UNIVERSITY OF AUCKLAND

Faculty of Science

Department of Statistics

Doctoral Research Proposal

A Platform for Large-Scale Statistical Modelling Using R

Supervisors:
Dr. Simon Urbanek (Main)
Assoc. Prof. Paul Murrell (Co)

Student:
Jason Cairns
jcai849
8092261

2021-05-18

Contents

1 Introduction **2**

2 Background **3**
2.1 Local Systems . . . 5
2.2 Distributed Systems . . . 5
2.3 Evaluation . . . 7

3 Methodology **7**

4 Preliminary Results **8**
4.1 Overview of **largeScaleR** . . . 9
4.2 **largeScaleR** System Usage Patterns . . . 10
4.3 Example Session . . . 12
4.4 Issues Encountered . . . 12

5 Future Work **15**

6 Objectives & Goals **16**

7 Deliverables and Program Schedule **17**

8 Budget **19**

List of Figures

1 **disk.frame** architecture with four chunks . . . 5
2 **SNOW** architecture with four chunks . . . 6
3 Example distributed object reference relations . . . 9
4 Example distributed object communication relations with redundancy example in chunk 1 . . . 10

List of Listings

1 Initial input to the distributed system . . . 12
2 Exploration of the structure of the flights dataset . . . 13
3 Dataset manipulation to attain final table . . . 14

List of Tables

1 Top 10 most important functions in the **largeScaleR** package . . . 8
2 Talks attended as part of first year goals . . . 18
3 A timeline of future work . . . 19

1 Introduction

Statistics is concerned with the analysis of datasets, which are continually growing bigger, and at a faster rate; the global datasphere is expected to grow from 33 zettabytes in 2018 to 175 zettabytes by 2025[30]. The scale of this growth is staggering, and continues to outpace attempts to engage meaningfully with such large datasets.

A complementary exponential growth in computational capacity, described by Moore's Law, underlies much of computational advancement over the past half-century[25]. Similar observations on computer storage posit an increase in density of storage media along with corresponding decreases in price, which has been found to track lower than expected by Moore's law metrics[11]. The increasing differentials between the generation of data, computational capacity, and constraints on data storage, have forced new techniques in computing for the analysis of large-scale data, with clear solutions still distant.

To take a concrete example of the problem, consider how a statistician may attempt to fit a novel model for a dataset consisting of roughly 160 million flight datapoints[38], using methods and computational facilities typical to a small dataset. This is actually a small dataset compared to many other large datasets, yet it is still not possible to perform an analysis in the same manner as would usually be conducted on small-scale datasets. `R`, or any other common statistical computing system, simply won't be able to read in the data in the same fashion, as it is too big to fit in memory. The reason for this failure lies in the memory hierarchy of computers, wherein different forms of data storage utilised by computers have varying response times and volatility. Using the Dell Optiplex 5080 as a typical desktop PC build, the statistician has 16 GB of Random Access Memory (RAM) for fast main memory, to be used as a program data store; and a 256 GB Solid State Drive (SSD) for slower long-term disk storage[16]. The referenced dataset takes up roughly 12 GB on disk, increasing to 16 GB when read into `R`—simultaneously competing for space with the operating system, which may be 8 GB in the case of Windows 10. There just isn't enough space on standard RAM to fit everything, and the modelling process will either crash `R` or leach out into swap space on disk as on a `UNIX` system, which is extremely slow and unstable (a situation known as "thrashing")[8]. Furthermore, even if the dataset was halved in size and able to be read into `R` or whichever other program used for modelling, operations on the dataset will necessarily use more memory. The program may potentially create entire copies of data, and the memory available would quickly be exceeded. The problem can be summed up in the need for real-time handling and modelling of datasets that are too large to fit in memory.

As a major and growing issue, there have been a plethora of responses over decades, which will be described in further detail in Section 2 below. None of the responses are entirely satisfactory for the statistician working with large datasets, who may reasonably be posited to possess the following demands:

- A platform that can enable the creation of novel models and apply them to larger-than-memory datasets.
- This platform must allow interactivity.
- It must be simple to use and easy to set up. Ideally, as close to existing systems as possible.
- It must be fast.
- It must take advantage of existing large ecosystems of statistical software.
- It must be robust.

- It must be flexible and extensible. A computational statistician may create custom classes and reasonably expect them to work well with the platform.

To this end, the use of the `R` programming language is a natural starting point. The means for writing software is typically through the use of a structured, high-level programming language. Of the myriad programming languages available, the most widespread language used for statistics is `R`. In August 2020, `R` reached its highest rank yet of 8th in the TIOBE index, a ranking of most popular programming languages, up from ranking 73rd in December 2008[6]. `R` also has a special relevance for this proposal, having been initially developed at the University of Auckland by Ross Ihaka and Robert Gentleman in 1991[15].

Major developments in contemporary statistical computing are typically published alongside an `R` code implementation, usually in the form of an `R` package, which is a mechanism for extending `R` and sharing functions. As of March 2021, the Comprehensive `R` Archive Network (CRAN) hosts over 17,000 available packages[28].

This project seeks to build and document the statistician's large-scale modelling platform in `R`. Preliminary results have been extremely encouraging to this endeavour, and are described in more detail in Section 4 below, having led to the creation of the **largeScaleR** package[5]. There remains plenty of future work, and this is described in Section 5, with tangible goals outlined in Section 6.

2 Background

There is more data, more data storage, far more powerful processing ability, and all are on an exponential path of growth, with standardly available RAM remaining far from capable of servicing the deluge of data made possible by all of this growth[36]. The problem of larger-than-memory datasets is certainly not diminishing.

In recent years, the use of the additional transistors in the CPU has been focussed not so much on increased clock speed as was the case pre-2003, instead granting a greater emphasis on multicore processing, for speed has met its limiting factors in the excessive heat produced and power consumed[36].

Software techniques are the best solution to the problem that hardware enables, as there is no alternative shy of investing in supercomputers, something far out of reach of most individuals and organisations. The software solution includes the programming in the little, to take advantage of greater hardware support for concurrency, as well as the design of systems that are architected in such a fashion so as to surmount constraints in hardware capacity. This project puts greater focus on the latter, as intelligent programming techniques can always be added to a well-designed system, but seldom *vice versa*.

The study of systems capable of handling larger-than-memory data extends back decades, and spans a very broad literature. Of utmost relevance to this project are modern systems with statistical capabilities. The approaches can be roughly divided into two main categories: *distributed* and *local*.

- Distributed systems spread the data and processing load across multiple computers. Though more complex than keeping everything on one computer, they can be faster and more capable when working with larger data, due to the greater parallelism afforded, as well as the larger pool of main memory available[12].

