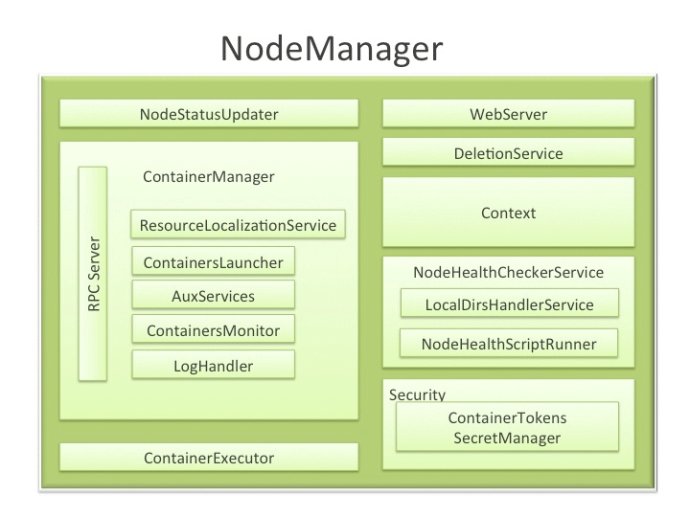
YARN NM AuxiliaryService

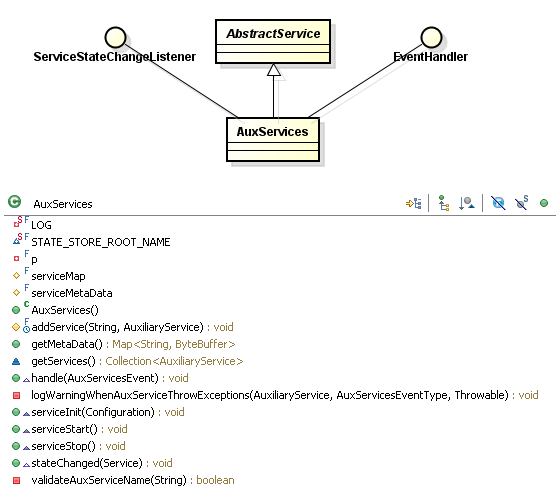
AuxiliaryService是NodeManager启动的通用服务，在NM中的位置如下图所示：



用户通过配置附属服务（AuxiliaryService）的方式扩展自己的功能，这使得每一个节点能够定制一些特定框架须要的服务。附属服务要在NM启动前配置好，并由NM统一启动和关闭。

# AuxServices

AuxServices用于处理AuxiliaryService服务集合,在NM中可以有多个AuxiliaryService，默认值是mapreduce\_shuffle，即MRv2中的ShuffleHandler，将节点内的map输出文件提供给Reducer，AuxServices类图如下所示：



1）AuxServices对象启动，从YarnConfiguration.NM\_AUX\_SERVICES，即参数：

*[yarn.nodemanager.aux-services](http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-common/yarn-default.xml)*

获得附属服务名，

2）获取AlluxiliaryService，从配置YarnConfiguration.NM\_AUX\_SERVICE\_FMT，即参数：

*[yarn.nodemanager.aux-services](http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-common/yarn-default.xml).%s.class*

获得对应的类型，例如mapreduce\_shuffle，即MRv2中的ShuffleHandler:

*<property>  
 <name>yarn.nodemanager.aux-services.mapreduce\_shuffle.class</name>  
 <value>org.apache.hadoop.mapred.ShuffleHandler</value>  
</property>*

3）将服务置入serviceMap并调用init方法对服务进行初始化，其执行如下：

*AuxiliaryService s = AuxiliaryServiceWithCustomClassLoader.getInstance(  
 conf, className, appLocalClassPath);*

*serviceMap.put(name, service); //将服务至于serviceMap并调用init方法进行服务初始化*

Hadoop的核心是事件驱动系统，AuxServices实现ServiceStateChangeListener及EventHandler接口，用于处理AuxServicesEventType事件：



在handle(AuxServiceEvent event)方法中，每个事件与AuxiliaryService中的API调用相关联，只要AuxService收到一个APPLICATION\_INIT事件，对应AuxiliaryService中initializeApp方法就会被调用，handle核心执行如下：

