Prototyping a Dataflow Implementation of the CARTA System

CADaFloP Project Proposal

Dylan Fouché*
Department of Computer Science
University of Cape Town
fchdyl001@myuct.ac.za

Zainab Adjiet*
Department of Computer Science
University of Cape Town
adjzai001@myuct.ac.za

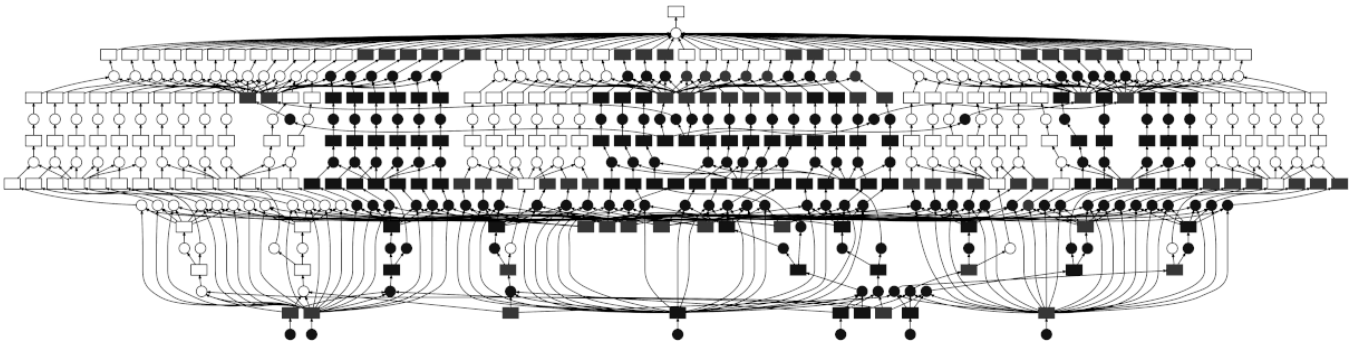


Figure 1: Distributed and parallel computing with Dask [8]

CCS CONCEPTS

• **Computer systems organization** → **Parallel architectures**; **Distributed architectures**; **Data flow architectures**; *High-level language architectures*; • **Software and its engineering** → *Massively parallel systems*; *Distributed systems organizing principles*; *Software prototyping*; Scheduling; • **Information systems** → *Enterprise information systems.*

KEYWORDS

dataflow, exascale, architecture, Python, visualisation, astronomy

1 PROJECT DESCRIPTION

The Cube Analysis and Rendering Tool for Astronomy (CARTA) [7] is a tool designed to visualise and analyse data from the Atacama Large Millimetre Array (ALMA) [2], the National Radio Astronomy Observatory (NRAO) [25], and the Square Kilometre Array (SKA) [14]. As the image size of these modern telescopes has been increasing rapidly over the past years, visualisation and analysis have become very computationally expensive tasks and CARTA aims to provide the needed scalability by utilising modern web technologies and computing parallelisation.

With parallelism and scalability in mind, the current CARTA back-end [9] is implemented as a multi-threaded imperative program written in C++ [5], a compiled, widely-regarded highly performant language. CARTA [9] uses several libraries, a key one being the Common Astronomy Software Applications (CASA) [23] library, which is maintained by a CARTA partner, the National Radio Astronomy Observatory (NRAO) [25], and predominantly implemented in C++ with an IPython interface.

However, the design of computational models and software architecture is shifting as we move into the exascale era, where computers are required to perform a quintillion calculations per second to cope with the increasingly large data sets that they are expected to process. With High-Performance Computing (HPC) approaching this exascale era, the NRAO is completely rewriting CASA and considering the use of a dataflow model [13] implemented in an interpreted language, Python, with the use of the Python Dask library.

Jack Dennis [16] presented the first concept of the data-flow architecture in 1974. This architecture differs from the traditional von Neumann architecture in that a program counter is not used to govern program flow. Instead, a *firing rule* specifies when instructions can execute, and this is based on the availability of the instruction input data. Furthermore, the architecture does not allow for global state storage, which eliminates side-effects [1].

