

Autosubmit GUI: A ReactJS-based Graphical User Interface to Monitor Experiments in a High Performance Computing Environment

Wilmer V. Uruchi Ticona¹, Miguel Castrillo¹, and Daniel Beltrán¹

¹ Barcelona Supercomputing Center

Summary

Autosubmit GUI is a front-end software developed using Javascript and ReactJS that aims to show complex information from the execution of the workflow of experiments (managed by Autosubmit) in a *High Performance Computing (HPC)*. Autosubmit is a Python-based workflow manager that handles complex tasks involving different sub steps (e.g. scientific computational experiments) which may be executed in one or different computing systems (platforms), from *High Performance Computers* to small clusters or workstations. This workflow manager is able to orchestrate the tasks (jobs) that constitute the workflow while respecting their dependencies and handling errors.

This front-end software consumes information served by an API (Autosubmit API) that collects information from the execution of the workflow of experiments. An experiment consists of jobs that are executed on the selected platform following an established sequence. The execution of an experiment and its jobs generates a high amount of information that needs to be visualized later by users. Autosubmit API summarizes this information and presents it as API requests. Autosubmit GUI consumes these API requests and shows the information available in a condensed, comprehensive, and dynamic way. Autosubmit GUI uses three highly popular and useful libraries: FancyTree, vis.js, and react-google-charts; among other web resources that facilitate the visualization of information and information updates.

Statement of need

The number of jobs in an experiment workflow (managed by Autosubmit) ranges from one job to several thousands, where the jobs can vary in the kind of task they perform, from data retrieval to complex climate simulation. The result of the execution of an experiment workflow is usually large amounts of information scattered in a file system. Originally, this information was accessed through the file system or by using terminal commands (provided by Autosubmit) to get the information as a text, pdf file, or terminal output. The *Computational Earth Science* team decided that a better way to access this information without risking unreliability nor loss of performance was needed. A web application was chosen as the solution architecture that would enable us to deploy this service easily and with minimum requirements for our users. Furthermore, this choice would allow developers to reuse existing technology, in the form of open-source visualization libraries, to allow Autosubmit GUI to show information in a graphical way. The software has been designed and is developed following an efficiency first premise, since the amount of information we need to show is considerable and we aim to maintain minor loading times.

The main experiment workflow representation generated by the users of Autosubmit was a graph generated through graphviz in a pdf file. Consequently, that was the first representation we tried to mimic in Autosubmit GUI. For this purpose we chose vis.js, a proven javascript library that provides graph representations for the provided data. This library also provides a

DOI: [DOIunavailable](#)

Software

- [Review](#) ↗
- [Repository](#) ↗
- [Archive](#) ↗

Editor: [Pending Editor](#) ↗

Reviewers:

- [@Pending Reviewers](#)

Submitted: N/A

Published: N/A

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

rich API through which we can manipulate it and dynamically update it. In our implementation we present several ways to access the `vis.js` objects and adapt them to our necessities in concordance with the requirements of *ReactJS*, we avoid unnecessary refreshes of the graph component by storing it in the React internal state. We believe that this can serve as a guide to those willing to work with graph representations in a web environment.

Later, Autosubmit GUI was required to implement an experiment workflow representation that showed more information at first glance, and that arranged the jobs in a hierarchical way. As a result, we decided to implement a tree view based on *FancyTree*, a Javascript library. This library provides a rich API that allowed us to implement a structured visualization of the experiment workflow. Again, we adapted this library to work smoothly with *ReactJS* and that is updated efficiently through the internal component infrastructure. It is also a good example on how to use the tools provided by this library. Moreover, statistics are shown using *react-google-charts*.

Finally, a word on our programming language and framework choice. *Javascript* was the obvious, and perhaps required, choice in this respect. This programming language has taken over the web and is becoming a standard, for better or worse. For our front-end purpose, it has all the tools that we needed and gives us access to a variety of useful extensions. The main vision of Autosubmit GUI was to develop a graphical interface that showed as much information as possible at first glance; moreover, this interface should be scalable, with more requirements of information taking an area of the browser window. It was decided that the interface should be divided in modules that would be displayed in a tiled design. The interface should be able to update the modules (tiles) individually and independently without affecting general performance. *ReactJS* provides the tools to accomplish this purpose, where we can translate our idea of modules to components. So, it was the chosen framework.

