BoscoR: Extending R from the desktop to the Grid

Derek Weitzel*, Jaime Frey†, Marco Mambelli‡, Dan Fraser§, Miha Ahronovitz¶, David Swanson*

*Computer Science & Engineering
University of Nebraska – Lincoln
Email: [dweitzel, dswanson]@cse.unl.edu

†Department of Computer Science
University of Wisconsin
Email: jfrey@cs.wisc.edu

‡Fermi National Accelerator Laboratory
Email: marcom@fnal.gov

§Argonne National Laboratory
Email: fraser@anl.gov

¶Ahrono Associates
Email: miha.ahronovitz@ahrono.com

***Abstract*—In this paper, we describe a framework to execute R functions on remote resources from the desktop using Bosco. The R language is attractive to researchers because of its high level programming constructs which lower the barrier of entry for use. As the use of the R programming language in HPC and High Throughput Computing (HTC) has grown, so too has the need for parallel libraries in order to utilize computing resources.**

Bosco is middleware that uses common protocols to manage job submissions to a variety of remote computational platforms and resources. The researcher is able to control and monitor remote submission from their interactive R IDE, such as RStudio. Bosco is capable of managing many concurrent tasks submitted to remote resources while providing feedback to the interactive R environment. We will also show how this framework can be used to access national infrastructure such as the Open Science Grid.

Through interviews with R users, and their feedback after using BoscoR, we learned how R users work and designed BoscoR to fit their needs. We incorporated their feedback to improve BoscoR by adding much needed features, such as remote package management. A key design goal was to have a flat learning curve in using BoscoR for any R user.

I. Introduction

Usage of the R language [1] by data miners has grown much faster than any other programming language [2], [3]. Data mining requires computational resources, sometimes more computational resources than can be provided by their desktop. In a recent study [2], "Available computing power" was the second most common problem for big data analysis. In addition, the respondents stated that distributed or parallel processing was the least common solution to their big data needs. This could be attributed to the difficulty of processing data with the R language on distributed resources, a challenge we set out to solve with BoscoR.

A reason that distributed computing is not seen as a popular solution to big data processing is that scientists are more familiar processing on their desktop than in a cluster environment. R is typically used by people that have not used distributed computing before and do most of their analysis on their local systems with IDEs such as RStudio [4]. Users are unaccustomed to the traditional distributed computing model of batch processing in which there is no interactive access to the running processes.

Though researchers may not have experience with distributed computing, most have computational resources available to them, either locally provided by their institution or university, or through national cyberinfrastructure such as the OSG [5] or XSEDE [6].

In this paper, we will describe the background of the two primary components combined to create BoscoR, Bosco [7] and GridR [8]. In section III, we describe how we combined these two components to create a fault tolerant framework that provides a positive user experience. Next, in Section IV, we discuss preferred submission methods based on the length of the R function, and show some results from numerous test runs against a production cluster. Finally, we offer some conclusions and future work.

II. Background

BoscoR is primarily made up of two components, Bosco and GridR. Bosco provides an simple to setup interface to the remote batch system, and provides fault tolerance for job submission and file transfers. GridR provides a user interface to create and initiate remote processing.

A. Bosco

A number of different methods have been used to distribute jobs across multiple resources. Inside a cluster, schedulers such as HTCondor [9] and PBS [10] have been used. Neither of these schedulers have been used to submit jobs from user's desktops, a key requirement in improving the user experience of R users. Remote submission is heavily used in computational Grids, and uses technology such as Globus [11] and UNICORE [12]. This remote submission requires software installation on a server that is inside the cluster which required an administrator.

Bosco [7] is used to effortlessly create a remote submission endpoint on a cluster without requiring the administrator to install any software. The architecture of Bosco is shown in Figure 1. Bosco is a remote submission framework based upon HTCondor. It uses the SSH [13] protocol to submit and monitor remotely submitted jobs. Additionally, it performs file transfers using the same SSH connection.

