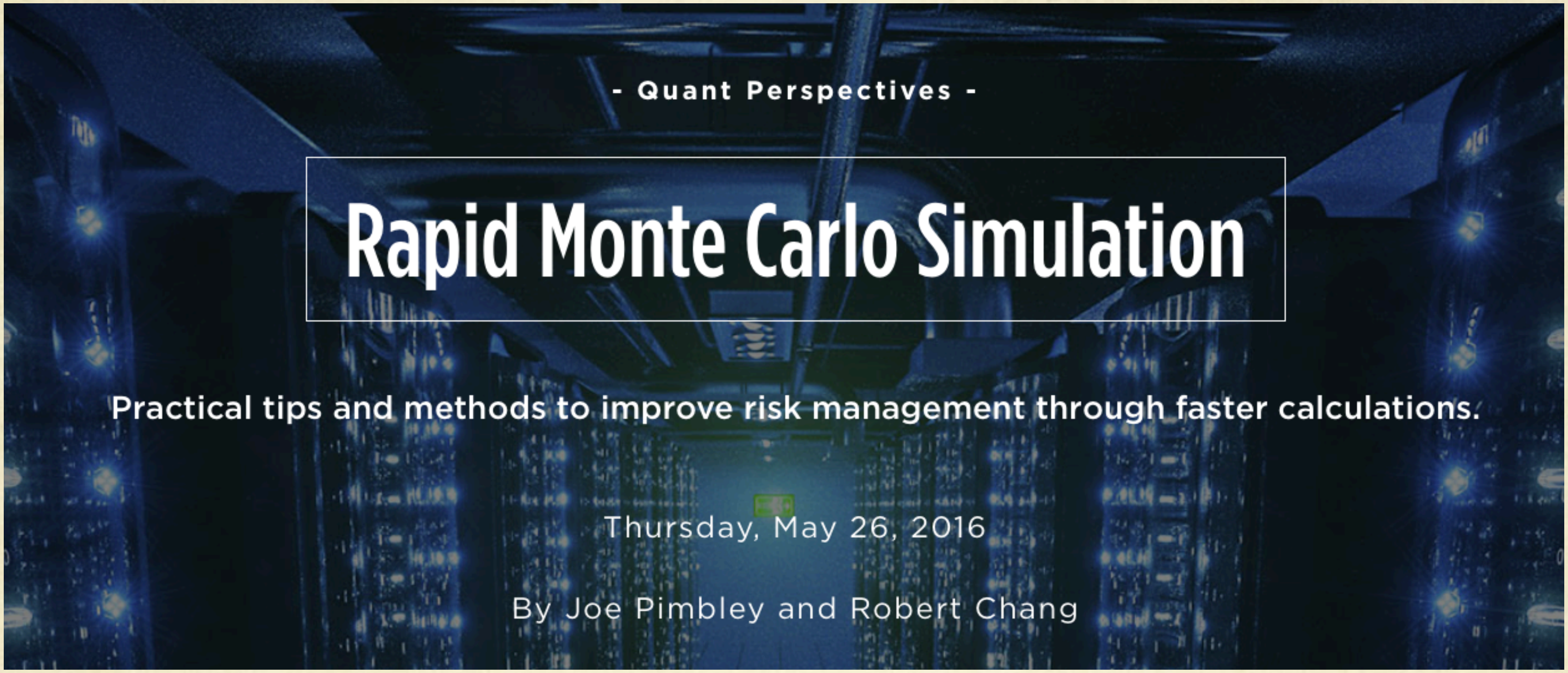
Rapid *Monte Carlo* Simulation - Hands-On Learning

