ISTITUTO NAZIONALE DI FISICA NUCLEARE

CNAF

INFN-14-02/CNAF
15th January 2014

ACCOUNTING DATA RECOVERY. A CASE REPORT FROM INFN-T1

Stefano Dal Pra[1]

1) *INFN-CNAF, Viale Berti Pichat 6/2, I-40126 Bologna, Italy*

Abstract

Starting from summer 2013, the amount of computational activity of the INFN-T1 centre reported by the official accounting web portal of the EGI community accounting.egi.eu, was found to be much lower than the real value. Deep investigation on the accounting system pointed out a number of subtle concurrent causes, whose effects dated back from May and were responsible for a loss of collected data records over a period of about 130 days. The ordinary recovery method would have required about one hundred days. A different solution had thus to be designed and implemented. Applying it on the involved set of raw log files (records, for an average production rate of jobs/day) required less than 4 hours to reconstruct the Grid accounting records. The propagation of these records through the usual dataflow up to the EGI portal was then a matter of a few days. This report describes the work done at INFN–T1 to achieve the aforementioned result. The procedure was then adopted to solve a similar problem affecting another site, INFN–PISA. This solution suggests a possible alternative accounting model and provided us with a deep insight on the most subtle aspects of this delicate subject.

Commissione Calcolo Reti

CCR-48/2014/P

Published by ***SIDS–Pubblicazioni***
Laboratori Nazionali di Frascati

1 INTRODUCTION

The Tier–1 at INFN–CNAF in Bologna is by far the largest data centre of INFN, partecipating as a Grid site to the WLCG infrastructure. The computing resources at the INFN–T1 have grown quite rapidly in the past few years, raising from 3000 cores in 2009, to 13000 cores (fall 2011), up to the current 17000 cores (summer 2013).

Two main components play a key role to make a Grid site out of a set of computing nodes. These are the *Resource Manager*, i.e. the batch system who dispatches user's jobs to the most appropriate computing resource, and the *Computing Elements*, providing an interface on top of the batch system as an access points for external users.

The log files produced by these two systems are the raw data source analyzed by the accounting system to produce its records. Each participating Grid site in the EGI community delivers its accounting data to the accounting.egi.eu portal, where they are made publicly available in aggregated form. This is the official accounting repository and accuracy in reported data matters since it is used as the referral one by site managers as also by members of the HEP community.

INFN adopted DGAS [1] as the official accounting tool for its Grid sites one decade ago. One of the requirements for such a system is the ability to be robust against temporary failures on any of the involved components, so that any possibly missing piece of data would be recovered or added at a later time. The administrator may also reconfigure DGAS to re-read anything starting from a given time in the past.

Beginning with May 2013 a number of problems affected the accounting at INFN–T1. As a consequence the activity reported for INFN–T1 by the EGI accounting portal was patently wrong.

What follows is a report on the analysis of the problem and its causes, and what was done to solve it. Finally, the work implemented will be discussed as a possible alternative accounting model.

2 THE DGAS ACCOUNTING LAYOUT AT INFN-T1

In this section a functional description of the accounting system is described, with particular focus on the components adopted in the centre and their interactions.

2.1 The accounting

IBM/Platform LSF is the Resource Manager adopted at INFN–T1. It currently dispatches computational tasks (jobs) to a farm of about 1400 nodes running nearly 18000 concurrent jobs. INFN holds an institution–wide usage license for LSF which is adopted on a number of INFN Grid sites.

The goal of the accounting system is to keep track of the computing resources usage

made by individual users or user communities. There is an important distinction between *local* and *Grid* users.

- Local users are directly known and authorized by the site's staff; they can usually perform direct job submission from a dedicated host in the farm, the *user interface*, to the batch system, where they are recognized and authorized basing on the user name and group of their local account. A local user only performs direct job submission to the batch system.
- Grid users may only use strong authentication through X509 certificates and submission request on their behalf are handled and authorized by one of the Computing Elements in the site. Upon succesful authentication, the CE then submits the job to the batch system, as a local submitter, and take care of any other subsequent passage on behalf of the original user. A grid user cannot perform direct job submission.