Bosco is designed to be run without administrator intervention. It automatically installs the Bosco client on a remote

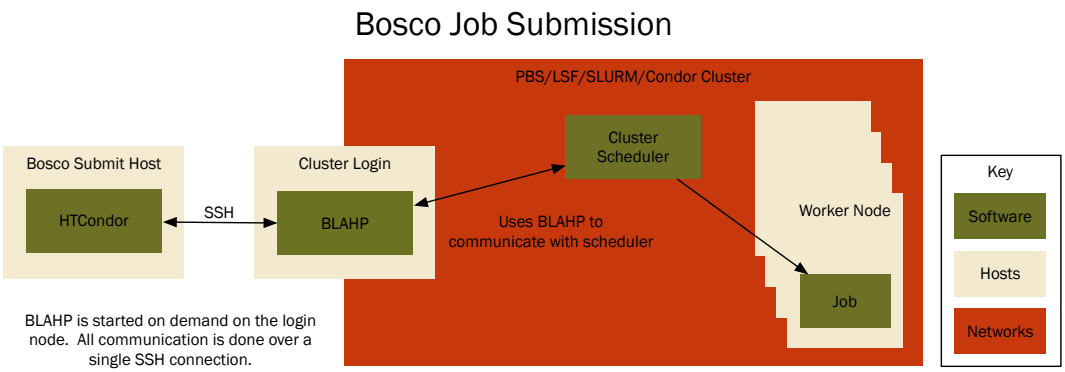

Fig. 1: Architecture of Bosco

cluster. SSH was chosen as the protocol since it is used nearly universally for cluster access. Jobs are submitted to Bosco, which then submits over ssh to the remote cluster. Input files are transferred over the SSH connection as well. Bosco then monitors and reports the status of the job on the remote cluster as idle, running, or completed. Once the job is completed, Bosco will transfer output files back to the submit host.

Bosco has two modes of job submission:

1) **Direct** – A single job on the Bosco submit host corresponds to a single job on the remote cluster. Each job is submitted individually to the remote cluster's scheduling system. This method is the simplest to run, and imposes no special requirements on the submit machine.
2) **Glidein** – Bosco submits many pilot jobs to the remote cluster using the Direct method. But, each of the pilots can service multiple user submitted jobs from the Bosco submit host. This method minimizes the overhead on the remote cluster since Bosco is not submitting many jobs through the cluster scheduler. The Glidein method requires that the Bosco submit host can be contacted from the remote cluster worker nodes. The Glidein submission method is based off of previously written software such as the Campus Factory [14].

The two modes of job submission allow users to optimize for their environment. If they are running many short identical jobs (which is frequent in High Throughput Computing), then the Glidein method is ideal for them. If they are running fewer, longer, and possibly unique requirement jobs, then the direct submission method would work best. Most users start with the direct method then graduate to glidein once they become accustomed to submitting batch jobs.

Bosco's fault tolerance is in it's handling of the remote cluster. For example, if the user's computer loses connection with the remote cluster due to network issues, or even if the user suspends their laptop, Bosco will place the jobs on hold. Although no new jobs will be submitted to the remote cluster, jobs that were already submitted will continue to run. Further, when Bosco re-establishes a connection to the remote cluster, Bosco will check the status of already submitted, bring back any output data from any that have completed, and continue to submit jobs to the remote scheduler.

B. GridR

Many parallel libraries are available for R. Most focus on managing the R processing on a single server such as the `parallel` package [15]. The `parallel` package comes bundled with R and provides for single machine parallelism. Parallelism is done by using variations of the R function `lapply`. A simplified definition of `lapply` is shown in Figure 2. `lapply` is the basis for nearly all parallel libraries in R.

`lapply`(**X**, **FUN**, **...**):

X	a vector (atomic or list) or an expression object.
FUN	the function to be applied to each element of X
...	optional arguments to FUN.
Returns	list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X.

Fig. 2: Funciton definition of `lapply`

The `lapply` function is ideally suited for high throughput computing. There is no communication between executions of the function on the array. The input vector can be easily partitioned in order to split the execution across multiple resources. It is because of these reasons that most parallel applications use the `lapply` model to provide parallelism. Examples are the parallel package which defines the functions `mcapply` (multi-core apply) and `parapply` (parallel apply). In the `parallel` package, calling `mcapply` causes R to fork a process that will execute the function **FUN** on each element in the input vector. A similar process happens when calling `parapply`.

