Mii: An Automated Search Engine for Environment Modules

JUSTIN STANLEY, Iowa State University

The use of environment modules in HPC systems is rapidly gaining traction due to the increasing complexity of managing research software. While environment modules improve organization for administrators, they can be unfriendly to end-users, especially those less familiar in UNIX-like shell environments. I created a search engine for modules that allows users to query available modules by the names of the commands they provide. Furthermore, I wrote integrations for the popular shells `bash` and `zsh` that completely automate the searching and the loading process.

CCS Concepts: • **Human-centered computing** → *Command line interfaces*.

Additional Key Words and Phrases: HPC, modules, accessibility

ACM Reference Format:
Justin Stanley. 2019. Mii: An Automated Search Engine for Environment Modules. 1, 1 (April 2019), 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Environment modules allow users to dynamically load and unload software into their working environment. Often, a module will introduce a new command into the user's shell. This is accomplished by modifying the `PATH` environment variable that is scanned by the operating system when the user tries to execute a command. However, the module rarely contains any metadata about *which* commands it will make accessible to the user once it is loaded. As a result, if the user needs to execute a certain command but does not know the name of the module which provides it, they must contact the system administrator to receive the module name. This is a less-than-optimal user experience and could be improved by tracking more relevant information about modules.

Making module environments more friendly and interactive will save both users and administrators much time and frustration. Users will struggle less at the command line and administrators can dedicate their time to more important matters.

To store more information about modules, each available module file must be analyzed. Modules can affect the environment in several ways. It can be difficult to predict how changes to environment variables will affect the execution of programs. Also, loading modules can be quite expensive; often a module will depend on many others, forming a tree of modules which all must be loaded first. This can cause loading individual modules to take several seconds, making loading every available module infeasible in a timely manner. As a

Author's address: Justin Stanley, jtst@iastate.edu, Iowa State University, Ames, Iowa.

Manuscript submitted to ACM

result, the indexing tool must analyze the properties and effects of module files without actually loading them.

While the concept of environment modules has been around since 1991 [1], it is still steadily gaining popularity in HPC systems. User issues with modules are relatively easy for an administrator to solve; the user likely knows the name of the tools they are looking for and the administrator only needs to inform them of the module name which provides it.

Mii constructs a local database of modules and their behaviors. The *Mii* engine walks through the user's `MODULEPATH` and finds module files. The module analyzer then infers the type of the module file and tries to extract potential modifications to the `PATH` variable. *Mii* then traverses through all of the potential `PATH`s and looks for programs that the user is able to execute. Then, for each program found a row is inserted into the local database associating the module file with the command it provides. Once the database is constructed, *Mii* is able to search for modules by command names.

1.1 Primary features and benefits

Once *Mii* is installed for a user it will:

- Construct an index from binary names to module names
- Hook into the user shell, automatically load modules as necessary
- Provide user with help and suggested commands

2 PRELIMINARIES

There are currently two major implementations of environment modules. *Environment Modules* [3] is the implementation derived from the original work introducing the concept of environment modules. [1] *Lmod*[2] is a Lua-based successor to *Environment Modules* with a variety of improvements. Both of these systems allow administrators to create *module files* that define modifications to a user's environment. These modifications usually enable some collection of software to be accessible to the user once applied. This method of dynamically loading software when needed instead of installing it globally greatly benefits software organization and minimizes conflicts between software installations. It also enables multiple versions of a software package to be installed at a time, allowing users to load any installed verison they need.

3 CONCEPT AND USAGE

Consider a researcher trying to use the common `NCBI BLAST` search tool on her organization's cluster. She has a tutorial that gives her instructions on the `blastx` command. The cluster has an *Lmod* installation available to the users, with the `MODULEPATH` already correctly configured by the administrators.

3.1 Module Autoloading

If the researcher is unfamiliar with module-based systems, she might try and execute `blastx (...)` and receive an error: `blastx: command not found`. Now, if she is aware of the environment module system she might try and load the module by executing `module load blast`. However, this is not the correct module name and she will receive another error: `(...) the following module(s) are unknown: "blast"`. At this point the friendly administrator is contacted who informs her the correct way to load the blast module is

`module load blast-plus`. Finally the researcher loads the module and is able to use the `blastx` tool as expected.

Mii eliminates this problem and makes loading modules more friendly and automated. Consider the same scenario as before, except now *Mii* is installed on the system with shell integration enabled. If the researcher is unfamiliar with module-based systems she might execute `blastx (...)`. The `blast-plus` module is not loaded so the shell is unable to execute the binary directly. Instead the shell-specific "command not found" hook is called with the command information. Here *Mii*'s shell integration intercepts the failure and searches for modules that provide the `blastx` command. The *Mii* search responds with an available module `blast-plus`. As only one version of this module is installed on the system, it is immediately loaded and the original command is re-executed with the original arguments. Our researcher sees *Mii*'s response before `blastx` is executed as originally intended: `[mii] autoloading blast-plus/2.7.1`. In the event that multiple versions of `blast-plus` were available as modules, the researcher would be prompted to select a version before proceeding with the command.

3.2 Fuzzy Searching

Mii also supports fuzzy searches. When the researcher executes the command `BlastX`, *Mii* is unable to find any modules that provide the command. The database is then searched for similar-looking commands, and the researcher receives a suggestion: `[mii] hint: try a similar command 'blastx' 'tblastx'` indicating the correct command to execute.

