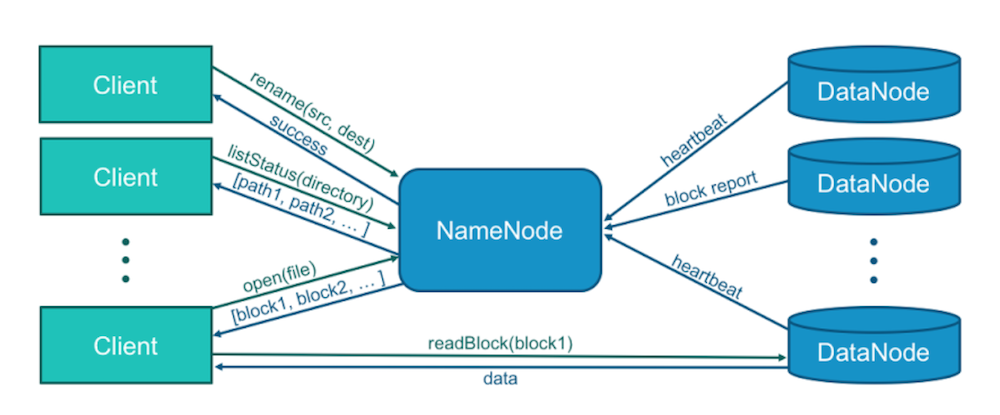
Linkedin Dynamometer

随着业务量的增长，需要对Hadoop集群规模的扩展，因此需要扩展性测试来进行资源的评估，但是在生产集群中进行性能及扩展性测试会影响业务。同时进行参数改变时，搭建相同规模的集群进行性能的测试也不现实。

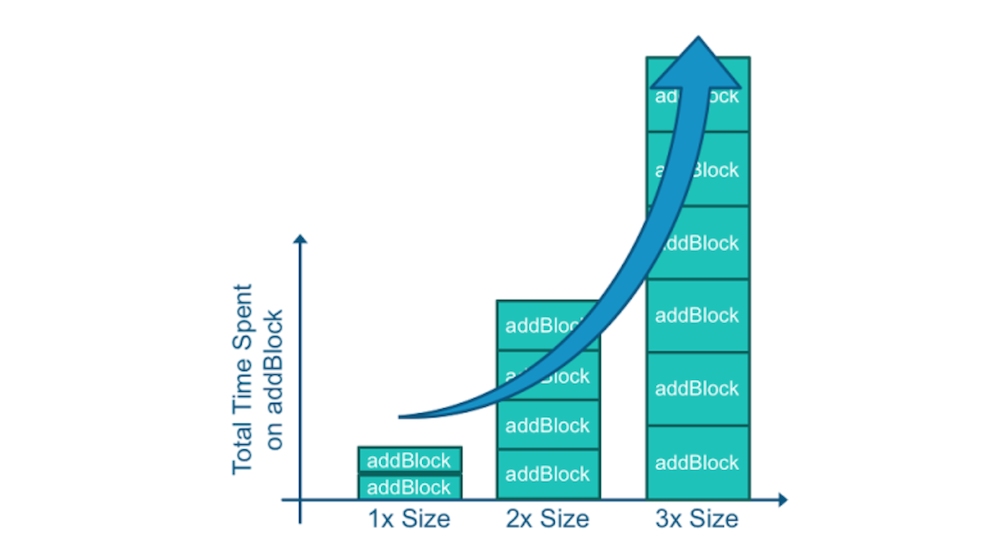
LinkedIn Dynamometer帮助企业在不使用大量基础架构的情况下对大规模Hadoop集群进行压力测试，用户使用Dynamometer在测试集群中执行相同类型的工作负载，从而获取Hadoop HDFS大规模集群运行的影响因素，HDFS的交互关系如下图所示：



在上图中，所有的文件读写等请求都需要从NameNode中获取元数据，而且是单一Server，因此是集群扩展的瓶颈点，主要影响因素：

* 由于数据分布在所有的DN上，NN维护所有的元数据而且管理所有的DN节点
* 当集群规模扩大时，文件、目录及DN信息相关的元数据会相应增加
* 当DN数量增加时，可以同时执行的Task增多，对NN的访问量会增加