- Conversely, local systems, seeking to solve the problem of larger-than-memory data, make use of a single computer for data processing, taking advantage of the larger data storage capacity of disk, and often making use of parallel processing.

A common local solution is to just treat disk memory as an extension of RAM, as working memory. Some systems do make use of this approach, and will be examined below, but this is a non-starter for complex analyses on truly large data, as it is orders of magnitude slower than if the dataset were being held in RAM[1]. The key to a fast and efficient approach is to take advantage of parallelisation, for both processing and memory advantages.

To parallelise is to engage in many computations simultaneously—this typically takes the form of either task parallelism, wherein tasks are distributed across different processors; data parallelism, where the same task operates on different pieces of data across different processors; or some mix of the two[35]. Parallelism can afford major speedups, albeit with certain limitations. Amdahl's law and Gustafson's law are two relevent attempts to capture the limitations of such a form of computing, with both pointing to the limitations imposed by inherently serial (non-parallel) portions of a program, as well as the diminishing returns offered by additional CPU's made available to the system [2][14].

An example of an ideal task for parallelisation is the category of embarassingly parallel workload. Such a problem is one where the separation into parallel tasks is trivial, such as performing the same operation over a dataset independently[12]. Many problems in statistics fall into this category, such as tabulation, monte-carlo simulation and many matrix manipulation tasks. With 16 processors, the 160 million rows of the aforementioned flights dataset may have each processor simultaneously working on 10 million rows each, giving a potential order-of-magnitude speedup.

Some additional terminology is made use of in distributed systems. In networks, individual computers are referred to as *nodes*, and a distributed system will be said to take advantage of some number of nodes[18]. The pieces of data that are split up and distributed over the nodes are referred to as *chunks* or *shards*, among a variety of other names. A program reference to these chunks can take the form of a *distributed object*, which serves as an object-oriented interface to enable data manipulation at varying degrees of transparency[10].

The tension between choosing local systems for large-scale data, versus distributed systems typically lies in the slowdowns and opportunities introduced by complexity.

In favour of local systems for large data, the complexity is significantly lower. Everything is already in one location, with possibly shared memory, as well as the advantages of coscheduling allowing parallel streams to be automated. One famous demonstration of standard `UNIX` tools on a single machine found a 235-fold speedup in attaining simple summary statistics on a 3.5 GB dataset, over using a Hadoop cluster to perform the same[9]. The parallel stream approach common to a local system typically has a very minimal memory footprint, with all components of the system close enough together that data movement between RAM and disk is still fast enough. However, such an approach falls down when greater flexibility is required. For one-pass data manipulation it remains valid and indeed a preferable approach, but for complex analyses involving iteration and very large datasets that demand real-time interactivity, the data is far better-off in RAM, with RAM speed sitting orders of magnitude above that of even the fastest available SSD's[17][20].

This leads to the benefits offered by a distributed system. Among them, the fact that all working data can potentially be held in RAM across disparate computers can lead to major speedups, with the caveat being that data movement between machines should ideally be minimised in order to maintain high speed. For long-running processes, the slowdown from initial data movement may be rendered minor in comparison with the greater speeds gained through keeping and manipulating in-memory ("online") data[10]. The risk of a total-system crash is also mitigated to a greater extent with distributed systems, as the additional computers may be used for redundancy, allowing one computer to go down but the system to keep running—even with backups, this is not possible in a single-computer system.

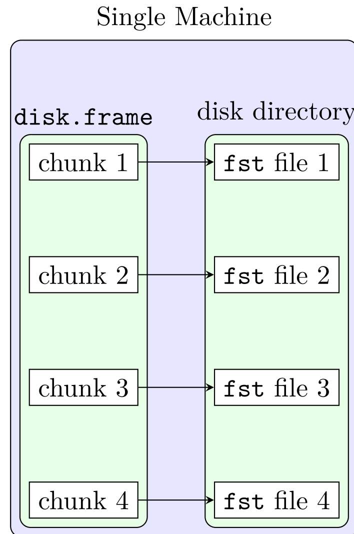

Figure 1: **disk.frame** architecture with four chunks

2.1 Local Systems

In the space of local solutions, offloading additional data to disk is a common solution. This is best illustrated through the **disk.frame** package[41]. An eponymously named dataframe replacement class is provided by **disk.frame**, which is able to represent a data set far larger than RAM, constrained only by disk size. The mechanism of action is to use chunks of data on disk, and provide a variety of methods taking advantage of data.frame generics, including **dplyr** and **data.table** functions. disk.frames are actually references to compressed files in a filesystem directory on disk, with each file serving as a chunk. Such an architecture is represented in Figure 1.

Operations are performed through manipulation of each chunk separately, loading a constrained number of chunks into RAM at a time, sparing the computer from dealing with a single monolithic file[42]. As mentioned above, this breaks down at scale, with the transfer of data from disk to RAM and back being far too slow for anything particularly big if used for anything more complex than single-pass statistics.

The chunking and loading strategy also finds its way into statistical models. An aspect of this stratedy is also offered by **disk.frame**, with linear modelling and generalised linear modelling functions calling the **biglm** package, which builds models a chunk at a time[21].

It is also worth noting that parallelism may be manifested within a single computer and works well for chunk processing. A number of `R` packages, **disk.frame** included, take advantage of various parallel strategies in order to process large datasets efficiently. One such package is **multicore**, now subsumed into the **parallel** package, that grants functions that can make direct use of multiprocessor systems, thereby reducing the processing time in proportionality to the number of processors available on the system[28].

2.2 Distributed Systems

At some stage however, the data gets too big, or analyses too complex, for one single computer. When this is the case, a variety of distributed system approaches have been put forward.

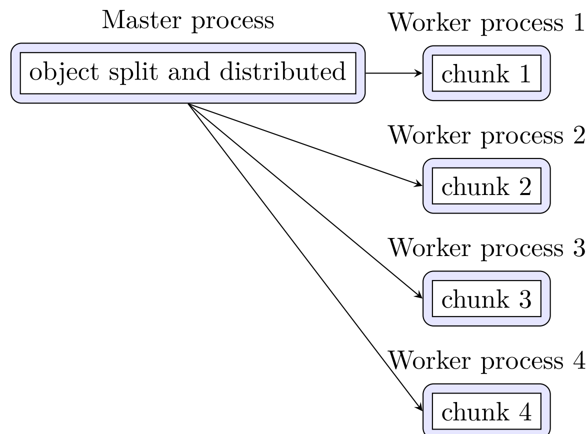

Figure 2: **SNOW** architecture with four chunks

This Section of the field is heavily dominated by major existing systems outside of `R`, so most current approaches serve as basic interfaces to the external system, complete with all of the expected abstraction leaks[34].

