Baruch MFE "Big Data in Finance" course ~ Quiz 2 ~ Model answers

Document name: "BDiF2015 – TN0011.pdf"
Date/revision: Sat 14th March 2015 / Rev A.
Author: Andrew Sheppard, © 2015.

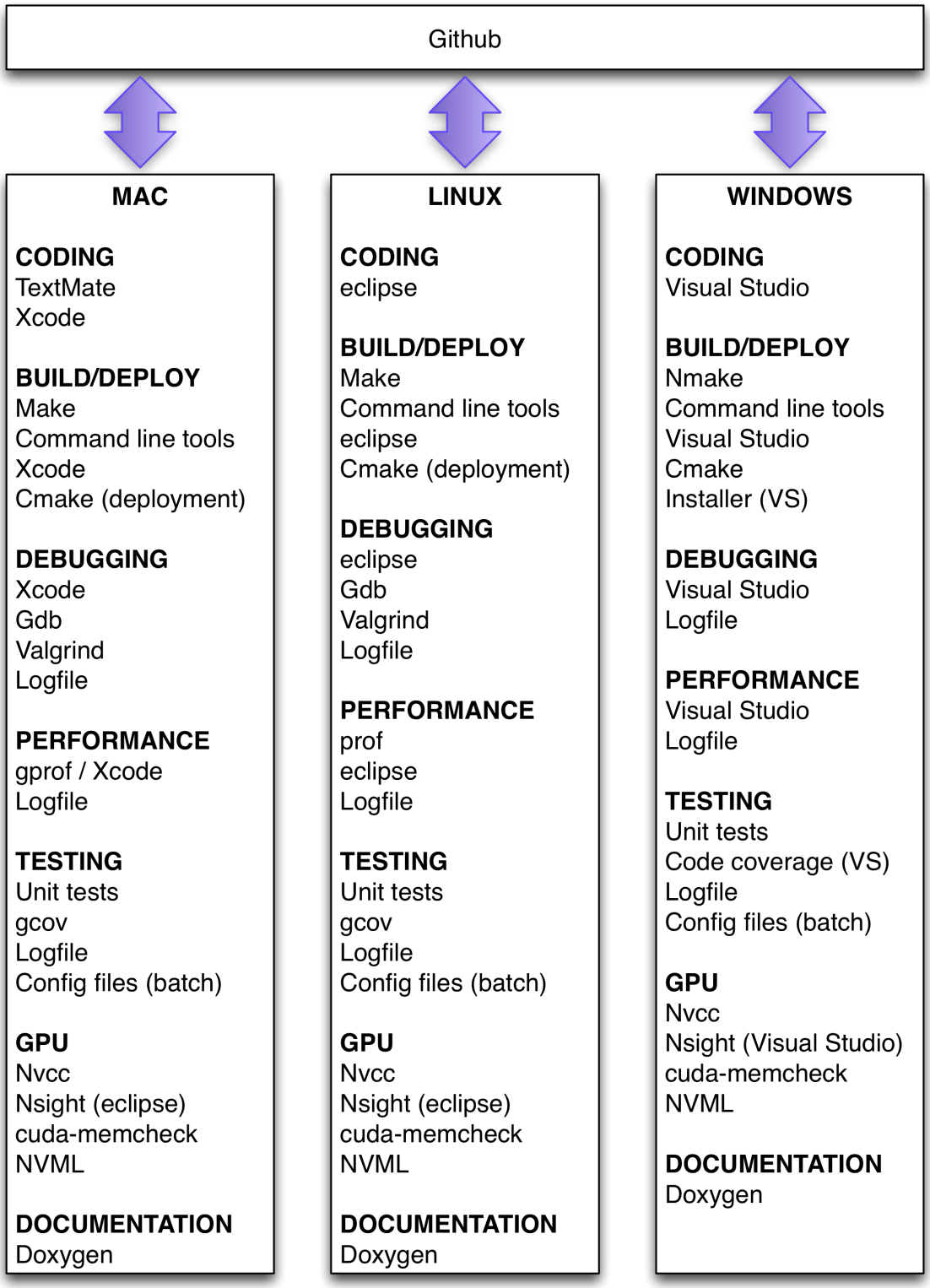
Quiz 2

Q1: In class I have spoken many times about the value of having a well defined "toolchain" (defined as all the tools use you to effectively and efficiently turn your ideas into code and from code into running programs; programs that run at scale); describe the advantages of having such a well thought out toolchain and sketch out your personal toolchain for working on course assignments (use the reverse of this page if you need extra space).