Strictly speaking, *Grid accounting* refers to only activities submitted by Grid users, and a *grid record* consists of two distinct and specific subsets of data pieced together, one from the Resource Manager, and the other from the CE. Here follows a short description, specialized for the LSF batch system as configured at INFN–T1.

Batch system

It provides a full accounting local report about *completed jobs*: job id, job name, user name, execution host, CPU time, start time, end time, among many others.

LSF logs its accounting data to a file in a shared directory. In particular, the LSF log directory is accessible from any CE. Furthermore, log files are rotated daily at midnight. The active log file name is lsb.acct and logs for the previous day are in the lsb.acct.1 file.

Computing Element

The *blah* module in the CE takes care to log a report for each *submitted job* by the CE to the batch system: submission time, credentials of the submitting user, derived from its X509 certificate, Virtual Organization membership, *job id* assigned by the Local Resource Manager (*lrms_id*, according to Grid terminology) among others. These files are stored in a local directory, thus they only are accessible from the CE itself, a new log file is created daily at midnight with the name reporting its creation date as a suffix, such as: blah.log-20130531.

In the following we refer to batch system logs as *lsb logs* or *lsf logs* and to logs from the CE as *blah logs* or *grid logs*.

2.2 The DGAS layout

We sketch here a short functional description of DGAS as deployed at INFN–T1.

2.2.1 Functional DGAS workflow

1. The original log files produced by LSF are split into three distinct parts on three distinct directories. This is because our batch system also manages the activities of two more Grid sites working with part of the same resources and DGAS assumes that one batch system only works for one Grid site.
2. for every finished job, the report produced by the batch system is parsed and an accounting record is created. This is assumed to be the record for a *local job*. This is done by the DGAS component *urcollector*.
3. Matches between records in blah logs and in lsb logs are searched. When a match is found, a record for a *grid job* is created. This is also done by an urcollector instance, running on each CE.
4. The records are locally enqueued in a SQLite buffer for deliver to a local Home Location Register (HLR) which is a separate host with a couple of MySQL databases.
5. When reaching the HLR, a grid record overwrites the local one, if it is present. By design, the opposite does not happen, thus it is possible to have duplicated records for the same job in the HLR.
6. Records collected in the HLR are delivered to the so called *second level* HLR, which is hosted by INFN-TORINO, and is the accounting repository for the Grid sites in the Italian Grid.
7. The second level HLR periodically checks for duplicated records (those possibly present in the lower level ones) and periodically delivers aggregated accounting data to the EGI portal, where they'll finally be available for reporting.

A few things are worth noticing here.

- The same lsb file is independently parsed by each CE, each one looking for matches with the records in its blah log file. For each row in the currently parsed lsb.acct.<n> file, up to three blah file are scanned for a match. Given the intrinsic quadratic nature of the method, and the quite rapid growth of the activity of the centre, this method looses performances with high production rates.
- Each record delivery happens after the instantiation of a Grid Security session (who provides X509 based strong authentication) which gets then closed. This is a costly operation; opening the secure connection absorbs data from a somewhat precious and scarce resource, /dev/random, also needed by other middleware modules running in

the CE. This further increases the needed time for the connection to open, the system load, the CE reactivity. Again, having a high volume of activity represents an issue and may occasionally interfere with the performances of the CE during peak load times.

- The lsb.acct.<n> files are independently read once per CE.

2.2.2 Implementation

Each CE runs a *sensor* to search for matches between records in blah logs and the ones from lsb logs. When a match is found, the information is enqueued in a sqLite database table. A second component reads records from the queue and transmit them one at a time to the HLR through a secure authenticated Grid Security Infrastructure connection.

