Baruch MFE "Big Data in Finance" course ~ Quiz 1 ~ Model answers

Document name: "BDiF2015 – TN0007.pdf"
Date/revision: Fri 6th Feb 2015 / Rev A.
Author: Andrew Sheppard, © 2015.

Quiz 1

Q1: In your own words, define what is "Big Data"?

Big Data is in the eye of the beholder, as the saying goes.

From an end user perspective, Big Data might merely mean that it doesn't fit into Excel. This is a way of saying that Big Data doesn't fit into my tool of choice (Excel) and that I have to learn some new, and more powerful, tool.

From a technologist perspective, Big Data means that Big Data doesn't fit into memory on a single machine. This again is like saying Big Data doesn't fit into my tool of choice (a single powerful computer) and new and more powerful technologies need to be used, such as Hadoop.

From a manager's perspective, Big Data means data that must be managed and analyzed by a specialized team and specialized hardware, and most likely means new and additional resources have to be found.

These are informal, and somewhat subjective, definitions of Big Data.

For a more formal definition, Big Data can be characterized along a number of "axes", so speak. The most common axes are Volume, Velocity, Value, Variety and Veracity. These are called the "5 V's", and are sometimes narrowed to "4 V's" (Volume, Velocity, Variety and Veracity) or "3 V's" (Volume, Velocity, and Variety) if you want to narrow your focus. Veracity is perhaps the axis along which most students have the most difficult time getting a grip on; it simply means correctness and accuracy—how do we know the data is telling the truth! Big Data in this more formal framework is any data set that is "Big" in one or more of the V's, with Volume being one of them.

Q2: This course teaches an approach—a methodology—for Big Data projects. There are 4 stages. Name them and give their meaning? I've filled in the first one for you!

D1: Data ___Exploration_____ → _____Know thy data_____

D2: Data __Programming____ → _____The data "plumbing" [1] _____

D3: Data _____Analysis_____ → ___Seek to find value in the data_____

D4: Data _____Insights_____ → _____Profit from your data_____

Note [1]: Many of the students wrote "Data Plumbing" for D2 which, while correct, doesn't sound as professional as "Data Programming". But have no doubt, in the D2 step you are building "plumbing" in the sense that you want your data management systems to be sound and not leak or break—just like plumbing!

Q3: Big Data can be characterized in different ways, commonly referred to as the "5 V's". Can you name them? And briefly describe what they represent; and their significance or implication? I've filled the first one in for you!

V1: _____Volume_____ → _____The quantity of data_____

→ _Data will not fit into machine RAM_

V2: _____Velocity_____ → Data rate; how fast the data is moving

→ _Keeping up with the data can be hard

V3: _____Value_____ → _Different data has different values_

→ _Half-life of data may be very short_

V4: _____Variety_____ → __Complexity and lack of structure__

→ _Data may be "ugly" (unstructured)__
and change often

V5: _____Veracity_____ → ___Is the data telling the truth?_____

→ _Apophenia; don't let the data fool you.
And can you trust the data? Verify?

Q4: Write down Amdahl's law and describe its significance. If you cannot write down the formula, just describe what Amdahl's law tells us about the potential speed up of a program.

