

**课程作业: BestConfig源码剖析**

**学 院：计算机科学与技术学院**

**专 业： 软件工程**

**班 级： 2019**

**学 号： 2019021964**

**学生姓名： 孙 武**

**指导教师： 申国伟**

2020 年 7月20日

**目录**

[1. BestConfig简介 1](#_Toc46773777)

[2. BestConfig术语 2](#_Toc46773778)

[2.1. SUT 2](#_Toc46773779)

[2.2. OPS 2](#_Toc46773780)

[2.3. Latency 2](#_Toc46773781)

[2.4. DDS 2](#_Toc46773782)

[2.5. RBS 2](#_Toc46773783)

[3. BestConfig架构 3](#_Toc46773784)

[3.1. 负载生成模块（Workload Generator） 3](#_Toc46773785)

[3.2. 系统操作模块（System Manipulator） 4](#_Toc46773786)

[3.3. 抽样模块（Configuration Sampler) 6](#_Toc46773787)

[3.4. 性能调优模块（Performance Optimizer） 10](#_Toc46773788)

[4. BestConfig生成负载 12](#_Toc46773789)

[5. BestConfig抽样算法DDS 14](#_Toc46773790)

[5.1. 确定参数范围、样本数量、抽样轮次 16](#_Toc46773791)

[5.2. DDS抽样 17](#_Toc46773792)

[5.3. 根据参数范围和样本量划分区间 22](#_Toc46773793)

[5.4. 根据DDS划分矩阵生成样本 23](#_Toc46773794)

[6. BestConfig调优算法RBS 24](#_Toc46773795)

[7. BestConfig与SUT交互 27](#_Toc46773796)

[8. 总结 28](#_Toc46773797)

1. BestConfig简介

BestConfig是一个在给定应用程序工作负载下，在系统资源限制内自动查找最佳配置的系统。说白了是个性能调优系统。BestConfig设计了一个可扩展的体系结构，以自动化一般系统的配置调优。

BestConfig 可以调优包括数据库、JVM、Apache以及Spark诸多类型的系统。通过配置调整，BestConfig就可以将Tomcat的吞吐量提高75%，Cassandra的吞吐量提高63%，MySQL的吞吐量提高430%，使Hive join作业的运行时间减少50%左右，Spark join作业的运行时间减少80%左右。

1. BestConfig术语
   1. SUT

SUT（ System Under Tune)是需要调优的目标系统。如需要使用BestConfig对Tomcat进行调优，那SUT就是指 Tomcat。

* 1. OPS

OPS 即吞吐量，是描述系统单位时间内处理业务的能力。例如，每秒能打2个字，那吞吐量就是 2 ops，即每秒2个字。在BestConfig中，其的关注的指标之一就是吞吐量。

* 1. Latency

latency即延迟，指从提交请求到收到响应结果的这段时间段。BestConfig的关注的另外一个指标就是延迟。

* 1. DDS

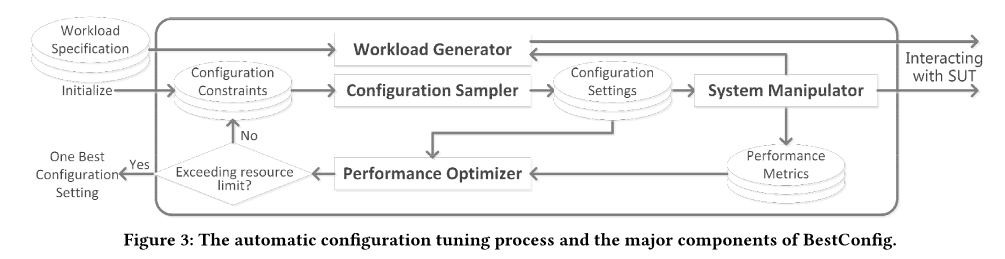
DDS（Divide and Diverge Sampling）是BestConfig 中使用的抽样算法，用于在参数空间中抽样，将在第五章BestConfig抽样算法中讨论。

* 1. RBS

RBS（Recursive Bound and Search）是BestConfig的优化算法。他在DDS划分的值区间的基础上，在区域内进行最优性能点的搜索。将在第六章中讨论。

1. BestConfig架构

BestConf主要包括负载生成模块（Workload Generator）、系统操作模块（System Manipulator）、抽样模块（Configuration Sampler)、性能调优模块（Performance Optimizer），各模块通过数据流连接起来，如下图所示。系统操作模块是与部署在目标环境中的SUT进行交互的接口，而工作负载生成器组件允许对任何目标系统生成工作负载，系统操作器和工作负载生成器是唯一与SUT交互的两个组件。



在样本有限的情况下，调优过程必须通过仔细选择来收集样本。部署的系统和工作负载的不同组合可能需要不同的采样选择。因此，BestConfig设计了带有采样器组件的架构，该组件在采样时与系统操作模块（System Manipulator）交互。此外，性能优化过程可以引入更多关于要示例的配置设置的信息,因此，BestConfig的性能优化器组件被设计为与采样器交互以传递信息。

性能优化模块（Performance Optimizer）将抽样方法DDS(除法和发散抽样)和优化算法RBS((Recursive Bound and Search)结合起来作为一个完整的解决方案。

为了处理各种部署系统和工作负载，BestConf被设计成一个灵活的体系结构, 包含一系列的组件，如图下图所示。这些组件通过数据流连接。该系统的组件是一个与部署在目标中的SUT交互的接口环境，而 workloadgenerator组件允许插入任何目标工作负载。

