Final Presentation

Team Blue

Test Result

- Completed in ~40s
- 4 worker nodes with 2 * 32MB input files each
- More test cases at the github repo

```
blue@vm42:~/cs434-project$ java -jar master.jar 4
20:25:45.713 [main] INFO  Master -- Server started at 2.2.2.142:30962
20:25:57.598 [grpc-default-executor-0] INFO  Master -- Received register request from 2.2.2.146
20:25:59.300 [grpc-default-executor-0] INFO  Master -- Received register request from 2.2.2.143
20:25:59.895 [grpc-default-executor-0] INFO  Master -- Received register request from 2.2.2.144
20:26:00.657 [grpc-default-executor-0] INFO  Master -- Received register request from 2.2.2.145
20:26:00.672 [scala-execution-context-global-14] INFO  Master -- Received all register requests, worker ips: List(2.2.2.143, 2.2.2.144, 2.2.2.145, 2.2.2.146)
20:26:00.728 [scala-execution-context-global-15] INFO  Master -- Received all register requests, ranges: List(<ByteString@51855b06 size=10 contents="  l;w?EBj0">, <ByteS
yteString@5e59fcd2 size=10 contents="f{UbSY+,Cy">)
20:26:00.752 [scala-execution-context-global-14] INFO  Master -- opened channels to all workers
20:26:01.352 [scala-execution-context-global-14] INFO  Master -- Sent distribute start request to all workers
20:26:08.622 [grpc-default-executor-2] INFO  Master -- Received distribute complete request from 2.2.2.146
20:26:10.596 [grpc-default-executor-2] INFO  Master -- Received distribute complete request from 2.2.2.145
20:26:12.321 [grpc-default-executor-2] INFO  Master -- Received distribute complete request from 2.2.2.144
20:26:12.574 [grpc-default-executor-2] INFO  Master -- Received distribute complete request from 2.2.2.143
20:26:12.576 [scala-execution-context-global-20] INFO  Master -- Received all distribute complete requests, worker ips: List(2.2.2.143, 2.2.2.144, 2.2.2.145, 2.2.2.146)
20:26:12.629 [scala-execution-context-global-19] INFO  Master -- Sent sort start request to all workers
20:26:17.659 [grpc-default-executor-2] INFO  Master -- Received sort complete request from 2.2.2.146
20:26:20.563 [grpc-default-executor-2] INFO  Master -- Received sort complete request from 2.2.2.145
20:26:22.652 [grpc-default-executor-2] INFO  Master -- Received sort complete request from 2.2.2.143
20:26:24.247 [grpc-default-executor-2] INFO  Master -- Received sort complete request from 2.2.2.144
20:26:24.248 [main] INFO  Master -- All the workers finished sorting, MasterComplete
```

Test Result

- By shell script(written by github copilot)
- Check each worker’s integrity -> checked whole workers

```
Records: 617098
Checksum: 4b417db3c855c
Duplicate keys: 0
SUCCESS - all records are in order
```

Worker 1

```
Records: 640160
Checksum: 4e32aedd8a12a
Duplicate keys: 0
SUCCESS - all records are in order
```

Worker 2

```
Records: 631097
Checksum: 4d14856f1b433
Duplicate keys: 0
SUCCESS - all records are in order
```

Worker 3

```
Records: 671645
Checksum: 520ae34eb55bb
Duplicate keys: 0
SUCCESS - all records are in order
```

Worker 4

Test Result

- By shell script(written by github copilot)
- Check each worker’s integrity -> checked whole workers

```
blue@vm43:~/val$ ../gensort/64/valsort -s final.sum
Records: 2560000
Checksum: 13893954f23074
Duplicate keys: 0
SUCCESS - all records are in order
```

Milestones

- Milestone 1
  - Generate input data - done
  - Connect Master & Worker nodes - done
  - Configure gRPC - done
- Milestone 2
  - Worker samples data: done
  - Master computes ranges for each worker nodes given worker's range -> distribute it to workers with (range, ipaddress) list: done
  - Workers merge data : Moved from 3 to 2, done
  - Can execute(send) protocols in other, concurrently in Worker & Master using future & promise with dummy data, done
- Milestone 3
  - Workers sort and partition its data (should decide the size of partitioned file - done) - done
  - Workers distribute its files to appropriate node range. - done
  - Workers terminate & notify master - done
  - Can execute(send) protocols in other, concurrently in Worker & Master using future & promise with real data (merge top-down & bottom-up work) - done

What we did

- Jeongho:
- Overall design
- Master/Worker general control/communication flow
- Local sorting
- Attach sample/merge logics