Another built-in parallelization package is Simple Network of Workstations (snow) [16]. Snow allows for multiple computers to organize and execute parallel processing of data. The

computers communicate over regular network sockets or using MPI [17]. This allows for multiple computers inside the same cluster to process data. Snow also has built-in `lapply` style functionality.

GridR also follows this `lapply` model for parallelization. It uses a function called `grid.apply`, which will apply a function to every element of an input vector, similar to `lapply`. Instead of forking a process like `mcapply`, it compiles the input data and function, and submits the execution to a grid endpoint.

GridR was originally written for use with data analysis in ACGT clinico-genomics trials [8]. It was written with the capability to submit with a limited set of grid protocols, some of which are no longer supported. Further, GridR made assumptions of the remote resources. These assumptions were:

- R is installed on all of the worker nodes.
- The R binaries are in the same location on all of the remote resources.
- The GridR package is installed on all of the remote resources.

All of these assumptions cannot be met on modern grid resources. Applications cannot assume that a (non-standard) processing tool, such as R, is installed on every computer or that it is installed at exactly the same location on all clusters in the grid. Modifications were made to GridR to erase these assumptions, as well as to adapt it to submit to Bosco.

III. Implementation

Bosco is designed to run on resources that are not controlled by the submitting user. Further, it is designed to run on resources without any conditions as to what is installed. In order to operate under these assumptions, Bosco must bootstrap itself by bringing in all the libraries and dependencies required to operate. Therefore, BoscoR must run under these same assumptions.

GridR was modified to submit to Bosco. The input generation of GridR is shown in Figure 3. When a user or script calls `grid.apply`, GridR compiles the input function and input data into a R data file, which can be read later by another R process. GridR handles function dependencies by using R's dependency detection and compiles any functions that may be required into the input.

GridR was modified to first create a submit file which will be submitted to the local system where Bosco is installed. The submit file explicitly lists the input files and the expected output file, all of which will be transferred by Bosco. The input files, as shown in Figure 3, are the compiled function and input data and a bootstrap executable. The output file contains the return value from the executed function.

The submit and polling script is executed by GridR after forking a new R process. This is a light weight process that submits the Bosco submission script and watches for any errors. If the input function is executed many times by many separate jobs, the polling script will aggregate the results as they are returned to the submit node into a vector that will be returned to the user.

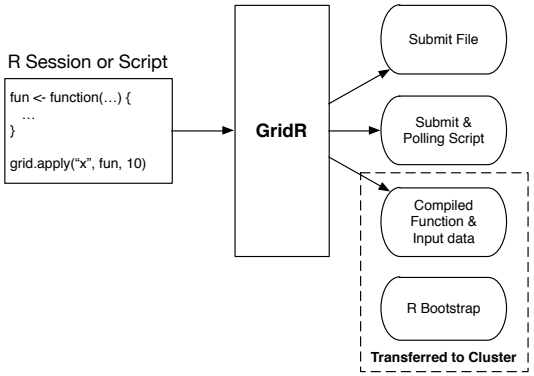

Fig. 3: GridR Input Generation

A. Bootstrap

Since Bosco cannot make assumptions as to what is installed on the remote cluster, neither can BoscoR. Therefore, GridR was modified to detect, and if necessary install, R on the remote system. This was performed by a bootstrap process.

In the GridR generated submit file, the listed executable to run on the remote system is not R, but the bootstrap executable. The bootstrap executable detects if R is installed on the remote system. If it is installed, it simply executes the user defined function against the input data. If R is not installed, the bootstrap downloads the appropriate version of R for the remote operating system. The supported platforms are identical to Bosco's, CentOS 5/6 and Debian 6/7. R is downloaded from a central server operated by the OSG's Grid Operations Center [18].

The bootstrap executable installs R in a shared directory. By utilizing a shared directory, subsequent GridR jobs may use the same R installation. Several bootstrap jobs could start at once on a remote cluster, so a simple transactional file locking mechanism was devised so only a single bootstrap executable on a cluster will download and install for the entire cluster if a shared file system is available. If a shared filesystem is not available for installation, R is installed in a temporary directory that is removed upon job completion.