Main features

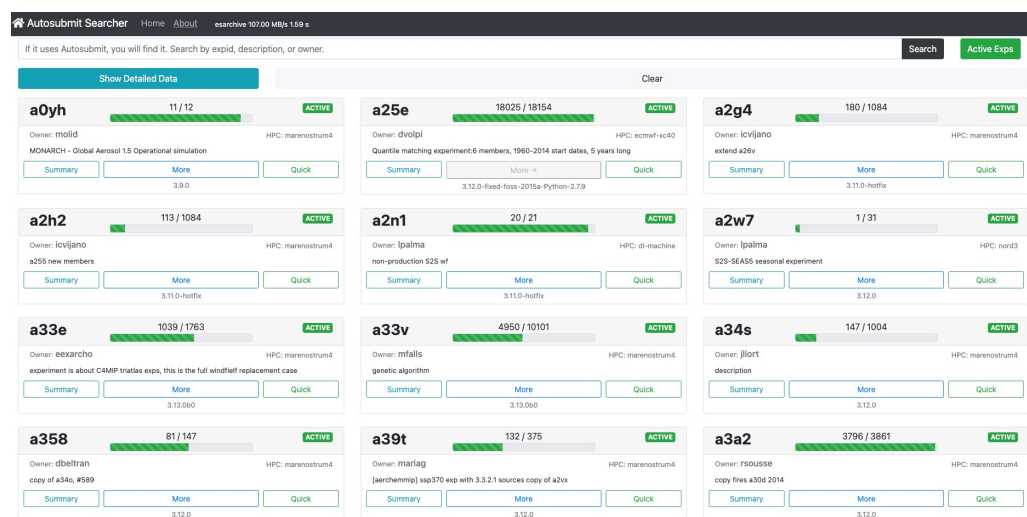


Figure 1: Autosubmit GUI main window.

- Users can search for experiments that match certain criteria: Currently active experiments (those being executed), experiments belonging to a certain user, description of experiments that contain certain word, and other filters. The result is presented in a main window [Figure 1](#) where the experiment progress is displayed for each result along other relevant information. Each experiment item in the result gives direct access to the experiment representation (tree view by default) or a quick view which displays only

the most important information. There is an option, from the experiment item, to see only a summary of the current progress of the experiment.

- User can see the representation of their experiment in three ways with some variations:
 - Tree View [Figure 2](#): Uses FancyTree to present the experiment jobs in a structured and hierarchical fashion.
 - Graph View [Figure 3](#): Uses vis.js to present the experiment jobs in a graph.
 - Quick View: Uses FancyTree to present a browsable list of the experiment jobs.