March 2017

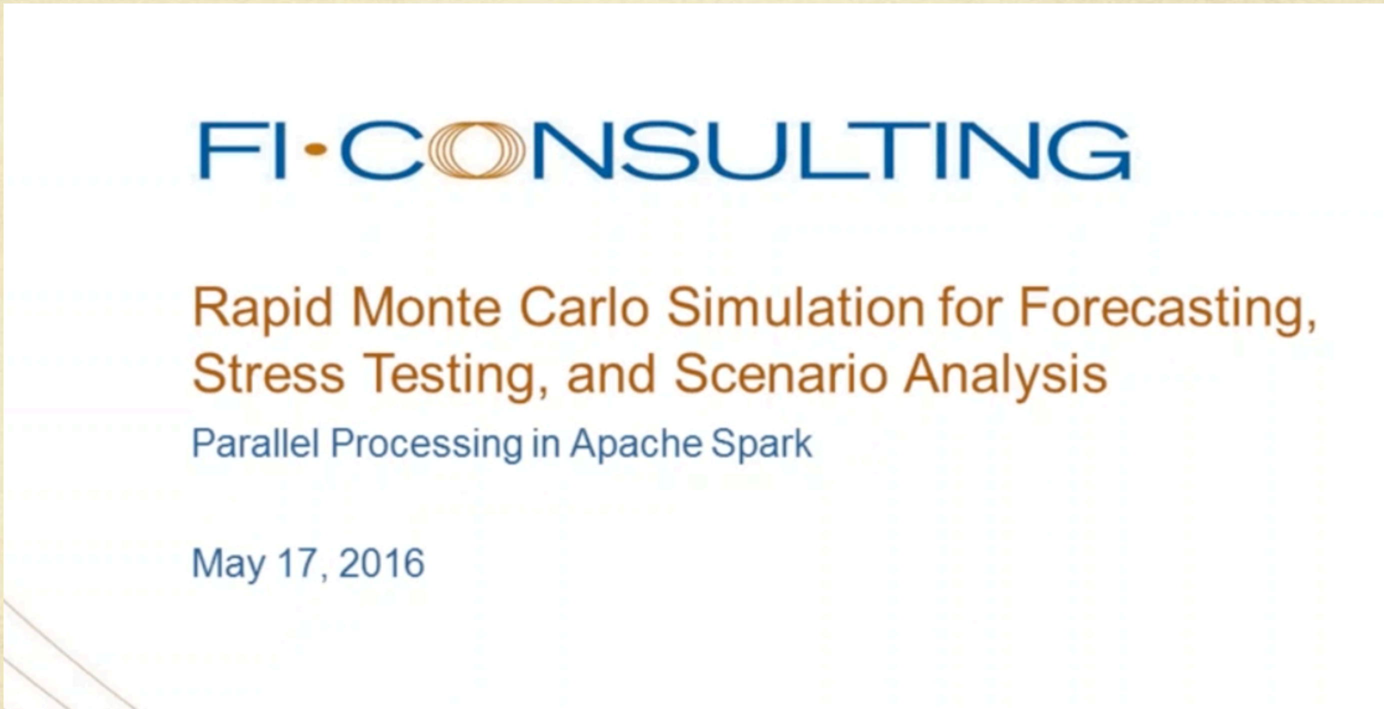
Rob Chang, FI Consulting
Joe Pimbley, Maxwell Consulting

- Monte Carlo Simulation Advice
- "Vasicek Test" Example for Code Acceleration
- Python, Parallel Processing, Spark
- Interactive Code Building – Excel VBA and Python

GARP Article



See the 2016 GARP article *Rapid Monte Carlo Simulation* at http://www.garp.org/#!/risk-intelligence/all/all/a1Z400000034RwEEAU



See this webinar posted to YouTube at https://youtu.be/A8E0Gue8ugY or at https://youtu.be/DcUD-Ezjw7c

In this Lab, we write the beginning code for Rapid Monte Carlo ....

Learning Lab Monte Carlo Files

Roll Dice: Excel VBA or Python

Log-Normal Equity: Excel VBA or Python

Video: Learning Lab for Excel VBA

Article: Rapid Monte Carlo Simulation

Video: Rapid Monte Carlo Simulation

Maxwell-Consulting.com/GARP.html

Faster is *Always* Better !

- Value of MC Simulation increases when calculations run faster
  - More precision in results
  - Run more "experiments"
  - Permits greater "realism" in adding features
  - Examples: Real-time risk management; nested MC methods

- Almost always possible to get faster speed
  - Hardware – eg, high-performance processors in parallel
  - Software – eg, optimized code
  - Disadvantage of "off-the-shelf" MC products

- Consider Portfolio of "Infinite" Number of Loans
- Each Loan has Identical Default Probability
- Each Loan has Identical Correlation to all Other Loans
- Vasicek: Analytical Solution for *Portfolio* Loss Distribution

$$F(x) = \Phi\left[\frac{-K + \sqrt{1-\rho}\,\Phi^{-1}(x)}{\sqrt{\rho}}\right] \quad \text{with} \quad K \equiv \Phi^{-1}(p)$$

$$F(x) = \Phi\left[\frac{-K + \sqrt{1-\rho}\,\Phi^{-1}(x)}{\sqrt{\rho}}\right] \quad \text{with} \quad K \equiv \Phi^{-1}(p)$$

- Real Project: Analysis of Auto Loan Pool Losses
- With Many Pools, MLE Estimation of PD & Correlation
- Need a Test for the MLE Estimators
- Build *Monte Carlo* Algorithm

Pool 1 | Pool 2 | Pool 3 | .... | Pool N

- Imagine M Loans in Each of the N Pools
- *Monte Carlo* Simulation to get Default Fraction each Pool
- Check MLE Estimators given “known” PD & Correlation
- Also a Test of Vasicek and Numerical Methods

Vasicek Analysis

The **Vasicek distribution** describes the probability density function for the fraction of defaulted loans within an infinitely diversified portfolio. Simple but restrictive assumptions specify a single default probability (PD) common to each loan and a single correlation parameter linking the behavior of all loans.

