The bulk dataTransfer procedures

(Cascina – Jan, 16th 2018)

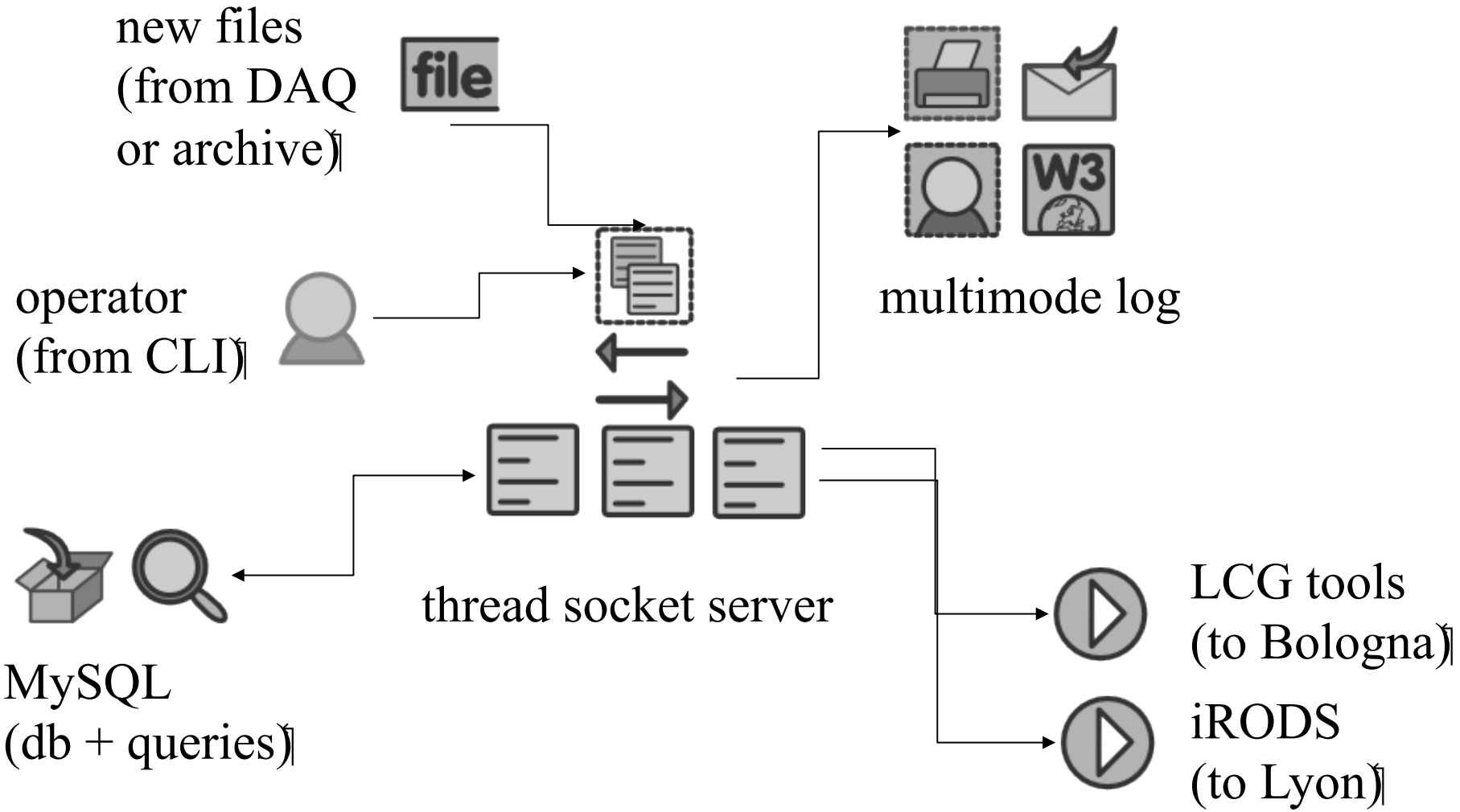
L. Salconi, A. Bozzi

Specs for O2 software module used

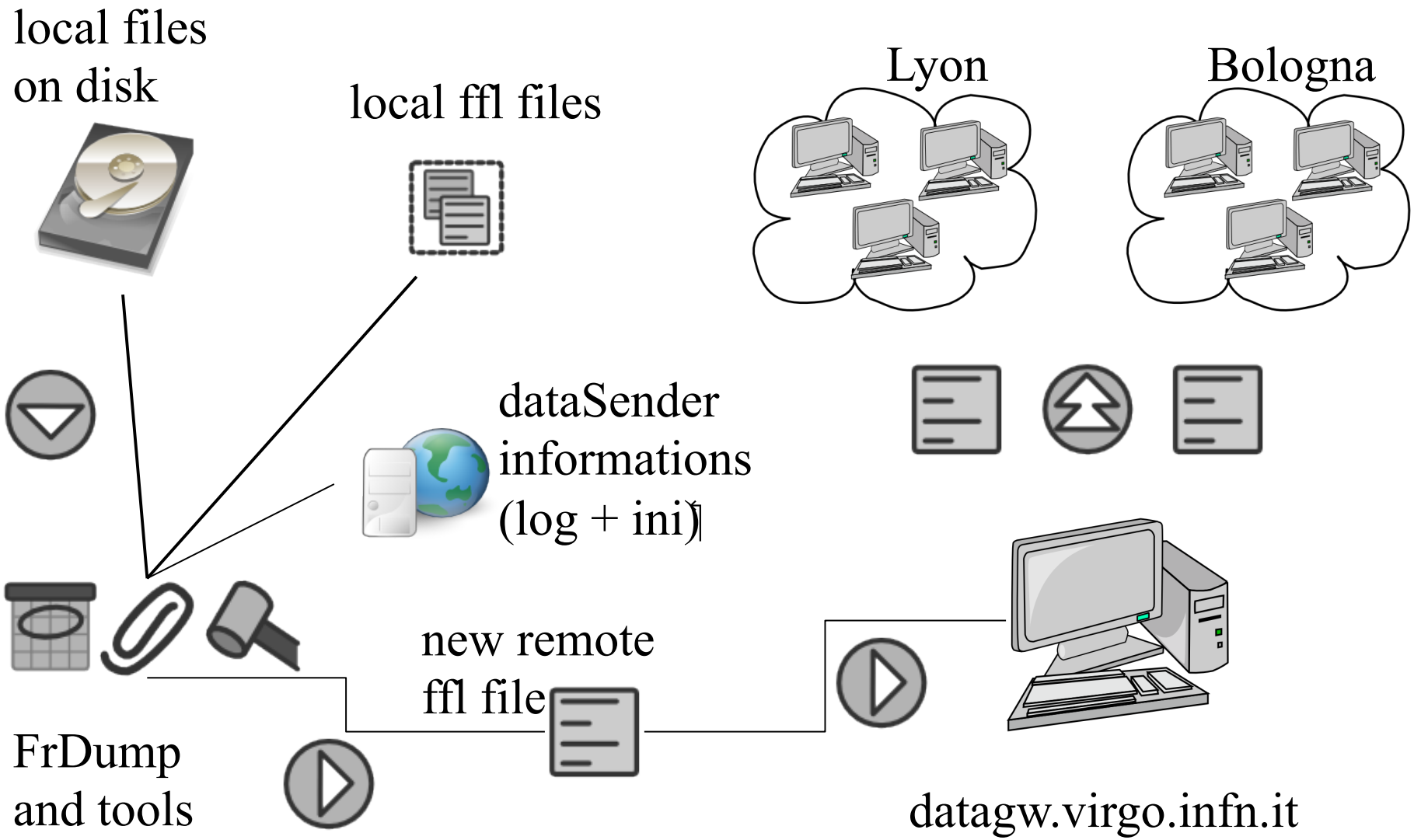
A modular program (written in Python) that manages the file replica to the remote repositories (the *datasender.py* module):

- class based architecture, modular and expandable for future requirements;
- database with MySQL interface (with full stats for each file);
- logging system with support support for file, smtp, http;
- new command line interface with feedback support;
- thread based;
- run-time parameter controls (bandwidth limits, destination dirs, ecc..);
- support for both data sender engines (***GRID lcg tools*** and ***iRODS***) via a wrapper classes with standard methods (sendFile, checkFile);
- support for standard .ini sections oriented configuration files;
- management for 2 main data flows (rawdata and RDS) for each remote repository;
- socket interface with the DAQ data production flow;

Simplified operational schema



The “fake” ffl generation



The O2 test: iRODS / LCG pro and cons

For LCG tools:

- (-) cerificates: two days normal expire time, extended UP to ine week with a workaroud;
- (+) easy way to set operations timeout;
- (-) url based syntax, not useful for scripting;
- (-) no recursive options;
- (+) only adler32 support (no extra load for check it);
- (+) transfer speed: 85-95 MB/s (stable performance).

For iRODS:

- (+) no cerificates used: only one authentication (lifetime?);
- (-) no way to set operations timeout (hanging commands!);
- (+) posix like based syntax, very useful for scripting;
- (+) it has recursive options;
- (-) only md5 support (some extra load for check it!);
- (-) transfer speed: 30-85 MB/s (variable performance).

Tips, tricks and desiderata (… 2018!)

◆ *a single transfer engine supported at all CC's:* this will avoid multiple engines procedures and will help in replicate

