Parallel R Computing on the Web

Ranjini Subramanian
University of Louisville
Louisville, KY, USA
r0subr05@louisville.edu

Hui Zhang
University of Louisville
Louisville, KY, USA
h0zhan22@louisville.edu

***Abstract*—R is the preferred language for Data analytics due to its open source development and high extensibility. Exponential growth in data has caused longer processing times leading to the rise in parallel computing technologies for analysis. Using R together with high performance computing resources is a cumbersome task. This paper proposes a framework that provides users with access to high-performance computing resources and simplifies the configuration, programming, uploading data and job scheduling through a web user interface. In addition to that, it provides two modes of parallelization of data-intensive computing tasks, catering to a wide range of users. The case studies emphasize the utility and efficiency of the framework. The framework provides better performance, ease of use and high scalability.**

***Index Terms*—Parallel computing, Divide-and-conquer methodologies, Framework, R, Web UI**

I. Introduction

R is an open-source programming language used mainly for statistical analysis and widely used in data science due to its ideal platform to conduct data analysis. R provides several features that are common in other programming languages such as loops, branching, conditional logic etc. R is easy to learn, open source language and can be run across a variety of operating systems. It is a "high productivity" language but it lacks the structures and control to support efficient code for large-scale computation. Rapid growth in data and methodological advances such as the use of simulation and re-sampling methods for analysis have led to challenges in processing times. It is a known fact that execution time can be prohibitively long as the size of data and computational complexity increases [20].

R framework is designed to use single thread execution mode. It is not designed to take advantage of modern infrastructure which includes computing clusters and parallel processors. In order to scale up computational capability, it is essential to utilize parallelism and high-performance computing resources.

A. Parallelism using R

In many areas of research, data is growing much faster than performance increases in hardware. To harness the full benefit offered by computing platforms, we must develop software with computing capabilities [1]. Although R is highly extensible through the use of packages, providing parallel packages or software for HPC was not a primary goal [2]. The sheer volume of data poses a challenge in terms of methodologies needed to reduce processing times. Some popular methods employed in statistical analysis ae bootstrapping, Monte Carlo simulation, Gibbs sampling etc. Rapid growth in data and the demand for simulation methods have been approached with the use of parallel computing. In the last decade more and more research has been focused on developing parallel packages for R in compute clusters, grid computing and multi-core systems. Popular parallel R packages include snow [14], multicore [15], parallel [16], R+Haddop [17], Rmpi [18], foreach [19] etc. The paper [2] reviews the various parallel packages comparing them on their usability, performance and parallel technology used. Usage of parallel packages requires extensive knowledge of parallelism offered by these packages.

B. Enabling parallelism using HPC

Propelled by the ever-increasing computation demand, significant efforts have been made in developing parallel computing technologies. The drivers for focus on HPC have been larger datasets and the computation power required for sophisticated methodologies. Parallel computation can be performed using multiple cores within a single node or by using multiple nodes. Although using a single node is less expensive and easier, it may not be sufficient as the number of parallel processes required for analysis increases. This leads to the utilization of multiple nodes for effective processing. Many libraries exist to enable easier access to parallel computing technologies to support data processing problems that can be regarded as Single Program Multiple Data (SPMD) model [5]. The commonly used libraries include MPI [3] and OpenMP [4]. They use programming methods to manipulate massive resources. A new programming paradigm, developed by Google, called MapReduce has become a standard for big data processing. It enables automatic parallelization and distribution thereby providing a powerful interface for large-scale computations. This model uses function-level implementation which requires advanced programming skills and good understanding of the underlying system architecture. It also requires fixed computational steps. It works by splitting the data and processing it in parallel while hiding the details of load balancing, synchronization and fault tolerance. It takes a set if input key/value pairs and produces a set of output key/value pairs. It comprises of two functions - map and reduce. Both functions are written by the user. The map function takes input key/value pairs and produces a set of

intermediate key/value pairs that are then fed to the reducer function. The reducer function accepts the key along with a list of values for each key. It then merges the values and produces output key/value pairs. Each input file contains a set of records and each record is treated as a key/value pair [21]. The input data is then partitioned and processed by the map function. MapReduce model is used for a wide range of applications including machine learning problems, graph computations, clustering problems etc. HPCs are used to empower R languages due to the following reasons:

- To address the computational complexity of big data processing
- To overcome the design shortcomings of R
- To utilize modern computing infrastructure
- To reduce processing time and improve efficient utilization of resources

C. Challenges

Despite the availability of parallel programming support in R for state-of-the-art HPC, there exists access barriers in bringing these resources to a wide range of users. Two major challenges are:

- Accessibility to HPC - State-of-the-art high-performance computing resources are often expensive and learning how to effectively use the tools provided by HPC takes a significant amount of time and training.
- Knowledge required to exploit parallelism - Although R provides a number of packages to enable parallel programming, it poses a significant challenge in terms of its usability. Often domain users lack the training needed to choose the most suitable package. This is important towards performance gain, suitability towards specific datasets and computing resource requirements.
  A common usage of parallel packages is to rewrite sequential function implementation with its corresponding parallel version. This requires extensive knowledge in both existing R code as well as parallel R programming. Users who are new to R programming are able to perform sequential code implementation and are not yet to be trained to take advantage of parallel packages.

