Beyond R? The Evolution of Statistical Computing Environments

Jay and Mike

Jay is Associate Professor of Statistics, Yale University

Mike submitted his dissertation on August 31, 2010,
and is now Research Faculty, Yale Center for Analytical Science

This informal Chicago Meetup presentation previews an Invited Session for JSM 2011 in Miami organized by Jay and Mike. We apologize in advance for any errors and omissions.

Outline

Duncan starts the discussion...

Quoted from his web page,
`http://www.stat.ucdavis.edu/~duncan/`.

- The goal is to think about where statistical computing should be in the future, e.g. 5 years from now, and to determine how to get practitioners there.

Ross: what's wrong with R?

Quotes from his JSM 2010 talk, "Lessons Learned, Directions for the Future."

- Because R now has a large number of users who require a stable platform for getting work done, it is no longer suitable as a base for experimentation and development.
- R is not very good at handling large-scale problems.
- The following present particular difficulties.
  - Execution of large amounts of R code.
  - Scalar (element-by-element) computation.
  - Computations on large volumes of data.
- Some computational problems involve a mix of all of these.

Luke: moving forward...

From his UseR! 2010 keynote.

- ... a few possible directions for development in the core R engine over the next 12 to 18 months:
  - Taking advantage of multiple cores for vectorized operations and simple matrix operations.
  - Byte code compilation of R code.
  - Further developments in error handling.
  - Increasing the limit on the size of vector data objects.

What can we do?

These points were from Ross' 2010 JSM address, "Lessons Learned, Directions for the Future."

- Wait for faster machines.*
- Introduce more vectorisation and take advantage of multicores.
- Make changes to R to eliminate bottlenecks.
  - Compilation.
  - Use non-copying semantics.
- Sweep the page clean and look at designs for new languages.
- Duncan Temple Lang, Brendan McArdle and I have begun examining what such new languages might look like.

*Ross doesn't like this suggestion, of which Robert seems to be a proponent. We agree with Ross, and note that faster machines aren't really the issue: the issue is having a language that takes full advantage (natively) of current hardware capabilities. R doesn't.

So: what next?

Simon: moving forward with R?

From personal communication with Simon:

- For those interested I have created a branch of R in which I am experimenting with the ideas of Aleph that have to do with the language. This allows us to experiment with the language additions (argument types, function vectors matching, fundamental class type etc.) directly in existing R. This means we can easily test existing code and actually try to use the new features for real work (as opposed to wait for Aleph that needs far more work).
- `https://svn.r-project.org/R/branches/R-exp-R5`
- I have just branched it today and the only addition are optional function types so you can define functions like `function(real x, integer n = 1L, ...) { ... }` but there is a lot more coming.

Luke: a virtual machine for R?

From his web pages and his 2010 UseR! address:

- `http://www.cs.uiowa.edu/~luke/R/bytecode.html`
- `http://www.cs.uiowa.edu/~luke/R/compiler/`
- From his `README`: "Snapshot of byte code compiler for R. The current version requires R 2.12.0 or later. Install the compiler package, and look at the help for cmpfun."
- "Virtual machines, and their machine code, are usually specific to the languages they are designed to support."

Remember this last point, it will be relevant later, and isn't always correct. We also note that Luke is also working on other changes to R noted in an earlier slide and not referenced here.

Mike, Jay, Bryan, ... extending R and more

From drunken conversations over the past few years...

- Memory
  - 8-byte integer indexing
  - Column-major matrices for scalable linear algebra
  - Shared memory for efficient concurrent programming
  - Memory-mapped files (Bryan pushed Jay and Mike on this point)
  - Current successes of the Bigmemory Project (`http://www.bigmemory.org`):
    - 800 GB test matrix on a laptop with a USB drive
    - 160 GB truncated singular value decomposition on the Netflix data (results at end time permitting)
- Thread safety (not our focus, but not unrelated to the memory discussion)

Simon: Aleph?

From Simon's Aleph Wiki:

- Aleph is an open-source project to create the next generation of statistical computing software, possibly as a successor to R. The goal is to provide a modern, flexible system suitable for statistical analysis. All aspects of the project are currently experimental and up for discussion.
- The current experimental implementation is written in C and features its own C-level object system.
- `http://www.rforge.net/mediawiki/index.php/Aleph`
- `http://rforge.net/aleph/`

Andrew: CXXR?

From `http://www.cs.kent.ac.uk/projects/cxxr/`

- The aim of the CXXR project is gradually to refactor (reengineer) the interpreter of the R language, currently written for the most part in C, into C++, whilst as far as possible retaining full functionality. CXXR is being carried out independently of the main R development and maintenance effort.
- It is hoped that by reorganising the code along object-oriented lines, by deploying the tighter code encapsulation that is possible in C++, and by improving the internal documentation, the project will make it easier for researchers to develop experimental versions of the R interpreter.