*switch (event.getType()) {*

*case APPLICATION\_INIT:*

*AuxiliaryService service = null;*

*try {*

*service = serviceMap.get(event.getServiceID());*

*service*

*.initializeApplication(new ApplicationInitializationContext(event*

*.getUser(), event.getApplicationID(), event.getServiceData()));*

*} ...*

*case APPLICATION\_STOP:*

*for (AuxiliaryService serv : serviceMap.values()) {*

*try {*

*serv.stopApplication(new ApplicationTerminationContext(event*

*.getApplicationID()));*

*}....*

*case CONTAINER\_INIT:*

*for (AuxiliaryService serv : serviceMap.values()) {*

*try {*

*serv.initializeContainer(new ContainerInitializationContext(*

*event.getContainer().getUser(),*

*event.getContainer().getContainerId(),*

*event.getContainer().getResource(), event.getContainer()*

*.getContainerTokenIdentifier().getContainerType()));*

*} ......*

*case CONTAINER\_STOP:*

*for (AuxiliaryService serv : serviceMap.values()) {*

*try {*

*serv.stopContainer(new ContainerTerminationContext(*

*event.getUser(), event.getContainer().getContainerId(),*

*event.getContainer().getResource(), event.getContainer()*

*.getContainerTokenIdentifier().getContainerType()));*

*} ......*

*default:*

*throw new RuntimeException("Unknown type: " + event.getType());*

*}*

*}*

NodeManager类包含ContainerManagerImpl对象变量，而ContainerManagerImpl类包含AuxServices对象变量，其有AsyncDispatcher，分发AuxServicesEventType类型事件：

*dispatcher.register(AuxServicesEventType.class, auxiliaryServices);*

ContainerImpl（Container启动）后，会调用RequestResourcesTransition:

