Proceedings of the 34th European Modeling & Simulation Symposium (EMSS), 043
19th International Multidisciplinary Modeling & Simulation Multiconference

ISSN 2724-0029 © 2022 The Authors.
doi: 10.46354/i3m.2022.emss.043

Efficient handling of lots of simulation data files

Roman Diviš[1,*], Zdeněk Novotný[2]

[1]University of Pardubice, Studentská 95, Pardubice, 532 10, Czech Republic
[2]University of Pardubice, Studentská 95, Pardubice, 532 10, Czech Republic
[*]Corresponding author. Email address: roman.divis@upce.cz

Abstract

Saving information to files is the most basic and simplest way to store data, so it is often used in simple simulators and simulation tools as the first choice for logging information about the simulation process and its results. Computer simulations often involve simulating a significant number of replications and accumulating large numbers of files. Today's filesystems are still not capable of efficiently storing and processing millions of files. This paper presents alternatives that allow for more efficient storage, transfer, and analysis of data, with an emphasis on easy migration or implementation from the initial data files. Simple approaches such as using TAR or ZIP archives to sophisticated approaches involving Parquet file, S3-like object storage (e.g., MinIO, OpenIO) will be compared.

Keywords: data files management; data storage; object storage

1. Introduction

Computer simulations are commonly used in the examination and analysis of various dynamic systems. Simulators can directly provide (i) statistical evaluation of measured data in the simulator itself, they can (ii) generate summary output files (with statistical evaluation) or simply provide (iii) natural (raw) measured data about the simulated system (used for later statistical evaluation).

Even in cases (i and ii) when the simulator does statistical evaluation itself, the simulator usually need to save measured data for the process of statistical evaluation. Simulators can hold necessary data in operating memory but because of their size it may be infeasible to hold all the required data for processing simulation, and recorded data in memory at once. So, these data are usually saved to external storage (such as files in filesystem/data store, database, specialized data logging tools, ...). These data for example consist of full saves of simulation states, simulation records of watched entities/values or already pre-processed summary values. In general terms, it is a heterogeneous dataset that has a similar structure for every simulation run (replication). For convenience, we will call this data a "simulation log".

There are many different options used for saving of simulation log to external storage:

- (Custom) plain text files
- (Custom) binary files
- Formatted/serialized text files (common formats such as CSV, JSON, YML, ...)
- Common serialization binary files (Python – Pickle, BSON, Parquet, ...)
- Archive (compressed files (TAR, ZIP, 7Z, ...)
- Relational database systems (MySQL, Maria DB, PostgreSQL, ...)
- Nonrelational database systems (MongoDB, Cassandra, ...)
- Block or other specialized storage systems (S3, ...)
- Specialized data storage software

1.1. Pros and cons

Each of these options has its advantages and disadvantages and a complete review and comparison of these options are out of the scope of this article. The main scope of this article is to address the implementation of simulation logs in custom-made simulation tools. These simulation tools can be also connected to time-limited projects and their analysis, design and development happen quickly, and the main focus is to provide the required functionality (simulate the defined system, analyze and provide results). So, the proper planning and design of every part of the simulator is not the main concern. This can result in using of suboptimal subsystems and handling of simulation logs is one of them. Problem is typically hidden for a later phase of the project. Until the simulation log is needed and processed or until its discovered that there are missing data/features (and they need to be added to the simulation log), developers can imagine that everything is correct and optimal.

1.2. Files versus database

For these and probably many other reasons, the text/binary files are often the first choice for the implementation of the simulation log. Natural text/binary (raw) files are simple, extendable, have natural support in programming languages and their format is given by the developer.

In simple comparison with relational database (such as MySQL) – developer often needs to install, configure and run database system, then download/link external library/dependency, learn specific API for accessing the database system and its data, design structure of tables and map natural data (objects, structures, ...) to the specific ones in the database system. This process adds many steps and planning before the developer can save any data to the database. Also, the process of adding new data is slowed down by necessary updates to table structures.

In so-called NoSQL databases, the developer may not be required to define table (storage) structures before the insertion of data. So, this downside is not present in this case while others remain. Also, the absence of structure can be problematic in future when new data needs to be added and the developer needs to migrate old data to the new format.

Various commercial simulators often use their own (proprietary) binary data files or common database systems. As both these options can provide good results in terms of speed and size of the simulation log.