接下来仔细介绍每个模块。

* 1. 负载生成模块（Workload Generator）

负载生成模块用于生产应用负载。

BestConf 主要是用一些广泛 benchmark 工具。如使用HiBench用于Hive+Hadoop，Cassandra使用YCSB，MySQL 使用 SysBench，Tomcat使用JMeter。

在BestConf 中，负载模块直接与SUT交互，直接与之进行数据等交互。举个例子来说，如果调优系统时 Tomcat ，使用 Jmeter 作为负载生成工具，那么，其工作形式是：启动BestConf 之后，BestConf 启动 Jmeter ，Jmter 直接将 HTTP 请求发送到 Tomcat。

* 1. 系统操作模块（System Manipulator）

系统操作模块用于控制目标系统的行为，如:

1）配置文件修改。

2）系统启动、关闭。

3）测试启动、终止与性能获取。

这个模块是 BestConf 中与SUT进行交互的模块。

为什么需要这个模块？

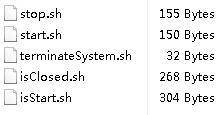
不同系统的操作方式是不一样，为了兼容各种各样的系统，BestConf 采用 shell 脚本的方式驱动系统。

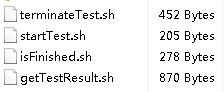
shell 脚本主要功能：

1）系统启动、关闭。

2）测试启动、终止与性能获取。

以 Spark 为例，包含如下脚本：

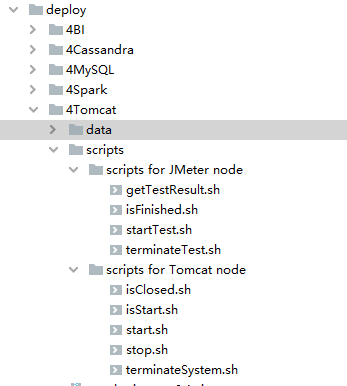




以 start.sh 为例，该脚本将会在 Master 节点上启动系统，代码如下图所示。其他脚本也是如此。篇幅原因，不再详述。



当然，也有 Tomcat 的启动脚本，如下图所示：



以上这些 shell 脚本的存放路径由配置文件 SUTconfig.properties 中的 shellsPath 键指定。最终将会被 SUTSystemOperation 类使用，如下图所示：

**

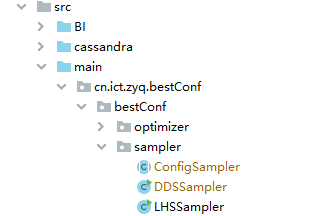
该模块的接口是cn.ict.zyq.bestConf.cluster.Interface.SystemOperation，实现是cn.ict.zyq.bestConf.cluster.InterfaceImpl.SUTSystemOperation，如下图所示：

**

* 1. 抽样模块（Configuration Sampler)

该模块用于在参数空间中进行抽样。

该模块有三个文件，分别是配置文件、DDS抽样实现文件、LHS抽样实现文件，其位于 cn.ict.zyq.bestConf.bestConf.sampler 下，如下图所示。

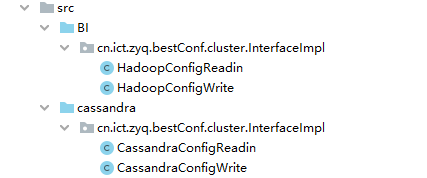


抽样的数据最先来源于配置文件，即文件defaultConfig.yaml 和 defaultConfig.yaml\_range。接下来简单介绍配置读取和写入模块。

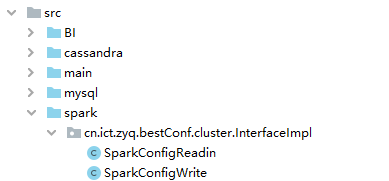
配置文件的读写模块：

不同的系统的配置文件是各不相同，为了兼容各种各样的系统，BestConf 采用了接口的方式。当用户要接入一个系统的时候，自己实现配置文件的读写功能即可。这样设置的目的是提高了兼容性。

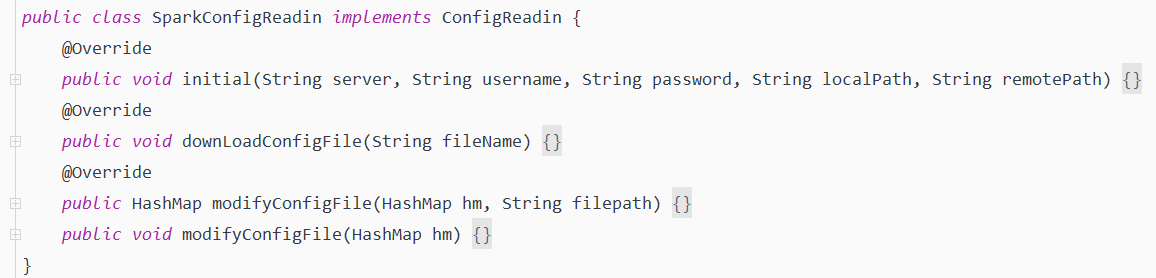
配置文件的读写接口位于 src 下，如下图所示。BI 目录下的主要是 Hadoop 的配置读写接口，Cassadra 目录下主要是数据库 Cassadra 的配置读写接口。当前，系统还可以包含任意其他系统的配置，不局限与我文中提及到的这两个系统。