$$S(n) = \frac{1}{a + (1 - a) / n}$$

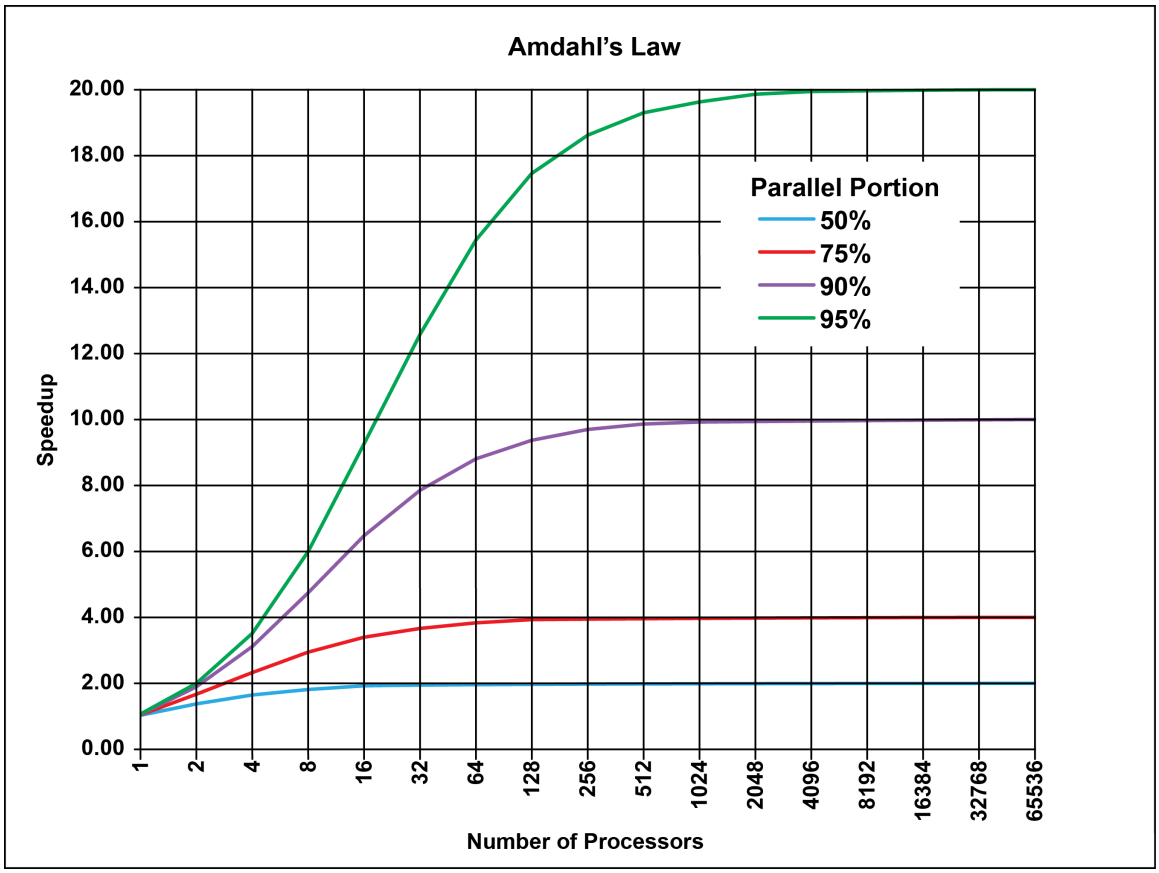
Where:

"S(n)" is the expected speed-up by having "n" processors.
"n" is the number of processors that can work in parallel.
"a" is the serial fraction of the program; that is, the fraction that cannot be made parallel.

At the extreme limits:

When the program is serial, "a = 1" and "S(n) = 1" and no amount of parallel hardware will speed up your program.
When the program can be made fully parallel, "a = 0" and "S(n) = n" and your program will scale linearly with the number of processors.

Between these two limits, the *theoretical* speed-up predicted by Amdahl's law varies with "a" and "n" as shown in the graph below. The speed up is *theoretical* because practical programming and hardware constraints may make your program fall short of the predicted speed-up. That is to say, Amdahl's law sets an *upper bound* on the speed-up you can expect from a given program; the speed-up in practice will always be lower.



Q5: What is the "half-life" of data? Can you write down its formula? What does it mean for Big Data projects? For example, what might you do if your program cannot keep up with the data if its half-life is very short?

The half-life of a data set is the time it takes for its economic value to half.

So, if the economic value of your data is $100 at some time t_0 (that is, you would be willing to spend $100 to buy it or spend $100 of resources to acquire it); but at some later time, t_1—for the exact same data set—you would only spend $50, then $(t_1 - t_0)$ is the half-life of that data set.

$$V(t) = V_0 \left(\frac{1}{2}\right)^{t/t_{1/2}}$$

Where:

"V(t)" is the value of the data set at time "t" in years from "t_0".
"V_0" is the value of the data set at the start, time "t_0".
"$t_{1/2}$" is the half-life of the data set under consideration, measured in years.

What this means for Big Data projects is that the effort and resources you put into obtaining and working with data sets should take into account the half-lives of those data sets; you should only put in effort and resources if you can capture most of the data's economic value before it decays away. It also means that for very short half-life data, if you program cannot keep up with the data it probably makes sense to just throw it way because it has become almost worthless.