What we did

- Jihun:
- Overall design
- Sample/Merge logics
- Integration/Smoke test

Design

No polling(no while() {sleep})

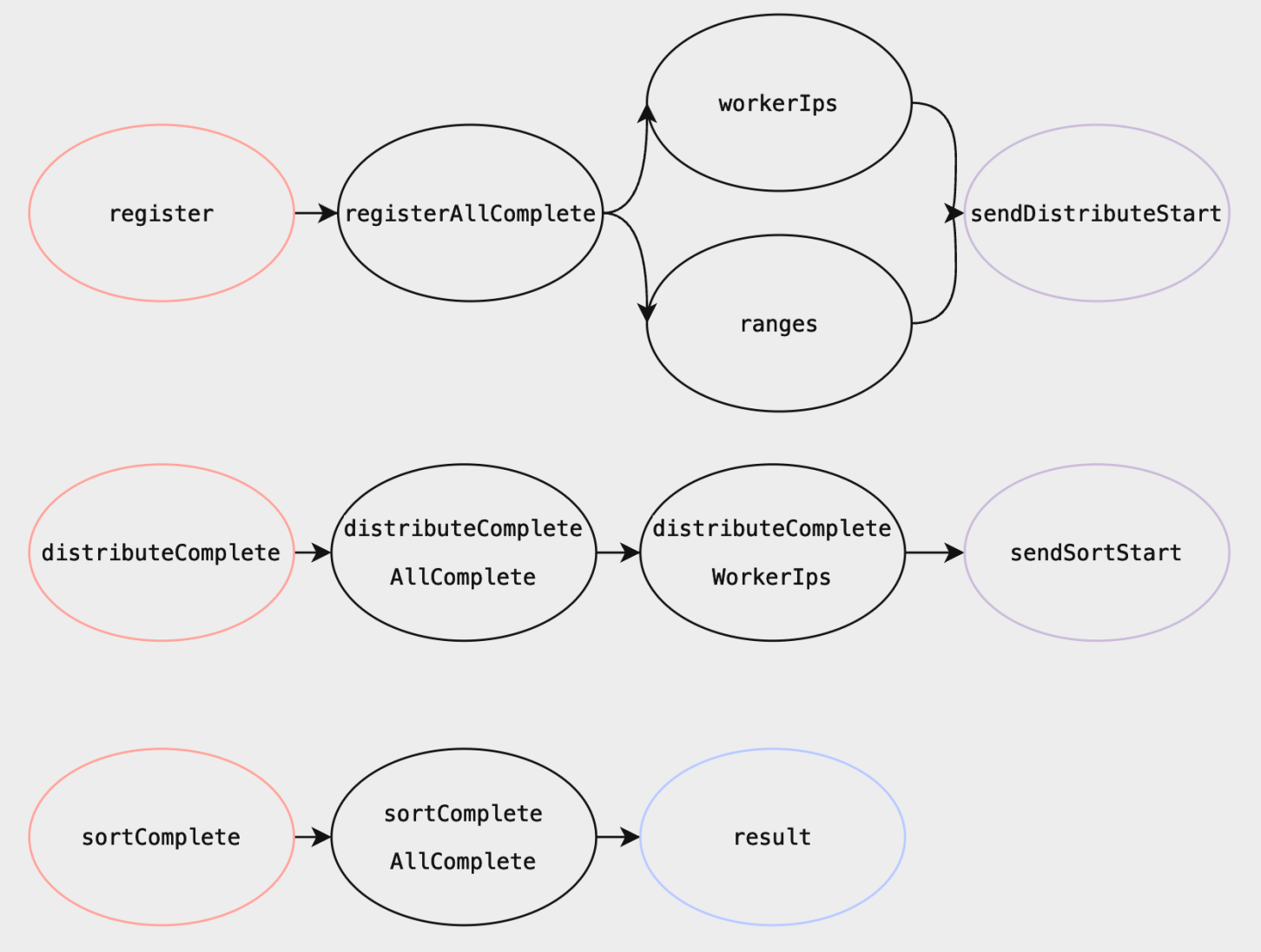
Only Future, Promise