Dask [11] is a flexible Python library for parallel programming, which comprises dynamic task scheduling and parallel collections like arrays and lists for larger-than-memory environments. Computations are structured as directed acyclic graphs called task graphs, and these task graphs can be run with different schedulers. The parallel collections are then built atop this infrastructure and based on their single-threaded counterparts.

Inspired by the work being done by the NRAO, we investigate a novel implementation of the CARTA back-end system with the Python-based Dask dataflow environment through an architectural re-design of the current system and the implementation of a set of prototype components.

*All authors contributed equally to this proposal

2 PROBLEM STATEMENT

2.1 Research Hypothesis

This project aims to test the feasibility of implementing the CARTA back-end functionality in a dataflow environment, and to provide aid towards concluding if it is a viable solution to adapt to the exascale way of computing.

Many astronomical computing libraries are being re-written to handle exascale throughput for image processing and analysis, but this does not have the same real-time requirements as interactive visualisation tools such as CARTA, and there exists no clear guidance on how best to scale out workloads that include this real-time interaction.

Consequently, this work will investigate the implications of using a dataflow model for the CARTA back-end. The research hypothesis is that it is possible to adapt existing dataflow tools to handle exascale-throughput interactive visualisation workloads.

2.2 Project Objectives

A comprehensive set of documentation will be created to describe the enterprise-grade systems architecture that would need to be implemented should this model be rolled out at scale. This document set comprises a systems engineering management plan, a set of use cases and requirements, and component-level architectural design specifications.

Furthermore, a set of back-end components will be prototyped in the Python language [31] using the Dask library [11, 12] for advanced parallelism and distributed computation. Other libraries such as AstroPy [4, 26] and H5Py [10] will be used to abstract away from domain-specific implementation details.

This new dataflow model will be compared with the current imperative implementation, and a set of recommendations will be made. The implications of this are broader than one implementation, however, as we aim to assess the validity of the dataflow model using interpreted languages for various exascale-throughput interactive scientific computations and visualisations.

3 RELATED WORKS

The move towards exascale computation has opened up new possibilities in the scientific community, yet new design paradigms are needed to leverage the processing power of these new computers. Shalf et al. [28] recognise that since it is mainly an increase in the number of processing cores that is driving the increase in processing power today, heterogeneity and concurrency are becoming increasingly integral in our modern computing infrastructure. Modern software architecture must be implicitly parallel and increasingly resilient to traditional software errors [6].

The dataflow model of computation is one such architecture that may satisfy this. Culler [13] originally describes this model of computation as a "machine language for parallel machines", but it has since been adapted as a software design pattern not dissimilar to the pipe and filter model. Representing functions as nodes on a directed acyclic graph and values as tokens that travel asynchronously along the edges of that graph, we are able to schedule computation over large heterogeneous compute clusters easily.

The dataflow model offers efficient use of fine-grain parallelism for computation [21]. It offers implicit parallel task synchronisation, and does not constrain instruction sequencing apart from enforcing data dependency constraints. It also tolerates memory latency as it processes other instructions while waiting for a response from memory, and thus has this advantage over the von Neumann control-flow model as well [13]. Dataflow programs can be composed effortlessly to form more extensive programs by connecting the output of one graph to the input of another [15, 29], which presents a way of dividing a program into distinct components.

Verdosica et al. [32] suggest in their position paper that the dataflow model is indeed a valid approach for exascale computation, and many successful implementations of this nature have been identified. Silva et al. [30] demonstrate the efficiency of this approach for analysing large raw-data files over distributed systems, which is a primary use case for the CARTA system. Mao et al. [22] successfully implemented a dataflow model for distributed scheduling of jobs to process the exascale throughput from the Square Kilometre Array. Zhang et al. [33] have demonstrated the vast scalability of this approach using the Amazon Web Services EC2 cloud compute infrastructure.

The Dask library [11] provides a dataflow environment for Python and while it offers simple, powerful tools for parallel programming, many similar tools exist. Mehta et al. [24] have evaluated how various Python libraries perform when completing tasks on large-scale image analysis systems. Dask is shown to have more scheduling overhead than other Python libraries, Spark [3] and Myria [27], when processing smaller, partitioned data sets. However, Dask does outperform its competitors with a faster runtime on larger data sets which is attributed to its more efficient pipelining and data caching capabilities after overcoming the initial start-up overhead as seen with the smaller data sets.