Q6: The starting point of any parallel program is to write a serial equivalent version. Why? How can you oftentimes start development as a parallel program and yet still have a serial equivalent within the parallel program? When is this not true and you have to build a true serial equivalent first?

By building a serial equivalent version of the program first you have something to compare your parallel program with.

First, the serial version is more likely to be correct because it is simpler. That is the serial provides something to compare to for correctness.

Second, the serial version provides a basis of comparison. So when someone asks "how fast is your parallel program" you can reply with confidence and say "my parallel version of the program is N times faster that the serial version". Oftentimes the serial version of a data processing program can be faster for small data sets because there is less overhead in manipulating and moving the data and partitioning the problem so that parallelism can come into play. Only when the data grows to be big does the parallelism become more efficient because the overhead of manipulating and moving the data and partitioning the problem becomes small in comparison to the work done by the parallel processors. This means that there is often a cross-over point in data size below which you would like to run the serial version of the program against the data and above which you would like to run the parallel version of the program. Indeed, it often makes sense to bundle both your serial and parallel versions into the same program and run one or the other based on the size of the data presented to your program.

You can begin developing your program in parallel and satisfy having a serial equivalent version if you can always run and debug your program while running just one thread. The only time this is not possible is when the parallel program has—for technical reasons—to be written in a radically different way from the serial version, so that running with one thread looks nothing like what a single thread version of the program would look like if written from scratch.

Q7: Having your program output a log file is like having your program talk to you, or talk an end user. What is the purpose of a log file? What, at a minimum, should it be able output (what fields should it have on lines of output)?

The purpose of having a log file is to have a permanent record of what happened for a given run of your program. The resulting log file can be used for debugging your program, improving the performance of your program (for example, by identifying bottlenecks), regression testing (making sure you haven't broken something that was previously working), and for audit purposes (the log file may be need to persuade a boss or a regulator that the program is working).

For debugging, a log file is useful both during development *and* when deployed to production. Oftentimes, production machines do not have developer tools installed and indeed running any such tools is seen as undesirable; it's in production after all! When your program is deployed to production, oftentimes the log file is the *only* debugging tool you have available to you!

Ideally, a log file should capture this information when the program starts:

- The name of the program being run.
- Its run-time parameters.
- Its data inputs.
- Details of the hardware on which it is running (memory, processor, etc.).
- Date and "wall clock" start time.

And while the program is running, it should output this information:

- Each line in the log file should have a unique ID, such as a sequence counter that is incremented every time a line is written to the log file.
- If your program is a parallel program running on a cluster of many machines, then the machine ID should be used to tag each line of output so you know where it came from.
- A timestamp with a time resolution commensurate with the accuracy needed for the task being executed. So, for file I/O, which is typically slow, millisecond resolution may be OK, but for a function call and execution time it may have to be at the microsecond level.
- Where the log entry is being reported from, such as a module or function within your program.
- The type of information being reported, such as "INFO" (just information, such as the RAM in the machine on which your program is running), "DATA" (name and location of a data file being read from, or written to), and so on. Use short hand tags such as "INFO" and "DATA" and so on so that you can easily search or filter the log file to find the level of output you want.
- Freeform text output describing what's going on, or the problem that has been encountered. "Freeform text" is just a fancy way of saying any text that you think may be useful.

In addition, since the computational overhead and file I/O of the log file can be substantial it is important that logging can be switched off and on, and turned up (logging lots of program activity) and turned down (logging just the minimum needed).

Q8: The Monte-Carlo method can be broken down into 4 simple steps. Briefly outline each step? I've filled two of the steps in for you!

Step 1: _____Random number generation_____

Step 2: _____Transform "1" into function inputs_____

Step 3: _____Function evaluation_____

Step 4: _____Aggregate outputs to obtain result_____

Note: Random number generation in parallel on many separate machines is much harder than you think and fraught with all sorts of pitfalls and gotchas. It's an important and current topic in parallel and distributed programming, and one which we will deal with during the course. Because parallel random number generation (PRNG) is hard, that's another reason why having a serial equivalent of your program is important, because serial number generation is well understood and so your serial program is easier to verify as being correct. Then, with a correct serial program you can proceed to write your parallel program with more confidence because you have the serial program to compare it to.
