21st International Congress on Modelling and Simulation, Gold Coast, Australia, 29 Nov to 4 Dec 2015
www.mssanz.org.au/modsim2015

Using APSIM, C# and R to Create and Analyse Large Datasets

J. L. Fainges [a]

[a] *CSIRO Agriculture, Australia*
Email: Justin.Fainges@csiro.au

Abstract: With the advent of cheap cluster computing, advanced models and improved analytical techniques, scientists are able to explore larger problem spaces than ever before. However, with this power comes added complexity. Large factorial simulations need to be designed and created then the output – which can be many hundreds of gigabytes or more – needs to be analysed and presented in a meaningful way. This paper details one such scenario using the Agricultural Production Systems Simulator (APSIM) (Holzworth et al, 2014). Almost 60 000 simulations were created using five base simulations that generated approximately 100GB of output data.

This output data was loaded into R using a new package designed for loading, testing, manipulating and exporting both input and output APSIM formatted data. This paper explores the mechanics of large scale APSIM simulations including how to leverage the APSIM User Interface (UI) to quickly build a starting point simulation, using the XML libraries in the .NET framework (C# in this case) to easily duplicate, replace and modify structures in the base simulation to create a large factorial and finally looks at methods to analyse large output data sets.

The techniques demonstrated here are scalable; while 60 000 simulations were generated for this project, the same method could be applied to a grid analysis with millions of simulations. Similarly, the processing in R will work with any size data set so long as the computer used has enough memory. Even the language used for generating the simulations can be changed. C# is used here but any language with a robust XML interpreter could be used.

In order to keep the analysis run time down and simplify factorial combination generation, the top level factor (crop type) was used as a divider to split up the output into individual batches that were then processed in R.

APSIM writes a single output file for each simulation. As such, large factorial simulations can produce millions of individual files that need to be processed. To assist with this, an R package has been created and is available on CRAN (under the package name 'APSIM') that automates the loading of multiple APSIM output files into a single R data frame or data table (Dowle, et al., 2014). This package is able to handle single output and factorial simulations as well as import constants as separate columns. Additionally, extra utilities have been added that simplify the process of creating meteorological data files for simulation input.

R has many ways of doing the same thing, some of which are more efficient than others. A number of processes for reading and parsing data were evaluated and the most efficient methods were incorporated into the APSIM package. This evaluation has applications beyond APSIM as they are methods that anyone importing large quantities of data into R will be able to implement.

The package assumes that any data sets loaded will fit into system memory. As simulations become more complex this assumption will become less valid and new approaches will need to be taken such as the use of databases to store data that is not being actively worked on. There are R packages already in existence that do this and the APSIM package will be expanded over time to utilise these options.

Optimisations used include minimising file access, binding files after reading all of them, using dedicated data reading packages and applying techniques that reduce the amount of memory management required. The optimised data reader was able to import and bind APSIM output data at speeds in excess of 11MB/s for 5MB data files.

Keywords: *APSIM, R, programming, XML, grid*

1. INTRODUCTION

With the recent advent of large scale, cheap cluster computing solutions, scientists working in the field of modelling and simulation are able to model scenarios on scales that were unprecedented only a few years ago. The availability of commodity servers run by external providers now means that computing power that was only accessible to the largest of research institutions is now in the hands of smaller groups and even individual researches. Simulations that may have taken months to complete can now be run in days or hours. Large models with high memory requirements that would have needed to queue on a large cluster system can now be rented out from cloud service providers for a few dollars per hour.

There is, however, a downside. The ability to run more accurate models faster and at higher resolution results in a corresponding increase in simulation output. Where researchers may once have dealt with outputs in the hundreds of megabytes, this explosion in computing power means that datasets in the hundreds of gigabytes or even terabyte range are not uncommon. This exponential rise in data production, known as 'big data', is changing both the commercial and scientific landscapes. As such, the tools and techniques that were once used for analysis are no longer suited to the task and new approaches must be found.

