ABSTRACT

BREIMYER, PAUL W. A Transparent Collaborative Framework for Efficient Data Analysis and Knowledge Annotation on the Web. (Under the direction of Professor Nagiza F. Samatova).

High-throughput experiments and ultrascale computing generate scientific data of growing size and complexity. These trends challenge traditional data analysis environments, most of which are based on scripting languages such as R, MATLAB or IDL, in a number of ways. To address some of these challenges, this research proposes a framework with the overarching goal to enable large-scale high-performance data analytics and collaborative knowledge annotation over the Web.

The proposed framework has three major components, which parallel the three core steps of the knowledge discovery cycle.

1. For the first step, defining the data analysis pipeline, the research designs and implements a Web-enabled interactive and collaborative statistical R-based environment. The component implements a memory management system that minimizes memory requirements thereby enabling multi-user scalability. To the best of our knowledge, this is the first Web-enabled R system that supports interactive remote access to R servers and enables users to share data, results and analysis sessions.

2. For the second step, executing the data analysis pipeline, the research investigates and proposes a transparent and low-overhead means for executing external compiled language parallel codes from within R, thus seamlessly bridging two code development paradigms: efficient, compiled parallel codes and high abstraction and easy-to-use scripting codes. This component contains three elements: a transparent bidirectional translation of data objects between R and compiled languages, such as C/C++/Fortran; seamless integration of external parallel codes; and automatic parallelization of data-parallel computations in hybrid multi-core and multi-node execution environments.

3. For the third step, annotating the predictive knowledge derived from community analysis pipelines, the research explores an environment for semantically rich, structured and queriable annotation of facts, relationships between those facts, and complex
events reported in scientific literature. The social networking nature of this compo-

nent allows the community to improve the predictions as well as generate new, higher-

level inferences, thus filling in the gaps in the communities’ understanding of physical

phenomena. The environment offers mechanisms for streamlining the annotated and
curated knowledge into distributed public databases, thus enabling a feedback loop
into the database-publication cycle to allow scientists to make connections between
data-driven predictions and published evidence.
A Transparent Collaborative Framework for Efficient Data Analysis and Knowledge Annotation on the Web

by
Paul Breimyer

A dissertation submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

Computer Science
Raleigh, North Carolina
2009

APPROVED BY:

Professor Tao Xie
Professor Steffen Heber
Professor Nagiza F. Samatova
Chair of Advisory Committee
Professor Mladen Vouk
BIOGRAPHY

Paul Breimyer received a BS in Mathematics and Computer Science, with honors, and a minor in Spanish, from Haverford College in 2001. He worked for EMC Corporation in their Hopkinton, MA headquarters from 8/2001-7/2005 as a Senior Software Engineer and managed a 4-member team on EMC Control Center, their flagship product offering. Paul enrolled in North Carolina State University’s Ph.D. program in 8/2005, was awarded the University’s Dean’s Fellowship in 2005, and won the 2006 Computer Science Outstanding Teaching Assistant Award. Paul successfully defended his Preliminary Oral Examination in 7/2008.
ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Nagiza Samatova, for her endless guidance and enthusiasm, as well as my committee for their insight and assistance: Professor Mladen Vouk, Professor Tao Xie, and Professor Steffen Heber.

I would also like to acknowledge and thank the SDM team at NCSU, led by Professor Mladen Vouk, as well as the following people for their continued support: Guruprasad Kora, Matt Schmidt, Nathan Green, William Hendrix, Vinay Kumar, Prachi Gauriar, and others.

Finally and especially, I want to thank my wife, Kelly Crowley, for all her love and support, both in general, and during my doctoral studies.

The work of P. Breimyer was funded by the Scientific Data Management Center (http://sdmcenter.lbl.gov) under the Department of Energy’s Scientific Discovery through Advanced Computing program (http://www.scidac.org) through a contract with Oak Ridge National Laboratory (ORNL).
## TABLE OF CONTENTS

### LIST OF TABLES

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>vii</td>
</tr>
</tbody>
</table>

### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>viii</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Iterative Steps of the Scientific Discovery Cycle ............................. 1  
1.2 Step 1: Defining Data Analysis Pipelines through WER ......................... 3  
1.3 Step 2: Executing Data Analysis Pipelines with pR ............................ 4  
1.4 Step 3: Annotating and Curating Knowledge via BioDEAL ....................... 6  
1.5 Publications
   1.5.1 Published ................................................. 9  
   1.5.2 Submitted .................................................. 11  
   1.5.3 Tutorials .................................................. 11

## 2 Executing Data Analysis Pipelines with pR

2.1 Motivation .......................................................... 12  
2.2 Major Contributions .................................................. 15  
2.3 Plugin Strategy
   2.3.1 Requirements ............................................... 16  
   2.3.2 R Details .................................................... 17  
   2.3.3 pR Middleware ............................................... 18  
   2.3.4 pR Performance ............................................. 23  
   2.3.5 Related Work ............................................... 26  
   2.3.6 Plugin Strategy Conclusion .................................. 27  
2.4 Parallelization Strategy
   2.4.1 Introduction ................................................. 27  
   2.4.2 Background .................................................. 28  
   2.4.3 Approach ..................................................... 29  
   2.4.4 Results ...................................................... 35  
   2.4.5 Related Work ............................................... 39  
   2.4.6 Parallelization Conclusion .................................. 40  
2.5 Automatic Parallelization Strategy
   2.5.1 Introduction ................................................. 40  
   2.5.2 Major Contributions ........................................ 42  
   2.5.3 Background .................................................. 42  
   2.5.4 Approach ..................................................... 43  
   2.5.5 Results ...................................................... 49  
   2.5.6 Related Work ............................................... 57  
   2.5.7 Automatic Parallelization Conclusion ............................ 60
LIST OF TABLES

Table 2.1 Data Type Mappings ................................................................. 19
Table 2.2 Method Performance Comparison in Seconds ....................... 24
Table 2.3 Comparing $R$ and $pR$ Calls to the Same External Methods .... 25
Table 2.4 $pR$ to $R$ Performance Comparison on One Processor .......... 36

Table 3.1 Web-Enabled $R$ Projects ....................................................... 65
Table 3.2 System Features ................................................................. 81
Table 3.3 Usability Features .............................................................. 82
Table 3.4 General Features ............................................................... 82

Table 4.1 Total Web and CoNLL annotations per day ....................... 101
Table 4.2 Recall and Precision for CoNLL annotations ...................... 102
LIST OF FIGURES

Figure 1.1 The SDAF Framework. This research focuses on the WER, $pR$ and BioDEAL components. PBA* [1] is not a primary component of this research, but some details are described in Chapter 2. ......................................................... 2

Figure 1.2 The WER front-end Web interface. ........................................ 4

Figure 1.3 $pR$ is an interface between $R$ and C/C++/Fortran serial and parallel codes. 6

Figure 1.4 Feedback loop between public data, BioDEAL and publication databases. 7

Figure 2.1 $pR$ is an interface between $R$ and C/C++/Fortran. ...................... 22

Figure 2.2 $Rmpi$ compute node model. An $R$ instance is required at every worker node. 31

Figure 2.3 $pR$ compute node model. Parallel third-party data analysis code is responsible for data distribution, inter-node communication and computation of the results. $pR$ provides means for in-memory data transfer between the $R$ and parallel computing environments, including invocation of parallel codes, as indicated by the solid brown circle. The third-party code can be written in MPI C/C++/Fortran...... 32

Figure 2.4 $pR$ enables “cross-talk” between $R$ and third-party parallel libraries by bridging in-memory data translation between the environments. In this way it allows third-party parallel analysis code to be “plugged-in” to $R$. ......................... 33

Figure 2.5 Factor improvement for $pR$ compared to serial $R$ matrix multiplication on 4096 x 4096 matrices using numbers of processors that are powers of two. .......... 37

Figure 2.6 The $pR$ overhead and total execution time for the matrix multiplication test case over powers of two numbers of processors (1-32) on matrix size 4096 x 4096. 38

Figure 2.7 Multi-processor speedup for $pR$, $C$, and $Rmpi$ compared to serial $R$ matrix multiplication implementation on 4096 x 4096 matrices. ................. 39

Figure 2.8 $pR$ software stack for automatic data-parallel statistical computation. .... 44

Figure 2.9 Multi-core speedup for $pR$, $snow$, and $multicore$ compared to a serial $R$ implementation. The results were gathered in the $E_2$ environment with 8 cores. ... 50

Figure 2.10 Multi-node speedup for $pR$ and $snow$ compared to the serial $R$ implementation. ................................................................. 51
Figure 2.11 Weak scaling comparison of snow and pR in the $E_1$ environment. 

Figure 2.12 pR's load-balanced execution in the $E_1$ hybrid environment using 16 nodes to sum 4096 vectors with length $2^{24}$. 

Figure 2.13 The pR overhead and total execution time compared to the multicore package over powers of two processes (1-64). 

Figure 2.14 Measured and Extrapolated Time to Perform MPI_Intercomm_Merge using OpenMPI. 

Figure 2.15 Parallel Paradigm Hierarchy. 

Figure 3.1 The RGui interface. The left window is the terminal and the right window is the graphical output from the R commands. 

Figure 3.2 WER Component Architecture. The dotted transition lines and the Graphics Generation and Disk I/O blocks represent transitions and blocks that are not always executed. The front-end Web Browser interacts with the Web Server, which manages all HTTP traffic. The R Interface manages interaction with R, including the R Initialization that starts the R Engine, which processes the R commands. Disk I/O is used whenever files are written to or read from the file system. Some of the existing projects rely heavily on Disk I/O, especially when graphics generation is required to transform image formats, sizes, or orientations. 

Figure 3.3 The WER browser interface. The bottom panel is a scratch pad area where users can develop R scripts. The upper left panel is the terminal window that mimics the RGui terminal window. The upper right panel is the R help pane. 

Figure 3.4 Results from the four single user testcases against each project. 

Figure 3.5 Results from the Null Multi-Commands testcase simulating concurrent users. 

Figure 3.6 Kepler WER interface. 

Figure 4.1 The database, BioDEAL and publication feedback loop. 

Figure 4.2 The BioDEAL Browser Frontend. 

Figure 4.3 A gene card for SO_2426 gene in Shewanella oneidensis MR-1. 

Figure 4.4 Search page results showing different genes and the publication IDs. 

Figure 4.5 The BioDEAL framework architecture.
Chapter 1

Introduction

1.1 Iterative Steps of the Scientific Discovery Cycle

The research cycle of science applications includes more than just designing an experiment (e.g., simulation model) and collecting data (e.g., simulation output). After, or while, the results are generated, scientists perform data analysis to discover, build, test, or annotate a new view of scientific reality (scientific discovery). These analytical predictions feed back into the design of new experiments.

Fundamental differences in how data analysis is performed exist as data increases in its size and complexity. This places some unique computation, memory, data, and knowledge management requirements on data analysis environments for massive scientific data sets. These requirements motivate research questions focused on the three steps of the iterative knowledge discovery cycle:

1. How will an end-user define a data analysis pipeline over massive data sets?
2. How will such analysis pipelines be efficiently executed?
3. How will analytical predictions derived from these analysis pipelines be annotated by the scientific community at large?

This research thesis proposes the Statistical Data Analysis Framework (SDAF) that aims to enrich and optimize the knowledge discovery cycle. In parallel with the three steps described above, SDAF contains the following three components (Figure 1.1 shows an overall SDAF architecture):
Figure 1.1: The SDAF Framework. This research focuses on the WER, pR and BioDEAL components. PBA* [1] is not a primary component of this research, but some details are described in Chapter 2.

1. WER, a Web-enabled statistical R-based environment, where R [2] is an open source mature statistical package (see Section 2.3.2), for defining data analysis pipelines in a remote, interactive and collaborative manner over the Web;

2. pR, a light-weight, easy-to-use plug-in that bridges compiled serial and parallel analysis routines into the R scripting environment for efficient execution of data analysis tasks;

3. BioDEAL, a semantically-rich annotation component for semantic annotation of published facts and complex events and their linkage as evidence for analytical predictions stored in distributed public databases.

For each of the three components, the problem, research challenges and proposed approaches to address them are briefly summarized below.
1.2 Step 1: Defining Data Analysis Pipelines through WER

**Problem:** Large-scale computing, high-throughput experiments and observatories are generating data at rapidly increasing rates and of growing complexity. These changes present a challenge to understand the science behind the data, which often necessitates that geographically distributed multidisciplinary research groups collaborate to solve data analysis problems. This creates a crucial need for Web-enabled collaborative data analysis environments capable of managing the increasing scale of data. Most traditional data analysis environments, such as R, are stand-alone applications that run on a researcher's desktop or laptop. Their usage by these emerging geographically distributed collaborative teams is quite limited. It is not advisable, nor sometimes feasible, to send petabyte, terabyte, or even gigabyte data among colleagues, and data analytics requires high-performance statistical computing (HPSC) resources, which further emphasizes the necessity for shared data and analysis results.

Web-enabled approaches are promising since they could provide a collaborative user environment that allows users to share data, analysis sessions and results. In addition, such approaches could leverage powerful hardware configurations with sufficient memory and parallel computation capabilities to handle larger computational demands. For these reasons, Web access is a fundamental improvement over the standard command line interfaces, or even local non-collaborative Graphical User Interfaces (GUIs). Arguably, client/server architectures could also address these issues, but they require a local installation, thereby limiting remote access. Support for different operating system platforms and versions also must be addressed, whereas Web browsers are almost universal.

**Research Questions:** Data analysis procedures can execute over days, weeks or months. How does the system handle discontinuous user sessions that allow users to login, submit a job, and return when the job is complete? What interface features are possible and necessary to promote collaborative data analysis? How does scalability with users and problem sizes affect system health and performance?

**Proposed Approach:** A general architecture is designed and a prototype, called Web-Enabled R (WER) [3], is developed for robust Web-enabled analytics of large-scale data in a collaborative environment, using R as its back-end analytical engine. A systematic evaluation and comparison with the current Web-enabled R projects demonstrated WER’s superior performance across a multitude of metrics derived from a comprehensive set of
the identified requirements. Figure 1.2 graphically shows WER’s environment that mimics and enriches R’s desktop environment over the Web. Chapter 3 describes the conducted research as well as performance benchmarks in more detail.

1.3 Step 2: Executing Data Analysis Pipelines with \( pR \)

Problem: The research for Step 1 primarily dealt with how to define data analysis over the Web, the next problem for data intensive statistical computing is how to execute these analysis scripts in the most efficient manner. The difficulty is that many of the current data analysis routines are written in non-parallel interpretable scripting languages such as \( R \), IDL, or MATLAB and are not scalable to massive data sets. The emerging approaches, such as those being developed by our lab, aim to provide parallel solutions. In the case of \( R \), several software packages including RScaLAPACK [4], \( pR \) [5], rpvm [6], Rmpi [7], etc. are maturing.

While promising, they assume that parallel implementations of data analysis routines either exist or are possible. However, there is a growing gap between the rate at which data analysis functions are developed and the rate at which they are parallelized. Moreover, the majority of domain scientists that develop domain-specific analysis routines are
typically not well versed in parallel computing and write serial data processing and analysis routines in compiled languages such as C/C++/Fortran. By doing so, they face a dilemma: on the one hand, they achieve improved performance by choosing the path of custom built compiled analysis codes. On the other hand, they lose access to the rich set of statistical analysis routines provided by scripting environments such as R or IDL.

Although scripting languages like R provide ‘hooks’ for calling external compiled functions in C/C++/Fortran, the burden on the user is quite high in terms of familiarity with R internals, which limits the effectiveness of these hooks. In addition, it is often the case that the overhead introduced by inter-language translation mechanisms is quite high and, in some cases, dominates the computational cost of executing analysis routines in a compiled language. As a result, these domain-specific analysis routines, while efficient and highly valuable for the community at large, rarely become an integral part of statistically rich and community-shared environments such as R.

**Research Questions:** There are numerous interesting open questions, such as how to enable an easy-to-use plugin for third-party domain specific analysis routines to interface with mature statistical environments, such as R? Specifically, how are the underlying inconsistencies of the data models resolved in order to interact with each other? Also, what is a proper API for the plugin and how are the internals of the statistical engine, such as R, hidden from non-expert third-party developers? How is the overhead induced as a result of the data model translation between the statistical engine and the third-party tools minimized? Do complex data structures or data models limit the effectiveness of the approach? Is it possible to semi-automate the parallelization and in what situations?

**Approach:** This research proposes pR [5, 8], a C/C++/Fortran open-source API that exposes third-party serial or parallel codes within the R [9] statistical package, provides automatic parallelization of certain data-parallel operations, and minimizes R overhead. Figure 1.3 graphically shows pR’s association with external languages and execution environments. Users can write analysis routines in any supported language and call them from R, therefore bridging the costly divide between current robust analysis tools. The ways to convert data types between R and a compiled languages (e.g., C/C++) are explored, and a templated approach that handles casting between systems is designed and implemented. To the best of our knowledge, pR is the first such system for R. pR often achieves superior performance speedups through parallelization and minimizing overhead compared to existing approaches. Chapter 2 describes the conducted research and performance benchmarks.
1.4 Step 3: Annotating and Curating Knowledge via BioDEAL

**Problem:** In many scientific domains, especially those of Biomedicine and Earth Science, the predictions derived from analytical pipelines are recorded in distributed public databases that are shared by their respective communities. It is often the case that these predictions (e.g., protein functions or anomalies in carbon pools) are imprecise or incomplete due to the noisy and uncertain nature of the data from which they are derived, as well as the incomplete physical models used to generate such data. As part of an iterative discovery cycle, these predictions may get tested, validated or improved (e.g., identified causes for anomalies or experimentally confirmed protein functions). The results of this downstream research process often get published but rarely are populated into the public databases as supporting evidence. There is a growing need for streamlining such downstream discovered evidence into the public databases to improve their analytical predictions. Currently, no capability of this sort exists.

**Research Questions:** The overarching question is how to close the growing gap
between these two paths of scientific discovery: the *data-to-predictions-to-databases* path and the *predictions-to-hypotheses-to-evidence-to-publications* path. Essentially, what is the proper infrastructure that would enable streamlining the published evidences, which rely on upstream database annotations, into the databases, thus establishing a feedback loop into the database-publication cycle? Researchers browse the databases for valuable predictions and spend tedious efforts to support their findings with possibly published evidences. However, without such an infrastructure, it is quite likely that this highly valuable knowledge extracted by researchers gets recorded only in researchers’ personal notebooks and does not become an integral part of the database-publication cycle to assist other researchers. Furthermore, how would scientists use information about the relationships between publication data and scientific results if it were available?

**Approach:** This research presents BioDEAL [10, 11, 12], a community *Biological Data-Evidence-Annotation Linkage* system that can introduce a feedback loop into the database-publication cycle to allow biologists to make connections between data-driven biological concepts and publications, and vice versa. The cycle is shown in Figure 1.4. By subscribing to the services provided by BioDEAL, an end-user, at a minimum, can annotate the facts reported in literature, associate these facts with ontological concepts, and share these literature annotations with other researchers in a social network. For example, while reading the paper by Lovley et. al, a genome annotation expert may decide to link the omcB (GSU2887) gene with electron carrier activity (GO:0009055) in the Gene Ontology (GO) and add a comment on experimental validation of its predicted function as the Fe(III)-
reductase. BioDEAL will record these annotations in a structured (XML) format so that other databases, such as GenBank, may parse this information and potentially update its “Related Articles in PubMed” field for this gene’s web page with this PubMed ID. Although a number of databases and frameworks can benefit from and/or enhance the functionality of BioDEAL, to the best of our knowledge, BioDEAL is the first system that enables such a feedback loop into the database-publication cycle. Although the proposed framework is tailored to the Biomedical community, its underlying architecture is quite generic in nature and is easily adaptable to other domains. Specifically, the demonstration of this architecture in the natural language processing domain has been performed. Chapter 4 describes the conducted research in more detail.
1.5 Publications

1.5.1 Published

1. Title: *pR: Lightweight, Easy-to-Use Middleware to Plugin Parallel Analytical Computing with R* [8]
   The International Conference on Information and Knowledge Engineering (IKE) 2009
   Authors: Paul Breimyer, Guruprasad Kora, William Hendrix, Nagiza F. Samatova

2. Title: *BioDEAL: Biological Data-Evidence-Annotation Linkage System* [11]
   Publication: Proceedings of the Conference on Bioinformatics and Biomedicine (BIBM) 2008
   Authors: Paul Breimyer*, Nathan Green*, Vinay Kumar, Nagiza F. Samatova

3. Title: *BioDEAL: Community Generation of Biological Annotations* [12]
   Publication: Journal of BMC Medical Informatics and Decision Making 2009
   Authors: Paul Breimyer*, Nathan Green*, Vinay Kumar, Nagiza F. Samatova (Invited)

4. Title: *WebBANC: Building Semantically-Rich Annotated Corpora from Web User Annotations of Minority Languages* [10]
   Publication: The 17th Nordic Conference on Computational Linguistics 2008
   Authors: Paul Breimyer*, Nathan Green*, Vinay Kumar, Nagiza F. Samatova

5. Title: *Web-Enabled R for Large-Scale Collaborative Data Mining: A Survey* [3]
   The International Conference on Information and Knowledge Engineering (IKE) 2009
   Authors: Paul Breimyer, Nagiza F. Samatova, Guruprasad Kora

6. Title: *PBA*: *Using Proactive Search to Make A* *Robust to Unplanned Deviations* [1]
   Publication: The 23rd Conference on Artificial Intelligence (AAAI) 2008
   Authors: Paul Breimyer, Peter Wurman

7. Title: *Parallel R for High Performance Analytics: Applications to Biology* [5]
   Publication: Scientific Data Management (Book In Press) 2009

* Both authors contributed equally
Authors: Nagiza F. Samatova, **Paul Breimyer**, Guruprasad Kora, Chongle Pan, Srikanth Yoginath
Editors: Arie Shoshani, Doron Rotem, Chandrika Kamath

8. Title: *An Outlook into Ultra-Scale Visualization of Large-Scale Biological Data* [13]
   Publication: Supercomputing Conference 2008
   Authors: Nagiza F. Samatova, **Paul Breimyer**, William Hendrix, Matthew C. Schmidt, Theresa-Marie Rhyne (Invited)

   Publication: Proceedings of the 22nd International FLAIRS Conference 2009 (Best Paper 2nd Runner-Up)
   Authors: Patricia Brent, Nathan Green, **Paul Breimyer**, Ramya Krishnamurthy, Nagiza F. Samatova

10. Title: *Coupling Graph Perturbation Theory with Scalable Parallel Algorithms for Large-Scale Enumeration of Maximal Cliques in Biological Graphs* [15]
    Publication: Journal of Physics 2008
    Authors: Nagiza F. Samatova, William Hendrix, Matthew Schmidt, **Paul Breimyer**

11. Title: *On Perturbation Theory and an Algorithm for Maximal Clique Enumeration in Uncertain and Noisy Graphs* [16]
    Publication: Conference of Knowledge Discovery and Data Mining; Workshop on Knowledge Discovery from Uncertain Data (U’2009)
    Authors: William Hendrix, Matthew Schmidt, **Paul Breimyer**, Nagiza F. Samatova

12. Title: *Incremental All Pairs Similarity Search for Variable Similarity Thresholds* [17]
    Publication: KDD 2009: The Third SNA-KDD Workshop on Social Network Mining and Analysis
    Authors: Amit Awekar, Nagiza F. Samatova, **Paul Breimyer**

13. Title: *Incremental All Pairs Similarity Search with Reduced I/O Overhead* [18]
    The International Conference on Information and Knowledge Engineering (IKE) 2009
    Authors: Amit Awekar, Nagiza F. Samatova, **Paul Breimyer**
1.5.2 Submitted

1. Title: *pR: Automatic Parallelization of Data-Parallel Statistical Computing Codes for R in Hybrid Multi-Node and Multi-Core Environments*
   Publication: The International Association for Development of Information Society (IADIS) 2009
   Authors: Paul Breimyer, Guruprasad Kora, William Hendrix, Nagiza F. Samatova

2. Title: *Theoretical Underpinnings for Maximal Clique*
   Publication: Theoretical Computer Science (Second Revision)
   Authors: William Hendrix, Matthew C. Schmidt, Paul Breimyer, Nagiza F. Samatova

1.5.3 Tutorials

1. Title: *Introduction to Scientific Workflow Management and the Kepler System* [19]
   Authors: Ilkay Altintas, Mladen Vouk, Scott Klasky, Norbert Podhorszki, Daniel Crawl
   Contributor: Paul Breimyer
Chapter 2

Executing Data Analysis Pipelines with $pR$

2.1 Motivation

The execution stage of the knowledge discovery cycle is arguably the most critical and represents a significant bottleneck in the cycle for large-scale datasets. Its performance also impacts the performance of the other stages. For these reasons the execution stage is presented first in this thesis, followed by the definition and analysis stages.