Having a well thought out toolchain has these advantages:

- It helps you write better programs. Better = faster, scalable, more robust and simpler.
- It allows you to develop those programs more quickly. Faster time to market.
- It makes you more productive. You can work on many things at once.
- It gives you control over development activities. Without a toolchain everything is ad-hoc and chaotic.
- It saves you time by automating tasks. If your toolchain saves you just 10 minutes per day, over the space of a work year of 250 days, that amounts to 42 hours. Equivalent to a whole work week! You could, for example, invest those 10min blocks into learning some new and valuable skill.
- It helps you coordinate development work with other people. You can work as part of a team. Most projects in the real world are team based.
- It makes you look more professional. People will want to hire you and work with you. The converse is also true; no toolchain and you like an amateur.
- A good toolchain helps you learn and grow. Like a carpenter, if you have good tools, you have a tendency to work on more ambitious projects.
- It helps you keep your sanity! Writing complex programs is hard. Don't make it more difficult by not having a well thought out toolchain.

My toolchain is somewhat unusual because I am a consultant. That means I have clients running Windows, Linux and Mac OSX. I have a toolchain on each of the platforms I develop for, as illustrated in the diagram below.



Q2: Outline how MapReduce works? Is MapReduce a general technique, or one that is unique to Hadoop? What is Hadoop, and how does it work in basic outline (what infrastructure does it provide to support MapReduce)?

MapReduce is a “divide and conquer” algorithim. It is a general technique, but is closely associated with Hadoop, which is a very (perhaps the most!) popular way of running Mapreduce programs. The reason why Hadoop is very popular is because it reduces (excuse the pun) a MapReduce job to just these parts:

- Input location of data.
- Output location of processed data.
- A map task.
- (An intermediate, and implicit, sort/combine step.)
- A reduce task.
- Job configuration.

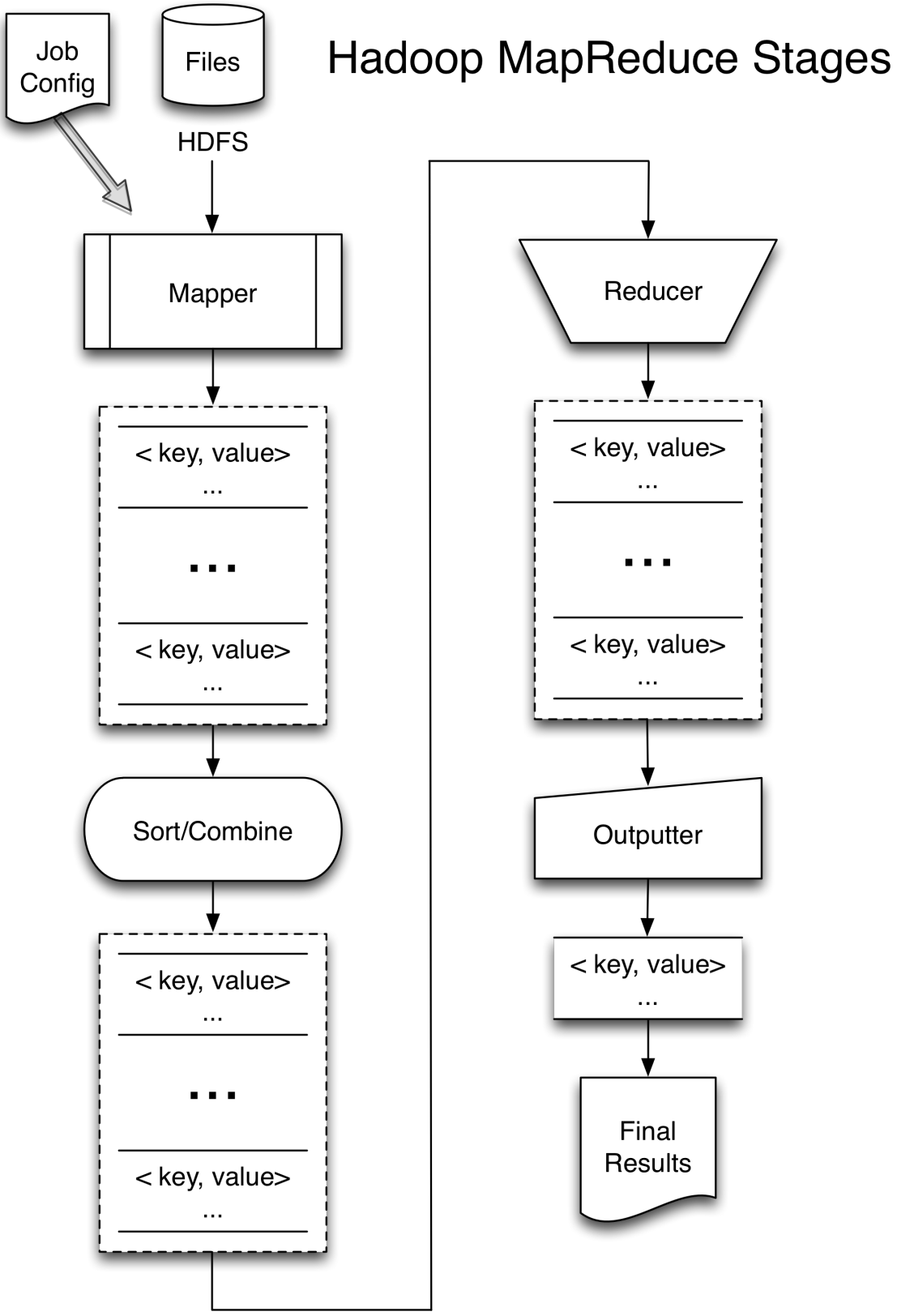
At intermediate steps in the MapReduce process, data is stored as <key, value> pairs.

