Data Mining by Scaling Up Open Source Software

Brian D. Ripley and Fei Chen
University of Oxford, Department of Statistics
1 South Parks Road
Oxford OX1 3TG, UK
{ripley,feic}@stats.ox.ac.uk

Data Mining is a modern buzzword for finding (useful) information from large (often massive) databases. If has also been an opportunity for the re-invention (or re-marketing) of many ideas from statistics and machine learning, and the marketing of many commercial programs for 'data mining'. In the words of Witten & Franke (2000, p. 26)

> What's the difference between machine learning and statistics? Cynics, looking wryly at the explosion of commercial interest (and hype) in this area, equate data mining to statistics plus marketing.

We are interested in applying good statistical practice to large datasets. This study has been motivated[1] by modelling of insurance databases. A modest database might contain a million insurance proposals as well as the subsequent claims records, and motor insurance databases exist[2] with over 60 million records. These are modest numbers alongside CRM[3] applications of data mining, but do have a clearly defined goal: a better formula for setting insurance premiums. They are not modest numbers for standard statistical software, and we are exploring how Open Source software can be adapted to meet the challenge. We have been working with the statistical system R and the database management systems MySQL (Axmark et al., 2002) and PostgreSQL.

The anatomy of statistical software

Most statistical packages today are monolithic systems. The statistical software alone is responsible for all tasks related to data analysis, from data management to data visualization. As data set size grows, it is sometimes difficult to extend these systems to meet the new computational needs. Although 1Gb memory is not uncommon these days, it is still a taxing job for R to fit models such as a generalized linear model on data sets of the sizes sketched in the introduction.

Another limiting factor probably is network bandwidth. Large data sets tend to be stored in databases. To fit a model using R usually means that the data has to be transferred across a network into R. As a result, computation time is dominated by the overhead of data transfer for large data sets.

An alternative is to take as much as possible out of the monolithic system and to design a modular system where different tasks run as separate processes on different processors (servers) outside the statistical package (the statistics server).

We often find when fitting a statistical model that there are two types of computations, each involving different amounts of data. One type requires the entire data set; the operations performed on the data tend to be primitive, usually involving some sort of summarizing operation. The other type requires more complicated manipulation of summary results, but the amount of data required is small.

For example, in optimization we want to maximize some objective function, $\max_\beta \mathcal{L}(\beta, X)$, that depends on some parameter vector β and some data X. This is usually solved iteratively, and at time step $t+1$ we use an iteration $\beta^{(t+1)} = \beta^{(t)} + U(X, \beta^{(t)})$. Here U is a correction factor defined by a particular algorithm and the objective function $\mathcal{L}(\cdot)$. The data-intensive part of the algorithm lies in calculating U. In all but the simplest algorithms, the calculation involves multiple steps; some of

[1]Fei Chen's D. Phil studies have been sponsored by EMB, a firm of insurance actuaries.
[2]The State Farm Insurance Company in the U.S.A has more than 66 million insurance policies.
[3]Customer relationship management: stores tracking customers via 'loyalty cards'.

them require X, some of them are just a simple update of intermediate results. The parts that require X are often fairly straightforward such as computing the objective function or its derivatives. For example, the quasi-Newton method with BFGS update has the form

$$\beta^{(t+1)} = \beta^{(t)} + \left(-\frac{\beta^{(t)} S S^T \beta^{(t)}}{S^T \beta^{(t)} S} - \frac{G G^T}{G^T S} \right),$$

where $S = \beta^{(t+1)} - \beta^{(t)}$ and $G = \boxed{\mathcal{L}'(X, \beta^{(t+1)})} - \boxed{\mathcal{L}'(X, \beta^{(t)})}$. As we can see from the framed boxes, the data-intensive part of the algorithm is straightforward (if the derivative is known).

