University of Zurich[UZH]

Exercise solved by:

Remy Egloff
Eleonora Pura
Jan Willi
Stefano Anzolut
Cédric Merz

Department of Informatics IfI
University of Zurich
Binzmühlestrasse 14
CH—8050 Zürich
Switzerland
URL: https://www.ifi.uzh.ch

Instructors:
Prof. Dr. Alberto Bacchelli
Prof. Dr. Burkhard Stiller
Prof. Dr. Christoph Lofi
Prof. Dr. Harald Gall
Dr. Marco D'Ambros
Assistants:
Dr. Bruno Rodrigues
Alexander Lill

Fundamentals of Software Systems (FSS) 03SMMINF4569

Distributed and Communication Systems E01: Practical Exercise on Serialization and IPFS

1 InterPlanetary File System (IPFS)

IPFS is a distributed system that allows storing and accessing files, Websites, applications, and data [1]. IPFS was designed to provide a distributed alternative to the widely used Hypertext Transfer Protocol (HTTP), which is the baseline of existing Websites [2]. An issue of the centralized nature of HTTP-based Websites is that they suffer from limited bandwidth to serve requests of multiple users, as highly susceptible to Distributed Denial-of-Service (DDoS) attacks. In contrast, IPFS distributes the hosted content across multiple nodes belonging to the peer-to-peer network in a way that is possible to retrieve pieces of the content from various nodes.

2 Decentralized Storage on IPFS

IPFS is capable of providing a lower latency time for retrieving large files than a traditional HTTP server. Such advantage stems from the capacity to distribute chunks of a single file into different peers and retrieve them simultaneously, instead of retrieving from a single server as in HTTP. However, depending on the file size and performance of the centralized server, HTTP may still present a better performance in contrast to IPFS once it may be required to split large files into multiple blocks. Additionally, serializing allows reducing the size of files, resulting in more efficient storage and transmission.

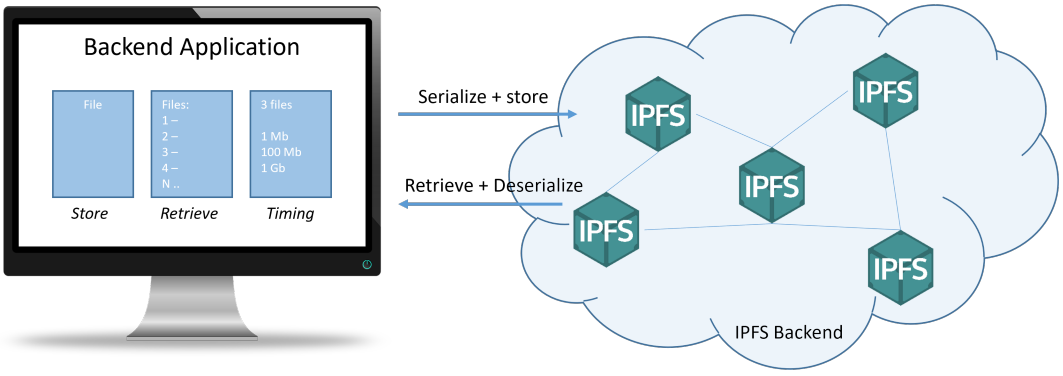


Figure 1: Application Scenario

2.1 Goal

The goal of this practical exercise is twofold: (1) to measure and compare the performance of storing and retrieving files in a centralized and decentralized manner, and (2) to observe the impact of data serialization in the performance.

2.2 Description and Key Requirements

Figure 1 illustrates the application scenario in which a backend service implements the file storage and retrieval functions. IPFS is the backend storage, whereas files should be decentralized upon storage (*i.e.,* store files in third-party peers rather than locally). Performance measurements (in milliseconds (ms) must consider the time elapsed to store (including serialization) and time elapsed to retrieve and deserialize. **Key requirements** are determined as follows:

1. Files must be serialized upon storage and deserialized upon retrieval. **Two** different serialization algorithms should be implemented [3].

2. Files on IPFS should be stored at other peers (not available locally as that would impair the decentralized storage).

3. Performance measurements must consider the (a) time elapsed to serialize and store and the (b) time elapsed to retrieve and deserialize. Plot the average of **10 runs** each serialization algorithm.

4. Similarly to the previous requirement, a picture **or** a video (*i.e.,* unstructured data) must be serialized/stored to observe the performance of serialization on different media types. For instance, **Pinata** uses IPFS as the backend to store NFT's (Non-fungible Tokens) off-chain. Differently of the previous requirement, plot only the average of **5 runs** for each serialization algorithm.