下图是不同集群规模的情况下，addBlock所需要的时间：



Block Allocation的时间和集群规模是成比例的关系，操作的时间受到block及文件数目的影响，因此提前预测集群的规模很难。根据Hadoop的执行分析，性能的影响因素：

* 集群中DN的规模
* NN要管理的实体数目和结构（文件/目录及blocks）
* 客户端负载，请求的类型和数量

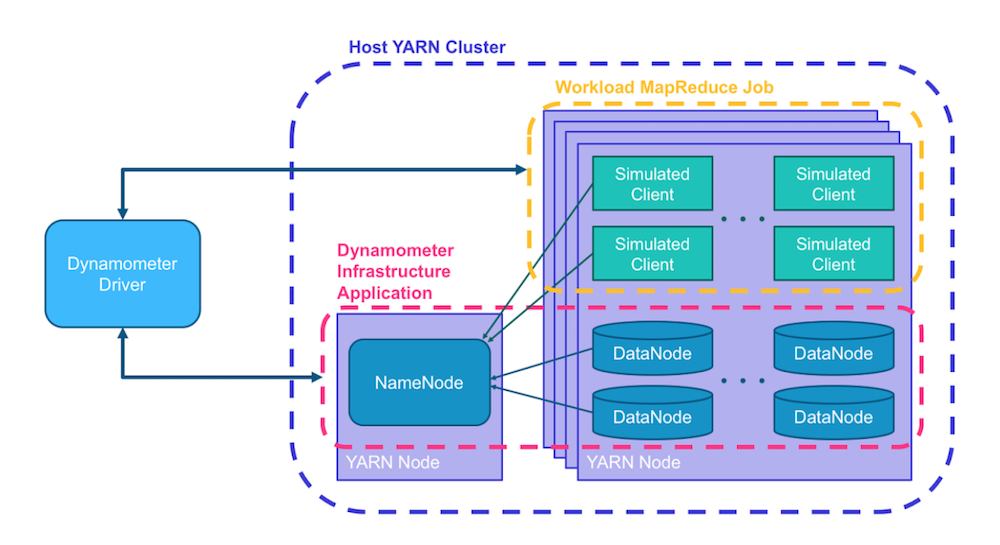
集群压力模拟测试工具要控制以上三方面的因素：在每个物理主机上运行多个DN进程，根据需要进行配置；启动管理相同元数据但是没有数据的NN；启动类似的客户端负载，不同请求类型所消耗的时间和资源是不同的，例如写请求所消耗的资源比读请求更大，大目录的list操作可能比单文件的操作所花费的时间超过几千倍，GC及其他参数也会影响其执行效率。

基于以上因素，Dynamometer要实现的功能如下：

* 模拟的Hadoop集群和生产集群相同，包括DN数目和文件的命名空间
* 在模拟集群中重复生产集群的相同负载，也可以动态调整负载
* 操作的便捷性，使用Dynamometer便捷的执行预期的变化来验证NN性能的变化
* Dynamometer和HDFS之间尽可能的解耦

# Dynamometer的系统架构

Dynamometer运行在YARN中，根据需要动态调整资源及将HDFS集群和host集群解耦。其系统架构图如下：



Dynamometer的构建包括三个主要组件：

1. HDFS集群仿真，从生产集群的NameNode中收集FsImage文件（包含所有的文件系统元数据）。通过Offline Image Viewer解析FsImage文件从中获取每个Block的元数据，然后将这些信息分发到仿真的节点上。
2. 在DN中运行SimulatedFsDataset，只保存Block的元数据，这种方式可以在物理节点上仿真多个DN
3. NN已经记录了所有服务的请求，也可以通过audit log获取，根据IP地址进行划分请求（相同节点上执行的请求是相关的）
4. 生产集群上每秒处理数千上万个请求，需要仿真多客户端来提交请求，为了尽可能的重复负载，会保证以原顺序执行请求。通过MR Job来启动客户端，每个Map处理步骤3中切分的Audit long文件，并重放其中的操作。
5. Dynamometer Driver来协调器各组件

# **Dynamometer的使用**

1. 下载源代码

*git clone https://github.com/linkedin/dynamometer.git*

项目结构如下所示：

*# tree -L 1*

.