Computing near the data

This observation suggests a distributed computing idea where we perform the data-intensive but algorithmically less sophisticated operations on the entire data set near where this data set is located The summary results (such as the value of an objective function or its derivatives) are sent back to the server responsible for the overall algorithmic flow, e.g. constructing the correction factor U.

We have been using a component-based design where the responsibilities of model fitting, data management and computation are delegated to different processors. A database system is responsible for managing data, including creating model matrices. Low level linear algebra routines are computational tasks. Communicating model formulas, managing algorithm flows as well as inter-system communications during algorithm execution are all parts of model fitting.

The system consists of three main parts (see the figure): a statistics server, a data server and a computing server. The computation server is a collection of processors awaiting jobs submitted from the central server-the statistics server. The statistics server is essentially an overarching "operating system", which, by communicating with a server that embeds R, dictates how the data server and the computation server are to interact with each other and how the computation server is to finish a computational task.

What do we gain by distributing the computations? There is a large gain by running computations local to the database and so avoiding transferring large datasets across the network. It is also convenient, and perhaps necessary, for the analysis to be run from the data analyst's local workstation. Our style (in common with many data analysts) is to look at a large number of graphical displays, and performance considerations often demand the graphical software to run on the local machine: we are envisaging applications in which the end user and the database server are connected over a wide-area network.

Whether we need an additional pool of computation servers depends on the algorithm and the hardware available. We are envisaging the use of algorithms such as the fitting of neural networks in settings in which the database will reside on a relatively slow server. In our current implementation this is not really exploited: we think of it as 'future proofing' the design.

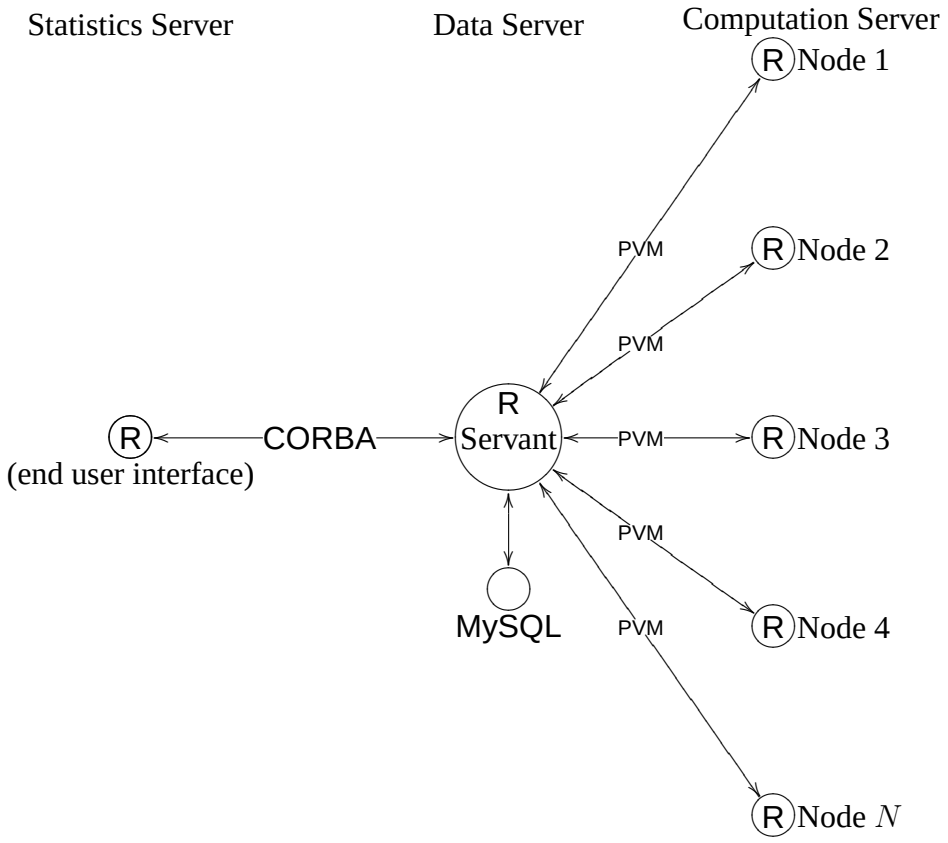
The requirements that we have of a distributed computing system are simple.