Terminology

Distribute = shuffle

Sort = merge

Master



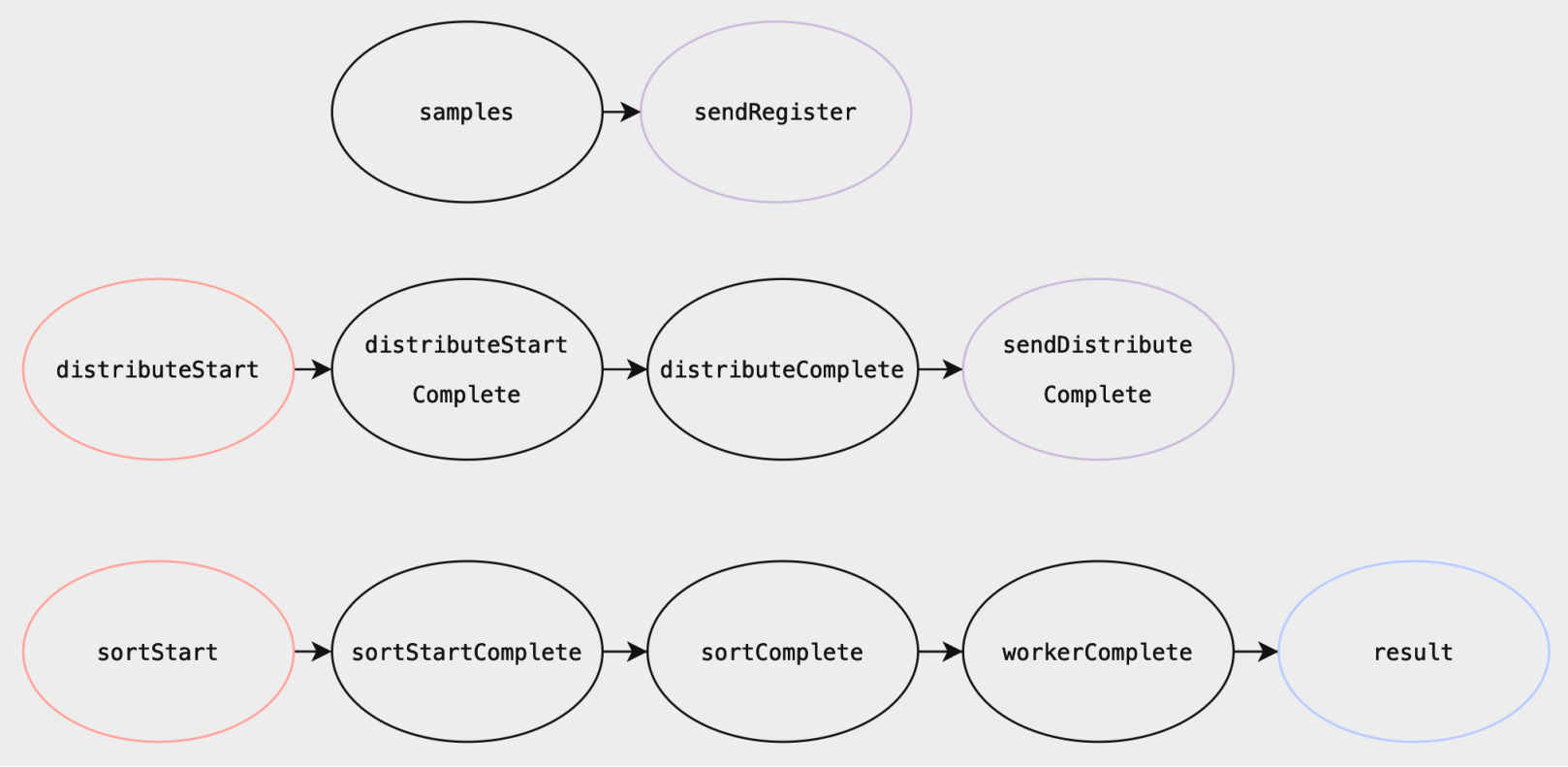
Pink: gRPC Service

Black: Promise(or Future)

Purple: function returning Future[Unit]

