

Scalable S: A Prototype Description of a Fast, Scalable Implementation of the S Language

Blair Christian
9 Feb 2007

Intended Audience: R/S users of large datasets, numerical analysts, interested HPC parties, R to C compiler group

Introduction

Disk space, memory and time are increasingly common constraints in applied statistics. The size of datasets is growing at a fast pace, yet the computational tools needed for this type of analysis are not growing at a sufficient pace. This document is an attempt to describe the incorporation of 3 ideas to improve this situation: 1) Having a standard language for describing a statistical model, as in Ch 2 of "Statistical Models in S" 2) Having a fast implementation (eg compiled and not interactive) of the S language which 3) uses ScaLAPACK to create a layer of transparency to problem size. Thus, a user should be concerned with creating a statistical model and analyzing its output, not having to worry about the computational platform and details underlying its implementation.

Choice of Model Standard

The initial focus of this project is getting a scalable implementation of the linear model (lm in Chambers) implemented, with future growth planned in generalized linear models and other topics of interest. The modeling language of Chambers and Hastie is chosen because it is a proper syntax for expressing statistical models in a concise fashion. In short, their model language was built exactly for the purpose of describing the computational implementation of the statistical model. Further, the S language is available in an open source project, the R project, <http://www.r-project.org>, which has a large usage and many contributed libraries.

Implementation details

The implementation of this modeling standard on a large scale involves two main parts: 1) a fast implementation of R and 2) a set of parallel algorithms to perform the necessary matrix operations. Fortunately versions of both are currently available in the form of 1) the R to C compiler (RCC) at Rice, <http://www.hipersoft.rice.edu/rcc> and 2) the ScaLAPACK project, <http://www.netlib.org/scalapack>. The *modus operandi* is as follows. First, without loss of generality, I will describe a message passing interface (MPI) approach, since I am familiar with that.

The user begins with a basic R command, such as `lm(y ~ x:log(z), data=bigData)`. The storage component of the data is not relevant now and is discussed later. At this point, assume that we have n nodes available for use, which, when combined, are ample for solving the desired regression. To get an idea of the scale, if we are dealing with a few gigabytes of data, perhaps n is less than 10, if we are dealing with terabytes of data, perhaps n is more than 100. This script is submitted to a wrapper `scalableR`, which uses metadata of the rough size of the data involved and relevant internal conversions (eg transforming categorical variables into the data matrix), roughly estimating the number of processors necessary to complete the job given constraints on time and size of the problem. More specifically, the first constraint is size, as we have memory limitations at individual nodes and will have a lower bound on the number of nodes required to perform the matrix operations. Second and less importantly, we may be interested in constraining the time in which it runs, which may increase the lower bound on the number of nodes needed. Thus, we have a script and an estimate of the amount of resources needed to solve the problem.

At this point, it is necessary to quickly solve the linear model (eg providing estimates and standard errors). At this point, it is not clear if one should compile an individual R script (ala RCC) and run it as compiled code; or if a wrapper can be called from within an interactive R session which calls a pre-prepared . Perhaps `scalable.lm()` would lead to an appropriate naming convention. At this point the compiled code, having been translated to C with MPI calls, is now called. The compiled code will be solving the linear model in a parallel fashion, as seen in the example below.

Example of LM and ANOVA

Suppose we have y and x which are too big for one machine to solve- again, solution here in the statistical sense- estimates and standard errors. In the `lm` case, we have a dense design matrix X which sets up the linear system solving for the estimate b such that it is the least squares solution of $y-Xb$. Assume for now that we have an overdetermined system. Typically this is solved by a QR algorithm, and in this case, a parallel implementation of the QR algorithm, `PxGEQRF`, would be called. Then we need to estimate the standard errors, given that we let $X=QR$, we must invert $R'R$. I know that there's an `ScaLAPACK` command to invert the cholesky decomposition of a matrix efficiently, and it is described in an R package `RScalAPACK`, <http://cran.r-project.org/doc/packages/RScalAPACK.pdf>. If we were running directly in R and had an MPI cluster callable from R, we could use `RScalAPACK`. If not or if we wanted to compile the full R script, hopefully the call will be compiled in a way that will maximize the use of `RScalAPACK` or some other currently available libraries. Solving an ANOVA could proceed in a similar way, but using a parallel implementation of a sparse QR decomposition.