One of the core challenges for data intensive statistical computing is determining how to execute analysis scripts in the most effective manner. $R$ is a scripting language, similar to IDL and MATLAB, that provides convenient high-level abstractions to high-level data analysis routines. Its interpretable nature often becomes a rate limiting step for desirable performance, in terms of scaling to desirable problem sizes within practical end-to-end execution times and within minimal additional memory requirements and communication overhead.

Several strategies to improve the efficiency of $R$ can be pursued. Existing efforts include:

- **Plugin Strategy.** Due to the barriers imposed by learning new technologies, especially less efficient scripting languages, such as $R$, scientists develop custom codes in compiled languages and achieve high performance but lose the ability to share their analysis routines with other community members and lose the ability to leverage the
A rich set of statistical analysis routines provided by environments like R and IDL. The **Plugin Strategy** allows users to plugin these existing codes into the R environment, thereby increasing the number and quality of codes available to R users.

- **Parallelization Strategy.** In the most ideal scenario, scalable and portable parallel implementations of underlying data analysis functions should be provided. Our previous work on RScaLAPACK [4] and its current development, the pR project [15, 20], is a step forward in that direction. While mimicking the API of native serial R functions, pR provides the capability to run analysis routines on clusters of computers. Its performance depends on the efficiency of the underlying parallel algorithms. pR is middleware that allows users to leverage parallel data analysis routines written in MPI C/C++/Fortran while minimizing the overhead introduced by the middleware. However, pR’s applicability largely depends on the availability of parallel implementations of the underlying analysis routines. Furthermore, we extended pR to explore performing automatic parallelization of data-parallel computations and executing them in hybrid multi-core and multi-node environments.

- **Perturbation Analysis Strategy.** For certain types of data, such as those modeled by graphs (e.g. biological networks, social networks), improved efficiency can be gained by exploring our novel Graph Perturbation Theory (GPT) and algorithms [15, 16, 21]. GPT is primarily being developed to deal with uncertain and noisy data from which these graphs are derived. In this case, multiple solutions to graph analysis problems are sought for various “perturbed” graphs. Perturbations may be induced by filtering out some edges due to applied edge weight cutoffs or by adding vertices and edges based on additional information sources. For example, two genes in a gene expression network can be viewed as co-expressed (i.e., connected by an edge) if their Pearson correlation, derived from quantitative proteomics data, is above a certain threshold; various thresholds will correspond to different expression network perturbations. Likewise, two proteins can be considered functionally related if, in addition to genomic context information based on their neighborhood co-location on the genome, a gene fusion event becomes available as more genomes get sequenced. The intuition behind Graph Perturbation Theory is quite simple: if a solution to the “reference” or “unperturbed” graph is known, then it can be used to find the exact solution for the “perturbed” graph more efficiently than complete re-computation,
especially when the perturbation is relatively small. Specifically, GPT achieved more than 80% efficiency improvement compared to traditional graph algorithms on real biological networks, even when the number of perturbed edges ranged between 20% and 136%. As perturbations decreased (under 20%), a speedup of more than 50 times was observed.

- **Caching and Reusing Strategy.** Data analysis often repeatedly performs the same calculations, especially with common functions such as $\text{std}$ and $\text{auto-correlation}$. The occurrence rate for these overlaps is likely increasing due to the centralization of resources caused by greater computing demands forcing more users to large computing environments. The PBA* [1] algorithm is proposed to store a limited amount of analysis history per dataset to reduce the overall time requirements by hashing intermediary values and re-using them when appropriate, rather than re-calculating from scratch. PBA* leverages A*, a popular search algorithm, and finds optimal goal paths between cells. By considering the cells to be nodes in a scientific workflow, PBA* would allow the system to identify if a sequence of events for a particular dataset has already occurred, and if so, return the previously calculated values rather than repeating the process.

I have been involved in exploring all four of the aforementioned strategies. As part of my collaboration, the manuscript summarizing the **Plugin strategy** has been accepted to the WORLDCOMP International Conference on Information and Knowledge Engineering (IKE) 2009 [8]; the **Parallelization Strategy** has been accepted as a chapter in the book entitled *Scientific Data Management* [5]; the **Perturbation Analysis Strategy** has been accepted in the Journal of Physics [15] and the Conference of Knowledge Discovery and Data Mining (KDD) Workshop on Knowledge Discovery from Uncertain Data (U’09) [16]; and the **Caching and Reusing Strategy** was published in AAAI 2008 [1].

This chapter focuses on the plugin and parallelization strategies. The former aims to bridge efficient compiled languages with the high-level scripting $R$ language, while minimizing the overhead induced by inter-language translation, easing the necessity to learn the internals of $R$, and allowing code developers to express routines in compiled languages with which they are familiar. While $R$ provides hooks for calling external C/C++/Fortran functions, the burden on the user in terms of familiarity with $R$ internals is quite high, which limits the effectiveness of these hooks. $R$ provides an alternative interface, however,
it introduces overhead that is often higher than the time required to execute the underlying compiled function. To alleviate these problems, the proposed middleware research, called \( pR \), aims to lower the plugin overhead while providing ease-of-use.

The parallelization strategy is investigated from two angles: by using the aforementioned plugin approach to integrate efficient, compiled parallel codes into the \( R \) environment, and by performing automatic parallelization of data-parallel computations and executing them in hybrid multi-core and multi-node environments.

### 2.2 Major Contributions

Unlike the native \( R \) external interfaces, the proposed and implemented Statistical Data Analysis Framework (SDAF) component, called \( pR \), offers an efficient and transparent bridge to interface external serial and parallel codes with \( R \), while automatically parallelizing some data-parallel codes. The major contributions of the research study underlying \( pR \) are the following:

- Proposed and designed a tightly-coupled \( R \) interface that manages communication between \( R \) and third-party code while providing a mostly \( R \) agnostic, easy-to-use interface that enables non-\( R \) users to leverage the powerful \( R \) statistical engine and communicate with other non-\( R \) projects through \( R \).

- Proposed and designed a bidirectional translation of data objects between \( R \) and compiled languages, like C/C++, using a memory management system that enables direct memory access to \( R \) objects.

- Proposed and designed an automatic parallelization strategy for data-parallel codes that executes in hybrid multi-core and multi-node environments.

The proposed advancements have enabled \( pR \) to bring the flexibility of direct memory access, allowing users to explicitly request memory copies, provide minimal overhead in the case of direct memory access, support richer data types, and allow users to program in their accustomed language. The interface could ultimately enable a semi-automatic plugin with a wizard-like interface that generates a template for users to review and modify \( pR \) compatible routines.
This chapter has three sections: the first discusses the plugin strategy, followed by using the plugin strategy to integrate compiled parallel codes, and the final section discusses automatic parallelization.

2.3 Plugin Strategy

This section discusses the Plugin Strategy and its requirements; Section 2.3.1 discusses the requirements for the system; Section 2.3.2 gives an overview of relevant R internal details; Section 2.3.3 describes the architecture; Section 2.3.4 presents the performance results; Section 2.3.5 discusses related work; and Section 2.3.6 concludes this section.

2.3.1 Requirements

In order for an R infrastructure to successfully enable third-party analysis routines written in compiled languages to communicate and provide a straightforward interface for developers, the following requirements have been identified:

- The infrastructure should integrate intuitively into external code, thereby enabling developers to easily expose existing code within the system. Part of the motivation for such a system is to bring powerful existing routines together, making straightforward migration a paramount issue.

- Use an open interface to allow straightforward implementation of APIs by third-party users for libraries, such as BLAS and LAPACK.

- Shield external code developers from the bridging R language. Users should not be required to learn the R language or its internals to integrate with the system.

- The external code must be able to return values to R upon completion without requiring that users have experience with R internals.

- Natively support all R primitive object types in the external code.

- Provide reasonable performance, where reasonable is defined as at least as fast as a competent R implementation. In many cases, performance should dramatically improve when the external code is written in a compiled language.
Before discussing how pR addresses these requirements, an understanding of some 
R details is necessary, which is presented in Section 2.3.2.

2.3.2 R Details

About R

R is a command-line driven free implementation of the S statistical and graphing 
environment that was originally developed at Bell Laboratories in 1976. It is licensed under 
the GNU General Public License (GPL) [22]. R is extensible, produces publication quality 
plots, is widely used in academia, and is becoming more common in industry [23]. R can 
perform diverse statistical analysis tasks such as linear regression, classical statistical tests, 
time-series analysis, and clustering. R supports various platforms including Linux, Macin- 
tosh and Windows. It also provides a variety of graphical functions such as histograms, pie 
charts and 3D surface plots.

R provides extensibility by allowing users to load external code called packages, 
such as BioConductor [24], an open-source software package for the analysis of biological 
data. It also provides function interfaces to compiled code in the form of shared objects 
in UNIX, and DLLs in Windows, which are loaded and unloaded from the R environment 
using R function calls.

R Internals

SEXP is an internal R list data type that points to R objects; the objects can 
be of any R data type, including int, double, or another SEXP list. SEXP stands for S 
Expression where S [25] is the parent language to R. R supports sending parameters to 
external functions using the SEXP data type, which is the approach employed by pR and 
described in Section 2.3.3.

R also supports interaction with external C/C++ and Fortran methods without 
using the SEXP data type. In R’s approach, users cannot return objects from external 
methods. Instead, all changes are reflected by updating the variables passed as function 
arguments. Additionally, R introduces overhead both before and after calling the external 
methods caused by copying and casting parameter variables, as discussed in Section 2.3.4. 
A C/C++ foo method signature employing R’s approach is shown below.
```c
void foo(int *nin, double *x)
{
  ...
}
```

This `foo` method may be accessed in `R` using:

```
.C("foo", n=as.integer(10), x=as.double(rnorm(10)))
```

### 2.3.3 `pR` Middleware

The `pR` architecture is designed to meet the requirements outlined in Section 2.3.1 by providing an alternate, more efficient and transparent `R` interface strategy than the `R` approach described in Section 2.3.2. `pR` contains the following two levels: the User Interface level and the Backend level. The User Interface level contains an API that provides users with C/C++ template functions, for each `pR` data type, that enable developers to control what data types are being used. The template design also avoids having to extend the API to support additional data types. For example, the PRVector class contains an `insert` method that uses a template parameter to accept any data type, assuming it is valid for the vector instance. Similarly, the `[]` operator is overloaded to return the desired data type.

The Backend level contains both the Data Translation and `R` Interface blocks. The former handles data translation to/from `R`’s SEXP data type from/to C/C++ representations. Because `R` is written in C, the underlying objects are defined in C/C++ data types and `pR` can extract these representations from `R`’s SEXP data type using internal `R` functions. For example, if an integer vector is passed from `R` in an SEXP object, then the Data Translation block queries the `R` Interface to use `R`’s internal methods to extract the C representation of the vector and uses C’s `reinterpret_cast` method to convert the vector object pointer to the expected template data type: `<int>`. Similarly, if the vector is created in C/C++, the Data Translation block communicates with the `R` Interface to allocate memory and create the variable in `R`’s stack, and stores a native C/C++ pointer to the variable. This approach provides powerful extensibility, but the templates and `reinterpret_cast` method also allow users to perform invalid casts that may cause errors during execution; it is the developer’s responsibility to ensure that correct and valid template data types are used. Data translation is supported for all `pR` data types, which map to `R`’s data types, and are discussed in Section 2.3.3.
**pR Supported Data Types**

The pR classes map between R data types and C/C++ data types. Additionally, there is one utility class called PRParameters that can interpret R SEXP objects and extract the component variables. Without this class, users would not be able to access the passed parameters. The following example passes the method argument *args* to the PRParameters constructor in the PR namespace.

```c
PR::PRParameters pR Args(args);
```

The PRParameters data type supports indexed lookup using the following format: *pRArgs*(0,-1), where *pRArgs* is the variable name, 0 represents the index to retrieve, and -1 denotes the end of the list. The -1 construct is necessary because the PRParameters class overloads the () operator using the C/C++ “...” construct to allow a variable number of input parameters. For example, users can call *pRArgs*(0,1,2,3,-1) and the first four variables in the *args* SEXP object will be returned in an enclosing SEXP object. A flag is necessary to delineate the end of the list and -1 was chosen for pR because index values must be greater than zero and -1 can never conflict with a user’s index.

In compliance with R, the other six pR data types enable bidirectional translation of objects between R and external codes: **PRArray**, **PRMatrix**, **PRObject**, **PRObjectList**, **PRParameters**, and **PRVector**; Table 2.1 contains all pR data types with corresponding R data types. Non-primitive external data types can also be supported by defining pR fields within these more complex data types. For example, a foo class can contain PRVector and PRMatrix objects and it is the user’s responsibility to assign and use these data types appropriately.

<table>
<thead>
<tr>
<th>R</th>
<th>pR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>PRVector</td>
</tr>
<tr>
<td>Array</td>
<td>PRArray</td>
</tr>
<tr>
<td>Matrix</td>
<td>PRMatrix</td>
</tr>
<tr>
<td>Object</td>
<td>PRObject</td>
</tr>
<tr>
<td>List</td>
<td>PRObjectList</td>
</tr>
<tr>
<td>N/A</td>
<td>PRParameters</td>
</tr>
</tbody>
</table>

The PRVector data type can create vectors in R’s memory space, but can be
accessed in both C/C++ and R environments. For example, the following statement will create a vector of 10 elements for a variable named `vec` in the C/C++ environment:

```cpp
PR::PRVector<double> vec(10);
```

The `vec` variable can be accessed from within the C/C++ environment using the method `vec.getNativePointer()`, which uses templates to ensure that the returned object is of the correct data type. Likewise, the native R representation is available using `vec.getRObject()`, even though the variable was created within C/C++ code but in R’s memory space.

Each data type has a corresponding `.h` header file that includes three pR header files: `PRLanguageStub.h`, `PRSTLStub.h`, `PRNativeRStub.h`. `PRLanguageStub.h` links to standard C/C++ libraries (e.g. `iostream`, `cmath`, `string`); `PRSTLStub.h` includes all required Standard Template Library (STL) [26] headers (e.g. `vector`, `list`, `numeric`); and `PRNativeRStub.h` links to several R internal header files (`R.h`, `Rinternals.h`, `Rdefines.h`, and `Rmath.h`) to support interpreting R’s data types, especially the SEXP type that is passed to every exposed function.

**Using pR to Expose C/C++ Methods within R**

Exposing third-party code within R allows developers to leverage the powerful R statistical engine, send data between R and their external code, and execute efficient compiled code instead of implementing slower interpreted R methods. pR is written in C++ and supports external code written in C/C++ and Fortran; Figure 2.1 shows pR’s association with external languages. Exposing methods in C/C++ and Fortran are accomplished in similar ways, as described below. For C/C++, developers must do the following:

1. Include `pR.h`. This header file contains references to all the pR data types and provides a single `include` statement for developers. Alternatively, users can include only the specific pR data types required. These types are described in Section 2.3.3 and are also listed in `pR.h`.

2. Use the `PR` namespace in header files to ease lookup of pR data types and methods, and ensure that user developed codes are built within the same scope as pR. An example is shown below:

```cpp
using namespace PR;
```
3. All exposed methods must use the following signature (the method and parameter names are customizable, of course):

   \[ \text{PR SEXP foo (SEXP args);} \]

   \textit{PR} indicates that the method should be made available to \textit{R}, and is an alias for \textit{extern “C”}, a standard C/C++ construct to define variables and functions globally and is required by \textit{R}. The alias is provided as a convenience, although users may elect to use the standard \textit{extern} syntax instead. \textit{SEXP} is both the input parameter type and return type, \textit{foo} is the method name, and \textit{args} are the input parameters sent from \textit{R} in an enclosing \textit{SEXP} object.

4. Build a \textit{PRParameters} object as follows, where \textit{args} is the SEXP method argument, and the variable name (PRArgs) is customizable:

   \[ \text{PR::PRParameters PRArgs (args);} \]

   This class is necessary to access and translate the method arguments passed to/from \textit{R} from/to the external function.

5. Return an SEXP object from all exposed methods. The \textit{pR} data types, described in Section 2.3.3, support member functions that return SEXP objects, such as the \textit{getRObject} and \textit{getRList} methods.

6. Update the \textit{Makefile} to build the external code into a shared .\textit{so} library for Unix, or dynamic .\textit{dll} library for Windows.

   Alternatively, the \textit{pR API} also supports interacting with code written in Fortran by leveraging the C/C++ built-in support for Fortran interaction. To expose Fortran routines, developers must include the Fortran header file in a C/C++ project, precede methods with \textit{PRFortran}, rather than \textit{PR}, to indicate that they should be made available to \textit{R}, and use the built-in \textit{F77_CALL} method to invoke the Fortran function. This method takes the name of the desired external method as an input argument, followed by the external method’s input parameters, and may return an object. In the example below, \textit{foo} is the external function name, \textit{bar} is an input parameter to \textit{foo}, and \textit{retValue} is the value returned by the method.

   \[ \text{object retValue = F77_CALL(foo)(&bar);} \]
Figure 2.1: \( pR \) is an interface between \( R \) and C/C++/Fortran.

Once the methods are exposed, using the libraries requires two steps: first, load the library into the \( R \) environment, and second, call the desired method with the appropriate parameters. This is illustrated in the \( pR \) example in Section 2.3.3.

A \( pR \) C/C++ Example

To better illustrate how developers can build \( pR \) compliant C/C++ code, consider the simple example below:

```c
#include "testPR.h"

PR SEXP foo(SEXP args)
{
    PR::PParameters pRArgs (args);
    PR::PRVector<double> vec(pRArgs(0,-1));

    ... //execute desired code

    /* Create an object list that may contain various types
     * of data and is returned to the R environment. */
    PR::PRObjectList retList;
    retList.add("intVal", 0); //Adding dummy int
    return retList.getRList();
}
```

Below is the corresponding `testPR.h` example header file:
The \texttt{pR} method supports returning an object of type \textit{SEXP}, as shown in the above \texttt{pR} example, and also reflects changes to the arguments. The \texttt{pR} method requires users to use the \texttt{PRParameters} object to interact with the passed parameters, and returning an SEXP object entails using a \texttt{pR} data type to create the object; this is \texttt{retList} in the above \texttt{pR} example. To access the method, it must be compiled into a shared library, call it \textit{foo.so}, and \textit{R} users access the \texttt{pR} external method using the following:

```c
load("foo.so");
x = 1;
result = foo(x);
```

where \(x\) is an argument passed to \textit{foo}. If the method has more parameters, they should also be specified in this call with commas separating each argument. The output of the method is stored in \textit{result}, which can be accessed like any other \textit{R} object after the method completes, and may be passed to other internal or external methods for further analysis.

### 2.3.4 \texttt{pR} Performance

To evaluate the performance of \texttt{pR}, several common statistical methods were used to compare performance metrics of implementations in three languages: Python, \textit{R} and C++. Some of the performance discrepancies may be partly due to implementation details, although a concerted effort was made to implement the routines in each language as similarly as possible, and to use built-in functions when available. Straightforward methods without complicated algorithmic details were also selected for two reasons: first, this greatly limits any effects of implementation detail disparities, and second, these functions are often the building blocks of commonly used statistical analyses. Therefore, performance improvements in these functions suggest that even greater benefits may be realized with more complex algorithms, and these improvements may have wide-reaching impact due to their prevalence in statistical analyses. Table 2.2 compares the performance of Python, \textit{R} and \texttt{pR} using C++. The methods were implemented in each language using the same structure, operated on the same data, and the timing was limited to the actual computation time and did not include environment setup or teardown, such as loading the data.
Table 2.2: Method Performance Comparison in Seconds

<table>
<thead>
<tr>
<th>Method</th>
<th>Python</th>
<th>R</th>
<th>pR</th>
<th>pR Times Improv. over Python</th>
<th>pR Times Improv. over R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>0.300</td>
<td>0.108</td>
<td>0.030</td>
<td>10.0</td>
<td>3.6</td>
</tr>
<tr>
<td>Geometric Mean</td>
<td>0.323</td>
<td>N/A</td>
<td>0.128</td>
<td>2.9</td>
<td>N/A</td>
</tr>
<tr>
<td>Average</td>
<td>0.071</td>
<td>0.049</td>
<td>0.012</td>
<td>5.9</td>
<td>4.1</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.420</td>
<td>0.020</td>
<td>0.019</td>
<td>22.1</td>
<td>1.1</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>18.1</strong></td>
<td><strong>2.8</strong></td>
</tr>
</tbody>
</table>

The results in Table 2.2 demonstrate performance improvements in pR compared to implementations in both R and Python, up to 3.6 and 22.1 times, respectively. The average improvement compared to R was over 2.8 times and Python was over 18 times, although the improvements varied greatly. The Python results are somewhat expected because C and C++ are compiled languages, whereas Python is an interpreted language, and compiled languages are almost always notably faster than interpreted languages because the code has been pre-processed and optimized. Python has the ability to call native C/C++ methods, but the built-in mathematical libraries do not leverage this functionality. However, while R is also an interpreted language, many R internal functions, including those in Table 2.2, call compiled C methods to perform the processing, thereby limiting the impact of the interpreted language slowdown. This shows that current R users may achieve computational improvements by migrating to pR implementations.

To compare the overhead associated with R and pR, several available methods were identified in R that call backend C/C++/Fortran, and experiments were run on these methods by calling them directly from R and pR using the same data. The results are shown in Table 2.3. The difference between the experiments in Tables 2.2 and 2.3 is that the former calls the R wrapper methods, which add pre and post processing around the function call that affects the overall performance, whereas the latter circumvents this overhead by calling the underlying C methods directly.

For example, the LAPACK Singular Value Decomposition method is dsvdc, and the R wrapper function for this method is modLa_svd, which makes a copy of all variables before calling the external function, therefore increasing the response delay and memory requirements. The purpose of the Table 2.3 results is to remove any additional overhead by calling dsvdc directly from both R to pR and place both approaches on equal footing.
In all five testcases, \( pR \) is as fast or faster than \( R \). The performance improvements are attained by limiting the pre and post processing that occurs within \( R \). Timers were introduced into the \( R \) source code to capture the \( R \) overhead, shown in Table 2.3. In general, the pre and post processing overheads for internal functions are similar; for example, the total overhead for \( ddot \) was 0.128, of which 0.07 was pre-processing, and 0.058 was post-processing. In contrast, the \( pR \ ddot \) method incurred 0.001 preprocessing and no measurable post-processing. The processing occurs in the \( /src/main/dotcode.c \) file, and primarily in the \( do\_dotcode \) method for internal functions, and \( do\_External \) for external methods, such as the methods exposed through \( pR \). \( do\_dotcode \) searches for the appropriate method to call, verifies that the correct number of parameters have been passed, and then creates a C pointer for each parameter by copying the parameters to new objects, which can be an expensive operation. After the method completes, \( do\_dotcode \) converts each parameter back into an \( R \) object by making another copy.