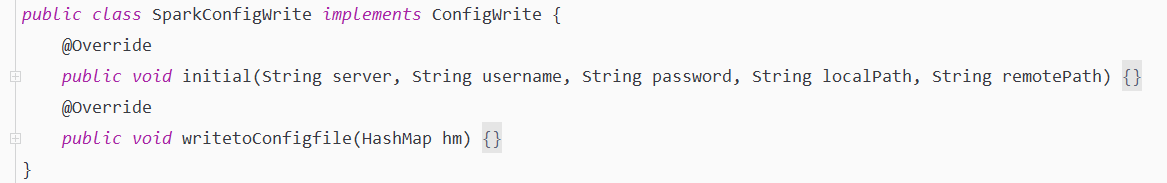
举个例子，如果对spark调优，需要手动实现ConfigReadin和ConfigWrite接口，接口位于包 cn.ict.zyq.bestConf.cluster.InterfaceImpl 下，如下图所示：



SparkConfigReadin.java



SparkConfigWrite.java



源码中，给出了 Tomcat 的读实现，有兴趣可直接阅读文件  src\tomcat\cn\ict\zyq\bestConf\cluster\InterfaceImpl\TomcatConfigReadin.java 。

实现配置的输入输出接口之后，下一步配置 Spark 中的需要调优的参数，在下一节中讨论。

配置程序的配置：

当实现配置文件的读写接口之后，接下来的工作便是要产生一个默认配置，供程序调优使用。

系统中主要包括如下几个配置文件

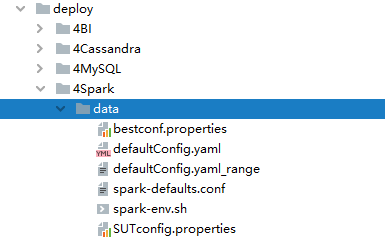
bestconf.properties：配置调优算法和样本集

defaultConfig.yaml：待调参数的默认取值

defaultConfig.yaml\_range：待调参数的取值范围

SUTconfig.properties：待调系统和测试相关配置

在源码中位于 deploy 目录下。



配置 Spark 的调优参数需要配置两个文件，一个文件是默认的 Spark 调优参数，一个是资源限制范围。

defaultConfig.yaml 是 Spark待调配置参数列表，如下所示。

Size.spark.memory.offHeap.size: 0

spark.shuffle.memoryFraction: 0.2

spark.storage.unrollFraction: 0.2

spark.memory.storageFraction: 0.5

spark.memory.fraction: 0.6

spark.storage.memoryFraction: 0.6

spark.speculation.quantile: 0.75

spark.task.cpus: 1

spark.rpc.message.maxSize: 128

spark.shuffle.sort.bypassMergeThreshold: 200

spark.shuffle.service.index.cache.entries: 1024

spark.scheduler.minRegisteredResourcesRatio: 0.8

spark.executor.cores: 1

Time.spark.speculation.interval: 100

Time.spark.executor.heartbeatInterval: 10000

Time.spark.network.timeout: 120000

Time.spark.rpc.lookupTimeout: 120000

spark.files.maxPartitionBytes: 134217728

Size.spark.driver.memory: 1048576

Size.spark.executor.memory: 1048576

Time.spark.scheduler.revive.interval: 1000

Time.spark.dynamicAllocation.schedulerBacklogTimeout: 1000

Size.spark.storage.memoryMapThreshold: 2048

Size.spark.shuffle.file.buffer: 32

Time.spark.rpc.retry.wait: 3000

Time.spark.locality.wait: 3000

Size.spark.reducer.maxSizeInFlight: 49152

Size.spark.broadcast.blockSize: 4096

Time.spark.shuffle.io.retryWait: 5000

Time.spark.files.fetchTimeout: 60000

Time.spark.dynamicAllocation.executorIdleTimeout: 60000

defaultConfig.yaml\_range 用于指定 Spark 待调配置参数范围，如下配置限制了 Size.spark.memory.offHeap.size 的调优范围为 0 至 16777216，在调优的时候，参数只能在这个范围变化。

Size.spark.memory.offHeap.size: "[0,16777216]"

bestconf.properties

configPath=data/SUTconfig.properties

yamlPath=data/defaultConfig.yaml

InitialSampleSetSize=500 # 设置初始样本集大小

RRSMaxRounds=1 # 设置算法的最大轮数

COMT2Iteration=10

COMT2MultiIteration=20

SUTconfig.properties

systenName=Sparx # 设置待调系统名字，

configF1leStyle"yaml # yaml是配置文件的类型

# 设置远程服务器IP，用户名及密码

numServers=1

username=root

password=root

# 调优程序的配置文件目录

localDataPath=data

# shell脚本在待调系统上的目录

shellsPath=/opt/yark

# 待调系统的配置文件目录

remoteConf1gF11ePath=/opt/H1Bench-master/cont/vork1oads/ml

# 接口文件所在程序目录

interfacePath=cn.ict.zyq.bestConf.cluster.InterfaceImpl

# 待调系统启动超时设置

sutStartTimeoutInsec=300

# 测试超时设置

testDurationTimeoutInsec=100

# 连接远程系统的最大失败次数

maxRoundConnection=60

# 测试失败的最大次数

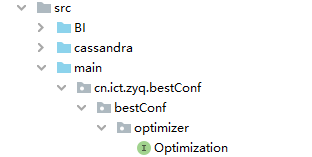
targetTestErrorNum=10

这些配置文件最终会在 BestConf 类中被用到，如下图所示：

* 1. 性能调优模块（Performance Optimizer）

该模块主要是对性能的调优。

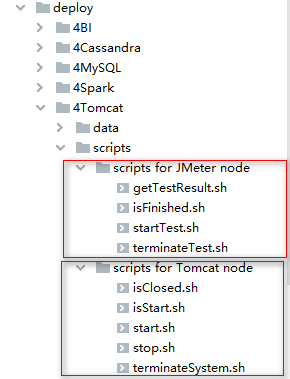
其接口位于文件 cn.ict.zyq.bestConf.bestConf.optimizer.Optimization中，算法由类 cn.ict.zyq.bestConf.bestConf.RBSoDDSOptimization 实现。



