# A Pre–Processor for SGAS Records

P. Flury

5th November 2010

## Introduction

The *ch.smscg.sgas* software package provides a daemon tool, which pre–processes original SGAS accounting records[1] and stores them into accounting aggregates. These aggregates can later be passed as input to tools that generate usage reports and/or visualize usage patterns (like views for websites).
Each aggregate is *time* sampled – typically into samples of multiples of one day – and is a composite sum of the properties like *wall_time, charge, etc.,* discriminated by any possible combination of the following key identifiers: of the *user* that submitted the jobs, of the *VO* under which the jobs have been sent, of the *Computing Elements (CE)* from which jobs were sent, and of the *job statuses*. As can be seen in figure 1, the combination of these properties results in 15 distinct tables.
Each table consists of a constant set of properties (like the number of jobs – *n_jobs* – the *charge* etc. of the aggregate), which we term **body**, and a set of properties, which varies for each table and which we term **key**.

The first and only aggregate, which is created from the original SGAS accounting data is described by the table with *key_0*. It is the most specific aggregate, which distinguishes the *user* (g_id), the *VO* (vo_id), the *Computing Element* (m_id), the *job status* (status), the *sampling rate* (resolution), and the *epoch time* of the aggregated record. Jointly, these six properties uniquely identify the aggregated record (or to put it in database parlance: they are the primary keys of the relation).
The aggregates denoted by key_01 to key_04 are all deduced from the key_0 aggregate (and not from the original SGAS accounting records anymore.). The key_01 aggregate on the contrary becomes the parent for the key_011 and key_012 aggregates. The same holds for key_02, key_03, key_04 etc.. This ' pipelining design' considerably reduces the amount of processing steps. On the Grid of the SMSCG project, for example, we observed a compression ratio from the original SGAS accounting data to the key_0 table by a factor 76[2]; That is, the number of records to process for the creation of every subsequent tables has been

[1]Notice, the ch.smscg.sgas package assumes the SGAS accounting records are stored in a relational database (i.e. it will not work for couchdb settings).

[2]The number was computed on about 120k accounting records and under the situation that most jobs came from very few power users. A more balanced the Grid should achieve better compression rates.