1.3. Problems with suboptimal formats

Suboptimal format of simulation log, in general, leads to problems with (i) big size of log, (ii) slow access times (read and write), (iii) inability to extend simulation log, (iv) incompleteness of data or missing version information.

The first problem is connected to the usage of text format and specifically formats with excessive overhead (such as XML format). Binary files can be smaller and faster. The second problem - read speed – can be caused by a big number of poorly organized files, the ineffective internal organization of files or the inability to seek specific data in files (text files without index). On the other hand, the inability to extend the simulation log (or inflexibility to do so) can be caused by custom-made binary formats or by using formats and systems that require defining data structure before use. The last problem – incompleteness or missing information – is often found later when many data are already produced without proper organization and labelling, and errors and missing information leads to an inability to distinguish between individual datasets and configurations of simulations.

This article focuses on the custom-made simulation tools that probably use a simple method for handling the simulation log and are considering a better approach. Considered options may be used in batch mode – post-processing the original simulation log to the new format or in online mode – direct implementation of the new format of simulation log to the simulation tool. The main focus is also given to simple yet effective variants for the handling of the simulation log so that the process of migration is short.

2. A brief overview of state-of-the-art

The topic of different variants of data storage, simulation log format or usage of various database systems is quite well discussed. Unfortunately, most of these articles focus only on a very narrow field of simulation or the comparison itself is limited to very specific cases. For example, in (Ng et al., 2004) authors focus on biomolecular simulation data and consider DB2, netCDF and Python Pickle options. Another article (Buyl, Colberg & Hofling, 2014) proposes a new structured file format for molecular data. In the thesis (So, 2016) author compares Parquet and JSON files in the case of the Particle Swarm Optimization algorithm.

Some articles are discussing underlying optimizations and not how to directly optimize stored data. As an example of optimization of an underlying database management system and its own data storage mechanism, we can refer to (Alagiannis et al., 2015) where authors optimize PostgreSQL storage and offer a comparison with other DBMS. A different example of underlying optimization is articles that refer to new and optimized file systems implementations, for example in (Ovsiannikov et al., 2013) authors present an alternative to Hadoop distributed file system.

3. Case study

For our case study, the MesoRail (Diviš and Kavička, 2015) simulation tool is used. Its custom-made simulator is written in Java programming language. The MesoRail simulation tool is specially designed to perform railway traffic simulations on the mesoscopic level of detail. Its main task is to serve in support of examination of the throughput/capacity of rail system infrastructures (especially railway stations and nodes). The simulator supports deterministic and stochastic traffic flows and implements several methods for conflicts that happen during stochastic simulations. Mainly it implements the reflective nested simulations method that uses recursive simulations with limited lookahead and different parametrization for evaluation of the best solution for a given conflict. This process results in the running of many simulations and for detailed analysis, it needs much data saved in the simulation log.

As a custom-made simulator for implementation of simulation log was chosen to use simple text and binary files and a few specific formats. For every simulation replication, there is generated 8 files plus the number of simulation state dumps for every conflict that happened in simulation. Simulation log files for one replication consist of (sizes of files can vary, presented values are average values for current simulation experiments):

- user readable simulation log with details of parametrization – text file, size around 10 kB,
- nested simulations log – text file, CSV like, size around 2-3 MB,
- train delays analysis – text file, CSV like, size around 1 kB,
- simulation executor log – text file, size around 30 kB,
- simulation run time – text file, size around 5 B,
- nested simulation graph – GraphML file (text-based XML), size around 50 kB,
- railway tracks occupation chart – SVG file (text-based XML), size around 70 kB,
- simulation conflicts log – text file, CSV-like, size around 5 kB.

And for every evaluated conflict there is:

- simulation state dump file – binary file, custom binary serialization, size around 200 kB.

In total there are 20 to 30 files per simulation replication.

For evaluation of the aforementioned methods, we gathered simulation logs from 10 000 replications:

- Total - 195 863 files, 38.6 GB,
- Text file logs – 78 166 files, 20.4 GB,
- Simulation state dump (binary) files – 117 697 files, 18.2 GB.

4. Format comparison

As mentioned before, the focus of this article is on simple yet effective methods that can be added to already existing or new simulation tools. For comparison we've chosen the following formats:

- original data,
- ZIP archive,
- TAR archive,
- Pickle serialization format,
- Parquet serialization format,
- S3-like object storage MinIO.

For each of these variants, we measured and reported:

- size of stored data,
- read speed (random access),
- brief description of the pros and cons of the method.