1. BestConfig生成负载

BestConfig主要是使用一些benchmark工具。如使用HiBench用于Hive+Hadoop，Cassandra使用YCSB，MySQL 使用 SysBench，Tomcat使用JMeter。

在源码中，带有Tomcat的benchmark例子，如下图所示。



BestConfig生成Tomcat的负载主要是通过Jmeter，生成负载的脚本startTest.sh如下：

#!/bin/bash

# 设置Jmeter的环境

JMETER\_HOME=/root/apache-jmeter-3.1

path=$JMETER\_HOME/testresults.txt

# 切换工作目录

cd $JMETER\_HOME/bin

#运行负载

nohup ./jmeter -n -t test4bestconf.jmx > $path &

获取测试结果的脚本getTestResult.sh如下：

#!/bin/bash

JMETER\_HOME=/root/apache-jmeter-3.1

# 生成负载脚本startTest.sh产生的结果文件

resultFile=$JMETER\_HOME/testresults.txt

# 获取吞吐量

minval=0

if [ -f "$resultFile" ]; then

tail -5 $resultFile > ./tempresult

lastline=""

while read line

do

result=$(echo $line | grep "Tidying up")

if [ "$result" != "" ];then

#we can get the result

result=$(echo $lastline | grep "summary = ")

if [ "$result" != "" ];then

#we actually have the result

throughput=`echo $lastline|awk -F ' ' '{print $7}'|tr -d ' '|tr -d '/s'`

break

fi

fi

lastline=$line

done < ./tempresult

if [ `expr $throughput \> $minval` -eq 1 ]; then

echo $throughput

else

echo "error"

fi

else

echo "not exist"

fi

isFinished.sh用于判断负载是否执行完成，terminateTest.sh用于终止负载。

由此可见，BestConfig主要是依赖第三方benchmark工作中，以shell脚本的方式驱动。

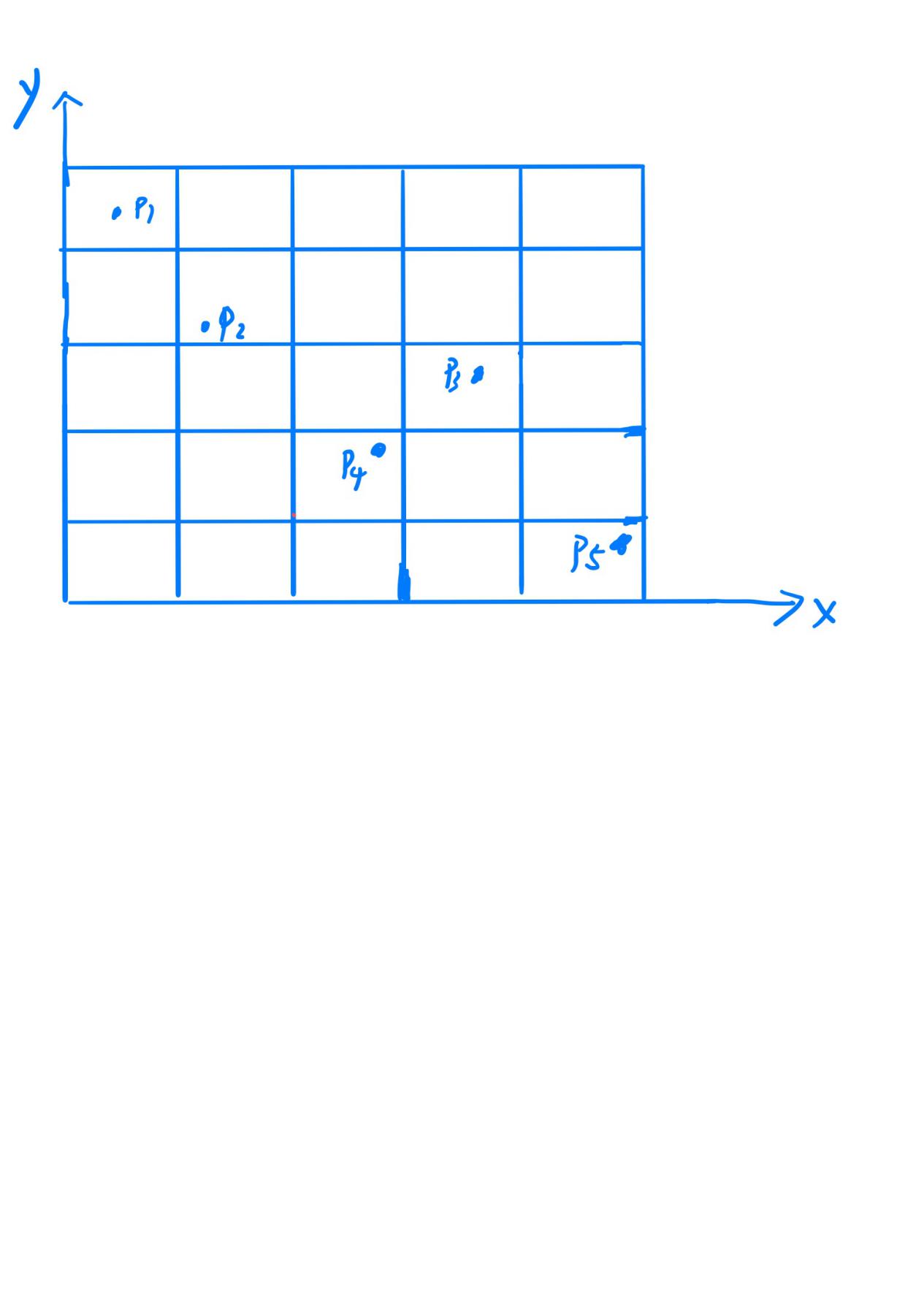
1. BestConfig抽样算法DDS

BestConfig抽样算法即DDS算法。DDS（Divide and Diverge Sampling）是BestConfig 中使用的抽样算法，用于在参数空间中抽样。