The \( do\_External \) method also searches for the appropriate external method to call and verifies that the correct number of parameters are passed, but does not copy the objects into C objects. Instead, it directly passes the arguments as \( R \) SEXP objects. \( pR \) accepts the SEXP parameters and can act on them within the same memory space, therefore avoiding the potentially expensive copy and casting operations. The performance improvements from this approach are shown in Table 2.3.

<table>
<thead>
<tr>
<th>Method</th>
<th>C++</th>
<th>( R )</th>
<th>( pR )</th>
<th>( R ) Overhead</th>
<th>( pR ) Overhead</th>
<th>Times Improvement over ( R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ddot )</td>
<td>0.003</td>
<td>0.131</td>
<td>0.003</td>
<td>0.128</td>
<td>0.001</td>
<td>43.7</td>
</tr>
<tr>
<td>( dnrm2 )</td>
<td>0.024</td>
<td>0.044</td>
<td>0.026</td>
<td>0.014</td>
<td>0.000</td>
<td>1.7</td>
</tr>
<tr>
<td>( dchdc )</td>
<td>N/A</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.000</td>
<td>1.0</td>
</tr>
<tr>
<td>( dsvdc )</td>
<td>N/A</td>
<td>0.661</td>
<td>0.516</td>
<td>0.026</td>
<td>0.000</td>
<td>1.3</td>
</tr>
<tr>
<td>( dgeev )</td>
<td>0.002</td>
<td>0.557</td>
<td>0.004</td>
<td>0.027</td>
<td>0.002</td>
<td>139.3</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>37.2</strong></td>
</tr>
</tbody>
</table>

The ratio between \( ddot \) and \( dnrm2 \) in C++ is 0.125, yet in \( R \) the ratio becomes 2.98 mostly due to the nearly 98% pre and post processing overhead introduced by \( R \). The \( pR \) ratio, 0.115, is in line with the C++ ratio because \( pR \) does not introduce the same overhead. It should be noted that some of the analysis functions, like \( svd \), alter the data
during processing. For this reason, in some cases R makes explicit copies of variables during pre-processing and uses them to restore the original variable during post-processing. The current pR implementation does not support this behavior.

2.3.5 Related Work

Advanced computational science software is often thousands, or even millions, of lines of code and written by different groups of people over long periods of time, which can yield non-standard and non-transferable code. Componentization is one way to focus and sandbox software development to enable more robust code. The field of component-based software engineering (CBSE) seeks to improve the flexibility, maintainability and reliability of software systems [27]. However, one problem with componentization is that software entities are written in different programming languages.

Several component systems have been proposed to address these issues within the scientific domain, including Model [28]; CODE [29]; the AVS system [30] and SCIRun [31] frameworks for the visualization domain; Webflow [32]; Discover [33, 34]; and the Common Component Architecture (CCA) project [35]. Please see [36] for more details.

CCA is the most large-scale and recent of these projects; it is a modular system that leverages the Scientific Interface Definition Language (SIDL) to couple codes written in different languages by building ‘glue code’ between components. It currently supports C, C++, Fortran 77, Fortran 90, Python, and Java.

CCA is a powerful tool, but is geared towards a different audience than pR. CCA uses Babel [37] for language interoperability, which introduces a moderate learning-curve by requiring experienced programmers to wrap their components using Babel’s interface. Developers must write a SIDL file to describe the interfaces between components, and parsing this information introduces a runtime overhead. CCA/Babel also does not support statistical platforms, such as MATLAB and R.

In contrast, pR is designed to shield users from low-level interoperability issues by providing a simple interface that does not require programmers to generate intermediate files. While CCA requires users to implement the Babel-generated interfaces, pR exposes a simple API that users can call directly within their code. pR strives for simplicity, but does not provide the breadth enabled by the complexities of Babel. However, pR users could call Babelized code, thereby creating a bridge between R and CCA.
There are also several $R$ packages that aim to bring specific languages into the $R$ environment, such as RRuby [38], which is part of Omegahat [39], and rJava [40], for the Ruby and Java languages, respectively. While these packages are powerful, they face the same performance issues as other interpreted $R$ code, which $pR$ aims to avoid by providing integration within compiled codes.

### 2.3.6 Plugin Strategy Conclusion

Section 2.3.4 demonstrates the possible performance gains that $pR$ may yield over existing implementations, especially those written in interpreted languages. $pR$ is a bridge for existing and proven analysis techniques to communicate within a single environment, and even pass information between these routines by using $R$ to hand data back and forth. For example, users can call a BLAS method, get the result, and pass it to LAPACK for further processing. $pR$ also greatly increases the number and quality of tools available to $R$ users.

$pR$ provides an intuitive way for external tools to communicate through $R$ while often realizing strong performance gains.

### 2.4 Parallelization Strategy

The previous section discusses using $pR$ to plugin efficient serial codes written in compiled languages to improve performance and increase the breadth of functions available to users. This section builds upon those ideas to integrate parallel codes within the $R$ environment. The increasing size and complexity of modern scientific data sets challenge the capabilities of traditional data analysis approaches. In the context of High-Performance Statistical Computing, scalable parallel data analysis algorithms are a promising strategy to address these challenges. The $R$ statistical environment is a powerful and widely-used software package for data analysis, but effectively incorporating parallel implementations of data analysis functions into the interpreted $R$ scripting environment is a difficult and time-consuming process. Likewise, available parallel computing libraries lack the capabilities of the statistically-rich $R$ engine.

This section describes using $pR$ as a bridge between these two environments, allowing the efficiency and scale of existing parallel data analysis functions to be “plugged-in”
to the flexible scripting statistical environment of R. pR offers a number of performance advantages over existing approaches to parallel analytical computing with R.

2.4.1 Introduction

The growing demand for analyzing data of increasing size and complexity has led to the emergence of the High-Performance Statistical Computing (HPSC) field, resulting in the development of several open source and commercial software tools and libraries. To meet the increasing demands of HPSC users, many mathematical systems are moving toward parallel computing strategies [41, 42]. Because of its inherent extensibility, the R statistical platform [9] has been a target of such efforts, though it has limited native support for parallel computing. Instead, previous researchers have produced add-on packages, such as Rmpi [7] and rpvm [43], to provide low-level support for writing parallel programs. These approaches to parallel programming place the burden of implementing parallel codes on the end-user, and can also result in slower end-to-end execution due to the scripting and interpreted nature of the R environment.

This section proposes a lightweight, easy-to-use pR middleware interface that bridges the R statistical environment with parallel computing libraries. pR abstracts away the parallel computing implementation details from packages like ScaLAPACK, allowing developers to leverage existing parallel codes written in compiled languages without modifying the R package.

The rest of this section is organized as follows: Section 2.4.2 introduces existing parallel computing strategies integrated with R. Section 2.4.3 presents pR’s approach in the context of the design strategies undertaken by Rmpi and RScaLAPACK. Experimental performance results are given in Section 2.4.4. Section 2.4.5 describes related work, and Section 2.4.6 concludes this section.

2.4.2 Background

R provides extensibility by allowing users to load external codes called packages, such as Rmpi [7] and RScaLAPACK [4]. Currently, there are nearly 1600 external packages available to R users [44]. The Rmpi package is an R wrapper around C++ implementations of the MPI specification (LAM/MPI [45], MPICH2 [46], OpenMPI [47]). It supports basic MPI routines such as MPI_Send, MPI_Gather, etc., and is maintained by Professor Yu
at the University of Western Ontario (http://www.stats.uwo.ca/faculty/yu/Rmpi).

The RScALAPACK package exposes parallel ScaLAPACK routines within R. ScaLAPACK is a library of high performance parallel linear algebra routines [48, 49], based on the Parallel BLAS [50] and Basic Linear Algebra Communication Subprograms (BLACS) [51] libraries. ScaLAPACK can solve systems of linear equations, eigenvalue problems, and singular value problems, and is optimized for MIMD distributed memory concurrent supercomputers. RScALAPACK is developed and maintained by Professor Samatova and her lab at Oak Ridge National Laboratory (http://cran.r-project.org/web/packages/RScALAPACK/index.html).

2.4.3 Approach

The proposed pR project extends the RScALAPACK package to allow more general parallel computation capabilities within the R environment. While existing R packages such as rpem [43] and Rmpi mimic communication level protocols at the level of the R scripting language (PVM and MPI, respectively), such packages may not be ideal in every case. For example, users may not have the resources or experience to implement the parallel computation details, and execution will likely take longer in the interpreted R environment than in a compiled language like C or Fortran, particularly if large amounts of data need to be distributed.

Likewise, if a computation is in a pipeline, developers may opt for a hybrid approach by calling high-level interfaces for some computations to leverage existing parallel computing libraries while managing low-level communication details for other components of the pipeline. For example, users may need to send and receive data between the server on which R is running and remote machines with specialized software necessary for different parts of the pipeline. Once the remote machines have received the data, they can, in turn, use low-level or high-level communication as necessary to farm out a parallel job.

For these reasons, the architecture of parallel programming systems can differ based on their target users. The rest of this section describes some of the design decisions and differences between pR, RScALAPACK and Rmpi.
Parallel Programming Paradigm

The responsibility of managing parallel computing intricacies, including data distribution and/or inter-node communication, can be placed at three distinct levels: with the end-user (e.g.: Rmpi); as middleware (e.g.: RScaLAPACK); or in third-party routines (e.g.: pR). The end-user level awards the user the most control over the implementation details but also places the most burden on the user, as they must have the resources and knowledge to implement parallel programs. Within interpreted environments like R, this burden is compounded by the performance penalty relative to compiled systems associated with executing low-level code. Rmpi is an example of an interpreted, end-user level parallel programming paradigm.

The main goal of Rmpi, as stated by the author [7], is to provide MPI functionality within the R environment so users are not required to know C or Fortran. While the MPI routines are executed as compiled C++ code, the non-MPI related code, such as data allocation and manipulation, is executed within the R environment as interpreted code, which may affect overall performance. Rmpi allows instances of R running on different machines to communicate with each other in order to solve a problem in parallel; Figure 2.2 shows a sample communication configuration.

The middleware paradigm differs from the end-user approach because it strives to hide parallel computing details from the user. Instead, it manages the data distribution and communication details internally, relieving the end-user from this responsibility. Because these parallel computing details are usually problem and function specific, the middleware generally requires a custom implementation for every function, limiting users to supported functions. RScaLAPACK is an example of a middleware package that provides R users with a transparent mechanism to perform parallel computations outside the R parent process without burdening R users with the complexities of parallel computing; however, users are limited to the functions that RScaLAPACK explicitly supports.

pR is designed to complement both RScaLAPACK and Rmpi and is an example of a third-party parallel programming paradigm. Similar to RScaLAPACK, the pR interface to parallel analysis functions mimics the serial functions’ interface, which is a single function call (see Section 3.4 for details). In contrast to RScaLAPACK, which is designed specifically for ScaLAPACK, the pR architecture has been abstracted to allow parallel third-party analysis functions to integrate with the R environment without requiring major
Figure 2.2: *Rmpi* compute node model. An *R* instance is required at every worker node.

modifications to either *pR* or to the external third-party libraries. Like *Rmpi*, *pR* supports data-parallel computation via data distribution among compute nodes and inter-node communication. However, unlike *Rmpi*, with *pR* these tasks are invoked in the third-party parallel data analysis codes rather than in the end-user *R* script (see Figures 2.2 and 2.3 for comparison). Specifically, through the *pR* interface, *R* end-users can perform an *in-memory transfer* of the data generated in *R* to the parallel computing environment that serves the third-party MPI C/C++/Fortran data analysis functions. The results of the parallel analyses are also returned to the *R* environment through *pR* using an *in-memory transfer*, with data distribution and inter-node communication managed by the third-party compiled codes, as shown in Figure 2.3. Note that *pR* also supports *network transfer* of data using sockets and/or MPI between *R* and parallel computing engines; the details are outside the scope of this work.

Rather than treating *R* as an independent process, *pR* interacts with *R*’s internal
Figure 2.3: \textit{pR} compute node model. Parallel third-party data analysis code is responsible for data distribution, inter-node communication and computation of the results. \textit{pR} provides means for in-memory data transfer between the \textit{R} and parallel computing environments, including invocation of parallel codes, as indicated by the solid brown circle. The third-party code can be written in MPI C/C++/Fortran.

engine by linking directly into \textit{R}'s source code. This technique is enabled by \textit{R}'s open-source code base and sharing a common programming language (C++) with \textit{pR}. In this way, \textit{pR} can directly access \textit{R}'s datatypes, libraries, computation engine, etc., thereby providing a low-level efficient connection between the two projects. The alternative would be to treat \textit{R} as an independent process, but such an implementation would likely result in slower performance and much more limited integration between \textit{R} and \textit{pR}.

The \textit{pR} software stack is shown in Figure 2.4 and illustrates the flow of data and control. When an end-user calls a \textit{pR} function, the compiled component of \textit{pR} receives any parameters that have been passed from the \textit{R} environment, and manages the invocation of the appropriate third-party parallel code and the in-memory data transfer to the parallel computing environment. Once the parallel computations are complete, \textit{pR} receives the result via an in-memory transfer from the attached compute node and passes the result back to the \textit{R} environment in-memory, where the \textit{R} end-user can interact with the data in a native \textit{R} environment. Effectively, the \textit{R} environment augmented with \textit{pR} is resident with one of the compute nodes (e.g. master process) in the parallel computing environment.
Figure 2.4: \( pR \) enables “cross-talk” between \( R \) and third-party parallel libraries by bridging in-memory data translation between the environments. In this way it allows third-party parallel analysis code to be “plugged-in” to \( R \).

**Performance Implications**

The design decisions and architecture for each project have performance implications, discussed below. While developers using \( Rmpi \) may not need to be familiar with \( C \) or \( Fortran \), users must face the challenge of managing low-level MPI details, and there may be scalability issues for large problems due to the overhead of managing data within the interpreted \( R \) environment. End-users also interact with \( RScaLAPACK \) from within the \( R \) environment, but all the parallel computing logistics are executed within compiled code. Therefore, \( RScaLAPACK \) does not contend with the same scalability concerns as \( Rmpi \). Similarly, \( pR \) executes in compiled codes and its performance is expected to closely mirror the performance of the third-party codes that it calls while introducing little overhead; experimental performance results are discussed in Section 2.4.4.

**Extensibility**

We define extensibility as the ability to introduce new parallel data analysis functionality within a software package, and \( Rmpi \), \( RScaLAPACK \) and \( pR \) each take a different approach to this feature. \( Rmpi \) allows end-users to implement new parallel codes in the \( R \) scripting language, and is therefore fully extensible, though the users must spend the
resources to engineer the solution themselves. In addition, the re-use of existing parallel data mining and analysis codes written in a compiled language becomes difficult unless the parallelization logic is re-implemented in the R scripting language. A second type of extensibility is also relevant for Rmpi: because it is a wrapper around MPI, future MPI functionality changes may necessitate expanding the Rmpi package. While Rmpi currently implements about 40 of the most common MPI methods, over 200 functions are identified in the MPI-2 specification.

While RScaLAPACK yields improved performance [4] compared to R’s integration of serial LAPACK functions, it was engineered specifically for the ScaLAPACK library, even though the design could theoretically support any computational system. Using other parallel computational libraries (beyond ScaLAPACK) would require significantly modifying the RScaLAPACK’s Parallel Engine (see [4] for the definitions and implementation details). Therefore, the extensibility of RScaLAPACK is more limited than Rmpi, though its computational performance is superior.

pR’s approach differs from both Rmpi’s and RScaLAPACK’s; because parallel routines are often complex, the pR design enables users to leverage existing parallel codes rather than spending time re-implementing them partially or entirely within the R environment, as in Rmpi, or within the Parallel Engine, as in RScaLAPACK. pR can serve as a bridge between R and third-party routines to enable cross-talk between R and existing, robust codes, allowing parallel computing environments to leverage the interactive, scripting, and statistically-rich powers of the R engine.

Usage

Within the R environment, end-users implement Rmpi-enabled codes differently than RScaLAPACK and pR codes, which both have similar interfaces. Whereas users of Rmpi would engineer their own parallel codes and call MPI-level communication commands, the other two packages allow end-users to make a single function call, as in the pR R code shown below:

```r
A = matrix(c(1:256),16,16);
B = matrix(c(1:256),16,16);
result = pr.mm(A, B);
```

A and B are 16×16 matrices that are passed as arguments to pr.mm(), where pr.mm() is a matrix multiplication method discussed in Section 2.4.4; if the method has more parameters,
they can be passed with commas separating each argument. The output of the method is stored in `result` and can be used just like any other R object. `pR` could simply support a parallel method named `mm()`, but we decided that the `pr.mm()` syntax would be more intuitive and explicit for end-users—this convention is analogous to the design decision that we made in `RScaLAPACK`.

### 2.4.4 Results

To evaluate the performance of `pR`, we compare it against `Rmpi`, `R` and `C` in Sections 2.4.4, 2.4.4, and 2.4.4, respectively, using several matrix multiplication test cases with different-sized matrices and varying numbers of processors. Matrix multiplication was chosen for two reasons: there is a published `Rmpi` example [52], which allows us to compare against “unbiased” `Rmpi` code, and matrix multiplication is both simple and commonly used as a component in various data analytics functions. Therefore, `pR` performance evaluations determined for this test case are likely applicable to other common functions and more complex scenarios.

The test environment is a dedicated 32 node Intel-based Infiniband cluster running Linux. Each node contains two 3.4 GHz Pentium IV processors and 6GB of memory. Only one processor per node was used as a compute node in our experiments; when we refer to the number of processors in the results below, we are specifying the number of compute nodes.

**pR vs. Rmpi Execution Time**

We compared `pR` to `Rmpi` by executing test cases on different datasets with varying numbers of processors. The `Rmpi` matrix multiplication code was written by Theussl [52], and each worker node calls R’s matrix multiplication operator. `R` then calls the `dgemm` Basic Linear Algebra Subprograms (BLAS) [53] Fortran function to compute the intermediate values. We implemented a test case in `C` that uses the same data distribution model and calls the same `dgemm` BLAS method, and exposed the method to `R` using `pR`. We randomly generated square matrices for all power-of-two sizes from $32 \times 32$ to $4096 \times 4096$ and ran each test case with `pR` and `Rmpi` for 1, 2, 4, 8 and 16 processors.

In all cases, `pR` yielded performance improvements over `Rmpi`; the average factor of improvement was 37x. Because both `pR` and `Rmpi` leverage the same underlying
BLAS function to perform the actual calculation, most of the performance difference can be attributed to data distribution: while Rmpi distributes data through the interpreted R interface, pR first passes the data to our C matrix multiplication code, which then distributes the data.

**pR vs. R Execution Time**

To compare the performance between pR and R, we executed the same matrix multiplication test cases on one processor and calculated the factor improvement achieved by pR by dividing the total execution time for serial R by the execution time for pR using one processor (i.e., without parallel programming). Table 2.4 contains the performance comparison.

Table 2.4: pR to R Performance Comparison on One Processor

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>R (sec)</th>
<th>pR (sec)</th>
<th>pR Factor Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>&lt; 1 \cdot 10^{-3}</td>
<td>1 \cdot 10^{-3}</td>
<td>0.0</td>
</tr>
<tr>
<td>64</td>
<td>1 \cdot 10^{-3}</td>
<td>8 \cdot 10^{-3}</td>
<td>0.01</td>
</tr>
<tr>
<td>128</td>
<td>1 \cdot 10^{-3}</td>
<td>8 \cdot 10^{-2}</td>
<td>0.07</td>
</tr>
<tr>
<td>256</td>
<td>5 \cdot 10^{-2}</td>
<td>1 \cdot 10^{-1}</td>
<td>0.41</td>
</tr>
<tr>
<td>512</td>
<td>8 \cdot 10^{-1}</td>
<td>2 \cdot 10^{-1}</td>
<td>5.63</td>
</tr>
<tr>
<td>1024</td>
<td>1 \cdot 10^1</td>
<td>6 \cdot 10^{-1}</td>
<td>17.18</td>
</tr>
<tr>
<td>2048</td>
<td>9 \cdot 10^1</td>
<td>4 \cdot 10^0</td>
<td>19.78</td>
</tr>
<tr>
<td>4096</td>
<td>7 \cdot 10^2</td>
<td>2 \cdot 10^1</td>
<td>28.32</td>
</tr>
</tbody>
</table>

For small matrices, R outperforms pR due to the total execution time being less than the small overhead incurred by pR. However, this margin monotonically decreases until matrix size 512 \times 512, where pR performs 5 times faster than R. As the matrices continue to increase in size, pR yields steadily larger end-to-end performance improvement factors. For a matrix multiplication between two 4096 \times 4096 matrices, pR performs over 28 times faster than R. The performance improvement of pR is expected because it performs all computation and data manipulation within C, a compiled language, rather than the interpreted R environment.

While the data in Table 2.4 fixes the number of processors at one and varies the size of the matrices, Figure 2.5 shows the log_2 of pR’s total execution time when the matrix size is fixed to 4096 and different numbers of processors are used. pR’s results are
Figure 2.5: Factor improvement for \( pR \) compared to serial \( R \) matrix multiplication on \( 4096 \times 4096 \) matrices using numbers of processors that are powers of two.

compared to those generated by a serial \( R \) matrix multiplication implementation that uses the standard \( R \) command \texttt{\%*\%}. The \( R \) engine computes the matrix product by calling the same BLAS \texttt{dgemm} function that \( pR \) uses, so the \( pR \) and \( R \) implementations and results are comparable. The ideal line was calculated using the actual \( R \) serial execution time for one processor and dividing this time by the number of processors. As shown in the figure, the \( pR \) factor improvements perform superlinearly and support the conclusion that \( pR \) can scale to both large numbers of processors and large datasets.

**Overhead Induced by \( pR \)**

We compare \( pR \) to the third-party codes that it calls by calculating the overhead induced by \( pR \), \( O(p) \), defined as follows:

\[
O(p) = pR(p) - T(p),
\]

where \( pR(p) \) is the total end-to-end execution time using \( pR \) on \( p \) processors and \( T(p) \) is the analogous time for the third-party code, which could be written in C/C++ or Fortran. Figure 2.6 depicts the \( pR \) overhead alongside the third-party execution time on matrices of size \( 4096 \times 4096 \) on powers of two processors from 1 to 32. As expected, the \( pR \) overhead
Figure 2.6: The pR overhead and total execution time for the matrix multiplication test case over powers of two numbers of processors (1-32) on matrix size 4096 × 4096.

remains constant due to its in-memory data transfer (i.e., by passing the pointers) and the total execution time decreases as more processors are available to share the work. The results suggest that at some number of processors greater than 32 the third-party execution time may become smaller than the fixed pR overhead.