2.3 DGAS Layout at INFN-T1

The main elements of interest for the DGAS system are described below:

- Three logical grid sites working with the physical farm and its batch system:
    - INFN–T1, with 7 CEs
    - INFN–CNAF–LHCB, with 2 Ces
    - INFN–BOLOGNA—T3, with 2 CEs
- DGAS Sensors are deployed on each Computing Element
- One on-site HLR
    - Holds a 84GB MySQL database (data + indexes) from December 2011
    - more than 55 x 10^6 accounting records
- Batch log files:
    - written by the LSF Master on a shared directory, readable from the CEs
    - LSF 7.06, lsb.acct.<n>
    - From a logical point of view, three grid sites
- CREAM/BLAH: blahp logfiles, local to the CE where the component runs.

It is important to stress a relevant fact: DGAS assumes that each site runs its own batch system instance. This is not our case; for this reason a cronjob nigthly splits the original lsb.acct.1 file into three parts, one per site, and stores them onto three distinct directories.

3 THE PROBLEM

A chronological report on the issues experienced, their investigation and the actions taken, up to the complete understanding of the problem is reported. Gory details are skipped when inessential. Rather, a few facts which are to be known before detailing the recovery process are explained.

3.1 Chronology

3.1.1 July 2013

The June monthly report from EGI shows very low production rate at INFN-T1. Apparently all DGAS components are running smoothly, according to the usual checks and log file inspection. Nothing obviously wrong can be seen.

After a few days it becomes clear from log files inspection that the urcollector component only iterates among a limited set of lsb logs. A plot of the timestamps (on a single CE) of the considered jobs shows a clearly periodic function, with the end of May as its maximum, suddenly falling back a number of days.

Upon ticket notification, the DGAS support readily provides a patch for urcollector and days after the fallback are observed no more.

3.1.2 August 2013

A cause of trouble is discovered: on May 16 the script that once a day splits the original lsb.acct.1 file is moved to another host. The new host has a difference in a network configuration parameter (default MTU value, without jumbo frame support). This occasionaly may cause problems when writing to a networked filesystem and the split files may contain malformed lines. We suspect this to be the reason for the "periodic syndrome": apparently, when reading a malformed log line the urcollector reacts by restarting the parsing from a number of days before. To fix things, after properly setting the MTU, the split lsb.acct.<n> are re–generated from the original ones.

3.1.3 September 2013

Again, the next monthly report has too low values. This time, there is no "periodic syndrome". After further analysis, a new explanation is found: DGAS orders the lsb.acct.<n> files according to their modify–time, not by name. However the re–generated files all have the same *mtime*, thus they are analyzed by the urcollector in an almost casual order. To fix things, the *mtime* of the files is changed to a value consistent with the file name.

3.1.4 Deep health check

Efforts were also spent in order to be able to analyze contents of the hlr database and check for consistency its contents. Accounting records are stored in the jobTransSummary table (JTS for short). A first anomalous fact was noticed: there were a quite significative number of duplicated records. *Table1* quantifies a monthly amount. Records referring to the same job (i.e. having the same jobid and the same enddate) were represented by two different records: one as a grid job and one as a local job.

month	VO	Jobs	CPUDays
2013-07	atlas	29390	2968.37
2013-07	alice	6868	1898.45
2013-07	cms	2382	610.26
2013-07	lhcb	6988	1354.09
2013-07	ops	466	0.01

Table 1. Duplicated jobs count during 2013 July. Two jobs are duplicated when they have the same jobid and the same end time.

Altough this seems to be a flaw, DGAS accepts this by design. Duplicate records are identified and removed at a later stage, in the second level HLR.

3.1.5 A smoking gun

With the intent of directly comparing contents in the JTS with the accounting logs from the batch system, a python script is written to extract accounting records from the lsb.acct.<n> files and populate a table in a PostgreSQL database with them. The parsing of the lsb.acct.<n> files is made quite easy thanks to a reliable python library, the python-lsf-collection (see next section).

A first exact *quantitative* measure of missing accounting records becomes possible. By comparing the number of jobs per day submitted by a single CE, it turns out that there are frequent gaps: roughly one day over three counts almost zero jobs.