B. Running on the Open Science Grid

Submitting to the Open Science Grid (OSG) is done by direct submission. The OSG hosts access nodes which can be used to submit to resources on the grid.

Most OSG sites do not have shared directories for grid users. Therefore, the bootstrap script must install R on every node in a temporary directory. In order to optimize the R installation, the bootstrap script utilizes the HTTP forward proxy infrastructure [19] available on the OSG to minimize requests to the central hosting server.

IV. Results

The results section is broken into two categories, results from user feedback and results from experimental runs on a production cluster as well as the OSG.

A. Results from feedback

A primary goal with BoscoR was to improve the user experience of using R on distributed resources. After acquiring a few users, we received feedback on how BoscoR could be improved. The improvements to GridR and the integration with Bosco made working with campus or institutional resources much better. In this section we will describe the improvements.

1) User Provided Packages: Many users require additional packages to be installed before their function can execute. It was assumed that most of these packages would be in the major R package repositories, such as the Comprehensive R Archive Network (CRAN) [20]. If the package is in CRAN, the user provided function can install the package. After receiving user feedback, it was found that not all desired packages are available in CRAN. A modification to both the submit file and the bootstrap executable was designed to install such packages.

In order to install a user provided package, it first needs to be transferred to the remote cluster. This is done by including the package in the list of input files to be transferred by Bosco for each job. Additionally, the packages should be installed before the user function is executed on the remote resources. This required modification to the bootstrap executable in order to install the packages after installing R, but before executing the user's function on the input data.

2) Quick Jobs: Since the GridR interface only provides for a single function to execute against the input data, it was assumed that the function would be a time consuming data processing function that may call many other functions. Therefore, the overhead Bosco introduces would not significantly effect the performance of the executions. After receiving feedback from users, it was determined that the more common use case is to use smaller functions that could execute in seconds or minutes. In order to accommodate shorter jobs, a modification to the GridR generated submit script was required.

In this case, we modified the submit file so that Bosco would use the Glidein method of job submission. Using the Glidein job submission method, the shorter jobs can be executed much more quickly, one after another, reusing the same resources. Additionally, this saves the remote cluster scheduler from scheduling many small jobs which can cause issues in many HPC schedulers.

B. Performance Results

1) Experimental setup: In order to test BoscoR, we have to simulate a R workload. We simulate a workload with varying lengths of the executed functions. This simulates a variety of workloads that we have seen from users. As noted before, we assumed that the functions would be long running data processing. But, we learned that users were instead submitting short functions to be executed quickly. We varied the length of function from 1 second to 30 minutes. To verify our solution to quick jobs, we tested different job lengths using both the Direct and Glidein submission method.

To execute the test jobs, we used the production cluster *Tusker* at the University of Nebraska – Lincoln Holland Computing Center. This cluster is composed of 106 nodes, each with 64 cores, for a total of 6784 cores. The cluster has numerous users that are submitting to the central SLURM [21] scheduler. The cluster traditionally runs at >90% utilization, with dozens of users jobs fair sharing the resources. The SLURM scheduler is a HPC orientated scheduler that matches submitted jobs to resources. Fair share scheduling is used on Tusker. Each group has equal priority with all others, therefore allowing the maximum number of users to run on the resources.

The Tusker cluster was chosen since access was easy for the authors of this paper. Further, it is utilized enough that the jobs would be competing against other user's jobs for available resources, and therefore not all submitted GridR functions would be able to execute simultaneously. We believe this best represents most clusters, which are typically highly utilized by many researchers. It is plausible that a cluster could be so utilized that no user jobs could be executed, while the other extreme could also be true, that enough resources are available for all submitted jobs to be executed immediately. We found that Tusker utilization is somewhere between these two extremes. It is capable of running many, but not all, jobs submitted to it immediately. The rest will execute as resources become available.

For our testing, we submitted 1000 GridR jobs per test run. 1000 jobs was chosen as a reasonable representation of workflows we have seen when helping users of GridR. They typically submit many jobs, sometimes reaching into the thousands. Our goal was to submit more jobs than could be run instantly by the remote cluster, but no so many that gathering repeated testing data would be impossibly time consuming.

