Map-Reduce源码分析

Split部分

Job类

waitForCompletion()方法

```
public boolean waitForCompletion(boolean verbose
                                   ) throws IOException, InterruptedException,
                                            ClassNotFoundException {
    if (state == JobState.DEFINE) {
      submit();
    }
    if (verbose) {
      monitorAndPrintJob();
    } else {
      // get the completion poll interval from the client.
      int completionPollIntervalMillis =
        Job.getCompletionPollInterval(cluster.getConf());
      while (!isComplete()) {
        try {
          Thread.sleep(completionPollIntervalMillis);
        } catch (InterruptedException ie) {
        }
      }
    }
    return isSuccessful();
  }
```

submit()方法

```
public void submit()
         throws IOException, InterruptedException, ClassNotFoundException {
    ensureState(JobState.DEFINE);
    setUseNewAPI();
    connect();
    final JobSubmitter submitter =
        getJobSubmitter(cluster.getFileSystem(), cluster.getClient());
    status = ugi.doAs(new PrivilegedExceptionAction<JobStatus>() {
      public JobStatus run() throws IOException, InterruptedException,
      ClassNotFoundException {
        return submitter.submitJobInternal(Job.this, cluster);
      }
    });
    state = JobState.RUNNING;
    LOG.info("The url to track the job: " + getTrackingURL());
   }
```

JobSubmitter类

submitJobInternal()方法

```
JobStatus submitJobInternal(Job job, Cluster cluster)
```

```java
  throws ClassNotFoundException, InterruptedException, IOException {

    //validate the jobs output specs
    checkSpecs(job);

    Configuration conf = job.getConfiguration();
    addMRFrameworkToDistributedCache(conf);

    Path jobStagingArea = JobSubmissionFiles.getStagingDir(cluster, conf);
    //configure the command line options correctly on the submitting dfs
    InetAddress ip = InetAddress.getLocalHost();
    if (ip != null) {
      submitHostAddress = ip.getHostAddress();
      submitHostName = ip.getHostName();
      conf.set(MRJobConfig.JOB_SUBMITHOST,submitHostName);
      conf.set(MRJobConfig.JOB_SUBMITHOSTADDR,submitHostAddress);
    }
    JobID jobId = submitClient.getNewJobID();
    job.setJobID(jobId);
    Path submitJobDir = new Path(jobStagingArea, jobId.toString());
    JobStatus status = null;
    try {
      conf.set(MRJobConfig.USER_NAME,
          UserGroupInformation.getCurrentUser().getShortUserName());
      conf.set("hadoop.http.filter.initializers",

"org.apache.hadoop.yarn.server.webproxy.amfilter.AmFilterInitializer");
      conf.set(MRJobConfig.MAPREDUCE_JOB_DIR, submitJobDir.toString());
      LOG.debug("Configuring job " + jobId + " with " + submitJobDir
          + " as the submit dir");
      // get delegation token for the dir
      TokenCache.obtainTokensForNamenodes(job.getCredentials(),
          new Path[] { submitJobDir }, conf);

      populateTokenCache(conf, job.getCredentials());

      // generate a secret to authenticate shuffle transfers
      if (TokenCache.getShuffleSecretKey(job.getCredentials()) == null) {
        KeyGenerator keyGen;
        try {
          keyGen = KeyGenerator.getInstance(SHUFFLE_KEYGEN_ALGORITHM);
          keyGen.init(SHUFFLE_KEY_LENGTH);
        } catch (NoSuchAlgorithmException e) {
          throw new IOException("Error generating shuffle secret key", e);
        }
        SecretKey shuffleKey = keyGen.generateKey();
        TokenCache.setShuffleSecretKey(shuffleKey.getEncoded(),
            job.getCredentials());
      }
      if (CryptoUtils.isEncryptedSpillEnabled(conf)) {
        conf.setInt(MRJobConfig.MR_AM_MAX_ATTEMPTS, 1);
        LOG.warn("Max job attempts set to 1 since encrypted intermediate" +
                "data spill is enabled");
      }

      copyAndConfigureFiles(job, submitJobDir);

      Path submitJobFile = JobSubmissionFiles.getJobConfPath(submitJobDir);
      // Create the splits for the job
      LOG.debug("Creating splits at " + jtFs.makeQualified(submitJobDir));
      int maps = writeSplits(job, submitJobDir);
      conf.setInt(MRJobConfig.NUM_MAPS, maps);
      LOG.info("number of splits:" + maps);

      int maxMaps = conf.getInt(MRJobConfig.JOB_MAX_MAP,
          MRJobConfig.DEFAULT_JOB_MAX_MAP);
      if (maxMaps >= 0 && maxMaps < maps) {
        throw new IllegalArgumentException("The number of map tasks " + maps +
            " exceeded limit " + maxMaps);
      }

      // write "queue admins of the queue to which job is being submitted"
      // to job file.
      String queue = conf.get(MRJobConfig.QUEUE_NAME,
          JobConf.DEFAULT_QUEUE_NAME);
      AccessControlList acl = submitClient.getQueueAdmins(queue);
      conf.set(toFullPropertyName(queue,
          QueueACL.ADMINISTER_JOBS.getAclName()), acl.getAclString());

      // removing jobtoken referrals before copying the jobconf to HDFS
      // as the tasks don't need this setting, actually they may break
      // because of it if present as the referral will point to a
      // different job.
      TokenCache.cleanUpTokenReferral(conf);

      if (conf.getBoolean(
          MRJobConfig.JOB_TOKEN_TRACKING_IDS_ENABLED,
          MRJobConfig.DEFAULT_JOB_TOKEN_TRACKING_IDS_ENABLED)) {
        // Add HDFS tracking ids
        ArrayList<String> trackingIds = new ArrayList<String>();
        for (Token<? extends TokenIdentifier> t :
            job.getCredentials().getAllTokens()) {
          trackingIds.add(t.decodeIdentifier().getTrackingId());
        }
        conf.setStrings(MRJobConfig.JOB_TOKEN_TRACKING_IDS,
            trackingIds.toArray(new String[trackingIds.size()]));
      }

      // Set reservation info if it exists
      ReservationId reservationId = job.getReservationId();
      if (reservationId != null) {
        conf.set(MRJobConfig.RESERVATION_ID, reservationId.toString());
      }

      // Write job file to submit dir
      writeConf(conf, submitJobFile);

      //
      // Now, actually submit the job (using the submit name)
      //
      printTokens(jobId, job.getCredentials());
      status = submitClient.submitJob(
          jobId, submitJobDir.toString(), job.getCredentials());
      if (status != null) {
        return status;
      } else {
        throw new IOException("Could not launch job");
      }
    } finally {
      if (status == null) {
        LOG.info("Cleaning up the staging area " + submitJobDir);
        if (jtFs != null && submitJobDir != null)
          jtFs.delete(submitJobDir, true);

      }
    }
  }
```