举个简单例子，假设需要对 mysql 的参数 thread\_cache\_size 调优，thread\_cache\_size 取值是0到无穷大。为了获得mysql的OPS和latency，需要将 thread\_cache\_size 设置为具体的值，那设置多少好呢？

对于一维来说，DDS将thread\_cache\_size 划分为k个区间，每一个区间取一个值做代表，及得到了k个值。比如，假设thread\_cache\_size 的取值范围为1到100，那么将1到100划分为十个区间，即[0,10），[10,20), [20,30), [30,40), [40,50),[50,60),[60,70),[70,80),[80,90),[90,100), 然后再每个区间中分别取一个值代表整个区间，如取值2可代表[0,10)这个区间，39可代表[30,40)这个区间，区间中取值是任意的。因此，在一维内，DDS和分层抽样的原理类似。

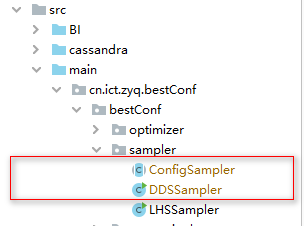
对于二维空间来说，同样需要进行区间划分，但它和分层抽样不同。假设有两个参数x、y需要进行调优，首先将x、y等分为6份，然后，取6个点Pi(i=1-6),使x和y的每个区间内都被抽样了一次，保证其样本点覆盖范围。



DDS在二位空间上的抽样方法并不是分层抽样或网格抽样，它的抽样方法减少了样本的数量，同时保持其覆盖范围的完备性。

如果要调优的参数多余两个，即在高位空间中抽样，原理课类别二维空间。

实现DDS的类如下两图中两个文件ConfigSampler、DDSSampler。

为了便于理解算法，先看一下算法的输入和输出，如下所示，

输入：(为了美观，将不用的代码去掉，用 ..... 表示）

//设置参数范围

p1.setProperty("range", "[0,1]");

......

p2.setProperty("range", "[321,1E9]");

......

p3.setProperty("range", "[1,30]");

......

//抽样轮数  
DDSSampler sampler = new DDSSampler(3);

//第一轮抽样

sampler.setCurrentRound(0);

Instances data = sampler.sampleMultiDimContinuous(atts, 2, false);

System.out.println(data);

//第二轮抽样

sampler.setCurrentRound(01);

data = sampler.sampleMultiDimContinuous(atts, 2, false);

System.out.println(data);

//第三轮抽样

sampler.setCurrentRound(2);

data = sampler.sampleMultiDimContinuous(atts, 2, false);

System.out.println(data);

算法输出：

......

@data # 第一轮数据，分别是p1,p2,p3 各自范围的值

0.338052,596639607,17

0.622994,666142,1

......

@data # 第二轮数据，分别是p1,p2,p3 各自范围的值

0.170678,685465692,21

0.805374,202337397,10

......

@data # 第三轮数据，分别是p1,p2,p3 的返回值

0.484047,800497705,18

0.749216,416724809,5

在输出中，共三组数据。第轮数据中，p1,p2,p3分别是{0.338052, 596639607, 17}和{0.622994, 666142, 1}，即两个样本，上面抽了三轮，因此有六个样本。

下面详细说明抽样流程。

* 1. 确定参数范围、样本数量、抽样轮次

参数范围只参数调优的范围，如cpu核的数量指定为1-8。样本的数量指定要多少个样本，对于cpu核数来说，如1个样本可以是3，也可以是8，指参数范围中具体的某一个值。在参数范围中取多少值，就是多少个样本。抽样轮次指定你要抽几轮。如下代码指定参数范围、样本数量、抽样轮次。(文件名与类名形同，因此类名可以确定文件名，因此不标注文件名了）

public class DDSSampler extends ConfigSampler{  
  
......

public static void main(String[] args){

/\*预处理调优参数\*/

//模拟调优的第一个参数

ArrayList<Attribute> atts = new ArrayList<Attribute>();

Properties p1 = new Properties();

p1.setProperty("range", "[0,1]");

ProtectedProperties prop1 = new ProtectedProperties(p1);

//模拟调优的第二个参数

Properties p2 = new Properties();

p2.setProperty("range", "[321,1E9]");

ProtectedProperties prop2 = new ProtectedProperties(p2);

//模拟调优的第三个参数

Properties p3 = new Properties();

p3.setProperty("range", "[1,30]");

ProtectedProperties prop3 = new ProtectedProperties(p3);

ArrayList<String> attVals = new ArrayList<String>();

for (int i = 0; i < 5; i++)

attVals.add("val" + (i+1));

atts.add(new Attribute("att1", prop1));

atts.add(new Attribute("att2", prop2));

atts.add(new Attribute("att3", prop3));

/\*预处理调优参数\*/

//实例化

DDSSampler sampler = new DDSSampler(3);