D. Goals

To address the aforementioned issues, we propose a framework with the following goals:

- Web UI to access HPC, upload data, execute scripts and download results
- Parallel job array mode to enable automatic parallelization of sequential code implementation
- Explicit parallel R mode to execute parallel code implementation

The framework provides the users the advantages of simplicity, customization and scalability.

II. Proposed Framework

The framework is implemented using R programming language. The overall architecture of the framework is as depicted in Fig. 1. The framework consists of three loosely-coupled layers - web UI, parallel scripts generator (PSG) and the underlying high-performance computing resources. The PSG layer is responsible for implementing both modes of parallelization. Depending on the input from the user via the UI, the appropriate mode is chosen for execution. Once the mode is invoked, PSG automatically creates batch jobs to execute user scripts on users' input data. All jobs are executed on the underlying HPC using SLURM resource manager.

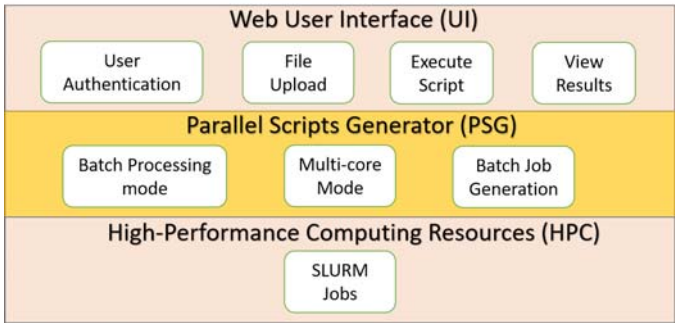


Fig. 1: Overall Architecture.

A. Divide-and-Conquer (DC) paradigm

DC paradigm is a popular approach in parallel computing owing to its versatility and computational benefits. It works by recursively breaking down a problem into subproblems of similar type and solving them in parallel. The results of the subproblems are then combined to produce the solution to the original problem. The independent nature of the subproblems make them best suited for parallel processing. DC paradigm is used to solve vast set of problems like matrix eigen value problem [22], simulating quantum mechanical systems on quantum computers using Hamiltonian structure [23], sorting and matrix multiplication [24].

DC paradigm works in two steps - Divide and Conquer. It can be implemented using sequential as well as parallel approach. In [25], [26], the different DC implementation strategies are presented, and their performance is analyzed. The study found that using the parallel mode reduced the execution time by over 98% compared to the serial mode. The processing time decreased further in parallel mode with the increase in the size of the dataset whereas in the serial mode, the execution failed due to insufficient system memory.

We explore two types of DC implementation in our framework for each of the modes. One, uses script-level parallelism using file-based processing and the other uses function-level parallelism using in-memory processing.

B. Web User Interface

Many models exist in the market that enable access to computing resources. Some of them include secure shell connection to submit batch processing jobs [11], remote session [12], web portal [13] etc. All these require the users to have compute resource allocation. In order to support intuitive

access and interactive analysis, we offer a web UI to enable easy access to the computing resources. We have developed a web user interface in collaboration with Weija Xu et al. at Texas Advance Computing Center, The University of Texas [10]. The web UI is designed to use a set of predefined tasks which is specified using a configuration file (workflow). The workflow is reproducible and reusable. Users can use the UI to upload data and user scripts, execute the scripts on the compute cluster and download results. Users require TACC account to login to the website in order to use the HPC and framework via a workflow.

The workflow for web UI is a JSON file. It contains a set of predefined tasks - File upload (Fig. 2), Execute script (Fig. 3,4) and View results (Fig. 5). We use a different workflow for each mode. The workflow differs only for the 'Execute script' task. Each of these tasks is described in detail below:

- File upload - This task allows the users to upload their scripts and input data as zip files. The framework then unzips the files and writes them to the appropriate folder.

