Design and Concept of an Osmotic Analytics Platform based on R Container

Dominik Grzelak[1], **Johannes Mey**[1], **and Uwe Aßmann**[1]
[1]Chair of Software Technology, TU Dresden, Dresden, Saxony, Germany

Abstract—*Advanced data analytics are adopted in nearly every business area like healthcare, financial services, and manufacturing. However, it takes time to develop a mature and profound analysis for decision-making processes. The increasing availability of cloud computing and capabilities at the network edge allows transferring computation of machine learning algorithms more seamlessly between cloud data centers and edge nodes aiming at highly distributed and federated environments. To address this challenge, we propose an osmotic computing platform named RAPTOR based on R containers which exploit resources in the cloud and edge infrastructures. R container enabling portable, shareable, version controlled and modularized data analytic applications supporting a DSL providing each user an appropriate view of the whole development life-cycle of advanced analytics applications.*

Keywords: osmotic computing platform, advanced data analytics, R container

1. Introduction

Currently, many big data processing frameworks and databases exist and more and more are evolving. Nevertheless, it takes time to develop a mature and profound analysis for mission-critical decision-making processes. Building new applications that include data analysis when starting from scratch is a time-consuming task that comprises rewriting analysis models, designing and deploying infrastructure, integrating and setting up big data frameworks, and much more. Building all of that necessary technical conditions takes a lot of time and effort.

Formerly, only statisticians provided predictive analytics. However, with the increasing amount of data, these solitary operations naturally evolved into specialized job profiles (cf. [1]) addressing the challenge of processing big data (see Fig. 1). Over the years it has become evident that these profiles have to collaborate effectively to master and benefit from the enormous amount of data.

We propose a platform, container format, and a domain-specific language (DSL) for the provision of stand-alone data analysis applications, allowing the creation and distribution of reliable and repeatable data analyses. For this purpose, R is used as the base computing resource.

Though R is an optimal choice as computing resource (see Section 2), it has limitations in an enterprise context. Firstly, the statistical software primarily runs in single-user mode [2]. Secondly, no native parallelization in R exists, but sophisticated package solutions can be included that are also exploited for the proposed architecture. To cope with these limitations, many solutions were developed [3], [4], [5], [6] trying to tackle the issues on different levels solving dedicated parts of the problem.

With our novel approach, the proposed platform aims at highly distributed and federated environments, and enables the automatic deployment of data analysis application that are composed and interconnected over both edge and cloud infrastructures which can be integrated in other systems and applications—therefore complying to the osmotic computing paradigm.

The goal is the *separation of concerns of scientific computing from application and implementation details*, thus, abstraction and an embedding of big data workflows in dedicated systems.

Applications integrating data analysis components require a programmable interface which defines statistical operations independently of any programming language – having a unified interface and infrastructure for deploying data analysis workflows (developed with R at the basis) and a DSL to facilitate the construction of different views on the analysis process.

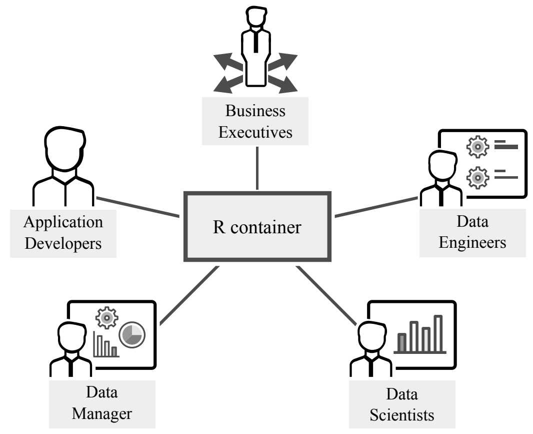

Fig. 1: In big data analysis environments, different profiles work together (cf. [1]), each with their own view on the platform, process, and data.

The main contributions of our work are as follows:

- We propose a novel methodology and data analytic Platform-as-a-Service using R as base programming language aiming at distributable, shareable, modularized applications that can be integrated into other systems and applications by utilizing the Function-as-a-Service paradigm.
- Defining important aspects and components of such a platform using a model-based approach which describes different applications within a container using a DSL to separate the different concerns of each actor of the system.
- Defining concepts for leveraging compute resources of analysis applications spanning cloud data centers as well as edge networks.