Ross/Duncan: Lisp?

This project proposes a completely new language! From "Back to the Future: Lisp as a Base for a Statistical Computing System."

- We don't have the resources to build a new system entirely from scratch. We need some giant shoulders to stand on.
- On the plus side:
  - Lisp is a well-established, widely-used system
  - There are a multiplicity of high-quality implementations
  - There are very good resources explaining Lisp at both the high and low levels.
- On the minus side:
  - Lisp has an image problem – it is perceived as a "dead" language.
  - Because Lisp is an amalgam of features there are some inelegances to deal with.

Is that it? A preview of a JSM session?

No, there's more. Give the speaker a beer and brace yourselves...

Past

First there was S

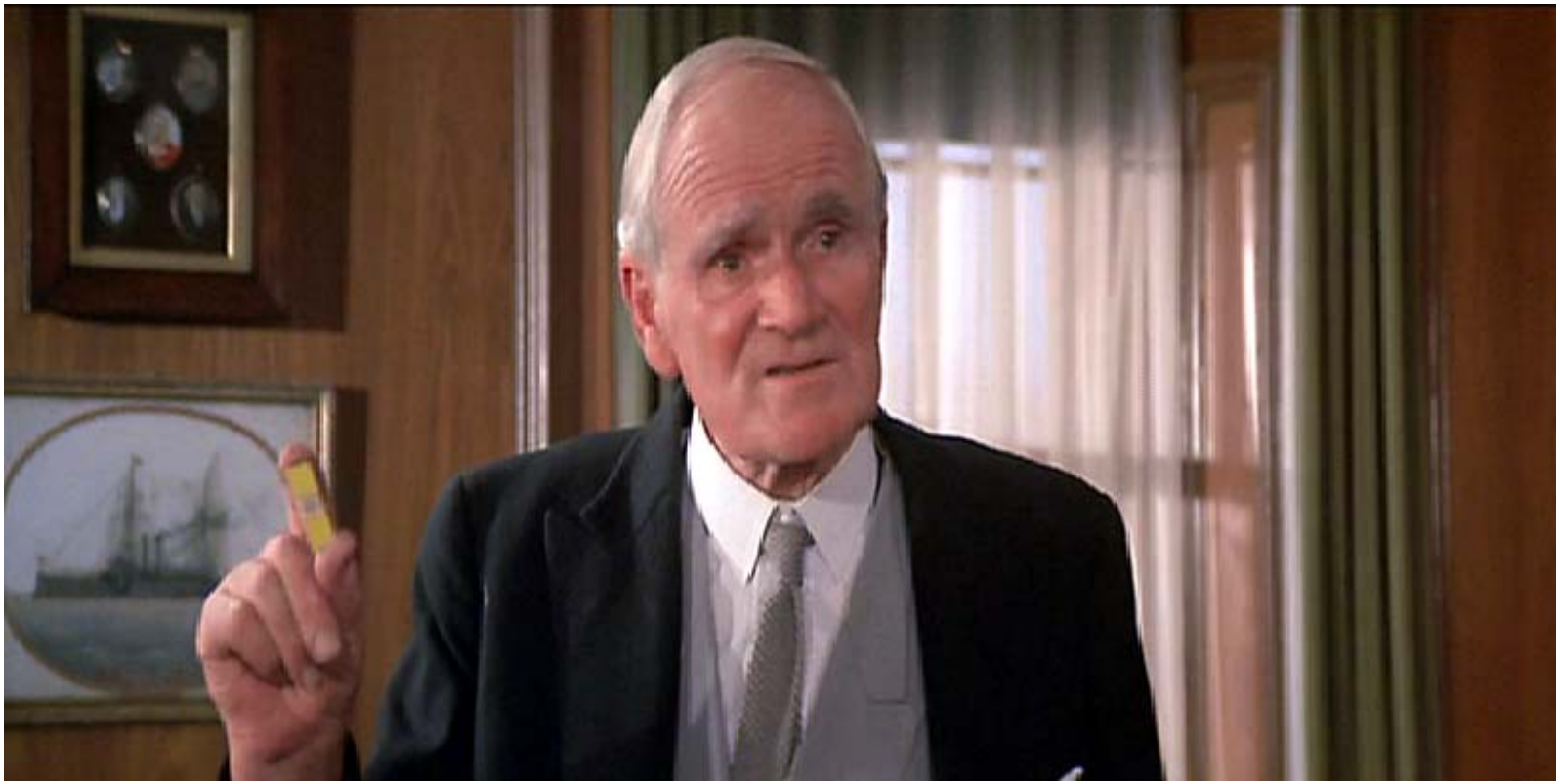
Present

And then there was R

Future?