├── *dynamometer-blockgen ，MR Job为DN生成输入文件，执行infra的预操作*

├── *dynamometer-infra，YARN App，启动Dyno-HDFS集群*

├── *dynamometer-workload， 启动MR Job来重放audit log*

├── *gradle*

......

1. 编译如下：

*./gradlew build*

在build目录下生成dynamometer-fat-VERSION.tar，解压后的目录如下：

*$ tree -L 2*

*.*

├── *bin*

   ├── *create-slim-hadoop-tar.sh*

   ├── *generate-block-lists.sh*

   ├── *parse-metrics.sh*

  |── *parse-start-timestamp.sh*

   ├── *start-dynamometer-cluster.sh*

   ├── *start-workload.sh*

   └── *upload-fsimage.sh*

├── *CONTRIBUTING.md*

├── *lib*

├── *LICENSE*

├── *NOTICE*

└── *README.md*

1. 使用步骤

* 从生产集群NN获取fsimage和相关文件:

fsimage\_TXID，元数据

fsimage\_TXID.md5: 含image文件的md5值

可以使用offline inage viewer来生成，执行命令如下：

*hdfs oiv -i fsimage\_TXID -o fsimage\_TXID.xml -p XML*

* 将生成的fsimage文件上传到hdfs

*upload-fsimage.sh 0000000000000000562 hdfs:///dyno/fsimage*

其中*0000000000000000562为txID，其对应文件fsimage\_0000000000000000562*

* 处理Hadoop tarball，要启动Dyno-NN和-DN构成的Hadoop集群，其中yarn功能是多余的jar，使用上面的命令进行处理

*./bin/create-slim-hadoop-tar.sh hadoop-2.8.2-bc1.4.0.tar.gz*

执行输出如下：

*Slimming hadoop-2.8.2-bc1.4.0.tar.gz; size before is 188M*

*Finished; size after is 53M*

Tarball大小有188M降低到53M

* 执行Block Generation Job，使用fsimage\_TXID.xml文件为每个Dyno-DN生成block列表，命令如下：

*./bin/generate-block-lists.sh*

*-fsimage\_input\_path hdfs:///dyno/fsimage/fsimage\_0000000000000000562.xml*

*-block\_image\_output\_dir hdfs:///dyno/blocks*

*-num\_reducers 1 -num\_datanodes 5*

在dyno/blocks目录中生成Blocks列表， -num\_datanodes用于配置测试DataNode数目。目录中文件内容如下：

*/dyno/blocks/\_SUCCESS*

*/dyno/blocks/dn0-a-0-r-00000*

*/dyno/blocks/dn1-a-0-r-00000*