The involved concepts for the realization of such an osmotic data analysis platform are introduced in Section 2. The remainder of this article describes the requirements of vital key features (Section 3) and discusses the general architectural design (Section 4).

2. Concepts

In this section, we describe the basic underlying concepts that are involved in designing the proposed data analysis platform.

2.1 R

R is an open-source statistical software and language, developed by Ross Ihaka and Robert Gentleman in 1990. It is widely used among data scientists, in industry, especially in bioinformatics, and chemistry, and is taught in schools and universities. R is placed among the popular languages used today [7] which is due to R's versatile nature and richness of functions [5], [8].

It is an imperative language in its core but also supports object-oriented and functional programming paradigm. R packages are available for nearly every problem in scientific research. Additional benefits are a sophisticated graphics and plottings system and import and export functionality for many commonly used data exchange formats. Utilizing the Rcpp packages allows the implementation of fast algorithms in C/C++. Further, the language is designed to interface with big data software (e.g., Hadoop, Spark, Cassandra, Hive).

2.2 Cloud, Edge and Osmotic Computing

Cloud computing is based on four layers and offers different services: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), Software-as-a-Service (SaaS) and Function-as-a-Service (FaaS). An important benefit of cloud computing is scalability, which allows elasticity of hardware resources. Additionally, cloud computing allows on-demand access to computing power and shared computing resources, and also releases it when not needed. The user does not need to focus on the hardware as it abstracts the physical layer entirely and instead can focus on storing and processing data.

Edge computing is pushing data from computing applications and services away from centralized server nodes to the logical extremes of a user's network (e.g., routers). This paradigm allows physical things (e.g., lamps, sensors) to process the data they gather locally together with additional domain knowledge leveraging artificial intelligence and machine learning algorithms to make sense of the data. Application processing is performed by a small edge server positioned between the cloud and the sensor in a location physically closer to the user. The workload from the cloud will be offloaded to a user's device for processing while at the same time speeding up applications that require a low latency response.

In *osmotic computing*, applications are broken down into smaller microservices, which then deftly exploit the resources in cloud and edge infrastructures. The same happens with the distribution of applications deployed in edge and cloud system. Thus, in an osmotic ecosystem two main infrastructure layers are involved, that is, cloud and edge computing [9].

3. Components and Methods of Osmotic Analytics

In this section we introduce the required components that are necessary to provide an individual view for each of the mentioned profiles illustrated in Fig. 1 as well as the mechanisms to deploy and use them effectively in a distributed edge cloud environment.

3.1 Collaborative Workbench

The online workbench allows the collaboration of the different profiles and provides consistent and concise cooperation, giving every actor their appropriate view to work on an analytics application throughout its life-cycle.

Access to data and workflow scripts are controlled in a centralized manner which enables sharing and knowledge management. All necessary information are stored in a single underlying model to execute and configure analysis workflows. Sharing of analysis applications between users is possible by utilizing an application container concept.

3.2 Application Container

Application containers are regarded as R containers in this paper as they utilize R as the base programming platform. Within such a container, different types of analysis application can be defined and implemented by using a DSL, for example, single jobs, workflows consisting of many jobs, and notebook systems (see Section 4.1.1). Workflows comprise R scripts that are composed to a process pipeline. Notebooks are interactive documents where the user can execute each step separately. Algorithms are written directly in R by the data scientist: no transformation of R in other high-level languages is necessary at any stage of the development process.

Due to the container-based approach, a standardized and packaged analysis workflow can be realized which makes data analysis application portable and shareable. Moreover, the container is version-controlled to reproduce past work. Data (e.g., files and plots) that are generated from these workflows are stored alongside the container as well. This functionality allows the comparison of results with previous versions which adds additional value for historical data analysis.

3.3 Integration Requirements

Nowadays, the focus lies on the integrability of data analytics in other systems in addition to the development process itself. Therefore, one goal is to minimize the efforts of building infrastructures, transforming data analysis model into higher-level languages which is an error-prone and costly process. A typical analytics process usually takes three to six months in average from the identification of the problem, until the delivery of the final application.

