUL	*NULL*	
jobid	varchar(256)	YES		*NULL*	
processtime	bigint(20)	YES		*NULL*	
hostname	varchar(256)	NO	MUL	*NULL*	

GROUPS

Field	Type	Null	Key	Default	Extra
id	bigint(20)	NO	PRI	*NULL*	auto_increment
name	varchar(128)	NO		*NULL*	
description	varchar(512)	YES		*NULL*	
path	varchar(256)	NO		*NULL*	

MEMBERSHIP

Field	Type	Null	Key	Default	Extra
id	bigint(20)	NO	PRI	*NULL*	auto_increment
uid	bigint(20)	NO	MUL	*NULL*	
gid	bigint(20)	NO		*NULL*	

tmpXCGJobInfo

Field	Type	Null	Key	Default	Extra
arid	bigint(20)	NO	PRI	*NULL*	
besaccountingrecordid	bigint(20)	NO		*NULL*	
besid	bigint(20)	NO		*NULL*	
exitcode	int(11)	NO		*NULL*	

Field	Type	Null	Key	Default	Extra
jobuserhrs	decimal(23, 4)	NO		*NULL*	
jobkernelhrs	decimal(23, 4)	NO		*NULL*	
jobwallclockhrs	decimal(23, 4)	NO		*NULL*	
maxrssbytes	bigint(20)	YES		*NULL*	
recordtimestamp	timestamp	NO	MUL	CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP
jobyear	smallint(6)	NO		*NULL*	
jobmonth	smallint(6)	NO		*NULL*	
jobmonthname	char(10)	NO		*NULL*	
jobday	smallint(6)	NO		*NULL*	
jobdayofyear	smallint(6)	NO		*NULL*	
jobmondaydate	date	NO	MUL	*NULL*	
besmachinename	varchar(256)	YES	MUL	*NULL*	
arch	varchar(64)	YES		*NULL*	
os	varchar(64)	YES		*NULL*	
ownercid	bigint(20)	YES		*NULL*	
username	varchar(256)	YES	MUL	*NULL*	
linuxuserhrs	decimal(23, 4)	NO		*NULL*	
linuxkernelhrs	decimal(23, 4)	NO		*NULL*	
linuxwallclockhrs	decimal(23, 4)	NO		*NULL*	
windowsuserhrs	decimal(23, 4)	NO		*NULL*	
windowskernelhrs	decimal(23, 4)	NO		*NULL*	
windowswallclockhrs	decimal(23, 4)	NO		*NULL*	
macosuserhrs	decimal(23, 4)	NO		*NULL*	
macoskernelhrs	decimal(23, 4)	NO		*NULL*	
macoswallclockhrs	decimal(23, 4)	NO		*NULL*	

tmpXCGUsers

Field	Type	Null	Key	Default	Extra
cid	bigint(20)	NO	PRI	*NULL*	
username	varchar(256)	NO		*NULL*	

XCGACCOUNTINGRECORDS

Field	Type	Null	Key	Default	Extra
arid	bigint(20)	NO	PRI	*NULL*	auto_increment
besaccountingrecordid	bigint(20)	NO	MUL	*NULL*	
besid	bigint(20)	NO	MUL	*NULL*	
exitcode	int(11)	NO		*NULL*	
usertimemicrosecs	bigint(20)	NO		*NULL*	
kerneltimemicrosecs	bigint(20)	NO		*NULL*	
wallclocktimemicrosecs	bigint(20)	NO		*NULL*	
maxrssbytes	bigint(20)	YES		*NULL*	
recordtimestamp	timestamp	YES		*NULL*	

XCGARECCREDMAP

Field	Type	Null	Key	Default	Extra
mid	bigint(20)	NO	PRI	*NULL*	auto_increment
cid	bigint(20)	NO	MUL	*NULL*	
arid	bigint(20)	NO		*NULL*	

XCGBESCONTAINERS

Field	Type	Null	Key	Default	Extra
besid	bigint(20)	NO	PRI	*NULL*	auto_increment
besepi	varchar(256)	NO	UNI	*NULL*	
besmachinename	varchar(256)	YES		*NULL*	
arch	varchar(64)	YES		*NULL*	
os	varchar(64)	YES		*NULL*	
timeadded	timestamp	NO		CURRENT_TIMESTAMP	

XCGCOMMANDLINES

Field	Type	Null	Key	Default	Extra
clid	bigint(20)	NO	PRI	*NULL*	auto_increment

Field	Type	Null	Key	Default	Extra
arid	bigint(20)	NO		*NULL*	
elementindex	int(11)	NO		*NULL*	
element	varchar(512)	NO		*NULL*	