Speedup

To compare pR, Rmpi, and C to each other, we calculated the speedup of each and compared them to the ideal. To normalize the comparison, we measured speedup as the relative performance between single-processor R and the corresponding multi-processor system (i.e., pR, Rmpi, and C), defined as:

\[
S(p) = \frac{T_{\text{serial}}}{T_{\text{parallel}}(p)},
\]

where \( T_{\text{serial}} \) is the R execution time on one processor and \( T_{\text{parallel}}(p) \) is the execution time of pR, C, or Rmpi using p processors. As in Section 2.4.4, the ideal speedup was determined by dividing the execution time of the serial R matrix multiplication on one processor by the number of processors.
The speedups achieved by \( pR \), \( C \) and \( Rmpi \) on square matrices of size 4096 are shown in Figure 2.7. The \( \log_2 \) of end-to-end times were used for clarity. As the graph demonstrates, \( pR \) and \( C \) achieve superlinear speedup with respect to an ideal serial implementation of matrix multiplication in \( R \), and \( Rmpi \) yields sublinear speedup. Because \( pR \) is a bridge to existing third-party routines, \( pR \) is inherently bound to the performance of these methods; if they perform well or poorly, so will \( pR \), as evidenced by the close proximity between end-to-end times of \( pR \) and \( C \) in the figure.

### 2.4.5 Related Work

In addition to \( R \), other closed-source scripting software supporting statistical computing, such as MATLAB and IDL, have pursued augmenting their environments with parallel computing capabilities, such as the *Parallel MATLAB* [42] and *Massively Parallel IDL* (MPIDL) [54] parallel computing plug-ins. A thorough investigation of parallel MATLAB approaches is available in [42]. The technology underlying \( pR \) is generic and could potentially be further generalized to support *octave* and *GDL*, which are open-source versions of MATLAB and IDL, respectively.
2.4.6 Parallelization Conclusion

$pR$ extends our previous $RScaLAPACK$ work and provides a bridge between existing third-party parallel analysis techniques and the $R$ statistical environment. $pR$ enables external codes to pass data to and from $R$ via in-memory transfers and therefore increases the number and quality of tools available to $R$ users. Section 2.4.4 demonstrates that $pR$’s performance closely mirrors the performance of the third-party codes that it calls, while introducing limited overhead. Specifically, $pR$ achieved an average performance improvement of 37 times compared to $Rmpi$, and yielded superlinear speedups compared to serial $R$ execution times while incurring overhead under a few micro-seconds in our experiments with $4096 \times 4096$ matrices across powers of two processors from 1 to 16.

2.5 Automatic Parallelization Strategy

2.5.1 Introduction

The emergence of the High-Performance Statistical Parallel Computing (HPSPC) field has resulted in the development of several open source and commercial software tools and libraries, such as those that build upon parallel mathematical libraries like $ScaLAPACK$ [55]. However, parallel statistical computing is not a trivial task. An ideal HPSPC system would automatically execute scientists’ serial data analysis codes in parallel. However, this holy grail of parallel statistical computing remains elusive. A simpler, yet powerful, approach involves automatic execution of the same task on different subsets of the data in parallel, or data-parallel statistical computing, thereby avoiding issues related to inter-process dependencies.

Arguably, many data intensive statistical methods rely on such data-parallel operations that repeatedly perform identical calculations on varying inputs; some examples include bootstrapping and cross-validation. These methods can be logical places to partition operations for parallel execution because each calculation for one input is inherently independent from the same calculation on another input.

$R$, a powerful and widely-used open-source statistical computing engine, is a natural target for automatic parallelization of data-parallel statistical computing methods. In particular, $R$ supports a commonly used family of apply methods that independently and serially execute a specified function against each variable in a list.
\( R \) has limited native support for parallel statistical computing. Instead, previous research has produced add-on packages that support high-level parallel implementations of apply-like functions that shield users from the low-level details of explicit parallelization. For example, a parallel apply method, called \( \text{lapply} \), that accepts a list and a function definition as its arguments can pass the first variable in the list with the specified function to one process, the second element and the same function to the next process, and so on, and gather the derived results. Current projects that implement the parallel \( \text{lapply} \) function include \textit{snow} [56], \textit{multicore} [57], and \textit{Rhipe} [58] for multi-node, multi-core, and Hadoop Map/Reduce [59] environments, respectively. However, to the best of our knowledge, none of them is designed for hybrid multi-node and multi-core environments.

Parallel implementation of the \( R \) \( \text{lapply} \) method for both multi-node and multi-core environments raises interesting challenges. For example, how do different processes share their computational environments? Computational environments may include loaded libraries and previously created variables that may be required for the execution of the statistical function passed as an argument to the \( \text{lapply} \) method. Solutions to environment sharing may differ in multi-node environments, where data transmission is expensive, and multi-core environments, where processes have access to the same shared memory.

Another challenge is how to protect users from the underlying complexities of parallel statistical computing. For example, the user can decide to broadcast all data to each compute node, which may be acceptable for certain environments and problem sizes, or instead, the user can determine a data distribution algorithm to send subsets of data to specific nodes. In both cases, the user must ensure that the execution environment on each compute node contains the required libraries and variables. These data distribution and environment integrity concerns can be a burden for end-users and are general challenges for all parallel implementations of the apply family of \( R \) methods.

To address these kinds of challenges, this section extends the \textit{pR} middleware library to support automatic parallelization of data-parallel statistical computations in \( R \) using a hybrid approach for both multi-node and multi-core environments. For example, if a cluster has 1024 nodes, and each node has 8 cores, then with few or no changes to the serial \( R \) code, \textit{pR} will distribute the data equally among the 1024 nodes and implicitly migrate the \( R \) environment on the user’s behalf. The work per node is further divided among the 8 cores and each process uses its own copy of the environment, thereby striving for a ‘best of both worlds’ approach. Using \( R \)’s \( \text{lapply} \) method, we demonstrate \textit{pR}’s benefits, such as
improved overall performance and transparent parallelization. pR is available upon request.

The rest of the section is organized as follows: Section 2.5.2 identifies the major contributions of this work. Section 2.5.3 introduces R and existing parallel computing R packages related to parallelization of methods from the apply family in R. Section 2.5.4 presents pR’s approach and experimental performance results are given in Section 2.5.5. Section 2.5.6 describes related work and Section 2.5.7 concludes the section.

2.5.2 Major Contributions

The major contributions of the pR middleware component presented in this section are the following:

- pR provides automatic parallelization of data-parallel codes in hybrid multi-core and multi-node environments.

- pR improves end-to-end execution time and scalability compared to the current state-of-the-art, the snow package for R.

2.5.3 Background

snow, multicore, and pR are R packages that provide parallel implementations of various methods from the apply family of R methods. Specifically, snow [56] (Simple Network of Workstations) supports basic parallel computing in R with a relatively simple interface that involves methods from the apply family. snow manages communication between compute nodes using several protocols, including Rmpi [7], an R wrapper around a C++ implementation of the MPI specification (LAM/MPI [45], MPICH2 [46], OpenMPI [47]). End-users can create clusters of compute node, apply functions on them, and retrieve results. snow is maintained by Professor Tierney at the University of Iowa; the current version is 0.3-3 and was released in 2008.

Likewise, multicore [57] supports running a parallel lapply method in R on Unix machines with multiple cores only. It was first released in early 2009, and the current version is 0.1-4. multicore is maintained by Simon Urbanek at AT&T Research Labs. R code written for multicore is intended to improve performance on multi-core machines and is not designed for multi-node environments.
PR is an umbrella framework that contains several parallel computing components for R, including a compiler parallelization approach to automatically parallelize R scripting codes [60][61], and a component that creates a bridge between existing compiled parallel codes and R [8][5]. The work presented in this section augments these approaches by introducing automatic parallelization of data-parallel statistical codes for R in hybrid multi-node and multi-core environments with specific demonstration of its performance using R’s `lapply` method. Throughout the rest of this section, when we refer to PR, we are referring to this component of the PR umbrella framework.

2.5.4 Approach

This section discusses design decisions, including PR’s software architecture and synchronizing the R environment between the end-user node and compute nodes. The following terminology is used throughout this section: the end-user node, rank 0, manages all communication between compute nodes, which have ranks greater than 0. Cores refers to the number of hardware processing units within a system; historically, computers had one core, but recent hardware advancements have trended toward multiple cores per system. We use the term processes to connote the tasks instantiated on the compute nodes.

Software Architecture

There are three layers to the PR software architecture stack, illustrated in Figure 2.8, and each layer exists on both the R end-user node and compute nodes: the R interface, the Environment/Data/Instruction (EDI) layer, and the communication layer. On a compute node, the R interface layer executes the user specified instructions on the node-specific data in the transmitted and loaded R environment.

The EDI layer is responsible for the exchange of the R environment, data, and instructions between the R end-user session and the compute nodes. Specifically, on the end-user side, it broadcasts the R environment and the code instructions from the end-user session to the compute nodes, properly distributes the data, gathers results from the compute nodes, and performs an in-memory transfer of the results to the end-user R interface. On the compute node side, the EDI layer launches an R compute engine, receives the R environment and loads it into the native R engine, receives and passes the corresponding data and instructions, and sends the results back to the EDI layer on the end-user. PR
Figure 2.8: pR software stack for automatic data-parallel statistical computation.

manages the cluster and interacts with the communication layer using MPI; it does not require the R end-user to explicitly deal with these issues. As discussed in Section 2.5.4, pR calls MPI directly in compiled C++ to manage all communications between nodes, thus minimizing the overhead.

The snow and multicore packages follow a similar architectural approach. Their R interfaces are fairly similar to pR’s, except for the distinctions discussed in Section 2.5.4, and snow explicitly exposes cluster management functions (e.g.: makeCluster). snow launches standard R sessions on compute nodes, while multicore forks the parent R session. The snow middleware layer is written entirely in R, while the multicore package calls C code from its custom R scripts to fork compute processes and manage the cluster. For the communication layer snow can interact with MPI via the Rmpi package, a wrapper around MPI functions, within the interpreted R environment, while multicore uses system level pipes.

R Environment Synchronization

In an ideal scenario, a parallel computing environment would automatically identify and transfer all dependencies to the appropriate compute nodes. For example, consider
the following R snow code (see [62] for R syntax):

```r
1 library(snow);
2 library(abind);
3 x <- as.list(1:16);
4 y <- matrix(1:12,3,4);
5 fn <- function()
6  z <- y+100;
7  b = dim(abind(y,z,along=0))
8 }
9
10 cl <- makeCluster(numProcs, type = "MPI")
11 clusterApply(cl, x, fn);
12 stopCluster(cl);
```

Because the compute nodes require the `abind` library and variables `x` and `y`, a simple parallel system would broadcast the entire environment with its loaded libraries and variables to the compute nodes to address these dependencies. In more sophisticated scenarios, the parallel system would distribute only those dependencies that are necessary for the computation by the appropriate process. In an ideal scenario, the system would do the latter, but in a proactive and transparent way.

snow requires end-users to explicitly manage dependencies by identifying and transferring all necessary environment objects to the appropriate compute nodes. Therefore, the above code will not run as shown if it were executed by snow because the required library (`abind`) and variables (`x,y`) do not exist on the compute nodes when function `fn` is called (Line 11). snow users can address these issues by explicitly sending and evaluating dependent libraries (Line 2), variables (Lines 3-4), and function definitions (Lines 5-8) to the cluster nodes, as in the following:

```r
1 clusterEvalQ(cl, library(abind));
2 clusterExport(cl, list(x, y));
3 clusterApply(cl, x, fn);
```

Alternatively, snow users can place the example code in a self-contained parent wrapper function that explicitly loads the required library and creates the variables on each compute node, such as:

```r
1 library(snow);
2 wrapper <- function()
3  library(abind);
```
While this example is relatively straightforward, scenarios can be more complex. Consider an R function that calls another function, which calls another function, and so on. For example, in the BioConductor [24], there are dependency depths of level three and above. snow users must specifically ensure that all of these functions and their dependencies must be available on each compute node.

`pR` shields R users from these R environment synchronization intricacies. Once the required libraries, functions and variables are created within the end-user’s R session and the user calls the `lapply` method, `pR` automatically updates all the compute nodes to use the same environment, thereby creating these entities on the compute nodes. This approach can help attenuate the burden of implementing parallel statistical computing.

However, `pR`'s approach does not achieve the ideal scenario described above because all environment objects are transmitted, even those that may not be necessary. For example, if a user creates a variable in the R session that is not used on compute nodes, then `pR` will unnecessarily transmit the variable. To the best of our knowledge, R does not expose an API to handle dependency analysis, or saving and loading a subset of the environment, in a convenient matter; it only provides an option of saving and loading the environment in its entirety.

Because the `multicore` package is designed for solely multi-core environments, and each process executes on the same system, the package uses the Unix `fork` system call to spawn copies of the current R process. Many modern operating systems use a `copy-on-write` (COW) approach, meaning that each process begins with a pointer to the same R session and is only allocated its own copy when a process tries to write to its session. Therefore, if the compute node does not alter its environment, no additional memory is allocated to it. The COW approach implies that any user interface elements are also shared, which can cause contention between processes. Therefore, the `multicore` authors advise never using
GUIs with the *multicore* package. Moreover, the current version of the *multicore* package enforces a memory copy during process creation.

**Communication Layer**

There are different communication approaches that packages can implement, including sockets, MPI, or system-level pipes if the processes coexist on the same machine. *pR* interfaces with MPI in compiled C++ due to MPI’s widespread availability and design for massively parallel applications. Furthermore, socket communication restrictions are prevalent in secure High-Performance Computing (HPC) environments, and *pR* is designed for multi-node, as well as multi-core, environments, which prevents the use of system level pipes for inter-node communication.

*snow* supports MPI communications by leveraging the Rmpi *R* package that provides a wrapper for MPI communications. While Rmpi makes MPI calls in compiled C, Rmpi users (e.g.: *snow*) interact with the package in the interpreted *R* environment, which introduces some overhead. Therefore, execution will likely take longer than calling the MPI functions directly within a compiled language like C or Fortran, particularly if large amounts of data need to be distributed. *snow* provides other communication mechanisms, including using PVM, sockets, or NetworkSpaces (nws) [63], which face similar concerns because they are also managed through the interpreted *R* environment.

The *multicore* project, which is intended for use on single-system multi-core machines, uses direct process-to-process communication through C level `read()` and `write()` pipe communications between processes. This approach provides good performance, but will not support multi-node environments.

**Ease-of-Use**

Ease-of-use is a paramount concern within parallel statistical computing environments, especially for systems aiming to protect users from the complexities of parallel computing, such as *pR*, which strives to minimize or avoid parallel computing code, discussed in Section 2.5.4. *pR* mimics the *R* function library by overloading the existing *R* `lapply` function with additional parameters for the number of desired compute nodes. This interface was chosen to give *pR* the look-and-feel of native *R* functions and to smooth the transition for *R* users to parallel `lapply` functions. The *pR* approach strives to improve ease-of-use by
not requiring users to explicitly manage compute node environments, such as broadcasting variables and libraries from the end-user node.

In comparison, snow users are responsible for creating and destroying clusters of compute nodes (makeCluster and stopCluster, respectively), sending self-contained functions or explicitly broadcasting dependent libraries, functions and variables to compute nodes (clusterEvalQ and clusterExport, respectively), and calling the desired function to be executed.

The multicore package exposes a custom mclapply method signature, requiring users to change their code to call this method. By default, multicore uses the maximum number of cores on the machine. However, there is no standard way to determine this value and multicore uses eight cores if it cannot assess a machine’s configuration. Users can explicitly specify a different number of cores by setting the mc.cores parameter.

**Memory Footprint**

There is a trend in computer architecture to increase the number of processing cores per system. While available memory is also increasing, it lags behind the pace at which cores are added. Therefore, the available memory per core is actually decreasing [64], which can affect the performance of parallel R packages if the necessary memory is greater than the system memory. For example, if an R session consumes 1 GB of memory on an 8-core machine with 4GB of memory, then the parallel R package is effectively limited to 4 cores if each spawned session creates a copy of the R environment.

As previously discussed, snow does not copy the entire environment and users must explicitly send any dependent data to compute nodes, which can decrease the amount of consumed memory at the cost of user interaction. Both the multicore and pR packages copy the entire environment, however, multicore is limited to a single system, while the pR package can distribute the work over many multi-core machines, thereby mitigating single-system resource restraints.

**Execution Flow**

pR implements a two-tiered execution strategy to take advantage of both multi-node and multi-core environments. In the first tier, pR broadcasts the R environment, the function to evaluate, sends a subset of the data to each compute node using MPI, and then
loads the end-user’s R environment on each compute node. In the second tier, pR uses
Unix’s fork command to create local processes, and each process opens system-level pipes
to read data from the compute node, and write results. This approach is similar in spirit
to the multicore package’s approach. Once each compute node receives its results, pR uses
MPI to send the results back to the end-user node, which assembles the results and returns
them to the R session.

2.5.5 Results

To evaluate the performance of pR, we conducted experiments to compare against
R, and two existing parallel computing packages, multicore and snow, because they are the
most active and robust packages that provide parallel lapply capabilities in multi-core and
multi-node environments, respectively [65]. snow is more established and frequently used
than multicore and it can leverage the Rmpi package to manage communication between
compute nodes.

Two test environments were used: E₁ is a dedicated 64 node Intel-based Infiniband
cluster running Linux kernel 2.6.18. Each node contains two 3.4 GHz Pentium IV processors
and 6GB of memory. The E₂ system also uses Linux kernel 2.6.18 and contains 4 dual-core
AMD processors and 64GB of memory. This system was used for the multi-core experiments
described below. We use the term processes in the results below to connote the processes
instantiated in an environment, regardless of whether they are in the hybrid E₁ or the
multi-core E₂ environments.

Speedup

We calculate the speedup of each of the R packages and compare them to the
ideal. To normalize the comparison, we measure speedup as the relative performance be-
tween single-processor R and the corresponding multi-processor system (i.e., pR, snow, and
multicore), defined as:

\[ S(p) = \frac{T_{\text{serial}}}{T_{\text{parallel}}(p)}, \]  

where \( T_{\text{serial}} \) is the R execution time on one processor and \( T_{\text{parallel}}(p) \) is the execution
time of pR, snow, or multicore using \( p \) processors. The ideal speedup was determined by
dividing the execution time using serial R on one processor by the number of processors.
Figure 2.9: Multi-core speedup for pR, snow, and multicore compared to a serial R implementation. The results were gathered in the E2 environment with 8 cores.

Figure 2.9 depicts the speedup using a testcase that sums 4096 vectors, each of size $2^{24}$. Both the multicore and pR packages perform close to the ideal for 2, 4, and 8 cores, and pR closely reflects multicore’s performance, implying that pR introduces low overhead (see Section 2.5.5). snow yields consistent speedup for 2 and 4 processes, and then speedup decreases. A performance drop-off at 8 cores is observed for all three packages, which may be due to the increased memory latency experienced as the number of cores increases; this latency can consume 30% or more of processing time [66] and further investigation is necessary to evaluate this possibility.

We also evaluated the speedup in the hybrid multi-node and multi-core $E_1$ environment to compare pR to snow, and the results are shown in Figure 2.10. pR closely mirrors the ideal speedup for all processor values and performs better than snow, which appears to level-off at 32 processors. It would be interesting to scale-up the experiment using larger numbers of processors, however, we currently do not have access to such a system. The average performance improvement for pR compared to snow on this testcase was 57%.
Environment Synchronization

While pR’s approach to synchronize R environments between the end-user and compute nodes may deliver a more straightforward end-user experience, we expect this feature to add overhead, which is determined by the size of the data and the number of compute nodes. In our experiments, using environment synchronization to communicate and load a $2^{24}$ vector on 32 compute nodes consumed about a second.

Weak Scaling

Weak scaling is used to determine the degree to which a task can be sped up by increasing the number of processes. To perform weak scaling analysis, we fixed the number of jobs per process to 128, and each job sums a $2^{24}$ element vector. This testcase was executed on the $E_1$ hybrid environment using powers of 2 processes from 1 to 64. Figure 2.11 shows the scaling results. snow performance is always less than the ideal, and trends towards lower performance as the number of processes and jobs increase. pR achieves ideal or better weak scaling and the trend suggests that greater numbers of processes may yield even better weak scaling.
Load Balancing

The \texttt{R lapply} method repeatedly calls the same function on each variable in a list, and the variables are often of similar size and complexity. For this reason, we implemented a block distribution strategy to uniformly allocate tasks to compute nodes and promote load balanced execution. In our experiments, we found that the \texttt{pR} compute nodes consistently yielded load-balanced performance. Figure 2.12 shows the evaluation time for 16 compute nodes that collectively sum 4096 vectors, each of size $2^{24}$.

\texttt{pR} Overhead in Multi-Core Environments

While \texttt{pR} is designed for hybrid multi-core multi-node environments, an interesting question is how it compares to the \texttt{multicore} package in a strictly multi-core environment. We expect some additional overhead due to \texttt{pR}'s support for multi-node environments, and we compared the two packages on the same testcase in the $E_2$ single node, 8 core system to determine what overhead \texttt{pR} may introduce. We evaluate the overhead introduced by \texttt{pR} by calculating the overhead, $O(p)$ as follows:

$$O(p) = pR(p) - M(p),$$

(2.4)

where $pR(p)$ is the total end-to-end execution time using \texttt{pR} on $p$ processors and $M(p)$ is the analogous time using the \texttt{multicore} package. Figure 2.13 depicts the overhead on a
Figure 2.12: \( pR \)'s load-balanced execution in the \( E_1 \) hybrid environment using 16 nodes to sum 4096 vectors with length \( 2^{24} \).

testcase that sleeps 120 times for 4 seconds each. We could have used another testcase, such as one that calculates histograms, but this could introduce other potential sources for timing discrepancies, such as file I/O, on each compute node. By using a sleep testcase, we can more accurately assess \( pR \)'s overhead by limiting other potential performance factors. With ideal scaling, this testcase would take 480, 240, 120, etc. seconds for 1, 2, 4, etc. processes, respectively. The testcase was run on powers of two processes from 1 to 64 on the 8 core \( E_2 \) system and the \( pR \)'s overhead compared to \textit{multicore} was consistently about 0.7 seconds. There is a small discrepancy between the \textit{multicore} package and the ideal due to the overhead induced by forking \( R \) processes, copying variables, etc. This overhead averaged about 3 seconds.

Note that the end-to-end execution time continues to decrease as the number of processes increases, even when the number of processes exceeds the number of available cores (8) on the system. We attribute this performance to the low computational demands of the sleep testcase.
Communication Overhead

We expect $pR$ to yield reduced communication time compared to $snow$ because $pR$ calls MPI functions directly in compiled C++, rather than using Rmpi through interpreted $R$. The improvement should be more pronounced in communication intensive scenarios. Furthermore, when multiple cores are available per system, we expect $pR$’s communication time to approximately equal $snow$’s communication divided by the number of cores because $pR$ will create smaller clusters than $snow$, and then distribute the computations among the cores on each compute node. For example, if a testcase needs 1024 processes and each compute node has 256 cores, such as the picoChip PC200 [67], then $snow$ will create a cluster of size 1024, whereas $pR$ will create a 4 node cluster, and then distribute work among the cores. While the distribution among cores will consume some time, we expect the distribution to be noticeably faster than initializing a larger cluster. This cluster size reduction should impact the time for cluster initialization and teardown, data broadcasts, and all data distribution because the end-user node is communicating with fewer nodes.

As expected, using the hybrid $E_1$ environment in which each compute node has 2 cores, $pR$’s MPI communication overhead averaged 54% less than $snow$’s across powers of
2 processors from 2 to 64.

**Atomic vs. Non-Atomic Execution**

There are two basic cluster execution approaches to consider: *atomic* and *non-atomic*. In the former, the cluster handle is created and destroyed before and after each `lapply` call. This approach grants more flexibility to the user to dynamically create clusters of different sizes, at the expense of repeated initialization and teardown overhead. Different analysis algorithms may scale differently for various data set sizes with the number of processes. Alternatively, the *non-atomic* approach initializes the cluster handle once, performs all the `lapply` function calls, and tears down the cluster handle at the conclusion of the R session. This technique prevents the dynamic allocation of cluster resources, but avoids the repeated overhead introduced by the *atomic* approach.