However, the whole development life-cycle of advanced analytics applications can be done in a single language, that is R. Machine learning algorithms written at the beginning of the process can be embedded into heterogeneous systems. That means that no transformation of predictive models in other high-level languages is needed at different steps in this development process. As a result, this facilitates rapid prototyping of new computational methods and the exploration of many concepts. Re-implementation is discouraged, instead direct operationalization of algorithms is optimized by this approach.

Data analysis workflows can be built and deployed as RESTful web services that can be consumed concurrently by multiple users and systems. That means R containers are the entry point for a serverless computing architecture. Application developers can integrate exposed endpoints of an R container in dedicated systems enabling the use case for FaaS by supporting on-demand functionality. Using this approach allows the exposure of business functionality as REST services.

Usually, analytics applications need to communicate with other resources like databases and file systems to gather data for processing. The container facilitates a way to conduct quantitative work using popular big data tools through the concept of bindings which are described with the DSL too. It supports the integration of common big data solutions in a way that these functionalities can be used directly in the analysis application within the container without specifying additional dependencies, absolute file paths or using system-wide libraries which also makes the application portable and shareable.

R containers are platform-neutral and allow the integration of analysis application in any infrastructure by utilizing the Platform-as-a-Service layer of cloud computing. As a result, the deployment and execution of analysis workflows are enabled on any compute node.

3.4 Parallel Computing

First, parallel computing within R is mainly supported by using specific R packages, e.g. snow [10], Rmpi [11] or pbdDMAT [12], and common big data tools like SparkR [13], Hadoop via Ricardo [14] or Rhipe [15]. Though, using this approach assumes that R users (i.e., primarily data scientists) are familiar with distribution mechanisms like Message Passing Interface (MPI) programming, and setting up compute clusters. Therefore, a primary objective for a data scientist must be that he/she need not call low-level R functions to create clusters, master, and worker nodes when using, e.g., a machine learning algorithm. Simplifying the adoption of these tools by data scientists, the usage and configuration are done by using the DSL (see section 4.1.2). The advantage is that only external configuration is necessary to use the parallelization frameworks in an application inside the R container. Thus, the platform manages the setup and provides all packages, with no effort required by the data scientist.

The second parallelization level of the platform is of a higher one. The parallel processing of workflow parts is done by orchestrating individual *stages* (that is, R scripts) of a workflow to be executed in parallel and allowing forks and joins to collect intermediate results. We defer Section 4.3 an explanation of the workings.

Edge nodes and cloud data centers have local and performance differences. The R container in conjunction with the DSL aims to make these parallelizations and distribution of workload transparent for the system user. An osmotic scheduler of the platform decides which part of an application of an R container should run in the cloud or the network edge. The configuration of the auto-scaling feature allows defining the minimum and the maximum number of nodes in a cluster and where parts of the application should run.

4. R data Analysis PlaTfORm

In this section, we explain what components and services are appropriate to implement the stated requirements described in Section 3 and provide a detailed description of the interactions and inner workings for each part of the system. The R data analysis platform (RAPTOR) introduces three key concepts:

1) the R container format,
2) a DSL, and
3) the platform itself

which are closely connected with each other. The high-level system architecture is depicted in Fig. 2 which shows all three main aspects in context.

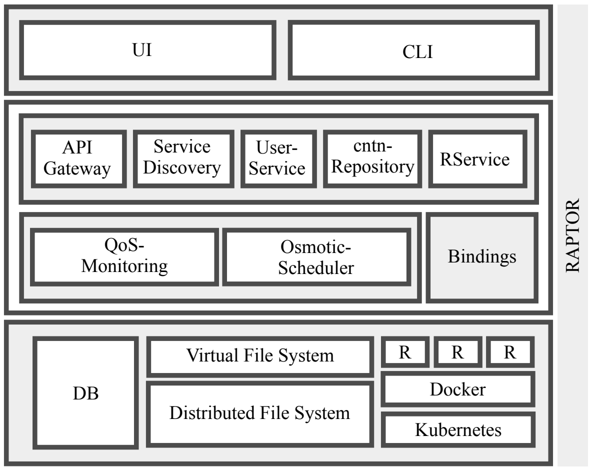

Fig. 2: High-level architecture of RAPTOR with all main components.