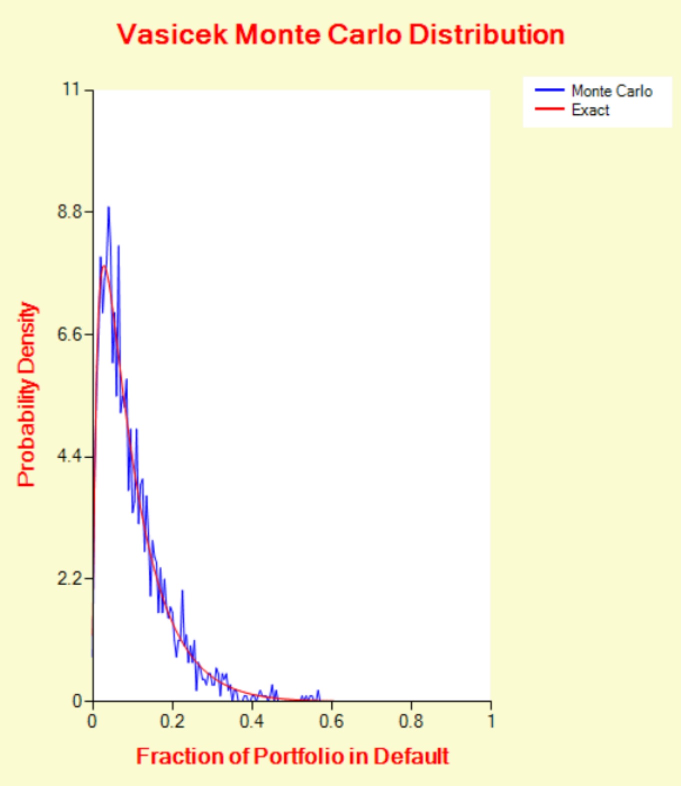
Variance Test: The numerical integration and Monte Carlo simulation are two viable methods to compute the variance of this Vasicek distribution - very important for understanding the risk of the loan portfolio!

Pool Test: Create an arbitrary number of pools with an arbitrary number of obligors per pool. Apply Monte Carlo simulation to determine fraction of defaults in each pool. Then compare to the analytical Vasicek distribution.

Vasicek Pool Test

Enter # Pools 2,000 Enter # Obligors per Pool 10,000 Calculate

Return to Main Page



Visual Studio .Net Web Application
Webjoe.azurewebsites.net/Vasicek_Variance

```
'   Generate Num_Pools to measure the default fraction of Number_Obligors
'   within each pool.  We provide a single-factor correlation and then
'   determine if the MLE extraction for correlation and Obligor default
'   probability works well.
For kount_Pools = 1 To Num_pools
    Def_number = 0
    '   Set the systemic random variable Pool_Y.
    Call Gauss_RV(G1, G2)
    Pool_Y = G1 * Sqr(rho)

    For kount = 1 To Number_obligors Step 2
        Call Gauss_RV(G1, G2)
        Ob_RV = Pool_Y + Sqr(1# - rho) * G1
        If Application.NormSDist(Ob_RV) < Def_prob Then Def_number = Def_number + 1
        Ob_RV = Pool_Y + Sqr(1# - rho) * G2
        If Application.NormSDist(Ob_RV) < Def_prob Then Def_number = Def_number + 1
    Next kount

    Def_fraction(kount_Pools) = CDbl(Def_number) / CDbl(Number_obligors)
    If Def_number = 0 Then
        x(kount_Pools) = Min_X
    Else
        x(kount_Pools) = Application.NormSInv(Def_fraction(kount_Pools))
    End If

Next kount_Pools
```

With 2,000 Pools and 10,000 Obligors per Pool, the Inner Loop Generates RVs for Default Determination of 20 Million Loans

```
'   Generate Num_Pools to measure the default fraction of Number_Obligors
'   within each pool.  We provide a single-factor correlation and then
'   determine if the MLE extraction for correlation and Obligor default
'   probability works well.
For kount_Pools = 1 To Num_pools
    Def_number = 0
    '   Set the systemic random variable Pool_Y.
    Call Gauss_RV(G1, G2)
    Pool_Y = G1 * Sqr(rho)

    For kount = 1 To Number_obligors Step 2
        Call Gauss_RV(G1, G2)
        Ob_RV = Pool_Y + Sqr(1# - rho) * G1
        If Application.NormSDist(Ob_RV) < Def_prob Then Def_number = Def_number + 1
        Ob_RV = Pool_Y + Sqr(1# - rho) * G2
        If Application.NormSDist(Ob_RV) < Def_prob Then Def_number = Def_number + 1
    Next kount

    Def_fraction(kount_Pools) = CDbl(Def_number) / CDbl(Number_obligors)
    If Def_number = 0 Then
        x(kount_Pools) = Min_X
    Else
        x(kount_Pools) = Application.NormSInv(Def_fraction(kount_Pools))
    End If

Next kount_Pools
```