*/dyno/blocks/dn2-a-0-r-00000*

*/dyno/blocks/dn3-a-0-r-00000*

*/dyno/blocks/dn4-a-0-r-00000*

每个DN对应一个文件

* 启动Dyno集群，执行命令如下：

*start-dynamometer-cluster.sh*

*-hadoop\_binary\_path hadoop-2.8.2-bc1.4.0.tar.gz*

*-conf\_path /usr/lib/hadoop/etc/hadoop*

*-fs\_image\_dir hdfs:///dyno/fsimage*

*-block\_list\_path hdfs:///dyno/blocks*

* 通过MR Job启动WorkLoad 运行程序，执行命令如下：

*./bin/start-workload.sh*

*-Dauditreplay.input-path hdfs:///dyno/audit\_logs/*

*-Dauditreplay.num-threads 50*

*-nn\_uri hdfs://namenode\_address:port/*

*-start\_time\_offset 5m*

*-mapper\_class\_name AuditReplayMapper*

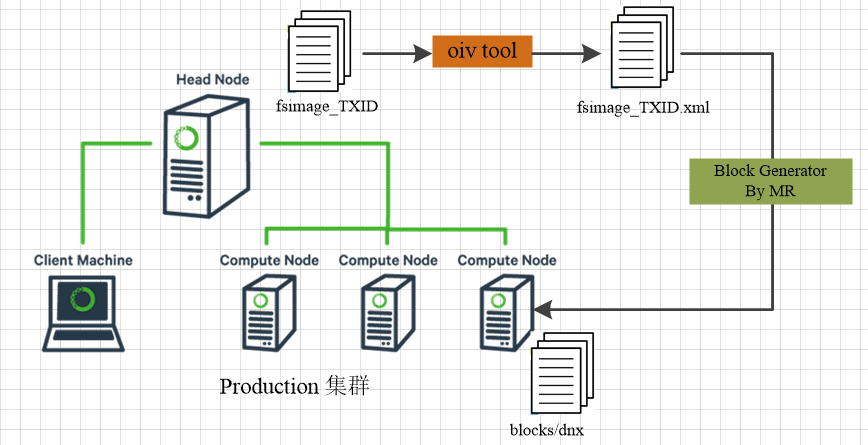
# **Dynamometer源码分析**

在Dynamometer中主要包含三个主要的组件：

1. Block Generator，MR Job用于生成每个Dyno-DN的输入文件，这是启动Dyno-HDFS的初始化文件
2. Infrastructure，YARN App，用于启动Dyno-HDFS集群
3. WorkLoad，MR Job用于重放audit log

## Block Generator

使用hdfs oiv命令将生产集群的Fsimage文件转换成XML格式文件，然后经过Block Generator生成对应每个DN要放置的Block，其执行流程如下：



1. 从生产集群中获取fsimage\_TXID文件，使用oiv tool解析成XML格式的fsimage文件
2. 在生产集群中通过Block Generator将xml格式的fsimage文件进行处理，根据配置的dn数目生成对应的blocks文件

生成的block文件作为启动Dyno集群的文件系统的初始化条件，处理模块为：

*dynamometer-blockgen*

执行命令如下：

*./bin/generate-block-lists.sh*

*-fsimage\_input\_path hdfs:///dyno/fsimage/fsimage\_0000000000000000562.xml*

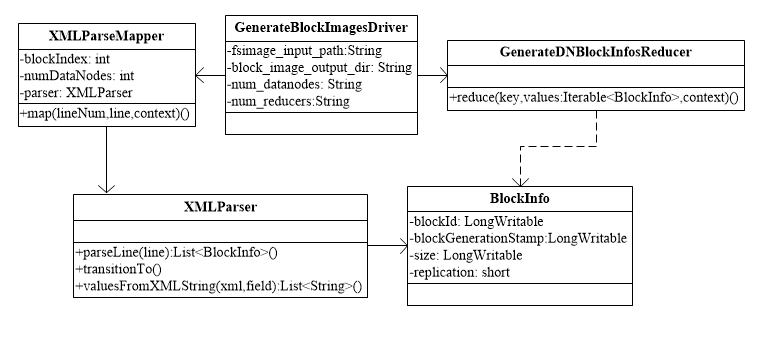
*-block\_image\_output\_dir hdfs:///dyno/blocks*

*-num\_reducers 1 -num\_datanodes 5*

参数描述：

|  |  |
| --- | --- |
| 参数 | 描述 |
| -fsimage\_input\_path | 待仿真集群的fsimage文件的输入文件 |
| -block\_image\_output | 为Dynamometer DN生成一个输出文件 |
| -num\_dataodes | Dyno DN的数目 |
| -num\_reducers | BlockGen程序的reducer个数 |