Data Format

The data format should not be an obstacle, and it is assumed that the data resides in a very large database somewhere, which can be accessed by the n nodes. Given the examples above, it is not clear which tools are currently available to work with matrices too large for a single machine. An implementation that passes a SQL call for the correct submatrix needed on a given node seems most appropriate, but the ultimate design may need to be customized for the individual circumstances.

Concerns, Limitations and Unanticipated Problems

Here is a list of off the cuff problems that have come up so far:

- Random number generators- the streams used on individual nodes must be constructed to be uncorrelated
- What is the best implementation strategy- An `S4` approach within R, an `RCC` approach, or is this something so specific that I shouldn't be thinking about these kind of details
- What issues will come with categorical variables, how low counts should be treated (eg allow categorical variables with large numbers of factors, or provide a mechanism for reducing/limiting the number of factor levels)
- How do we deal with missing data?
- Which things are not truly parallelizable (eg median), and how does one bootstrap over multiple machines
- What alternative implementations would be more efficient/appropriate? Something `JINI` based?
- What techniques will be most important to have in the toolbox- eg model selection, perhaps a parallel implementation of something like `LARS`?

Conclusions

The need for a more seamless access to a scalable `S` language is needed; there are current approaches, but nothing that is close to a final solution. Perhaps it is naive to try and map out such an ambitious project when technology is changing so quickly, but perhaps it is time to try and establish some standards that allow for reproducible research when using very large datasets. This situation has been arising more and more frequently in my work and I anticipate that the need for these types of tools is only going to grow with time. Perhaps it might be useful to have a dedicated server setup (eg on the Rice tera-scale cluster) that is maintained by someone, and allow remote jobs to be run for a reasonable fee.

Bibliography

```
@book{chambers1991sms,  
  title={{Statistical Models in S}},  
  author={Chambers, J.M. and Hastie, T.J.},
```

```

    year={1991},
    publisher={CRC Press, Inc. Boca Raton, FL, USA}
}

@article{gropp1996hpp,
  title={{A High-Performance, Portable Implementation of the MPI Message Passing
Interface Standard}},
  author={Gropp, W. and Lusk, E.L. and Doss, N. and Skjellum, A.},
  journal={Parallel Computing},
  volume={22},
  number={6},
  pages={789--828},
  year={1996}
}

@book{gropp1999ump,
  title={{Using MPI: portable parallel programming with the message-passing
interface}},
  author={Gropp, W. and Lusk, E. and Skjellum, A.},
  year={1999},
  publisher={MIT Press}
}

@book{blackford1997sus,
  title={{ScaLAPACK User's Guide}},
  author={Blackford, L.S. and others},
  year={1997},
  publisher={SIAM Philadelphia}
}

@article{choi1992ssl,
  title={{ScaLAPACK: a scalable linear algebra library for distributed
memoryconcurrent computers}},
  author={Choi, J. and Dongarra, JJ and Pozo, R. and Walker, DW},
  journal={Frontiers of Massively Parallel Computation, 1992., Fourth Symposium on
the},
  pages={120--127},
  year={1992}
}

@article{yoginath2005rhp,
  title={{RScalAPACK: High-Performance Parallel Statistical Computing with R and
ScaLAPACK}},
  author={Yoginath, S. and Samatova, NF and Bauer, D. and Kora, G. and Fann, G. and
Geist, A.},
  journal={Proc. 18 thInt'l Conf. on Parallel and Distributed Computing Systems},
  pages={12--14},
  year={2005}
}

```