A Hadoop job that has the above parts scales from 1 node to 10,000s nodes unchanged. It has scalability built in.

Hadoop is made up from several components. Underlying the whole Hadoop system is a parallel distributed file system called HDFS (Hadoop Distributed File System). HDFS is responsible for the efficient storage of very large (terabyte TB or petabyte PB) data sets. It keeps multiple copies of the data so that data is not lost even when hardware fails. Hadoop also efficiently distributes compute jobs that use the data so that compute is close to the data and, again, makes things are robust to failure so that parts of jobs that fail are automatically re-started. In some ways you can think of Hadoop as an “operating system” for Big Data; it provides all the infrastructure for MapReduce jobs to run at scale reliably.

Hadoop is also an “operating system” in the sense that are many other programs that come for free and run within the context of Hadoop, such as Mahout and Hive and many others. In that sense Hadoop is the center of a whole ecosystem of tools for Big Data.

The key stages of MapReduce as implemented by Hadoop are illustrated in the following diagram.



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?

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: What is NoSQL? What are the key differences between traditional relational databases and NoSQL databases? When would you use a relational database to store your data? And when would you use a NoSQL database to store your data? (Hint: Think in terms of the V's; and one V in particular.)

NoSQL stands for "Not only SQL".

Relational databases are usually row oriented and the relationships between tables of data are relatively static. That is, they have a fixed schema.

NoSQL databases on the other hand are usually column oriented and have a flexible relationship between the data. That is, their schema is more adaptable. With data becoming more and more unstructured (high variety), this explains the rise in popularity of NoSQL databases over traditional relational databases.

If the structure of the data is stable (grows slowly and predictably) and relationships well known and understood, and queries are of a well known and transactional nature, then NoSQL might not be the best choice to store and manage your data; it might be better stored in a relational database. The converse is also true.

Q5: What is a "make" and a "makefile"? Why is make useful? Briefly describe how it works? And write a basic makfile for a simple "Hello, world!" program main.cpp (very plain vanilla C++ with no special dependencies or libraries)? That is, assume main.cpp exists and your makefile must simply build it with, say, gcc.

"make" is a general utility for building programs and their associated resources, such as documentation, in an automated and predicable way.

A "makefile" has entries that follow this general format:

```
targets: dependencies
        actions
```

A target has dependencies. When a target is specified when running make, "make target", make looks at the dependencies and, if any of them are newer than the target, then only those dependents that are newer than the target and the target itself if built using the actions specified. In this way, only things that need to be built are built. If you have 1,000's of source files this can make an enormous difference in the build time of your program, making development work much easier and quicker. It can literally make the difference between your program building in seconds or minutes versus hours.

