Simple Data Architecture Best Practices for AI Readiness

Dr. Vijay Gadepally & Dr. Jeremy Kepner - MIT

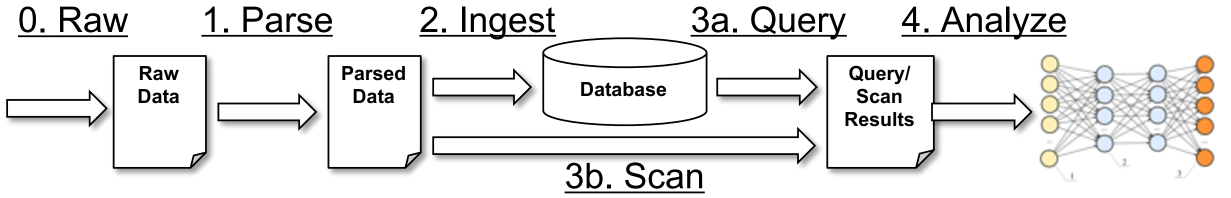

Figure 1: Standard data collection and management steps for AI

AI[1] requires data. A core requirement for AI techniques to be successful is high quality data. Hence, preparing systems to be "AI Ready" involves collecting raw data and parsing it (steps 0 and 1 in Figure 1) for subsequent ingest, scan, query, and analysis (steps 2, 3, and 4 in Figure 1). There are simple techniques that can be applied during initial parsing of raw data that can dramatically reduce the effort of applying AI. This parsing is much more efficient to do during initial collection setup when the knowledge of the data exists with the programmer. Requiring an AI analyst to later deduce this knowledge is a primary reason why "data wrangling" is often 80% of the effort in building an AI system.

This document provides a short list of a few of these best practices, which can be summarized in one word: *tables*. In particular, wherever possible, parse as much data as is practical into tabular files. Try to avoid proprietary formats. If possible, use *.csv* (comma separated values) or even better *.tsv* (tab separated values) file formats. Include column labels (each column label should be unique within the file). Have row labels in the first column, such as row number or record number (each row label should be unique within the file). In cases where the number of columns might be large and lots of entries in each column might be empty (i.e., the data is sparse), it can be more efficient to write out data in a triple format where each non-empty entry is a row in the .csv/.tsv file as follows:

```
row, column, value
```

The triples format also encompasses the popular key-value pair model.

Avoid lots of tiny files and compress with zip when possible. Use hierarchical directories to keep the number of items in a directory reasonable (<1000). Use well-defined directory structures to provide easy to access information about the dataset. For example:

```
source/YYYY/MM/DD/hh/mm/source-YYYY-MM-DD-hh-mm-ss.tsv
```

or

```
YYYY/MM/DD/hh/mm/source/YYYY-MM-DD-hh-mm-ss-source.tsv
```

Databases and files can often be used together. Use databases if you need to quickly find particular data items (i.e., you are looking for a small number of records when compared to the entire dataset). SQL systems are good for medium size datasets that require ACID (atomicity, consistency, isolation, and durability) guarantees. NoSQL systems are good for large datasets where ACID guarantees aren't required. NewSQL systems are good when you have a need for scalability + ACID compliance. Scanning files in the file system can be best when reading in a majority of a dataset. Given that AI is trying to predict outcomes, any outcome data that can be recorded (in any format) is extremely helpful.

[1] For the purpose of this short guide, we refer to AI synonymously with statistical Machine Learning (ML)

Appendix: Why Tabular?

There are several benefits of placing data in a tabular format.

AI. Most AI algorithms, such as, deep neural networks (DNNs), use matrix mathematics to perform their computations. Data in a tabular format is easily read into a matrix. Most AI tools support reading .csv/.tsv files for this purpose.

General Analysis. Most data analysis tools provide support for reading .csv/.tsv files. Python, R, Julia, and Matlab programmers often call tables dataframes and the machine learning (ML) community often refer tables as tensors.

Databases. Databases are ideal for sorting data and enabling fast retrieval of specific records. Most database store data in tables and support reading .csv/.tsv files.

Debugging. Log files can be very helpful for debugging applications. Raw logs are often difficult to view and analyze. Placing data in a tabular .csv/.tsv allows the programmer to quickly view a log with Microsoft Excel or other spreadsheet tool.

Incidence Response. Security compliance guidelines (such as the RMF risk management framework), recommend/require that logs be kept. If logs are in a readily analyzable tabular .csv/.tsv format they can play an important role in accelerating the containment assessment phase of an incidence response.

Appendix: Serial/Parallel Files/Databases

In Figure 1, we have two alternatives (3a. querying a database and 3b. scanning of files) that can be used together to access data. Figure 2 gives a high-level overview of when to use files vs. databases on a serial or a parallel (distributed) system. Usage of files or databases is often determined by how much data is desired (data request size) and where the total data volume physically fits: from the memory of a single (serial) processor to many (parallel) storage devices.

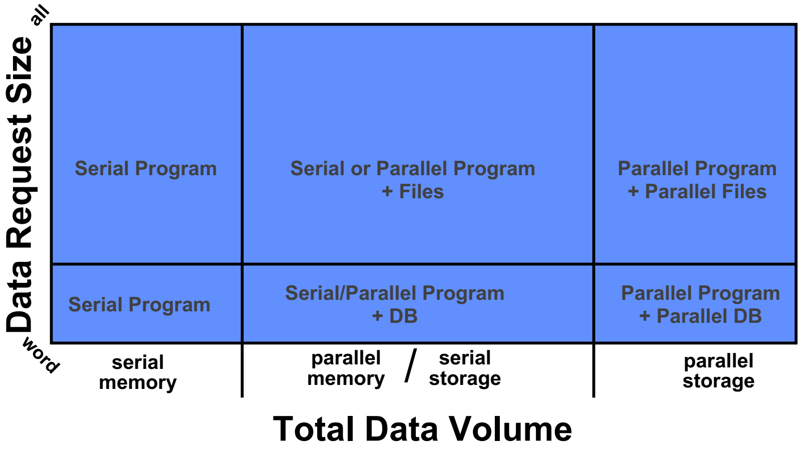
Data Request Size
all
word
Serial Program
Serial or Parallel Program + Files
Parallel Program + Parallel Files
Serial Program
Serial/Parallel Program + DB
Parallel Program + Parallel DB
serial memory
parallel memory / serial storage
parallel storage
Total Data Volume

Figure 2 Files and databases can be used together. As a high-level rule of thumb, retrieving >10% of a full dataset, may be well-served dealing by scanning files. For very large datasets, a parallel program leveraging distributed storage can accelerate this scanning.

Additional References

Mathematics of Big Data: Spreadsheets, Databases, Matrices, and Graphs
https://mitpress.mit.edu/books/mathematics-big-data

Data Management, Polystores, and Analytics for Healthcare
https://link.springer.com/content/pdf/10.1007/978-3-030-14177-6.pdf

AI Enabling Technologies: A Survey
https://arxiv.org/abs/1905.03592