Parallel codes can scale differently depending on the computational complexity of the data analysis algorithms, problem sizes, etc. Additionally, users may have knowledge about situations in which it is advantageous to use functions and data for a specific number of processors to improve processor utilization efficiency. `snow` creates a persistent cluster using the *non-atomic* approach, which can be beneficial, as described above, but can also introduce unnecessary overhead if a computation requires fewer nodes than the cluster has initialized.

To demonstrate this possible overhead, we designed a testcase that creates a cluster of size 64, and calls `snow`’s `clusterApply` method to generate histograms on 8 integer vectors of length 1024; effectively, only 8 of the 64 compute nodes will be used. We expect that the initialization and teardown will be expensive relative to the execution time. The testcase took 67 seconds, of which 51 seconds, or 76%, represent the cluster initialization and teardown. The same testcase using `snow` that only initializes 8 compute nodes completes in 8 seconds or over 8 times faster. In comparison, `pR`, which uses an *atomic* approach, will only initialize 8 compute nodes for this testcase, rather than 64, because the number of jobs (8) is known at the initialization moment. Alternatively, `snow` users can explicitly re-size clusters.

The large performance discrepancy exhibited by `snow` in the above example is largely due to the time associated with MPI’s `MPI_Intercomm_Merge` method that is called after spawning compute nodes to ensure that every node uses the same MPI commu-
Figure 2.14: Measured and Extrapolated Time to Perform MPI_Intercomm_Merge using OpenMPI.

Figure 2.14 shows both actual and extrapolated execution times to perform MPI_Intercomm_Merge using OpenMPI. 128 nodes takes over 2,000 seconds (30 minutes) to merge communicators, and if this trend continues, then 256 nodes will take over 50,000 seconds (almost 14 hours). Therefore, pR’s approach that creates smaller MPI clusters based on the number of cores available may yield performance improvements.

Map/Reduce

An emerging approach to improving High-Performance Statistical Parallel Computing (HPSPC) scalability and performance involves using distributed Map/Reduce approaches, such as Hadoop [59]. In the Map/Reduce system, mapping involves the end-user node dividing work into sub-problems and distributing the smaller problems to compute nodes. Reducing refers to the end-user node aggregating the compute node generated results. Two primary advantages of this approach are that the computing environment is responsible for providing redundancy and the system can scale to extremely large datasets (∼petabytes), even when compute nodes fail [59].
pR supports Hadoop processing by allowing users to call a pRhadoop method, which passes data to pR’s C environment. pR then calls the Hadoop streaming Java environment with the input filename, and the mapper and reducer codes. Therefore, Map/Reduce users can leverage the powerful R engine for pre and post-processing data.

Rhipe [58] is another R package that offers Hadoop functionality and was released in early 2009. It is designed similarly to pR, although the mapping and reducing scripts must be written in R, which necessitates installing R on all compute nodes. Furthermore, a new R session is created for each map and reduce operation, which can introduce overhead. In comparison, pR allows users to specify any application, including shell scripts or compiled scripts. Rhipe is currently in beta release, and we are in communication with the author to configure the package for our system.

Another related R package, called HadoopStreaming [68], was also released in early 2009 and takes a different approach: it is executed outside of the R environment using a shell script to call the Hadoop streaming Java environment and R is used to perform the actual mapping and reducing operations, which can suffer from the same overhead concerns as Rhipe.

The differences between pR and pRhadoop are analogous to those between any parallel statistical codes and a corresponding Hadoop implementation: the non-Hadoop version is expected to execute more quickly, especially on smaller clusters, due to the absence of Hadoop’s overhead and redundancy. However, the Hadoop version is expected to yield improved large-scale scalability and fault-tolerance.

2.5.6 Related Work

The parallelization sections in this Chapter (2.4 and 2.5) follow complementary parallel approaches. In general, there are two general categories of Parallel Paradigms, as shown in Figure 2.15: implicit and explicit parallelism. In the latter, end-users are responsible for explicitly managing all details of the parallel computation, including designing the parallel algorithm, data distribution, and inter-node communication (ex: MPI). This category grants the end-user the most power and control, at the expense of performance and ease-of-use. Some examples of explicit parallelism within R include Rmpi [7] and rpvm [43].

In contrast, implicit parallelism places the responsibility of managing the parallel statistical computing details on efficient, compiled codes, tools and libraries, rather
than on the end-users. This approach often improves performance and ease-of-use from the end-user perspective, at the expense of on-the-fly prototyping of parallel computing algorithms. Implicit parallelism can be further divided into three groups: task-parallelism, data-parallelism, and hybrid parallelism (task and data parallelism).

Task-parallelism involves a heterogeneous set of tasks that can be performed on the same, or different, datasets in parallel. Task-parallelism can suffer from under-utilized processors due to the common scenario where tasks wait for dependencies to be resolved. An R example of task-parallelism is shown in the following code. Lines 4, 5, and 7-9 each represent an independent task that can be run in parallel:
taskPR [69] is one example of a task-parallel R package; it provides a simple way to detect out-of-order independent execution of R tasks and intelligently delegates each task to remote compute node processes to execute them concurrently.

In contrast to task-parallelism, data-parallelism distributes different subsets of the data across the compute nodes, and performs the same task on each subset. Data-parallelism can be further divided by the amount of inter-process communication: either intense or light-weight. Approaches that involve intense communication include our RScalAPACK package, a specialized parallel R project that exposes ScaLAPACK routines within R. ScaLAPACK is a library of high performance parallel linear algebra routines [48][49], based on the Parallel BLAS [50] and Basic Linear Algebra Communication Subprograms (BLACS) [51] libraries. RScalAPACK is developed and maintained by Professor Samatova and her lab at Oak Ridge National Laboratory (http://cran.r-project.org/web/packages/RScalAPACK/index.html). While RScalAPACK is specific to ScaLAPACK, our pR component presented in [8] generalized the approach and is presented in Section 2.3.

In contrast to intense communication, data-parallel approaches with little or no communication, sometimes called embarrassingly parallel, include parallel implementations of R’s apply family of methods, such as snow, multicore, and the pR component presented in Section 2.5.

The hybrid-parallelism approach combines both task and data parallelism, and yields the advantages of both, but can still suffer from the under-utilized processors issue introduced by task-parallelism. In our previous work, we investigated a simple hybrid case by implementing a loop-unrolling strategy to achieve parallelism [69][61].

```r
1  y = sample(1:1000, 100);
2  x = sample(1:1000, 100);
3
4  hist(x);
5  s = svd(x);
6
7  d = dist(y);
8  hclust(d)
9  plot(d);
```
2.5.7 Automatic Parallelization Conclusion

This section presented an extension to the \textit{pR} package that introduces automatic parallelization of the \textit{R lapply} method for data-parallel statistical methods in hybrid multi-core and multi-node environments. Related packages include the \textit{multicore} package for multi-core environments, and the \textit{snow} package designed for multi-node environments. As demonstrated using \textit{pR}'s implementation of \textit{R}'s \textit{lapply} method, \textit{pR} achieves improved overall end-to-end performance by utilizing a two-tier execution strategy. The first tier allocates work to nodes in a multi-node infrastructure, and the second partitions the work for local execution on individual cores in multi-core environments. The performance results in Section 2.4.4 demonstrate that \textit{pR} can outperform \textit{snow} in hybrid multi-core and multi-node environments. Specifically, \textit{pR} achieved an average performance improvement of over 50\% compared to \textit{snow} in our tests.
Chapter 3

Defining Data Analysis Pipelines through WER

3.1 Motivation

In the previous chapter the research primarily dealt with how to execute data analysis scripts in the most efficient manner, using $R$. The proposed $pR$ plugin provided a mostly $R$ agnostic interface for third-party developers to integrate with $R$, while minimizing memory overhead.

This chapter presents research on the Web-enabled analytics component in the Statistical Data Analysis Framework (SDAF). The overarching research goal is to design and implement a prototype, called Web-Enabled R (WER) [3], that enables large-scale high-performance data analytics and user collaboration accessible via the Web. As indicated in Section 1.2, there is a pressing need in the scientific community for Web-enabled access to mature statistical engines, such as $R$, to address the growing complexity and scale of data-intensive high-performance statistical computing. Web-enabled tools untether scientists from their local machines, allow organizations to support powerful machine configurations that possibly enable complex computing (e.g., parallel computing), and provide a maturing environment for online collaboration.

Many scientists use $R$ [2], the most statistically rich and commonly used open-source platform available. However, $R$ only runs locally, which limits scientists to their local resources and impedes data and code sharing with other scientists. A few Web-enabled $R$
projects, such as Rpad, R-php and Rweb, have recently emerged (see Section 3.3.2). However, their architectural design strategies offer relatively simple copy-paste-submit interfaces through HTML Web pages with capabilities far below the ones provided by the stand-alone R application interface.

3.2 Major Contributions

In contrast to existing projects, the Statistical Data Analysis Framework (SDAF) Web component, called Web-Enabled R (WER), was designed and implemented to offer a dynamic, interactive, collaborative remote Web interface to R. The major contributions of the research study underlying WER are the following:

- Proposed and designed a remote execution framework for R that highly improves the performance and usability of R over the Web.
- Proposed and designed a memory management system that greatly minimizes memory requirements and enables previously unparalleled Web-enabled R multi-user scalability.
- Proposed and designed a data analysis results and analysis code management system that supports collaboration and sharing.

The proposed advancements have enabled WER to support all the existing features in R through the Web interface and to include various additional features to ensure ease-of-use and reliable performance. To the best of our knowledge, this is the first Web-enabled R system that provides the following two fundamental advancements, from the users’ perspective:

**Supports interactive remote access to R servers.** Existing tools use simple Web architectures that effectively prevent interactivity and collaboration.

**Enables users to share data, results and analysis sessions,** thereby encouraging team and interdisciplinary work.

All data resides on the server with WER, therefore cross-application data sharing is possible with other resources on the server. For example, if a statistical result is saved on
the server, a visualization program that also resides on the server can immediately leverage
the results.

A comparison of the existing technologies to the one presented herein suggests
that WER outperforms the existing approaches for many of the performance evaluation
requirements including those of user scalability and collaboration.

The manuscript summarizing this work has been accepted into the WORLDCOMP
International Conference on Information and Knowledge Engineering (IKE) 2009 [3] (See
Section 1.5). The particular application of WER in the context of the Kepler Scientific
Workflow system [70] has been demonstrated as part of the SuperComputing 2007 Confer-
ence Tutorial [19].

The rest of the chapter is organized as follows: Section 3.3 discusses R and other
Web R projects. Section 3.4 discusses the proposed WER architecture and justifies the
design decisions. Section 3.5 evaluates WER and compares it to the other Web R projects.
Section 3.6 discusses a real-world application that integrates WER with Kepler using Web
services. Section 3.7 concludes the chapter.

3.3 Background

3.3.1 R User Interface

The R statistical framework, introduced in Section 2.3.2, provides a front-end
desktop Graphical User Interface (GUI) called RGui, shown in Figure 3.1, that runs on a
user’s local machine. RGui contains a terminal window for R code that runs against the R
Engine on the local machine; remote access is not supported.

There are two execution modes supported by R: interactive and batch. In the
interactive mode, R runs via a command-line interface, or the RGui, and the user retrieves
the results from the standard output by typing variable names and commands. In batch
mode, the user supplies a file system path to R that contains the R script. Results can be
written into output files or retrieved from the standard output as in the interactive mode.

R is not thread-safe, and therefore allowing concurrent user sessions to access the
same R Engine creates an unstable environment. In order to support concurrent users,
each user session requires a unique R instance on the server. This requirement introduces
memory overhead into the R server, which the Web-enabled R projects must handle to
Figure 3.1: The RGui interface. The left window is the terminal and the right window is the graphical output from the R commands.

provide multi-user scalability.

3.3.2 Existing Web-Enabled R Projects

Several projects exist that provide Web-enabled access to R. A systematic comparison of WER to these projects is presented in the following sections. Here is a brief introduction to the most advanced and actively developed Web-enabled R projects: Rpad [71, 72], R-php [73, 74] and Rweb [75, 76], shown in Table 3.1; they will be referred to as projects throughout this chapter. The details are described comparatively with the proposed WER project in the sections that follow. Rpad is an interactive, Web-based analysis program that provides interactive workbook-type sheets that leverage R. It is also the only project that supports custom Web pages. R-php is a Web-oriented statistical software project that uses PHP: Hypertext Preprocessor (PHP) and MySQL to provide a textbox front-end where users can input R scripts and click a button to execute the commands on the R Engine. Rweb is a Web-based interface to R that executes submitted code against the R Engine and returns the text and graphical output in the Web browser.
Table 3.1: Web-Enabled R Projects

<table>
<thead>
<tr>
<th>WER</th>
<th>Maintainer</th>
<th>Version</th>
<th>Download URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rpad</td>
<td>Electric Power Research Institute (EPRI)</td>
<td>1.3.0</td>
<td><a href="http://www.rpad.org/downloads/Rpad_1.3.0.tar.gz">http://www.rpad.org/downloads/Rpad_1.3.0.tar.gz</a></td>
</tr>
<tr>
<td>Rweb</td>
<td>Mathematics Dept. University of Montana</td>
<td>1.03</td>
<td><a href="http://bayes.math.montana.edu/Rweb/Rweb1.03.tar.gz">http://bayes.math.montana.edu/Rweb/Rweb1.03.tar.gz</a></td>
</tr>
</tbody>
</table>

### 3.4 Architecture for WER

Figure 3.2 presents the WER architecture, which we will also use to make comparisons to the existing Web-enabled R projects when relevant. The Browser block on the front-end side of Figure 3.2 represents the user’s Web browser. The R projects all support multiple browsers including Internet Explorer, FireFox and Safari, the three most popular browsers, although Rpad requires a special plugin to enable text input.

The back-end Web Server block handles Hypertext Transfer Protocol (HTTP) traffic on the server. The projects all specify Apache Tomcat, although theoretically other products can be used, such as Microsoft’s Internet Information Services (IIS).

The R Interface block communicates the user’s R commands to the R Engine and returns the results to the Web Server. The R Engine block is the component that processes the R commands and the R Initialization block starts the R Engine. There are two types of interfaces to R: batch and integrated. The batch interface, implemented by R-php and Rweb, uses system calls to invoke the R Engine in batch mode. The R Engine writes text and image output to files and every time the R Interface block is called, the R Initialization block is executed and launches a new instance of the R Engine. If a user submits \( a=1 \), and then submits \( b=a \), the second command will return ‘Error: object “a” is not found’ because \( a \) was created in a different R Engine, which has terminated. This approach is straightforward to implement, but limits interactivity with R and introduces file system latency.

Instead, WER uses the integrated interface that is tightly coupled to the R Engine and only executes the R Initialization block once per user Web session, in contrast to the batch interface. In the \( b=a \) example, \( b \) will equal 1 because both commands are executed
Figure 3.2: WER Component Architecture. The dotted transition lines and the Graphics Generation and Disk I/O blocks represent transitions and blocks that are not always executed. The front-end Web Browser interacts with the Web Server, which manages all HTTP traffic. The R Interface manages interaction with R, including the R Initialization that starts the R Engine, which processes the R commands. Disk I/O is used whenever files are written to or read from the file system. Some of the existing projects rely heavily on Disk I/O, especially when graphics generation is required to transform image formats, sizes, or orientations.

within the same R Engine and the variable a still exists; this is called R Environment Persistence. WER closes the R Engine when the Web session timeouts.

WER uses the R Engine to output PNG files (except in rare software configurations where PNG files are not supported), which are directly Web accessible without the need to apply image post-processing. As shown in Section 3.5.1, the graphics generation in the other projects incurs noticeable overhead that delays response times. When the graphics generation is complete, the R interface returns the text and image results to the Web server, which relays the resultant Web page to the front-end Browser. The Disk I/O block is used whenever the R Engine produces images.

Section 3.5 further discusses WER features using the components described in Figure 3.2.
3.4.1 Back-End

The requirements for the WER back-end design include support for simultaneous \( R \) environment sessions and using multiple servers. The back-end is divided into two components to meet these requirements: the Web Server and the \( R \) Engine. The components can reside on the same or different systems. The Web Server and WER interface to \( R \) are written in Java to provide a uniform technology, rather than using separate platforms on the front and back-ends, like PHP Hypertext Preprocessor (PHP), on the front-end and Java on the back-end.

3.4.2 Front-End

The WER front-end terminal behavior was designed to mimic the \( RGui \) terminal to provide consistent interfaces for \( R \) users. WER supports additional features to improve the Web experience, which are described below, such as sharing \( R \) environments between users.

Implementing a front-end that adheres to this architecture requires looking beyond the classic Web application model from the past decade because they are inherently disruptive: user actions trigger Hypertext Transfer Protocol (HTTP) requests that are sent to the server, which responds with a new Hypertext Markup Language (HTML) page. This model prevents a continuous user experience because every user request results in a new Web page. Instead, Asynchronous JavaScript and XML (AJAX) and Dynamic HTML (DHTML) were used to provide a rich, continuous and interactive Web application that meets the design requirements. In AJAX, user actions that generate an HTTP request in the classic model instead call the AJAX engine using JavaScript. The engine evaluates the call and handles any requests that do not require server interaction, such as data validation, editing data in memory, etc. If server communication is necessary, the engine asynchronously calls the server without interrupting the user’s interaction with the application.

The WER GUI is shown in Figure 3.3 and the main features are described below.

**Terminal Window:** Emulates the \( RGui \) terminal interface, provides command history via the arrow keys, user specific color settings, cursor handling and customization, raw text cut/copy/paste, and others.

**Help Pane:** Standard \( R \) help documentation is available to the user.
Figure 3.3: The WER browser interface. The bottom panel is a scratch pad area where users can develop R scripts. The upper left panel is the terminal window that mimics the RGui terminal window. The upper right panel is the R help pane.

**Scratch Pad Area:** Users can develop and execute R commands in a text editor pane.

**Share WER Session:** Users can send an email to a collaborator that contains a URL link to the current user’s WER session, therefore allowing multiple users to share the same R environment.

**Publish Common Scripts:** Users can publish common scripts under the *Examples* menu to allow research groups to share scripts.

**Offline Access:** Users can download their R environment and run it on a local R installation. Users can also upload an R environment to WER and continue using the environment. This is a particularly useful feature if users are traveling.

### 3.5 Evaluation of WER

WER is intended to represent the next generation of Web-enabled R projects, help foster new levels of scientific user collaboration within the R community, encourage R to play a more active role in general data flow solutions, and address the growing issues
related to data intensive high-performance parallel statistics. This section evaluates WER by conducting a performance analysis and comparing the results to the other projects, followed by outlining and comparing WER features to the other projects.

3.5.1 System Evaluation

Performance

A test framework was developed to compare WER’s performance to the other projects. The framework reads an XML file of R commands, executes each command on each project, and logs numerous metrics, such as the test duration, average response time per request, etc. The results were gathered on the same server and are organized in two sections: end-to-end execution time and scalability in the number of users. In the former, only one request is sent to the server at a time in order to compare WER to the other projects under simple one-command scenarios using both null and image commands. The second section involves scalability results where the projects were stressed with simultaneous user requests to compare overall performance between WER and the other projects.

End-To-End Execution Time  The following testcases compare the overhead associated with WER to the other projects using testcases identified to exercise different aspects of the Web interfaces.

Null Single Command: executes a single R command, \( a=1 \), to evaluate the overhead incurred due to the \( R \) Initialization block. The simple nature of the command effectively ignores the \( R \) Engine.

Image Single Command: accessing the system’s graphics device to produce images is an expensive operation. This testcase executes a single R command, \( \text{plot}(1) \), that produces an image output file. This test primarily compares WER’s graphics generation to the other projects.

Null Multi-Commands: performs the same as the Null Single Command testcase, except that each user executes the same command multiple times in serial. This compares WER’s \( R \) Environment Persistence to Rpad’s persistence implementation, and to R-php and Rweb, which lack environment persistence and initialize \( R \) for each user request.
**Image Multi-Commands**: performs the same as the Image Single Command, and is analogous to the Null Multi-Commands testcase, except the R command is `plot(1)`.

The Multi-Command testcases were chosen to compare WER to projects that use batch mode (Rweb, R-php) and to evaluate WER’s implementation by comparing against Rpad’s interactive implementation. WER was expected to perform faster than the batch mode projects after the initial command because the R connection is already open; in batch mode, the projects must open a new R connection for each test case, therefore increasing response times.

The testcases were executed on all projects and the test framework executes the R requests serially to simulate a single user; these testcases are not load tests. The tests were run five times and the average results are shown in Figure 3.4. WER is the fastest on the image testcases, and second to R-php for the null testcases. Rweb is consistently third and Rpad is clearly the slowest with an average of about two seconds per request. It is also the only project that relies on an intermediate project to communicate with the R Engine, the Statistics::R Perl package, which probably causes the slower response times. An increase in response times was expected for Rweb and R-php on the multi testcases because both interact with R in batch mode. However, this conjecture was disproved, which implies that spawning a new R thread does not introduce significant overhead. Despite the performance indifference, R Environment Persistence is recommended over batch mode for user experience reasons discussed below in Section 3.5.1.

The Graphics Generation block can be an expensive operation. WER and Rpad generate Web-friendly images directly from the R engine, and therefore experience better graphics performance than R-php and Rweb, which output images in PostScript format and then use third-party tools to convert the images into Web-friendly formats. This conversion process is time consuming and results in overall response time increases of 8% for Rweb and 60% for R-php, because R-php executes multiple steps to convert images into a Web accessible format, whereas Rweb only requires one conversion.

**Scalability in the Number of Users** Remote applications allow multiple users to access shared resources and therefore must handle the issues that arise in multi-user environments, including the ability to handle different numbers of users. Scalability is crucial to the success and viability of any server application, especially within the HPC domain due to
the extensive resource requirements. R is not threadsafe, so this issue must be thoroughly addressed by any enterprise level release. To test project scalability in terms of multiple users, the Null Multi-Commands testcase was used to simulate 2, 4, 8, 16, and 32 users making simultaneous calls to the server. This testcase was chosen because it emulates user scenarios by submitting several commands to the Web Server during each user session. Results are shown in Figure 3.5. WER had the best performance, and Rweb and R-php both outperformed Rpad, which was the only project to experience timeouts. The percentage of requests that timed-out for 2, 4, 8, 16, and 32 users was 1.36%, 14.29%, 18.51%, 26.71%, and 63.00%, respectively. For the 32 user experiment, response times increased from the single user Null Multi-Commands testcase results by only 506% while the other projects experienced 3151%, 1932% and 1775% for Rpad, R-php and Rweb, respectively.

Scalability of Dataset Size Web-enabled R projects should manage large datasets, including identifying and gracefully handling situations where the size of the data exceeds the available resources. For example, what happens if the Web session times out while...
uploading a large dataset to the server? What if the data is bigger than the available storage space on the server machine? Even more pressing, what happens if the Web project tries to display the dataset to the user and it is larger than the available memory on the server? Displaying large amounts of data in a user’s browser has the potential to negatively affect their system by excessively consuming memory resources. Neither the current WER implementation, nor the other projects, address these issues, but the topic deserves investigation.