This paper looks at one such scenario where approximately 60 000 simulations were generated from five base simulations that were built by hand. APSIM contains a lot of flexibility to specify how much output is generated, however this project needed daily output over 60 years for a number of variables so the raw output of these runs was around 100 gigabytes (5.5MB per file). While the simulator used here was the Agricultural Production Systems Simulation (APSIM), the described method can be used by any simulator with a textual input and output. Similarly, the simulations that were generated were created in the C# programming language, the method is language agnostic and can scale to any number of simulations.

2. CREATING THE BASE SIMULATIONS

The first step in any large scale simulation set is to plan all factors and levels. The purpose of the demonstration analysis used here was to visualise photo thermal quotient (Silva et al., 2014) across Australia at different time periods for different crops and cultivars. The factors used were crop, cultivar, sowing time and site. There were five crops, three cultivars (nine were used for wheat to better simulate the geographic segregation of varieties), five sowing dates and 660 sites around Australia. The sites were taken from the ApSoil data base (ApSoil, 2013).

Figure 1. Factor hierarchy that generated 49 500 combinations. Extra cultivars were used for wheat giving a total of just under 60 000 simulations.

Once the problem domain has been specified and appropriate factors planned, a number of base simulations should be prepared that capture all information that will remain static across factor levels. The method used for this analysis involved creating one complete simulation for each of the highest factor levels. While not strictly necessary, this approach was taken to simplify processing during the analysis. Splitting a large factorial analysis into smaller batches also has benefits for processing runtime as will be shown below. One base simulation was built by hand for each level of the highest factor which was the crop in this case.

The base simulations contain everything needed for a single combination. To ensure they are complete, a combination was built as though it was a standalone simulation and run to ensure it completed successfully. As five base simulations were being used for the five crops, this simulation was copied then modified to run each of the crops. Note that the only important part here is that the base simulations run successfully. The actual output does not matter. Since the base simulations will not form part of the factorial no effort was made to ensure that they were spatially correct. For instance, weather and soil data were not parameterised for any of the sites that were being modelled; they were just left at their defaults.

3. CREATING THE FACTORIALS

An APSIM input simulation file is a simple XML structured text file.

```
<folder version="36" creator="Apsim 7.7-r3601" name="simulations">
  <simulation name="WheAxe">
    <metfile name="met">
      <filename name="filename" input="yes">8001.met</filename>
    </metfile>
    <clock>
      <start_date type="date" description="Enter the start date of the
          simulation">01/01/2005</start_date>
      <end_date type="date" description="Enter the end date of the
          simulation">31/12/2013</end_date>
    </clock>
    <summaryfile />
...
  </simulation>
</folder>
```

Figure 2. A sample from an APSIM input file showing the XML structure.

Once the base simulations were running, they were modified so that a C# program could replace tokens in the file with actual factorial parameters. Where the factor to replace was a complete XML element, no change was made. The C# XML engine will replace the entire element. Where the parameter is part of an element, a token was manually inserted as shown in figure 2.

```
[EventHandler] public void OnInitialised()
{
   reportGS31 = (Outputfile) MyPaddock.LinkByName("reportGS31");
   reportGS65 = (Outputfile) MyPaddock.LinkByName("reportGS65");
   town = "@TOWN";
   trial = "@TRIAL";
}
```

Figure 3. A portion of an XML element (C# script) where placeholder tokens (highlighted) are being used.

While either method can be used alone, this hybrid approach strikes a good compromise between speed and simplicity. Using only the XML element replacement method means that large elements need to be repeated in their entirety which can be cumbersome for large elements such as manager scripts (especially where two or more parameters need to be changed as above), while using only the token replacement approach means that each parameter needs to be manually modified to accept a token which can be slow when there is a large number of parameters. Once the base simulations were prepared, a C# program was used to read in all factor combinations and create simulations by replacing tokens and elements.