Of the few systems which are unique to `R`, the **SNOW** (Simple Network Of Workstations) package stands out. It composes part of the **parallel** package, which is contained in the standard `R` image[37]. Support for distributed computing over a simple network of computers is provided by **SNOW**. The general architecture of **SNOW** makes use of a master process that holds the data and launches the cluster, pushing the data to worker processes that operate upon it and return the results to the master. This is represented in Figure 2

Several different communications mechanisms are made use of by **SNOW**, including `ssh` and user-created sockets. It's greatest shortcoming is the lack of persistent data, and the mechanism of distribution employed disallows the usage of very large datasets.

The two dominant external-based packages in `R` revolve around the `MPI` system, and `Spark`.

One such package making heavy use of `MPI` is **pbdR**[27]. The **pbdR** (programming with big data in `R`) project provides persistent data, with the **pbdDMAT** (programming with big data Distributed MATrices) package offering a user-friendly distributed matrix class to program with over a distributed system[31]. This abstraction breaks down at a certain level of complexity, and a deep understanding of the powerful `MPI` system is eventually required; a task out of the league of most practicing statisticians.

The central interface to `Spark` in `R` is given through the **Sparklyr** package, which combines common **dplyr** methods with objects representing `Spark` dataframes[23]. Simple analyses are made very simple (assuming a well-configured and already running `Spark` instance), but custom iterative models are extremely difficult to create through the package in spite of **Spark's** support for it.

In the search for a distributed system for statistics, the world outside of `R` is not entirely barren. The central issue with non-R distributed systems is that their focus is very obviously not statistics, and this shows in the level of support the platforms provide for statistical purposes.

The `Hadoop` project provides a system predating and influencing `Spark`[33]. The project is a collection of utilities that facilitates cluster computing. Jobs can be sent for parallel processing on the cluster directly using `.jar` files, "streamed" using any executable file, or accessed through language-specific APIs. `Hadoop` consists of a file-store component, known as Hadoop Distributed File System (HDFS), and a processing component, known as MapReduce. The processing model

is powerful, though incapable of the rapid iteration required in complex models[40]. However, the filesystem is widely used and provides an effective interface to large datasets.

In `Python`, the closest match to a high-level distributed system that could have statistical application is given by the `Python` library **dask**[29]. Dynamic task scheduling is offered by **dask** through a central task graph, as well as a set of classes that encapsulate standard data manipulation structures such as NumPy arrays and Pandas dataframes. The main difference from the standard structures is that the **dask** classes take advantage of the task scheduling, including online persistence across multiple nodes. It is a large and mature library, catering to many use-cases, and exists largely in the Pythonic "Machine Learning" culture in comparison to the `R` "Statistics" culture. Accordingly, the focus is more tuned to the `Python` software developer putting existing ML models into a large-scale capacity. Of all the distributed systems assessed so far, **dask** comes the closest to what an ideal platform would look like for a statistician, but it misses out on the statistical ecosystem of `R`, provides only a few select classes, and is tightly bound to the concept of the task graph.

2.3 Evaluation

The task of modelling over larger-than-memory data has a rich history of attempts to provide a solution. For the requirements listed in Section 1, none of these come close to providing a satisfactory solution, though the python library `dask` may come the closest. The approaches put forward often do solve problems as defined otherwise, but all are either too cumbersome, incapable of handling large datasets, non-persistent, non-interactive, outside of the statistical ecosystem, excessively tied to their architecture, or all of the above. The **largeScaleR** package is already performing better than some of the surveyed approaches, and Section 4 will outline what has been demonstrated in the prototype.

3 Methodology

This project centres around the creation of a distributed system in `R`, serving as a platform for the implementation and execution of large-scale, multi-node statistical models. It will meet the statistician's demands given in Section 1, with further details given in Section 7. Several measures for the success of the project are proposed.

Principally, a test-driven development methodology is utilised, where unit and regression tests are created for the **largeScaleR** system, with the package continually run against these tests and developed in the direction of passing all tests. A code coverage of these tests nearing 100% is aimed at, as measured by the **covr** package. Memory usage and processing times are also profiled with every release, with the aim for every release of the system to run faster than the previous release on standard tests.

Real-world datasets are made use of and modelled in development, including the flights dataset referenced above, as well as the very large New York "taxicab" dataset[26].

Planned major features outlined in Section 5 have specific expected completion times, as laid out in Section 7, with their success being judged by a combination of automated tests and real-world usage, and summarised in feature-specific reports including in-depth comparisons to existing software projects with such features.

As the project matures and becomes feature-wise comparable to existing systems, benchmarking will take place on existing hardware, such as the Statistics department's *Ihaka* cluster. The principal systems for comparison are `Hadoop` and its `R` frontends, `Spark` through **Sparklyr**, and `dask`, among others. Real datasets, such as flights or taxicab, as well as realistic and complex

Function	Description
`start()`	Start the cluster, given some vector of hostnames.
`do.dcall()`	Perform some given function on a distributed object.
`read.dcsv()`	Read the appropriate pieces of a `csv` file that has been scattered along the cluster prior and return the distributed object reference.
`distribute()`	Split up and scatter the chunks of some local object across the cluster, returning the distributed object reference to the chunks.
`emerge()`	Copy a remote chunk or distributed object to the local user, and re-combine if needed.
`preview()`	Print a small snapshot of the distributed object, including potential errors.
`split()` and `combine()`	Methods that when defined, grant a class access to automatic `distribute()` and `emerge()` functionality.
`worker()`	Used by the `start()` function to initialise a worker.
`final()`	Automatically run at the end of an `R` session to shut down the cluster.
`alignment()`	Used in the worker evaluation process to align distributed objects of disparate sizes in order to implement recycling

Table 1: Top 10 most important functions in the **largeScaleR** package

models, such as Random Forest, or a Generalised Linear Model, will be used to test the systems in detail.

All referenced milestones and features are accompanied by formal reports, with the more significant to be submitted for publishing in *arXiv*, *the R Journal*, *JSS*, or similar.

Further work will continue from the existing prototype described in Section 4.

The project is fully open-source. Progress on written PhD work and experimentation is freely accessible from the `jcai849/phd` GitHub repository, and work on the **largeScaleR** package is available from the `jcai849/largeScaleR` GitHub repository[5].

