Little, Medium, and Big Data: Choosing Data Tools for Social Science Research

Stephen P. Tarzia
Northwestern University
September 2017

***Abstract*—This article provides guidance for social science researchers navigating the overwhelming variety of data analysis tools now available. There is no one-size-fits-all solution, and it's important to understand the capabilities and drawbacks of each option. *Big Data* is often defined by the "three V's" of velocity, volume, and variety. This article provides some quantitative guidelines on data *velocity* and *volume* to allow researchers to match their data problems to the appropriate tools**

To clarify the meaning of *Big Data*, I introduce the terms *Little Data* and *Medium Data* to describe two distinct types of data analysis that are possible with a single machine. Generally, techniques designed to handle larger data sets come with drawbacks inherent to parallelism and decentralization. So, one should not automatically choose the latest and greatest Big Data tools, but instead choose an approach that is sized just right for a particular problem. I provide a simple flowchart in Figure 1 to guide the choice of computational tools. The article gives researchers a basic understanding of the performance limitations of a single computer to recognize instances where data velocity and volume call for Big Data tools.

I. Little Data

The simplest research scenarios involve data sets that are small enough to fit in a single machine's random access memory (RAM), and I call this "Little Data." In 2017, a top-of-the-line laptop has 16 GB of RAM and most servers can be provisioned with up to 1.5 TB of RAM. Any of the data residing in a machine's RAM can be quickly read or written by its central processing unit (CPU).[1]

With Little Data, analysis programs can access the data files just once to load everything into RAM, typically as arrays, then proceed to the analysis steps with all the data readily available. This represents a baseline case for computing, and we will see that data become much more dispersed in the Medium and Big Data paradigms.

In my past experience as a computational research consultant, I found that most academic social science research problems can be handled with Little Data tools, particularly if the university provides access to a shared server with lots of RAM.

Examples of Little Data tools include Excel, Stata, and Matlab. Programs that researchers write in general-purpose programming languages like R or Python most often also use a Little Data approach, but the following section explains some ways to expand the scale of your custom-written programs.

II. Medium Data

"Medium Data" techniques are needed when a data set surpasses RAM size. In these cases, most of the of data must remain in the storage system during analysis. By "storage" we usually mean the machine's hard disk drives, which are much slower than RAM.[2]

There are several common strategies for analyzing data larger than RAM size:

Streaming

Streaming involves doing an analysis while scanning through the data file(s). For example, one may search for instances of a certain keyword in a 10TB log file by using the `grep` command on Unix-based systems, and this will not require much RAM because the data can be discarded immediately after

[1]This is a slight simplification. CPUs have multiple levels of memory caching that effectively makes some RAM more quickly accessible at any given time, but I ignore this effect because such caching is unpredictable and is generally does not affect programmers' design choices.

[2]Disks are "slower" than RAM in two ways. There is a greater delay between requesting a piece of data and receiving it (known as *latency*) and fewer data bytes can be transferred in a given time period (known as *throughput*). Disks using solid state technology are significantly faster than those using rotating magnetic disk technology, but they are still slower than RAM.

it is read. Streaming can work well for filtering, but more complex analyses are intractable because cross references are impossible. In other words, streaming works when you need to examine all the data once, but it affords no way to quickly find particular pieces of data.

File-based indexing

File-based indexing involves splitting the data into many files and using a file naming convention or an index file to determine which file holds the particular data you are interested in. Let's imagine that you're analyzing historical data on stock market trades. The data could be organized into folders, one for each year, and within each folder there could be a file for each company (having a filename corresponding to the company's stock ticker symbol or another well-known identifier, such as the CUSIP). Under such a scheme, a program can quickly determine exactly which of the thousands of files contains the data of interest.

Relational databases

Relational databases are the ultimate Medium Data tool because they allow the researcher to index the data in an arbitrary number of dimensions. You can think of a relational database as an alternative to a filesystem. Both are used to store data permanently on disk. They differ in how data is organized and found. Filesystems are organized by a hierarchy of named folders and files, and each file contains a chunk of data. On the other hand, relational databases organize data into one or more tables (having rows and columns). Any of the columns can be indexed, so you can efficiently retrieve table cells using a variety of criteria.

Let's revisit the stock trading data example from above. A relational database would allow you to efficiently look up trades for certain years, or companies, or numbers of shares, or prices, or time of day. By contrast, if you were relying on file-based indexing you have just the filename to work with so you could not index along all five of these dimensions. Explaining all the capabilities of relational databases is beyond the scope of this paper, but there are many good books and online tutorials on the topic.