3.3 Implementation Details

Mii reconstructs its database whenever the user logs into a shell. This ensures that the module index is always up to date with the installed modules on the system. *Mii* is written in POSIX-compliant C and uses *SQLite 3* to store the module index. The search engine is designed to have minimal dependencies and to be easy to deploy into an existing system. Integration scripts are shell-specific and must be written for each supported shell. I only wrote scripts for `bash` and `zsh` but any shell with a command-not-found hook can be supported.

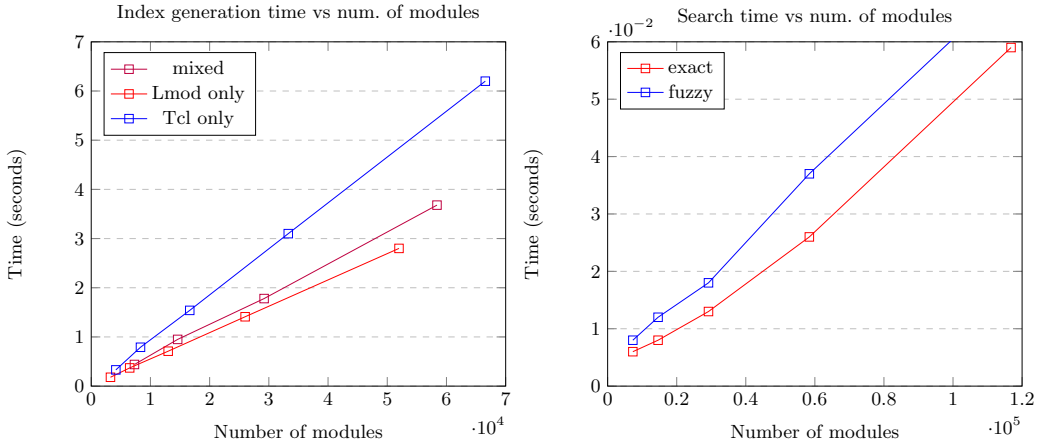
4 PERFORMANCE EXPERIMENTS

I conducted extensive performance testing with the *Mii* environment, analyzing the speed of building the database as well as querying it. As *Lmod* module files are scanned in a different manner than *Environment Modules (Tcl)* module files separate data was collected for each module type.

5 ANALYSIS

The index building performance data from figure 1 indicates that *Lmod* modules are significantly faster to index than *Tcl* modules. This is to be expected as the *Lmod* analysis method executes a simple regular expression on the module file while the *Tcl* analysis method must simulate the whole file. This is because *Tcl* modules can contain variable expansions in the `PATH` variables which requires a more expensive analysis. Both the searching and indexing performance appear to execute in linear time which is to be expected.

Fig. 1. Performance graphs



6 CONCLUSIONS

In summary, *Mii* has been developed to be a useful extension to an existing module-based system. Based on the performance analysis *Mii* is fast enough to be integrated with most existing systems without any noticeable performance impact. The *Mii* engine introduces a new type of module interaction and interactive user experience.

7 FUTURE WORK

While *Mii* is in a working state and accomplishes its primary goal, there is still much room for improvement and new features. *Mii* does not take advantage of multithreaded systems, which could radically improve the index building performance. Also, rebuilding the module index on every login is often unnecessary; perhaps another trigger could be introduced that would only rebuild the index when modules are added or removed.

Mii currently only indexes modules by binaries provided in their `PATH`s. The index could be expanded to store even more relevant module information for other uses, such as manual pages provided in `MANPATH`s. *Mii*'s shell integration could then be further expanded to automatically load modules for missing manual pages.

ACKNOWLEDGMENTS

Thanks to Levi Baber for introducing me to HPC environments entirely, and encouraging me to contribute to PEARC. And special thanks to Miles Perry for ideas and feedback!

REFERENCES

[1] John L. Furlani. 1991. Modules: Providing a Flexible User Environment. Retrieved April 14, 2019 from http://modules.sourceforge.net/docs/Modules-Paper.pdf

[2] Peter W. Osel John L. Furlani. 2019. LMod: A New Environment Module System. Retrieved April 14, 2019 from https://lmod.readthedocs.io/en/latest/

[3] Peter W. Osel John L. Furlani. 2019. Modules – Software Environment Management. Retrieved April 14, 2019 from http://modules.sourceforge.net/

Table 1. Index construction time: mixed module types

Num. modules	Time (s)
7300	0.44
14600	0.95
29200	1.78
58400	3.68

Table 2. Index construction time: Lmod modules

Num. modules	Time (s)
3249	0.18
6498	0.37
12996	0.71
25992	1.41
51984	2.80

Table 3. Index construction time: Tcl modules

Num. modules	Time (s)
4160	0.33
8320	0.79
16640	1.54
33280	3.10
66560	6.20

Table 4. Query time: exact search

Num. modules	Time (s)
7300	0.006
14600	0.008
29200	0.013
58400	0.026
116800	0.059

A PERFORMANCE DATA

For performance data please refer to tables 1, 2, 3, 4 and 5.

Table 5. Query time: fuzzy search

Num. modules	Time (s)
7300	0.008
14600	0.012
29200	0.018
58400	0.037
116800	0.070