Only one thing makes sense...

Q?



Q?

Jay and Mike: Q?

```
$ ./cleanandbuild.pl
Cleaning and building Q for Parrot VM.
Copyright (C) 2010, John W. Emerson, Michael J. Kane.

Checking src/ for *.pir...found some, ok
Removing *.pir in src...ok
Building Q, please wait...ok
Testing Q, please wait...ok, tests passed.

You may now run interactive Q.
$
```

Jay and Mike: Q?

```
$ ./Q

Q for Parrot VM version 0.0.1 (2010-10-19)
Copyright (C) 2010, John W. Emerson, Michael J. Kane.

Q is a case study and comes with ABSOLUTELY NO WARRANTY.
You are not welcome to redistribute it under any
circumstance.  See LICENSE for details.

  Natural language support?  Probably not.

Q is a collaborative project with two contributors.
Type 'help.start()' to get started and 'q()' to quit Q.

>
```

Jay and Mike: Q?

```
> a <- 1
> if (a!=1) print("Yikes!") else { print("a is 1") }
a is 1
> myfun <- function(x) { return(x+1) }
> b <- myfun(a)
> print(b)
2
> set_seed(1,2)
Seed has been set.
> rexp(1)
Your exponential: 0.128221109772152
> rexp(1)
Your exponential: 0.866102216996375
```

Jay and Mike: Q?

Based on the Parrot virtual machine (from `http://www.parrot.org`):

- Parrot is a virtual machine designed to efficiently compile and execute bytecode for dynamic languages.
- Parrot currently hosts a variety of language implementations in various stages of completion, including Tcl, Javascript, Ruby, Lua, Scheme, PHP, Python, Perl 6, APL, and a .NET bytecode translator.
- Parrot is designed to provide interoperability between languages that compile to it.
- Uses the Artistic License 2.0

Jay and Mike: Q?

Based on the Parrot virtual machine (from `http://www.parrot.org`):

- Parrot exposes a rich interface for high-level languages to use, including several important features:
  - a robust exceptions system
  - compilation into platform-independent bytecode
  - a clean extension and embedding interface
  - a just-in-time compilation to machine code
  - native library interface mechanisms
  - garbage collection
  - support for objects and classes
  - a robust concurrency model
- Designing a new language or implementing a new compiler for an old language is easier with all of these features designed, implemented, tested, and supported in a VM already.
- The only tasks required of compiler implementers who target the Parrot platform is the creation of the parser and the language runtime.

Why not Java VM or .NET?

- They don't automatically provide
  - multiple dispatch (in the S4 sense)
  - introspection (as in an object being associated with its methods)

  whereas Parrot standardizes them for all languages under the umbrella of the Parrot VM.

- In Parrot, writing a new language is a matter of mapping objects defined in the grammar to the existing facilities of the virtual machine. This is critical for the interoperability between languages.

Jay and Mike: Q?

Where are we now?

- The first 3/4 of this talk was completely serious, and previews an exciting Invited Session for JSM 2011 in Miami.
- The last 1/4 was intended partly in jest:
  - R is here to stay – this is for fun.
  - Q exists (barely) as an experimental project at the moment.
  - Best case? Q may become an open-source experimental project seeking collaborators in the near future. Or it may die quietly.
  - Unless a new language/environment can provide almost complete backwards-compatibility with R, we don't think it will take off. And that level of compatibility seems unlikely.
  - Parrot is still evolving. An original basic Q parser working under Parrot 2.4.0 ceased working under 2.6.0; 2.8.0 was just released last month.

Is this for real?

Not really. But...

- Code will go up somewhere by the end of November (we hope).
- There is no "Q Core" at this point, but collaborators are welcome.
- If there is ever any sense of inertia, at that point we'll start thinking about license issues and formal project organization, etc...
- Could this be a candidate for GSOC?

Q parser grammar example

```
rule block {
    {*}                                          #= open
    <maybenewline> '{' <maybenewline>
    <statement>*
    <maybenewline> '}' <maybenewline>
    {*}                                          #= close
    | <statement>?
    {*}                                          #= close
}
rule if_statement {
    'if' '(' <expression> ')' <block>
    ['else' <else=block>]?
    {*}
}
```

Q action example (nqp: not quite Perl)

```
method if_statement($/) {
    my $cond := $<expression>.ast;
    my $then := $<block>.ast;
    my $past := PAST::Op.new( $cond, $then,
                              :pasttype('if'),
                              :node($/) );

    ## if there's an else clause,
    ## add it to the PAST node.
    if $<else> {
        $past.push( $<else>[0].ast );
    }
    make $past;
}
```

Netflix: a truncated singular value decomposition