4 RESEARCH METHODS

4.1 Full Back-end System Design

The feasibility of a dataflow model [13] for the CARTA back-end system will be evaluated by re-designing the back-end to follow a dataflow model, opposed to the traditional von Neumann model that it is currently based on.

The existing system will be analysed to collate the functional and non-functional requirements of the new system. This will involve examining various sequence diagrams for different operations of the back-end system made available to us, as well as examining the existing back-end codebase. The gathered requirements will then be summarised in the form of numerous user stories. These will be reviewed occasionally with the project supervisor and advisers to ensure the user stories encapsulate all the use cases the current back-end satisfies. This will also ensure that the non-functional requirements of the system such as expected performance, scalability and maintenance are also encapsulated by the user stories.

Various UML diagrams will be constructed to describe the structure of the new system, starting with the system overview and narrowing down to the classes and objects required to provide the functionality. The operation of the system will then be explained using behavioural UML diagrams. We will seek assistance from the project advisers on how to best construct these diagrams so they

can be thorough and well-understood. The diagrams intended to serve this purpose include, but are not limited to:

- Package diagram
- Class diagrams
- Use case diagrams
- Sequence diagrams

The system design will be regularly evaluated against the user story requirements as a measure of progress and success, and the design will be adjusted accordingly if need be. The system behaviour will be evaluated against the existing back-end operation performance to measure if the new system provides any notable improvements.

4.2 Prototype Component Implementation

To evaluate the research hypothesis, a number of components from the CARTA back-end [9] will be prototyped in Python using a dataflow framework. This framework is the Dask [11] library and will be complemented with a set of popular libraries for scientific computing [4, 10, 26]. The objective here is not to provide a production back-end system to replace the current implementation, but rather to demonstrate the efficacy of the dataflow approach for this problem through the use of several component prototypes.

The set of components to be prototyped will be chosen from the most computationally expensive procedures implemented in the back-end, such that a meaningful set of conclusions can be drawn regarding performance and scalability. These may include, but are not limited to and are subject to change:

- Histogramming
- Profiling
- Gaussian smoothing
- Computing region statistics
- Swizzling
- Contouring

The current CARTA back-end will be used for testing. This can be accessed programmatically with a Python-based scripting interface [18]. The purpose of these tests is to determine (i) that the output of the prototype components is correct and (ii) if the prototype components outperform the existing implementation. The former will be answered by comparing the outputs of the existing and new implementations, while the latter will be answered by timing each implementation under various conditions and computing precisely the performance difference.

This testing data will comprise randomly generated data as well as real images sourced from the public domain, ranging from very small images to larger-than-memory images. To assess the scalability of the prototype components, the performance testing will evaluate several deployment environments ranging from parallelism on a single machine to distributed computation over a high-performance cluster.

The prototyped components need not integrate with the current CARTA front-end, as this would involve implementing a heavyweight message protocol that exceeds the scale of this project. Instead, demonstrations will be done with the use of Jupyter notebook [20]. This allows code to be demonstrated in real-time with accompanying explanation, and libraries such as matplotlib [17] allow for inline data visualisation. It is important to note that these visualisations are intended as a sanity check rather than an end feature of these components.

Through thorough testing and evaluation of these components, we will be able to assess the validity and performance of this model for the primary use cases of the CARTA system.

5 EXPECTED PROJECT OUTCOMES

5.1 Project Impact

There has been much work showing the ability of the distributed dataflow model to handle exascale throughput. What is not clear, however, is the ability of this model to incorporate interactivity, visualisation, and elements of computational steering into its workflow. Additionally, there have been many successful implementations of this model but not any holistic review of the impact of this model for a broader problem domain.

For CARTA, this project will provide a proof of concept for a new dataflow implementation of the back-end system. Prototype components will provide a starting point around which a new full back-end implementation can be built. The systems architecture will guide this new implementation with design recommendations as well as structural and behavioural documentation.

The success of this implementation could be used to advise related work in different domains, especially for those systems that share similar computational requirements. This problem is not unique to the astronomical domain, and there may be many scientific workflows that could benefit from a robust, distributed architecture to handle large-scale interactive workloads.

