Astro-**W**ise **E**nvironment

Distributed processing - proposal

January 19, 2005

Contents

Chapter 1

Distributed Processing

1.1 Definitions and Use Cases

Distributed processing means that certain recipes, like processes, can or will be executed on specially devoted processors. A recipe like process is a process which is started from the shell prompt (or via a gui) and takes only directions via arguments on the command line.

A data server is a server which can retrieve image files from the astro-wise image database, provide a local copy (cache) and store new images.

A database server (read Oracle server) is a server which provides the metadata for the astronomical images and the astronomical sourcelists.

A user is a person which has read and write access to (parts of) the database and therefore has a database username and password.

A visitor is only allowed to browse (parts of) the database and therefore does NOT (need to) have a database username and password.

A user can work in the AWE environment in different ways:

Type 1 Via a web-interface (TBD): the user does not have to install anything on his own workstation. He can only run the standard recipes and browse the database. The processes are run on specified machines of one of the AW compute centres (e.g. OmegaCEN).

Type 2 Via a local install on one network node in a LAN. Here the user will hook up to the closest database server (Groningen, Munich, Napels) and the closest data server. If the user is not in the same domain as one of the data server centers (e.g. in Leiden) a caching data server could (should) be installed. He uses the standard python code.

Type 3 As above, but now the user wishes to modify the standard python code. He can do a cvs checkout of the opipe stuff and make local modifications. He does not have to install all the python bindings since they are installed on a central node of the LAN. This is typical for experienced users or top-level software developers.

Type 4 Via a complete local install on the user's own workstation. The user has to install everything from python to swarp and should make his own python bindings to for example eclipse and oracle. This is only for advanced software developers !

Certain processes are best performed on dedicated machines, for example

Science This task is used to create calibrated science frames. It runs best in parallel on a dedicated server (i.e. like the hpc of Groningen University).

Coadd This task is used to coadd a set of regridded frames. It runs best on a machine which has a lot of memory.

SourceList This task is used to create sourcelists from science or coadded frames. It runs best on a machine which is closely coupled to the database server.

1.2 Requirements

It is obvious that different types of users require different degrees of freedom. A user of type 1 has the least influence on his process: it is completely directed by some settings from the 'management' which the user cannot change.

A user of type 2 has the means to modify (and so influcence) the AWE environment by directing certain processes to specific machines, deviating from the default settings.

Users of type 3 want their modified python code executed (on a special machine) so they should be allowed to ship their code to that special machine so that it gets executed.

Users of type 4 are on their own. Since they have modified non-python code it cannot be executed on a different node. They should always do their testing on the local node where non python things are adapted or setup their own system of remote modified nodes.

1.3 Available tasks

The following tasks should be executed on a cluster:

ReadNoise, Bias, HotPixels, ColdPixels, DomeFlat, TwilightFlat, FringeFlat, NightSkyFlat, MasterFlat, Gain, Zeropoint, Illumination, Science, Reduce.

The following tasks could/should be executed on a separate, single node:

Coadd, SourceList, AssociateList

Chapter 2

Implementation

2.1 User interface

Simple command line interface:

```
awrecipe [-node nodename] [-mycode] -task recipename ...

-node nodename:  the specified node on which the recipe

is to be executed

-mycode:  designates that modified python code is

to be executed

-task recipename:  designated the recipe to be executed

....:  recipe parameters
```

2.2 Code shipping

Python code shipping can be done by zipping the whole local opipe tree and sending it to the node on which the execution should take place. Python 2.3 has the nice feature of the `writepy` function in the `PyZipFile` module. Within a few seconds the code can be packed into a zip file and shipped elsewhere, where the packed (or unzipped) zip file can be placed in the PYTHONPATH i.e.

```
env PYTHONPATH=/home/kgb/pipe.zip python2.3 ...
```

2.3 Problems

Problems encountered sofar:

- The `writepy` function only ships binary python code so the configuration files which are in the opipe tree don't get shipped. Solutions: migrate the configuration files from text to python code or (easier) ship the configuration files as well.

- The user specific AWE settings which also contains the context specification (i.e. in `$HOME/.awe/Environment.cfg` must be shipped. This could very wel be put in the same zip file.
- Copying the shell environment variables which overrule the AWE configuration settings is also necessary. They could very well be stored in the same zip file.

  N.B. The last 2 items must also be performed if the local code is NOT modified (user type 2).
- The recipe to be executed elsewhere can only be started from the command line since the PYTHONPATH has to be set before running the recipe. This is in fact not a problem but makes it incompatible with the current taskservers. It might be necessary to create a new server thingy.
- Since username/password combinations and other sensitive information are going to be 'piped' over the internet to remote taskservers secure connections are necessary.
- Some parallel clusters will have a job queueing system (like the Groningen HPC). A taskserver must also be able to queue the jobs on such a system. Maybe it is possible (and allowed) to run a taskserver on the frontend machine of the new HPC.