相关的类图如下：



执行后生成的Block文件内容如下：

*hdfs dfs -cat /dyno/blocks/dn0-a-0-r-00000*

*1073741840,1016,149168 //BlockInfo<blockId,timeStamp,size>*

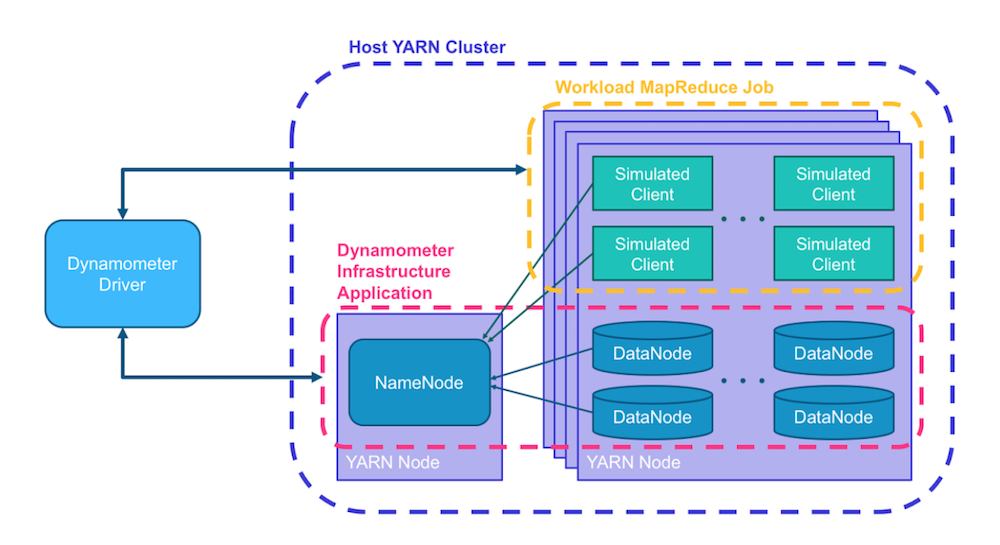
*1073741852,1028,33422*

*1073741881,1057,4838*

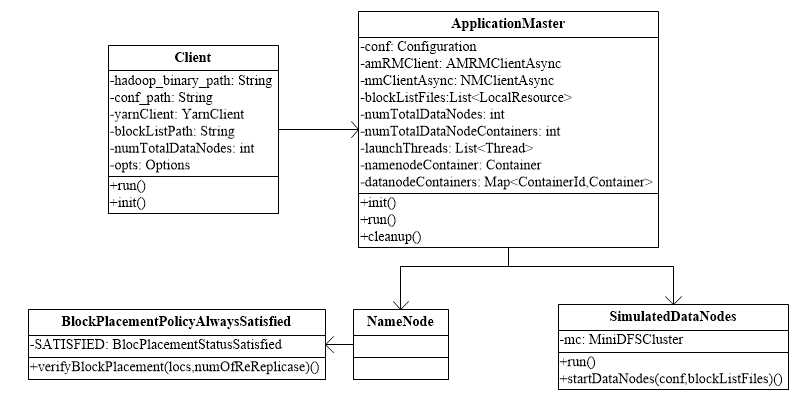
*......*

## **Infrastructure**

用于启动Dyno Cluster，其系统架构如下图所示：



1. Dynamometer Driver，对应类dynamometer#Client，用于向测试集群中提交Dyno-Cluster的构建命令
2. ApplicationMaster，Dyno-Cluster的管理Container，负责启动NameNode和DN
3. NameNode，启动Dyno-Cluster的NameNode，其中BlockPlacementPolicyAlwaysSatisfied
4. SimuatedDataNodes，一个节点可能要启动多个DN实例，调用MiniDFSCluster来启动多个DataNode，其中数据维护结构为：SimulatedFSDataset，相关的类图如下所示：