writeSplit()方法

```
private int writeSplits(org.apache.hadoop.mapreduce.JobContext job,
      Path jobSubmitDir) throws IOException,
      InterruptedException, ClassNotFoundException {
    JobConf jConf = (JobConf)job.getConfiguration();
    int maps;
    if (jConf.getUseNewMapper()) {
      maps = writeNewSplits(job, jobSubmitDir);
    } else {
      maps = writeOldSplits(jConf, jobSubmitDir);
    }
    return maps;
  }
```

writeNewSplits()方法

```
private <T extends InputSplit>
  int writeNewSplits(JobContext job, Path jobSubmitDir) throws IOException,
      InterruptedException, ClassNotFoundException {
    Configuration conf = job.getConfiguration();
    InputFormat<?, ?> input =
      ReflectionUtils.newInstance(job.getInputFormatClass(), conf);

    List<InputSplit> splits = input.getSplits(job);
    T[] array = (T[]) splits.toArray(new InputSplit[splits.size()]);

    // sort the splits into order based on size, so that the biggest
    // go first
    Arrays.sort(array, new SplitComparator());
    JobSplitWriter.createSplitFiles(jobSubmitDir, conf,
        jobSubmitDir.getFileSystem(conf), array);
    return array.length;
  }
```

FileInputFormat类

getSplits()方法

```
public List<InputSplit> getSplits(JobContext job) throws IOException {
    StopWatch sw = new StopWatch().start();
    long minSize = Math.max(getFormatMinSplitSize(), getMinSplitSize(job));
    long maxSize = getMaxSplitSize(job);
```

```java
    // generate splits
    List<InputSplit> splits = new ArrayList<InputSplit>();
    List<FileStatus> files = listStatus(job);

    boolean ignoreDirs = !getInputDirRecursive(job)
      &&
job.getConfiguration().getBoolean(INPUT_DIR_NONRECURSIVE_IGNORE_SUBDIRS, false);
    for (FileStatus file: files) {
      if (ignoreDirs && file.isDirectory()) {
        continue;
      }
      Path path = file.getPath();
      long length = file.getLen();
      if (length != 0) {
        BlockLocation[] blkLocations;
        if (file instanceof LocatedFileStatus) {
          blkLocations = ((LocatedFileStatus) file).getBlockLocations();
        } else {
          FileSystem fs = path.getFileSystem(job.getConfiguration());
          blkLocations = fs.getFileBlockLocations(file, 0, length);
        }
        if (isSplitable(job, path)) {
          long blockSize = file.getBlockSize();
          long splitSize = computeSplitSize(blockSize, minSize, maxSize);

          long bytesRemaining = length;
          while (((double) bytesRemaining)/splitSize > SPLIT_SLOP) {
            int blkIndex = getBlockIndex(blkLocations, length-bytesRemaining);
            splits.add(makeSplit(path, length-bytesRemaining, splitSize,
                        blkLocations[blkIndex].getHosts(),
                        blkLocations[blkIndex].getCachedHosts()));
            bytesRemaining -= splitSize;
          }

          if (bytesRemaining != 0) {
            int blkIndex = getBlockIndex(blkLocations, length-bytesRemaining);
            splits.add(makeSplit(path, length-bytesRemaining, bytesRemaining,
                       blkLocations[blkIndex].getHosts(),
                       blkLocations[blkIndex].getCachedHosts()));
          }
        } else { // not splitable
          if (LOG.isDebugEnabled()) {
            // Log only if the file is big enough to be splitted
            if (length > Math.min(file.getBlockSize(), minSize)) {
              LOG.debug("File is not splittable so no parallelization "
                  + "is possible: " + file.getPath());
            }
          }
          splits.add(makeSplit(path, 0, length, blkLocations[0].getHosts(),
                      blkLocations[0].getCachedHosts()));
        }
      } else {
        //Create empty hosts array for zero length files
        splits.add(makeSplit(path, 0, length, new String[0]));
      }
    }
    // Save the number of input files for metrics/loadgen
    job.getConfiguration().setLong(NUM_INPUT_FILES, files.size());
    sw.stop();
    if (LOG.isDebugEnabled()) {
      LOG.debug("Total # of splits generated by getSplits: " + splits.size()
          + ", TimeTaken: " + sw.now(TimeUnit.MILLISECONDS));
    }
    return splits;
  }
```