**Scalability in the Number of Parallel Computing Processors** One of the primary motivations for remote R access is to address the rapid growth in computation demands as datasets increase in size and complexity. Parallel computing must play an active role to achieve analysis tractability by incorporating projects such as pR [5, 8], RScaLAPACK [4], which connects R to the popular ScaLAPACK library, rpvm [6] and Rmpi [7, 77], which directly integrate with MPI and PVM, the two standard parallel programming interfaces, and snow [56], which supports embarrassingly parallel statistical computations.
WER does not currently support parallel computation, and neither do the other projects. Local \textit{R} installations are limited by the locally connected resources, but if remote \textit{R} access is widespread, organizations will be able to configure \textit{R} servers with diverse computing capabilities, at which point parallel computation becomes much more feasible and there will be a profound need to address this functionality.

**Memory Efficiency** Supporting multiple users requires WER to efficiently handle memory on both the client and server machines. The \textit{R Engine} may consume large amounts of memory during complex analyses on large datasets, and tight-coupling between the \textit{R Engine} and the projects can mitigate the impact. WER addresses this issue by using a custom \textit{R} build that minimizes the per instance memory requirements; a typical idle \textit{R} instance consumes less than 2MB of memory. The other Web-enabled \textit{R} projects do not specifically address memory efficiency, which limits their ability to support scalable performance.

**Back-End interface**

WER provides a back-end remote interface to \textit{R} using Web services that allows scientists to leverage statistical analysis routines within their code. The other projects restrict remote \textit{R} users to their Web GUI interfaces.

**Web Services** WER supports back-end access to the \textit{R Engine} via Web services that can be consumed programmatically. The other projects do not offer this functionality. This work was performed with help from the Kepler Scientific Workflow [70] group and a set of Web services was implemented to interact with the \textit{R Engine} (see Section 3.6.1). The Kepler workflow sends \textit{R} commands to our Web services, which submits an HTTP POST to a custom page that contains a text input box and a submit button. The Web service programmatically enters \textit{R} commands into the textbox and clicks the submit button, which launches the WER on the \textit{R} commands. The Web service waits for the Web page to load with text and any image results and extracts the text results from the underlying HyperText Markup Language (HTML). It also identifies any output images and locates their source URL. The Web service returns the text results and an array of image URLs to Kepler, which downloads the image results to the local machine. From the user’s perspective, there is transparency between using Kepler’s native \textit{R} interface and using remote \textit{R} Web services. In both cases, the user specifies the input and receives the results, and is not affected by
the location of the $R$ Engine. Admittedly, there are ample opportunities to improve upon this solution by providing tightly integrated $R$ Web services.

**Client/Server Interface** Client/Server applications are often more feature-rich than corresponding Web-enabled applications due to the technological limitations of Web technologies. However, the gap between the two approaches is decreasing as Web-enabled technologies mature and support expanding feature sets. Client/Server interfaces still provide a richer environment, enable remote execution, and remain a viable Web-enabled $R$ project direction, despite portability limitations. Neither WER nor the other projects offer a client/server implementation.

**User environment**

**Secure User Environment** Security must be preserved and enforced in a multi-user environment, especially within user collaboration enabled spaces. Shared data is often sensitive or restricted and security breaches are unacceptable in any professional environment.

Unlike the other projects, which do not address most of these issues, WER enforces a sandbox per user session and data files are preserved so they can be accessed at a future date. Users are required to login and SSL is supported to encourage a secure environment. $Rpad$ allows full access to all $R$ and system shell commands, thereby opening a security hole on the server. $Rweb$ allows the same actions, although it deletes all files created in a session, unlike the other projects. Therefore, if there was any sensitive information in the input or output files, $Rweb$ prevents anyone from accessing them, although this approach impedes user collaboration. $R$-php uses a local MySQL database to store a list of predetermined dangerous commands that includes `basename`, `write`, `parse`, and 167 others. While extensive, this list is probably not exhaustive, although a system administrator can insert any additional commands.

**$R$ Environment Persistence** As described in Section 2.3.2, there are two approaches to managing the $R$ Environment: batch and interactive modes. The interactive interface tightly couples the $R$ Engine block in Figure 3.2 with the project and only executes $R$ Initialization once per user Web session, in contrast to the batch interface. Therefore, all previous variables and results are available in the $R$ Engine during subsequent commands
in the same user session. This is called *R Environment Persistence* and is the approach implemented by WER.

**Preserve Results on Server** To support future data retrieval and collaboration, users should have the ability to preserve results on the server for future use, which WER supports. *Rpad* and *R-php* permanently store all results on the server, never allow users to delete data, but also do not provide users with a means to access the data. In contrast, *Rweb* deliberately deletes all results from the server.

### 3.5.2 Usability Evaluation

**Collaboration**

**User collaboration** User collaboration is fundamental to future Web-enabled *R* projects. Science is becoming more distributed and scientific tools, such as *R*, must respond to user requirements and support user collaboration. Users separated by geography should be able to collaborate in the same *R* environment, view the same data, run time-intensive commands asynchronously, and have access to previously generated data and *R* analysis scripts; this list is not exhaustive.

WER contains several features to support user collaboration, such as allowing users to download their *R* environment, which they can share with other *R* users. Additionally, scientists can share their current *R* environment by sending an email within WER to their collaborators. The email contains a link that allows different users to login to the exact same *R* environment as the original user. For example, if user A invites user B, B joins A’s *R* environment, and A creates a new variable, then B can also see the same variable.

In contrast, the *Rpad*, *Rweb* and *R-php* projects support minimal data sharing. *Rpad* developers can build Web pages that contain predetermined variables in the form of textboxes, dropdowns, etc. The developers can supply the data or allow users to provide their own by uploading a file or directly entering data into a textbox. For example, consider a state water drought dataset. An *Rpad* developer can create a page with this data, and let the user specify values for predetermined variables, such as the number of residents, the average water use per resident, etc. This page is available to all users with access to the server, so the data is shared among them, but it is controlled by the developer and users have no control over what information is shared. However, according to the *Rpad*
author, the default server installation does not support remote connections: “For security reasons, the server can only run for a local client. However, it is very easy to eliminate this limitation by hacking the startSocketServer() function in the svSocket package." Therefore, in the drought dataset example, data can only be shared with remote users if the ‘svSocket’ package is modified to allow remote access.

*R-php* expressly states that they do not support data sharing. A temporary directory is created for each user Web session and all files created during the session are placed within it. The files are never removed. They could theoretically be shared since they exist in a Web accessible directory, but *R-php* does not provide information regarding the location or existence of these files to the user, which effectively eliminates the ability to share them. *R-php* does allow the user to save images in PDF format, which the user can then share using conventional means, such as placing them in a shared folder or emailing them to a group, but these practices are independent of *R-php*.

*Rweb* places generated files from all users in the same *tmp* directory and immediately deletes them as soon as they are rendered in the browser. Because *Rweb* cannot know precisely when the image files are rendered, it passes the images to a Perl script that opens and sends them to the browser. In this manner the script can ensure that the image files are no longer needed and delete them. *Rweb* does not support file saving, although a user can right-click the image on the Web page and save it themselves on their local machine.

**Access to Preserved Results on Server** WER allows users to retrieve stored results. However, of the other projects, only *Rp* and *R-php* preserve results, and neither support future retrieval.

**Share Results on Server** Users should have the ability to share preserved results with collaborators. WER does not provide this functionality, but this is viewed as a feasible feature to implement. The other projects also do not support this feature.

**Interactive R Interface** WER user sessions maintain *R Environment Persistence* rather than a *Batch R Interface*, discussed in Section 3.5.1. Otherwise, previously used variables are inaccessible, which unnecessarily disrupts the user session. *Batch* mode is not intuitive for a user executing a series of commands and does not mimic the standard *R* terminal. *Rp* is the only other project that supports this feature.
Continuous User Experience  There are two possible categories of browser front-end user experiences: disruptive and continuous. In the disruptive approach, the user enters R commands into a text area, hits a Submit button, and waits for the results. In order to submit new R commands, the user must navigate to the input Web page using the browser Back button, or click in the text-area with the commands, depending on the project, and repeat the input and submission process.

In the continuous approach, R commands are processed as the user types into the Web page and hits the Enter key. The Web page immediately presents the user with the command prompt and the user is able to continue entering commands. Ideally, the continuous behavior mimics the RGui terminal interface to support continuous commands while providing R users with the same environment to which they are accustomed. To the best of our knowledge, WER is currently the only project that supports a continuous front-end; Rpad, Rweb and R-php all implement disruptive front-ends.

Offline Access  One of the primary limitations to Web-enabled technology is that it requires a network connection, which may be unavailable, for example, during travel. Therefore, a robust Web-enabled R project should allow users to download their session to a local machine and load the session into a local installation of R. When a network connection is available, users should have the option to upload their current session to the server and continue working and collaborating with others. WER accomplishes offline access by allowing users to download their R session. None of the other projects support offline access.

R Environment Sharing  Collaboration involves more than sharing results and should include sharing R environments in real-time. If multiple collaborators login to the project they should have the option to join the same R environment to allow them to develop scripts and perform analyses together. WER allows users to invite other users to share the same R environment, while none of the other current projects support this feature.

Data input

There are multiple ways Web-enabled R projects can support data input, including through Web browsers, client/server architectures, databases, Web services, etc. To support the growing R community, future projects should implement multiple approaches that cover both front-end and back-end implementations.
WER allows users to upload data files and issue R commands against them as if the files were local. It behaves the same as RGui and does not introduce third-party assumptions or file formats. It also provides back-end programmatic access via robust Web services, which developers are encouraged to use. A discussion of the other projects’ data input approaches is below.

Rpad’s primary strength is the ability for Rpad developers to design custom Web pages that leverage R on the back-end, without requiring any R specific knowledge from the Web page users. For example, Rpad can support user specified R commands if an Rpad developer builds a Web page with a textbox for user commands, and otherwise this type of data input is not available.

Rpad scans the root Web directory for any Rpad files (.Rpad, .HTML, .Rdata, .R); if any are found, it displays them as links on the Web page. If the user clicks one of the links, the file is loaded. .Rpad and .HTML files are loaded into the browser and .Rdata and .R files are loaded directly into the R environment for processing.

There are two experimental interfaces in Rpad called Basic and Alternate. They support What You See Is What You Get (WYSIWYG) editable Web pages that allow users to directly manipulate the text on the page. They require browsers that support the contentEditable feature, which currently is not standard. However, the Rpad author acknowledges that these editors have open issues: “In the default configuration, Rpad pages no longer have default in-page editing. As such, Rpad pages are normally created with an external text or HTML editor. In-browser editing is still too immature”[71]. Changing between interface versions involves opening each Web page file and changing an attribute value. Therefore, it is unlikely users will switch between versions.

Rweb supports uploading a dataset by specifying a URL. It copies the entire dataset file to the server, which may be convenient for small datasets, but is not an efficient way to share large datasets. All the Web projects allow users to upload dataset files to the server, but Rweb uses spaces as the field delimiter within the files, so users must ensure that all cell values are concatenated. For example, if a column is titled “Average Growth”, it must be renamed to “AverageGrowth”, or a similar name without spaces, before it is uploaded to Rweb.

Rweb supports three versions, a standard form-based Web interface, a more advanced JavaScript version, and a Point & Click (P&C) module that does not require R domain knowledge. Each version handles data output differently. After hitting the Submit
button, the standard interface brings the user to a new page that contains the \textit{R} environment input and output. If there are any images to display, they appear below the I/O text. If the user wants to submit another set of commands, she must click the \textit{Back} button in the browser, enter the new commands, and click \textit{Submit} again.

In the JavaScript version, when the user clicks the \textit{Submit} button, \textit{Rweb} launches a new window to display the input to \textit{R}, another window for the text output, and if there are any graphics, they appear in a third new window. Using a Web browser that supports tabs, \textit{Rweb} still launches new windows. Therefore, after a command that involves graphics, there are four windows for the user to manage.

The Web module version is similar to the standard version: after inputting the data and selecting the module to perform (e.g., Probability, Regression) it brings the user to a new page where she can input values for module specific parameters. When the user clicks \textit{Submit}, she is brought to another page with the results. If the user wants to resubmit new data, she must use the \textit{Back} button in the browser. In all versions, \textit{Rweb} converts the PostScript image files that \textit{R} creates into Web compatible GIF files using the \textit{Netpbm} \cite{79} package.

\textit{R-php} supports uploading datasets, but not \textit{R} commands, by specifying a file path. Therefore, users cannot write a series of commands and submit them using the upload utility. The \textit{R-php} P&C interface allows users to interact with the uploaded data by clicking on each data cell and changing the value using an elegant interface. This feature is useful for small changes but is cumbersome for large modifications. \textit{R-php} identifies all commas, tabs and spaces as delimiters in the dataset files. Therefore, if a dataset contains any of these characters within data fields, they must be removed before uploading the data to the server.

WER supports the data input features listed below:

\textbf{Web Browser Text Input}  Allow users to type \textit{R} commands directly into the browser page. \textit{Rpad}, \textit{Rweb}, and \textit{R-php} also contain this feature. Additionally, \textit{Rweb} allows users to upload a dataset by specifying a URL to the data.

\textbf{File Upload}  Support file uploading through the browser. The other projects also support this feature.
**P&C Web Modules**  
*Point & Click* (P&C) modules allow non-*R* users to leverage the *R Engine* through prefabricated Web pages that provide specific predetermined functionality, such as Linear Regression. While WER does not attempt to implement these custom interfaces because they are deemed project or domain specific, the other projects do support them, although the interfaces are less mature and are specialized for the projects’ audiences.

**Data Output**

Analogous to data input, WER also supports both front and back-end data output interfaces, using both the Web browser front-end and the Web services. The other projects only support front-end data output approaches.

**Web Browser Text Output**  
WER displays the *R Engine* text output after the *R* command script completes. The other projects also display results in the Web browser.

**Web Browser Image Output**  
If the *R* script generates any images, WER displays them on the Web browser, as do the other projects.

*Rpad* uses an advanced output technology that allows an *Rpad* developer to install the *R Imagemap Creator* [80] package to create clickable image maps that support drill-down behavior within *R* graphics. *Rpad* is not capable of determining when image output is generated and requires users or developers to call `HTMLplot()` in the *R2HTML* [81] library when graphical output is expected from *R*. This illustrates the need for *Rpad* developers, or advanced *Rpad* users, to understand the differences between *R* code and *Rpad* code to use *Rpad* correctly.

**3.5.3 General Features and Project Comparison**

**General**

**Technology**  
It is crucial to select technologies for Web-enabled *R* projects that can support the outlined features. Possible existing technologies that can support this functionality include AJAX and Adobe Flash. WER uses AJAX due to its lower memory requirements and wider accessibility. *Rpad*, *Rweb* and *R-php* leverage Perl, which can also support AJAX, although the current implementations of the projects do not incorporate AJAX.
Asynchronous Execution  High-performance parallel computing over large data often involves complex analyses that require extensive CPU hours. This creates a need for asynchronous execution to allow users to submit their commands, log off of the system, and later return to check the execution status and retrieve results. None of the current Web-enabled projects allow execution during an inactive user session; if the user session ends for any reason, the analysis output is lost.

3.5.4 Feature Comparison of WER to other Web-enabled $R$ projects

Tables 3.2, 3.3 and 3.4 summarize the aforementioned comparison of WER to the other Web-enabled $R$ projects by system, usability and general features. WER is the only Web-enabled $R$ project that implements the majority of the features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Rpad</th>
<th>Rweb</th>
<th>R-php</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td></td>
<td></td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>End-To-End Execution Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalability in the Number of Users</td>
<td></td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalability of Dataset Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalability in the Number of Parallel Computing Processors</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Efficiency</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Back-End Interface</td>
<td></td>
<td></td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Web Services</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client/Server Interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Environment</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure User Environment</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R$ Environment Persistence</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Preserve Results on Server</td>
<td>×</td>
<td></td>
<td>×</td>
<td></td>
</tr>
</tbody>
</table>

3.6 Applications

There are two steps to enable remote Web data analysis environments: first, users must have access to their data, and second, they perform analysis routines on the data.
Table 3.3: Usability Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Rp</th>
<th>Rweb</th>
<th>R-php</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Collaboration &amp; Interactivity</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Collaboration</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access to Preserved Results on Server</td>
<td></td>
<td></td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>Share Results on Server</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactive R Interface</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continuous User Experience</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Offline Access</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R Environment Sharing</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Input</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Browser Text Input</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Upload Dataset by URL</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File Upload</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Point &amp; Click (P&amp;C) Web Modules</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Data Output</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Browser Text Output</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Web Browser Image Output</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 3.4: General Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Rp</th>
<th>Rweb</th>
<th>R-php</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td>WYSIYWG, Perl</td>
<td>Perl</td>
<td>Perl</td>
<td>AJAX, Java</td>
</tr>
<tr>
<td>Asynchronous Execution</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.6.1 Web Services Based Integration with Kepler Workflow System

The second step, performing data analysis, is addressed by integrating WER with the Kepler scientific data analysis pipeline project. Scientific workflows often involve extremely large datasets and require complex analysis. Workflows that leverage the R statistical and graphing platform are limited by the specifications of the local machine because R only runs locally. WER supports a set of Web services that facilitate integration with other software projects, including Kepler, by enabling data uploading and submitting R commands for execution. Kepler contains an RExpression actor that interacts with a local R installation. The RExpression actor was extended to support WER by adding two options to the RExpression parameter window: a Boolean Use Remote Server and a String Remote Server URL. The first parameter specifies whether the user wants to use WER
and defaults to False, and the second allows the user to enter the location of the WER server. Figure 3.6 below shows the RExpression parameter window and the new options are outlined by the red border. If the workflow designer selects WER in the actor, the backend calls the WER Web services to submit the R commands, and retrieves the results. This process is seamlessly hidden from the user.

3.7 Conclusion

Members of the scientific computing community are increasingly interdisciplinary and geographically separated, which increases the need and difficulty of sharing High-Performance Computing (HPC) data analytic results. Enabling scientists to leverage shared resources addresses these issues by providing a common destination for results and analysis routines. It also creates new opportunities for user collaboration because group members have access to the same resources. For example, multiple members can log into the same data analysis session and contribute ideas and techniques for the problem at hand.

Many scientists use the R [2] statistical platform. However, R only runs locally,
which limits scientists to their local resources and impedes data sharing with other scientists. This research proposes, designs, and prototypes an architecture and interface, called WER, that provides a remote R server to allow organizations to support powerful R servers that can meet scientists’ needs and enable user collaboration. A systematic comparison was performed between WER and the most active Web-enabled R projects available: Rpad, Rweb and R-php, and determined that the WER project provides many advancements including improved scalability, collaboration functionality, and R environment persistence. WER provides a straightforward interface to third-party applications via Web services, such as workflow systems like Kepler [70], and represents a possible future direction for Web-enabled R projects.
Chapter 4

Annotating and Curating Knowledge via BioDEAL

4.1 Motivation

In the previous chapter the research primarily dealt with defining data analysis over the Web using the open source mature R statistical package. The proposed WER Web-enabled R system provides an environment that mimics the application based R environment over the Web; it supports a terminal type interface, access to R help pages, a scratch pad, etc. In addition to preserving the features that R users are familiar with, it has enriched R’s capabilities with collaborative features including sharing analysis scripts and results.

The next challenge for data intensive statistical computing is knowledge annotation and curation of the data analysis results. Due to the interdisciplinary nature of large-scale scientific efforts, there is an emerging trend of recording predictions from data analysis routines in community sharable databases. Certain communities, such as Biology, are particularly advanced in sharing their knowledge through public databases. Others are moving quickly in that direction through the creation of community data sets, such as the Earth Science community with the Program for Climate Model Diagnosis and Intercomparison (PCMDI) archive [82], for example. Others, such as Fusion and Combustion, are moving in that direction, but in a less systematic way at this point.

Knowledge curation and annotation by the community at large opens up the opportunity to improve predictions and generate new higher-level inferences to fill in the gaps
in the community’s understanding of complex physical phenomena. Our climate collabora-
tor at ORNL, John Drake, for example, claimed that “the annotation of historical events 
through simulation and observational data is an important part of the process of under-
standing climate events, their causes and consequences. Since expertise is partly a matter 
of knowing what to look for, the annotation is dependant on many eyes wired to the public 
brain. If a meteorologist annotates the Argo sea surface temperature data with El Nino 
(ENSO) information, then a plant biologist may realize the connection with increased plant 
productivity in the southeast US due to a typically wetter season. Annotation will focus 
first on anomalies in the carbon pools and seek to trace back to causes.”

Although different communities have distinct requirements, preliminary studies 
indicate that the design of a relatively generic infrastructure could be pursued with the 
possibility to customize and specialize that framework to the needs of individual communi-
ties.

4.2 Biomedical Community Driver

Over the past decade, systems Biology research has undergone two key trans-
formations. On one hand, public databases of experimentally generated -omics data are 
increasing in number, size and diversity, along with annotations predicted from these data 
by computational tools. Such annotations may include the predicted protein functions as 
part of genome annotation pipelines, the predicted high resolution 3-dimensional struc-
tures of proteins from amino acid sequence information alone, the predicted protein-protein 
interactions and interaction networks derived from databases of yeast-2-hybrid, or mass 
spectrometry pull-down experiments.

On the other hand, the number of on-line research articles, many of which are 
open access, is continually growing. There are currently over 20 million scientific abstracts 
in MEDLINE, growing at 500,000 articles per year [83]. Such articles often report the dis-
covered evidence (e.g., mutagenesis experiments) for various hypotheses derived via mining 
these heterogeneous databases of publicly available data and annotations. For example, 
GenBank [84] may report the predicted annotations for the two genes, designated omcB 
and omcC, to encode putative outer membrane polyheme c-type cytochromes, which are 
important for Fe(III) reduction by Geobacter sulfurreducens bacteria. Later, the open ac-
cess paper by Professor Lovley’s lab (PMID 12644478) may report experimental evidence
indicating that OmcB, but not OmcC, has a major role in electron transport to Fe(III) in this organism.

The interesting question is how to close the growing gap between these two paths of scientific discovery in biomedical sciences: the data-to-annotations-to-databases path and the annotations-to-hypotheses-to-evidence-to-publications path. Essentially, what is the proper infrastructure to enable streamlining published evidences, which rely on upstream database annotations, into the databases, thus establishing a feedback loop into the database-publication cycle? Without such an infrastructure, it is quite likely that highly valuable knowledge extracted by researchers, who browse the databases for valuable annotations and spend tedious efforts to support their findings with possibly published evidences, is recorded only in researchers’ personal notes and is not integrated into the database-publication cycle to assist other researchers.

This chapter presents the Biological Data-Evidence-Annotation Linkage system (BioDEAL) \cite{10, 11, 12} that introduces a feedback loop into the database-publication cycle to allow scientists to make connections between data-driven biological concepts and publications, and vice versa. The cycle is illustrated in Figure 4.1. By subscribing to the services provided by BioDEAL, an end-user can annotate the facts reported in literature, associate them with semantic concepts, link them to semantically annotated biomedical events or relationships, and share these literature annotations with other researchers in a social network. For example, while reading the paper by Lovley et. al (PMID 1264447), a genome annotation expert may decide to link the omcB (GSU2887) gene with electron carrier activity (GO:0009055) in the Gene Ontology (GO) \cite{85} and add a comment on experimental validation of its predicted function as the Fe(III)-reductase. Likewise, a reader interested in _Shewanella oneidensis_ may annotate the event reported by Thompson et. al (PMID 11823232) that the chromate shock in _Shewanella_ causes the repression of omcA and omcB genes and significant up-regulation of two-component signal transduction systems (SO,2426).

BioDEAL records these annotations in a structured (XML) format so that other databases, such as GenBank or UniProt, may parse this information and potentially update its “Related Articles in PubMed” field for this gene page with the PubMed ID. Typically, public databases reference publications on the original genome sequencing and annotation. Other references are not captured, such as those associated with the genes, pathways or subsystems-level and the events or relationships between them, for example. It is typically
Figure 4.1: The database, BioDEAL and publication feedback loop.

very tedious work for people studying a particular system/gene(s) to track this information through time-consuming literature reading. Although a number of databases and frameworks can benefit from and/or enhance the functionality of BioDEAL, to the best of our knowledge, BioDEAL is the first system that enables such a feedback loop into the database-publication cycle.