2) Direct submission: Direct submission is defined as submission using Bosco's direct submission method. In GridR, the library is initialized with the argument `service="bosco.direct"`. When this setting is used, GridR generates submission scripts that use Bosco's default routing mechanism to submit to a single cluster. The GridR functions are submitted as jobs directly to Tusker's SLURM scheduler.

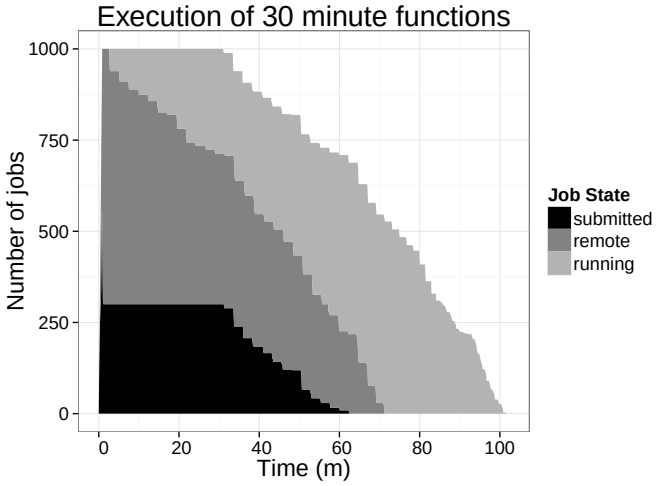

Fig. 4: Direct submission

A timeline of the GridR submissions to Bosco is shown in Figure 4. The submitted jobs are jobs which are submitted

locally, but not yet submitted to the remote scheduler. Remote represents the jobs which are submitted to the remote SLURM scheduler. The submission to the remote scheduler is very rapid. Bosco is able to submit 700 jobs, the limit in this implementation to submit at a single time, within a minute. SLURM is able to rapidly begin executing many, but not all, of the submitted jobs. Bosco maintains constant pressure in the form of idle jobs in case resources become available on the cluster. You will notice the straight line in the number of submitted jobs which dips after 30 minutes. At 30 minutes the first jobs begin to complete and Bosco begins to submit more jobs to the cluster, attempting to always keep the maximum of 700 jobs either idle or running on the remote cluster. In this workflow, all 1000 30 minute jobs finish in just over 100 minutes.

The Open Science Grid runs use the direct submission method as well. But, since the OSG access nodes run HTCondor, the jobs are capable of starting much quicker after being submitted to the remote resource. In this way, the OSG provides the best of both the direct and glidein approaches. It is simple to setup like any direct submission method. And as with the glidein submission method, the jobs start quickly on the remote resource.

The OSG direct submission presents different failure modes than a traditional HPC cluster. For example, in one of our experimental runs with 1 second jobs, a single job took over 30 minutes to complete. The issue with this particular job was that the job was matched to a single node that was misconfigured. In this case, the job eventually finished after being matched to a different node. The OSG may not be an ideal solution for short executions of GridR functions.

3) Glidein: Glidein submissions use a pilot that is submitted directly to the SLURM scheduler. Once the pilot starts, it calls back to the submission node to request work. This method allows multiple jobs to run within the same SLURM job, independent of the cluster's scheduler.