ProcPopulateXCGJobInfo routine_definition

```
BEGIN

DROP TABLE IF EXISTS tmpXCGUsers;
DROP TABLE IF EXISTS tmpXCGJobOwnerIds;
DROP TABLE IF EXISTS tmpXCGJobOwners;
DROP TABLE IF EXISTS tmpXCGJobInfo;


CREATE TABLE `tmpXCGUsers` (
 `cid` bigint(20) NOT NULL,
 `username` varchar(256) NOT NULL,
 PRIMARY KEY (`cid`)
) DEFAULT CHARSET=latin1;

DELETE FROM tmpXCGUsers;

INSERT INTO tmpXCGUsers
(
 `cid`,
 `username`
)
SELECT
 `xcgcredentials`.`cid`,
 substring_index(substr(`xcgcredentials`.`credentialdesc`,42),',',1)
FROM
 `xcgcredentials`
WHERE
 ((`xcgcredentials`.`credentialtype` = 'User') AND
  (`xcgcredentials`.`credentialdesc` like '%X509AuthnPortType%'));


CREATE TABLE `tmpXCGJobOwnerIds` (
 `arid` bigint(20) NOT NULL,
 `ownercid` bigint(20) NOT NULL,
 PRIMARY KEY (`arid`)
) DEFAULT CHARSET=latin1;

DELETE FROM tmpXCGJobOwnerIds;

CREATE INDEX `tmpXCGJobOwnerIds_ownercid_Idx`
ON `tmpXCGJobOwnerIds` (`ownercid`);
```

```
INSERT INTO tmpXCGJobOwnerIds
(
 `arid`,
 `ownercid`
)
SELECT
 `ar`.`arid`,
 min(`cred`.`cid`)
FROM
 `xcgaccountingrecords` `ar`,
 `xcgareccredmap` `map`
  LEFT JOIN `xcgcredentials` `cred` ON (((`map`.`cid` = `cred`.`cid`) AND
   (`cred`.`credentialtype` = 'User') AND
   (not((`cred`.`credentialdesc` like '%Admin%')))))
WHERE (`ar`.`arid` = `map`.`arid`)
GROUP BY
 `ar`.`arid`;

CREATE TABLE `tmpXCGJobOwners` (
 `arid` bigint(20) NOT NULL,
 `ownercid` bigint(20) NOT NULL,
 `username` varchar(256) NULL DEFAULT NULL,
 PRIMARY KEY (`arid`)
) DEFAULT CHARSET=latin1;

DELETE FROM tmpXCGJobOwners;

CREATE INDEX `tmpXCGJobOwners_ownercid_Idx`
ON `tmpXCGJobOwners` (`ownercid`);

INSERT INTO tmpXCGJobOwners
(
 `arid`,
 `ownercid`,
 `username`
)
SELECT
 `jo`.`arid`,
 `jo`.`ownercid`,
 `users`.`username`
FROM
 `tmpXCGJobOwnerIds` `jo`,
 `tmpXCGUsers` `users`
WHERE
 (`jo`.`ownercid` = `users`.`cid`);

DROP TABLE IF EXISTS tmpXCGJobOwnerIds;
```

```
DROP TABLE IF EXISTS tmpXCGJobInfo;

CREATE TABLE `tmpXCGJobInfo` (
 `arid` bigint(20) NOT NULL,
 `besaccountingrecordid` bigint(20) NOT NULL,
 `besid` bigint(20) NOT NULL,
 `exitcode` int(11) NOT NULL,
 `jobuserhrs` decimal(23,4) NOT NULL,
 `jobkernelhrs` decimal(23,4) NOT NULL,
 `jobwallclockhrs` decimal(23,4) NOT NULL,
 `maxrssbytes` bigint(20) DEFAULT NULL,
 `recordtimestamp` timestamp NOT NULL,
 `jobyear` smallint NOT NULL,
 `jobmonth` smallint NOT NULL,
 `jobmonthname` char(10) NOT NULL,
 `jobday` smallint NOT NULL,
 `jobdayofyear` smallint NOT NULL,
 `jobmondaydate` date NOT NULL,
 `besmachinename` varchar(256) NULL DEFAULT NULL,
 `arch` varchar(64) NULL DEFAULT NULL,
 `os` varchar(64) NULL DEFAULT NULL,
 `ownercid` bigint(20) NULL DEFAULT NULL,
 `username` varchar(256) NULL DEFAULT NULL,
 `linuxuserhrs` decimal(23,4) NOT NULL,