*Map<String,ByteBuffer> csd = ctxt.getServiceData();*

*if (csd != null) {*

*// This can happen more than once per Application as each container may*

*// have distinct service data*

*for (Map.Entry<String,ByteBuffer> service : csd.entrySet()) {*

*container.dispatcher.getEventHandler().handle(*

*new AuxServicesEvent(AuxServicesEventType.APPLICATION\_INIT,*

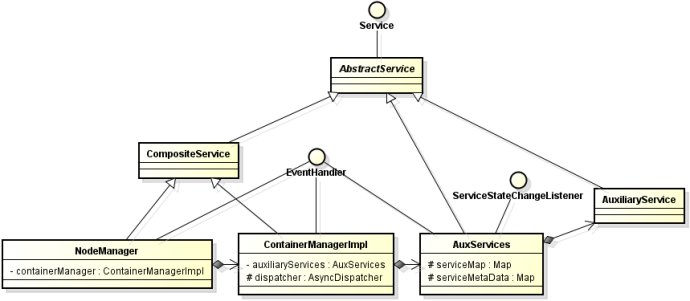
*container.user, container.containerId*

*.getApplicationAttemptId().getApplicationId(),*

*service.getKey().toString(), service.getValue()));*

*}}*

事件传递的AuxServices相关类图如下：

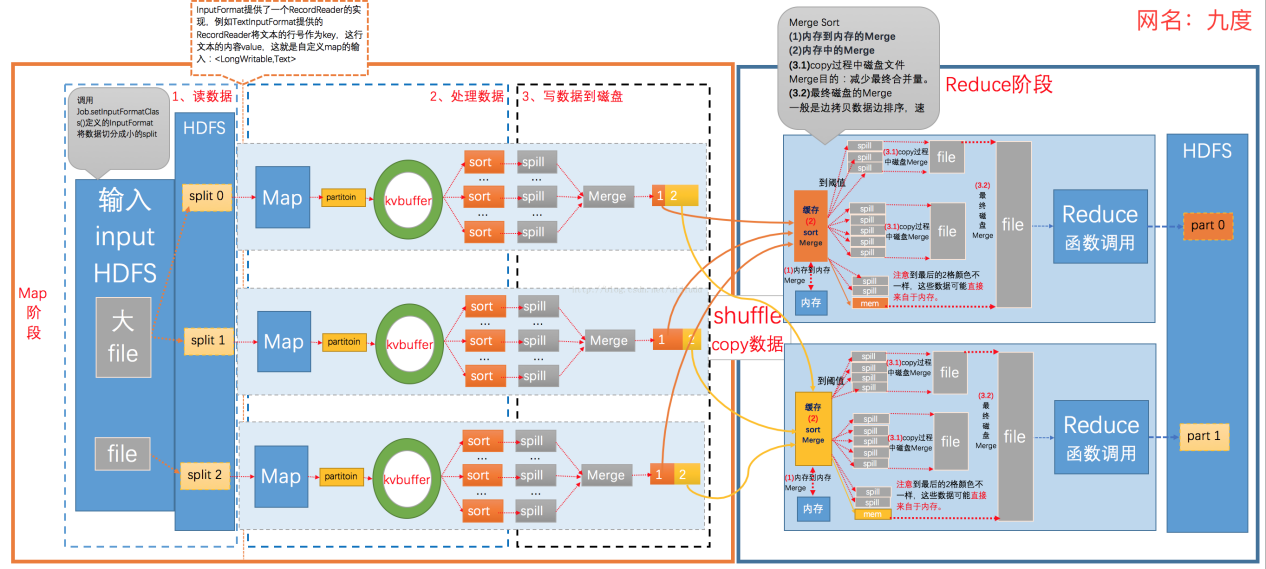


AuxServicesEventType.APPLICATION\_STOP事件在ApplicationImpl类中被创建，由节点管理器中应用表述的状态机触发。其他AuxServicesEventType事件，如APPLICATION\_INIT、CONTAINER\_INIT和CONTAINER\_STOP，在ContainerImpl类中随着容器的生命周期被创建。

参考文献：http://blog.sina.com.cn/s/blog\_72ef7bea0102vfcf.html

# MapReduce ShuffleHandler

MapReduce的执行过程如下图所示：



其中Shuffle是MapReduce框架中特定阶段，将Map输出结果分发到Reduce以实现结果汇总，在MRv2中通过ShuffleHandler实现文件的输出，其配置参数为：

*<property>  
 <name>yarn.nodemanager.aux-services</name>  
 <value>mapreduce\_shuffle</value>-  
</property>*

*<property>  
 <name>yarn.nodemanager.aux-services.mapreduce\_shuffle.class</name>  
 <value>org.apache.hadoop.mapred.ShuffleHandler</value>  
</property>*

ShuffleHandler是Hadoop内嵌的AuxiliaryService，其本质是HTTP Server，NodeManager启动时启动ShuffleHandler，其类图如下：



## 2.1 Shuffle Server端处理逻辑

### **2.1.1 Shuffler Server执行**

ShuffleHandler使用Netty作为Server，其中Shuffle作为Handler接收Reduce的连接并返回数据，启动过程如下：

*protected void serviceStart() throws Exception {*

*......*

*ServerBootstrap bootstrap = new ServerBootstrap(selector);*

*try {*

*pipelineFact = new HttpPipelineFactory(conf, timer);*

*} .....*

*bootstrap.setOption("backlog", conf.getInt(SHUFFLE\_LISTEN\_QUEUE\_SIZE,*

*DEFAULT\_SHUFFLE\_LISTEN\_QUEUE\_SIZE));*

*bootstrap.setOption("child.keepAlive", true);*

*bootstrap.setPipelineFactory(pipelineFact);*

*port = conf.getInt(SHUFFLE\_PORT\_CONFIG\_KEY, DEFAULT\_SHUFFLE\_PORT);*

*Channel ch = bootstrap.bind(new InetSocketAddress(port));*

*accepted.add(ch);*

*port = ((InetSocketAddress)ch.getLocalAddress()).getPort();*

*conf.set(SHUFFLE\_PORT\_CONFIG\_KEY, Integer.toString(port));*

*pipelineFact.SHUFFLE.setPort(port);*

*super.serviceStart();*

*.....*

*}*

在pipelineFact中封装了Shuffle，其实现类SimpleChannelUpstreamHandler，业务的主要实现类，接收Reduce的Shuffle请求并返回数据。

2）接收客户端请求

在Shuffle中，维护了pathCache:LoadingCache<AttemptPathIdentifier,AttemptPathInfo>，即Map的输出结果，接收到客户端请求后，处理逻辑如下：