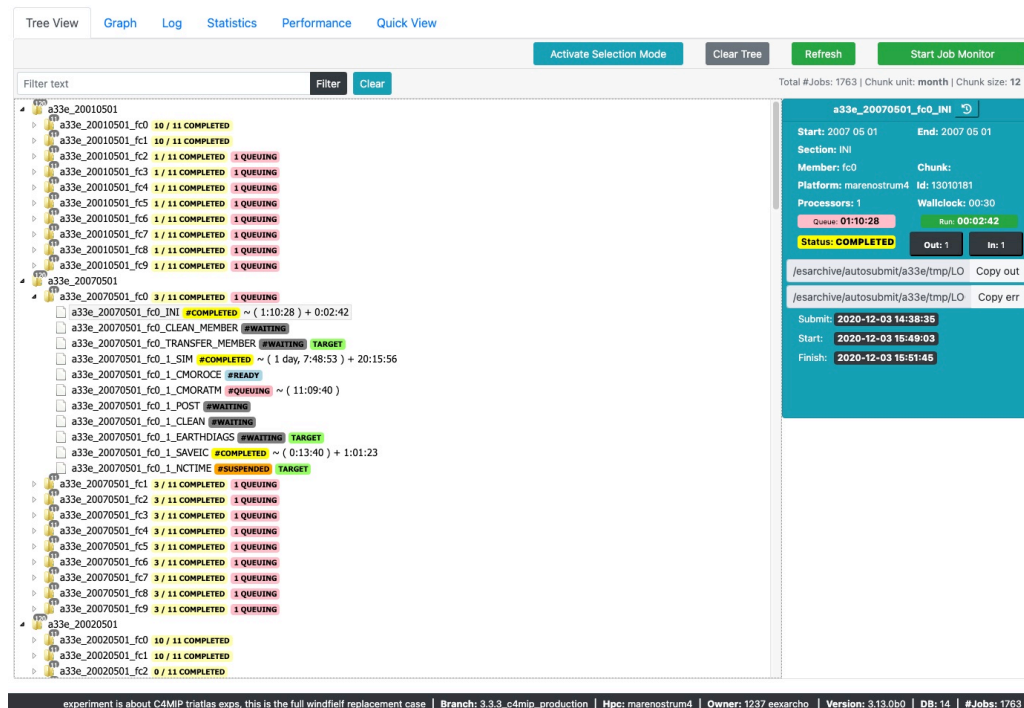


Figure 2: Tree View.

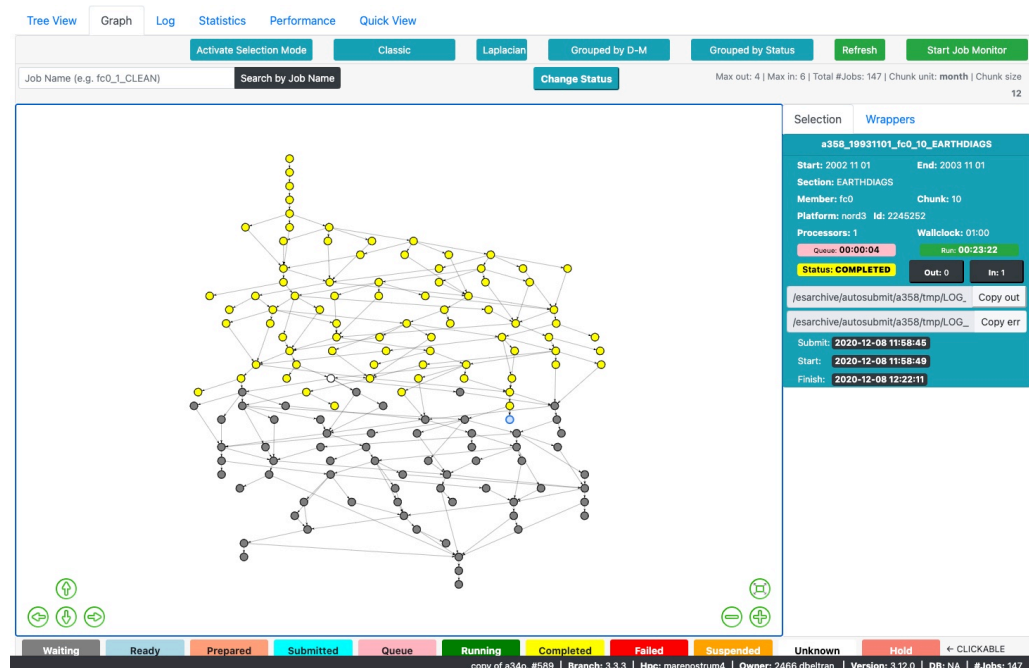


Figure 3: Graph View.

- All experiment representations allow search of jobs by name, or by patterns in the name using the * wildcard.
- The graph view allows the user to select some nodes (jobs) and generate a command through the Change Status button.
- The graph view presents variations of the representation for ease of usage.
- The graph and tree views allow the user to see detailed information of a job by clicking on the node or item.
- The tree view allows the user to select some items (jobs) and generate a command through the Activate Selection Mode button.
- ACTIVE experiments (shown at the top of the page next to the experiment identifier) can update the representation (graph or tree) according to changes in real time through the Start Job Monitor option. This functionality doesn't prompt a full component rerender, instead, the nodes or items are update individually and independently.
- The last lines of the log of the experiment can be visualized.
- Statistics of the workflow execution are provided through the Statistics tab using react-google-charts.
- Performance metrics of the experiment are displayed in the Performance tab.
- Historical data is displayed for individual jobs through a button next to the job's name in the selection tile.

Citations

If you want to cite a software repository URL (e.g. something on GitHub without a preferred citation) then you can do it with the example BibTeX entry below for Smith (2020).

For a quick reference, the following citation commands can be used.

Acknowledgements

We acknowledge contributions from Francisco Doblas-Reyes, Kim Serradell and all our Earth Science users.

References

No referencess

Smith, A. (2020). Fidgit: An ungodly union of GitHub and figshare. In *GitHub repository*. GitHub. <https://github.com/arfon/fidgit>