

VaspHandler

DFT calculations across a lab of unreliable machines.

Stack	Python – a command-line client, a daemon on each machine, an HTTP API between them, and a pendrive installer that carries its own interpreter. A companion PHP tool on navyasharma.org shows the lab from a browser. pytest, plus a simulated lab for the things pytest cannot reach
Status	Complete. In use in a materials lab since mid-2026; releases continue as maintenance
Size	About ninety commits over eight tagged versions, and a little under 800 tests

The problem

VASP is the workhorse of computational solid-state physics: give it a crystal and it returns energies, forces, band structures. It is slow, it is licensed per machine, and the machines a university lab actually owns are a dozen desktops of mixed age under different desks, switched off by whoever leaves last. A single relaxation can run for hours; a campaign is hundreds of them.

Before this existed, running a campaign meant walking round the room, starting jobs by hand, remembering which machine had which result, and discovering in the morning that the one with the important calculation had been turned off at six. VaspHandler is what I wrote so that the pile of hardware behaves like one machine, and so that a person can submit work and leave.

What it does

Each computer runs a small daemon. A member of the lab submits a job from a command line or from a page on this site; the job is broken into tasks, each task is *leased* to a machine that is awake, and the daemon on that machine runs VASP in a staged directory and reports back. Results are content-addressed and copied to more than one disk, so a machine going dark does not take the only copy with it. The lab's whole state – what is queued, what is running where, what is done, which machines are alive – is mirrored every minute to the website, so it can be read from anywhere without a VPN or a walk.

A calculation that is interrupted does not start again. VASP writes its current geometry as it goes; an interrupted relaxation is resumed from that file on the next machine that picks the task up. Multi-step workflows – relax, then a self-consistent run, then bands and a density of states – are one job whose later tasks depend on the earlier ones, and the scheduler keeps them in order.

The lab is installed from a pendrive. The stick carries a bundled Python, the release, and an installer that proves VASP actually runs on the machine before it says ready. None of the lab's computers can be trusted to have a working interpreter, and this was the first design decision that survived contact with the building.

No election

The early versions used Raft. Every machine held a council seat and the council elected an admin to run the queue. It is the textbook answer to “who is in charge,” and it was the wrong one for this lab.

A five-seat council needs three machines awake to do anything. This lab is a dozen machines that people switch off on their way home. Losing the majority meant losing the lab – nothing could be queued, nothing could be assigned – until somebody drove in and pressed power buttons. It happened twice and cost days each time, and on the second occasion nobody had done anything wrong: four of the five seats had simply been switched off overnight, as they are every night.

So now each machine is given a rank when it is installed, and the admin is simply the reachable machine with the best rank. No vote, no quorum, no minimum. One machine on its own runs the lab.

What is given up is written down rather than glossed. During a network partition two machines can briefly both believe they are the admin; Raft prevented exactly this. The trade is sound only because of properties the system already had: tasks are leased and idempotent, results are content-addressed, an interrupted calculation resumes from its own files. So a split costs some duplicated work, while a lost quorum cost a fortnight. The docstring on the leadership module carries the full argument, including the conditions under which it would stop holding – if tasks ever stopped being idempotent, the council would have to come back.

The migration happened on real hardware and kept the queue: the new admin found the old council's state on disk, read its jobs and tasks and artifacts into the new form, and carried on. The test suite went from about a hundred seconds to about thirty, which was a consequence rather than an optimisation – the old council tests started real Raft nodes and killed them, and most of their time was spent waiting for elections. Their claims were not deleted with them; each is re-made against what replaced it.

Testing a lab that does not exist

Beside pytest there is a simulated lab: real daemons, a fake VASP that produces plausible output in seconds, and a power switch. It can turn a machine off mid-task, partition the network, bring a machine back with a stale view of the world – everything the real lab does by accident, on demand and in a test. Most of the bugs that mattered were found there before they could be found in the building.

Not all of them. One release died at startup on every real machine. The daemon asks the network what version it should be running before it serves anything, and the council removal had taken away two names that method used while leaving the calls. Every test and the whole simulated lab ran with that check switched off, so the first branch a real machine takes was the one branch nothing exercised. Nearly 750 tests passed over it. The fix took an hour; the lesson took longer, and there are two of them. A static checker now runs inside the suite as a test that fails rather than a lint that warns. And a release is not called built until the installed daemon has been started from the installed interpreter, on the bytes the stick carries.

A second bug of the same family lived in the lab for weeks without anybody seeing it. Every machine logged, every ten minutes, that it could not fetch a release from the website – the host in front of the site refuses a bare programmatic request, and every call to the site sent the header that satisfies it except the one that downloads a release. The machines' status syncs landed in the same minutes, so from outside the site looked reachable. The page for pushing a release to the lab had never once worked, and no page said so. The cure was one keyword argument; the finding was reading the lab's own logs rather than the workstation's.

One file for the present tense

Six versions in a row shipped with a document claiming something that had stopped being true, and twice a false claim decided how somebody spent their day – once sending a session to rebuild a pendrive that was already current, once sending one after the wrong problem for an hour because a document still said VASP had never run.

The cause was arithmetic rather than carelessness. The same fact lived in a README, a handover file, four typeset documents, their PDFs and their copies on the stick: eight places, updated by hand, and eight is too many to keep honest. So present-tense facts now live in one file and nowhere else, that file carries the date it was last reconciled, and a test fails the suite when it and the repository disagree. Everything else that describes the project is allowed to be historical.

It is a small rule. It is also the one thing from this project I have carried into every project since, including the register that tracks all of them.