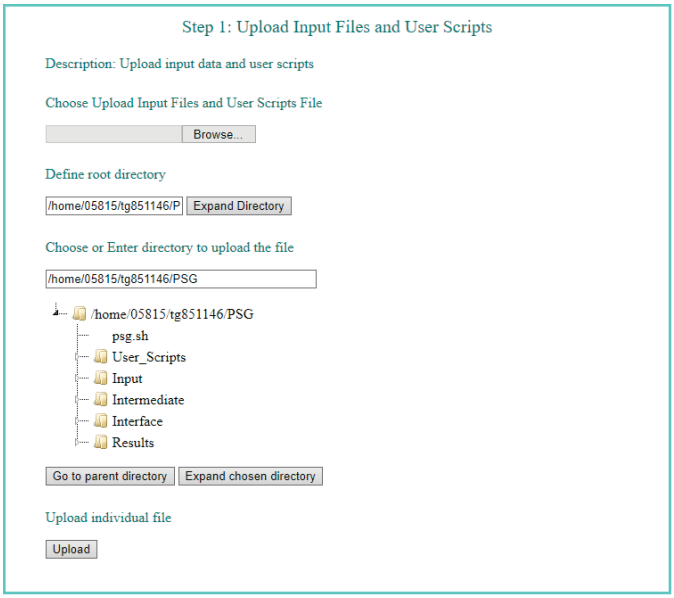


Fig. 2: File Upload.

- Execute script - This task involves executing the user scripts on the input data to produce results. It takes a number of parameters from the user. The parameters for 'parallel job array' mode are:
  - Application name
  - Divide script name
  - Conquer script name
  - No. of cores for parallelization

  The input parameters for 'explicit parallel R' mode are:
  - Application name
  - Parallel script name
  - No. of cores for parallelization

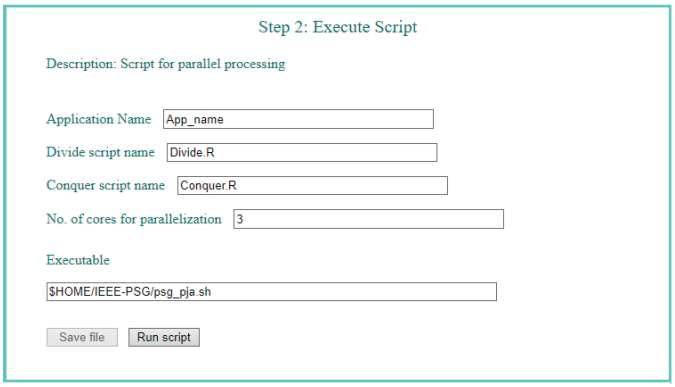


Fig. 3: Execute Script - Parallel Job Array Mode.

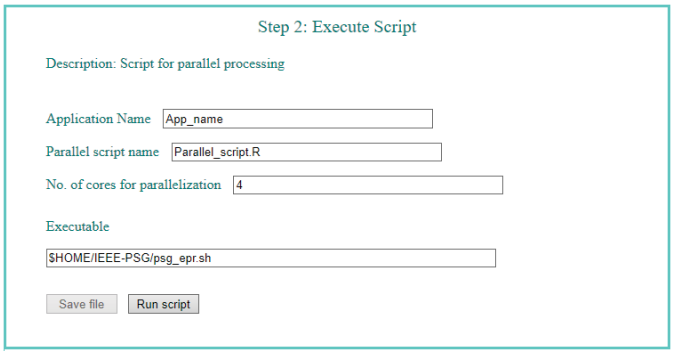


Fig. 4: Execute Script - Explicit Parallel R Mode.

- View Results - This task allows the users to view/download the result. At this time, the users can only view the first 10 lines of the results if it is a text file. All other formats can only be downloaded.

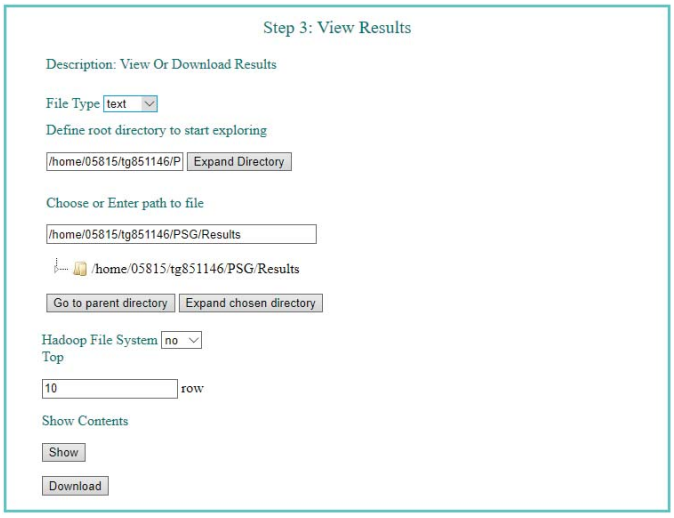
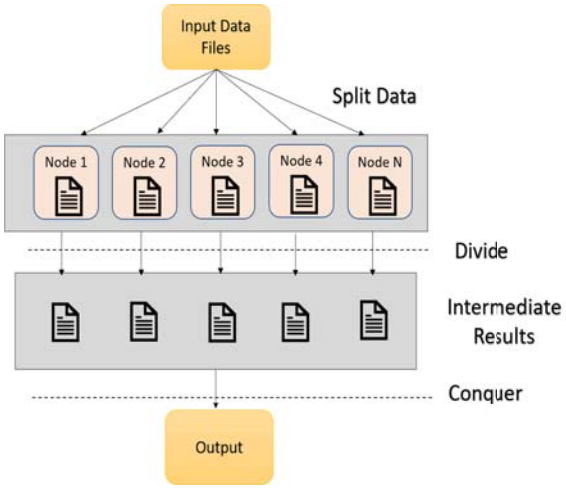