This architecture is designed to be highly distributed platform and opted for performance, scale, and reliability which is achieved through the service-oriented architecture—every component in each layer can be distributed and replicated. Moreover, osmotic specific variants of each component exist to operate efficiently in the particular environments, for example, lightweight versions that can be executed on edge nodes. The platform serves the Platform-as-a-Service concept to realize the requirements that are implemented with the R container in combination with the DSL.

The collaborative workbench is implemented as web-based user interface and a command-line interface. Both clients access underlying microservices via a unified interface, namely the *API gateway* which serves as a single entry point that proxies and routes all requests made by the clients. The *UserService* is a microservice used for identity management. The *cntnRepositoryService* allows access to private and public-made R containers. All microservices automatically register at the *ServiceDiscovery* once started. It knows the virtual or physical location of each service and can provide other services with that information. For instance, the *API gateway* contacts the *ServiceDiscovery* to route the request to the corresponding service.

The *QoSMonitoring* service provides Quality of Service (QoS) metrics and continuously measures and logs the performance of all microservices and R containers. In cooperation between this QoS service and the *OsmoticScheduler*—which is utilized as Kubernetes Scheduler—the deployment and distribution of these microservices and computing applications of an R container are implemented. The *RService*'s primary task is to manage the execution of R container applications which is explained in the following sections.

In the following, the architectural concepts of the individual parts are explained.

4.1 R Container

In this section, we introduce the R container (cntnR). This type of container is a specialized application container that packages and runs a defined data analysis services (see Fig. 3). The DSL in conjunction with the cntnR represents the single underlying model which allows the provision of stand-alone data analysis applications in a reliable and repeatable manner by the platform, giving each user (refer to Fig. 1) a different view at the analysis development process.

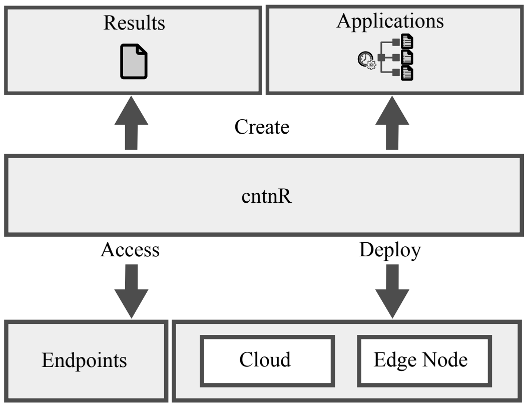

Fig. 3: The R container (cntnR) approach: creation, deployment and access of data analytics applications.

The cntnR is a data format where a DSL is used to: i) describe various kinds of applications (Section 4.1.1), ii) bind and interact with external tools like servers, databases or file systems (Section 4.1.2), iii) defining endpoints to R scripts and functions with access restrictions, iv) storing the actual R scripts and results like plots and files.

The applications defined inside a cntnR are deployed via Docker on various computing nodes. Rserve [16] is used to access R in a high-level programming language like Java or Python. A Docker container is orchestrated by where individual applications and their parts defined in a cntnR are executed. A cntnR can be deployed into the cloud or in the network edge. The cntnR gets user-defined storage allocated and has access to the distributed file system (DFS) whenever an application is executed. Generated results are stored permanently as long as it is not deleted and are connected with the particular container. Data, plots, and scripts are accessed via web endpoints.

Accessing the underlying DFS is provided by a virtual file system (VFS) layer which provides a unified API for controlling access to different file system types as well as files of various types. This approach abstracts the access to the data for every cntnR and microservice of RAPTOR. Therefore, achieving general and direct access across different platforms.

By using a DSL different types of applications can be specified where the actual workflows are composed of R scripts (Section 4.1.1). The usage and communication with other big data tools and databases are supported by using the bindings concept. Moreover, R version and packages are persisted and standardized to provide reasonable assurance regarding the reliability of data analysis applications within a cntnR. A configuration file stores all used packages and the currently used version of the R instance.