*@Override*

*public void messageReceived(ChannelHandlerContext ctx, MessageEvent evt)*

*throws Exception {*

*//根据Shuffle及Request，获取mapIds,reduceId，生成mapOutputInfoMap*

*ReduceContext reduceContext = new ReduceContext(mapIds, reduceId, ctx,*

*user, mapOutputInfoMap, jobId, keepAlive);*

*for (int i = 0; i < Math.min(maxSessionOpenFiles, mapIds.size()); i++) {*

*//将map文件数据发送给Reduce端*

*ChannelFuture nextMap = sendMap(reduceContext);*

*if(nextMap == null) {*

*return;*

*}*

*}*

*}*

3）发送文件

文件的发送过程为sendMap，核心代码如下：

*protected ChannelFuture sendMapOutput(ChannelHandlerContext ctx, Channel ch,*

*String user, String mapId, int reduce, MapOutputInfo mapOutputInfo)*

*throws IOException {*

*....*

*final File spillfile =*

*new File(mapOutputInfo.mapOutputFileName.toString());*

*RandomAccessFile spill;*

*try {*

*spill = SecureIOUtils.openForRandomRead(spillfile, "r", user, null);*

*} .....*

*ChannelFuture writeFuture;*

*if (ch.getPipeline().get(SslHandler.class) == null) {*

*final FadvisedFileRegion partition = new FadvisedFileRegion(spill,*

*info.startOffset, info.partLength, manageOsCache, readaheadLength,*

*readaheadPool, spillfile.getAbsolutePath(),*

*shuffleBufferSize, shuffleTransferToAllowed);*

*writeFuture = ch.write(partition);*

*writeFuture.addListener(new ChannelFutureListener() {*

*// TODO error handling; distinguish IO/connection failures,*

*// attribute to appropriate spill output*

*@Override*

*public void operationComplete(ChannelFuture future) {*

*if (future.isSuccess()) {*

*partition.transferSuccessful();*

*}*

*partition.releaseExternalResources();*

*}*

*}); } ...}*

### **2.1.2 MRAppMaster记录Shuffle URL**

MapTask结束后，将Map输出文件所在URL(包括Shuffle 的主机和端口)发送给AM，执行如下：

*public void done(TaskUmbilicalProtocol umbilical,TaskReporter reporter) {*

*...*

*sendDone(umbilical); <= umbilical.done(getTaskID());*

*}*

在TaskImpl状态机中设置Shuffle Port，执行如下：

*private void handleTaskAttemptCompletion(TaskAttemptId attemptId,*

*TaskAttemptCompletionEventStatus status) {*

*TaskAttempt attempt = attempts.get(attemptId);*

*...*

*if (attempt.getNodeHttpAddress() != null) {*

*TaskAttemptCompletionEvent tce = recordFactory*

*.newRecordInstance(TaskAttemptCompletionEvent.class);*

*tce.setEventId(-1);*

*String scheme = (encryptedShuffle) ? "https://" : "http://";*

*tce.setMapOutputServerAddress(StringInterner.weakIntern(scheme*

*+ attempt.getNodeHttpAddress().split(":")[0] + ":"*

*+ attempt.getShufflePort()));*

*tce.setStatus(status);*

*tce.setAttemptId(attempt.getID());*

*int runTime = 0;*

*if (attempt.getFinishTime() != 0 && attempt.getLaunchTime() !=0)*

*runTime = (int)(attempt.getFinishTime() - attempt.getLaunchTime());*

*tce.setAttemptRunTime(runTime);*

*eventHandler.handle(new JobTaskAttemptCompletedEvent(tce));*

*}*

*}*

MRAppMaster将信息保存在对应的Job中

*TaskAttemptCompletionEvent tce =*

*((JobTaskAttemptCompletedEvent) event).getCompletionEvent();*

*tce.setEventId(job.taskAttemptCompletionEvents.size());*

*job.taskAttemptCompletionEvents.add(tce);*

在TaskAttemptCompletionEvent中记录MapOutputServerAddress

### **2.1.3 ShufflePort**

AM发送Container启动请求后，NM响应ShufflePort，ContainerLaunchImpl的执行如下：

*StartContainerRequest startRequest =*

*StartContainerRequest.newInstance(containerLaunchContext,*

*event.getContainerToken());*

*List<StartContainerRequest> list = new ArrayList<StartContainerRequest>();*

*list.add(startRequest);*

*StartContainersRequest requestList = StartContainersRequest.newInstance(list);*

*StartContainersResponse response =*

*proxy.getContainerManagementProtocol().startContainers(requestList);*

*if (response.getFailedRequests() != null*

*&& response.getFailedRequests().containsKey(containerID)) {*

*throw response.getFailedRequests().get(containerID).deSerialize();*

*}*

*ByteBuffer portInfo =*

*response.getAllServicesMetaData().get(*

*ShuffleHandler.MAPREDUCE\_SHUFFLE\_SERVICEID);*

*int port = -1;*

*if(portInfo != null) {*

*port = ShuffleHandler.deserializeMetaData(portInfo);*

*}*

NM中的ContainerManagerImpl#startContainers执行如下：

*return StartContainersResponse  
 .newInstance(getAuxServiceMetaData(), succeededContainers,  
 failedContainers);*

