Description and Analysis of the Web3 Storage Benchmarking Experiment

György Barabás and Marko Zidarić

Purpose and experimental design

The purpose of this project was to compare data download speeds and reliability across three different Web3 storage platforms: Arweave, IPFS, and Swarm. The data were gathered by first uploading files of various sizes to all three platforms, and then downloading them under different circumstances. We then compared the times needed for data retrieval to see whether and when some platform allowed for shorter download times than the others.

Our goal was to implement this speed comparison as a proper, repeatable, well-designed digital experiment. This meant that we always uploaded unique, randomly generated files of fixed size, which were then downloaded from the same server to remove the confounding influence of server identity (the experiment as a whole can, and was, then repeated over several different servers; see below). We up- and downloaded the exact same number of files of specified sizes on each platform. We additionally varied some other parameters as well to gauge their influence on download speeds. We varied all parameters in a fully factorial way, to get at all their possible combinations. The factors were as follows:

- `size`: The size of the uploaded random file. We had 6 distinct factor levels: 1 KB, 10 KB, 100 KB, 1 MB, 10 MB, and 100 MB. (In other words, starting at 1 KB and always increasing by a factor of 10 up to 100 MB.) Importantly, every single upload was a unique random file, even if the file sizes were otherwise equal.
- `platform`: Whether the target platform is Arweave, IPFS, or Swarm. Additionally, Swarm itself breaks up into several sub-factors depending on:
  - the strength of erasure coding employed (`0` = None, `1` = Medium, `2` = Strong, `3` = Insane, and `4` = Paranoid);
  - the used redundancy strategy (we use `NONE` whenever erasure coding is set to 0; otherwise we use `DATA` and `RACE`).

  Together, these lead to 11 distinct factor levels for `platform`, namely `Arweave`, `IPFS`, and all combinations of the above for Swarm: `Swarm_0_NONE`, `Swarm_1_DATA`, `Swarm_1_RACE`, `Swarm_2_DATA`, `Swarm_2_RACE`, `Swarm_3_DATA`, `Swarm_3_RACE`, `Swarm_4_DATA`, and `Swarm_4_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—which is what we have done. This means that we have an extra experimental factor, with as many distinct levels as the number of distinct servers used. Here we have used three distinct servers.

- `replicate`: To gain sufficient sample sizes for proper statistical analysis, every single combination of the above factors was replicated 30 times. For example, given the unique combination of 1MB files uploaded to IPFS on Server 1, we actually up- and downloaded at least 30 such files (each uniquely and randomly generated, of course).

The above design leads to (6 filesizes) x (11 platforms) x (3 servers) x (30 replicates) = 5940 unique download experiments. They are summarized in tabular format as well in the README of the GitHub repository https://github.com/darkobas2/util_web3_storageperf/tree/V2.

Additionally, here are some further notes and points about the experimental design outlined above:

- For Swarm (and only Swarm), upload speeds were also measured, using the tags API to make sure that all chunks have been properly placed and uploaded to the system before declaring a file fully uploaded.
- The reason for insisting on unique, randomly generated files for uploading was to eliminate the possibility of cached downloads (if supported by the target platform).
- All uploads to Swarm were deferred. This is not ideal for this experiment, so we will likely change this and re-run everything by setting upload type to direct.
- We needed to make sure that no download started after the system has properly stored the data. Since our files are relatively small, uploading should be done in about 10-20 minutes at most. 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.
- When testing IPFS, the data were uploaded from one server but downloaded from another.
- All Swarm downloads were done using nodes with an active checkbook. (Eventually we might also repeat the experiment with no checkbook, but for now we are not dealing with this issue.)
- Every download was re-attempted in case of a failure. In total, 15 attempts were made before giving up and declaring that the file could not be retrieved.

Preliminary data analysis

After we load and compile the data, we get a table with 6966 rows and 8 columns. Each row corresponds to a single download. The first ten rows are:

platform	server	size_kb	erasure	strategy	time_sec	sha256_match	attempts
IPFS	Server 1	1	NONE	NONE	1.6987	TRUE	1
IPFS	Server 2	1	NONE	NONE	1.5403	TRUE	1
IPFS	Server 3	1	NONE	NONE	0.0949	TRUE	1
IPFS	Server 1	1	NONE	NONE	0.1942	TRUE	1
IPFS	Server 2	1	NONE	NONE	0.1948	TRUE	1
IPFS	Server 3	1	NONE	NONE	0.0555	TRUE	1
IPFS	Server 1	1	NONE	NONE	0.2391	TRUE	1
IPFS	Server 2	1	NONE	NONE	0.3857	TRUE	1
IPFS	Server 3	1	NONE	NONE	0.0503	TRUE	1
IPFS	Server 1	1	NONE	NONE	0.2390	TRUE	1