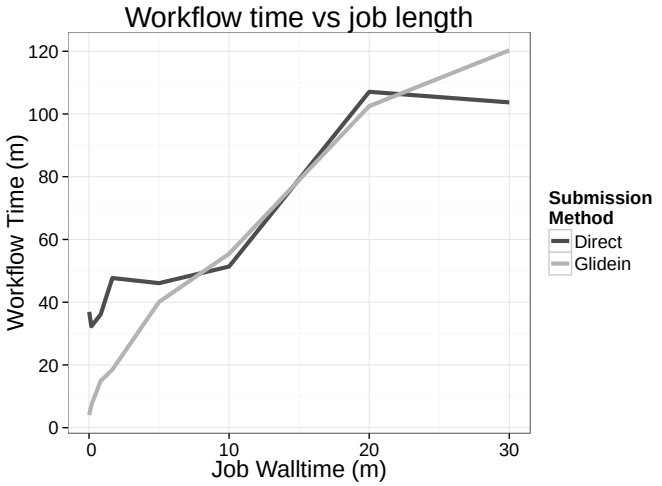

Fig. 5: Comparison of submission methods

Comparing the Direct submission to the glidein submission is shown in Figure 5 As you can see, for longer jobs, direct and glidein submission methods have approximately similar workflow runtimes. But, for short jobs, glidein has significantly shorter workflow runtimes. This can be attributed to the advantages of using a high throughput scheduler over a high performance scheduler.

Bosco is built on top of HTCondor. HTCondor is a very efficient high throughput scheduler that can quickly start the execution of jobs upon available resources. Since many R workflows are designed to run a short function upon a large amount of data, HTCondor is a good fit. By submitting HTCondor pilots to the remote scheduler, Bosco is able to utilize its strength of running many small jobs quickly, resulting in a shorter workflow completion time for shorter jobs.

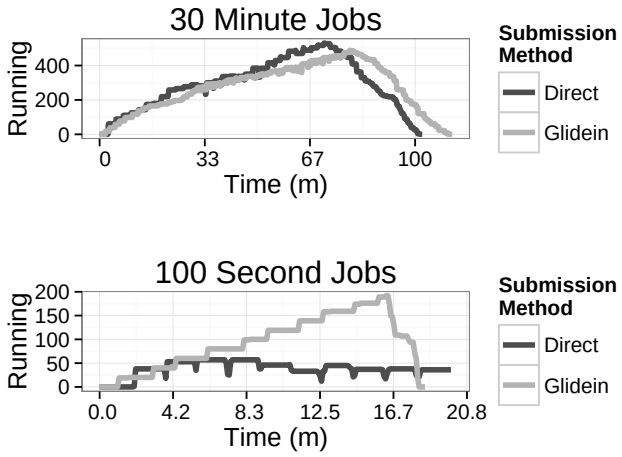

Fig. 6: Number of simultaneously running jobs

Figure 6 illustrates how the glidein submission method is superior to the direct submission method for short jobs, and why both methods are roughly equivalent for longer jobs. For longer jobs, you can see that both glidein and direct submission methods start jobs at nearly equal rates. The variation is relatively small, and could be explained by variations in the available resources at the time of running the experiments.

For the shorter 100 second jobs the start rate begins nearly the same, but then Bosco and SLURM are unable to sustain the job start rate. Since the jobs only run for 100 seconds the overhead from Bosco submission and SLURM starting the jobs becomes a bottleneck. Bosco is only able to sustain roughly 50 jobs running on the cluster. On the other hand, the Glidein submission method continues to grow in the number of jobs running. This is due to eliminating the Bosco submission overhead, as well as the SLURM scheduling overhead. Instead, the Glidein method is utilizing the much more efficient HTCondor scheduler, which is able to start jobs much faster than the SLURM scheduler. We can conclude that the glidein submission method results in a shorter total workflow execution time for shorter R functions.

V. Conclusion

BoscoR is a framework to execute R functions on distributed resources. It is a simple method for users to distribute

processing to remote resources. BoscoR incorporated user feedback in order to improve the framework.

As with any complicated system, many parameters can be varied in order to obtain different results. For example:

- Resource contention may be high which could cause the cluster not to start any GridR jobs.
- Resource contention may be low, which would cause SLURM to start all submitted jobs immediately.
- The number of glideins submitted in a batch could be varied in order to optimize the start rate for a particular cluster. Any lower and it would slow job starts, increasing the workflow run time for both short and long jobs. Any higher, and it could overwhelm the remote scheduler.

We chose reasonable values for these parameters that an end user may use. In the future we will tune these parameters automatically using the feedback provided by Bosco. Although Bosco and the bootstrap process significantly improved the fault tolerance of GridR, further fault tolerance testing and development is needed to provide a positive user experience when running on national infrastructure such as the OSG.