A downside of relational databases is that they are less portable than simple files. Unlike files, a database cannot be easily moved from one system to another (eg., from Mac to Windows).[3] On the other hand, there are many different types of free and commercial relational databases and they all work slightly differently (eg. MySQL, PostgreSQL, Microsoft SQL Server, Oracle, SQLite). Because of this variety, there is usually some effort and expertise required to load a given data set into the particular relational database. One has to define the tables, the columns' data types, the relationships between the tables, and then load the data from a simpler format, such as spreadsheets in the comma separated value (CSV) files.

One can do many kinds of analyses using the relational database itself, by executing code written in standard query language (SQL). However, most researchers will want to use their preferred statistical analysis program. Fortunately, these can all load data from a relational database in a similar fashion as you would load data from a file. The difference is that you would write queries in the SQL language to retrieve aggregations or subsets of the data. In this way, you can quickly access the particular data you need while leaving most of the data untouched on disk.

SAS

Unlike other popular statistical analysis programs, SAS is designed to handle Medium Data problems. Tables in SAS actually function very much like relational database tables. They are stored on disk, and they can be indexed by multiple columns.[4].

A very common Medium Data workflow in the social sciences is to preprocess the full data set with SAS, export a slice of the data to a file, then load it into Stata for the final analyses. If the data were originally loaded into a relational database, then a simpler workflow could be used. All the analysis could be done from Stata by using SQL queries to retrieve the data cells of interest.

[3] An exception is the SQLite database, which conveniently stores a relational database in a single portable file.

[4] http://www2.sas.com/proceedings/sugi29/123-29.pdf
http://www2.sas.com/proceedings/sugi30/008-30.pdf

Swapping

If you ignore all the options above and try to apply Little Data methods to a Medium Data problem, your machine will experience a shortage of memory and start swapping. Swapping is a process by which the operating system (OS) makes "virtual memory" available to the program by automatically shuffling data between RAM and disk. Swapping is generally much less efficient than other Medium Data approaches; the OS must guess which data to "evict" to disk without any knowledge of when that data will be accessed again by the program, and this leads to more disk activity than is strictly necessary. Swapping does extend the capacity of Little Data tools, but analyses that rely on swapping can be painfully slow.

III. Pushing the limits of Medium Data

Focusing on and velocity of data, we can think of Big Data as the set of problems that require reading or writing data at a faster rate than is possible on a single machine. It's much more difficult to program a distributed system, so it's worth asking just how much data processing capacity can we achieve on a single machine. In other words, how far can we scale the Medium Data approach?

It's somewhat tricky to calculate the threshold between Medium and Big Data. Let's start by estimating the data processing capacity of a single machine; an upper-bound estimate for this is the maximum sustained rate at which the CPU can read or write data from its storage system.

A server or workstation-class CPU in 2017 can read data from storage using up to 40 PCI Express 3.0 lanes, which limits it to pulling in at most 40 GB[5] per second of data.[6] We'll call this the input/output (IO) rate of the CPU. On the other hand, the speed of the storage systems can vary dramatically from one machine to another.

The slowest common storage option would be a single high-capacity 7200 RPM magnetic disk, which might transfer about 150 MB per second at best. Solid-state disks (SSDs) have less capacity but are about ten times faster. My Macbook Pro's 1 TB SSD can sustain 1.5 GB/s. A better choice for Medium Data work would be a system that is equipped with some kind of storage *array*. We can achieve increased capacity and higher transfer rates by distributing the data over multiple disks which can be accessed in parallel. Remember that the CPU can handle 27 times more data transfer than even the fastest disk (40 vs. 1.5 GB/s), so it makes sense to allocate multiple disks per CPU.

Storage arrays come in many different varieties with different costs, speeds, and capacities. I examined the following storage arrays:

- One of my personal servers has an array of eight 7200 RPM 4 TB SATA magnetic disks configured in Linux as a ZFS pool with two hot spares. This gives 24 TB of effective capacity and it achieves about 400 MB/s.
- A university-owned application server connected to two storage arrays, each of which has 24 Serial Attached SCSI (SAS) magnetic disks providing 24 TB of storage. There is also an 11.5 TB array built from SSDs. In my tests, each of these arrays achieved about 1 GB/s.
- A university-owned database (DB) server that was carefully optimized for maximum IO rate. It has eight RAID controller cards connected to 264 10k RPM SAS magnetic disks with 600 GB capacity each. The total capacity is around 150 TB and its advertised aggregate transfer rate is 16 GB/s. I was not able to test this system, but we can estimate the maximum throughput a user will experience. Assume that each of the 8 RAID controllers is responsible for controlling a one array of 33 disks (and assume two hot spares). This gives at most 18 TB of capacity and roughly 4.5 GB/s of throughput (31×150 MB/sec). The DB server's design demonstrates that hundreds of disks must be connected to make full use of a CPU's IO capacity.
- The university's Quest High Performance Linux Cluster, which has a 3.5 PB GPFS parallel file system using IBM's Elastic Storage Server technology. It's the fastest system that I measured, achieving 3.8 GB/s.