getAuxServiceMetaData调用ShuffleHandler如下：

*public synchronized ByteBuffer getMetaData() {  
 try {  
 return serializeMetaData(port);   
 } catch (IOException e) {  
}*

## 2.2 Shuffle客户端处理逻辑

Reduce启动后，与NM的ShuffleHandler交互，并获取文件信息，其Shuffle处理类的配置如下：

*<property>*

*<name>mapreduce.job.reduce.shuffle.consumer.plugin.class</name>*

*<value>org.apache.hadoop.mapreduce.task.reduce.Shuffle</value>*

*</property>*

Shuffle的类图如下所示：



核心类为Shuffle，其通过EventFetcher和Fetcher获取Task信息及Map输出数据，执行如下：

*@Override*

*public RawKeyValueIterator run() throws IOException, InterruptedException {*

*// 获取Map End事件*

*final EventFetcher<K,V> eventFetcher =*

*new EventFetcher<K,V>(reduceId, umbilical, scheduler, this,*

*maxEventsToFetch);*

*eventFetcher.start();*

*//获取Map Output数据*

*Fetcher<K,V>[] fetchers = new Fetcher[numFetchers];*

*if (isLocal) {*

*fetchers[0] = new LocalFetcher<K, V>(jobConf, reduceId, scheduler,*

*merger, reporter, metrics, this, reduceTask.getShuffleSecret(),*

*localMapFiles);*

*fetchers[0].start();*

*} else {*

*for (int i=0; i < numFetchers; ++i) {*

*fetchers[i] = new Fetcher<K,V>(jobConf, reduceId, scheduler, merger,*

*reporter, metrics, this, reduceTask.getShuffleSecret());*

*fetchers[i].start();*

*}*

*}*

*// 等待结束*

*while (!scheduler.waitUntilDone(PROGRESS\_FREQUENCY)) {*

*reporter.progress();*

*synchronized (this) {*

*if (throwable != null) {*

*throw new ShuffleError("error in shuffle in " + throwingThreadName,*

*throwable);*

*}*

*}}*

*//返回结果*

*RawKeyValueIterator kvIter = null;*

*try {*

*kvIter = merger.close();*

*} catch (Throwable e) {*

*throw new ShuffleError("Error while doing final merge " , e);*

*}*

*.....*

*return kvIter;}*

### 2.2.1 与AM交互获取Map Hosts

Shuffle通过EventFetcher线程，从am中所有job完成的map的Event，将这些保存到ShuffleSchedulerImpl实例中主要的信息：

*private Map<String, MapHost> mapLocations = new HashMap<String, MapHost>();  
private Set<MapHost> pendingHosts = new HashSet<MapHost>();*

MapHost信息包括：

*private State state = State.IDLE;  
private final String hostName; //主机名  
private final String baseUrl; //url  
private List<TaskAttemptID> maps = new ArrayList<TaskAttemptID>(); //attempId*

1）Reduce EventFetcher的执行

*protected int getMapCompletionEvents(){*

*int numNewMaps = 0;*

*TaskCompletionEvent events[] = null;*

*do {*

*MapTaskCompletionEventsUpdate update =*

*umbilical.getMapCompletionEvents( //TaskUmbilicalProtocol*

*(org.apache.hadoop.mapred.JobID)reduce.getJobID(),*

*fromEventIdx,*

*maxEventsToFetch,*

*(org.apache.hadoop.mapred.TaskAttemptID)reduce);*

*events = update.getMapTaskCompletionEvents();*

*for (TaskCompletionEvent event : events) {*

*scheduler.resolve(event); //解析Event，将map输出设置到scheduler中*

*if (TaskCompletionEvent.Status.SUCCEEDED == event.getTaskStatus()) {*

*++numNewMaps;*

*}*

*}*

*} while (events.length == maxEventsToFetch);*

*return numNewMaps;*

*}*

2）MRAppMaster返回信息

TaskUmbilicalProtocol协议为Task与MRAppMaster的RPC协议，其AM中通过TaskAttemptListener来提供Task End信息