Using the XML functions in C# it becomes very easy to replace individual elements or even entire blocks. In this case there were five tokens and seven elements to be replaced, one of which (the soil) had multiple child nodes. The tokens were replaced first by converting the entire XML document to a string, doing the replacements then converting the string back to an XML document. Next, the elements were replaced by retrieving the element from the document then changing its value. Since the element value is passed by reference there is no need to write it back to the document. The process for changing the most complex element, the soil, is shown in figure 3.

```
// Get the soil from the simulation using built in APSIM XML
functions.

// These are standard .NET XML functions customised for APSIM
structures.
XElement Soil = Simulation.Descendants("Soil").ToList()[0];

// Get a new Soil from the ApSoil database (read elsewhere).
XElement NewSoil = XElement.Parse((string)Row.Cells["SoilXML"].Value);

// Set the name of the element and replace the old one.
NewSoil.FirstAttribute.Value = "Soil";
Soil.ReplaceWith(NewSoil);
```

Figure 4. The code used to replace a multi-child XML node.

These four lines are all that is needed to replace any element in the simulation. Even a complex multi-child node such as the soil is very simple to replace. Similar code can be used for any language with an XML parser.

4. ANALYSING THE DATA SET

The purpose of the analysis was to create maps showing the intensity of a number of variables at various sites across Australia. 660 sites from the ApSoil database (ApSoil, 2013) were analysed and this data was interpolated to create the final graphic. The plotted region is the Australian wheat belt (Zheng et al., 2013).

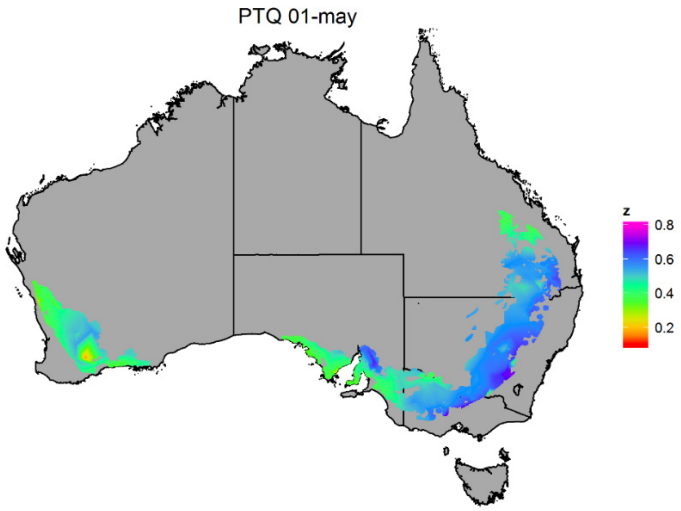

Figure 5. An example of one of the maps created for the analysis showing the photothermal quotient, which is the ratio of solar radiation to mean temperature, for a crop. Units are MJ m^{-2} d^{-1} °C^{-1}.

Each graph was generated from approximately 4GB of raw data. There were 5 crops and each crop had 5 graphs for a total of 100GB of simulation output data.

4.1 Preparing the simulation output

As was mentioned, the data was split by crop into batches and each batch was analysed individually with its own R script. While the entire data set could have been done in a single batch, having multiple smaller batches made for simpler scripts and faster runtimes due to the way R handles memory management

(Sridharan et al., 2015). Data was loaded into R as a data frame which is an object that can contain multiple rows and columns of different data types, similar to an Excel table or a group of vectors where each vector is of one data type and all have the same length. Note they are not a list of vectors as a list is a different structure in R. Analysis run time for each crop was approximately 24 hours. Most of this time was spent in two lines that used the ddply function in the plyr package (Wickham, 2011) to split the initial very large (~ 60 million rows) data frame into individual combinations then run a function on each. First, the data is split into groups according to one or more variables similar to an Excel filter or SQL 'GROUP BY' statement. Each group is then treated as a separate data frame that has a user defined function applied to it. Finally the results of each grouped calculation are bound together into a final d. The number of combinations plyr was splitting and combining was over 108 000. The amount of memory management this required resulted in very inefficient processing. This was later reduced to around 2 hours by splitting the dataset before processing and looping over other factor levels then joining the resultant data frames once all loops were complete. By reducing the number of combinations in each call to ddply, extra memory management is avoided which drastically improves run time.