//第一列轮数

sampler.setCurrentRound(0);

Instances data = sampler.sampleMultiDimContinuous(atts, 2, false);

System.out.println(data);

sampler.setCurrentRound(01);

data = sampler.sampleMultiDimContinuous(atts, 2, false);

System.out.println(data);

sampler.setCurrentRound(2);

data = sampler.sampleMultiDimContinuous(atts, 2, false);

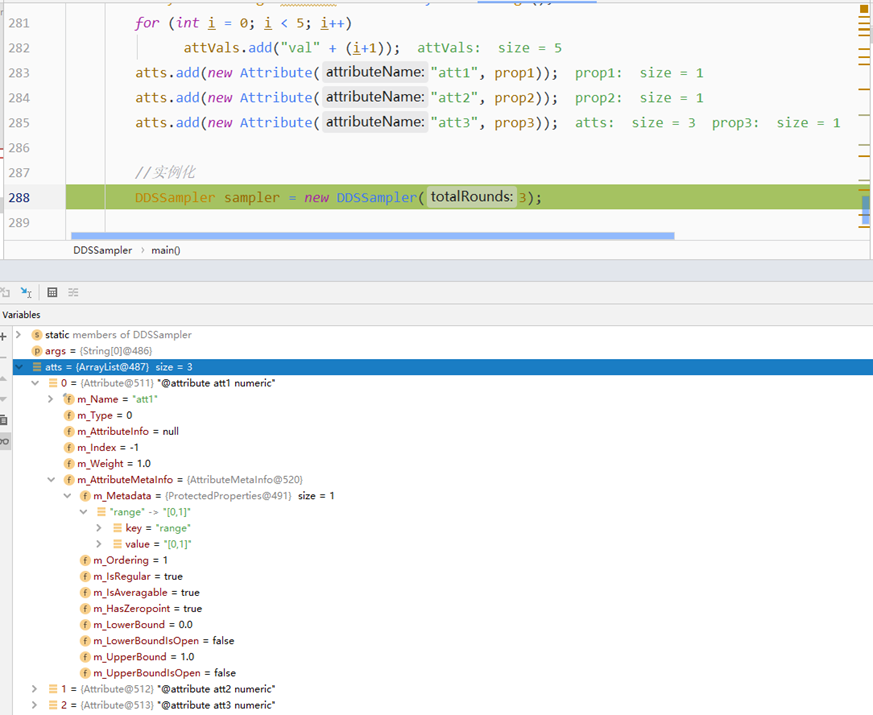
System.out.println(data);

}

}

在以上代码中，p1的参数范围为[0,1]，p2的范围为[321,1E9]，p3的范围为[1,30]，它模拟实际调优中有三个参数的情况。通过预处理之后的参数的形式如下图所指示：

打断点调试程序，可查看其参数值被封装为一个对象，如下所示：



* 1. DDS抽样

为了便于说明，先看一下本部分的输入输出是什么。

生成DDS抽样矩阵需要两个参数，分别是样本数量、参数个数，与具体的参数值无关。若有两个参数、三个样本，那么生成的结果是：

[0,1,2]  
[1,0,2]

若有两个参数、四个样本，那么生成的结果是：

[0,1,2,4]  
[1,0,2,3]

由此可得，第一行的值为{1,2,...,样本数量}，列二行的值为第一行的一个排列。如果有第三行，那么他也是第一行的第一个排列，且不能与其他行相同。

以下为DDSSampler.java 中 sampleMultiDimContinuous函数的部分代码及注释：

public class DDSSampler extends ConfigSampler {

......

/\*\*

\* 抽样算法的实现

\*/

public Instances sampleMultiDimContinuous(ArrayList<Attribute> atts, int sampleSetSize, boolean useMid) {

// 随机生长的所有样本中，算出的距离数组总值最大的那个数组

ArrayList<Integer>[] crntSetPerm;

//只初始化一次

if (sets == null) {

// 样本数量：可能的样本集数量不会超过$sampleSetSize的2次幂，以后要拿这个来进行参数切分。

// 轮数同样决定样本的数量

int L = (int) Math.min(rounds,

atts.size() > 2 ? Math.pow(sampleSetSize, atts.size() - 1) :

(atts.size() > 1 ? sampleSetSize : 1));

//距离 distances 向量，两点之间连线的距离，保存的是 generateOneSampleSet 产生的抽样矩阵中最小的值

dists = new long[L];

//样本集 sets 向量，l是其样本的数量

sets = new ArrayList[L][];

//初始化距离为 -1，样本集为null

for (int i = 0; i < L; i++) {

dists[i] = -1; // set -1 for all item in dists

sets[i] = null; //tow dimension array

}

//算出来的 欧几里得距离中最大的值 和 索引

long maxMinDist = -1;

int posWithMaxMinDist = -1;

for (int i = 0; i < L; i++) {

//attrs 属性存放的数组，参数的个数

ArrayList<Integer>[] setPerm = generateOneSampleSet(sampleSetSize, atts.size());

//两个参数，三个样本结果

//[0,1,2]

//[1,0,2]

//两个参数，四个样本结果

//[0,1,2,4]

//[1,0,2,3]

//由此可见，其生成的结果是以参数个数为行的数量，样本数量为列的数量。

//第一行的值为1,2,...,样本数量

//列二行的值为xi={random x,1<x样本数量}，且值唯一，二行的值为第一行的一个排列。

//继续样本集的生成，直到得到不同的样本，保证生成的是排列

while (inAlready(sets, setPerm))

setPerm = generateOneSampleSet(sampleSetSize, atts.size());

sets[i] = setPerm;

//计算setPerm集合中任意样本对之间的最小距离。欧几里得距离，即两点之间的长度

//sets [i] 的最小距离为 dists[i]

dists[i] = minDistForSet(setPerm);

//选择 dists 中距离值最大的

if (dists[i] > maxMinDist) {

posWithMaxMinDist = i;

maxMinDist = dists[i];

}

}

//将第一个和最大的一个交交换。就是将上文中参数的 sets 的第一个换位 distance 最大的那个

positionSwitch(sets, dists, 0, posWithMaxMinDist);

}

//计算的distance 最大的那个数组

crntSetPerm = sets[sampleSetToGet];

.....