4.1.1 Application Types

The main purpose of such a container is to "package" data analysis algorithms, workflows, notebooks and web services into a standardized format ready for distribution and deployment. As illustrated in Fig. 3 different applications can be created with a DSL. Applications usually generate data which can be accessed externally via pre-defined endpoints.

a) Workflows: Workflows are an expression of an user-defined data analytics pipeline and are modeled via a DSL. The entire data analytics process is defined and separated in *stages* which includes probably data preprocessing, model fitting, evaluation, prediction, and testing. Single stages of a workflow are user-defined R scripts. Using parallelization frameworks from R allows multi-threaded execution of algorithms within a stage and can be incorporated by defining bindings or including R packages for parallel computing. Various constructs achieve the workflow composition. That is, the workflow DSL supports fork/join functionality, loops, and parallel execution of stages and assigning stages to edge nodes or into the cloud. Furthermore, workflows can be scheduled supporting reoccurring tasks. A particular type of a workflow is a job, consisting only of one stage. Thus, allowing fine-grained services that can be reused easily.

b) Notebook: Notebook systems facilitate an interactive user interface and offer the user a hands-on approach to interact with the data analytics application directly. Data can be accessed with different programming languages, e.g., SQL and R, to change input parameters to parametrize models. Moreover, it supports web-based programming and plotting. Therefore, the results and the performance of machine learning algorithm can be verified. One versatile framework of such a notebook system is Apache Zeppelin (https://zeppelin.apache.org/) which enables data-driven data analytics through interactive documents for collaborative work. The Apache Zeppelin interpreter allows to plugin the R language to be used within the notebook system which makes this framework an optimal choice to create notebook application defined within a cntnR.

c) Web services: R functions in a cntnR can be deployed as RESTful web services by exposing them as endpoints allowing serverless computing. The execution of the function is made available as service by the proposed platform. As REST is platform-agnostic the consumption of these services is possible from external systems. Consuming REST services are provided on every platform in several ways. This makes the integration of data analytics in every generic application straightforward. The application developer configures endpoints to specific functions within a cntnR, specifying who and what can be accessed and executed externally. Thus, supporting the integration in other systems so that the algorithm and results can be operationalized directly with no transformation cost.

4.1.2 Bindings

The bindings concept enables the integration of external systems and big data tools, web services and file systems within a cntnR making it available for the different application types. Therefore, a DSL is utilized to describe the required configuration. Fig. 4 shows an extract of possible bindings that can be used within a cntnR.

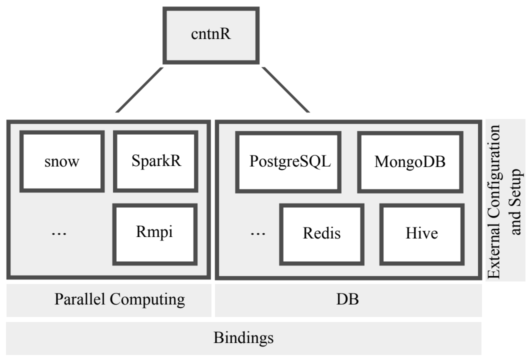

Fig. 4: Types of bindings that can be described with a DSL to be included in an application within a cntnR.

Hence, this makes data analysis application portable by discarding system-wide libraries or additional dependencies in the workflow itself, thus, separating the configuration details and implementation. A data scientist concentrates on using the functionality whereas big data engineers describe the configuration. Therefore, the integration of popular big data tools is simplified and is easier to be adopted by data scientists.

We illustrate this with an example for SparkR. Spark is a hybrid processing engine that can process in batch-mode as well as process streaming data. The configuration of this framework is described by the DSL to define the master host and port, flags and the version. For testing purposes, a standalone pseudo-cluster variant of Spark can be created on-the-fly. This makes cntnR workflows optimal for early development of data analytics applications.

4.2 DSL

In order to achieve the separation of concerns regarding the analysis process logic and computing application details, a DSL is used to generically compose and orchestrate data analysis applications, defining bindings and endpoints and, thus, giving each user of the platform (see Fig. 1) the appropriate view of the process.

