Benchmarking experiment: purpose, design, and automation

György Barabás

2025-08-21

The purpose of the benchmarking

The purpose has gone through several iterations, and the original impetus for doing it is not quite what we are using it for now. Back in the day (around April-May of 2024, but he will know better), Marko Zidaric wanted to check how Swarm's data retrieval speeds compare with those of other decentralized data storage platforms. So he set up an experiment where he up- and downloaded files to Swarm, IPFS, and Arweave. The first results were promising, but Viktor Trón wanted a more rigorous procedure with systematically varied factors, an equal number of replicate trials per each factor combination, and the guaranteed elimination of confounding factors such as cached downloads. (See the next section for a description of the experimental protocol.)

Eventually, this was implemented, and we compared Swarm with the other two platforms (spoiler alert: Swarm is faster for small files but slower for big ones). By that time, however, our priorities have shifted, and we became more interested in how Swarm (and only Swarm) performs compared to earlier versions of itself. That is, the benchmarking experiment became a tool to check how new Swarm features affect up- and download speeds. This means that much of the data analysis consists of comparing results from two different runs of the experiment; one without, and one with some feature. For example, we compared the performance of the 2.5 and 2.6 releases this way, as well as 2.6 with the same release but with PR 5097 (erasure coding) enabled. I think the plan is to keep using the experiment in this way in the future as well.

The current experimental protocol

Here is a description of the current experimental design (as of August 2025). The purpose is to measure up- and download speeds on Swarm for various file sizes, erasure settings, and retrieval strategies. We vary all parameters in a fully factorial way, to get at all their possible combinations. The factors are as follows:

- `size`: The size of the uploaded random file. We have 6 distinct factor levels: 1 KB, 10 KB, 100 KB, 1 MB, 10 MB, and 50 MB. Every file has a size that matches one of these values exactly. Importantly, every single upload is a unique random file, even if the file sizes are otherwise equal—this removes the confounding effects of caching.
- `erasure`: The strength of erasure coding. We have five factor levels: `0` (= `NONE`), `1` (= `MEDIUM`), `2` (= `STRONG`), `3` (= `INSANE`), and `4` (= `PARANOID`).
- `strategy`: The retrieval strategy used to download the file. Its value is necessarily `NONE` in the absence of erasure coding—i.e., when `erasure = 0`. Otherwise, it is either `DATA` or `RACE`.
- `server`: The identity of the server initiating the downloads might influence download speeds. For a fair comparison, servers should be identical within an experiment, but it makes sense

to perform the whole experimental suite over multiple different servers, to control for server-specific effects. This means that we have an extra experimental factor, with as many distinct levels as the number of distinct servers used. Here we use three distinct servers: `Server 1`, `Server 2`, and `Server 3`.
- `replicate`: To gain sufficient sample sizes for proper statistical analysis, every single combination of the above factors is replicated 30 times. For example, given the unique combination of 1MB files uploaded without erasure coding on Server 1, we actually up- and downloaded at least 30 such files (each being a unique random file).

The above design has (6 file sizes) $\times$ (5 erasure code levels) $\times$ (3 retrieval strategies) $\times$ (3 servers) $\times$ (30 replicates). However, the `NONE` retrieval strategy is only ever used when `erasure` is `NONE`, and the `DATA` and `RACE` strategies only when `erasure` is not `NONE`. So the total number of unique download experiments is (30 replicates) $\times$ (6 file sizes) $\times$ (3 servers) $\times$ (1 strategy & erasure level + 2 strategies $\times$ 4 erasure levels), or 4860.

Some further notes about the experimental design:

- All uploads are direct, as opposed to deferred.
- We need to make sure that no download starts after the system has properly stored the data. Since our files are relatively small, uploading should be done in a few minutes at most (as we will see later, the longest upload in our data took below 3 minutes). So we opted for a crude but reliable way of eliminating any syncing issues: we waited exactly 2 hours after every upload, and began downloading only then.
- All downloads are done using nodes with an active chequebook.
- Every download is re-attempted in case of a failure. In total, 15 attempts are made before giving up and declaring that the file cannot be retrieved.

What is automated, what is not, and what could be

Currently, the data collection, together with generating random files, uploading them, and then downloading, are all automated. What isn't yet automated is the infrastructure setup. Before the experiment is run, Marko Zidaric has to make sure that all participating nodes are topped up, have active chequebooks, and so on. The question is whether this could be automated as well. I believe the answer is yes, but we should discuss with Marko. The question is whether one could write a script that automatically looks through a set of nodes and tops them up, waits for this to take effect, and then runs the data collection script.

One question is whether the current data we collect are good enough. I actually think the answer is yes. We systematically go through all possible erasure coding, retrieval strategy, file size, and server combinations, to make sure they aren't showing up as confounds later. When comparing results (across releases, for instance), this allows one to look at parts of the data where we see the most interesting signal. It means that maybe not all data are equally valuable at all times, but having them cannot hurt.

Bonus suggestion: it might be worth increasing the number of replicates from 30 to some higher number—say, 50. This is likely not critical, but more data are always better, as long as we can afford it. At the moment, the file sizes go up to 50 MB only, which means that the experiment would finish in a reasonable time even with an increased sample size. It ought to be a trivial change in the run script (and of course it means that perhaps a larger top-up will be needed for the nodes, to support the uploading of the extra data).