4.2 Creating the APSIM Package

APSIM output files are not standard tabulated data. Figure 5 shows that the actual data is preceded by a number of constants, units and headings.

```
ApsimVersion = 7.7
Title = HGriffith
           Date        biomass          yield
   (dd/mm/yyyy)        (kg/ha)        (kg/ha)
     10/11/2008        18278.6         6983.4
     09/11/2009        16263.5         6057.9
...
```

Figure 6. The first few lines from an APSIM output file showing extra data before the simulation output.

As such, the usual data import functions such as read.table will not work. Additionally, the units and header rows must be read separately. The number of constants (key = value above) can also change. The development of the loadApsim function went through a few iterations to address these conditions and optimise for the best speed.

While the number of constants and the line where the data starts is unknown, the order they appear in is not. Constants will come first, then the header row, units and finally data. Constants will always contain an equals (=) sign. In the first iteration of the function, the entire file was read line by line. A cascading if..else statement was used to capture the correct data in each line. When the output table was reached, it would continue reading line by line and bind everything together at the end. This process turned out to be very slow as it did not use the optimisations available when reading large blocks of data.

Since the constants and header information is a very small portion of the entire output file, a second approach was taken where the file was read line by line as before until the start of the tabular data was reached. A counter kept track of this point and read.table was then used with the 'skip' parameter to read the rest of the file. Constants were then bound to the data frame which was added to a list which used data.table::rbindlist to create a final bound data frame after all files had been read. This process sped up the reading of multiple files considerably especially when the files were large. A file handle was kept open for this process to avoid the overhead of opening and closing files for each read operation.

An important point to note is that each individual file was read into a list of data frames first. Binding each data frame one at a time using rbind (which works by taking all the rows of a dataframe and appending it to the bottom of another one) is both computationally and memory intensive due to the amount of checking and resizing that R needs to do (Wickham, 2014). Since all the data is already loaded into the list, we know how

much space to allocate to the final data frame. We can then use data.table::rbindlist and use optimisations provided by the data.table package to bind much faster.

4.3 Benchmarking and Optimisations

A number of benchmarks were run during the writing of the APSIM output reader to explore different methods and identify the most efficient. The original pseudocode for the function is shown in Figure 6:

```
For each file to be read:
       While NOT end of file, read a line:
       Trim all white space
              If the string "factors = " is present
              # This indicates the output files came from a factorial APSIM
              # run and the user has chosen the 'Use a single line' option
              # for factor level output.
              Split the line by ';' to separate the factors. Unlist and add each
              vector to constants

              Else if the line contains '=' it is a constant:
                     Split the line by '=' and add to constants

              Else the next two lines will be column headers and units:
                     Split the lines, remove extra whitespace and store in vectors

              Everything after this is data so store this line number and break from
              the while loop
       End While
       Use read.table to read in the data from the stored line number
End For
Add constants as columns
Add column names
Use data.table::rbindlist to merge all files into a single data table
Return data frame or table according to user choice
```

Figure 7. Pseudocode for the APSIM output reader.

This original iteration took 14.55 seconds to load one thousand 10KB files. Table 1 shows the optimisations that were made.

Table 1. Effect of optimisations on run speed. Times are cumulative.

Optimisation	Run Time (s)
Base (no optimisation)	14.55
Unlist(…, use.names = FALSE)	13.95
Remove trim (only trim when required)	7.78
Strsplit(…, fixed = TRUE)	3.87

Removing the trim statement had the largest effect on runtime. It was determined that trimming every line was not required as any extra elements caused by white space could be removed more efficiently by sub setting the resultant object. Moving to read.table for the data portion also negated the need for trimming each line. The next largest effect was that of splitting strings into elements. By default, strsplit will use a regular expression to do the splitting even if a literal character or string is given.

Table 2. Run times for various options in two string manipulation functions. Numbers in brackets are relative to the default options in the base package.

String Splitting Optimisations	strsplit {base} Run Time (µs)	str_split {stringr} Run Time (µs)
Regex using literal space	37.75 (1.00)	45.39 (1.20)
Regex using whitespace delimiter '\s'	47.04 (1.25)	48.42 (1.28)
Fixed search for literal space	4.73 (0.12)	53.77 (1.42)

More subtle optimisations include precompiling the function using the compiler::cmpfun function which resulted in an overall 11% performance increase, and a 313% increase using square bracket notation for sub setting data frames instead of the subset function when working with very large data frames.