4 Preliminary Results

To overcome the problem of large-scale statistical analysis in `R`, what is needed is a platform that is fast and robust, with a focus on a simple interface for fitting statistical models, and the flexibility for implementation of arbitrary new models within `R`. As of May 2021, a prototype distributed system holding many of the described desired characteristics, has been implemented in `R` as part of this research. This system is tentatively named **largeScaleR**, and takes the form of an `R` package, complete with minor documentation and a moderate proportion of tests. It has been used to successfully read and manipulate data over a cluster of multiple nodes, including multiple processes on each node, as well as non-trivial distributed manipulations such as tabling of dataframes, all operating at a very high speed of operation. As it currently stands, the package comprises around 230 individual functions. A table of the 10 most important functions, and their descriptions, is given in Table 1.

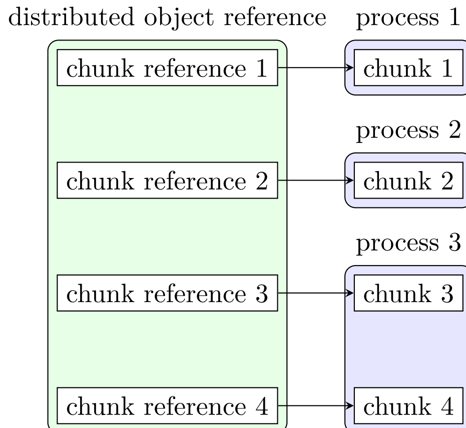

Figure 3: Example distributed object reference relations

4.1 Overview of largeScaleR

The system operates in a very loosely coupled manner between processes, each able to send requests indirectly to any other, equally functional as a peer-to-peer network as a master-worker setup. A user process, treated as something akin to a master process, is the typical initator in requests. A master process runs as a regular `R` session, operated interactively by the user or by batch script. This master process can then initialise other processes, sitting on any node, to perform work. These secondary processes are dubbed "worker processes".

The worker processes are entirely independent of the master process, and none of these processes contain any information to identify other processes. The only mechanism these processes have to communicate is via a communication queue, which serves as the primary mechanism behind the operation of the main conceptual pieces interacted with by a user: distributed objects.

Distributed objects are a means of access to chunks of objects on a distributed system[10]. They serve as a reference that acts as a transparent handle to fragmented referents (*chunks*) over a distributed system, with each chunk being a portion of data residing on some worker process.

To take a concrete example, consider the flights dataset as described above. Assume that the dataset is split into four pieces row-wise, with the first two pieces residing on two separate processes, and the last two on a single process. Such a topology is capable of being represented in the **largeScaleR** system, with diagrammatic representation as in Figure 3.

Distributed object references are effectively proxies, with generic methods passing on their standard form to the constituent chunks of the distributed object, returning another distributed object as reference to the return value of the methods acting on the chunks. The returned distributed object is given immediately, with worker processing occuring asynchronously, giving lazy, future-like, behaviour to distributed objects[3].

Each chunk has a "descriptor"—some unique name that exclusively references that chunk. When they are not performing operations on a chunk, workers are monitoring all of the queues whose names correspond to the descriptors of the chunks which the respective worker holds. Actions to be performed on the chunks are transmitted through these queues.

The master process enacts requests on these queues through methods on the distributed objects being intercepted and sent as possibly modified messages to their referent chunk queues, where they are then operated upon by the worker process. Key to the flexibility of **largeScaleR** is that the queue serves as a level of indirection, so the requesting process doesn't need to know

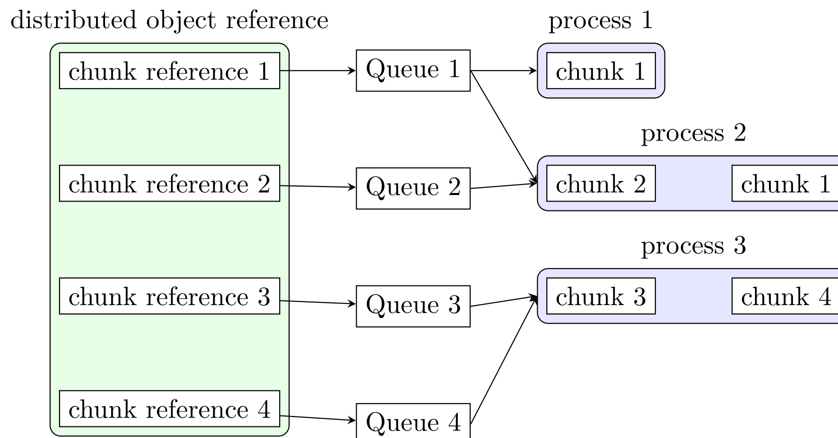

Figure 4: Example distributed object communication relations with redundancy example in chunk 1

precisely where a chunk is stored, only that it can be reached via its queue. This flexibility, mirroring the benefits of information hiding encouraged by message-passing object-oriented programming, allows chunks to be held arbitrarily, including on multiple nodes simultaneously. The capacity for redundancy grants future potential for fault tolerance and resilience to nodes crashing. Figure 4 depicts this additional detail, continuing from the example of the flights dataset, with an additional process holding a redundant first chunk.

A major supporting component of the system's distributed architecture is the act of "emerging", wherein a reference is used to pull all of its referents locally[10]. This takes place through directly sending serialised chunks to the requester, where methods exist to combine them.

Multivariate manipulations of the data make use of emerging on the worker end, where multiple distributed objects are referenced in one single function request on a queue, and the worker must determine the appropriate alignment of chunks, including the use of `R`'s recycling rules, before emerging all distributed objects and performing the operation.

Distributed objects stand-in for regular `R` objects, and can represent any class that has split and combine methods defined. These include all atomic vectors, lists, dataframes, matrices, and arbitrary user-created classes.

Another key aspect to the architecture of the system is detailed logging, with all changes of state in a node recorded and the information dispatched to a central logger, which allows monitoring of the system in one location. The collection of logs is sufficient to build a complete picture of the system, with a Model-View-Controller pattern in an external program able to parse the logs, calculate system state, and display that in a simple interface[13].

4.2 largeScaleR System Usage Patterns

An exceedingly important consideration for the user is the manner in which the program is interfaced with.

As mentioned above, the **largeScaleR** program is distributed as an `R` package, and initial setup follows standard package protocols.

The system starts through getting a cluster running. It is assumed that the hardware and network is already set up. If the **largeScaleR** cluster is already running, the master can just connect directly, with some descriptive functions entered by the user allowing it to connect. The

cluster can be initialised entirely by the master session, through the use of functions taking a simple description of the intended cluster. For ease of use, this can be given through a programmable configuration file describing the nature of the network, including addresses and specialised services such as a communication server and log server, as well as descriptions of the master and all the worker processes.