A quick note: we just calculated that the experimental design leads to 5940 downloads, yet here we have 6966 rows of data. This is because for Arweave and IPFS we had 60 replicates per factor combination instead of 30 (with one exception: we only had 51 replicates for files of size

100 MB). Instead of discarding the extras, we kept them to lend further statistical power down the line. This was also important to do because some of the downloads failed: 327 out of the 6966 downloads could not finish even in 15 attempts. This is how these failures are distributed across the parameterizations:

platform	server	size_kb	erasure	strategy	number of fails
IPFS	Server 1	1e+05	NONE	NONE	38
IPFS	Server 2	1e+00	NONE	NONE	30
IPFS	Server 2	1e+01	NONE	NONE	30
IPFS	Server 2	1e+02	NONE	NONE	30
IPFS	Server 2	1e+03	NONE	NONE	30
IPFS	Server 2	1e+04	NONE	NONE	30
IPFS	Server 2	1e+05	NONE	NONE	45
IPFS	Server 3	1e+05	NONE	NONE	38
Swarm	Server 1	1e+03	STRONG	DATA	2
Swarm	Server 1	1e+04	INSANE	DATA	2
Swarm	Server 1	1e+04	MEDIUM	DATA	1
Swarm	Server 1	1e+05	INSANE	DATA	5
Swarm	Server 1	1e+05	MEDIUM	DATA	1
Swarm	Server 1	1e+05	PARANOID	DATA	6
Swarm	Server 1	1e+05	STRONG	DATA	2
Swarm	Server 2	1e+03	STRONG	DATA	2
Swarm	Server 2	1e+04	INSANE	DATA	2
Swarm	Server 2	1e+04	MEDIUM	DATA	1
Swarm	Server 2	1e+05	INSANE	DATA	5
Swarm	Server 2	1e+05	MEDIUM	DATA	1
Swarm	Server 2	1e+05	PARANOID	DATA	6
Swarm	Server 2	1e+05	STRONG	DATA	2
Swarm	Server 3	1e+03	STRONG	DATA	2
Swarm	Server 3	1e+04	INSANE	DATA	2
Swarm	Server 3	1e+04	MEDIUM	DATA	1
Swarm	Server 3	1e+05	INSANE	DATA	4
Swarm	Server 3	1e+05	MEDIUM	DATA	1
Swarm	Server 3	1e+05	PARANOID	DATA	6
Swarm	Server 3	1e+05	STRONG	DATA	2

On the other hand, all the other 6639 downloads succeeded, and all but two of them on the first attempt. (One download of a 100 MB file on Arweave completed on the 8th attempt, and another 100 MB file on Swarm on the 9th.) Before analyzing the data, we remove the 327 failed downloads from the table and work only with the rest.

Visualizing and interpreting the raw data

The data from the experiment are shown in Figure 1. We have a grid of panels, with 7 rows and 3 columns. The columns are the three different servers used for downloading. The rows indicate storage platform and level of erasure coding (if applicable). Within each panel, file size is along the x-axis and retrieval time along the y-axis, both on the log scale. Colors show retrieval strategies: when erasure coding is absent, blue means NONE, whereas in the presence of erasure coding, it means DATA. Yellow indicates the RACE strategy. Each point corresponds

to one download event. The points have been jittered sideways in a way that reflects the general shape of their distribution. This means that the only file sizes are 1 KB, 10 KB, 100 KB, 1 MB 10 MB, and 100 MB; points that do not fall at those sizes are jittered and so they still have one of these six sizes.

Some important take-aways from this figure:

- Arweave's download times increase the slowest, though it also takes longer to download small files from it.
- IPFS download times increase somewhat faster.
- Swarm increases the fastest, so for large files it is the least efficient. This was expected, given its underlying DISC model.
- The level of erasure coding does not appear to have much of an effect on download speeds.
- The DATA and RACE retrieval strategies do lead to differences. For small files RACE is faster; for larger files, sometimes RACE is faster, sometimes DATA.

Modeling download times

We now turn to building simple predictive models for these data. Their purpose is to be able to extrapolate how long it would take to download larger dara files than the ones used in the data. The models are phenomenological, and therefore should be used with care: they will not extrapolate well for file sizes much, much larger than the ones tested. But they can give an idea of how each parameter is expected to influence the results.

We will build separate models for the three platforms. Since it is simpler, let us start with Arweave and IPFS, and then move on to Swarm.