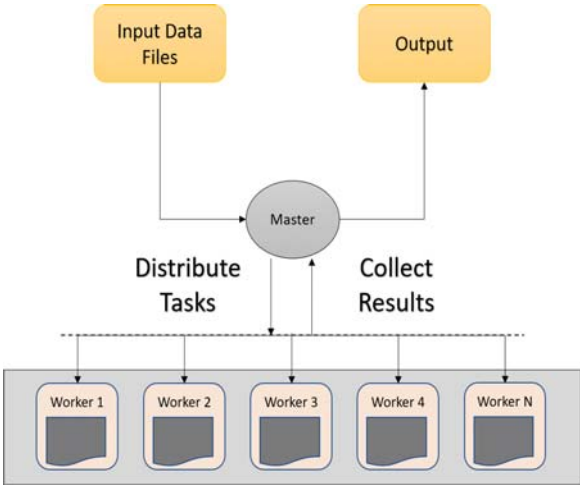
Fig. 5: View Results.

C. Modes of Parallelization

The framework offers two modes of parallelization to cater to the experience level of the user.



(a) Parallel Job Array Mode



(b) Explicit Parallel R Mode

Fig. 6: Modes of Operation. The framework has two modes - parallel job array mode and explicit parallel R mode. Parallel job array mode uses file-based divide-and-conquer paradigm for processing. Explicit parallel R mode uses in-memory processing via master-slave architecture.

1) Parallel Job Array Mode: The goal of this mode is to have a reusable framework for executing sequential algorithms in a parallel environment while hiding the parallelization details from the end user. One approach to parallel processing is to divide the data into chunks and process each chunk in parallel by executing the serial code on each of them. This assumes that each data chunk does not depend on other chunks. This is also the case for simulation runs that are independent of other simulations and can therefore be executed in parallel. This mode uses three-step file-based processing as depicted in Fig. 6a. The proposed algorithm for divide and conquer is as shown in Algorithm. 1 and Algorithm. 2. This mode uses script-level parallelism thereby providing a wider scope for input and output formats. The granularity is at the file-level and this reduces communication and synchronization overhead and enables processing heterogeneous data. It also allows the user to test and debug the sequential code implementation locally before executing it on computing cluster.

The Divide step entails the bulk of the processing defined by user-defined business logic. The result of this step is stored in an intermediate folder which is then used as input for the Conquer step. The conquer step entails light weight processing. It involves aggregating the intermediate results of the divide step and producing the final output. We have parallelized only the divide step because the sequential code can be used without any modification and synchronization issues. Also, the conquer step is not as data intensive. The conquer step will start upon the successful completion of divide step thereby saving precious resources at the time of failure.

The framework follows a predefined folder structure. The user scripts are uploaded to *User_Scripts/App_name* folder. The input data is stored in *Input/App_name* folder. The intermediate results from batch processing mode is written to *Intermediate/App_name* folder and the final results for both modes are written to *Results/App_name* folder.