5. Compare the IPFS time to retrieve files with HTTP. Unstructured (and flat data) test files of 1 Mb, 100 Mb, and 1 Gb are hosted at SWITCH and must be used to compare with IPFS results. Measure the time taken to **retrieve/download**

these files and compare with IPFS retrieval. A hint is to measure the serialization/deserialization time separately of the entire storage/retrieval process. Plot the average of **5 runs/downloads**.

3 Details on the Performance Measurement

Elapsed times must be measured at application to evaluate the IPFS performance. For example, a timer must be started at the application upon request to store/serialize a file and concluded when the file is stored. Similarly, a timer must be initiated upon receiving a request to retrieve/deserialize and end when the file is ready locally.

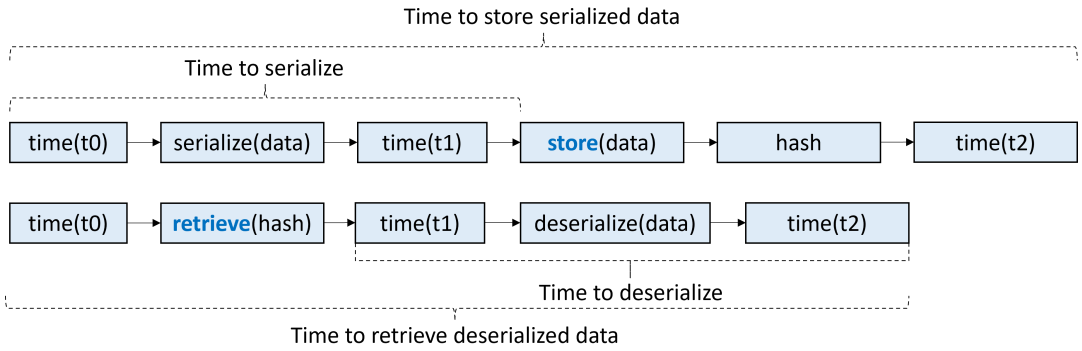


Figure 2: Measurement of Elapsed Time

- 1 Gb file: **Direct link**.
- 100 Mb file: **Direct link**
- 1 Mb file: **Direct link**
- Test files are available **here**.

4 Useful links/hints

- Simplicity is key, no need to fancy your application with added complexity.
- You will need a command-line installation of IPFS to build your application on top. Take a look **here**.
- IPFS official documentation is available **here** [1, 2].
- There are cool tutorial on how to use IPFS. For example, check **here** and **here** [4, 5].
- A quick guide on serialization in Python is available **here** [3].

4.1 Discussion Points

- Which serialization algorithm performs better, and why? Algorithms may perform differently based on the type of data at hand (*i.e.,* structured, semi-structured, unstructured). In this sense, developers need to identify what types of data the application handles to determine which algorithm is most suitable.

  The csv serialization performed worse than pickle. The input file is unstructured data, therefore we decided to add a comma at the end of each line to get a csv file. To do this, we have to loop over all lines of the input file. We assume that pickle is much more optimized compared to this iterative approach. Therefore, the csv serialization took longer than pickle.

- Aligned to the previous discussion point, what is the impact on file size (in percentage terms) for the test files? Is it possible to conclude that serialization is particularly advantageous for relatively large files (*e.g.,* 1 Gb)?

  Applying pickle serialization and csv serialization increase the file size. In case of pickle more than csv. For larger files, we were only able to serialize the file using pickle, as our csv serialization approach took forever. Therefore, this question remains unanswered.

- At what point (if any) does the decentralized storage perform better than centralized? As previously mentioned (*cf.* Section 2), depending on factors such as the size of the file to be stored and the performance (involving several factors) of the server, the centralized approach can be more advantageous. For example, for relatively small files (*e.g.,* 1 Mb), the use of a decentralized server may represent an unnecessary overhead, but for relatively large files (*e.g.,* 1 Gb), there may be significant performance gains in file retrieval.