Based on the binding description by the DSL, the *RService* creates the required R code and passes the necessary information to Kubernetes for execution and distribution. If workflows are created, the DSL is used for configuration which workflow stages should run in the cloud or edge (see Section 4.3). That means, the platform uses Kubernetes to auto-scale and distribute the workload of a cntnR application.

4.3 R Container Workflow Orchestration

The platform utilizes Kubernetes to orchestrate the applications defined within a cntnR on cloud data centers and edge devices, as well as the containerized microservices included in RAPTOR. Kubernetes (https://kubernetes.io) is an open-source system for automating deployment, management of clusters of containerized applications, and resource management.

Firstly, Kubernetes manages the deployment and provision of the microservices *UserService*, *RService* and *CntnRepository*. Monitoring the performance of the microservices is accomplished with the help of the *QoSMonitoring* service. Several parts of the platform are also deployed to edge nodes like routers or small single-board computers by providing a specific lightweight component variant of these services. The *OsmoticScheduler* is a custom Kubernetes scheduler implementation using QoS monitoring metrics provided by the *QoSMonitoring* service. This custom scheduler is containerized as Docker image and then deployed as a Kubernetes pod. The osmotic scheduler of the platform decides, based on multiple metrics, which part of a cntnR application can be run in the cloud or the network edge. Additionally, the autoscaling feature of Kubernetes is utilized, based on metrics like observed CPU utilization, to withstand heavy demands and achieve high utilization. Moreover, we can configure the minimum and the maximum number of nodes in a cluster.

Secondly, the *RService*'s task is to execute applications defined within a cntnR by instructing Kubernetes to start and distribute Docker containers, set up the defined bindings, and managing R sessions and resources, according to the definition by the DSL. The *RService*'s responsibility is to construct the defined bindings by generating the corresponding R code that is necessary to set up the environment for the included library (see Fig. 4). Using this approach, the data scientist can immediately incorporate the binding's functionality without having to worry about configuration. Utilizing existing parallelization methods within a workflow stage (that is, a R script) is done by explicitly including R packages. Moreover, workflows support high-level parallel computing features for stages by defining abstract distribution patterns for them (see Section 4.1.1).

The use of Kubernetes effectively proved that those devices incorporated in an application could be run in a managed way where the user does not have to worry about capacity planning and the scaling.

5. Summary

We have outlined the essential characteristics and concepts of RAPTOR, an osmotic data analysis platform based on R container for the creation, deployment, and integration of data analysis applications. The result is a highly-scalable and distributable platform which uses osmotic computing paradigms to provide advanced data analytics for the cloud and the edge.

The primary goal is to embed machine learning algorithms and analysis workflows into complex systems. The direct operationalization of analytic processes written in R is the main motivation for abstraction and embedding in dedicated systems and applications using cntnRs with the corresponding DSL. The DSL allows organizations and users flexibility in deciding how to interact with the data and applications created within a cntnR that can adapt to the current environment. Analysis workflows defined within a cntnR can be migrated across cloud-centric data centers to the network edge. Kubernetes is used, which effectively enables automatic scaling to distribute large CPU and memory footprints and leverage compute resources more efficiently in cloud and edge infrastructures.

As a result, our proposed cntnR format in conjunction with a DSL is a concise concept that is suitable for distribution and deployment. Moreover, this holistic and flexible approach allows the definition of various types of applications. The workflow and notebook paradigm is easier to be adopted by data scientists using R while web services are the lingua franca for application developers. The platform is endowed with multiple ways to utilize parallelization on different levels: i) by using R packages directly in a workflow, ii) the bindings concept for the usage of big data tools within R without dealing with configuration details, iii) or the DSL to compose workflow stages for distribution at a higher level. Moreover, the deployment of data analysis algorithms within a cntnR as web service is supported to provide these algorithms concurrently to multiple users and to integrate them into dedicated systems. Thus, making computing applications shareable and portable with no direct dependencies to these tools and R packages. The platform and cntnRs ensure that packages and R versions comply with the definition via the DSL which makes reproducible work feasible and reliable. Additionally, every change within a cntnR is tracked automatically and version controlled making reproducible research possible for the dissemination of statistical methodology and arguments.