Blue: Result of Await.Result

Worker

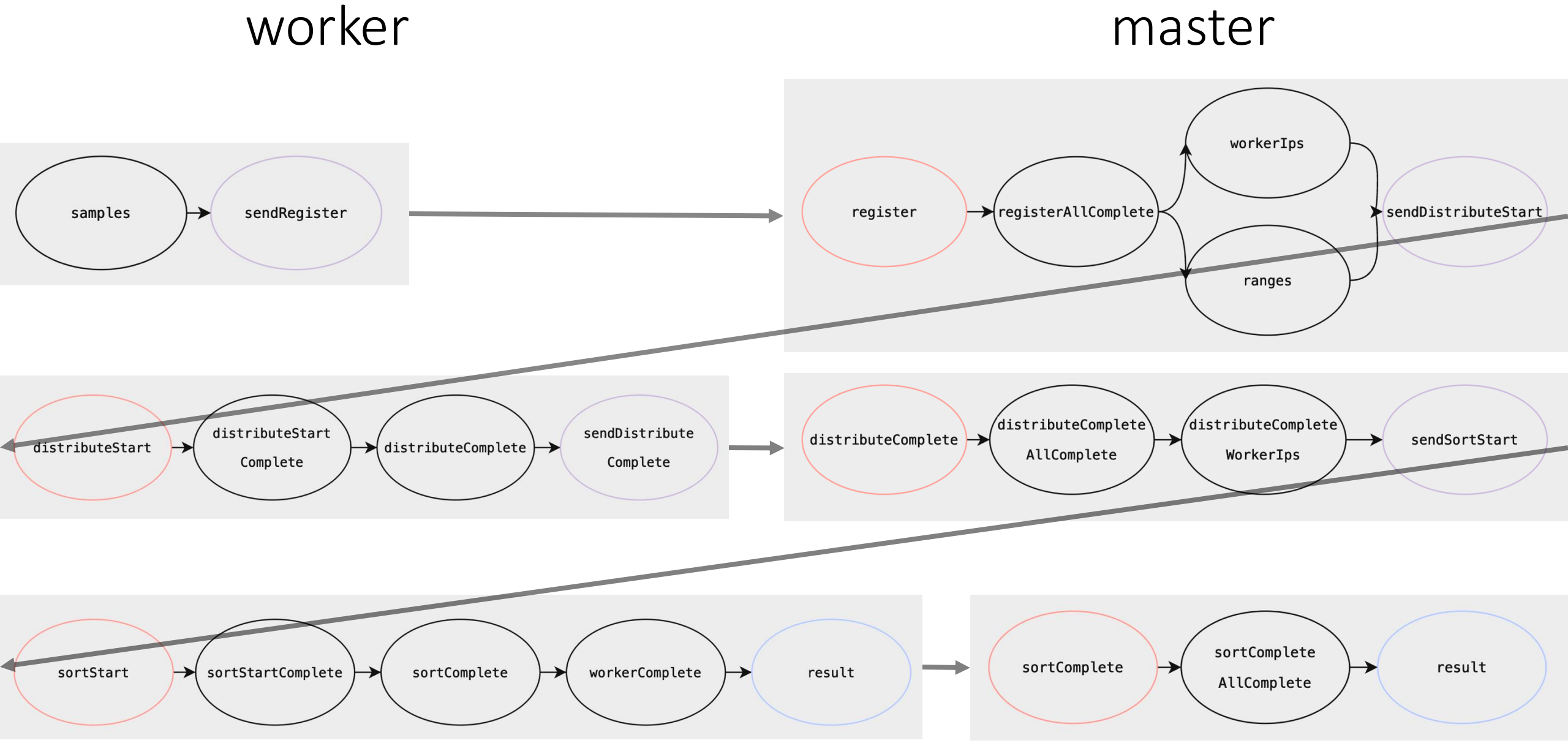


Pink: gRPC Service

Black: Promise(or Future)

Purple: function returning Future[Unit]

Blue: Result of Await.Result


worker
master
samples
sendRegister
register
registerAllComplete
workerIps
ranges
sendDistributeStart
distributeStart
distributeStart Complete
distributeComplete
sendDistribute Complete
distributeComplete
distributeComplete AllComplete
distributeComplete WorkerIps
sendSortStart
sortStart
sortStartComplete
sortComplete
workerComplete
result
sortComplete
sortComplete AllComplete
result

```
private val distributeComplete: Future[Unit] = sendDistribute
```

```
private def sendDistribute: Future[Unit] = async {
  val ranges: SortedMap[String, ByteString] = await(distributeStartComplete.future)
  val workerIps: List[String] = ranges.keys.toList
  val rangeBegins: List[ByteString] = ranges.values.toList
  val (toClose: List[BufferedSource], recordsToDistribute: List[Iterator[Record]]) =
    await(recordFileManipulator.getRecordsToDistribute).unzip
  try {
    logger.info(s"Sending DistributeRequest for all samples to designated workers(Distribution started)")
    val channels: List[ManagedChannel] = workerIps map {...}
    val blockingStubs: List[WorkerGrpc.WorkerBlockingStub] = channels map WorkerGrpc.blockingStub

    @tailrec
    def distributeOneBlock(records: List[Record]): Unit = {...}

    def getDesignatedWorker(record: Record): Int = {...}

    blocking {
      recordsToDistribute foreach { iter: Iterator[Record] => distributeOneBlock(iter.toList) }
    }
    channels foreach (_.shutdown)
  } finally {
    toClose foreach recordFileManipulator.closeRecordsToDistribute
  }
  /.../
  logger.info(s"Sent DistributeRequest for all samples to designated workers(Wait for response)")
  ()
}
```

Distribute(Shuffling)

Shuffling is done in sendDistribute.
sendDistribute sends distribute requests.