◆ *a local disk bucket at remote CC's:* we must decouple the data transfer from local storage procedures. The transfer engine must be used only for the data replica and, after this, the file must be moved and registered locally from the bucket to the final storage location, following multiple methos: by operator (hand), by script (crond) or by software (daemon).

◆ *a logging and bookeping database:* no more fake ffl, the users will ask for data from a web based interface and ffl can be produced "on the fly" like a query output after a request (GPS start, end segment(s), data type, data location, metadata, etc … ;

◆ *a recursive parser for scripting:* the current architecture do not performs well on big subdirectory layout (like LIGO's, 50Hz, etc…) and do not support recursive replication (this issue is closely related to the used engines) -> crond scripts are built to take care of this data type (async from DAQ);

◆ *an homogeneus checksum control:* this will avoid a double file reading, less metadata exchange and will introduce a standard check parameter for all repositories;

Tips, tricks and desiderata (from 2015 slide!)

Some tips, tricks and desiderata to improve the software:

- *a permanent daemon module installed at CC's:* no more fake ffl, better control on file's true final location, use of a true md5 or adler32 checksum code, ffl can be produced "on the fly" after a request;

- *a recursive parser:* the current architecture do not performs well on big subdirectory layout (like LIGO's) and do not support recursive replication, this issue is closely related to the used engines;

- *a local disk bucket at remote CC's:* this will avoid the coupling with the local engines. Any engine can be used for the transfer and, after the succesfully replica, the file can be moved and registered locally to final destination.

- *a longer proxy certificate expire time:* this will help with LCG tools and daily authentication;

Final architecture (desiderata … from 2015!)