Note that targets can also be dependencies in later target → dependencies → actions entries in the makefile.

If "make" is run without any target being specified, then the first target entry in the makefile will be built.

To build main.cpp the makefile might look like this:

```
CC = gcc
CCFLAGS = -O3

main: main.o
     $(CC) main.o -o main_program

main.o: main.cpp
     $(CC) $(CCFLAGS) -c main.cpp
```

```
clean:
	rm -f main_program
	rm -f *.o
```

Q6: What is CVA? What does it measure? Why is it important? Briefly give an economic argument as to why doing CVA effectively and efficiently at the enterprise level is very beneficial (quantify in $ with an example if you can)? BONUS QUESTION: what is DVA?

CVA is new, it's been around for more than a decade. What changed things was the recent financial crisis when things changed. We went from seeing almost no credit risk, to seeing credit risk everywhere; now we have CVA (Counterparty Valuation Adjustment), DVA (Debt Valuation Adjustment), FVA (Funding Valuation Adjustment), and LVA (Liquidity Valuation Adjustment); indeed we are losing count of XVA (fill in the "X") adjustments. In short, banking and finance now sees credit risk under every rock and stone in sight—lift up any rock and you will find credit risk lying underneath so it seems! CVA has become important because the regulators expect it and management has come to rely on it as a component of the overall risk for their institution.

CVA measures the cost of counterparty credit risk. It's the varying amounts that have to be added to deals with that counterparty to offset the risk that they may default.

Suppose you are a bank with a regulatory capital requirement of $10billion. The bank regulators will give you a break on that capital requirement (i.e. allow you operate your business at the same level with less capital) if you can demonstrate that you have the systems and processes in place to manage your risks, specifically credit risk (see history above). The cost of capital, even for a bank, must be at least 10%. Further suppose that the regulators give you a $2billion break on capital; that is, you need only $8billion of capital. That means you save having to have $2billion more in capital. At a 10% cost of capital than means an annual saving of $200million a year. This year, next year, and every year thereafter; so you should value that $200million/year saving as a perpetuity. The value of a $200million/year perpetuity at a 10% discount rate is $200million/0.2 = $2billion. That means, if the band does CVA properly, to the regulator's satisfaction it can add $200million/year in profit to the bottom line! And that without having 1 extra counterparty, or 1 extra deal, or restructuring their business in any way beyond doing CVA properly! CVA can have a huge payoff.

DVA is "Debt Valuation Adjustment" and measures the code of credit should the bank default.

Q7: Describe the Monte-Carlo CVA methodology in outline? How do you calculate CVA (formulas)?

The Monte-Carlo method can be broken down into these 4 simple steps:

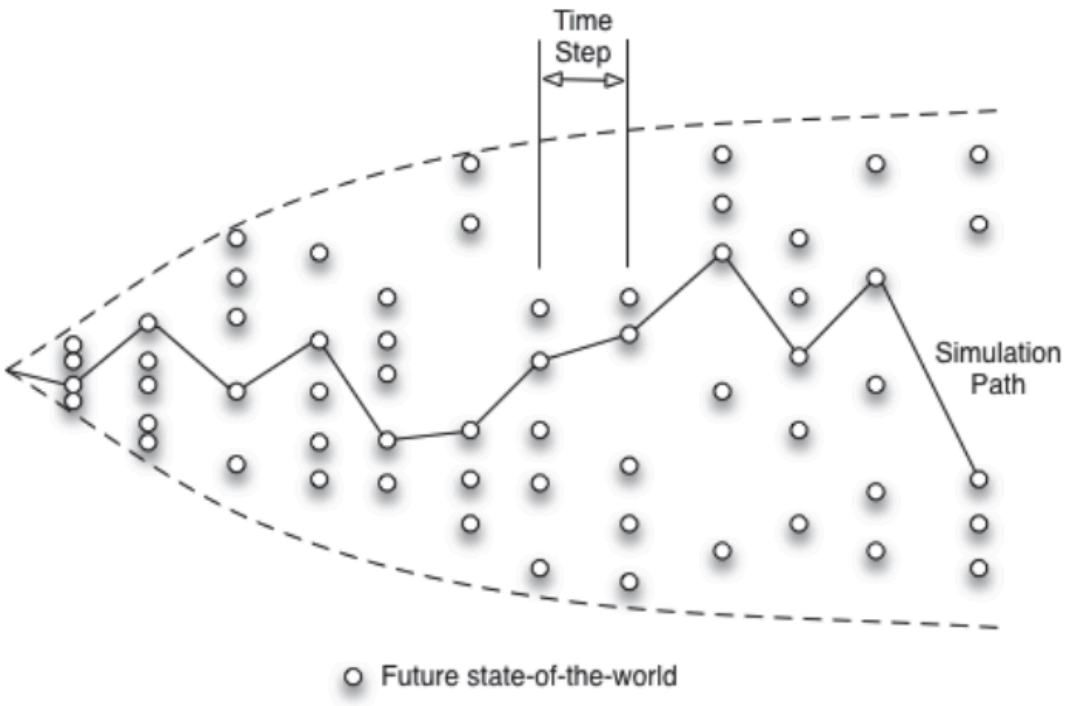
Step 1: _____Random number generation_____