Upon successfully running the cluster, all processes involved will log any changes in state, including which chunks are held by which workers, and this can be viewed in an included interface.

Data is initialised in the system through several different pathways. The most straightforward for the user is to take existing data in an `R` session, and run a package-provided method on the data to distribute and form a reference to it. This serves to distribute the data as chunks across a number of worker processes, commonly referred to as "scattering" in `MPI` parlance[39]. This is a similar interface to that of the **SNOW** package[37]. While good for medium-sized data and demonstration purposes, it is unrealistic in that by definition, very large data is unable to fit in the memory of the master `R` session for it to be sent out in the first place.

Therefore a more standard method of initialisation when data originates from local disk is to use a package-provided reading function that streams raw data from a `csv` file or similar from disk through a root communications queue and into all workers. This is acknowledged to be inefficient, but it currently works well under all tests when data is not already distributed. Using the flights dataset as an example, this method reads in the dataset a limited number of rows at a time, and propogates each read as a chunk to some node via the queue system. Each chunk may be limited by size as well, so that the 160 million rows may be scattered into 100 pieces of 1.6 million rows each, distributed across some arbitrary number of machines.

The fastest method of data initialisation follows the creation of a character vector Listing either `URL`'s or files local to each of the workers. This is then distributed to the workers and an appropriate read operation is pushed to them via the distributed character vector. This is the only method enabling full parallelisation, and it is general enough to be extended for access to distributed filesystems such as `HDFS`.

Alongside distribution of the data, the user is returned a distributed object to use for referencing the distributed data. In the case of the third method described, this occurs near-instantaneously, with the data being read concurrently.

The data referenced by distributed objects can be sent from workers more simply; once established, an `emerge()` method is run over a distributed object, and the underlying chunks are sent directly from the worker, to be combined at the master end. Such movement is taken advantage of by workers as well, when they are faced with operations on multiple disparate distributed objects—this is hidden from the user, however.

The benefits of distributed objects grow commensurately with their degree of transparency, and **largeScaleR** has transparency as a central goal. Many common functions have methods provided operating on distributed objects, including most `Group` methods such as `Ops`, `Math`, and `Summary`. More complex methods such as `table()` and `rbind()` are also given, and for very simple analyses, these are often enough to serve as the backbone of the analysis until the data is summarised sufficiently that an `emerge()` can be performed and more complex analyses run locally.

Alternatively, an extra layer of control is granted to the user looking for more than pre-formed functions: functionality inspired by the `do.call()` function in `R` allows passing anonymous or existing functions, along with a list of distributed and potentially local data, and the provided functions are run over the referent data pointed to by the distributed objects. A distributed object referring to the results is returned. This is actually how most of the transparent methods were implemented, with the distributed `do.call()` serving as a wrapped intermediary. Such a technique is equivalent to a Map, with a reduce also possible through either reducing at the

```
> init("config")

> cols <- c("Year"="integer","Month"="integer","DayofMonth"="integer",
+          "DayOfWeek"="integer","DepTime"="integer","CRSDepTime"="integer",
+          "Ar ..." ... [TRUNCATED]

> flights <- read(localCSV("/tmp/1987flights.csv", header=TRUE, colTypes = cols),
                  max.size=1024^2)
```

Listing 1: Initial input to the distributed system

worker end, in parallel, or even followed by an `emerge()` and local reduction[24].

4.3 Example Session

For the purposes of demonstration, an example session with the prototype **largeScaleR** package is given. The data is a subset of the flights dataset, including the latter months of the 1987 flights. The object of investigation in this simple analysis is the variation in cancellation of Monday flights along different months.

The cluster is started as in Listing 1, with an `init()` function, which reads a configuration file describing the cluster. The flights data is then read and distributed with column type descriptions, at a small chunk size for the purpose of demonstrating a large number of chunk references.

Upon successful reading, this dataset exists as a distributed object reference in the user session, serving as a proxy object pointing to the many chunks distributed across processes and hosts. Printing the `flights` object reveals the number of references as 122, as shown in Listing 2. A `preview()` of the distributed object can be taken, which pulls a portion of the first chunk for viewing. Standard functions such as `nrow()` behave as expected on the distributed object. “Meta” functions that give information on the distribution of the object can be run as well, including information on descriptors with `desc()` and the hostnames of where each chunk is stored with `host()`.

To attain a table of Monday flights, standard `R` methods of data manipulation can be used, shown in Listing 3. Creating a logical vector and using it to subset the originating data, a mechanism known as a Boolean mask, is commonly performed in `R`, and enabled entirely transparently in **largeScaleR**. The creation of such a vector in distributed fashion involves a significant amount of work behind the scenes, including the distribution of the comparative operator (`1L` in this case), communications regarding the intended function to be performed, and achieving the appropriate alignment of the comparative operator for recycling. To demonstrate functionality, the `sum()` of the distributed logical vector is taken and should be found to be the same as the `length()` of the subset resulting from the distributed logical vector. These vectors, like all distributed objects, can be previewed, regardless of underlying class. Finally, a `table()` method exists to tabulate the chunks of the distributed vectors in parallel, and combine them locally. This can then be used for further analysis, such as in a χ^2-test or similar.

4.4 Issues Encountered

The initial development of **largeScaleR** has been highly experimental, with the current offering being the third total rewrite. While the development process has been flexible enough to accomodate this, persistent issues inherent in the field have been repeatedly appearing.

```
> print(flights)
Distributed Object Reference with 122 chunk references.
First chunk reference:
Chunk Reference with Descriptor 1
> preview(flights)
  Year Month DayofMonth DayOfWeek DepTime CRSDepTime ArrTime CRSArrTime
1 1987    10         14         3     741        730     912        849
2 1987    10         15         4     729        730     903        849
3 1987    10         17         6     741        730     918        849
4 1987    10         18         7     729        730     847        849
5 1987    10         19         1     749        730     922        849
6 1987    10         21         3     728        730     848        849
  UniqueCarrier FlightNum TailNum ActualElapsedTime CRSElapsedTime AirTime
1            PS      1451      NA                91             79      NA
2            PS      1451      NA                94             79      NA
3            PS      1451      NA                97             79      NA
4            PS      1451      NA                78             79      NA
5            PS      1451      NA                93             79      NA
6            PS      1451      NA                80             79      NA
  ArrDelay DepDelay Origin Dest Distance TaxiIn TaxiOut Cancelled
1       23       11    SAN  SFO      447     NA      NA         0
2       14       -1    SAN  SFO      447     NA      NA         0
3       29       11    SAN  SFO      447     NA      NA         0
4       -2       -1    SAN  SFO      447     NA      NA         0
5       33       19    SAN  SFO      447     NA      NA         0
6       -1       -2    SAN  SFO      447     NA      NA         0
  CancellationCode Diverted CarrierDelay WeatherDelay NASDelay SecurityDelay
1              NA        0           NA           NA       NA            NA
2              NA        0           NA           NA       NA            NA
3              NA        0           NA           NA       NA            NA
4              NA        0           NA           NA       NA            NA
5              NA        0           NA           NA       NA            NA
6              NA        0           NA           NA       NA            NA
  LateAircraftDelay
1                NA
2                NA
3                NA
4                NA
5                NA
6                NA
> nrow(flights)
[1] 1311826
> desc(flights)
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19   ...
> host(flights)
  [1] "127.0.0.2" "127.0.0.2" "127.0.0.2" "127.0.0.2" "127.0.0.2" "127.0.0.2"
  [7] "127.0.0.3" ...
```