1. We would like to assume that the end users operate in only one language, S, and thus avoid having to issue SQL-like statements such as `select x from X` from within R.

2. We would like to have transparency of computation from the perspective of the end user; that is, we would like distributed computing to take place automatically as much as possible.

3. We need a system that can handle data sets of the scale of the introduction.

Computing near the data



This figure illustrates the idea and implementation of performing computation where the data resides. The end user interface is accessible through the Statistics Server, R, which sends out job requests for statistical analysis via a CORBA interface. The execution of an analysis actually takes place on the data server, MySQL, through a CORBA servant running embedded R. The servant thus understands instructions sent by the statistics server and is able to execute any R code. The computing server, a cluster of processors each also running an embedded R process, is available to the servant to speed up computation using PVM running on a N-node process grid.

A few of the details

The system Fei Chen has constructed is based on modifying the R version 1.6.x sources. In order to hide details of SQL from the end user, we decided that tables in databases should behave like data frames. This means we first need to have a method in R to refer to these tables. Our solution is to implement methods to attach MySQL databases as well as search paths of other (remote) R sessions. In the first case, all data tables residing in the database are declared as external pointers.

Making references from R to database tables is the first step toward transparent distributed computing. The next functionality we needed to provide is to make external pointers behave as if they were R data frame objects. Then we can do things like

```
R> # initialization omitted
R> ls(corba:database:mysql)
[1] X Y Z
R> X[[1]]
CORBA: executing "X[[1]]" and returning
[1]  1  2  3  4  5  6  7  8  9 10
```

```
R> lm(x ~ y, data=X)
CORBA: executing "lm(x~y,data=X)" and returning
...
```

with data residing in the MySQL database.

Once all the pieces are in place, we can use code as follows (although obviously with a user-friendly wrapper). The response and the model matrix are computed near the database, and the weighted-least-squares fit is done on a 2×3 of computational nodes.

```
R> corba <- init.corba()
R> gridinfo <- init.grid(nprow=2, npcol=3, mb=2, nb=2)
R> eval.corba("db <- init.mysql()", corba)
R> eval.corba("attach.mysql(db)", corba)
R> attach.corba("database:mysql", corba);
R> ls(corba:database:mysql)
[1] X Y
R> names(X)
[1] "x1" "x2"
R> names(Y)
[1] "y"
R> fam <- poisson()
R> mustart <- Y + 0.1
R> eta <- fam$linkfun(mustart)
R> mu <- fam$linkinv(eta)
R> mu.eta.val <- fam$mu.eta(eta)
R> z <- eta + (y - mu)/mu.eta.val
R> w <- sqrt((mu.eta.val^2)/fam$var(mu)
R> mod <- terms(y ~ x1 + x2)
R> Xmod <- distribute.modelmatrix(X, mod, gridinfo)
R> Z <- distribute.scalapack(z, gridinfo)
R> W <- distribute.diag.scalapack(w, gridinfo)
R> XWmod <- pdgemm.scalapack(W, Xmod, gridinfo) # sqrt(w)%*%X
R> fit <- lm.fit.scalapack(XWmod, Z, gridinfo)
R> beta <- collect.scalapack(fit, m=3, n=1)
... repeat until convergence
```

REFERENCES

Axmark, D., Widenius, M., Cole, J., Lentz, A., and DuBois, P. (2002) *MySQL Reference Manual*, `http://www.mysql.org`.

Blackford L.S. and 11 others (1997) *ScaLAPACK Users' Guide*, SIAM.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. (1994) *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press.

Witten, I. H. and Franke, E. (2000) *Data Mining. Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann.

RÉSUMÉ