During follow up interviews with users after using BoscoR, we received many positive reviews of the framework. Improving the user experience of using R on distributed resources was a primary goal of BoscoR. One example of a positive review was from a Micro-Biology researcher from the University of Wisconsin:

> I have a huge set of data, which I have to split into pieces to be handled by each node. This is something I can do with the ”grid.apply” function. This reduces the submit time from several hours, to several seconds... it is a phenomenal improvement. This will greatly increase my use of grid computing, as right now, I only use grid computing when I have no other choice.

The experimental testing we ran showed that glidein submission method is significantly better at running short R functions than the direct submission method. At longer job runtimes, the difference between direct and glidein submission to remote resources is negligible.

ACKNOWLEDGMENT

This research was done using resources provided by the Open Science Grid, which is supported by the National Science Foundation and the U.S. Department of Energy's Office of Science.

This work was completed utilizing the Holland Computing Center of the University of Nebraska.

REFERENCES

[1] R. C. Team *et al.*, “R: A language and environment for statistical computing,” 2012.

[2] K. Rexer, “Data miner survey-2013 survey summary report,” *Rexer Analytics, Winchester*, 2013.

[3] KDnuggets. (2013) Top languages for analytics, data mining, data science. [Online]. Available: http://www.kdnuggets.com/2013/08/languages-for-analytics-data-mining-data-science.html

[4] J. S. Racine, “Rstudio: A platform-independent ide for r and sweave,” *Journal of Applied Econometrics*, vol. 27, no. 1, pp. 167–172, 2012.

[5] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein *et al.*, “The open science grid,” in *Journal of Physics: Conference Series*, vol. 78, no. 1. IOP Publishing, 2007, p. 012057.

[6] “Extreme science and engineering discovery environment (xsede).” [Online]. Available: http://www.xsede.org

[7] D. Weitzel, I. Sfiligoi, B. Bockelman, J. Frey, F. Wuerthwein, D. Fraser, and D. Swanson, “Accessing opportunistic resources with bosco,” in *Journal of Physics: Conference Series*, vol. 513, no. 3. IOP Publishing, 2014, p. 032105.

[8] D. Wegener, T. Sengstag, S. Sfakianakis, S. Ruping, and A. Assi, “Gridr: An r-based grid-enabled tool for data analysis in acgt clinico-genomics trials,” in *e-Science and Grid Computing, IEEE International Conference on*. IEEE, 2007, pp. 228–235.

[9] M. J. Litzkow, M. Livny, and M. W. Mutka, “Condor-a hunter of idle workstations,” in *Distributed Computing Systems, 1988., 8th International Conference on*. IEEE, 1988, pp. 104–111.

[10] R. L. Henderson, “Job scheduling under the portable batch system,” in *Job scheduling strategies for parallel processing*. Springer, 1995, pp. 279–294.

[11] I. Foster, “The globus toolkit for grid computing,” in *Cluster Computing and the Grid, IEEE International Symposium on*. IEEE Computer Society, 2001, pp. 2–2.

[12] M. Romberg, “The unicore grid infrastructure,” *Scientific Programming*, vol. 10, no. 2, pp. 149–157, 2002.

[13] T. Ylonen and C. Lonvick, “The secure shell (ssh) protocol architecture,” 2006.

[14] D. Weitzel, “Campus grids: A framework to facilitate resource sharing,” Ph.D. dissertation, University of Nebraska, 2011.

[15] “Package parallel.” [Online]. Available: https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf

[16] “Package snow.” [Online]. Available: http://cran.r-project.org/web/packages/snow/index.html

[17] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. MIT press, 1999, vol. 1.

[18] “Osg operations.” [Online]. Available: https://twiki.opensciencegrid.org/bin/view/Operations/WebHome

[19] G. Garzoglio, T. Levshina, M. Rynge, C. Sehgal, and M. Slyz, “Supporting shared resource usage for a diverse user community: the osg experience and lessons learned,” in *Journal of Physics: Conference Series*, vol. 396, no. 3. IOP Publishing, 2012, p. 032046.

[20] “Comprehensive r archive network.” [Online]. Available: http://cran.r-project.org/

[21] A. B. Yoo, M. A. Jette, and M. Grondona, “Slurm: Simple linux utility for resource management,” in *Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60.