As a result, only one single language (i.e., R) and a DSL leveraging an osmotic PaaS is necessary throughout the whole life-cycle of advanced analytics processes from the development to the final distribution of the finished product.

Acknowledgment

This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 692480 (IoSense). This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Spain, Austria, Belgium, Slovakia.

Additionally, this work has partially been funded by the German Research Foundation within the Collaborative Research Center 912 Highly Adaptive Energy-Efficient Computing, and the research project "Rule-Based Invasive Software Composition with Strategic Port-Graph Rewriting" (RISCOS).

References

[1] T. M. Matt Ariker, "Five Roles You Need on Your Big Data Team," July 2013. [Online]. Available: https://hbr.org/2013/07/five-roles-you-need-on-your-bi

[2] R. Kabacoff, *R in Action: Data Analysis and Graphics with R*, 2nd ed. Manning Publications, 2015.

[3] H. Lin, S. Yang, and S. P. Midkiff, "RABID – A General Distributed R Processing Framework Targeting Large Data-Set Problems," in *2013 IEEE International Congress on Big Data*, June 2013, pp. 423–424.

[4] Y. Tong, Z. Zheng, D. Fu, Y. Fu, and S. Li, "A Cloud Computing Platform for Data Analysis Based on R Cluster," in *2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, Oct. 2016, pp. 243–248.

[5] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. Yang, and J. Zhang, "Bioconductor: open software development for computational biology and bioinformatics," *Genome Biology*, vol. 5, p. R80, Sept. 2004. [Online]. Available: https://doi.org/10.1186/gb-2004-5-10-r80

[6] J. Ooms, "The OpenCPU System: Towards a Universal Interface for Scientific Computing through Separation of Concerns," *arXiv:1406.4806 [stat.CO]*, 2014. [Online]. Available: http://arxiv.org/abs/1406.4806

[7] S. Cass, "The 2017 Top Programming Languages," July 2017. [Online]. Available: https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages

[8] J. Fox and A. Leanage, "R and the Journal of Statistical Software | Fox | Journal of Statistical Software," Sept. 2016. [Online]. Available: https://www.jstatsoft.org/article/view/v073i02

[9] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic Computing: A New Paradigm for Edge/Cloud Integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, Nov. 2016.

[10] L. Tierney, A. J. Rossini, and N. Li, "Snow: A Parallel Computing Framework for the R System," *International Journal of Parallel Programming*, vol. 37, no. 1, pp. 78–90, Feb. 2009. [Online]. Available: https://link.springer.com/article/10.1007/s10766-008-0077-2

[11] H. Yu, "Rmpi: Interface (Wrapper) to MPI (Message-Passing Interface)," Apr. 2018. [Online]. Available: https://CRAN.R-project.org/package=Rmpi

[12] D. Schmidt, W.-C. Chen, G. Ostrouchov, P. Patel, Z. Wang, and M. Lawrence, "pbdDMAT: 'pbdR' Distributed Matrix Methods," Oct. 2016. [Online]. Available: https://CRAN.R-project.org/package=pbdDMAT

[13] S. Venkataraman, Z. Yang, D. Liu, E. Liang, H. Falaki, X. Meng, R. Xin, A. Ghodsi, M. Franklin, I. Stoica, and M. Zaharia, "SparkR: Scaling R Programs with Spark," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: ACM, 2016, pp. 1099–1104. [Online]. Available: http://doi.acm.org/10.1145/2882903.2903740

[14] S. Das, Y. Sismanis, K. S. Beyer, R. Gemulla, P. J. Haas, and J. McPherson, "Ricardo: Integrating R and Hadoop," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 987–998. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807275

[15] E. McCallum and S. Weston, *Parallel R*. O'Reilly Media, Inc., Oct. 2011.

[16] S. Urbanek, "Rserve – A Fast Way to Provide R Functionality to Applications," in *Proc. of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, 2003. [Online]. Available: http://Rosuda.org/Rserve