CE	date	Jobs (JTS)	Jobs (lsb)
ce04-lcg	2013-08-06	8763	7914
ce04-lcg	2013-08-07	8968	8399
ce04-lcg	2013-08-08	3	12936
ce04-lcg	2013-08-09	6440	5871
ce04-lcg	2013-08-10	9044	8580
ce04-lcg	2013-08-11	8377	7875
ce04-lcg	2013-08-12	9	9810
ce04-lcg	2013-08-13	8494	7974
ce04-lcg	2013-08-14	10213	9246
ce04-lcg	2013-08-15	6	9025

Table 2. Missing Jobs.

The reason is found that the patched urcollector happen to skip a lsb file, once in a while.

This ends the sequence of accounting issues. Now comes the time to recover the missing data.

At this point, the canonical solution would be to rollback the patch and restart the sensors from May. However the needed recover time (two days of accounting processed per day) would be a too long one to wait. An alternative solution must be evaluated and possibly exploited.

4 DATA RECOVERY

The most delicate task is the data extraction from the raw accounting logfiles produced by LSF. This is fortunately made easy and reliable by the python-lsf-collection library, wich comes with an accounting module and suitable parsing methods. One of them can read a raw log line and return a python dictionary.

4.1 Accounting data retrieval

4.1.1 Batch system logs

Using the aforementioned python library, a script was written to read a given lsb.acct.*n* file, extract the fields of interest from each record in it and store it in a job table of a PostgreSQL database. The import of the content of 130 files, corresponding to 130 days of activity and roughly 13M records required three hours.

This was done working on a CE dedicated for testing activities. The PostgreSQL

database is the one provided by default with the RHEL 6.x distribution, without any tuning of sort. It's worth to mention that adopting simple and common tuning practices, the import time could be reduced to 1.5h.

4.1.2 Blah logs

All the available blahp-log-<*date*> files for the same time period were also imported into a blahjob table in the same database. To do this another python script was written. The file parsing was directly done by the script, being the content much easier to parse in this case. These log files were made locally accessible by mounting the remote log directories through sshfs, for a total of 7 distinct mount points.

Import time was a matter of 1 to 3 seconds per file, one file per CE, 10000 to 15000 records each.

After a worth the database was populated with the needed *batch system*–side data, in the job table and the *grid*–side ones, in the blahjob table. Obtaining the grid records is now fairly easy by mean of SQL join and subsidiary steps:

4.1.3 Obtaining grid records

retrieve the indexes of the matching rows from job and blahjob and insert them into a job2blah table:

```
INSERT INTO job2blah (
  SELECT j.id, bj.id FROM job AS j, blahjob AS bj
WHERE
  j.jobname = bj.clientid AND
  j.jobid = bj.lrmsid AND
(date(bj.submtime)=date(to_timestamp(j.submittimeepoch)) OR j.submittimeepoch = 0));
INSERT 0 8227850
Time: 124390.153 ms
```

1. Use the indexes above to build the grid records and insert them into the gridjob table:

```
INSERT INTO gridjob (
      SELECT j.*, bj.submtime, bj.ceid,
          bj.userdn, bj.userfqan
      FROM job j, blahjob bj, job2blah j2b
```

```
        WHERE
                j.id = j2b.job_id AND bj.id = j2b.blah_id
    );
    INSERT 0 8227850
    Time: 232462.443 ms
```

2. Now the records used from job and blahjob are to be deleted, and the service table job2blah can be truncated:

```
DELETE FROM job WHERE id IN (select job_id from job2blah);
DELETE FROM blahjob WHERE id IN (select blah_id from job2blah);
TRUNCATE job2blah;
```

The three steps above are performed as a single transaction, to ensure data consistency. Steps 1 and 2 together took 6 minutes.