Listing 2: Exploration of the structure of the flights dataset

```
> isMondayFlights <- flights$DayOfWeek == 1L

> print(isMondayFlights)
Distributed Object Reference with 122 chunk references.
First chunk reference:
Chunk Reference with Descriptor 246

> preview(isMondayFlights)
[1] FALSE FALSE FALSE FALSE  TRUE FALSE

> sum(isMondayFlights)
[1] 190711

> mondayFlights <- subset(flights, isMondayFlights)

> length(mondayFlights)
[1] 190711

> cancelledMondays <- table(mondayFlights$Month, mondayFlights$Cancelled)

> print(cancelledMondays)

         0     1
  10 58573   670
  11 72283   774
  12 55320  3091
```

Listing 3: Dataset manipulation to attain final table

Communication is one such issue. Communication between processes in **largeScaleR** uses `redis` queues, with blocking pop operations to read from them. An `S3` "message" class was defined in **largeScaleR** to standardise communication between nodes and is the only accepted form of message to be placed and parsed from a queue. This has proven to be a complex model, with the internal handling of queues having major effects on the manner of communication taking place in the system. Alternative communication protocols have the same associated issue, and there has not been one single obvious communications system that grants great implementation-independence, though `redis` surely comes the closest[7].

Following acts of communication, the evaluation of a message has its own complexities, with the following example given as the current implementation with two distributed objects:

1. The distributed objects first go through a complex act of alignment.

2. They are all compared with the target chunk, and aligned accordingly.

3. If the beginning and end of the target chunk are completely outside those of the offered object, the indices of the object corresponding to those at the correct corresponding multiple are taken, and this object is then emerged. In this way, recycling is implemented in a distributed fashion, with each worker determining the appropriate recycle.

4. Once all appropriate chunks are emerged, a regular `do.call` is run with the function and now-local objects

For alignment to take place, metadata associated with each chunk, such as its corresponding beginning and end indices, as well as its deduced size, must be associated with a distributed object before it is made use of. As there is purposefully no mechanism for responders to communicate with requesters, an alternative mechanism was devised; metadata requests operate just as any other distributed function call, using the standard distributed `do.call` interface, but the functions sent are unique to the chunk they are associated with, using metalinguistic evaluation to create a function that when evaluated on the worker end, sends the metadata information to a unique temporary queue, which is then listened to at the requester end, popped, and returned. The infinite possible forms of this implementation, each with their own unique efficiencies and drawbacks, has been a source of continuous research, with data structure alignment being a research project in its own right[4][19].

Other issues include the difficulty of debugging and testing a parallel/distributed system, as existing `R` tools are set up for serial code evaluation, without taking into consideration complexities such as the non-deterministic nature of communication between computers of varying speeds and loads. Race conditions, that is, behaviour dependent on timing, also become a problem where for example, a computer blocks and awaits results from another computer, which in turn only returns results based on some future input from the original, which would never arrive. Such conditions are very difficult to determine even in languages that specifically provide a framework for testing race conditions, as in most langauages with threading support, and are even harder to debug, test for, and eliminate in the open-ended system being created[32]. Synchronisation of the system is another related problem cropping up, again serving as a research project in itself. So far, the best results gained have been to deny the possibility of synchronisation, and engineer the processes comprising the system to be independent of each other as much as practicable.

5 Future Work

To improve upon the preliminary results and other systems, there are several problem-areas that are planned to be worked upon.

For it to function as well as any other distributed system, **largeScaleR** must exhibit fault tolerance. This is needed in every large distributed system, as each machine that comprises it may fail with some probability p. A system with n machines will fail with probability $1-(1-p)^n$, and the system will almost certainly fail as $n \to \infty$, for any $0 < p \leq 1$ as may be expected in reality. Thus redundancy and tolerance of machine faults is essential at scale. This is possible to implement in **largeScaleR** without any major rewrite, as it has been architected with fault tolerance in mind.

More efficient memory usage will serve to improve the efficiency of the system. With memory being the motivating constraint, improvements on software usage of it translate to a more efficient system in the large. As the system currently stands, there is some intelligent caching being performed, but it can stand to have at least half of the current memory footprint, particularly through further work on supporting packages, which this project can contribute to. Such external contributions serve to aid not only the **largeScaleR** package, but the state of computational statistics and the open source community in general.

Interfacing with other systems is another important feature that will require more work. `HDFS` is one example among many filesystems, such as `EFS`, which are already widely used in the sphere of big data, and present a useful opportunity to provide a native interface. Hadoop may also be interfaced with in MapReduce jobs, whether for populating data or serving as a portion of processing—this remains to be explored in detail, and is certain to offer a far higher ease-of-use to those with data already within these systems.

The creation of a more user-friendly and informative monitoring system will also serve as a great boon to the usability of the package.

The implementation of a variety of models, serving equally as proof-of-concept and for providing direction to the project, will be a major priority. Existing modelling systems that have flexibility for parallelisation or streaming can be taken advantage of, with examples being the **biglm** package as well as more experimental work such as that produced by sampling and one-step polishing[21][22].

All of these features are meaningless if they are not demonstrated and published, and to that end, the goal of publishing the **largeScaleR** package on CRAN has been set. This will be co-ordinated with articles relating to testing the package at a large scale. Such an article or technical report may include benchmarking a novel analysis on the platform with real-world data at a scale of 64+ nodes and 100+ GB data source size.

6 Objectives & Goals

The objective of this research project is to create a platfom for large-scale statistical computing, utilising the versatility and power of `R`. Such a platform will allow statisticians to easily define and run complex distributed algorithms from within the `R` environment, rather than having to rely on external tools that never had statistical computation as a goal. This platform will be demonstrated through the implementation of iterative models in `R`, and applying these models to real-world tasks on large-scale problems.