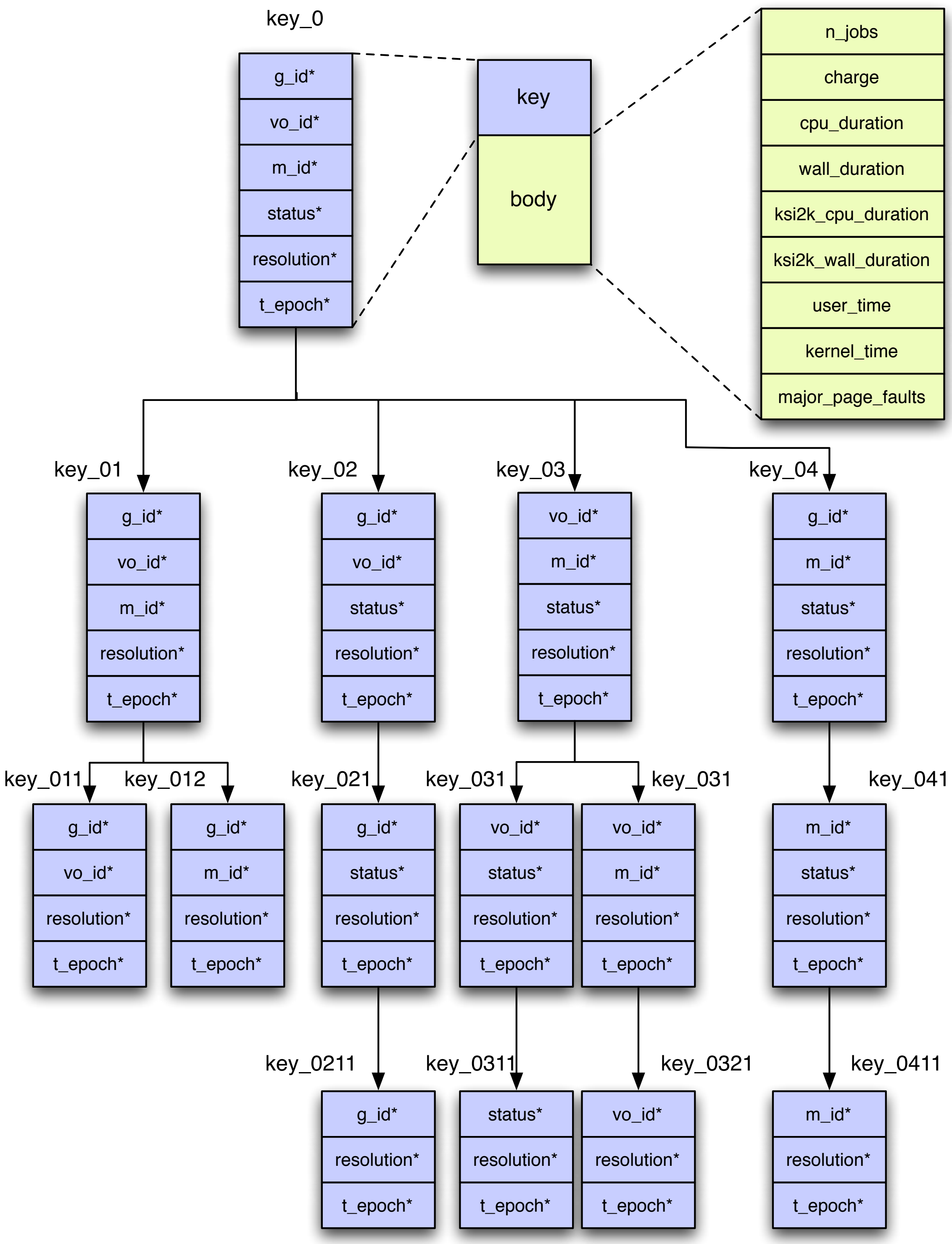

Figure 1: **Different SGAS Aggregates**

reduced by at least a factor 76.

The processing pipeline of figure 1 only depicts the 'vertical' aggregation, that is every depicted aggregate has the same constant sampling rate (i.e. at a fixed *resolution*). The daemon tool, however, also aggregates 'horizontally', that is by further sampling the aggregates. For example, an aggregate that was created at a resolution of 1 day, can further be aggregated at resolutions which are multiples of 1 day, typically at week and month resolutions.

## Installation

The *ch.smscg.sgas* package is written in Python. The package has the following requirement:

- `sqlalchemy  >= 0.6.0`

The installation of the package is done by:

```
user$ cd ch.smscg.sgas
root$ python setup.py install
```

This will install the package into the python default path and create the following files:

```
/opt/smscg/sgas/etc/config.ini
/opt/smscg/sgas/etc/logging.conf
/opt/smscg/sgas/etc/init.d/sgas_aggregator.py
```

## Configuration

The aggregator daemon has two configuration files. The *config. ini* file is the main configuration file (default location */opt/smscg/sgas/etc/config.ini*. Its entries read:

**sqlalchemy_sgas.url** This contains the URL to the SGAS database, which keeps the original SGAS accounting data. In the same database the aggregated tables will be created. Example: *sqlalchemy_sgas.url=postgresql://sgasuser:passwordhost:5432/sgas*

**sqlalchemy_sgas.pool_recycle** The frequency for recycling connection pools to database (in seconds). The default value is set to 3600.

**periodicity** This parameter defines the processing interval for the aggregation of the accounting records (i.e. the daemon runs every *periodicity* interval). The unit is seconds.

**resolution** The base resolution of the aggregated records (or if you prefer the sampling rate). The unit is seconds.

**factors** A list of numbers by which the value of the *resolution* parameter will be multiplied. For each of the resulting new resolutions aggregates will be computed.

**refresh_days_back** At **start–up**, the daemon starts creating aggregates starting from *refresh_days_back* (compared to the start–up time). After the first run the daemon only considers original SGAS accounting records, which were inserted (entry of *insert_time*) since the daemon's last execution. In more detail, the daemon looks for the record of the oldest job (in terms of its *end_time* value ) and recomputes all aggregates starting from then on. Hence, if a site is publishing comparably old SGAS accounting records, the aggregates will still be updated correctly.
Beware, if you run the daemon for the very first time on an existing SGAS database, you need to make sure to set the *refresh_days_back* value so it creates aggregates starting from the oldest entries of the SGAS database. After having run the daemon once successfully you should set the value back to a small value, as to avoid unnecessary re-computation of the aggregates upon a restart (e.g. like after a reboot)

The *logging.conf* file is used to configure where to log the output of the daemon and on what log–level. The configuration of the logger is standard and we thus do not go into it.

## Running the Daemon

The daemon can either be started and stopped as root by

```
root$ /opt/smscg/sgas/etc/init.d/sgas_aggregator.py start
root$ /opt/smscg/sgas/etc/init.d/sgas_aggregator.py stop
```

You may also create a symlink to /etc/init.d and add it to the startup scripts of your system (e.g. by update-rc.d or chkconfig)