Arweave and IPFS By looking at the data in Figure 1, we see that log download times increase nonlinearly with log file size. The relationship may be quadratic or cubic; here we assume it is cubic (this assumption does have limitations, as we will see later). To avoid bias arising from the fact that the log function compresses large values more than small values, we want to use a generalized linear model with a log link function to predict download times. Also, since the data come from three different servers, we will take the influence of server identity into account as a random effect. The model therefore reads

$$(\text{time})_i = \exp\left[\beta_0 + \beta_1 \log^3(\text{size})_i + \mu_{0,\text{v}(i)} + \mu_{1,\text{v}(i)} \log^3(\text{size})_i\right]. \tag{1}$$

Here $(\text{time})_i$ and $(\text{size})_i$ are respectively the ith download time (in seconds) and file size (in kilobytes) in the data, β_0 is an intercept, β_1 is a slope, and $\mu_{\text{v}(i)}$ is a random effect, where $\text{v}(i)$ returns 1, 2, or 3, depending on which server the ith data point was downloaded from.

Swarm Since Swarm additionally has erasure coding and different retrieval strategies, the fitted model is also more complex, although it is still a Gaussian generalized linear mixed model with a log-link function. It has the form

$$\begin{aligned}(\text{time})_i = \exp[&\beta_0 + \beta_1 \log^2(\text{size})_i + \beta_2(\text{erasure})_i + \beta_3(\text{strategy})_i \\ &+ \beta_4 \log^2(\text{size})_i(\text{erasure})_i + \beta_5 \log^2(\text{size})_i(\text{strategy})_i + \beta_6(\text{erasure})_i(\text{strategy})_i \\ &+ \mu_{0,\text{v}(i)} + \mu_{1,\text{v}(i)} \log^2(\text{size})_i + \mu_{1,\text{v}(i)}(\text{erasure})_i].\end{aligned} \tag{2}$$

In words, we predict log(time) using the main effects of $\log^2$(size), erasure level, and strategy, and all their possible two-way interactions. Plus we add a random intercept and slope (for both $\log^2$(size) and erasure level) based on server identity. In this model, erasure level is not a factor but a covariate reflecting the corresponding baseline probability of a faulty chunk retrieval