Look Carefully at Each Line of Code in the Inner Loop to Reduce "Expensive" Calculations

```
'   Generate Num_Pools to measure the default fraction of Number_Obligors
'   within each pool.  We provide a single-factor correlation and then
'   determine if the MLE extraction for correlation and Obligor default
'   probability works well.
For kount_Pools = 1 To Num_pools
    Def_number = 0
    '   Set the systemic random variable Pool_Y.
    Call Gauss_RV(G1, G2)
    Pool_Y = G1 * Sqr(rho)

    For kount = 1 To Number_obligors Step 2
        Call Gauss_RV(G1, G2)
        Ob_RV = Pool_Y + Sqr(1# - rho) * G1
        If Application.NormSDist(Ob_RV) < Def_prob Then Def_number = Def_number + 1
        Ob_RV = Pool_Y + Sqr(1# - rho) * G2
        If Application.NormSDist(Ob_RV) < Def_prob Then Def_number = Def_number + 1
    Next kount

    Def_fraction(kount_Pools) = CDbl(Def_number) / CDbl(Number_obligors)
    If Def_number = 0 Then
        x(kount_Pools) = Min_X
    Else
        x(kount_Pools) = Application.NormSInv(Def_fraction(kount_Pools))
    End If

Next kount_Pools
```

Look Carefully at Each Line of Code in the Inner Loop to Reduce "Expensive" Calculations

```
'   Generate Num_Pools to measure the default fraction of Number_Obligors
'   within each pool.  We provide a single-factor correlation and then
'   determine if the MLE extraction for correlation and Obligor default
'   probability works well.
For kount_Pools = 1 To Num_pools
    Def_number = 0
    '   Set the systemic random variable Pool_Y.
    Call Gauss_RV(G1, G2)
    Pool_Y = G1 * Sqr(rho)

    For kount = 1 To Number_obligors Step 2
        Call Gauss_RV(G1, G2)
        Ob_RV = Pool_Y + Sqr(1# - rho) * G1
        If Application.NormSDist(Ob_RV) < Def_prob Then Def_number = Def_number + 1
        Ob_RV = Pool_Y + Sqr(1# - rho) * G2
        If Application.NormSDist(Ob_RV) < Def_prob Then Def_number = Def_number + 1
    Next kount

    Def_fraction(kount_Pools) = CDbl(Def_number) / CDbl(Number_obligors)
    If Def_number = 0 Then
        x(kount_Pools) = Min_X
    Else
        x(kount_Pools) = Application.NormSInv(Def_fraction(kount_Pools))
    End If

Next kount_Pools
```

Look Carefully at Each Line of Code in the Inner Loop to Reduce "Expensive" Calculations

Code Improvement

```
Sq_rho = Sqr(rho)
Sq_wmrho = Sqr(1# - rho)

'   Generate Num_Pools to measure the default fraction of Number_Obligors
'   within each pool.  We provide a single-factor correlation and then
'   determine if the MLE extraction for correlation and Obligor default
'   probability works well.
For kount_Pools = 1 To Num_pools
    Def_number = 0
    '   Set the systemic random variable Pool_Y.
    Call Gauss_RV(G1, G2)
    Pool_Y = G1 * Sq_rho

    For kount = 1 To Number_obligors Step 2
        Call Gauss_RV(G1, G2)
        Ob_RV = Pool_Y + Sq_wmrho * G1
        If Application.NormSDist(Ob_RV) < Def_prob Then Def_number = Def_number + 1
        Ob_RV = Pool_Y + Sq_wmrho * G2
        If Application.NormSDist(Ob_RV) < Def_prob Then Def_number = Def_number + 1
    Next kount

    Def_fraction(kount_Pools) = CDbl(Def_number) / CDbl(Number_obligors)
    If Def_number = 0 Then
        x(kount_Pools) = Min_X
    Else
        x(kount_Pools) = Application.NormSInv(Def_fraction(kount_Pools))
    End If

Next kount_Pools
```