5.2 Testing and Evaluation

The system design will be evaluated by how well the proposed specification meets the functional and non-functional requirements as stated in the user stories which will first be approved by the project supervisor. The system structure will be compared to the existing back-end codebase to ensure it clearly follows a different computer model from the existing one and that the proposed system can perform the same operations. To test if the proposed system follows a dataflow model specifically, the system will be compared to existing dataflow systems and will also undergo reviews by the project supervisor and advisers to ensure this.

The success of the dataflow model applied to the system can be further supported by systems engineering benefits that it may provide to the system and its users. These benefits can include system scalability and the ability for system components to be easily composed into larger systems or replaced by similar components. These aspects can prove valuable for future development on the CARTA dataflow back-end.

The success criteria for the prototype component implementation is as follows:

(1) There exists a set of prototype components that mimics the behaviour of certain CARTA back-end functionality using Dask.
(2) The behaviour of the prototype components is shown to be correct.

(3) There exists precise metrics on the performance difference between the prototype components and the current implementation.
(4) There exists a visualisation environment that demonstrates these prototype components and their performance, with sufficient explanation.

Note that this success does not depend on the new components outperforming the current implementation, as reporting a precise performance deficit would be no less valuable than reporting a precise performance gain.

6 ETHICAL, PROFESSIONAL AND LEGAL ISSUES

Since no testing or evaluation will be performed on human subjects, there is no need to obtain ethical clearance for this project. All astronomical imagery used for testing and evaluation will be sourced from the public domain and thus raises no legal implications. All research conducted will be made publicly available, and all software developed will be open-source, released under the GNU General Public License (GPLv3). The authors report no potential conflict of interest.

7 PROJECT PLAN

7.1 Project Milestones and Deliverables

The project will run from 13 May 2020 to 19 October 2020. Over this period, the following list of deliverables will need to be completed:

- Project proposal
- Python prototype code
- Data-flow system design documentation
- Project final paper draft
- Project revised final paper
- Project demonstration
- Project website
- Project poster

The full project plan for completing these deliverables can be seen in the Gantt chart in appendix A.

7.2 Required Resources

Various advisers are needed to ensure that positive progress is made on the project at all times. These include the project supervisor, Rob Simmonds, to guide the aim and direction of the project to the team as well as project advisers, Kechil Kirkham and Adrianna Pinska, to provide guidance on the respective design and implementation deliverables.

All team members will use their personal laptops or computers to complete the required deliverables. A remote development environment with access to all the relevant libraries and the CARTA back end system will be required to develop and test the prototype components. A high-performance cluster of remote virtual machines on the ilifu [19] cloud will be required to properly assess the performance of Dask's distributed functionality.

7.3 Anticipated Risks

The possible risks for the duration of this project and their respective management strategies are outlined in the risk analysis matrix in appendix B. The probability of occurrence ranges from Rare (1) to Common (5), and the impact of the occurrence on the project ranges from Insignificant (1) to Catastrophic (5). While some are fairly likely to occur, they do not pose a major threat to the progression of the project but merely a delay. Furthermore, the few risks which could be catastrophic are improbable to occur.

7.4 Division of Work

Two aspects of the project have been described previously, namely the system design section and the prototype component implementation section. Zainab Adjiet will work on the system design while Dylan Fouché will work on the prototype component implementation.

While the two sections are closely related, the prototype development work is not implementing the architecture defined in the design documentation. Rather, this design documentation will describe the architecture of a production system that could be developed around a successful set of prototype components. Thus, the two sections can be worked on in parallel and evaluated independently.

ACKNOWLEDGMENTS

This work is based on the research supported wholly or in part by the National Research Foundation of South Africa (grant number MND190716456261).