The deletions in step 3 are the most time consuming if executed as reported above, and they roughly takes one hour. The following form however only takes 566 seconds:

```
BEGIN;
-- defer consistency checks at the transaction end.
SET CONSTRAINTS ALL DEFERRED;
CREATE TEMPORARY TABLE resjob AS (
        SELECT * FROM job WHERE id IN
        (SELECT id FROM job
EXCEPT
        SELECT job_id FROM job2blah)
) ON COMMIT DROP;
CREATE TEMPORARY TABLE resblah AS (
        SELECT * FROM blahjob WHERE id IN
        (SELECT id FROM blahjob
        EXCEPT
        SELECT blah_id FROM job2blah)
) ON COMMIT DROP;
```

```
TRUNCATE job2blah;
TRUNCATE job;
TRUNCATE blahjob;
INSERT INTO job (SELECT * FROM resjob);
INSERT INTO blahjob (SELECT * FROM resblah);
COMMIT;
```

After the deletion phase the residual rows in the two tables, not collected to take part of a gridjob are:

blahjob

maybe related to a running job, and as such not yet accounted in a lsb.acct file or the matching record is missing in the job table. In the former case the record would match at a later update, i.e. when feeding the database with newer accounting logfiles. The latter case is a pathological one and doesn't happen in normal circumstances. It can be ignored.

job

This maybe a *local job* and as such it has to be treated differently. Usually local jobs are submitted from dedicated hosts of the centre and they are recognized that way. If the submission host is however a CE, the record is most probably orphan of the matching blah log. This may happen at rare times.

4.1.4 Obtaining local records

Local records are built by using the residuals of the job table. It is thus important to actually catch all the grid jobs before looking for local jobs. This essentially means that all the blah logs produced on the period of interest by all the CE submitting to the batch system must have been imported in the database. If this assumption is satisfied, then everything remaining in the job table after having extracted the grid records, must be accounting for locally submitted jobs. As a further measure, the submission host is also considered and matched against the CE hostnames. If it matches then it cannot be a local job, since CE are only enabled for grid job submission. Local jobs are only submitted by dedicated *user interfaces*, and jobs submitted from there are certainly local jobs; different cases are not expected. Should they appear they would be investigated consequently. Chances are simply that a new UI or a new CE have been activated.

To obtain a local record one need a mean to provide the analogous of the information provided by blah records: while a Grid user is member in a Virtual Organization whose name is known from the blah record, in this case we need to map the local user who ran the job to

the experiment he belongs to. This is usually done by mapping the user name and the batch queue name to the experiment name, and locally maintaining an up–to–date map on purpose.

5 DATA PROPAGATION

Once the accounting records are safely rebuilt in the acct database, a couple of alternatives are possible:

Direct delivery to the EGI accounting portal

Although this is feasible, this option has been discarder for a number of reasons:

- A considerable effort would be necessary to correctly convert our record in a suitable format and deliver them in a reliable and compatible way.
- The unavoidable testing phase would most probably require help, assistance, time and work from the apel team [3], to check and confirm the correctness of the delivered data. This would also delay the overall recovering time.
- This solution would skip the second level HLR and then there would be no more a complete accounting repository for the Italian Grid. As a consequence the HLRmon portal [2] would miss INFN–T1 accounting data.
- a solution whose correctness can be verified locally would be preferrable, since there would be little or no time lapse in waiting for confirmations from third parties.

Feed the HLR database

Accounting data for the Italian Grid are currently delivered to the egi accounting database by the second level HLR, and the second level HLR gets data from the lower level ones.

Directly feeding the hlr-t1 database would bypass the task accomplished by the dgas sensors, where the pathologic behaviour was found, and would leave intact the remaining part of the dataflow.

To do so we have to implement an interface to adapt and convert the accounting records in a format compliant with the one adopted by DGAS. Any needed test and check for correctness can be made quickly, thanks to a fair insight with the DGAS system and because any hypothesys can be quickly experimentally verified.

5.1 Feeding the HLR

HLR makes use of twelve tables in a couple of MySQL databases. The pushd component, running in the CE delivers record to a hlr_tmp database. After a fair amount of intermediate processing the received data are finally stored to the only table providing a

complete representation of the accounting records: the hlr.jobTransSummary table (JTS for short, in the following) whose description counts 33 fields.

