Pipeline Basics:
A brief and general review of data pipelines.

Jared Crossley, NRAO
November 27, 2007

A software pipeline is a program, or collection of programs, that performs a task with no, or minimal, user interaction.

Why Use a Pipeline?

If the input parameter space is large, many decisions about data processing must be made in moving from the raw to final data format. It is customary for software engineers to leave decisions about data reduction to the user. This is quite effective. However, even for an expert user, an interactive requirement will likely demand a large amount of time. By reducing the amount of user interaction, data reduction time may decrease dramatically.

In addition to saving time, data reduction that requires less interaction increases the accessibility of the data reduction system. A pipeline will make the system useful not only to experts, but also to the novice. But a pipeline is not merely a tool for the lazy. Rather, it can actually be a learning mechanism, whether the object of the learning is the data reduction software package, the applicable science in general, or both. Data processing cookbooks are popular because they contain "recipes", or short and to the point instructions on how to get the job done. A pipeline is like a cookbook brought to life (assuming the pipeline is well documented). The user need not type the recipe -- the pipeline does it for you.

There are many uses for pipelines which I will not attempt enumerate. However, one task is particularly appropriate to a pipeline: surveying. Generating a large survey requires processing of many data files. Depending on the size of the input parameter space and raw data set this could take a very long time using interactive programs. A pipeline is an ideal solution. Now the survey is generated quickly, and using a standardized method not subject to user errors like, for example, typos.

Implicit in the use of the term "pipeline" is the idea that something more interactive existed before. Interactive data reduction systems usually come first, and pipelines later. A pipeline thus becomes a mechanism for invoking the individual data reduction packages until the whole data processing job is accomplished. Be aware: it may happen that building a robust pipeline requires modifying some of the packages to allow for automated use.

Constructing the Pipeline

Build it in layers! If packages for performing parts of the data reduction task already exist, it will be easy to begin a pipeline by offering the user a means to specify all of the essential input parameters. Then, execute the packages with each package using the specified parameters. This is a low-level pipeline, but it can still save time if most parameters are constant throughout the domain of raw data.

The next step is to identify which parameter values tend to be most useful within the domain of input data. These values will become the default parameter values. Although the pipeline controls remain complex (lots of parameters are available to the user), the pipeline can be run under most conditions without changing any of the default values.

Over time, use of the low-level pipeline will reveal parameter dependencies and data processing algorithms (perhaps performed initially by the user) that can be incorporated into pipeline to further automate the system.

One way to greatly reduce user interaction is by acquiring metadata. For example, the raw data file name, the observing center frequency and band width, the observing mode, etc., may be used to set initial pipeline parameters. If the raw data is stored in an archive, the archive will contain the meta-data for each data file.

Concerns

1. How much control should the user be given?
This depends on the pipeline's target audience. An expert may want lots of control, while a novice will want less control in exchange for greater simplicity. Bulk processing may present an exception to the expert case, since more controls mean more interaction time.

2. How many output diagnostics should the pipeline produce?
This varies depending on the task at hand and the preferences of the user. It may be wise to include a pipeline input parameter that determines the amount of output. By default, diagnostics may be turned off or minimized, but when necessary the user can increase the diagnostic output.

Expert users may be interested in more than just the “final” data product (image, spectrum, etc.). It probably won't hurt to include the calibrated data and any calibration tables in the pipeline output. This will allow the user to tweak data after running the pipeline. For surveys, keep in mind that the pipeline and its packages will improve with time. Thus, the calibrated data can be used to "upgrade" the survey contents, without redoing all processing.

Validating Pipeline Output

The final step in the pipeline process is necessarily interactive: validating the output. However, the validation process can be simplified by providing a fast way for the user to view pipeline output, including diagnostics, and delete (or flag) the output if it is deemed unacceptable. Again, remember that pipeline upgrades happen. Simply because output is poor does not mean it is a total waste. A future version of the pipeline may be able to take old calibrated data and improve it.

Gotchas -- problems to avoid

1. Filled disks. When running a pipeline on a large amount of data, take care not to fill the hard disk. This has happened a number of times while running the VLA pipeline. Disk space will be consumed by:
a. raw data,
b. output data,
c. diagnostic plots and log files,
d. temporary files generated by and during data reduction,
e. and (least controllable of all...) other users on the computer!

References

Capabilities of the VLA pipeline in AIPS, AIPS Memo 112, L. Sjouwerman, Mar 2007. `http://www.aips.nrao.edu/aipsmemo.html`

VLA Archive Imaging Pilot, L. Sjouwerman, Feb 2007. `http://www.aoc.nrao.edu/~vlbacald/`

NRAO VLA Archive Survey Pipeline System Users Manual, J. Crossley, Nov 2007. `http://www.aoc.nrao.edu/~jcrossle/`