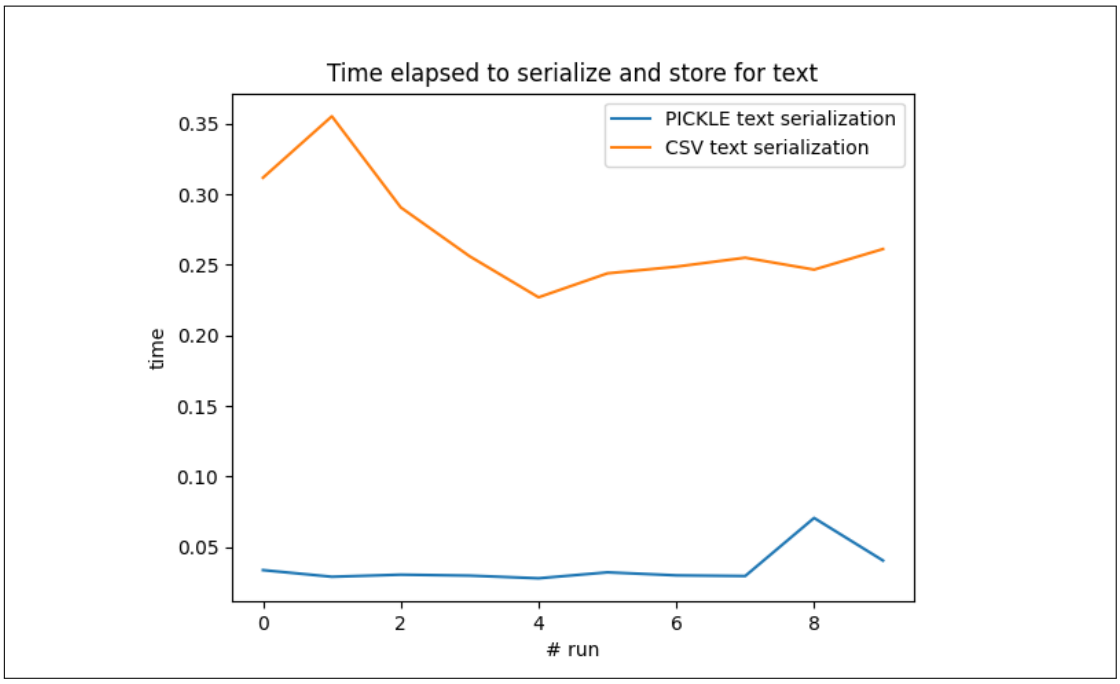
For the 1Mb file, the centralized approach was faster than the decentralized approach due to overhead. To download the 1Gb file in the decentralized approach took forever in our case. But from what we have learned, the larger the file is, the better suited the decentralized approach should be. However, several factors, like the server load could affect the results.

- One of the great advantages of HTTP is its extensive documentation as it is an ”industry-standard” protocol. In this sense, how does the group evaluate the experience using IPFS (1-10) considering aspects like available documentation and implementation experience?

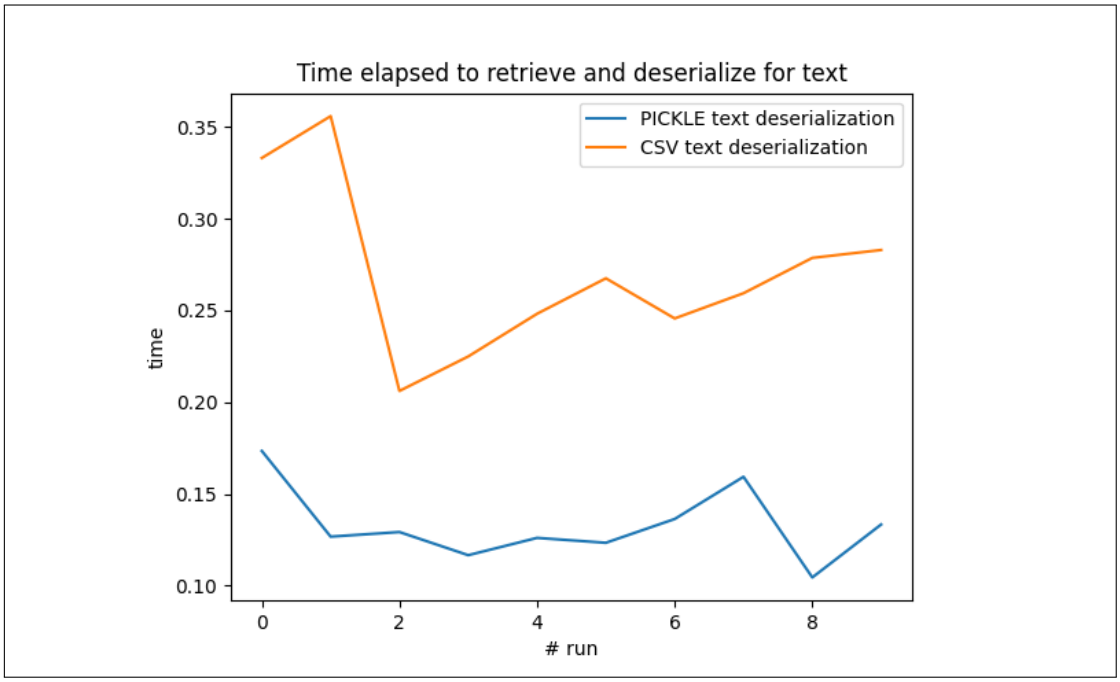
  Rating: 7/10

  The approach is well documented and the installation was easy. However, the used APIs in python were outdated and not so well documented. We had to try out multiple versions until it finally worked. What was further helpful is that even if it is an experimental approach, we were able to find multiple resources discussing it on the internet. However, we perceive IPFS still as kind of a "blackbox". As a user, it is not so clear what happens after adding a file to the network, as this process is not transparent.

