Sia Load Test Plan

Status: Open for community review (2018-02-08)
Author: Michael Lynch, blogger at SpaceDuck
Reviewers

- Luke Champine, CTO of Nebulous

Objective

Determine the maximum renter storage capacity of a single Sia node.

Background

The Sia community does not have empirical data about how much data a single Sia node can rent. Several Sia users have uploaded data to Sia, but nobody has published results of a rigorous experiment to determine Sia's limits.

An accurate estimate of Sia's per-node capabilities will help the community understand the actual costs of storage on Sia, and will help third-party developers build solutions on top of Sia.

Test Environment

I will perform this test using consumer PC hardware in my home.[1]

Sia will run on a Windows 10 PC. Data files for for the test will be stored on a Synology NAS device.

PC

- OS: Windows 10 x64
- CPU: Intel i7-5820K @ 3.3 GHz
- RAM: 32 GB
- Disk: 512 GB SSD

NAS

- Synology DS412+
- 4 TB free space

[1] See Appendix A: Rationale for using home infrastructure

Network

- Local network: 1 Gbps
- Internet: Verizon FiOS
    - Advertised: 940 Mbps download, 880 Mbps upload
    - Actual: 300-600 Mbps download, 8-175 Mbps upload

Sia

- Build: 1.3.1-windows-amd64 (release build)

Sia Configuration

It is important to reset state between tests so that there is no cross test contamination. Between test cases, I will wipe all Sia data except for two files:

- `consensus/consensus.db`
    - This takes a very long time to sync and does not contain any node-specific state. I will not delete this file between tests.
- `renter/hostdb.json`
    - It takes several days for the host database to stabilize. Instead of re-using it across tests, I will snapshot it before tests begin, then use the snapshot copy in each testcase.

For each test, the Sia node will start with a fresh wallet. Between tests, I will delete all Sia data folders except for the `consensus` folder, which I will reuse across tests for convenience. I believe there is a low risk of cross-test contamination from re-using the `consensus` folder.

I will initialize each wallet with a newly generated seed, funded with 5 KS.

```
Renter Prices (estimated):
  Fees for Creating a Set of Contracts:   98.7 SC
  Download 1 TB:                         18.89 SC
  Store 1 TB for 1 Month:                133.9 SC
  Upload 1 TB:                           27.6 SC
```

Sia renter price estimates, as of 2018-01-13.

The wallet amount is based on an upper limit of uploading 10 TB of data for a three month contract:

```
98.7 + (27.6 * 10) + (133.9 * 10 * 3) = 4391.7 SC
```

I round this 4391.7 SC up to 5 KS to add a bit of buffer if renter prices change between the test plan and the test execution.

The test script will call the `/renter` POST API[2], specifying `funds` to the full wallet balance amount and `period` to `(4320 * 3)` blocks.[3] The load test will begin when the Sia node creates 50 renter contracts.[4]

Test Format

A Python script (using pysia) will perform each of the tests. The script will continue uploading files from the test dataset until uploads stop making progress. “Progress” here is defined by uploading >= 100 MB of data in aggregate over the past hour.

The script will not begin uploading a new file until < 5 uploads are in progress. An upload is considered “in progress” if the `/renter/files` API[5] returns a value < 100 for the file’s `uploadprogress` property.

If the tests exhaust the free space of the NAS, I will manually delete already uploaded files, generate additional data files, and continue the test script.

If the siad process crashes or becomes unresponsive to RPCs, I will manually restart it up to 5 times per testcase. After 5 crashes or hangs, the test is considered complete. I will otherwise not restart the siad process.

Note that this test does **not** exercise download functionality. I assume that files uploaded successfully can be downloaded with their integrity preserved.

Test Cases

1. Optimal Case

Data consists of files exactly 41942760 bytes (~40 MiB) in size, filled with random data.

This is optimal size for Sia, as each file will be exactly one full data chunk in Sia:

- chunkSize = pieceSize * dataPieces (source)

---

[2] /renter API documentation
[3] This mirrors Sia-UI’s behavior.
[4] “How to Put Data on the Sia Network”
[5] /renter/files API documentation

    - pieceSize = SectorSize - TwofishOverhead (source)
    - pieceSize = 2^{22} (source) - 28 (source)
    - pieceSize = 4194276 (~4.2 MB)

    - dataPieces = 10 (source)

- chunkSize = 4194276 * 10
- chunkSize = 41942760 bytes (~40 Mib)

2. Worst Case

Data consists of files 1 byte in size.

3. Actual Data

Data consists of 4.33 TB of actual DVD/Blu-Ray data (raw ISOs and compressed mp4s). Files range in size from ~100 MB to as large as 48 GB.

Outputs

At the end of the test, I will publish:

- Source code of load test script
    - I will publish this to my personal Github, under the MIT license.
    - The repository will include everything needed for another party to reproduce my results or extend the code, including:
        - Documentation
        - Unit tests
        - A Travis CI configuration
- Result report
    - I will publish a report on a public website detailing the results of this test.
    - For each testcase, the report will include:
        - Total amount of data uploaded
        - Total cost (in SC and USD equivalent)
        - All log files
        - siac output for relevant modules:
            - `siac renter -v`
            - `siac renter contracts`
            - `siac wallet`
        - Notable events that occurred during the test (e.g., crashes, unexpected log messages).

Timeline

Coding begins 2018-02-02. Outputs published by 2018-02-16.

Appendix A: Rationale for using home infrastructure

A “pure” test would occur on cloud infrastructure, such as Amazon EC2 or Google Compute Engine. This would aid in reproducibility and measurement because it would eliminate some sources of interference that happen in a home, such as competing network activity (e.g. watching streaming 4K video on the same connection while the test runs).

Despite this, I chose to use home infrastructure because:

- Cost of cloud infrastructure is prohibitive
    - On AWS, the cost of bandwidth alone is $92 per TB transferred out. These tests can generate up to 30 TB of bandwidth, which would cost almost $3,000.
    - There are unmetered VPS offerings outside of EC2/GCE/Azure, but we still have the nontrivial problem of storing several terabytes of data for the test.
- I consider the chances of interference to be low
    - We are not measuring throughput, so changes to available RAM, CPU, or network bandwidth within the test environment should not affect the results of the test.