Algorithm 1: Parallel Job Array Mode - Divide

```
1 Divide (f, i);
  Input:
        ds: Dataset
        f: Filename of ds
        i: Intermediate folder path
2 Read f
3 Perform User-defined computation on f
4 Write IntermediateResult to i
```

Algorithm 2: Parallel Job Array Mode - Conquer

```
1 Conquer (i, o);
  Input:
        i: Intermediate folder path
        o: Output folder path
2 for each IntermediateResult in i do
3 |   Result ← Aggregate
4 end
5 Write Result to o
```

This mode consists of three tasks:

- Split data task - The split data task is responsible for splitting the input files equally among *N* nodes to process them in parallel. This is done by getting the list of all file paths *F* and writing *F/N* file paths to each file. The files are then passed to individual nodes to obtain the file paths for processing. Fig. 7 illustrates the code automatically generated by this task. In line 1, *input* points to the

data location containing the input dataset from user. In line 4, *numNodes* variable holds the number of nodes specified by the user. Lines 6-8, read the number of files in the input folder, obtains the full path for each file by recursively looping through all the sub folders and splits them into a number of chunks which is equal to *numNodes*. In lines 11-18, each chunk is then written to a file. In our example, the path to input files are split into 3 chunks and written to files, files-1.txt, files-2.txt and files-3.txt.

```
1  #User specified dataset - path to input
       files
2  input <- "home/PSG/Input/App_Name"
3  #User specified # of nodes for
       parallelization
4  numNodes <- 3
5  #List of full paths to all input files
6  pathlist <- dir(input, ".*",
       recursive=TRUE,full.names=TRUE)
7  numfiles <- 1: length(pathlist)
8  #Split the paths into 3 chunks
9  chunk <-
       split(numfiles,sort(numfiles%%numNodes))
10 #Each chunk is written to a file - In this
       case 3 files. files-1.txt,
       files-2.txt, files-3.txt
11 for (x in 1:numNodes)
12 {
13   for (c in 1:length(chunk[[x]]))
14    {
15      index <- chunk[[x]][c]
16      write.table(pathlist[index],
            paste0("files-",x,".txt"),
            row.names = FALSE, col.names =
            FALSE,append =TRUE)
17    }
18 }
```

Fig. 7: Code generated by Split Data task. Code fragments in red are user specified parameters.

- Divide task - The user-defined divide script is designed to process one file at a time. The file path is obtained from the file from the previous task. The framework automatically generates batch job to execute the divide task in all the nodes. It loops over all the file paths listed in the file for that node and process them one at a time using the user-defined divide script. Node 1 reads files-1.txt, node 2 reads files-2.txt and so on. This is performed simultaneously in all the nodes until all the files are processed. The results of this task are written to an intermediate folder. The code generated by this task is depicted in Fig. 8. In line 2, *node* variable stores the node#. Since there are 3 nodes as specified in the previous step, the nodes will be numbered from 0 to 2. Lines 3-7 takes input parameters for user-defined divide script name and intermediate folder path to store the output of this step. In line 9, depending on the node#, *filename* variable stores the name of the file auto-generated in the split data task.

```
1  #Environment variable used to obtain the
       node #. The nodes are numbered from 0
       to (numNodes-1)
2  node <-
       as.integer(Sys.getenv()["PMI_RANK"]) +
       1
3  args <- commandArgs(trailingOnly = TRUE)
4  #User defined script name for divide task
5  divide_script <-
       "home/PSG/User_Scripts/App_Name/Divide.R"
6  #Path to intermediate output folder
7  IntermediateFolder <-
       "home/PSG/Intermediate/App_Name"
8  #Name of auto-genegrated file from task 1
9  filename <- paste0("files-",node,".txt")
10 filepaths <- scan(filename,what="")
11 for (fp in filepaths)
12 {
13 system(paste("Rscript ", divide_script,fp,
       IntermediateFolder  ))
14 }
```

Fig. 8: Code generated by Divide task. Code fragments in red are user specified parameters.

- Conquer task - The framework then automatically generates batch job to execute conquer task. The conquer task executes the user-defined conquer script on the intermediate results from the divide task. The conquer script loops over all the files and aggregates them to produce the final output. This task is performed in serial fashion in one node. The code generated by this task is depicted in Fig. 9.

```
1  Rscript
       "home/PSG/User_Scripts/App_Name/Conquer.R"
       "home/PSG/Intermediate/App_Name"
       "home/PSG/Results/App_Name"
```

Fig. 9: Code generated by Conquer task. Code fragments in red are user specified parameters.

2) Explicit Parallel R Mode: In many domains, there are numerous opportunities for discovering new things due to the availability of high volumes of complex data. However, the sheer volume of data poses a challenge in terms of methodologies needed to reduce processing times. The drivers for focus on HPC have been larger datasets and the computation power required for sophisticated methodologies. Some popular methods employed in statistical analysis are bootstrapping, Monte Carlo simulation, Gibbs sampling etc. Rapid growth in data and the demand for simulation methods have been approached with the use of parallel computing.

The goal of this mode is to provide a platform to execute parallel code implementation using multiple cores. This mode does not require the user to follow predefined algorithm

thereby reducing programming cost. The user has full control over the number of steps and the input and output format. The framework provides a convenient way to specify the configuration and job scheduling.