4.3 Major Contributions

Unlike existing annotation systems, the proposed and implemented Statistical Data Analysis Framework (SDAF) component, called BioDEAL, supports an environment for the annotation and curation of evidence from publications in standard and consummable formats. The major contributions of the research study underlying BioDEAL are the following:

- Proposed and designed an approach to standardize diverse annotations and publish them in a publicly consummable format.

- Proposed and designed a system to enable Web annotation of complex events.

- Proposed and designed a social networking extension that allows the annotating community to self-govern and share results.
The proposed advancements have enabled the BioDEAL SDAF component to introduce a feedback loop into the database-publication cycle to allow biologists to make connections between data-driven biological concepts and publications, and vice versa. Public biological databases, such as GenBank, can subscribe to the BioDEAL FTP output files to update their data with annotated published evidence. In contrast to Natural Language Processing (NLP) tools, BioDEAL allows users to annotate not only terms, but also events, which may be complex and contain other events.

The usability and applicability of the BioDEAL component is demonstrated for two distinct communities: NLP and Biology. The NLP related research is described in [10], while the Biology research has been published in [11, 12].

The primary focus of this chapter is the core Computer Science components of this framework and its customizations driven by the needs of the Biomedical community. The rest of this chapter is organized as follows: Section 4.4 describes the results from the perspective of the biologist end-users; Section 4.5 discusses the design; Section 4.6 identifies potential benefits of the framework; Section 4.7 presents related work; and Section 4.8 concludes.

4.4 Results

4.4.1 User Frontend Interface

BioDEAL supports a Web browser plugin interface that allows users to create and store document evidence while reading publications. The frontend contains the following core component panels: the Publication panel, the Database Reference panel, the Entity panel that contains the Term and Event/Relationship tables, and the Ontology panel, all shown in Figure 4.2. The Publication panel (top-right) contains the publication text, for example, from PubMed or MEDLINE; BioDEAL supports both PDF and text (HTML, PHP, etc.) documents. The former is typically a full publication identified by its URL on the journal Web site, while the latter may be an abstract from PubMed.

The BioDEAL Database Reference panel (top-left) contains database indexable fields at the gene level (e.g., TIGR or UniProt locus tag), protein domain level (Pfam or InterPro ID), or pathway level (KEGG or BioCyc ID). Such fields enable data exchange
between public databases. It also contains common fields at the organism taxa level (e.g., organism name) and gene attributes level (e.g., function description). Although those fields do not typically have associated ID numbers, they are frequently used. For a similar purpose, this panel also includes event/relationship fields such as primitive associations (e.g., Is-A, Part-Of) as well as common interest ones (e.g., Cause, Binding, Up-regulation, Down-regulation, Expression).

The BioDEAL Entity panel (bottom) supports two fundamental entity types: terms and events/relationships (Sections 4.4.2, 4.4.3). It contains a tabular list of terms and events annotated for a given publication. Each term annotation associates a word phrase in the publication with a semantic concept from the Ontology or Database Reference panels. For example, the word phrases omcB, GSU2887, and Geobacter sulfurreducens from the PMID:1264447 publication can be linked to the semantic concepts of Gene name, Locus tag and Organism name, respectively, from the Database Reference panel. GSU2887 can be associated via a HAS−A relationship with the ‘electron carrier activity (GO:0009055)’ GO node selected from the Ontology panel. Likewise, the ‘outer membrane’ phrase can be associated with the ‘Localization’ event concept in the GENIA ontology [86, 87]. The BioDEAL Ontology panel (bottom-left) includes multiple ontologies such as GO and GENIA and supports extensions of other ontologies of interest, discussed in Section 4.5.2.

4.4.2 Semantic Annotation of Terms

Semantic annotation of biomedical terms in a publication is an association of term semantic concepts (or keys) such as proteins, compounds, amino-acids, etc. with word phrases (or values) such as OmcB in the publication. The annotation process is quite simple; it involves highlighting a word phrase of interest in the Publication panel and double-clicking on the target semantic concept in the Database Reference panel or the Ontology panel. Similarly, a user may double-click on the target concept, which creates a new row in the Term table of the Entity panel, and then drag the target term into the Value column in the table.

Users can annotate terms in a document and link them with any number of semantic concepts. Each annotation action creates a row in the Term table with the key/value pair and the context in the publication in which the term appears, which is displayed in the Context column. While only a few neighboring words around the term are shown in this
column, mousing over the table cell displays the entire field. If a term contains multiple words that need to be jointly associated with the same semantic concept, then BioDEAL assumes that these words appear as a continuous span in the text, which is supported by the previous study [87, 88] concluding that 98% of terms appear in continuous spans.

Although terms denote semantically simple concepts, various ambiguities may arise depending on users and biomedical domains. To reduce such ambiguity, BioDEAL allows users to create key/value mappings with keys selected from the standardized semantic concepts that are created and agreed upon by user groups or domain experts; these normalized semantic concepts are called ontologies and are accessible through the Ontology panel (Section 4.5.2). However, to accommodate domain-dependent definitions, for which standardized ontologies may not exist, BioDEAL provides ways to extend the supported ontologies. Mapping terms (e.g., omcB) to standard semantic concepts (e.g., gene_name), allows users and systems to conduct a dialogue with the same agreed upon terminology and enrich the
types of searches that databases may choose to support. BioDEAL facilitates semantic associations that span multiple ontologies that may be concurrently used.

4.4.3 Semantic Annotation of Events and Relationships

BioDEAL also supports semantic annotation of biomedical events and relationships associated with semantically annotated biomedical terms. Examples may include the annotation of a Binding event through the linkage of a substrate term (e.g., malate) with a protein term (e.g., Mdh) or the annotation of a more complex Cause event through the linkage of the stress term (e.g., chromate shock) with the Up-regulation and Down-regulation events, which are, in-turn, linked with the proper gene-related terms (e.g., SO_2426 and omcB). BioDEAL supports semantic annotation of events expressed by their related entities. An entity may be a term or another event. Therefore, BioDEAL supports complex events by allowing events to contain other events.

Users can create a semantic annotation of an event by double-clicking on the semantic concept denoting the event of interest from the Ontology or Database Reference panels. This action creates a new row in the Event/Relationship table in the Entity panel (see Figure 4.2). He/she can then drag-and-drop into that row previously annotated term and event rows from the Term and/or Event/Relation tables, respectively. The relationships may be unary, binary or k-nary, all of which are supported by BioDEAL. In addition, if the event directionality matters, BioDEAL allows each entity associated with the event to be labeled as either on the Left (an input) or the Right (an output). If an entity is bidirectional, than it appears as both Left and Right. The event ontology contains an attribute that specifies whether the events are directional (see Section 4.5.2).

Unlike terms, events are often not situated near related events or terms, so there is no corresponding Context column in the Event table. Instead, the Keyword column serves a similar function for events by allowing users to highlight and drag keywords to the appropriate Keyword cell in the table. The Event table also contains an index column, the type of event (e.g., Up-regulation, Binding), an auto-generated value column constructed from the underlying terms, and a dynamic number of associated entities, which may be terms or other events.
4.4.4 Gene Cards and Search

Many public databases (e.g., TIGR, UniProt, KEGG) support site-specific representations of fundamental genomic information called gene cards. Gene cards provide users with a simple interface to retrieve basic gene information in one place. Many of these databases present common fields, such as Locus_tag and Gene_name, and database-specific fields, such as KEGG pathways, as well as cross-references between fields from different databases, such as TIGR, NCBI-GI and UniProt locus tags. Existing gene cards, however, rarely contain records of relevant publications, and generally only include the original genome sequencing and annotation publications but not future evidence associated with specific genes, pathways, subsystems, etc. They also do not contain biological events, such as expression, binding and regulation, which could provide users with a new level of information.

BioDEAL supports gene cards that organize all user annotated evidence regarding a gene by the source publications. To create gene cards, BioDEAL allows users to group entities from the entity tables related to the target gene of interest by selecting the checkboxes in the Gene Card column. The grouping must contain one Locus_tag row associated with the target gene. An example gene card derived from such a grouping is shown in Figure 4.3.

The underlying gene card representation is in XML/OWL (see Section 4.4.5) and can therefore be consumed by external database systems by linking to the fields from the Database Reference panel discussed in Section 4.4.1. All gene card information is present in the containing publication with the possible exception of the Locus_tag field that is user-editable in case it does not appear in a publication (see Section 4.5.4).

Users have access to the BioDEAL gene cards through the BioDEAL search page. BioDEAL supports a simple query engine, shown in Figure 4.4, that allows users to search for data using some basic fields: PubMedID, Locus_tag, Gene description, Gene_name, etc. Each row in the search results corresponds to a gene card indexed by Locus_tag, along with gene-related information and the publication source (e.g., PubMed ID and hyperlink) in which evidence recorded in gene cards is reported. The Locus_tag is a hyperlink to the BioDEAL gene card. Basic gene-related information presented to the user includes, if available, the organism name for that gene, the common gene name, and the gene function description.
Figure 4.3: A gene card for SO_2426 gene in *Shewanella oneidensis* MR-1.

Although the current BioDEAL implementation only supports gene cards, the system is extensible to other types and searches, such as Pathway cards or Gene Mutant cards.

### 4.4.5 Data Accessibility

The BioDEAL design supports storing information and recalling it in formats consumable by public databases, such as GenBank or UniProt. BioDEAL publishes OWL evidence files regularly on an FTP server that allows any system to retrieve public evidence. The primary keys in the evidence files are the BioDEAL Database Reference fields discussed
in Section 4.4, with the *Locus_tag* serving as the database cross-reference field. The success of the system as a feedback mechanism depends on whether existing systems incorporate BioDEAL into their life cycles.

During the course of annotating publication evidence, some data may not be attributed to a BioDEAL Database Reference field, and therefore is not included in the FTP file. However, this data is still important because users can search on these fields. For example, consider a compound *X* that is not associated with a single gene. If evidence exists in BioDEAL for *X*, then users can query for all publications that refer to *X*.

### 4.4.6 Social Networking

The BioDEAL server also contains a Web portal with several features, including a social networking component where users can login, publish, and edit annotations within their permissions’ sandbox. Collaborative features are becoming more crucial for scientists as researchers are increasingly geographically dispersed. Therefore, users can annotate as an individual or within communities. A scientist may publish annotations with different intentions: he/she may want to inform other scientists about a result, solicit opinions, or foster dialogue, to name a few. The environment supports collaborative functionality, shown in Figure 4.5, by displaying annotations to users within their permission groups.

Incorporating social networking features into BioDEAL is intended to foster dialogue in the research community, and support a checks-and-balances system to improve the integrity of ontological annotations by allowing research group members to review the annotations of other members and make suggestions. While ontologies have been enthusiastically received in Biology, biologists may lack the background required to use them appropriately, and integrating social networking may produce better overall annotations. Annotations are public by default, but BioDEAL allows research groups to configure privacy settings that prevent non-group members from accessing their annotations. This feature allows biologists to annotate evidence without exposing research-in-progress. In this scenario, the evidence is useful for the research group, but does not affect the data-publication cycle because the data cannot be consumed by public databases, unless the research group decides to make the evidence public at a later date.
4.5 Methods

4.5.1 Architecture

The BioDEAL framework consists of the following main components (Figure 4.5): an Annotation Server, the Annotation database, an OWL Ontology Interface, a Query and Retrieval Interface, a Social Networking block, and an Annotation Frontend. The Annotation Frontend, shown in Figure 4.2, is a Firefox plugin that supports two interfaces: one handles standard text and builds upon the W3C Annotea project and the other supports PDF documents.

The BioDEAL framework lets users expose any ontology by extending a Java class or implementing specific Web services. The OWL Ontology Interface sends available ontologies from the server to the Annotation Frontend in OWL format. Once an ontology is loaded the Annotation Frontend can query the OWL Ontology Interface for specific ontological categories via a search method. BioDEAL uses OWL for ontology communication because it is a W3C standard and will help enable others to develop new semantic tags and relationships as well as ease the development of new Annotation Frontends.

The Web services for incorporating ontologies and Natural Language Processing models, described in Section 4.5.3, into BioDEAL use REST technology, rather than the more common SOAP protocol, or XML-RPC. The REST protocol is straightforward and uses Uniform Resource Locator (URL) addresses to make calls. SOAP is a powerful protocol, but it introduces levels of complexity and redundancy that are not justified for the framework. XML-RPC is also more complex than REST.

The Annotation Server handles the communication between the Annotation Frontend and the backend BioDEAL database, which uses MySQL 5. Communication between clients and servers uses XML, and specifically either RDF or OWL, depending on the request.
4.5.2 Normalization of Semantic Concepts Through Ontologies

Ontologies are tagging formats that can support the semantic enrichment of data by embodying the abstract knowledge contained in the data, which can be used for data integration and analysis. Ontologies are used in BioDEAL to standardize the semantic concepts used for annotating entities, which allows data exchange between public databases and BioDEAL, and enables powerful user searches for evidence, which is displayed in gene cards.

BioDEAL allows users to incorporate any ontology into the system. Users may submit specific ontologies to the framework, such as the Systems Biology Ontology (SBO) and Gene Ontology (GO), both of which are expressed in OWL format. SBO and other ontologies can be found in The Open Biomedical Ontologies (OBO) Foundry [89]. BioDEAL backend clients receive OWL data by sending requests to the server, which directly queries the appropriate ontology and returns a subset of the data to the client. Using the server as an OWL gateway allows BioDEAL to manage the amount of data sent to the client.

The ontology data format will vary among different ontologies. Therefore, a universal ontology manager that uniformly communicates with all ontologies is not possible. Instead, an OWL interface is provided that must be implemented per ontology. While the communication uses OWL, the underlying ontologies may be in any format. The primary functionality of the interface is to return the immediate children of a particular node; each child contains the class ID, stored in the rdf:about attribute, and a text label.

Users can submit ontologies to the framework by publishing a Web service or writing a Java class that extends the Ontology abstract class. Both approaches require two methods: getRootNode and getOntologyNode. The former returns the root node of the ontology, and the latter returns any child node. The object returned is an OntologyNode, which contains information about the current node and its children. The Web service returns the OntologyNode in serialized XML. All ontology communication between the BioDEAL server and client uses OWL, although non-OWL compliant ontologies can be published in the framework by following the same approach. The OWL format is described below.

BioDEAL further enriches ontologies by classifying each annotated word phrase as a term or relationship; the default is a term. Entities can also be grouped with a
The OWL data returned by the methods and sent to the backend clients must adhere to the structure shown below. The first example shows the backend client OWL.

```xml
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#">
    <owl:Class rdf:about="http://purl.org/obo/owl/GO#GO_0000001">
        <rdfs:label xml:lang="en">mitochondrion inheritance1</rdfs:label>
        <rdfs:range>0</rdfs:range>
    </owl:Class>
    <owl:Class rdf:about="http://purl.org/obo/owl/GO#GO_0000002">
        <rdfs:label xml:lang="en">mitochondrion inheritance2</rdfs:label>
        <rdfs:range>0</rdfs:range>
    </owl:Class>
    <owl:Class rdf:about="http://purl.org/obo/owl/GO#GO_0000003">
        <rdfs:label xml:lang="en">mitochondrion inheritance3</rdfs:label>
        <rdfs:range>0</rdfs:range>
    </owl:Class>
    <owl:Class rdf:about="http://purl.org/obo/owl/GO#GO_0000004">
        <rdfs:label xml:lang="en">mitochondrion inheritance4</rdfs:label>
        <rdfs:range>3</rdfs:range>
        <owl:Class rdf:about="http://purl.org/obo/owl/GO#GO_0000004a">
            <rdfs:label xml:lang="en">mitochondrion inheritance4a</rdfs:label>
            <rdfs:range>0</rdfs:range>
        </owl:Class>
        <owl:Class rdf:about="http://purl.org/obo/owl/GO#GO_0000004b">
            <rdfs:label xml:lang="en">mitochondrion inheritance4b</rdfs:label>
            <rdfs:range>0</rdfs:range>
        </owl:Class>
        <owl:Class rdf:about="http://purl.org/obo/owl/GO#GO_0000004c">
            <rdfs:label xml:lang="en">mitochondrion inheritance4c</rdfs:label>
            <rdfs:range>0</rdfs:range>
        </owl:Class>
    </owl:Class>
</rdf:RDF>
```

The second example is the FTP output file for the annotations shown in Figure 4.2:
<?xml version="1.0"?>
<r:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#">
    <owl:DatatypeProperty rdf:ID="event" />
    <owl:DatatypeProperty rdf:ID="term" />
    <owl:DatatypeProperty rdf:ID="publication">
        <rdfs:domain rdf:resource="#term" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="locus_tag">
        <rdfs:domain rdf:resource="#term" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="gene_name">
        <rdfs:domain rdf:resource="#term" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="experiment">
        <rdfs:domain rdf:resource="#term" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="stress_conditions">
        <rdfs:domain rdf:resource="#experiment" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <publication rdf:ID="PUBMED#16788180">
        <rdfs:label xml:lang="en">16788180</rdfs:label>
    </publication>
    <locus_tag rdf:ID="TIGR_ID#SO_2426">
        <rdfs:label xml:lang="en">SO_2426</rdfs:label>
        <rdfs:subClassOf>
            <owl:Class rdf:about="E2#Positive_regulation" />
            <owl:Class rdf:about="PUBMED#16788180" />
        </rdfs:subClassOf>
    </locus_tag>
    <event rdf:ID="E3#Cause">
        <rdfs:subClassOf>
            <owl:Class rdf:about="TIGR_ID#SO_2426" />
        </rdfs:subClassOf>
    </event>
    <owl:Class rdf:ID="E3#Cause#Left">
        <rdfs:subClassOf>
            <owl:Class rdf:about="E3#Cause" />
        </rdfs:subClassOf>
    </owl:Class>
</r:RDF>
<owl:Class rdf:ID="E3#Cause#Right">
  <rdfs:subClassOf>
    <owl:Class rdf:about="E3#Cause"/>
  </rdfs:subClassOf>
</owl:Class>

<stress_conditions rdf:ID="Cr(VI)">
  <rdfs:subClassOf>
    <owl:Class rdf:about="E3#Cause#Left"/>
    <owl:Class rdf:about="TIGR_ID#SO_2426"/>
  </rdfs:subClassOf>
</stress_conditions>

<event rdf:ID="E1#Negative_regulation">
  <rdfs:subClassOf>
    <owl:Class rdf:about="E3#Cause#Right"/>
    <owl:Class rdf:about="TIGR_ID#SO_2426"/>
  </rdfs:subClassOf>
</event>

<event rdf:ID="E2#Positive_regulation">
  <rdfs:subClassOf>
    <owl:Class rdf:about="E3#Cause#Right"/>
    <owl:Class rdf:about="TIGR_ID#SO_2426"/>
  </rdfs:subClassOf>
</event>

<gene_name rdf:ID="omcA">
  <rdfs:subClassOf>
    <owl:Class rdf:about="E1#Negative_regulation"/>
  </rdfs:subClassOf>
</gene_name>

<gene_name rdf:ID="mtrA">
  <rdfs:subClassOf>
    <owl:Class rdf:about="E1#Negative_regulation"/>
  </rdfs:subClassOf>
</gene_name>

<gene_name rdf:ID="recX">
  <rdfs:subClassOf>
    <owl:Class rdf:about="E2#Positive_regulation"/>
  </rdfs:subClassOf>
</gene_name>
User Study

It is important to evaluate the usability of BioDEAL as a proof-of-principle research tool. BioDEAL is not mature enough for a mainstream audience, and access to biologists was not available, so an experiment was designed in which untrained users were directed to spend fifteen minutes a day during four consecutive days using BioDEAL to annotate any Web pages they visit. The Event/Relationship table was hidden for this experiment and users were asked to annotate the common NLP concepts of Person, Location and Organization using a simple ontology. Users were also instructed to annotate approximately 60 sentences extracted from the 2003 Conference on Natural Language Learning (CoNLL) [90] training corpus with these NLP concepts; the sentences were un-tagged prior to the experiment. We refer to the training corpus as the CoNLL corpus, and selected it for the evaluation due to its widespread adoption as a benchmark corpus. The first scenario is loosely analogous to biologists annotating publications that they would normally read while conducting research. The CoNLL annotations component is similar to asking biologists to annotate the GENIA corpus, i.e. previously annotated data, to allow comparisons between BioDEAL annotations and expertly curated data.

The results in Table 4.1 were generated by seven users over four days and created a corpus of 1,634 annotations; 1028 for general Web pages and 606 for the CoNLL subset.

Table 4.1: Total Web and CoNLL annotations per day

<table>
<thead>
<tr>
<th>Day</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoNLL</td>
<td>164</td>
<td>148</td>
<td>228</td>
<td>66</td>
<td>606</td>
</tr>
<tr>
<td>Web</td>
<td>378</td>
<td>238</td>
<td>201</td>
<td>211</td>
<td>1028</td>
</tr>
<tr>
<td>Total</td>
<td>542</td>
<td>386</td>
<td>429</td>
<td>277</td>
<td>1634</td>
</tr>
</tbody>
</table>

To test users’ annotation accuracy their annotations were directly compared to the expertly created standard CoNLL 2003 corpus. Table 4.2 shows that users annotated every Person entity tagged by CoNLL, giving a recall of 1. User-level annotation of the Location entity also achieved a high recall of 0.94, but the Organization entity yielded a lower recall of 0.82. The average recall over the three entities is 0.92, which is an improvement over the average recall of 0.81 provided by the sixteen automated predictive tools available through CoNLL.

User-level annotations demonstrated the following precision: 0.79, 0.95, and 0.69
Table 4.2: Recall and Precision for CoNLL annotations

<table>
<thead>
<tr>
<th></th>
<th>Per</th>
<th>Loc</th>
<th>Org</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall (All Data)</td>
<td>1.00</td>
<td>0.94</td>
<td>0.82</td>
<td>0.92</td>
</tr>
<tr>
<td>Precision (All Data)</td>
<td>0.70</td>
<td>0.82</td>
<td>0.42</td>
<td>0.58</td>
</tr>
<tr>
<td>Precision (Majority Voting)</td>
<td>0.76</td>
<td>0.86</td>
<td>0.48</td>
<td>0.64</td>
</tr>
<tr>
<td>Precision (Coverage Req.)</td>
<td>0.73</td>
<td>0.90</td>
<td>0.55</td>
<td>0.69</td>
</tr>
<tr>
<td>Precision (Majority Voting + Coverage Req.)</td>
<td>0.79</td>
<td>0.95</td>
<td>0.69</td>
<td>0.79</td>
</tr>
</tbody>
</table>

for Person, Location, and Organization entities, respectively, with an average of 0.79. These results were calculated using majority voting after removing annotations with singular coverage (called Coverage Req. in the table). Based on users’ feedback, annotating the Organization entity was the most unclear of the three. The average precision for the Person and Location entities was 0.87. Again, the casual user-level precision was comparable to the automated tools that attained an average precision of 0.82 over the three entities.

4.5.3 Semi-Automation via NLP Tools

It is impossible for biologists to manually process all available data. Natural Language Processing (NLP) tools can provide semi-automatic semantic annotation of evidence such as human gene names by GIANT or protein-protein interactions by PIE (Protein Interaction Extraction) (see [91] for a survey of tools). Such annotations are essential for filtering data and providing the opportunity to inject supplemental tags into the data to support more robust and semantically meaningful analysis. Scientists can therefore focus their efforts on data that requires their attention, either because the model is unable to process it with a high degree of certainty, or to verify the integrity of the model and provide feedback for model calibration.