The following tasks will be undertaken as a part of the proposed research:

1. Development of further proposed platform features, including:
    - (a) Fault tolerance.
    - (b) Efficient memory usage through transient data and command chaining.
    - (c) Interfaces with external systems such as `Hadoop`.

2. The demonstration of such a platform through implementing complex iterative statistical models on larger-than-memory datasets, including a Generalised Linear Model and variations thereof, as well as some multivariate techniques such as PCA or cluster analysis.
3. Publishing the **largeScaleR** package on `CRAN`. This requires additional development including:
   (a) Stability as may be expected from 90+% test coverage.
   (b) Full documentation for all base functions.
   (c) A package vignette.
   (d) Passing `CRAN` checks.
4. Contributing to supporting packages.
5. Publishing a technical report on the Stats blog, or arXiv.
6. Publishing an article in an international, refereed journal, such as JSS, the R Journal, or similar.
7. Presenting on the platform at useR!, or similar R developer conferences.
8. Benchmarking a novel analysis on the platform with real-world data at a scale of 64+ nodes and 100+ GB data source size.

7 Deliverables and Program Schedule

All provisional goals have been completed, with the goal of proposal approval being contingent on the committee. A description of the goals and statements on their completion follows:

1. Approval of the full thesis proposal by the appropriate departmental/faculty postgraduate committee. This will be granted contingent on the reception of this proposal.
2. A substantial piece of written work, such as a literature review, completed to the satisfaction of the main supervisor. A 15,000 word literature review document has been produced, at a draft stage that is completed to the satisfaction of the main supervisor. This can be found in the `phd` git repository under `doc/lit-review.tex`.
3. Ethics approval/s and/or permissions obtained for the research (if required). No ethics approval or permissions are required.
4. Attendance at one of the Doctoral Skills Programme Induction Days. This was completed on 2020-05-29.
5. Successful completion of the Academic Integrity Module. This was completed during undergraduate study.
6. A needs analysis to determine training and other requirements that must be completed before candidature can be confirmed. This was completed, with the document presented to the PYR committee.
7. Completion of a health and safety risk assessment and training for any laboratory/studio/field and related work activities. No health and safety training was required for the type of work needed for this project.

Date	Speaker	Title
2020-19	Malia Puloka	Posing Investigative Questions about Categorical Data—a Year 9 Case Study
2020-07-22	Yifu Tang	Likelihood Approximations for Time Series and Calibration of Approximate Bayesian Credible Sets
2020-07-29	Luke Boyle	Understanding Surgical Outcomes in New Zealand
2020-11-12	Andrew Holbrook	Bayes in the Time of Big Data
2020-11-24	Richard Perry	Modelling for COVID in Official Economic Time Series
2020-11-25	Rolf Turner	A Versatile Discrete Distribution
2020-12-02	Innocenter Amina	Integrative Analysis of High-Dimensional Data with Application to Soil Microbiome Data
2021-02-25	Charco Hui	Natural Language Processing in Clinical Trials
2021-03-31	Zehua Zang	Branching with Decision Detection

Table 2: Talks attended as part of first year goals

8. Undertake Diagnostic English Language Needs Assessment (DELNA) online screening. If a full assessment is advised, complete full diagnostic test and participate in any language enrichment recommended by the DELNA Language Advisor. This was completed during undergraduate study.

9. Write a review of existing methods and literature on approaches to distributed computing with R and current solutions in other language/systems with similar goals for statistical modelling like Python or Spark, to the sa tisfaction of the primary supervisor. This was completed and can be found in the `phd` git repository under `doc/survey*`.

10. Implement a prototype `R` software capable of performing operations on multiple chunks of data in parallel on different machines and use the prototype to implement one statistical model, to the satisfaction of the primary supervisor. This has been satisfied through the development of the **largeScaleR** package.

11. Attend at least 10 relevant research presentations per annum (student needs to verify participation by filling out and handing in the departmental attendance form for each presentation to the Statistics Department office . This has been completed, with the talks attended summarised in table 2, attendance at PhD talks day counting for double, and hard copy forms with further details available at the Statistics Department Office.

12. Participate in the Department of Statistics PhD Talks Day and/or give a departmental seminar, to the satisfaction of the main supervisor. Also, maintain a personal profile page (www.directory.auckland.ac.nz), providing information on scholarly activities and objectives to the satisfaction of the main supervisor and a Department of Statistics PhD Officer. This proposal will accompany a departmental seminar, and the personal profile page can be located at `https://directory.auckland.ac.nz/people/profile/jcai849`.

13. Attendance at one of the Faculty of Science Doctoral Induction Workshops. This was completed on 2020-09-16.

An approximate timeline of projected future work completion is given in Table 3.

Date	Event
2021-06	Submitting a technical report for publication in the *Stats Tech Blog*, or *arXiv*
2021-07	Presenting on the platform at *UseR!* or a similar developer conference
2021-12	Platform feature development
2022-02	Publishing package on CRAN
2022-03	Submitting an article relating to the research for publication in the *R Journal*, *JSS*, *JCGS*, or similar
2022-05	Development of complex statistical model demonstration for package
2022-09	Benchmarking at scale
2023-05	Thesis submission

Table 3: A timeline of future work

8 Budget

The development of the project itself, revolving around open-source software, does not come with any budgeting demands. However, the field of research is rapidly moving, and requires conference attendance and presentations in order to maintain relevance. This has been budgeted at $1000.00 per annum, with the funds to be derived from the Postgraduate Research Student Support (PReSS) account, on an as-needed basis. This is referenced in the Doctoral Provisional Year Review document, and is less than the budget cap of $1200.00 for the Statistics Department.

References

[1] Bowen Alpern et al. “The uniform memory hierarchy model of computation”. In: *Algorithmica* 12.2 (1994), pp. 72–109.

[2] Gene M Amdahl. “Validity of the single processor approach to achieving large scale computing capabilities”. In: *Proceedings of the April 18-20, 1967, spring joint computer conference.* 1967, pp. 483–485.

[3] Henry C Baker Jr and Carl Hewitt. “The incremental garbage collection of processes”. In: *ACM SIGART Bulletin* 64 (1977), pp. 55–59.

[4] Randal Bryant. *Computer Systems GE.* Melbourne: P. Ed Australia, 2015. ISBN: 9781488672071.

[5] Jason Cairns. *largeScaleR: Provides a Distributed Framework for Statistical Modelling.* R package version 0.2. 2020. URL: `https://github.com/jcai849/distObj`.

[6] TIOBE - the software quality company. *R | TIOBE - The Software Quality Company.* Apr. 14, 2021. URL: `https://www.tiobe.com/tiobe-index/r/` (visited on 04/14/2021).