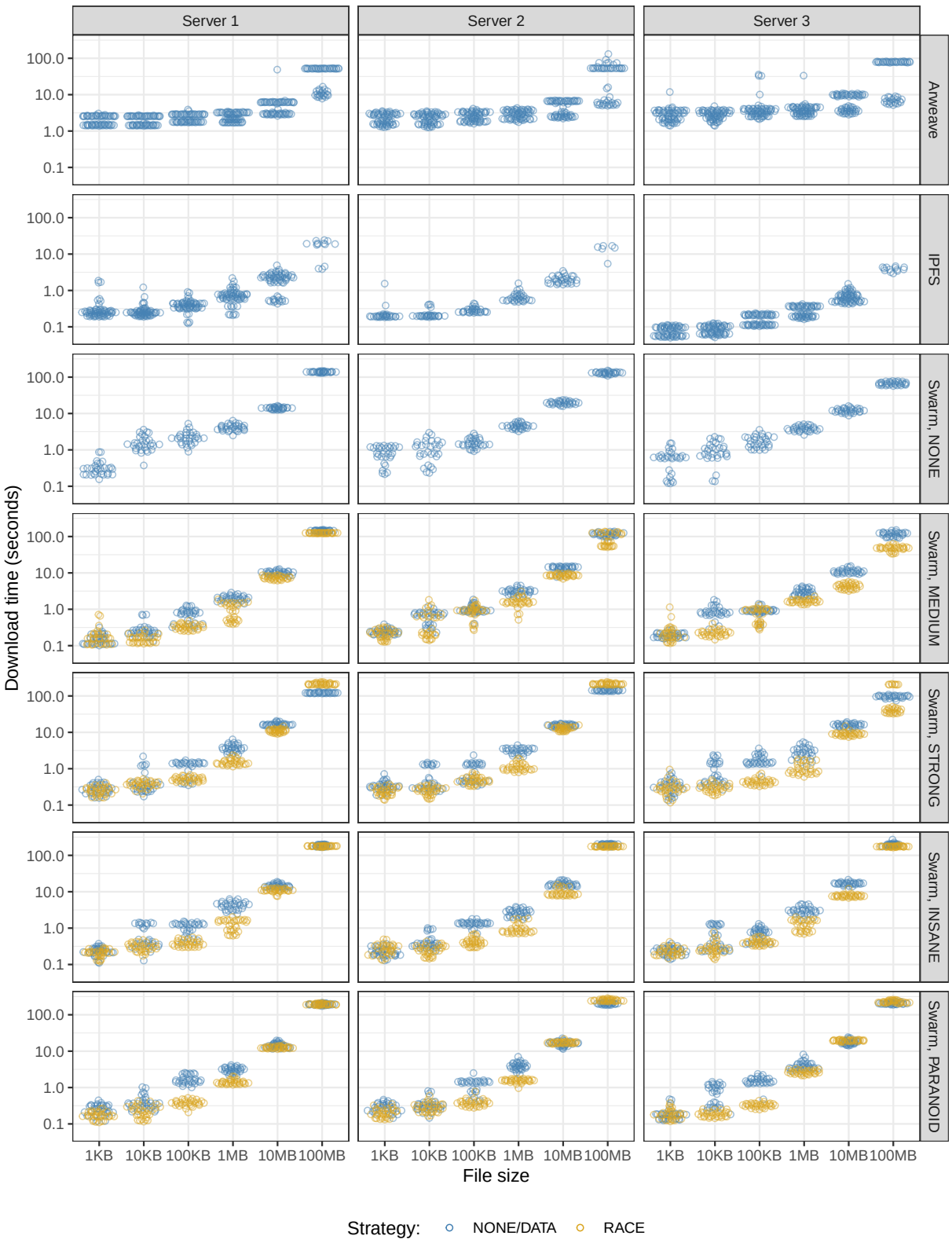

Figure 1: Empirical results from the benchmarking experiment

event: it is taken as 0 for no erasure coding, as 1 for MEDIUM, 2 for STRONG, 3 for INSANE, and 4 for PARANOID. (Model selection indicated this as the best choice among a few other candidates, such as using the probability of individual chunk failure or using the fraction of chunks that are parity chunks.)

Before doing anything else, let us confirm that the models do capture the data well. To do so, we plot the raw data (points) together with the model predictions (lines) in Figure 2.

The models generally fit well, though they are sometimes off at small file sizes. Note however that, since the y-axis is on the log scale, a difference that appears large at small download times is not actually a relevant difference. For example, the average download time for Swarm on Server 1 at PARANOID erasure and RACE strategy is 0.189 seconds. The model's prediction is 0.06 seconds which, although more than a threefold difference, is only 0.13 seconds off the empirically observed value.

Now that we have more confidence in the models, we can use them to make out-of-sample predictions. Figure 3 shows model-predicted extrapolations made from the data, averaged over the random effects.

Three remarks about these results are in order. First of all, the modeled curves for Arweave and IPFS eventually increase faster than the curves for Swarm. This would mean that for very large files, Swarm becomes the most efficient platform. This is definitely an artifact and reflects the limitations of the models, rather than telling us something about how these systems work in reality. In particular, this is the result of having modeled Swarm with a quadratic and Arweave/IPFS with a cubic curve. Since cubic functions increase faster than quadratic ones for large arguments, the model's predictions will only be reliable up to some limited file size. Second, on Swarm and for large files, RACE is on average not a better retrieval strategy than DATA: the dashed lines lie above the solid ones for files over 1GB. And third, while for large files the RACE curves are only slightly above the DATA curves, one must remember that the y-axis is on the log scale. This means that the actual difference is substantial. For example: for 10 GB files, the DATA strategy takes around 8 hours and the RACE strategy around 18 hours to download, with erasure levels playing only a minor role in shaping these figures.