BioDEAL provides a Web browser tab to an NLP interface where users may execute NLP tools on publications to populate the BioDEAL database with automatically annotated evidence that is presented to the user through the User Frontend Interface (Figure 4.2). Model authors can submit NLP models to the framework in two ways: by creating
Web services or extending an abstract Java class called *Model*. In both cases users must implement the `parseData(Object data)` method, which outputs an array of prediction objects that are inserted into the BioDEAL database, along with meta information regarding the tools and execution environment used. Semantic annotations from NLP tools are highlighted in green to indicate to the users that they are NLP-based predictions.

However, while NLP tools can provide high-throughput and be effective at simple term identification, existing tools generally lack the maturity necessary to adequately process complex entity relationships because creating a semantically rich document representation is a hard problem. There is a lack of naming conventions, frequent abbreviations, common use of synonyms and homonyms, and terms often contain multiple words, such as: “human T-cell leukemia lymphotropci virus type 1 Tax protein” [92]. 53% of the corpus names in BioCreAtIvE, a community challenge to evaluate biological text mining, have more than one token [93]. Similarly, the acronym *ACE* has multiple meanings, including *angiotensin converting enzyme*, *affinity capillary electrophoresis*, *acetylcholinesterase*, although research has yielded correct acronym meaning detection with over 95% accuracy [94].

BioDEAL provides a mechanism to annotate these complexities, and the NLP tools may augment BioDEAL by automatically identifying simple terms in a document.

### 4.5.4 Quality Assurance and Control

Annotation quality and quality control mechanisms are critical and non-trivial issues for a framework such as BioDEAL. Unlike manually annotated corpora by domain experts, annotations by Web users will likely be noisy and inter-annotator discrepancies should be expected [86]. About 70–80% of annotations have inter-annotator agreement even if the annotators are instructed with well-formulated guidelines [95]. Although BioDEAL does not automatically address such discrepancies, it can show all available annotations for a given term and let users decide how to resolve inconsistencies, either through re-annotation or augmenting the proper comments with the displayed annotations.

BioDEAL enabled annotated and curated Web corpora can be utilized for manual curation by developers of public databases or NLP tools. It would be desirable for a framework, such as BioDEAL, to provide analytical intelligence to make decisions about collating and resolving possibly conflicting and uncertain annotations from potentially many users and/or various NLP tools. This is an open area of research, and deserves an active
Another source of potential errors comes from allowing the values for *Locus_tags* to be manually edited. Ideally, the terms should be semantically annotated only if they are present in the document. However, certain terms can be implicitly inferred from other terms, and are often not mentioned in the publication. For example, given a gene name (e.g., omcB), an organism name (*Geobacter sulfurreducens*) may be sufficient to infer the implied TIGR locus tag (e.g., GSU_2737), which is often omitted from the publication. Therefore, BioDEAL allows users to manually enter a value for the *Locus_tag* terms to facilitate the desirable streamlining of evidence recorded in publications directly to public databases that require these unique identifiers, although it inevitably raises data integrity concerns.

Public databases may use different *Locus_tags* for the same gene, and therefore the gene card specifies what type of *Locus_tag* is referenced (e.g., TIGR, UniProt). At this time, BioDEAL does not cross-reference *Locus_tags* between databases; instead, users must specify the appropriate database reference.

Finally, determining publication IDs (e.g., PubMed ID) is potentially an error-prone process. It can be difficult because the BioDEAL plugin only has access to the source URL when a user opens a publication, which often does not include the publication ID. Instead, a publication ID lookup prototype was created that queries publication databases (currently only PubMed) with word phrases from a publication. The initial findings are very encouraging: with only a few phrases of more than a few words the prototype can identify the source publication with high certainty. As the number of annotated word phrases increases, so does the certainty.

### 4.6 Potential Benefits

The actual impact of Web user annotated corpora creation remains to be seen, since BioDEAL is still a research tool, however, the potential benefits of such a framework are manifold. For example, public databases (e.g., TIGR, GenBank, BioDAS servers) can link to supporting evidence with predicted annotations. This may ultimately improve annotations through resolving inconsistencies and confirming their validity from published evidence.

The public databases can leverage BioDEAL annotation-evidence data using the FTP output files either directly or through an interface such as BioDAS [96], a communication protocol to exchange biological data annotations. Likewise, BioDEAL can present
annotations generated by external projects such as BioCreAtIvE [91, 97], whose overarching
goal is to enhance abstracts with annotations. BioCreAtIvE users can gain from BioDEAL
by querying the system to determine what has been annotated. BioCreAtIvE is limited to
NLP tasks, which does not include many common tasks, such as identifying up and down
regulation, therefore BioDEAL can augment BioCreAtIvE results.

Developers of NLP tools can generate annotated corpora beyond those currently
supported (e.g., protein/gene names, functional and interaction annotations). NLP de-
signers often lack the corpora necessary to develop their algorithms and BioDEAL builds
corpora during its normal lifecycle, which NLP designers can use to reduce the difficulty
of corpora generation, while improving the quality because they are manually curated by
domain experts. It is not expected that users will annotate every entity in a publication,
however, it is likely that the key facts and evidence will be identified. Therefore, BioDEAL
may be more suitable for the enrichment of public databases with evidence information, al-
though the NLP community can likely benefit from this expert knowledge recorded through
structured annotations.

Experts in different fields of biomedicine may exchange their annotations, com-
ments, and open issues, thus improving the quality of genome annotations through col-
laborative knowledge creation. BioDEAL can help identify ambiguous annotations and
facilitate community discussion and consensus on annotations, perhaps using an alert sys-
tem when conflicting annotations are found. BioDEAL can also enable better classification
of biological information and structured representation, while facilitating better search and
information retrieval functionality beyond the standard available fields (e.g., author, affil-
iation, etc.).

4.7 Related Work

Within the Biology community, the various annotation efforts can loosely be or-
ganized into the following categories: Biological Text Mining (BTM), including both Term
and Event Identification, Biological Genetic Annotations (BGA), Biological Annotation
Systems (BAS), Biological Ontologies and Tools (BOT), and Biological Community Tools
(BCT). Some tools fall into more than one of the above proposed categories. BioDEAL is
part of each of the above categories, except BGA; the categories are described below with
specific project examples.
4.7.1 Biological Text Mining (BTM)

Text Mining (TM) has been recognized as one of the key technologies for future bio-medical research [86]. However, NLP approaches are generally domain specific due to different goals, vocabulary, semantics, etc. Therefore, NLP tools must be constructed expressly for each domain. Within the past few years several biological entity lexicons have been released, including BioThesaurus [98] and The Unified Medical Language System (UMLS) [99] to assist and enable NLP and BTM.

Within BTM there is Term and Event identification. The former involves identifying spans of text that refer to biological entities such as gene names and proteins. This is a non-trivial problem in general, and is often more difficult in specific domains, such as Biology, due to complex vocabularies, opaque synonyms and author-specific verbiage. Despite these obstacles, BTM has seen steady performance improvements by leveraging NLP techniques, open access vocabularies and domain specific annotated corpora, and increased community interest and support.

There are numerous BTM projects, including the ones presented below. ARROWSMITH [100] is a set of interactive software and database search strategies that facilitate the discovery of possible relationships between biological entities. An ARROWSMITH user begins with an experimental finding or hypothesis that two items A and C are related in some way. The titles of papers indexed in MEDLINE which contain the word ‘A’ (or synonyms) are downloaded into a file A, and similarly a file C is created. The software constructs a list of words and phrases B common to files A and C; automatic and manual editing are used to filter out uninteresting B-terms. For each B-term, the software generates an AB file of titles containing both ‘A’ and ‘B’, and a BC file of titles containing both ‘B’ and ‘C’; these titles are juxtaposed to facilitate the user judging whether there is likely to be a biologically significant relation among A, B and C.

XplorMed [101] maps PubMed search results to the eight main Medical Subject Headings (MeSH) categories and then extracts topic keywords and their co-occurrences. XplorMed then suggests main groups of related topics and documents, sparing the user the need of reading all abstracts. Textpresso [102] is an ontology-based information retrieval and extraction system for biological literature that infers semantic relationships in articles by cross-referencing ontological terms.

PubFinder [103] is a publicly available web tool designed to improve the retrieval
rate of scientific abstracts relevant for a specific scientific topic. Users submit a small set of articles that PubFinder uses to search PubMed abstracts for related publications. MedMiner [104] searches and filters literature based on user-specified genes or concepts of interest and presents the most relevant results to the user. Chilibot [105] uses an NLP-based text-mining approach to construct relationship networks among biological concepts, genes, proteins, or drugs.

PreBIND and Textomy [106] use a Support Vector Machine (SVM) approach to locate interaction information in the literature and present it to curators or public users for review and submission to the BIND database. PubGene [107] uses automated extraction of explicit and implicit biomedical knowledge from publicly available gene and text databases to create a gene-to-gene co-citation network for 13,712 named human genes by automated analysis of titles and abstract and is in over 10 million Medline records. The method is based on the assumption that if two genes are co-mentioned in a Medline record there is an underlying biological relationship. GENIES [108] is a natural-language processing (NLP) system for the extraction of molecular pathways from journal articles.

BioBibliometrics [109] is an information retrieval and visualization system that searches for co-occurrences of gene names in Medline abstracts. It is posited on the idea that if two genes have a related biological function, the co-occurrence of two gene names (or aliases of those genes) within the biomedical literature is more likely, similar to the PubGene hypothesis mentioned above. MedMOLE [110] attempts to improve the comprehension of microarray experimental results by identifying the main functions among a group of genes, as reported in Medline articles. It also strives to assist in identifying function/disease-specific genes, thereby simplifying the design of specially-devoted microarray.

The PubCrawler [111] WWW service is an automated update alerting service for users of NCBI’s PubMed (literature) and GenBank (DNA sequence) databases. PubCrawler executes personalized searches at NCBI at regular intervals (e.g., daily), keeps track of what records have been seen previously, and produces a WWW page listing the latest hits that match the user’s interests. PubCrawler helps keep scientists informed of the current contents of Medline and GenBank by listing new database entries that match their research interests. EDGAR [112] is a pilot project that mines information regarding macromolecular binding relationship from the Molecular Biology literature. It takes advantage of NLP techniques and is supported by two repositories of biomolecular knowledge. TEXTQUEST [113] performs large-scale document clustering of biological text obtained from Medline abstracts.
The algorithm is based on statistical treatment of terms, stemming, the idea of a ‘go-list’, unsupervised machine learning and graph layout optimization.

Event identification, however, is an even harder task due to increased complexity. Events have their own internal structure and may involve terms or other events, therefore allowing for recursive definitions of events. Events have one or more components with relationships between the components (e.g. binds to, up-regulated by, is-a), and event descriptions may occur in discontinuous locations in the text. Propbank [114] and FrameNet [115] are two domain agnostic projects that use general linguistic techniques to address these issues. Within the Biology domain, some projects have started to address these issues. The GENIA [86] ontology and corpora have been augmented to support future event annotation progress, and the same research group released the XConc Suite, built upon the common Eclipse software platform [116], to enable users to annotate events in biological literature within a desktop application.

NLP tools need to be trained against, and compared to, standard corpora and there are several existing biological corpora that meet this requirement, including GENETAG [117], a tagged corpus of twenty thousand Medline sentences for gene/protein Named-Entity Recognition (NER), and GENIA [87], a general Biology corpus with nearly one hundred thousand biological terms annotated by two domain experts. The growing interest and supporting tools in the Biology community attest to the developing trend towards event identification in BTM.

4.7.2 Biological Genetic Annotating (BGA)

The BTM effort is distinct from biological genetic annotation systems aimed at annotating genetic sequences. Some genetic tools include the following: Apollo [118] is an interactive tool to allow biological experts to annotating genome sequences by viewing and independently evaluating the data supporting each annotation. Apollo enables curators to inspect genome annotations closely and edit them. DAVID [119] is a web-accessible program that integrates functional genomic annotations with intuitive graphical summaries. Lists of gene or protein identifiers are annotated and summarized according to shared categorical data using GO, protein domains, and biochemical pathway membership. GATO [120] is an automatic annotation system for preliminary analysis of DNA sequences.

Genepi [121] is a framework for developing automatic genetic annotation systems.
The system is not bound to any specific annotation strategy. Instead, the user will specify a blackboard structure in a configuration file and the system will instantiate and run this particular annotation strategy. LeARN [122] is a software package that integrates automatic detection with human curation to annotate RNA sequences. ProFAT [123] is a web-based tool for annotating protein sequences. Ensembl [124] is a database project that provides a bioinformatics framework to organize Biology around the sequences of large genomes. It contains confirmed gene predictions that have been integrated with external data sources and is available as either an interactive web site or as flat files.

Numerous other tools exist under the genetic annotation umbrella, including ENSMART [125], MatchMiner [126], Resourcerer [127], GIANT [91], AKANE++ [91], PIE [91], iHOP [91], and many others.

4.7.3 Biological Annotation Systems (BAS)

BAS systems are aimed at annotating free text in Biology publications using automated, manual, or hybrid approaches. BioLit [128] is a web server which provides metadata describing the semantic content of open access, peer-reviewed articles that describe research from the major life sciences literature archive, PubMed Central. Specifically, these metadata include database identifiers and ontology terms found within the full text of the article. BioLit is the most similar of all the existing projects to BioDEAL and is also a new effort, released in 2008. The recent timing of BioDEAL and BioLit suggest that both the technology and Biology community are prepared to address the next step in managing biological evidence.

Both projects consider annotations in any text in the article, as opposed to other tools that limit annotations to the abstract section. Also, both projects provide a web interface and publish the annotations via publicly accessible XML files. While both projects support term annotations, only BioDEAL supports the more complicated event annotations. Additionally, both systems link to external biological databases, such as Uniprot, although BioLit only allows users to annotate papers that it has pre-processed from PubMed Central, currently about 52,000 out of the millions of PubMed articles. In contrast, BioDEAL users can annotate any publication or web page, without restriction. However, a principle difference remains: BioLit does not allow users to create annotations and relies solely on automated BTM, unlike BioDEAL, which allows users to execute BTM tools on the
document text, but incorporates user generated annotations.

The Critical Assessment of Information Extraction systems in Biology [129], called BioCreAtIvE, community challenge is an effort to evaluate text mining and information extraction (IE) systems within the biological domain. BioCreAtIvE annotations consist of identifying gene or protein names, labeling the content with gene/protein and taxonomic IDs, and a confidence score if the text contains protein-protein interaction details. BioCreAtIvE merges DNA, RNA and protein names into the same ontological category, therefore integrating all domains, complexes, sequences, etc. because even human annotators only agree 77% of the time [117].

Two particularly relevant components of BioCreAtIvE are the electronically annotated information (EAI) and the BioCreAtIvE MetaServer (BCMS). BCMS is a service that promotes IE in Molecular Biology to automatically generate annotations for PubMed and MEDLINE abstracts. EAI is a proposed metadata format that contains expert biological annotations created by the document author(s), and is therefore likely to have even higher accuracy and reliability than community annotated data. The EAI data would be available through standard publication web sites, such as PubMed and MEDLINE, and would identify significant concepts within publications. Programmatic algorithms can leverage these metadata concepts to perform more focused analyses and help biologists locate more pertinent sources and collaborators.

EAI and BioDEAL are complementary efforts: the former is performed by the document authors during pre-publication, whereas BioDEAL annotations are created post publication by the domain community. We are not aware of any publications that require or accept EAI metadata, and existing publications do not contain EAI, therefore there are millions of research documents that can be annotated with BioDEAL and do not contain EAI data. If EAI is available in the future, it will improve semantic knowledge computations and can work in conjunction with the BioDEAL community approach. Members of the research community may have different opinions regarding the most appropriate annotations; in the EAI approach, only the author’s perspective is considered, whereas BioDEAL can reflect the overall community’s position.

The Biomolecular Interaction Network Database (BIND) [130] archives biomolecular interaction, complex and pathway information. A web-based system is available to query, view and submit records. The Conceptual Open Hypermedia Service (COHSE) [131] is an ontological reasoning service which is used to represent a sophisticated conceptual
model of document terms and their relationships. It is integrated to form a conceptual hypermedia system to enable documents to be linked via metadata describing their contents to improve the consistency and breadth of linking of WWW documents at retrieval time (as readers browse the documents) and authoring time (as authors create the documents).

4.7.4 Biological Ontologies and Tools (BOT)

Ontologies enable systems to introduce structure and standardization within annotations and the Biology community has actively pursued these approaches to organize their data. Several examples are described below.

SEGOPubmed [132] uses the concept of latent semantic analysis (LSA) to link PubMed abstracts to GO. The basic idea behind LSA is to map both the documents and the query vector into semantic space before comparison. GOPubmed [133] is a web server which allows users to explore PubMed search results with the Gene Ontology (GO), a hierarchically structured vocabulary for molecular biology. GoPubMed performs IE on published abstracts by providing an external search page that organizes PubMed results using the GO ontology to present the results in a standardized structure. The GoPubMed authors state that the service can help uncover otherwise difficult to determine associations between biological entities and can also provide author profiles by searching for an author and seeing how their papers are organized within the GO ontology.

There are many differences between GoPubMed and BioDEAL: GoPubMed provides a valuable service that allows scientists to conduct a more focused search for research articles, but it does not strive to build, collect and publish biological evidence for future use, as BioDEAL does. GoPubMed only considers the abstract of each article and requires users to navigate to GoPubMed.org to leverage the service; BioDEAL allows users to process the entire article and the Web browser plugin enables users to read the articles in a native browser interface.

The Gene Ontology Knowledge Discovery System (GO-KDS) [134] is a publicly available web application that uses machine learning techniques to automatically connect biomedical documents to GO terms. The National Center for Biomedical Ontology (NCBO) [135] seeks to provide tools and methods to enhance the use of ontologies throughout biomedicine and is developing new ontology: the Open Biomedical ontology (OBO). The OBO Foundry is a collaborative experiment involving developers of science-based on-
tologies who are establishing a set of principles for ontology development with the goal of creating a suite of orthogonal interoperable reference ontologies in the biomedical domain. The groups developing ontologies who have expressed an interest in this goal are listed below, followed by other relevant efforts in this domain.

The goal of the Systems Biology Ontology (SBO) is to develop controlled vocabularies and ontologies tailored specifically for the kinds of problems being faced in Systems Biology, especially in the context of computational modeling. The goal of the Gene Ontology (GO) Consortium is to produce a dynamic, controlled vocabulary that can be applied to all eukaryotes even as knowledge of gene and protein roles in cells is accumulating and changing. To this end, three independent ontologies accessible on the World-Wide Web are being constructed: biological process, molecular function and cellular component. The Medical Subject Headings (MeSH) was first introduced in 1960 and was a pioneering effort as a controlled vocabulary that was applied to early library computerization. It has been updated over the years and was recently thoroughly revised. GENIA is a simplified general Biology ontology and corpus created to provide a more general and shallow ontology compared to existing ontologies, such as GO.

4.7.5 Biological Community Tools (BCT)

Several tools currently exist that enable biologists to collectively perform BGA annotations within communities. Three of the most prominent are mentioned below. ORegAnno [136] is an open-source, open-access database and literature curation system for community based annotation of experimentally identified DNA regulatory regions, transcription factor binding sites and regulatory variants. Apollo2Go [137] is a web service adapter for the Apollo genome viewer to enable distributed genome annotation. The Distributed Annotation System (DAS) [138] allows sequence annotations to be decentralized among multiple third-party annotators and integrated on an as-needed basis by client-side software. The communication between client and servers in DAS is defined by the DAS XML specification. Annotations are displayed in layers, one per server. Any client or server adhering to the DAS XML specification can participate in the system.
4.8 Conclusion

This chapter presented the BioDEAL biological evidence and curation system that introduces a feedback loop into the database-publication cycle by allowing scientists to link experimental data-driven biological concepts to published evidence. BioDEAL supports a Web browser frontend that allows biologists to semantically annotate evidence from publications within a native browser setting. Evidence can be simple terms or complex events, and users can run NLP tools to facilitate semi-automation. External databases can use the reference fields (e.g., TIGR ID, Pfam ID) in the FTP output files to map evidence to their internal data. The social networking component allows the Biology community to annotate and curate evidence en masse, and provides a mechanism for research groups to verify the integrity of annotated evidence while providing a secure environment to conduct research.

BioDEAL may introduce a transformative shift in the way biologists relate published evidence with experimental data. Instead of biologists or research groups searching and managing evidence independently, the community can collectively build and share this knowledge.
Chapter 5

Conclusion

This research thesis proposes the Statistical Data Analysis Framework (SDAF), with the goal to enrich and optimize the knowledge discovery cycle. SDAF contains the following three components:

1. \textit{pR}, a light-weight, easy-to-use plug-in that bridges compiled serial and parallel analysis routines into the \textit{R} scripting environment for efficient execution of data analysis tasks;

2. \textit{WER}, a Web-enabled statistical \textit{R}-based environment for defining data analysis pipelines in a remote, interactive and collaborative manner over the Web;

3. \textit{BioDEAL}, a semantically-rich annotation component for semantic annotation of published facts and complex events and their linkage as evidence for analytical predictions stored in distributed public databases.

Chapter 2 presents \textit{pR} and describes three approaches to improve end-to-end execution performance of parallel statistical scripting codes. Section 2.3 proposes a bridge for existing analysis techniques to communicate within a single environment, and pass information between these routines by using \textit{R} to hand data back and forth, thereby greatly increasing the number and quality of tools available to \textit{R} users. Section 2.4 extends this bridge to existing third-party parallel analysis techniques. \textit{pR}’s performance closely mirrors the performance of the third-party codes that it calls, while introducing limited overhead. Specifically, \textit{pR} achieved an average performance improvement of 37 times compared to \textit{Rmpi}, and yielded superlinear speedups compared to serial \textit{R} execution times while limited
incurring overhead of a few microseconds in our tests. Section 2.5 presents a further extension to the \textit{pR} package that introduces automatic parallelization of the \texttt{R lapply} method for data-parallel statistical methods in hybrid multi-core and multi-node environments. \textit{pR} achieves improved overall end-to-end performance by utilizing a two-tier execution strategy and the performance results in Section 2.4.4 demonstrate that \textit{pR} can outperform the commonly used \texttt{snow} package for \texttt{R} in hybrid multi-core and multi-node environments. Specifically, \textit{pR} achieved an average performance improvement of over 50\% compared to \texttt{snow} in our tests.

Chapter 3 proposes, designs, and prototypes an architecture and interface, called WER, that provides a remote \texttt{R} server to help meet scientists’ HPC needs and enable user collaboration. A systematic comparison was performed between WER and the most active Web-enabled \texttt{R} projects available: \texttt{Rpad}, \texttt{Rweb} and \texttt{R-php}, and determined that the WER project provides many advancements including improved scalability, collaboration functionality, and \texttt{R} environment persistence. WER provides a straightforward interface to third-party applications via Web services, such as workflow systems like Kepler \cite{70}, and represents a possible future direction for Web-enabled \texttt{R} projects.

Chapter 4 presents the BioDEAL biological evidence and curation system that introduces a feedback loop into the database-publication cycle by allowing scientists to link experimental data-driven biological concepts to published evidence. BioDEAL supports a Web browser frontend that allows biologists to semantically annotate evidence from publications within a native browser setting. The social networking component allows the Biology community to annotate and curate evidence en masse, and provides a mechanism for research groups to verify the integrity of annotated evidence while providing a secure environment to conduct research.
Bibliography