The systems are summarized below.

[5]Data measurements in this article are all in bytes, not bits.

[6]Usually some of these 40 PCI Express lanes will be reserved for networking and graphics, but we'll use 40 as an upper bound.

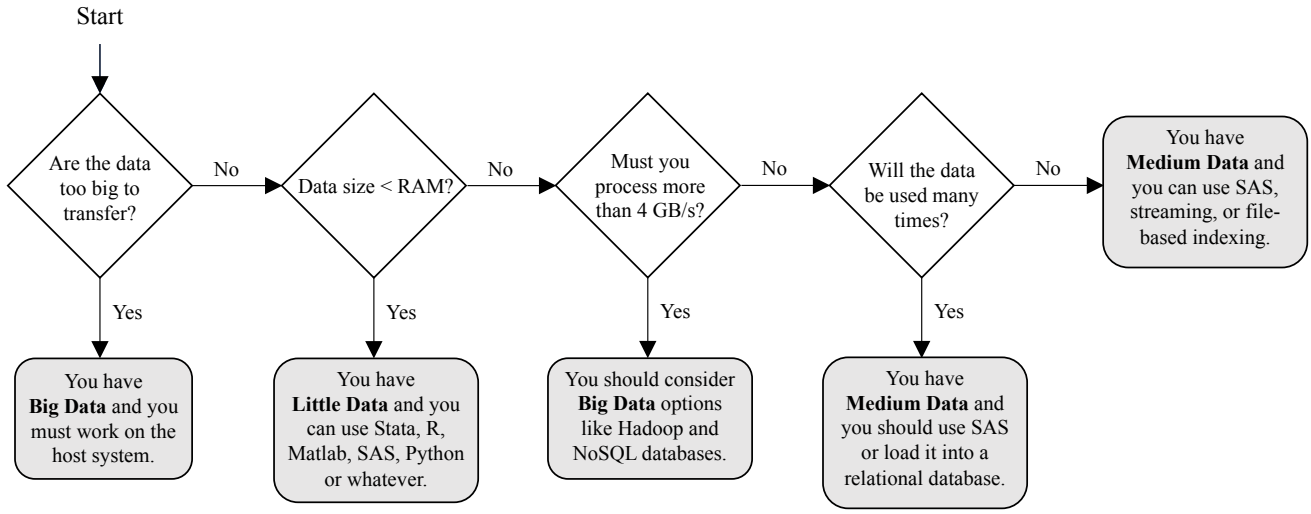

Fig. 1. A high-level flowchart to help the social sciences researcher choose tools to use for empirical research.

TABLE I
STORAGE PEFORMANCE EXAMPLES

System	Capacity per filesystem	Throughput
7200 RPM magnetic	8 TB	150 MB/s
Macbook Pro SSD	1 TB	1.5 GB/s
ZFS	24 TB	400 MB/s
App	24 TB	1 GB/s
Database	∼18 TB	∼4.5 GB/s
Quest/GPFS	80 TB	3.8 GB/s

IV. BIG DATA

As a starting point of our discussion, I defined Big Data as a set of problems that require processing data at a rate faster than a single machine can handle. Table I suggests that this threshold is currently about 4 GB/s. Social science research rarely passes this mark. Empirical academic research is usually a two-step process of data collection followed by analysis. Let's assume that the researcher can wait up to one week for the analysis to complete. Using a Medium Data approach on the DB server, an analysis code could access 2,200 TB of data in one week. I have never heard of a social science research project involving data on that scale.

On the other hand, it is possible that the *computational* capacity of a single machine will be insufficient to complete a social science researcher's analysis within that week-long deadline. In this case, some form of parallelism will be needed, but we call this high-performance computing (HPC), not Big Data. HPC systems, such as those at the United States' various national supercomputing facilities, have lots of memory and CPUs (and sometimes GPUs) but their ability to read or write data at a massive scale is usually a secondary design criterion. In fact, most HPC systems (eg., Northwestern University's Quest computing cluster) place the storage arrays in a separate system, apart from the compute nodes, which significantly limits the data processing rate.

Big Data problems often involve either processing massive streams of new data (as in particle physics experiments), or giving quick answers to queries involving huge amounts of distributed data (as in a web search). In both of these cases, the data is so large that it cannot reasonably be moved. Academic researchers will only encounter Big Data if they are granted access to the internal storage systems of a large organization generating such data. In these cases, there is little reason to debate what the best Big Data tools and strategies are; the researcher will be confined to using whatever tools are already supported for the organization's own use of the data.