*public MapTaskCompletionEventsUpdate getMapCompletionEvents(  
 JobID jobIdentifier, int startIndex, int maxEvents,  
 TaskAttemptID taskAttemptID) throws IOException {*

*org.apache.hadoop.mapreduce.v2.api.records.TaskAttemptId attemptID =  
 TypeConverter.toYarn(taskAttemptID);  
 TaskCompletionEvent[] events =  
 context.getJob(attemptID.getTaskId().getJobId()).getMapAttemptCompletionEvents(  
 startIndex, maxEvents);  
  
 taskHeartbeatHandler.progressing(attemptID);  
   
 return new MapTaskCompletionEventsUpdate(events, shouldReset);  
}*

信息维护在RunningAppContext#jobs <= Map<JobId,Job>中，其更新见上文。

3）ShuffleSchedulerImpl记录Map信息

ShuffleSchedulerImpl#resolve(TaskCompletionEvent)

*case SUCCEEDED:  
 URI u = getBaseURI(reduceId, event.getTaskTrackerHttp());  
 addKnownMapOutput(u.getHost() + ":" + u.getPort(),  
 u.toString(),  
 event.getTaskAttemptId());  
 maxMapRuntime = Math.max(maxMapRuntime, event.getTaskRunTime());  
 break;*

### **2.2.2 与Map交互获取数据**

通过Fetcher获取数据，通过scheduler.getHost获取map输出主机，然后调用copyFromHost获取数据：

*protected void copyFromHost(MapHost host) throws IOException {*

*//获取该host上所有的End Task*

*List<TaskAttemptID> maps = scheduler.getMapsForHost(host);*

*//获取MapOutput的URL*

*URL url = getMapOutputURL(host, maps);*

*DataInputStream input = null;*

*try {*

*input = openShuffleUrl(host, remaining, url);*

*while (!remaining.isEmpty() && failedTasks == null) {*

*try {*

*//获取数据*

*failedTasks = copyMapOutput(host, input, remaining, fetchRetryEnabled);*

*} ...*

*}*

通过URL将请求发送NM ShuffleServer上。

https://blog.csdn.net/u014393917/article/details/25792325

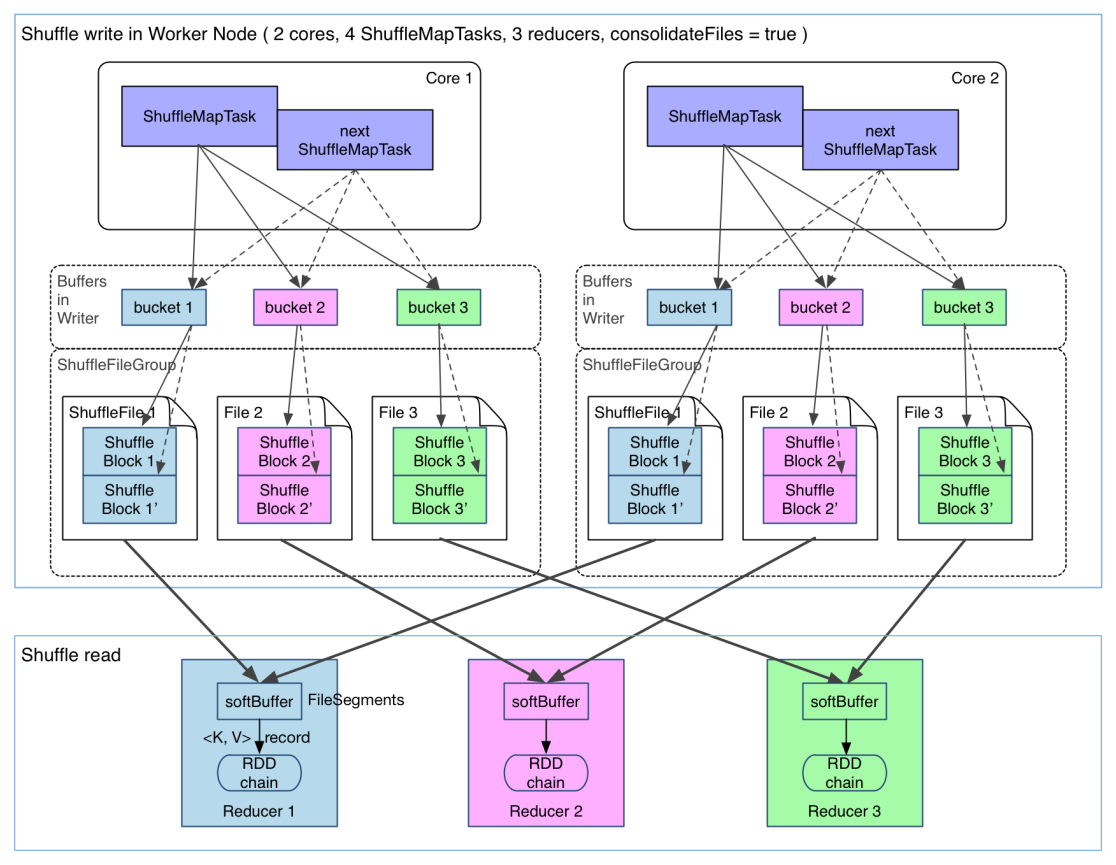
## 2.3 Shuffle执行流程



具体执行流程见上面代码分析。

# Spark YarnShuffleService

Spark是MapReduce框架的一种实现，其涉及到Shuffle的Job的执行如下图所示：



执行过程和Hadoop MapReduce类似，在Map阶段将结果写入到本地磁盘中，Reduce启动后通过Shuffle将数据读到本地进行处理。Shuffle是整个Spark中最消耗性能的阶段。

Spark on YARN是目前常用的Spark程序运行方法，可以使用External Shuffle Server来提高性能，其实现类为YarnShuffleService，通过这种方式较少Executor的压力，在GC的时候不影响其他Executor的执行，配置参数如下：

*<property>*

*<name>yarn.nodemanager.aux-services</name>*