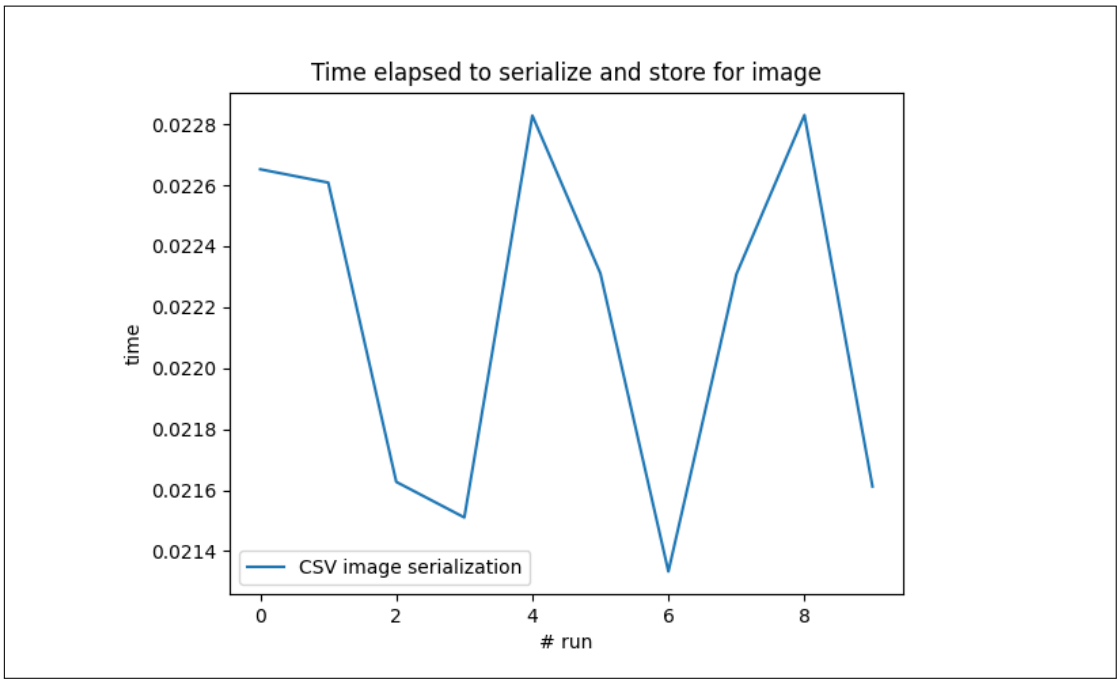
- Time elapsed to serialize and store (combined plot comparing two selected algorithms):



- Time elapsed to retrieve and deserialize (combined plot comparing two selected algorithms):



- Time elapsed to serialize and store a picture or image (combined plot comparing two selected algorithms):



Submission Guidelines

- This task must be realized in groups of **4 or 5** students.
- The source code must be submitted until: **18.10.2021** *i.e.,* until midnight of Monday.
- Any submitted code must compile without error. Include the name of participants in a separate ”.txt” file within the submitted ”.zip”.
- Submit assignments via OLAT file-drop folder. Only one submission per group.
- Zip your files using the name pattern: FSS_Ex+ExerciseNumber_Initials (*e.g.,* FSS_ex01_BR.zip)

References

[1] Juan Benet. IPFS-content Addressed, Versioned, P2P File System. *arXiv preprint arXiv:1407.3561*, 2014.

[2] InterPlanetary File System. IPFS documentation. https://docs.ipfs.io/, September 2021. Accessed: 2021-09-21.

[3] Kenneth Reitz and Tanya Schlusser. The Hitchhiker's Guide to Python: Best Practices for Development. https://docs.python-guide.org/scenarios/serialization/, September 2021. Accessed: 2021-09-26.

[4] Matt Zumwalt. The IPFS Primer. https://flyingzumwalt.gitbooks.io/decentralized-web-primer/content/, September 2021. Accessed: 2021-09-21.

[5] Kojo Addaquay. What is IPFS? - A Beginner's Guide. https://hackernoon.com/a-beginners-guide-to-ipfs-20673fedd3f, September 2021. Accessed: 2021-09-21.