So let's say you actually do have Big Data

Every approach to big data involves some kind of distributed, fault-tolerant storage system. This can be a distributed file system like the Hadoop Distributed File System (HDFS), Amazon's S3, or Google's proprietary Colossus (formerly called GFS). It can also be a distributed NoSQL database like Cassandra, MongoDB, DynamoDB, BigTable, or HBase.

To analyze your data you can use a parallel computing framework like Hadoop MapReduce, Spark, Pig, or Hive, or you can use an ad-hoc, divide-and-conquer approach in a general-purpose programming language like Python or R.

V. Conclusion

The various data analysis choices are summarized in a simplified flow chart in Figure 1. I hope that the preceding information will allow researchers to take full advantage of the plethora of robust Little and Medium Data tools without being unnecessarily distracted by the buzz surrounding "Big Data."

Appendix: Data hosting costs

Researchers, academic administrators, and support staff are often presented the opportunity to acquire large research data sets. Aside from the data licensing costs there are long-term storage and computational costs associated with large data sets. To correctly estimate such costs one must be familiar with the various data storage and processing options described in this paper.

If the flowchart in Figure 1 says that you're dealing with Big Data, then you'll have to purchase a cluster of machines, host them somewhere, and connect them all on a fast network. However, this would probably only apply if the researcher was somehow generating a huge quantity of data from an experiment. It's generally not possible to license or acquire data on the scale that requires Big Data solutions.

In the majority of cases (Little and Medium Data), the only costs are those associated with expanding the storage capacity of existing systems. For example, at Northwestern University this would mean either purchasing additional GPFS storage from central IT on for use on the Quest cluster or purchasing additional hard drives and enclosures for the application server or DB server. The Serial Attached SCSI protocol allows 255 disks to be connected to each physical connector using an *edge expander* and up to 65,535 disks through the use of *fanout expanders*. So, the only factor limiting the number of disks you can connect to a single machine is the data throughput you wish to achieve. Recall that a CPU is limited to 40 GB/s of PCIe bandwidth for input/output. The 264 disks already connected to the DB server are already capable of saturating that bandwidth:

$$264\,\text{disks} \times 150\,\text{MB/s/disk} = 38.7\,\text{GB/s}$$

So, adding more disks to the DB server would hurt its performance in the worst case scenario, when all disks are being simultaneously accessed. However, this is not a very likely scenario, especially if the majority of the stored data are are simply being warehoused rather than being actively analyzed. I do not have the access necessary to determine the level of input/output load typical of the DB server, but I'm speculating that it can tolerate further expansion. If another DB server were required, the cost of the machine itself would be roughly $2k to $5k. If a Windows environment is desired then an additional $882 would be required for a Windows Server 2016 SE license, plus $3,717 for SQL Server Standard license (up to 128 GB of RAM).

Next, we'll estimate the price of two different storage expansion options for the DB server: a high-performance and a low-cost option. This pricing exercise will use hardware similar to what was already purchased for the DB server's main and backup storage (high-performance and low-cost, respectively).

The high-performance storage option has 2.5 inch SSDs with 1 TB capacity, each costing about $300. They would be housed in a SuperMicro 88-disk JBOD chassis, costing about $3,000. If we reserve 16 disks for data redundancy, this leaves 72 TB of ultra-fast storage for $29,400.

The low-cost storage option has 3.5 inch 7200 RPM SAS magnetic disks with 8 TB capacity, each costing about $370. They would be housed in a SuperMicro 45-disk JBOD chassis, costing about $2,000. If we reserve 8 disks for data redundancy, this leaves 296 TB of storage for about $18,650.

Either of these storage arrays would occupy 4U of rack space and consume about 500 watts of power on average. We'll add another 500 watts for air conditioning costs. Assuming a charge of 8 cents per kilowatt-hour, the annual electricity cost is

$$1\,\mathrm{kW} \times 24\frac{\text{hour}}{\text{day}} \times 365\frac{\text{day}}{\text{year}} \times \$0.08/\mathrm{kWh} = \$700/\text{year}$$

Thus, if we assume full occupancy, the best-case cost to store data are shown below.

TABLE II
STORAGE COSTS PER TERABYTE

	fixed cost / TB	annual electricity cost / TB
high performance	$408	$9.72
lost cost	$63	$2.36

	annual cost / TB
NUIT Quest	$71
NUIT Research Data Storage	$137
AWS S3 standard access	$287 + transfer fees
AWS S3 infrequent access	$154 + transfer fees
AWS Glacier	$49 + transfer fees

These costs do not include any backups because I have assumed that the data is stored in read-only files and therefore is not susceptible to user error. For unique data sets it would be prudent to also budget for a backup.