For every suitable format, we tried to make an archive file per replication, per 100 replications, per 1000 replications or per 5000 replications. In RAW terms file counts and sizes are shown in table 1.

Table 1. File counts and sizes for chosen scenarios

Dataset size	File count	Total file size (GB)
1	25	0,01
100	2045	0,42
1000	21126	4,32
5000	83730	17,10

For every considered format we benchmarked the creation of archive files and then tried to perform many random access reads.

It is necessary to mention that the main target of our article is to evaluate simple implementations. This leads to suboptimal implementation and performance of several considered formats. We want to find the best simple solution and we know that optimal implementation of any of mentioned formats may lead to much better results. Unfortunately, optimal implementation can be time-consuming and possibly limits the further expansion of stored data in the simulation log. Basically, every considered data format is filled with the natural format of presented text and binary data files, instead of parsing data and transforming them into much better data representation.

Gathered results of random access read speeds are presented in table 2. For every chosen data format we created archive files according to the presented sizes of datasets and then performed a series of random access reads of stored files.

Table 2. Measured random read speeds for benchmarked scenarios

Read speed (MB/s) Dataset size	RAW	Zip	Tar	Pickle	Parquet	MinIO
1		253,9	96,2	72,1	28,0	192,1
100	260,8	926,7	874,7	924,2	470,3	196,8
1000	205,4	425,4	112,9	79,4	33,1	190,3
5000	171,8	240,1	17,2	11,6	0,5	47,9

4.1. Benchmark results

From performed benchmarks, we measured best read speeds usually when we made archives of 100 replications (about 2000 files with a size of about 0,5 GB). And for extremely big archive files performance dropped significantly, only in the case of MinIO (S3-like storage) performance was quite stable, dropped only in the biggest scenario.

Size of resulting archives we're usually. Only the Parquet format performed better than other options and its size was about ¼ the size of other formats. But formats like Zip offer different compression levels and thus may lead to smaller archive sizes.

MinIO was used in a single-node server scenario without the utilization of cluster storage features or additional options such as compression.

Parquet format can offer better results with a more focused implementation of this data format. Storing of RAW files is suboptimal and leads to decreased performance.

The overall best performance was measured in Zip format and Pickle (Python binary serialization) format. As Zip format is supported by most programming languages and operating systems it's the best choice of our benchmark for archiving simulation logs.

5. Conclusions

We proposed simple options to enhance implementations of the so-called simulation logs. This topic is mostly useful for authors of custom-made simulation tools where much development time is needed to create the simulation itself instead of optimising of simulation log.

After consideration of several simple archive formats and then benchmarking them, we can say that a simple way of the utilization of Zip archives of carefully chosen size can lead to a big benefit in terms of storage size needed, read speed of simulation logs, transfer speed of simulation logs over the internet or long term storage of simulation logs.

Of course, for specific scenarios, other users may find different formats more beneficial. Also, specific formats for data storage (such as Parquet) with correct usage may yield even better results. Our focus was on finding a simple yet effective way of enhancing handling of our simulation logs.

Acknowledgements

The work has been supported by the Funds of the University of Pardubice, Czech Republic. This support is very gratefully acknowledged.

References

Alagiannis, I., Borovica-Gajic, R., Branco, M., Idreos, S., Ailamaki, A. (2015). NoDB: Efficient Query Execution on Raw Data Files. J. Commun. ACM,12:112-121.

Buyl, P., Colberg, P. H., Hofling, F. (2014). H5MD: A structured, efficient, and portable file format for molecular data. J. Computer Physics Communications, 6:1546-1553.

Diviš R., Kavička A, (2015). Design and development of a mesoscopic simulator specialized in investigating capacities of railway nodes. Proceedings of the European Modeling and Simulation Symposium, 52-57.

Ng, M. H., Johnston, S., Murdock, S., Wu, B., Tai, K., Fangohr, H., Cox, S., Essex, J. W., Sansom, M., Jeffreys, P. (2004) Efficient data storage and analysis for generic biomolecular simulation data. 3rd UK e-Science Programme All Hands Meeting (AHM 2004), Nottingham, UK, 443-450.

Ovsiannikov, M., Rus, S., Reeves, D., Sutter, P., Rao, S., Kelly, J. (2013). The Quantcast File System. Proc. VLDB Endow, 11:1092-1101.

So, G. (2016). Comparing Performance of Parquet and JSON Files for a Particle Swarm Optimization Algorithm on Apache Spark. Ms. Thesis California State University, Northridge.