REFERENCES

[1] Tilak Agerwala et al. 1982. Data Flow Systems: Guest Editors' Introduction. *Computer* 15, 2 (1982), 10–13.
[2] A Worldwide Collaboration ALMA. 2020. Atacama Large Millimeter/submillimeter Array. https://www.almaobservatory.org/en/home/
[3] Apache. 2020. *Welcome to Spark Python API Docs!* http://spark.apache.org/docs/latest/api/python/index.html
[4] Astropy Collaboration, T. P. Robitaille, E. J. Tollerud, P. Greenfield, M. Droettboom, E. Bray, T. Aldcroft, M. Davis, A. Ginsburg, A. M. Price-Whelan, W. E. Kerzendorf, A. Conley, N. Crighton, K. Barbary, D. Muna, H. Ferguson, F. Grollier, M. M. Parikh, P. H. Nair, H. M. Unther, C. Deil, J. Woillez, S. Conseil, R. Kramer, J. E. H. Turner, L. Singer, R. Fox, B. A. Weaver, V. Zabalza, Z. I. Edwards, K. Azalee Bostroem, D. J. Burke, A. R. Casey, S. M. Crawford, N. Dencheva, J. Ely, T. Jenness, K. Labrie, P. L. Lim, F. Pierfederici, A. Pontzen, A. Ptak, B. Refsdal, M. Servillat, and O. Streicher. 2013. Astropy: A community Python package for astronomy. 558, Article A33 (Oct. 2013), A33 pages. https://doi.org/10.1051/0004-6361/201322068 arXiv:astro-ph.IM/1307.6212
[5] British Standards Institute. 2003. *The C++ Standard: incorporating Technical Corrigendum 1: BS ISO* (second ed.). Wiley, New York, NY, USA. xxxiv + 782 pages.
[6] Franck Cappello, Al Geist, Bill Gropp, Laxmikant Kale, Bill Kramer, and Marc Snir. 2009. Toward exascale resilience. *The International Journal of High Performance Computing Applications* 23, 4 (2009), 374–388.
[7] CARTA. 2018. *Cube Analysis and Rendering Tool for Astronomy.* https://carta.readthedocs.io/en/latest/index.html
[8] Anaconda Cloud. 2020. Distributed and parallel computing with Dask: Memory and Parallelism. http://matthewrocklin.com/slides/images/grid_search_schedule.gif
[9] CARTAvis community. 2020. *CARTA Backend.* https://github.com/CARTAvis/carta-backend/
[10] Andrew Collette & contributors. 2020. *HDF5 for Python.* https://www.h5py.org
[11] Dask core developers. 2019. *Dask: Natively scales Python.* https://dask.org
[12] James Crist. 2016. Dask & Numba: Simple libraries for optimizing scientific python code. In *2016 IEEE International Conference on Big Data (Big Data).* IEEE, 2342–2343.
[13] David E Culler. 1986. Dataflow architectures. *Annual review of computer science* 1, 1 (1986), 225–253.
[14] David Davidson. 2012. MeerKAT and SKA phase 1. *2012 10th International Symposium on Antennas, Propagation and EM Theory, ISAPE 2012*, 1279–1282. https://doi.org/10.1109/ISAPE.2012.6409014
[15] Alan L Davis and Robert M Keller. 1982. Data flow program graphs. (1982).

[16] Jack B Dennis. 1974. First version of a data flow procedure language. In *Programming Symposium*. Springer, 362–376.

[17] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

[18] idia. 2020. carta-python. https://github.com/idia-astro/carta-python

[19] ilifu. [n.d.]. Cloud computing for data-intensive research. http://www.ilifu.ac.za/

[20] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks-a publishing format for reproducible computational workflows.. In *ELPUB*. 87–90.

[21] Ben Lee and Ali R Hurson. 1994. Dataflow architectures and multithreading. *Computer* 27, 8 (1994), 27–39.

[22] Yishu Mao, Yongxin Zhu, Tian Huang, Han Song, and Qixuan Xue. 2016. DAG Constrained Scheduling Prototype for an Astronomy Exa-Scale HPC Application. In *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 631–638.

[23] J. P. McMullin, B. Waters, D. Schiebel, W. Young, and K. Golap. 2007. *CASA Architecture and Applications*. Astronomical Society of the Pacific Conference Series, Vol. 376. 127.

[24] Parmita Mehta, Sven Dorkenwald, Dongfang Zhao, Tomer Kaftan, Alvin Cheung, Magdalena Balazinska, Ariel Rokem, Andrew Connolly, Jacob Vanderplas, and Yusra AlSayyad. 2016. Comparative evaluation of big-data systems on scientific image analytics workloads. *arXiv preprint arXiv:1612.02485* (2016).

