Assignment 3

The goal of this assignment is to build a parallel cluster to analyze bond portfolios. What is different in this assignment is you develop the dispatcher and worker to coordinate "node" objects which, given a randomized collection of portfolio ids, prices the corresponding portfolios using parallel collections.

Background
The objective is to analyze all 100,000 portfolios in the ParaBond repository using the DW pattern and a cluster of three hosts as the figure below suggests.

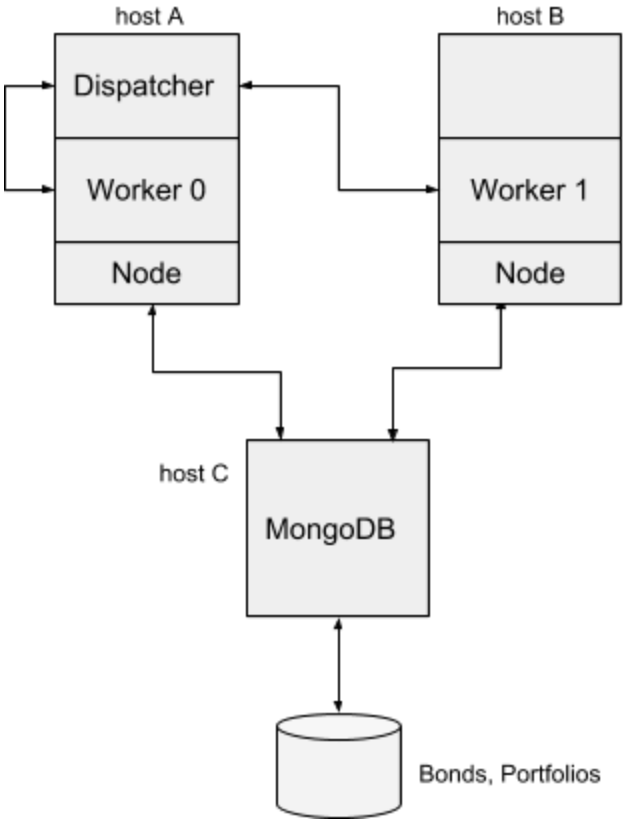

Figure 1. Cluster architecture

The `parabond.cluster` package of ParaScale contains `Node` classes. The worker simply needs to pass to the node class the partition information and a pointer to the Mongo host. Node prices the portfolios in the partition using parallel collections, updates the price in the database, and returns to the dispatcher the results.

The dispatcher needs to get only the T_1 data back from the workers; the dispatcher does not need the price data because the dispatcher can get prices from Mongo, if needed.

There are four concrete subclasses of Node as the table below suggests.

Table 1. Node subclasses.

Node subtype	Semantics
`BasicNode`	This node loads a portfolio into memory, prices it, updates the database, and repeats this for all portfolios within a parallel collection.
`MemoryBoundNode`	This node caches all the portfolios into memory first before pricing them and updating the database within a parallel collection
`FineGrainNode`	This node loads a portfolio into memory and prices each bond separately as a parallel collection within a parallel collection
`CoarseGrainNode`	This node batches n / M portfolios within a parallel collection.

A `Node` has one method, `analyze`, which takes one parameter, `Partition`, and returns a `Analysis` which contains pricing and performance data. The `Partition` defines…

- random seed
- number of portfolios to price
- starting index in the range of portfolios to price

The `analyze` method of `Node` uses the partition information to retrieve the portfolios from the database using a pattern, called `Casa`, and prices them according to bond pricing theory, namely,

$$B = \sum_{t=1}^{n \times T} \frac{C}{(1 + r_t)^{t/n}} + \frac{M}{(1 + r_T)^T} \tag{1}$$

where r_t is the time-dependent yield curve, in this case, on-the-run US treasuries. Equation 1 is implemented in a class, `SimpleBondValuator`, in the `parabond.value` package. The Node uses Equation 2 to price the portfolio, that is,

$$P = \sum_{i=1}^{k} B_i \tag{2}$$

See the sequence diagram for the dynamic relationship between worker and nodes.

Tasks

Part I - Configure and test the cluster.

1. Get three hosts, preferably next to one another in a lab. Let's call these hosts A, B, and C.

2. Download, install, and start Mongo(i.e., `mongod`) on host C.

3. On host A, clone (or fetch) the latest ParaScale project.

4. On host A, run `DbLoader` (it's in the `parabond.db` package) as follows on host A.
    a. Create a run configuration for DbLoader.

    b. Set the VM option: -Dhost=*C* where C is the host address of the Mongo server.

    c. Run DbLoader which populates Mongo.

5. On host C, use the Mongo shell to reset the prices of the check portfolios to -1.0. For instance, here is the command for the first check portfolio:

    ```
    db.Portfolios.update({id: 41361}, $set{price: -1.0});
    ```

    (I've created a Mongo script to automate this, `check-reset.js_`. However, it will be best to build this into the dispatcher--see below.)