[7] Maxwell Dayvson Da Silva and Hugo Lopes Tavares. *Redis Essentials.* Packt Publishing Ltd, 2015.

[8] Peter J Denning. “Thrashing: Its causes and prevention”. In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I.* 1968, pp. 915–922.

[9] Adam Drake. *Command-line Tools can be 235x Faster than your Hadoop Cluster*. Jan. 18, 2014. URL: `https://adamdrake.com/command-line-tools-can-be-235x-faster-than-your-hadoop-cluster.html` (visited on 04/14/2021).

[10] Wolfgang Emmerich. *Engineering distributed objects.* Chichester New York: John Wiley & Sons, 2000.

[11] Robert E Fontana Jr and Gary M Decad. “Moore’s law realities for recording systems and memory storage components: HDD, tape, NAND, and optical”. In: *AIP Advances* 8.5 (2018), p. 056506.

[12] Ian Foster. *Designing and building parallel programs : concepts and tools for parallel software engineering*. Reading, Mass: Addison-Wesley, 1995. ISBN: 9780201575941.

[13] Erich Gamma. *Design patterns : elements of reusable object-oriented software*. Reading, Mass: Addison-Wesley, 1995. ISBN: 0201633612.

[14] John L Gustafson. “Reevaluating Amdahl’s law”. In: *Communications of the ACM* 31.5 (1988), pp. 532–533.

[15] Ross Ihaka and Robert Gentleman. “R: a language for data analysis and graphics”. In: *Journal of computational and graphical statistics* 5.3 (1996), pp. 299–314.

[16] Cornell IT. *Standard Computer Build Specifications*. URL: `https://it.cornell.edu/citsg-cit-intranet/standard-computer-build-specifications` (visited on 01/20/2021).

[17] Dongkyun Kim et al. “23.2 a 1.1 V 1ynm 6.4 Gb/s/pin 16Gb DDR5 SDRAM with a Phase-Rotator-Based DLL, high-speed SerDes and RX/TX equalization scheme”. In: *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE. 2019, pp. 380–382.

[18] Martin Kleppmann. *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. " O’Reilly Media, Inc.", 2017.

[19] Jingke Li and Marina Chen. “The data alignment phase in compiling programs for distributed-memory machines”. In: *Journal of parallel and distributed computing* 13.2 (1991), pp. 213–221.

[20] Samsung Electronics Co. Ltd. *Samsung V-NAND SSD 980 PRO 2020 Data Sheet*. 2020. URL: `https://s3.ap-northeast-2.amazonaws.com/global.semi.static/SSD_980_PRO_Brochure.pdf`.

[21] Thomas Lumley. *biglm: bounded memory linear and generalized linear models*. R package version 0.9-1. 2013. URL: `https://CRAN.R-project.org/package=biglm`.

[22] Thomas Lumley. “Fast Generalized Linear Models by Database Sampling and One-Step Polishing”. In: *Journal of Computational and Graphical Statistics* 28.4 (2019), pp. 1007–1010. DOI: `10.1080/10618600.2019.1610312`. eprint: `https://doi.org/10.1080/10618600.2019.1610312`. URL: `https://doi.org/10.1080/10618600.2019.1610312`.

[23] Javier Luraschi et al. *sparklyr: R Interface to Apache Spark*. R package version 1.1.0. 2020. URL: `https://CRAN.R-project.org/package=sparklyr`.

[24] Michael McCool. *Structured parallel programming : patterns for efficient computation*. Amsterdam Boston: Elsevier/Morgan Kaufmann, 2012. ISBN: 9780124159938.

[25] Gordon E Moore et al. “Progress in digital integrated electronics”. In: *Electron devices meeting*. Vol. 21. Maryland, USA. 1975, pp. 11–13.

[26] NYC Taxi & Limousine Commision. *TLC Trip Records*. 2021. URL: `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`.

[27] G. Ostrouchov et al. *Programming with Big Data in R*. 2012. URL: `http://r-pbd.org/`.

[28] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2020. URL: `https://www.R-project.org/`.

[29] Matthew Rocklin. “Dask: Parallel computation with blocked algorithms and task scheduling”. In: *Proceedings of the 14th python in science conference*. 130-136. Citeseer. 2015.

[30] David Reinsel–John Gantz–John Rydning. “The digitization of the world from edge to core”. In: *Framingham: International Data Corporation* (2018).

[31] Drew Schmidt et al. *pbdDMAT: pbdR Distributed Matrix Methods.* R package version 0.5-1. 2020. URL: https://cran.r-project.org/package=pbdDMAT.

[32] Konstantin Serebryany and Timur Iskhodzhanov. “ThreadSanitizer: Data race detection in practice”. In: *Proceedings of the workshop on binary instrumentation and applications.* 2009, pp. 62–71.

[33] Konstantin Shvachko et al. “The hadoop distributed file system”. In: *2010 IEEE 26th symposium on mass storage systems and technologies (MSST).* IEEE. 2010, pp. 1–10.

[34] Joel Spolsky. *The Law of Leaky Abstractions.* Nov. 11, 2002. URL: https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/ (visited on 04/14/2021).

[35] Jaspal Subhlok et al. “Exploiting task and data parallelism on a multicomputer”. In: *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming.* 1993, pp. 13–22.

[36] Herb Sutter. “The free lunch is over: A fundamental turn toward concurrency in software”. In: *Dr. Dobbs journal* 30.3 (2005), pp. 202–210.

[37] Luke Tierney et al. *snow: Simple Network of Workstations.* R package version 0.4-3. 2018. URL: https://CRAN.R-project.org/package=snow.

[38] USA States Bureau of Transportation. *Data Expo 2009: Airline on time data.* Version V1. 2008. DOI: 10.7910/DVN/HG7NV7. URL: https://doi.org/10.7910/DVN/HG7NV7.

[39] David W Walker and Jack J Dongarra. “MPI: a standard message passing interface”. In: *Supercomputer* 12 (1996), pp. 56–68.

[40] Matei Zaharia et al. “Spark: Cluster computing with working sets.” In: *HotCloud* 10.10-10 (2010), p. 95.

[41] Dai ZJ. *disk.frame: Larger-than-RAM Disk-Based Data Manipulation Framework.* R package version 0.3.4. 2020. URL: https://CRAN.R-project.org/package=disk.frame.

[42] Dai ZJ. *Ingesting Data disk.frame.* Exploiting the Structure of a disk.frame. 2019. URL: https://diskframe.com/articles/04-ingesting-data.html#exploiting-the-structure-of-a-disk-frame (visited on 04/01/2020).