This mode employs in-memory processing model as depicted in Fig. 6b. The model works as follows:

(a) *N* 'worker' processes are initiated. *N* is the number of cores specified by the user.
(b) Any data required for each task is sent to the workers.
(c) The task is split into *N* roughly equally sized chunks, and the chunks (including the R code needed) are sent to the workers.
(d) Wait for all the workers to complete their tasks and obtain their results.
(e) Steps (b-d) are repeated for any further tasks.
(f) Worker processes are shut down.

Although many studies have shown the advantage of using parallel R code over sequential R code, adapting the code for parallel computing, requires skills and extra effort on the part of the researcher. It is up to the user to efficiently use the cores assigned to obtain the desired performance. Parallel code implementations incur computational overhead. Allocating higher number of cores for smaller datasets will result in less or no work for some cores thereby lowering performance and wasting compute resources. Using lower number of cores may not improve performance significantly compared to their sequential counterparts. The users must allocate optimal number of cores for best performance. Users can increase the number of cores for each run to compare performance and analyze the gain per core added. This will help them predict the number of cores needs as the size of the dataset increases.

D. Compute Cluster

We used the XSEDE resource Wrangler [8] at The University of Texas at Austin's Texas Advanced Computing Center (TACC) as the backend to perform computing tasks. Wrangler is the most powerful data analysis system available in XSEDE. The system is designed for large-scale analytics, data transfer and provides support for a wide range of workflows. It is highly scalable for growth in the number of users and data applications. It has more than 3000 embedded processors for data analysis.

E. Resource Manager

Resource management tool can alleviate the challenges that arise with the deployment of parallel computing applications. These include number of jobs, compute cores etc. Resource managers help users submit, control and monitor jobs. SLURM is a highly scalable resource manager. We have used SLURM to execute all computing jobs using the batch processing model. The framework can be put into practice in any computing center where SLURM scheduler has been deployed. SLURM has been deployed at various national and international computing centers, and by approximately 60% of the TOP 500 supercomputers in the world [9].

III. Performance Evaluation

We demonstrate the usability of the framework using two popular use cases - wordcount and Pi estimation using Monte Carlo simulation. We chose these two examples to cover text analysis and simulation, both of which are widely used in analytics. We used XSEDE resource wrangler to evaluate the performance of the framework. Each compute node on wrangler has 48 cores. We have analyzed the performance on three settings - serial setting using for-loops, auto-parallelization using parallel job array mode and explicit parallel R mode using *mclapply* with multiple cores.

A. Wordcount

Wordcount is the problem of counting the number of occurrences of each word in a collection of documents. For performance evaluation, we have used 6 Kb files and varied the number of files from 10 to 200. For parallel job array mode, we have set the number of cores to 10, which means that 10 files will be processed simultaneously. The divide and conquer tasks are executed as two jobs requiring a different set of compute resources for each task. We compare this to sequential code implementation using for-loop by setting the number of cores to 1. For explicit parallel R mode, we have tested *mclapply* with cores=10 and cores=48. In this mode, the code is implemented in one script thereby requiring only one batch job.

Table. I shows performance comparison between the two modes. We can observe that auto-parallelization reduces the execution time by 75 to 78% compared to serial mode. Explicit parallel R mode reduces this further up to 81%. This is due to the fact that this mode requires only one batch job eliminating the wait time in acquiring compute resources for the second job as is the case for the first mode. The second reason is the use of in-memory processing as opposed to file-based processing eliminating the cost associated with file read and write operations.

B. PI Estimation Using Monte Carlo Simulation

Monte Carlo method involves the process of repeated random sampling to make numerical estimations of unknown parameters. It is widely used in the fields of finance, game theory etc. They rely on random number generation to solve probabilistic problems. One example of this method is Pi estimation. For performance analysis, we have varied the number of iterations from 1000 to 100000. We used one core for serial execution. In case of automatic parallelization using parallel job array mode, we set the number of cores to 1000. Whereas for *mclapply* using explicit parallel R mode we set the number of cores to 48. The performance of each setting is depicted in Table. II. At lower iterations, serial mode performs better than auto-parallelization mode. This is caused by the overhead due to workload distribution and output collection. As the number of iterations is increased, parallelization overcomes the delay caused by overhead and offers performance gains. *mclapply* offers the best performance compared to the other two settings. This is due to the fact that serial and auto-parallelization

TABLE I: Comparison of speed-up for Wordcount

Parallel Mechanism	10 files		50 files		100 files		200 files	
	Serial	Parallel	Serial	Parallel	Serial	Parallel	Serial	Parallel
Parallel Job Array Mode								
Auto-parallelization	4	1	15	3	30	8	69	15
Explicit Parallel R Mode								
mclapply (48 cores)	4	0.6	15	0.869	30	1.233	69	2.651
mclapply (10 cores)	4	0.526	15	0.905	30	1.365	69	2.86