6. On host C, launch the Mongo shell (i.e., `mongo`) and index both Bonds and Portfolios collections as:

    ```
    db.Bonds.createIndex({id: 1});
    db.Portfolios.createIndex({id: 1});
    ```

Part II - Write the code using the Scala style guide here.

1. In IntelliJ, open 679Project which you created in the last assignment and create a new package, `assign3`, and add the ParaScale.jar artifact as a dependency.

2. In the assign3 package create two classes, `ParaDispatcher` and `ParaWorker`, as subclasses of `Dispatcher` and `Worker`, respectively.

    `ParaDispatcher` and `ParaWorker` interact in much the same way as actors in the prior assignment, namely, through partition and result classes.

    The `Partition` class is part of the `cluster` package and does not need to be re-written.

    The `Result` class is part of the `util` package and does not need to be re-written.

3. Create the `Result` case class which extends `Serializable`. Minimally, this class needs to contain partial data to compute T_1.

4. Write ParaDispatcher using the following algorithm:
    a. Output the report header.

    b. Get the next *n*, that is, number of portfolios to price.

    c. Reset the check portfolio prices.[1]

    d. Create two workers by passing the dispatcher constructor two sockets.

    e. Create two partitions:
    A) Partition(seed=0, n=n/2, begin=0) and
    B) Partition(seed=0, n=n/2, begin=n/2)

    f. Send worker(0) the first partition and worker(1) the second partition.

    g. Wait for results.

    h. Test the check portfolios.[2]

    i. Output the performance statistics.

    j. Repeat step **b**.

5. Write the ParaWorker code to run on a single host using the following algorithm:

    a. If worker running on a single host, spawn two workers else spawn one worker.

    b. Wait for a task.

    c. Create a Partition.

    d. Create a Node with the Partition.

    e. Invoke analyze on the Node and wait for it to finish.

---

[1] Invoke `checkReset(n)` where n is the number of portfolios being priced. This method currently checks 5% of the portfolios to be priced.
[2] Invoke `check(portfIds)` where `portfIds` are the check portfolio ids returned by `checkReset`. The `check` method returns a list of "missed" portfolios that were not priced. This everything worked, the list should have zero length.

f. Reduce the partial T_1 results.

g. Reply to dispatcher with T_1.

h. Repeat step **b**.

Part III - Test on single host cluster for n=100.
Use the BasicNode.

Part IV - Test on multi-host cluster.
Configure the environment using Figure 1 using the BasicNode.

Part V - Analyze all 100,000 portfolios.

1. Use the BasicNode.

2. Disable all diagnostics using log4j.properties.[3]

3. Use the ramp below.

Table 2. Analysis ramp

Rung	n
1	1,000
2	2,000
3	4,000
4	8,000
5	16,000
6	32,000
7	64,000
8	100,000

Part VI - Choose another node type.

1. Count the letters in your first name and take mod three. If the result is…

mod 3	Use
0	MemoryBoundNode
1	CoarseGrainNode
2	FineGrainNode

2. Repeat Part V for the indicated Node.

Deliverables

[3] If you get the console diagnostic, "ERROR StatusLogger No log4j2 configuration file found..." add log4j2.xml to the project root directory of the project. See this link for more details.

1. Upload the IntelliJ project as a *.zip* compressed file.

2. Upload the report (see below) as a *.txt* file.

3. Upload all diagnostic files.

Evaluation
I will evaluate the project on the basis of conformance to these specs. Late assignments will be penalized 10%. Requests for resubmission for any reason will be penalized 10%. Assignments that fail to load, compile, run successfully, etc., cannot be accepted.

Report generation
The dispatcher outputs the following sample report for the first two rungs of the analysis ramp.

```
ParaBond Analysis
By Ron Coleman
12 Apr 2018
BasicNode
Workers: 2
Hosts: localhost (worker), 148.10.191.1 (worker), 148.10.191.2 (mongo)
Cores: 12
     n  missed      T1       TN     R     e
  1000       0   20.50     2.50  8.20  0.68
  2000       0   43.75     4.50  9.72  0.81
```

Note #1
All times in the above report are in seconds. I've written a Scala class, `NanoToSecondsCoverter`, in the that implicitly converts nanoseconds to seconds. Here's an example:

```
import parabond.cluster._
val nano = System.nanoTime()
println(nano seconds)
```

Note #2
You can automate the choice of Node without having to rewrite the code if you give the code the class name (say through VM options) and construct the Node reflectively. Here an example:

```
import parascale.util.getPropertyOrElse
val prop = getPropertyOrElse("node","parabond.cluster.BasicNode")
val clazz = Class.forName(prop)

import parabond.cluster.Node
val node = clazz.newInstance.asInstanceOf[Node]
```