[25] National Radio Astronomy Observatory. 2020. *National Radio Astronomy Observatory*. https://public.nrao.edu/

[26] A. M. Price-Whelan, B. M. Sipőcz, H. M. Günther, P. L. Lim, S. M. Crawford, S. Conseil, D. L. Shupe, M. W. Craig, N. Dencheva, A. Ginsburg, J. T. VanderPlas, L. D. Bradley, D. Pérez-Suárez, M. de Val-Borro, (Primary Paper Contributors, T. L. Aldcroft, K. L. Cruz, T. P. Robitaille, E. J. Tollerud, (Astropy Coordination Committee, C. Ardelean, T. Babej, Y. P. Bach, M. Bachetti, A. V. Bakanov, S. P. Bamford, G. Barentsen, P. Barmby, A. Baumbach, K. L. Berry, F. Biscani, M. Boquien, K. A. Bostroem, L. G. Bouma, G. B. Brammer, E. M. Bray, H. Breytenbach, H. Buddelmeijer, D. J. Burke, G. Calderone, J. L. Cano Rodríguez, M. Cara, J. V. M. Cardoso, S. Cheedella, Y. Copin, L. Corrales, D. Crichton, D. D'Avella, C. Deil, É. Depagne, J. P. Dietrich, A. Donath, M. Droettboom, N. Earl, T. Erben, S. Fabbro, L. A. Ferreira, T. Finethy, R. T. Fox, L. H. Garrison, S. L. J. Gibbons, D. A. Goldstein, R. Gommers, J. P. Greco, P. Greenfield, A. M. Groener, F. Grollier, A. Hagen, P. Hirst, D. Homeier, A. J. Horton, G. Hosseinzadeh, L. Hu, J. S. Hunkeler, Ž. Ivezić, A. Jain, T. Jenness, G. Kanarek, S. Kendrew, N. S. Kern, W. E. Kerzendorf, A. Khvalko, J. King, D. Kirkby, A. M. Kulkarni, Astropy Contributors, et al. 2018. The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. 156, Article 123 (Sept. 2018), 123 pages. https://doi.org/10.3847/1538-3881/aabc4f

[27] PyPI. 2020. *myria-python*. https://pypi.org/project/myria-python/

[28] John Shalf, Sudip Dosanjh, and John Morrison. 2011. Exascale Computing Technology Challenges. In *High Performance Computing for Computational Science – VECPAR 2010*, José M. Laginha M. Palma, Michel Daydé, Osni Marques, and João Correia Lopes (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–25.

[29] Jurij Silc, Borut Robic, and Theo Ungerer. 1998. Asynchrony in parallel computing: From dataflow to multithreading. *Parallel and Distributed Computing Practices* 1, 1 (1998), 3–30.