TABLE II: Comparison of speed-up for Pi Estimation

Parallel Mechanism	1000 iterations		5000 iterations		10000 iterations		50000 iterations		100000 iterations	
	Serial	Parallel	Serial	Parallel	Serial	Parallel	Serial	Parallel	Serial	Parallel
Parallel Job Array Mode										
Auto-parallelization	2	2	10	13	21	21	100	93	207	187
Explicit Parallel R Mode										
mclapply (48 cores)	2	0.371	10	0.767	21	1.078	100	3.939	207	7.617

settings use file-based processing. Cost associated with read and write operations increases as the number of iterations increases. This cost is eliminated in explicit parallel R mode because it uses in-memory processing.

IV. Related Work

Many software frameworks have been developed for parallel data analysis. These framework emphasize on different design goals, programming language, algorithmic support, parallel computing libraries etc.

SDPPF [27] is a MapReduce based framework for processing spatial data using Java programming language. It enables parallelization of existing binary code with minimum code modification. Compared to SDPPF, our framework also enables automatic parallelization of sequential code using minimum programming effort. However, our framework can be applied for wider range of problems and it support R programming language which is widely used for data analysis. In addition to that, R users are mostly domain experts who are beginners in programming whereas, Java users typically have stronger programming skills. In [31], Anala et al. propose a framework to automatically parallelize serial C code. It works by replacing 'for' blocks with corresponding parallel functions. Our framework, instead, proposes a simple divide and conquer algorithm to rewrite loops and automatically splits the data among different nodes thereby parallelizing computing tasks.

In [28], Craus et al. propose a framework for executing sequential algorithms in parallel environment using MPI library. PEACE [29] is a framework for enabling easier access to clustering methods without compromising performance. It uses MPI to enable parallel computing. These framework focus on the performance offered by using HPC. Our framework emphasizes on providing easier access to HPC using a web UI without compromising on the performance.

Huang et al. in [32] have proposed a framework to parallelize R using Hadoop. It works by allowing users to execute their R scripts developed in single machine environment to be executed on Hadoop without modification. Users must have knowledge of the mechanisms of Hadoop in order to use this system. Our framework provides access to HPC using a web based environment to execute the parallel code implementation without requiring additional knowledge about the underlying system and skills to access it.

V. Discussion and Future Work

This paper proposes a framework that offers two modes of parallel computing for a range of users. It reduces the access barrier in bringing HPC to users and provides two methods for parallel R computing. The current architecture supports large-scale computations and offers a web UI to enhance accessibility. The framework appeals to a wide range of problems, provides flexibility in terms of input and output formats and simplifies job configuration and scheduling.

The framework is still a prototype. There also remain some open questions. Currently, the data must be transferred to HPC for analysis leading to high latency. It also does not support real-time processing. Comprehensive evaluation using varied use cases is required to measure the usability, scalability and performance under the two modes of operation.

Acknowledgment

This work was supported by NSF award #1726532. The workflow and web-based UI is developed in collaboration with TACC and tested on Wrangler cluster.

References

[1] Ostrouchov, George & Chen, Wei-Chen & Schmidt, Drew. (2017). Parallel Statistical Computing with R: An Illustration on Two Architectures.
[2] Markus, Schmidberger & Martin, Morgan & Eddelbuettel, Dirk & Hao, Yu & Tierney, Luke & Mansmann, Ulrich. (2009). State of the Art in Parallel Computing with R. Journal of Statistical Software. 31.10.18637/jss.v031.i01.
[3] MPI (The Message Passing Interface). http://www.mcs.anl.gov/research/projects/mpi/
[4] OpenMP. http://openmp.org/
[5] F. Darema, "SPMD model: past, present and future", Recent Advances in Parallel Virtual Machine and Message Passing Interface, 8th European PVM/MPI Users' Group Meeting, Santorini/Thera, Greece, 2001.
[6] T. Iwashita, A. Ida, T. Mifune and Y. Takahashi, "Software framework for parallel BEM analyses with H-matrices," 2016 IEEE Conference on Electromagnetic Field Computation (CEFC), Miami, FL, 2016, pp. 1-1.doi:10.1109/CEFC.2016.7816379.