## **WorkLoad**

解析Audit Log，将记录的操作进行replay，命令如下：

*./bin/start-workload.sh*

*-Dauditreplay.input-path hdfs:///dyno/audit\_logs/*

*-Dauditreplay.num-threads 50*

*-nn\_uri hdfs://namenode\_address:port/*

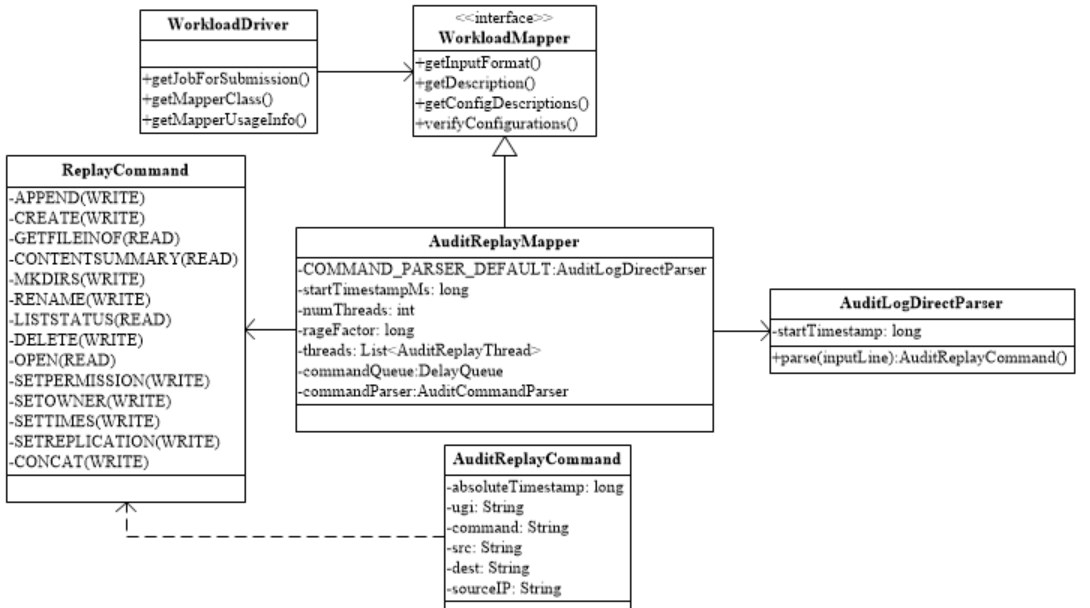
*-start\_time\_offset 5m*

*-mapper\_class\_name AuditReplayMapper*

相关参数：

|  |  |
| --- | --- |
| 参数 | 描述 |
| auditreplay.input-path | Audit Log文件目录 |
| auditreplay.num-threads | 执行线程数目 |
| auditreplay.create-blocks | create操作是否写入一个block |
| auditreplay.command-parser.class | 默认AuditLogDirectParser |
| auditreplay.rate-factor | 时间的伸缩参数 |
| start\_timestamp\_ms | 开始时间 |
| start\_time\_offset | 时间间隔 |
| nn\_uri | NameNode URI |

相关类图如下：



AuditReplayMapper在执行过程中，从AuditLog中读取日志并解析出ReplayCommand，加入到调度队列，AuditReplayThread执行如下：

*public void run() {*

*...*

*AuditReplayCommand cmd = commandQueue.take();*

*while (!cmd.isPoison()) {*

*replayCountersMap.get(REPLAYCOUNTERS.TOTALCOMMANDS).increment(1);*

*delay = cmd.getDelay(TimeUnit.MILLISECONDS);*

*...*

*if (!replayLog(cmd)) {*

*replayCountersMap.get(REPLAYCOUNTERS.TOTALINVALIDCOMMANDS).increment(1);*

*}*

*cmd = commandQueue.take();*

*}*

*....*

*}*

核心方法是replayLog(cmd)

*private boolean replayLog(final AuditReplayCommand command) {*

*final String src = command.getSrc();*

*final String dst = command.getDest();*

*FileSystem proxyFs = fsCache.get(command.getSimpleUgi());*

*......*

*final FileSystem fs = proxyFs;*

*ReplayCommand replayCommand;*

*= ReplayCommand.valueOf(command.getCommand().split(" ")[0].toUpperCase());*

*switch (replayCommand) {*

*case CREATE:*

*FSDataOutputStream fsDos = fs.create(new Path(src));*

*if (createBlocks) {*

*fsDos.writeByte(0);*

*}*

*fsDos.close();*

*break;*

*case GETFILEINFO:*

*fs.getFileStatus(new Path(src));*

*break;*

*.....*

*}*

参考文献：

https://github.com/linkedin/dynamometer

https://engineering.linkedin.com/blog/2018/02/dynamometer--scale-testing-hdfs-on-minimal-hardware-with-maximum

https://siliconangle.com/blog/2018/02/08/linkedin-open-sources-dynamometer-new-tool-testing-hadoop-cluster-performance/