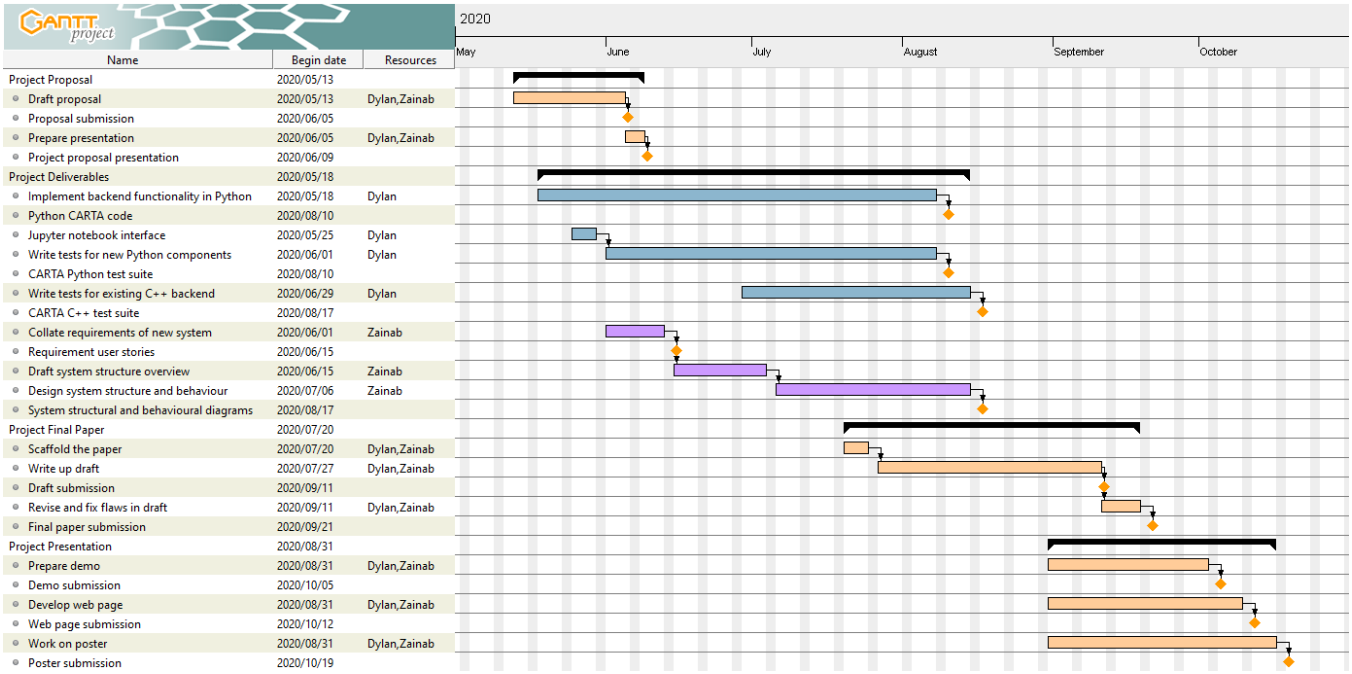
[30] Vitor Silva, Daniel de Oliveira, and Marta Mattoso. 2014. Exploratory analysis of raw data files through dataflows. In *2014 International Symposium on Computer Architecture and High Performance Computing Workshop*. IEEE, 114–119.

[31] G. van Rossum. 1995. *Python tutorial*. Technical Report CS-R9526. Centrum voor Wiskunde en Informatica (CWI), Amsterdam.

[32] Lorenzo Verdoscia and Roberto Vaccaro. 2013. Position paper: Validity of the static dataflow approach for exascale computing challenges. In *2013 Data-Flow Execution Models for Extreme Scale Computing*. IEEE, 38–41.

[33] Zhao Zhang, Kyle Barbary, Frank Austin Nothaft, Evan Sparks, Oliver Zahn, Michael J Franklin, David A Patterson, and Saul Perlmutter. 2015. Scientific computing meets big data technology: An astronomy use case. In *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 918–927.

A PROJECT GANTT CHART



B RISK ANALYSIS MATRIX

Risk	Probability	Impact	Management Strategy
Team member falls ill, physically unable to contribute to the project or leaves project team	2	4	Ensure there is an even workload amongst team members so work can be more easily redistributed
Project supervisor is unable to guide the team or is unhelpful	1	3	Ensure the direction of the project is clarified earlier in the project and use guidance from project advisors well
Project advisors are unable to guide the team or are unhelpful	1	3	Accept this risk and if it occurs ask project supervisor if they can refer a different source of guidance
Team member unable to contribute to project due to external issue such as laptop malfunction or no internet connectivity	3	4	Have another means of working on the project and have backup mobile data to ensure connectivity
Git repository on Github becomes corrupted	1	5	Keep local storage of code and documentation
Python Dask library becomes outdated or obsolete during the project	1	4	Accept this risk and if it occurs, discuss with project supervisor on a new way forward with project aim
Requirements for the full back-end design change frequently as the project progresses	2	4	Regularly check requirements with supervisor
Requirements are drafted incorrectly rendering the subsequent design inadequate	4	2	Regularly evaluate incremental design against requirements captured and make changes in necessary
Unable to properly test Dask.distributed on a cluster of virtual machines	3	3	Follow up with supervisor regularly to ensure this can be set up in due time, otherwise accept this risk