[7] M. E. Azema-Barac, "A conceptual framework for implementing neural networks on massively parallel machines," Proceedings Sixth International Parallel Processing Symposium, Beverly Hills, CA, 1992, pp. 527-530. doi: 10.1109/IPPS.1992.222973
[8] Wrangler computing cluster https://www.tacc.utexas.edu/systems/wrangler
[9] SLURM Workload Manager https://hpcc.usc.edu/support/documentation/slurm/
[10] Xu, Weijia Huang, Ruizhu Wang, Yige. (2018). Enabling User Driven Web Applications on Remote Computing Resource. 10.1109/SERVICES.2018.00038.
[11] Yoo, M. Jette, and M. Grondona, "Slurm: Simple Linux Utility for Resource Management," Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, vol. 2862, pp. 44-60, 2003
[12] Rstuido Team. R Studio. [Online]. https://www.rstudio.com/
[13] Merchant, Nirav, et al., "The iPlant Collaborative: Cyberinfrastructure for Enabling Data to Discovery for the Life Sciences," PLOS Biology, 2016.
[14] R snowfall package. http://cran.r-project.org/web/packages/snowfall/index.html/.
[15] R multicore package. http://cran.r-project.org/web/packages/multicore/index.html/.
[16] Package 'Parallel'. https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf.
[17] RHadoop. https://github.com/RevolutionAnalytics/RHadoop/wiki/.
[18] R MPI package. http://cran.r-project.org/web/packages/Rmpi/index.html/.
[19] R foreach package. https://cran.r-project.org/web/packages/foreach/index.html.
[20] Xu, Weijia & Huang, Ruizhu & Zhang, Hui & el-Khamra, Yaakoub & Walling, David. (2016). Empowering R with High Performance Computing Resources for Big Data Analytics. 10.1007/978-3-319-33742-5_9.
[21] Dean, Jeffrey Ghemawat, Sanjay. (2004). MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM. 51. 137-150. 10.1145/1327452.1327492.
[22] Yuhuan Cui, Jingguo Qu, Weili Chen and Aimin Yang, "Divide and conquer algorithm for computer simulation and application in the matrix eigenvalue problem," 2009 International Conference on Test and Measurement, Hong Kong, 2009, pp. 319-322.doi:10.1109/ICTM.2009.5412930.
[23] Stuart Hadfield and Anargyros Papageorgiou, "Divide and conquer approach to quantum Hamiltonian simulation", 2018 New Journal of Physics.
[24] Horowitz and Zorat, "Divide-and-Conquer for Parallel Processing," in IEEE Transactions on Computers, vol. C-32, no. 6, pp. 582-585, June 1983. doi:10.1109/TC.1983.1676280
[25] Hui Zhang, Yiwen Zhong, Juan Lin, "Divide-and-Conquer Strategies for Largescale Simulations in R", 2017 IEEE International Conference on Big Data (BIGDATA)
[26] Subramanian, Ranjini Zhang, Hui. (2018). Performance Analysis of Divideand-Conquer strategies for Large scale Simulations in R. 4261-4267. 0.1109/Big-Data.2018.8622068.
[27] D. Zhao and Z. Huang, "SDPPF - A MapReduce based parallel processing framework for spatial data," 2011 International Conference on Electrical and Control Engineering, Yichang, 2011, pp. 1258-1261. doi: 10.1109/ICECENG.2011.6057775.
[28] M. Craus and L. Rudeanu, "Parallel framework for ant-like algorithms," Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, Cork, Ireland, 2004, pp. 36-41. doi: 10.1109/ISPDC.2004.37.
[29] Dhananjai M. Rao. 2018. A parallel framework for ab initio transcript-clustering. In Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings (ICSE '18). ACM, New York, NY, USA, 331-332. DOI: https://doi.org/10.1145/3183440.3194995.
[30] L. Lingqiao, Y. Huihua, H. Qian, Z. Jianbin and G. Tuo, "Design and Realization of the Parallel Computing Framework of Cross-Validation," 2012 International Conference on Industrial Control and Electronics Engineering, Xi'an, 2012, pp. 1957-1960. doi: 10.1109/ICICEE.2012.520.
[31] M R, Anala & Dash, Deepika. (2018). Framework for Automatic Parallelization. 112-118. 10.1109/HiPCW.2018.8634283.
[32] Y. Huang, Y. Chen, C. Tsai and H. Hsiao, "Parallelizing R in Hadoop (A Work-in-Progress Study)," 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), Chengdu, 2015, pp. 1114-1116. doi: 10.1109/SmartCity.2015.218.
[33] M. Liang, C. Trejo, L. Muthu, L. B. Ngo, A. Luckow and A. W. Apon, "Evaluating R-Based Big Data Analytic Frameworks," 2015 IEEE International Conference on Cluster Computing, Chicago, IL, 2015, pp. 508-509. doi: 10.1109/CLUSTER.2015.86.