When worker receives a distribute request(containing records less than 2MB), it simply saves the records as a file.

```
override def distribute(request: DistributeRequest): Future[DistributeResponse] = {
  val records = request.records
  logger.info(s"Received DistributeRequest from ${request.ip} with ${records.size} records")
  recordFileManipulator saveDistributedRecords records
  Future(DistributeResponse())
}
```

Merging

Need to merge “m” sorted record blocks into 1 sorted record block.

Let there are total ”n” records.

We need to find the smallest record among m blocks. (and repeat this n times)

Basic way is to simply compare m times O(n m)

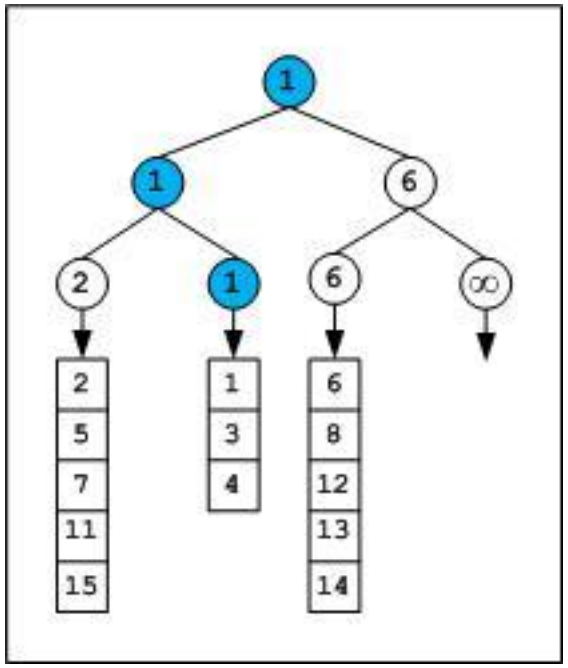
We use tournament tree O(n log m)

If there are total 2GB of records, m is 2 GB / 2 MB = 1000.

m increases linearly as the total size increases

Tournament tree

Basically a modified merge sort



Tournament tree

Implemented in RecordFileManipulator

```
private def mergeIterators(iter1: Iterator[Record], iter2: Iterator[Record]): Iterator[Record] = {
  val head1: Option[Record] = if (iter1.hasNext) Some(iter1.next()) else None
  val head2: Option[Record] = if (iter2.hasNext) Some(iter2.next()) else None
  (head1, head2) match {
    case (None, None) => Iterator()
    case (None, Some(record)) => Iterator(record) ++ iter2
    case (Some(record), None) => Iterator(record) ++ iter1
    case (Some(record1), Some(record2)) =>
      if (record1.key < record2.key)
        Iterator(record1) ++ mergeIterators(iter1, Iterator(record2) ++ iter2)
      else
        Iterator(record2) ++ mergeIterators(Iterator(record1) ++ iter1, iter2)
  }
}
```

```
private def mergeSortIterators(iterators: List[Iterator[Record]]): Iterator[Record] = {
  val (iter1: List[Iterator[Record]], iter2: List[Iterator[Record]]) =
    splitIterators(iterators, List(), List())
  (iter1, iter2) match {
    case (Nil, Nil) => Iterator()
    case (_, Nil) =>
      Check.weakAssert(logger)(iter1.length == 1, s"iter1.length is not equal to 1")
      iter1.head
    case (_, _) =>
      val (iter1_sorted: Iterator[Record], iter2_sorted: Iterator[Record]) =
        (mergeSortIterators(iter1), mergeSortIterators(iter2))
      mergeIterators(iter1_sorted, iter2_sorted)
  }
}
```

Design Remarks

- Control states as future & promise to schedule the overall flow
- Sort&Sample before register to the master
- Separate DistributeStart/DistributeFinish & SortStart/SortFinish requests so that the distribution and sorting(merging) does not happen in the httpclient’s request-response context
- Decided the size of temp&output files as 2MB, because of OOM error during file save

If we were to redo the project from beginning

- Jihun:
- Decide the data representation of the saved file & protocol & method interfaces first more than anything else.
- Change the signature and communication protocol more test friendly
- Set detailed milestone & due date

- Jeongho:
- Think more about time complexity(e.g., list concat vs flatmap?, list or cats chain?)
- Automate testing, formatting