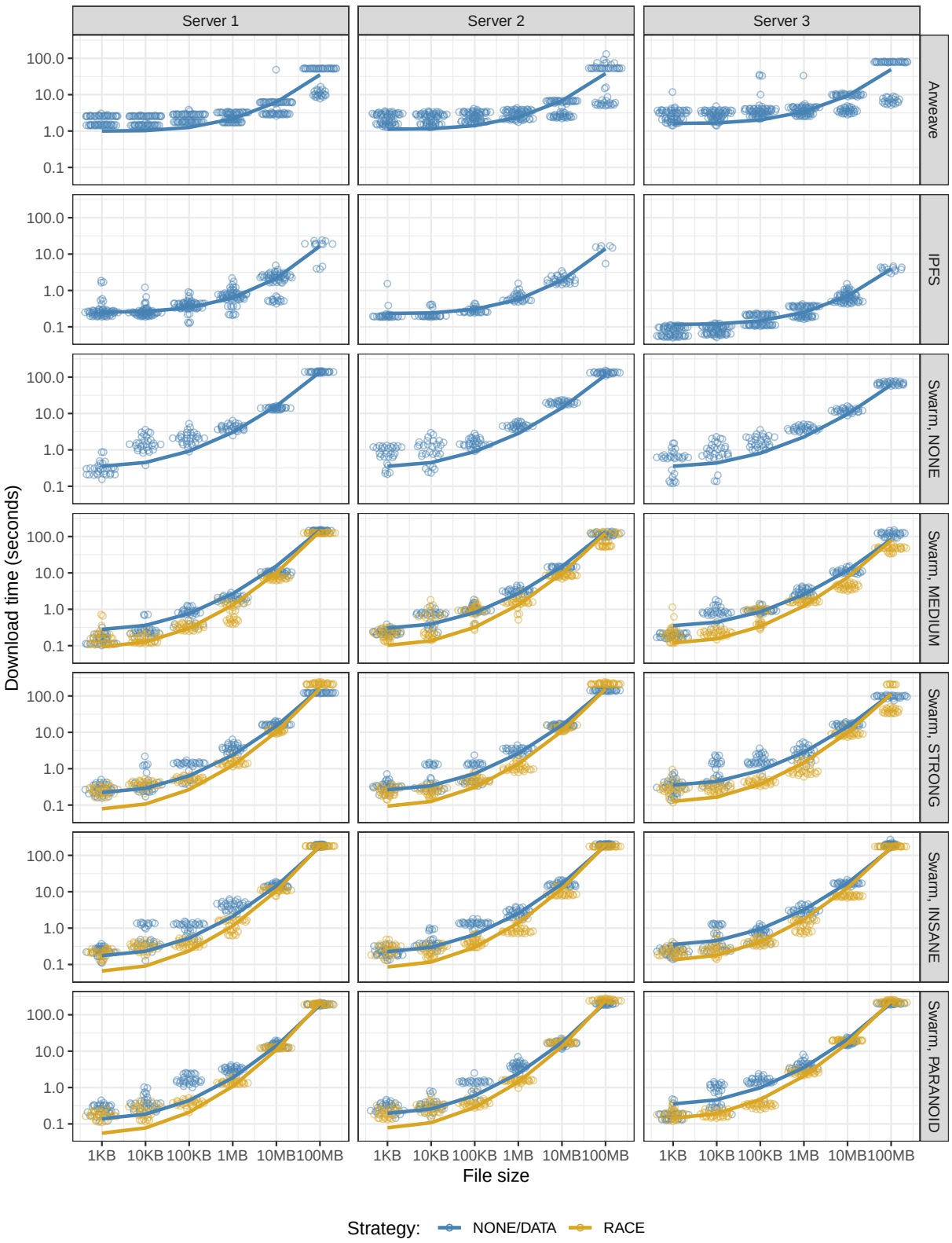

Figure 2: Data together with predictions from the fitted models

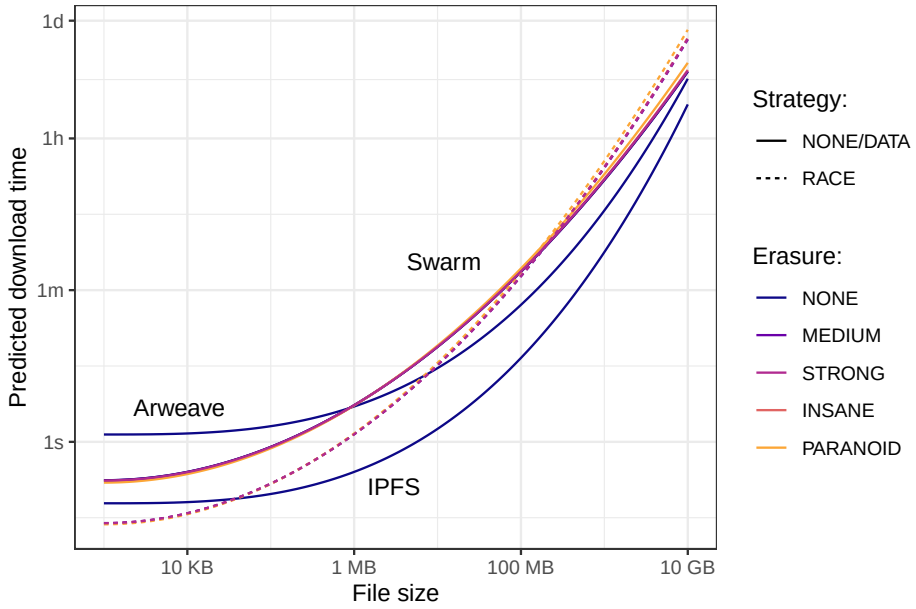

Figure 3: Model-based predictions for mean file retrieval times. Random effects are averaged over, so server identity no longer plays a role.