}

其中，generateOneSampleSet 函数是DDS抽样算法的实现，指每行、每列仅包含一个样本的方阵，其返回的是一个矩阵（二位数组）。当两个参数、四个样本的情况下，其产生的矩阵如下：

[0,1,2,3]

[2,1,0,3]

[0,1,2,3]

[0,3,1,2]

[0,1,2,3]

[1,2,0,3]

三个矩阵的不同之处在于，第二行是第一行的一个排列，但是三个矩阵的第二行彼此不相同。

如下为generateOneSampleSet 算法的核心代码：

public class DDSSampler extends ConfigSampler {

......

/\*\*

\* 根据LHS抽样方法的要求生成一个样本集，参考：

\* https://www.sciencedirect.com/topics/engineering/latin-hypercube-sampling

\* 分层次抽样方法

\* 在统计抽样中，LHS是指每行、每列仅包含一个样本的方阵

\*

\* @return 生成的示例集指定在每个示例的每个属性下选择哪个子域，每个arraylist是每个属性的子域的排列

\*/

private static ArrayList<Integer>[] generateOneSampleSet(int sampleSetSize, int attrNum) {

ReadingDebugTools.getFunName();

//attrNum 参数的个数，比如线程数量和内存大小

ArrayList<Integer>[] setPerm = new ArrayList[attrNum];//sampleSetSize samples; each with atts.size() attributes

int crntRand;

// 生成参数个数对sampleSetSize整数的排列

for (int i = 1; i < attrNum; i++) {

setPerm[i] = new ArrayList<Integer>(sampleSetSize);

//为sampleSetSize整数随机生成一个排列

for (int j = 0; j < sampleSetSize; j++) {

crntRand = uniRand.nextInt(sampleSetSize);

// 保证唯一

while (setPerm[i].contains(crntRand)) {

crntRand = uniRand.nextInt(sampleSetSize);

}

setPerm[i].add(crntRand);

}

}

//将第一列置为{1,2,3,...,sampleSetSize}

setPerm[0] = new ArrayList<Integer>(sampleSetSize);

for (int j = 0; j < sampleSetSize; j++) {

setPerm[0].add(j);

}

return setPerm;

}  
 ......

}

函数 minDistForSet 用于计算 generateOneSampleSet 函数返回的矩阵的最小距离，怎么算呢？

设generateOneSampleSet 生成的矩阵一个矩阵设为res，取res的第一个元素的第一列和第二个元素的第一列，组成一个二元组，命名为z1; 取res的第一个元素的第二列和第二个元素的第二列，组成一个二元组，命名为z2;以此类推，可以得到四个二元组分别是z1、z2、z3、z4。接着，从四个二元组中任取两个组合，并计算其欧氏距离（两点之间的直线距离），根据组合的性质，可以得到6个距离。六个距离中最小的一个，就是minDistForSet 函数计算的距离。

举个例子，假设generateOneSampleSet 生成的矩阵是：

[0,1,2,3]

[0,3,1,2]

首先，取第一行的第一个数和第二行的第一个数，组成z1=（0，0）；取第一行的第二个数和第二行的第二个数，组成z2=（1，3）；取第一行的第三个数和第二行的第三个数，组成z3=（2，1）；类似的，z4=(3,2)。

其次，计算z1、z2、z3、z4任意两个点之间的欧氏距离为Li. 如z1和z2之间的欧式距离为L1（z1，z2）=(0-1)2+(0-3)2=1+9=10; 如z1和z3之间的欧式距离为L2（z1，z3）=(0-2)2+(0-1)2=4+1=5; 接着算出L3（z1，z4)、L4（z2，z3）、L5(z2、z4）、L6(z3、z4）。

最后，取min( L1、L2、L3、L4、L5、L6）（六个值总最小的值），即为minDistForSet所计算的结果。

generateOneSampleSet 会产生多个结果，但最终只需要一个。代码中选择minDistForSet计算结果中最大的一个。举个例子，假设generateOneSampleSet 生成三个矩阵m1、m2、m3，minDistForSet的计算结果为：d1=1、d2=3、d3=2，那么，就选择结果为m2的这个矩阵,因为 d2>d3>d1，目的是让各样本点尽量分散。核心代码如下：

public class DDSSampler extends ConfigSampler {

public Instances sampleMultiDimContinuous(ArrayList<Attribute> atts, int sampleSetSize, boolean useMid) {

......

{

dists[i] = minDistForSet(setPerm);

//选择 dists 中距离值最大的

if (dists[i] > maxMinDist) {

posWithMaxMinDist = i;

maxMinDist = dists[i];

}

}

//将第一个和最大的一个交交换，就是将上文中参数的 sets 的第一个换位 distance 最大的那个

positionSwitch(sets, dists, 0, posWithMaxMinDist);

}

//获取的distance 最大的那个数组，sampleSetToGet的值是0， 函数 positionSwitch() 将距离最大的哪个样本矩阵设置为0号元素

crntSetPerm = sets[sampleSetToGet];

......