*<value>spark\_shuffle</value>*

*</property>*

*<property>*

*<name>yarn.nodemanager.aux-services.spark\_shuffle.class</name>*

*<value>org.apache.spark.network.yarn.YarnShuffleService</value>*

*</property>*

*<property>*

*<name>spark.shuffle.service.port</name>*

*<value>7337</value>*

*</property>*

其类图如下：



YarnShuffleService通过NettyServer提供服务，业务逻辑处理类ExternalShuffleBlockHandler，核心代码如下：

*public void receive(TransportClient client, ByteBuffer message, RpcResponseCallback callback) {*

*BlockTransferMessage msgObj = BlockTransferMessage.Decoder.fromByteBuffer(message); handleMessage(msgObj, client, callback);*

*}*

主要的业务在handleMessage中进行处理

*protected void handleMessage(*

*BlockTransferMessage msgObj,*

*TransportClient client,*

*RpcResponseCallback callback) {*

*if (msgObj instanceof OpenBlocks) {*

*try {*

*OpenBlocks msg = (OpenBlocks) msgObj;*

*checkAuth(client, msg.appId);*

*long streamId = streamManager.registerStream(client.getClientId(),*

*new ManagedBufferIterator(msg.appId, msg.execId, msg.blockIds));*

*......*

*//发送数据*

*callback.onSuccess(new StreamHandle(streamId, msg.blockIds.length).toByteBuffer());*

*} else if (msgObj instanceof RegisterExecutor) {*

*final Timer.Context responseDelayContext =*

*metrics.registerExecutorRequestLatencyMillis.time();*

*try {*

*//Executor将其输出的数据注册到blockManager*

*RegisterExecutor msg = (RegisterExecutor) msgObj;*

*checkAuth(client, msg.appId);*

*blockManager.registerExecutor(msg.appId, msg.execId, msg.executorInfo);*

*callback.onSuccess(ByteBuffer.wrap(new byte[0]));*

*} finally {*

*responseDelayContext.stop();*

*}*

*} .....*

*}*

在executorInfo中封装了中间数据的信息：

*public final String[] localDirs;*

*public final int subDirsPerLocalDir;*

*public final String shuffleManager;*

https://github.com/jacksu/utils4s/blob/master/spark-knowledge/md/hash-shuffle.md

https://0x0fff.com/spark-architecture-shuffle/

Run the shuffle service inside the YARN NM as an AuxiliaryService

https://issues.apache.org/jira/browse/SPARK-3797