DC3 Desired 3rd Party Software

from: Data Challenge 3 Management.

Required Software Libraries and Utilities

Keep VisionWorkbench?

?VisionWorkbench is the C++ foundation to our representation of images. Robert Lupton has several times questioned whether we should continue using it. It rather heavyweight and complex considering how little of it we are using. Also we have some evidence that pixel access is needlessly slow.

We are also using its interface to LAPACK for singular value decomposition and matrix inversion, but there are other options including boost libraries that we already have. Some issues:

- Robert Lupton points out that using vw's interface might make it harder to fully optimize our lapack (e.g. g. use Atlas).
- Andrew Becker found the boost interface much slower than vw. This may have been due to not compiling the boost library with optimization, which bears on the previous point.
- Andrew Becker is suspicious that there may be bugs in vw's interface to LAPACK.

C/C++ Math

Robert Lupton says:

> I think that we need something like ?gsl (Gnu Scientific Library). We'll probably need FFT-based convolutions, in which case probably ?fftw3 too.
>
> One issue with fftw3 is that it's a pain to build both single and double versions (and there can be name conflicts). We should probably settle on only one floating image type, and I'd suggest float. We should probably do this anyway--otherwise you double the number of explicit template instantiations needed.

Andrew Becker would like us to upgrade our ?minuit (function minimizer) from version 1 to version 2.

Shall we include matplotlib (or another Python plotting package)?

It would be useful to have a plotting package.

?matplotlib is a the most popular solution for Python. It supports the usual 2d plots and histograms and has some support for displaying images. It is rather slow (especially for image display), but quite flexible.

It offers both a good interactive mode (modelled after MatLab?) and also a reasonable API. One of its strengths is that it can work with all popular graphical back ends, including Tcl/Tk?, WX, Qt and GTK (the first three of which are cross-platform). Matplotlib is the only python plotting pacakge I know of that works with Tcl/Tk?, which is pre-installed on most coputers. But its Tcl/Tk? interface is starting to fall behind some of its other interfaces.

Competing graphing packages include (but are not limited to):

- ?HippoDraw: It is very fast and well designed. The Apollo project is using it to plot lunar laser ranging data in real time. They decided against matplotlib because it was too slow. HippoDraw? requires the Qt GUI library, which is huge, though high quality.
- ?Chaco by Enthought: I know little about Chaco. I've heard that one strength is interactive modification of graphs and that there is not much documentation. It requires the WX GUI library, which is probably the most popular GUI library for Python, but not a package we will necessarily need for anything else.

Python Math: SciPy

?SciPy is the shaping up as the standard Python package for for high-level math and statistics routines. It includes routines for matrix manipulation, statistics and lots more (depending on what components you build); here is a ?list of packages included in SciPy?.

?A list of other scientific packages (the competition).

Note that ?SciPy has not fully stabilized (it is at version 0.60) and the documentation is not very complete. Nonetheless it is quite useful in its current form. It had a reputation for being difficult to install, but apparently that has gotten better.

C++ Unit Testing

We occasionally need unit tests written in C++. Which package shall we use?

Robyn Comments

> RHL suggested cxxtest which creates the unit testing framework using headers only. Comes with good reviews. I've not used it;, only read the online documentation.
> Serge suggested boost's unittest.
> JeffK said cppunit was our default choice unless someone has better experience with another.

Debugging and Profiling

Including some standard tools may be especially desirable if we supply our code as a virtual machine.

- valgrind: popular memory checker and profiler
- gprof: profiler
- gcov: test coverage program; helps make sure your unit tests cover all lines of code.

Should we consider any commercial tools? Licensing would presumably be challenging.

Robyn Comments

> ?ggcov looked like an interesting equivalent to gcov. I've not used it, only read the online doco.
> We'll be buying licensed tool for coding standards checking. Such code usually comes bundled with other testing tools.

Boost or tr1

Should we switch to tr1, the "standard" C++ libraries (or use boost tr1 emulation, so our code looks as if it's using tr1)?

This would require using g++ 4.?, which is newer than that supplied on some platforms. The advantage is better long-term compatibility.

Fortran <-> Python Wrapping

There is need for a simple way to wrap Fortran code into Python. SWIG requires writing intermediate glue code in C (with the obvious disadvantages). SciPy? has f2py (which we have been using so far in MOPS) which sort of works but it is not actively maintained, is a mess to modify and does not easily supports new compilers.

tcmalloc

Google has made their malloc library available ?http://code.google.com/p/google-perftools/wiki/GooglePerformanceTools; it is supposed to be fast and efficient, even in a threaded environment. It also includes a heap profiler and checker that may be useful in conjunction with Citizen.

Keep CORAL/SEAL?

CORAL gives us the ability to switch RDBMS packages with minimal changes to LSST code. On the other hand, it has caused us some difficulty in the past with building on multiple platforms, including MacOS and Linux64. There have been difficulties mapping C++ types to database column types. CORAL is also missing `CREATE TABLE [IF NOT EXISTS] LIKE` and `LOAD DATA LOCAL INFILE` functionality that we have needed.

One option is to enhance CORAL to add the functions we need and alter the type mappings. Build difficulties would persist.

Another option would be to code directly to the native RDBMS API. Changing databases would then involve reimplementation of two classes (`DbStorageImpl` and `DbTsvStorage`). These two classes, in their CORAL versions, use about 1000 lines of code.

A third option would be to use a different database isolation layer like ?SOCI. This could help with the build difficulties, but it still introduces a dependency and has some quirks of its own. It also might require rewriting SQL statements (but not other code) if the RDBMS is changed.

Keep Jaula?

Jaula is the JSON parser used for Policies. Since PAF seems to be the accepted Policy format, removing JSON support and the Jaula dependency would seem to be a no-brainer.