New records coming on this table are delivered to the second–level HLR at regular times by another DGAS component, the hlr-urforward agent.

Although our acct.grid_job table represents accounting data with a count of 25 fields, the raw informational content superseeds the one in JTS (for example, the pending time is not reported), thus all the information needed to build a single JTS record is present in a single acct.grid_job record. As for local jobs, the same consideration holds, except that the record would come this time from the job table only.

The conversion has been implemented in PostgreSQL by mean of a couple of views, jts_cream for the grid jobs and jts_local for the local ones:

- CREATE VIEW jts_cream AS SELECT (...) FROM gridjob;
- CREATE VIEW jts_local AS SELECT (...) FROM job;

Worth noticing is the fact that the JTS.siteName field is directly generated by the view, and this means that there is no more need to split the lsb.acct files on one part per site.

The correctness of the view has been verified by direct comparison (records created by jts_* must be equal to the ones already present in the JTS and entirely created by DGAS). However, one difference has been intentionally left in JTS.accountingProcedure. This varchar field reports the nature of the record, whether it is a grid one or local; a short suffix have been prepended so that it is possible to distinguish between records generated by the recovery process and the original ones.

5.1.1 Repairing the JTS table

Up to now, we are able to directly generate JTS records at will. Next step is to “repair” the JTS table, which suffers of two problems: missing records and duplicated records. DGAS accepts duplicated records in the first level HLR as a design choice; duplicates are delivered to the second–level HLR where they are finally discarded before the final deliver to the egi portal. We would delete duplicated records anyway as part of the repair process. This eases contents comparison between the acct and the hlr databases.

The repairing is performed by a python script, connected to both the PostgreSQL acct and the MySQL hlr databases. During the repairing phase all the DGAS component are shut down. After the first repairing is done and after any safety check has been completed, and after we are pretty sure about correctness of the intervention, the hlr-urforward component can be started. This would reactivate delivery of new records to the second level HLR. Should something still go wrong, well, we have a second chance: having been alerted, the

administrator of the second level HLR can delete the records received from our one (they are detectable, as said) and we can repeat the process.

In a sketch, the python script works this way:

- for each record R returned by the following query on the acct database:

  SELECT * from jts_cream;

- Search in the hlr database for records with the same Local Resource Manager ID and the same end time:

  "SELECT id,accountingProcedure FROM jobTransSummary

  WHERE

  lrmsid=%s AND enddate=%s" % (lrmsid,edate)

  There cannot be two different jobs having the same ID and the same end time. The couple thus constitutes a Primary Key.

- If none are found, then we have a missing record and all we have to do is to simply add it into JTS

- If one or more record is found: we check them for accountingProcedure to see whether they are grid or local records. We delete those different to R and eventually insert our copy, only if needed.

6 PERFORMING THE RECOVERY

6.1 Recover at INFN-T1

The actual operation was performed on 2013, October 2 and took roughly 24 hours to recover into the hlr database a worth 140 accounting days, dating from May 11 to September 30. A number of 7,288,440 missing records were restored: 49% of the total over the period.

A couple of days after all the newly inserted records were propagated to the second level HLR. Days after again, an unnoticed format mismatch in a field finally revealed itself, causing incorrect values to be delivered to the egi repository. This was the effect of a transient fixed bug. After having identified the reason, a SQL UPDATE was sufficient to definitively fix things. Two more days after the accounting reported by the egi portal for INFN–T1 was finally correct and up to date.

6.2 Recover at INFN-PISA

The INFN-PISA Tier2 had a similar accounting problem, affecting a period of a few months. A few days after completion at INFN-T1, the procedure was successfully replicated.

The procedure proved to be robust and easily adaptable to the different setup of this Tier2. One main difference is that they miss a local HLR, so their CEs are configured to deliver records to the HLR managed by the operation team at INFN-CNAF. This makes things a little less straightforward for the recovery scripts, since a direct connection to the MySQL database of the HLR must be settled.