Making this Change Reduces Execution Time from 120 s to 117 s – Small Saving.

```
Sq_rho = Sqr(rho)
Sq_wmrho = Sqr(1# - rho)

'   Generate Num_Pools to measure the default fraction of Number_Obligors
'   within each pool.  We provide a single-factor correlation and then
'   determine if the MLE extraction for correlation and Obligor default
'   probability works well.
For kount_Pools = 1 To Num_pools
    Def_number = 0
    '   Set the systemic random variable Pool_Y.
    Call Gauss_RV(G1, G2)
    Pool_Y = G1 * Sq_rho

    For kount = 1 To Number_obligors Step 2
        Call Gauss_RV(G1, G2)
        Ob_RV = Pool_Y + Sq_wmrho * G1
        If JNormsDist(Ob_RV) < Def_prob Then Def_number = Def_number + 1
        Ob_RV = Pool_Y + Sq_wmrho * G2
        If JNormsDist(Ob_RV) < Def_prob Then Def_number = Def_number + 1
    Next kount

    Def_fraction(kount_Pools) = CDbl(Def_number) / CDbl(Number_obligors)
    If Def_number = 0 Then
        x(kount_Pools) = Min_X
    Else
        x(kount_Pools) = Application.NormSInv(Def_fraction(kount_Pools))
    End If

Next kount_Pools
```

Change: Reduces Execution Time from 117 to 28 – Big Saving!

Code Improvement

```
Sq_rho = Sqr(rho)
Sq_wmrho = Sqr(1# - rho)
Phi_inv_Def_prob = Application.NormSInv(Def_prob)

'   Generate Num_Pools to measure the default fraction of Number_Obligors
'   within each pool.  We provide a single-factor correlation and then
'   determine if the MLE extraction for correlation and Obligor default
'   probability works well.
For kount_Pools = 1 To Num_pools
    Def_number = 0
    '   Set the systemic random variable Pool_Y.
    Call Gauss_RV(G1, G2)
    Pool_Y = G1 * Sq_rho

    For kount = 1 To Number_obligors Step 2
        Call Gauss_RV(G1, G2)
        Ob_RV = Pool_Y + Sq_wmrho * G1
        If Ob_RV < Phi_inv_Def_prob Then Def_number = Def_number + 1
        Ob_RV = Pool_Y + Sq_wmrho * G2
        If Ob_RV < Phi_inv_Def_prob Then Def_number = Def_number + 1
    Next kount

    Def_fraction(kount_Pools) = CDbl(Def_number) / CDbl(Number_obligors)
    If Def_number = 0 Then
        x(kount_Pools) = Min_X
    Else
        x(kount_Pools) = Application.NormSInv(Def_fraction(kount_Pools))
    End If

Next kount_Pools
```

$\Phi(RV) < PD$

equivalent to

$RV < \Phi^{-1}(PD)$

Change: Reduces Execution Time from 28 to 8.4 – Big Saving! End Result is 15x Speed Improvement

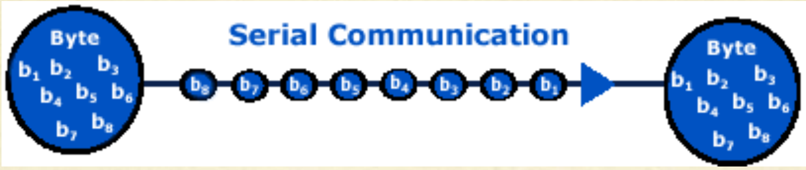
Many or most of our programs are Serial.

- **A Serial Program consists of a sequence of instructions, where each instruction executes one after the other.**
- **Serial programs run from start to finish on a single processor.**

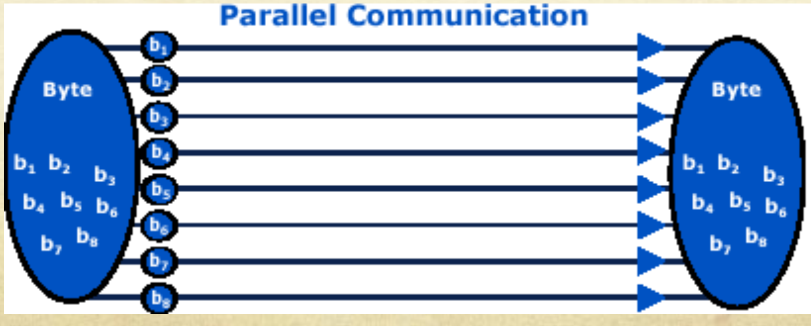
Parallel programming developed as a means of improving performance and efficiency.

- **Parallel Programs are usually ran on a set of computers connected on a network, or a pool of CPUs.**
- **Parallel Programs can be used to solve problems involving large datasets and non-local resources.**

A Serial Program consists of a sequence of instructions, where each instruction executes one after the other.