The final benchmarks for the optimised function are shown in Table 3. Benchmarks were run on an Intel Core i5 CPU at 1.9GHz with a solid state drive.

Table 3. Benchmark times for reading different numbers of files at different sizes. Numbers in brackets are size of files in bytes.

	10K (10 077)			100K (100 152)			1M (1 000 215)			5M (5 000 301)		
Number	**Time**	**SD**	**MB/s**	**Time**	**SD**	**MB/s**	**Time**	**SD**	**MB/s**	**Time**	**SD**	**MB/s**
10	0.082	0.012	1.23	0.181	0.02	5.53	1.004	91.10	9.95	4.4	0.055	11.32
100	0.404	0.04	2.5	1.4	0.08	7.14	9.9	164.87	10.14	44.6	0.293	11.20
1000	3.9	0.188	2.58	14.5	0.685	6.91	108.2	5.9	9.24			
10000	42.4	1.814	2.38	156.7	5.8	6.39						

5. CONCLUSION

The ability to generate large data sets brings with it the requirement to analyse that data in a timely manner. The first obstacle to analysis is the importation into an analysis pipeline. The R package 'APSIM' presented here provides a simple, fast and flexible interface for the researcher to import their data in a format that is easy to manipulate and does not require pre-processing. It can read data at speeds in excess of 11MB/s (dependant on size of individual data files and speed of CPU and storage) and provides functions for creating simulation input weather files. Large files read faster so care should be taken to ensure that fewer, longer simulations are run where possible. For example, run long multi-year simulations with soil resets instead of multiple single-year runs.

The R APSIM package is under continuous development and future releases will include the ability to read directly into databases for data sets that are too large to fit in memory. Support for APSIM Next Generation output is also planned.

REFERENCES

ApSoil (2013). "APSoil." Retrieved 05 July 2015, from https://www.apsim.info/Products/APSoil.aspx.

Holzworth, D. P., N. I. Huth, P. G. Devoil, E. J. Zurcher, N. I. Herrmann, G. McLean, K. Chenu, E. J. van Oosterom, et al. (2014). "APSIM - Evolution towards a new generation of agricultural systems simulation." *Environmental Modelling & Software* **62**: 327-350.

M Dowle, T. S., S Lianoglou, A Srinivasan, R Saporta, E Antonyan (2014). data.table: Extension of data.frame. Retrieved 05 July 2015 from http://CRAN.R-project.org/package=data.table.

Sadras, V. O., L. Lake, K. Chenu, L. S. McMurray and A. Leonforte (2012). "Water and thermal regimes for field pea in Australia and their implications for breeding." *Crop & Pasture Science* **63**(1): 33-44.

Shriram Sridharan, J. M. P. (2015). Profiling R on a Contemporary Processor. *41st International Conference on Very Large Data Bases*. Hawai'i Island, Hawaii, VLDB Endowment. **8:** 173-184.

Silva, R. R., G. Benin, J. A. Marchese, E. D. B. da Silva and V. S. Marchioro (2014). "The use of photothermal quotient and frost risk to identify suitable sowing dates for wheat." *Acta Scientiarum-Agronomy* **36**(1): 99-110.

Team, R. D. C. (2011). R: A language and environment for statistical computing. Vienna, Austria, R Foundation for Statistical Computing.

Wickham, H. (2011). "The Split-Apply-Combine Strategy for Data Analysis." *Journal of Statistical Software* **40**(1): 1-29.

Wickham, H. (2014). Memory. *Advanced R*, Chapman and Hall/CRC**:** 377-394.

Zheng, B. Y., B. Biddulph, D. R. Li, H. Kuchel and S. Chapman (2013). "Quantification of the effects of VRN1 and Ppd-D1 to predict spring wheat (Triticum aestivum) heading time across diverse environments." *Journal of Experimental Botany* **64**(12): 3747-3761.