 `linuxkernelhrs` decimal(23,4) NOT NULL,
 `linuxwallclockhrs` decimal(23,4) NOT NULL,
 `windowsuserhrs` decimal(23,4) NOT NULL,
 `windowskernelhrs` decimal(23,4) NOT NULL,
 `windowswallclockhrs` decimal(23,4) NOT NULL,
 `macosuserhrs` decimal(23,4) NOT NULL,
 `macoskernelhrs` decimal(23,4) NOT NULL,
 `macoswallclockhrs` decimal(23,4) NOT NULL,
 PRIMARY KEY (`arid`)
) DEFAULT CHARSET=latin1;



CREATE INDEX `tmpXCGJobInfo_besmachinename_Idx`
ON `tmpXCGJobInfo` (`besmachinename`);

CREATE INDEX `tmpXCGJobInfo_username_Idx`
ON `tmpXCGJobInfo` (`username`);

CREATE INDEX `tmpXCGJobInfo_recordtimestamp_Idx`
ON `tmpXCGJobInfo` (`recordtimestamp`);

CREATE INDEX `tmpXCGJobInfo_jobmondaydate_Idx`
ON `tmpXCGJobInfo` (`jobmondaydate`);
```

```
INSERT INTO tmpXCGJobInfo
(
 `arid`,
 `besaccountingrecordid`,
 `besid`,
 `exitcode`,
 `jobuserhrs`,
 `jobkernelhrs`,
 `jobwallclockhrs`,
 `maxrssbytes`,
 `recordtimestamp`,
 `jobyear`,
 `jobmonth`,
 `jobmonthname`,
 `jobday`,
 `jobdayofyear`,
 `jobmondaydate`,
 `besmachinename`,
 `arch`,
 `os`,
 `ownercid`,
 `username`,
 `linuxuserhrs`,
 `linuxkernelhrs`,
 `linuxwallclockhrs`,
 `windowsuserhrs`,
 `windowskernelhrs`,
 `windowswallclockhrs`,
 `macosuserhrs`,
 `macoskernelhrs`,
 `macoswallclockhrs`
)
SELECT
 `ar`.`arid`,
 `ar`.`besaccountingrecordid`,
 `ar`.`besid`,
 `ar`.`exitcode`,
 (`ar`.`usertimemicrosecs` / 3600000000),
 (`ar`.`kerneltimemicrosecs` / 3600000000),
 (`ar`.`wallclocktimemicrosecs` / 3600000000),
 `ar`.`maxrssbytes`,
 `ar`.`recordtimestamp`,
 year(`ar`.`recordtimestamp`),
 month(`ar`.`recordtimestamp`),
 monthname(`ar`.`recordtimestamp`),
 dayofmonth(`ar`.`recordtimestamp`),
 dayofyear(`ar`.`recordtimestamp`),
 cast((`ar`.`recordtimestamp` - interval weekday(`ar`.`recordtimestamp`) day) as date),
 `bes`.`besmachinename`,
```

```
 `bes`.`arch`,
 `bes`.`os`,
 `owners`.`ownercid`,
 `owners`.`username`,
 IF(bes.os = 'LINUX', (`ar`.`usertimemicrosecs` / 3600000000), 0),
 IF(bes.os = 'LINUX', (`ar`.`kerneltimemicrosecs` / 3600000000), 0),
 IF(bes.os = 'LINUX', (`ar`.`wallclocktimemicrosecs` / 3600000000), 0),
 IF(bes.os = 'Windows_XP', (`ar`.`usertimemicrosecs` / 3600000000), 0),
 IF(bes.os = 'Windows_XP', (`ar`.`kerneltimemicrosecs` / 3600000000), 0),
 IF(bes.os = 'Windows_XP', (`ar`.`wallclocktimemicrosecs` / 3600000000), 0),
 IF(bes.os = 'MACOS', (`ar`.`usertimemicrosecs` / 3600000000), 0),
 IF(bes.os = 'MACOS', (`ar`.`kerneltimemicrosecs` / 3600000000), 0),
 IF(bes.os = 'MACOS', (`ar`.`wallclocktimemicrosecs` / 3600000000), 0)
FROM
   ((`xcgaccountingrecords` `ar`
    LEFT JOIN `xcgbescontainers` `bes` ON((`bes`.`besid` = `ar`.`besid`)))
    LEFT JOIN `tmpXCGJobOwners` `owners` on((`owners`.`arid` = `ar`.`arid`)));

DROP TABLE IF EXISTS tmpXCGJobOwners;

END
```