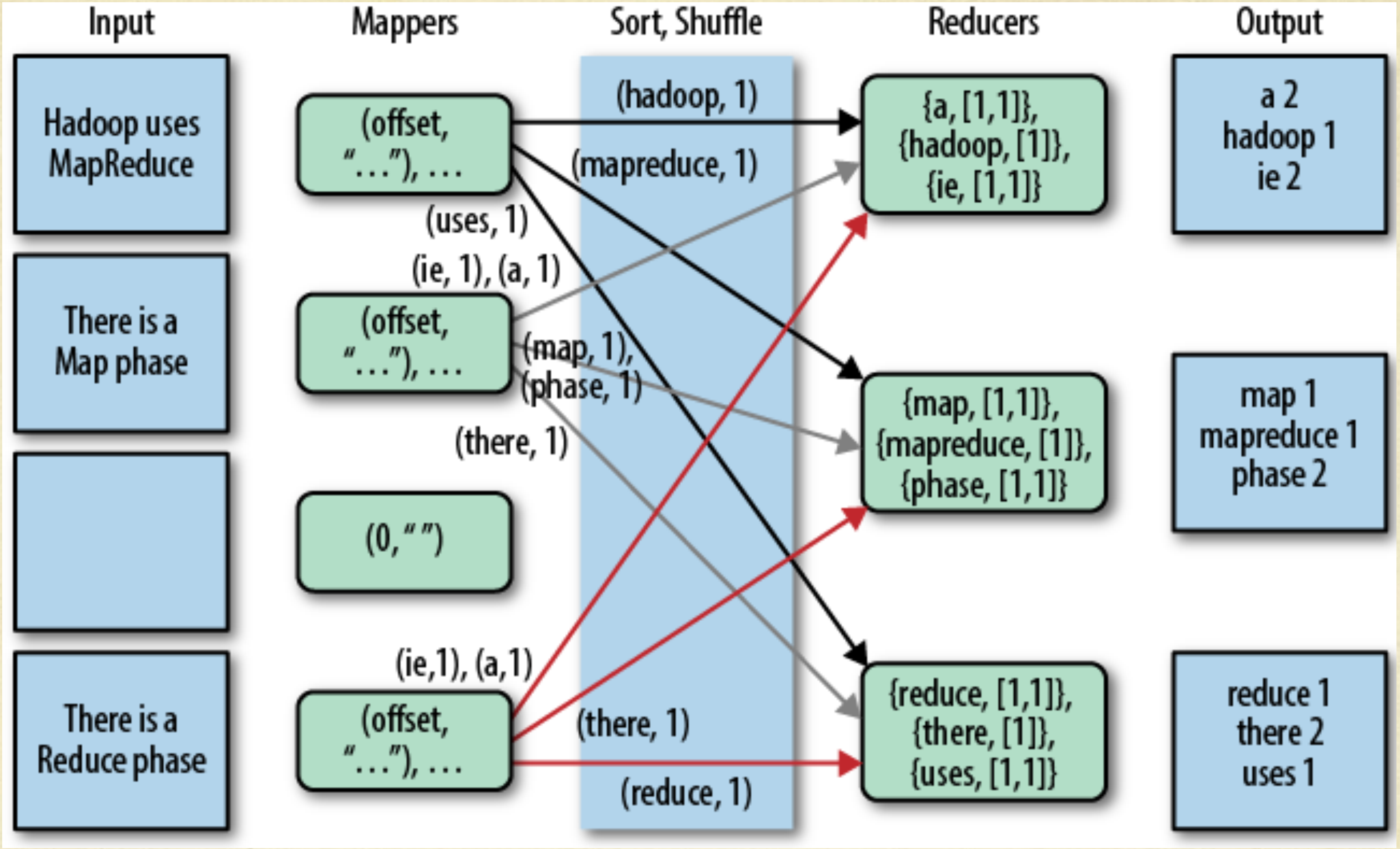
In a Parallel Program, the processing is broken up into parts, each of which could be executed concurrently on a different processor



- **MapReduce was developed within Google as a mechanism for processing large amounts of raw data, for example, crawled documents or web request logs.**

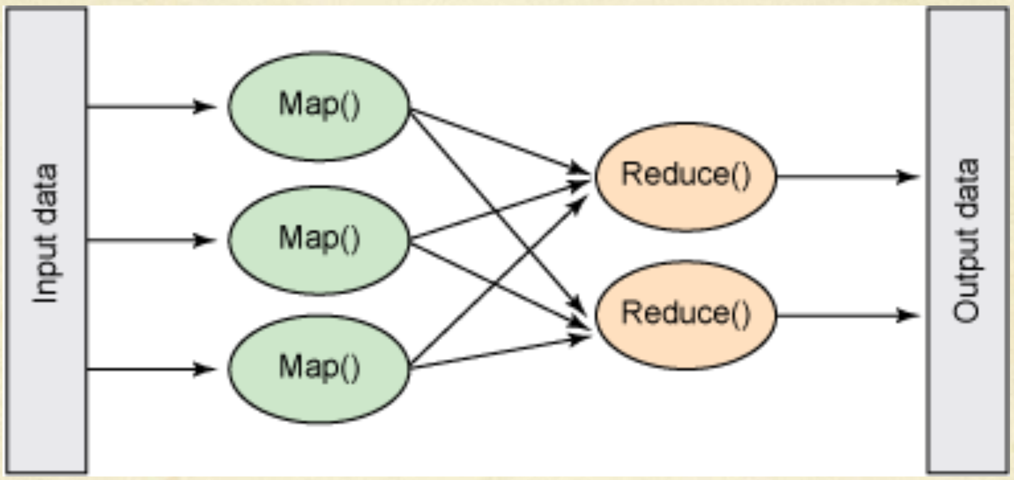
- **Google data is so large, it must be distributed across tens of thousands of machines in order to be processed in a reasonable time.**

- **The distribution implies parallel computing since the same computations are performed on each CPU, but with a different portion of data.**



MapReduce divides a task into subtasks, handles the sub-tasks in parallel, and aggregate the results of the subtasks for the final output.