To ease things, the acct database has been locally built and populated; then a dump have been sent to the author, which has then locally at the HLR executed the second step.

6.3 After the recover

After having recovered the missing data in the JTS table, the DGAS sensors can be restarted, as INFN-PISA actually did. Another option has been pursued at INFN-T1. The "rescue scripts" simply can be executed once a day, running against the accounting files of the previous day. Doing this has number of advantages:

- The DGAS sensors needs no more to run on each CE, thus sensibly reducing the load (see. FIG. 1)
- Splitting the lsb log files, one per logical site is no more needed.
- In the HLR database, only the JTS table is needed. This reduces to roughly one third the needed storage for that database.
- Partial or distributed malfunctioning cannot happen: all data are parsed and matched by a single component in a single place; transaction in the acct database prevents failures at uncertain positions.
- Comparison with raw log file is more straightforward. DGAS accepts by design duplicated entries in its database. This makes difficult direct comparison with LSF logs.
- Richest data retention: the subset of data retained in the acct database from the LSF log records collects more data than those strictly needed to populate the JTS table (the batch system queue name is an example) These can be useful for troubleshooting purposes in the T1 farm.
- A strong insight has been gained while working on accounting issues. As a Tier1, being able to quickly fix or tune the accounting is important, and permits to report finer measures of our activity. For example, the Normalized CPUTime can be exactly computed from the actual computing power of the single Worker Node, and no more as a mere average power of the Worker Nodes in the site.

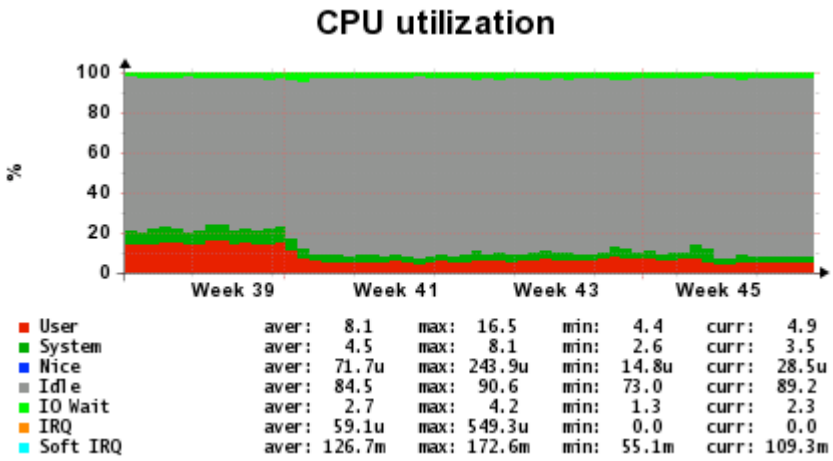

Fig. 1: CPU load on a CE before and after turning off the DGAS sensors

7 CONCLUSIONS

The troubleshooting work to understand and solve a serious accounting problem has been described. The need for a different approach to the recovery process has been explained. An alternative solution has been designed, implemented and used. This has effectively recovered in a couple of days the accounting reports on the EGI portal for INFN-T1 and INFN-PISA. The traditional method would have required much more time. The implementation proved to be flexible enough to be used on a daily base, thus replacing the DGAS sensors.

8 ACKNOWLEDGEMENTS

Andrea Guarise, for the support provided during the investigation and the ready responsiveness when and after applying the recovering steps.

9 REFERENCES

(1) R.M.Piro, A.Guarise, G.Patania, A.Werbrouck: “Using historical accounting information to predict resource usage of grid jobs”, Future Generation Computing System (2008), doi:10.1016/j.future.2008.11.003.

(2) S. Dal Pra, E. Fattibene, G. Misurelli, F Pescarmona and L Gaido: “HLRmon: a role-based grid accounting report web tool”, doi:10.1088/1742-6596/119/5/052012, Journal of Physics, 2008

(3) https://wiki.egi.eu/wiki/APEL