Step 2: _____Transform "1" into function inputs_____

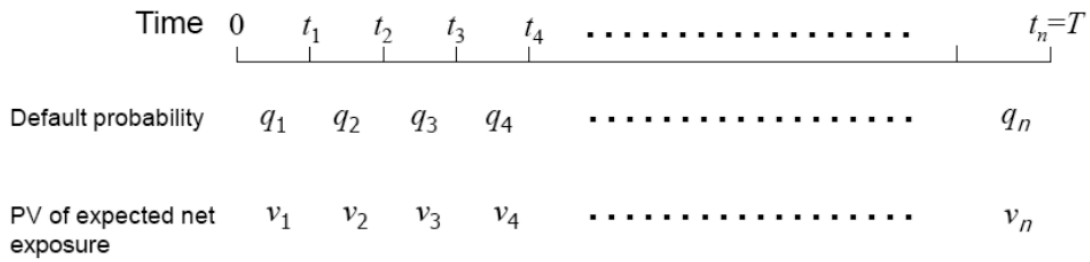
Step 3: _____Function evaluation_____

Step 4: _____Aggregate outputs to obtain result_____

In terms of enterprise level CVA, we model all the future states-of-the-world through Monte-Carlo simulated paths, and then apply some simple formulas, as illustrated in the diagrams below.



Now, on to the technical aspects of simplified CVA.



$$\text{CVA} = (1-R)\sum_{i=1}^{n} q_i v_i$$ where R is the recovery rate

The formula for DVA, incidentally, is very similar except that the default probability is that of the bank rather than the counterparty.

Q8: Why is CVA both a Big Data problem and a Big Compute problem? Briefly describe how CVA can be done in parallel and at scale? (Hint: What numerical technique is typically used to solve CVA? Why is that technique intrinsically parallel?)

As illustrated in Q7, CVA at its heart is conceptually simple, but implementing CVA is not!

The reason is because at an enterprise level, say a large bank, the data involved is very large. It can also be very difficult to get at as it make be stored in many systems spread across the globe. For a typical bank, the number of (capital markets) counterparties will likely be in the 10,000's, and the number of deals in the 10,000,000's; the number of legal agreements (netting sets) with counterparties will also likely be in the 10,000's. That's why enterprise level CVA is a Big Data problem.

As we saw in Q7, to arrive at a CVA number requires all the deals to be revalued in every state-of-the-world along every path and at every time step. That's what makes enterprise level CVA a Big Compute problem.

Deals as associated with counterparties, so `Cpty(deals)[i]` and be dealt with independently. (Deals are not truly independent because they normally live within netting sets that are associated with counterparties.) States-of-the-world are associated with simulation paths, so `Paths(states-of-the-world)[j]`

can be dealt with independently. Wherever you find independence in a problem there is a good chance you can also find parallelism.

Monte-Carlo is usually used to calculate CVA, and Monte-Carlo is intrinsically parallel because it does lots of calculations independently.

Taken together—independence of the parts, and Monte-Carlo simulation—and CVA has lots of opportunity for parallel execution.