A <u>map job</u>, takes a set of data and converts it into another set of data, where individual elements are broken down into key/value pairs.



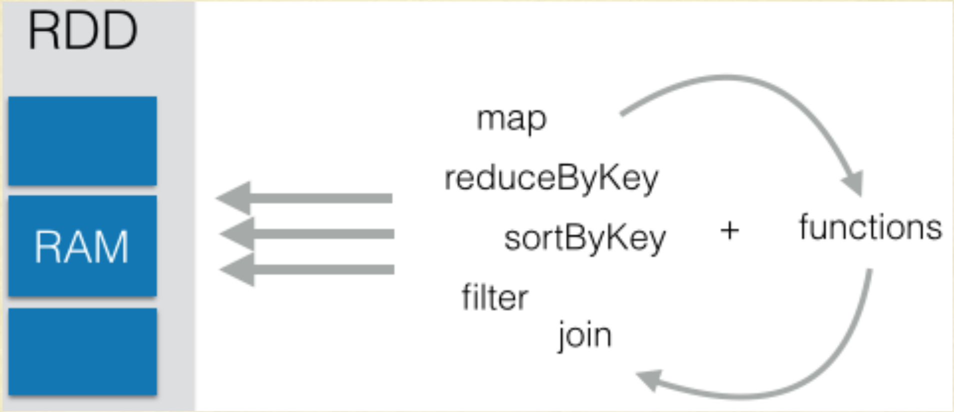
A <u>reduce job</u> takes the output from a map as input and merges together these values to form a possibly smaller set of values.

- **Spark extends MapReduce model to efficiently support more types of computations, including interactive queries and stream processing.**

- **Spark is an open-source software solution that performs rapid calculations on in-memory distributed datasets.**

- **Spark is designed to be highly accessible, offering simple APIs in Python, Java, Scala, and SQL, and rich built-in libraries.**

Spark is an open-source software solution that performs rapid calculations on in-memory distributed datasets.



Spark's uses Resilient Distributed Datasets (RDDs). RDDs can be automatically recomputed on failure and are resilient and fault-tolerant.

```
def default(idx):
    def_number = 0
    temp_gauss = Gauss_RV()
    ob_rv = pool_y + sq_wmrho * temp_gauss[0]
    if ob_rv < phi_inv_def_prob: def_number += 1
    ob_rv = pool_y + sq_wmrho * temp_gauss[1]
    if ob_rv < phi_inv_def_prob: def_number += 1
    return def_number

for kount_pools in range(0,num_pools):
    temp_gauss = Gauss_RV()
    pool_y = temp_gauss[0] * sq_rho

    count = sc.parallelize(xrange(number_obligors_half)).map(default)
    count = count.reduce(add)

    def_fraction[kount_pools] = count / float(number_obligors)
```

We can use the "parallelize" function to run each Monte Carlo trial in parallel rather than sequentially.

```
def default(idx):
    def_number = 0
    temp_gauss = Gauss_RV()
    ob_rv = pool_y + sq_wmrho * temp_gauss[0]
    if ob_rv < phi_inv_def_prob: def_number += 1
    ob_rv = pool_y + sq_wmrho * temp_gauss[1]
    if ob_rv < phi_inv_def_prob: def_number += 1
    return def_number

for kount_pools in range(0,num_pools):
    temp_gauss = Gauss_RV()
    pool_y = temp_gauss[0] * sq_rho

    count = sc.parallelize(xrange(number_obligors_half)).map(default)
    count = count.reduce(add)

    def_fraction[kount_pools] = count / float(number_obligors)
```

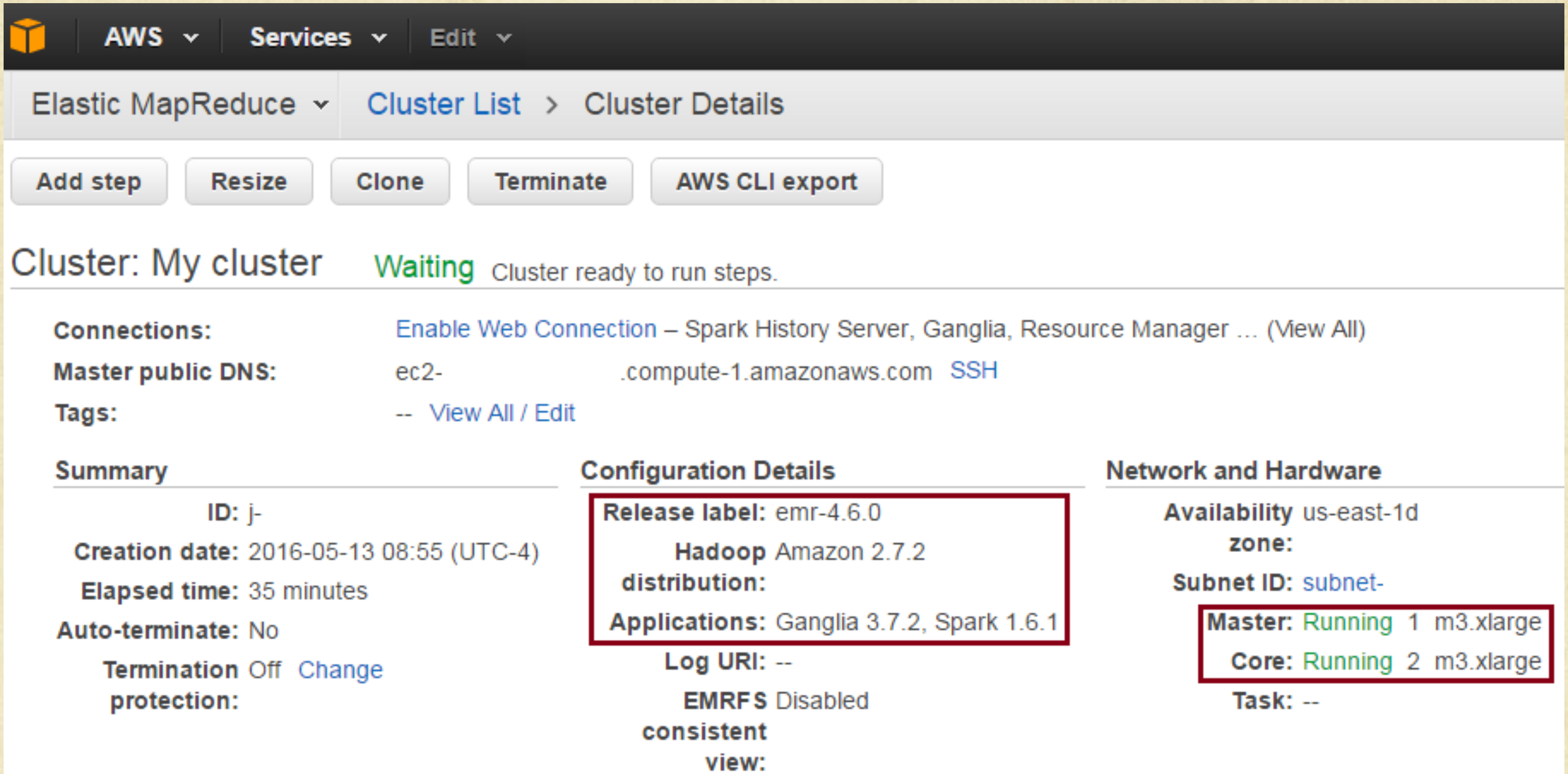
We "map" the default function to each parallel node, returning a 1 or 0 value corresponding to default status.

Parallel Processing of Monte Carlo Samples

```
def default(idx):
    def_number = 0
    temp_gauss = Gauss_RV()
    ob_rv = pool_y + sq_wmrho * temp_gauss[0]
    if ob_rv < phi_inv_def_prob: def_number += 1
    ob_rv = pool_y + sq_wmrho * temp_gauss[1]
    if ob_rv < phi_inv_def_prob: def_number += 1
    return def_number

for kount_pools in range(0,num_pools):
    temp_gauss = Gauss_RV()
    pool_y = temp_gauss[0] * sq_rho

    count = sc.parallelize(xrange(number_obligors_half)).map(default)
    count = count.reduce(add)

    def_fraction[kount_pools] = count / float(number_obligors)
```

We "reduce" by simply summing the default statuses, and using the resultant sum to calculate the default percentage in each loan pool.

Running Monte Carlo in the Cloud



We will run today's example on Amazon Web Services, however there are numerous cloud providers to choose from, i.e. MS Azure, Digital Ocean etc.

We will now go over some Monte Carlo examples together, in both Excel VBA and Python. Feel free to grab a partner for this activity!

To learn more, please contact us!

Robert Chang, Model Validation Lead
chang@ficonsulting.com
FI Consulting at www.ficonsulting.com

Joe Pimbley, Principal
pimbley@maxwell-consulting.com
Maxwell Consulting at www.maxwell-consulting.com