}

那为什么取最大的一个呢，为了取离散程度最大的样本。

这里用一个例子说明上述过程：

generateOneSampleSet函数会生成一系列的样本（样本个数=样本数\*属性个数），记为 Si ；接着，使用minDistForSet函数计算每个样本的距离，记为 Di ；最后，取Si 中Di值最大的一个样本作为最终样本。

* 1. 根据参数范围和样本量划分区间

比如，取三个样本，一个参数，参数范围为[1,6]，那么该参数会被被三次（划分次数与样本数量相同）划分，产生4个值,如 [1,6] 被划分为[1.0, 2.66, 4.33, 6.0]。如果是多个参数，那么就会被划分为多次。

参数划分核心代码如下所示：

public class DDSSampler extends ConfigSampler {

......

public Instances sampleMultiDimContinuous(ArrayList<Attribute> atts, int sampleSetSize, boolean useMid) {

......

//划分参数范围

double[][] bounds = new double[atts.size()][sampleSetSize + 1];

Iterator<Attribute> itr = atts.iterator();

Attribute crntAttr;

boolean[] roundToInt = new boolean[atts.size()];

// 多个参数，划分多次

for (int i = 0; i < bounds.length; i++) {

/\*

crntAttr 是用户传递过来的那个参数范围，如[1，6]

\* \*/

crntAttr = itr.next();

// [1,6] ==> [1.0，2.25，3.5，4.75，6.0]

uniBoundsGeneration(bounds[i], crntAttr, sampleSetSize);

if (bounds[i][sampleSetSize] - bounds[i][0] > sampleSetSize)

roundToInt[i] = true;

}

......

}

uniBoundsGeneration 就是实际划分的函数。

* 1. 根据DDS划分矩阵生成样本

当抽样矩阵、参数区间确定之后、即可生成样本了，核心代码如下所示：

public class DDSSampler extends ConfigSampler {

......

public Instances sampleMultiDimContinuous(ArrayList<Attribute> atts, int sampleSetSize, boolean useMid) {

......

Instances data = new Instances("SamplesByLHS", atts, sampleSetSize);

for (int i = 0; i < sampleSetSize; i++) {

double[] vals = new double[atts.size()];

//每次确定绑定的值

for (int j = 0; j < vals.length; j++) {

// 在已划分的区间中，选定一个区间的分界点的之后，在这个点与下一个点的这个间隔内随机取一个点，

// 比如[1,2,3,4] 是一个已划分的区间，

// 根据上面的 crntSetPerm 确定选中的一个点为 2，那么他就在 [2,3]之间随机取一个点。即 2+(3-2)\*uniRand.nextDouble()

//

vals[j] = useMid ?

(bounds[j][crntSetPerm[j].get(i)] + bounds[j][crntSetPerm[j].get(i) + 1]) / 2 :

bounds[j][crntSetPerm[j].get(i)] +

(

(bounds[j][crntSetPerm[j].get(i) + 1] - bounds[j][crntSetPerm[j].get(i)]) \* uniRand.nextDouble()

);

// useMid=true 取区间的中间值

// bounds[j][crntSetPerm[j].get(i)]+bounds[j][crntSetPerm[j].get(i)+1])/2 取中间的一个点

//

// useMid=false 区间的随机值

// bounds=[1.0 2.666666666666667 4.333333333333334 6.0] 第一个参数的划分范围

// [1.0 2.666666666666667 4.333333333333334 6.0] 第二个参数的划分范围

//

// vals[j]= bounds[j][crntSetPerm[j].get(i)]+ ((bounds[j][crntSetPerm[j].get(i)+1]-bounds[j][crntSetPerm[j].get(i)])\*uniRand.nextDouble())

// bounds[j][crntSetPerm[j].get(i)]：根据之前生成的抽样模板数组，获取值：

// (bounds[j][crntSetPerm[j].get(i)+1]-bounds[j][crntSetPerm[j].get(i)]) 当前值到下一个值之间的间隔，乘以一个随机数，就是在这个区间随机取点

// 上述代码的实际作用就是在每个区间直接随机取一个值

if (roundToInt[j])

vals[j] = (int) vals[j];

}

//给这个范围加个权重，没别的意思，只是这里的权重都为1而已

data.add(new DenseInstance(1.0, vals));

}

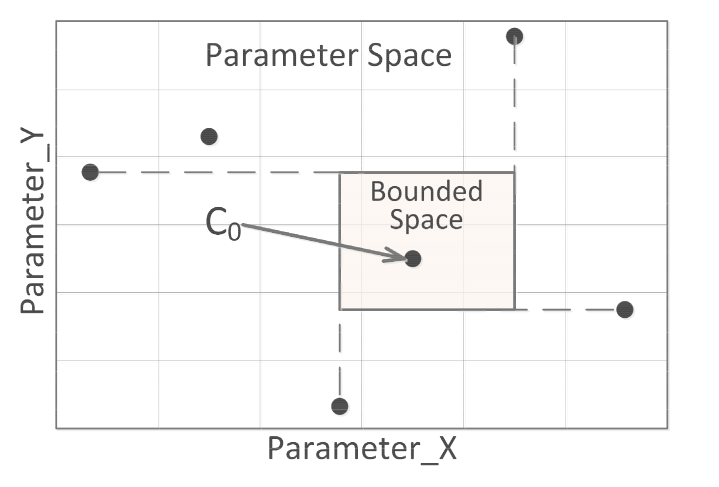
......

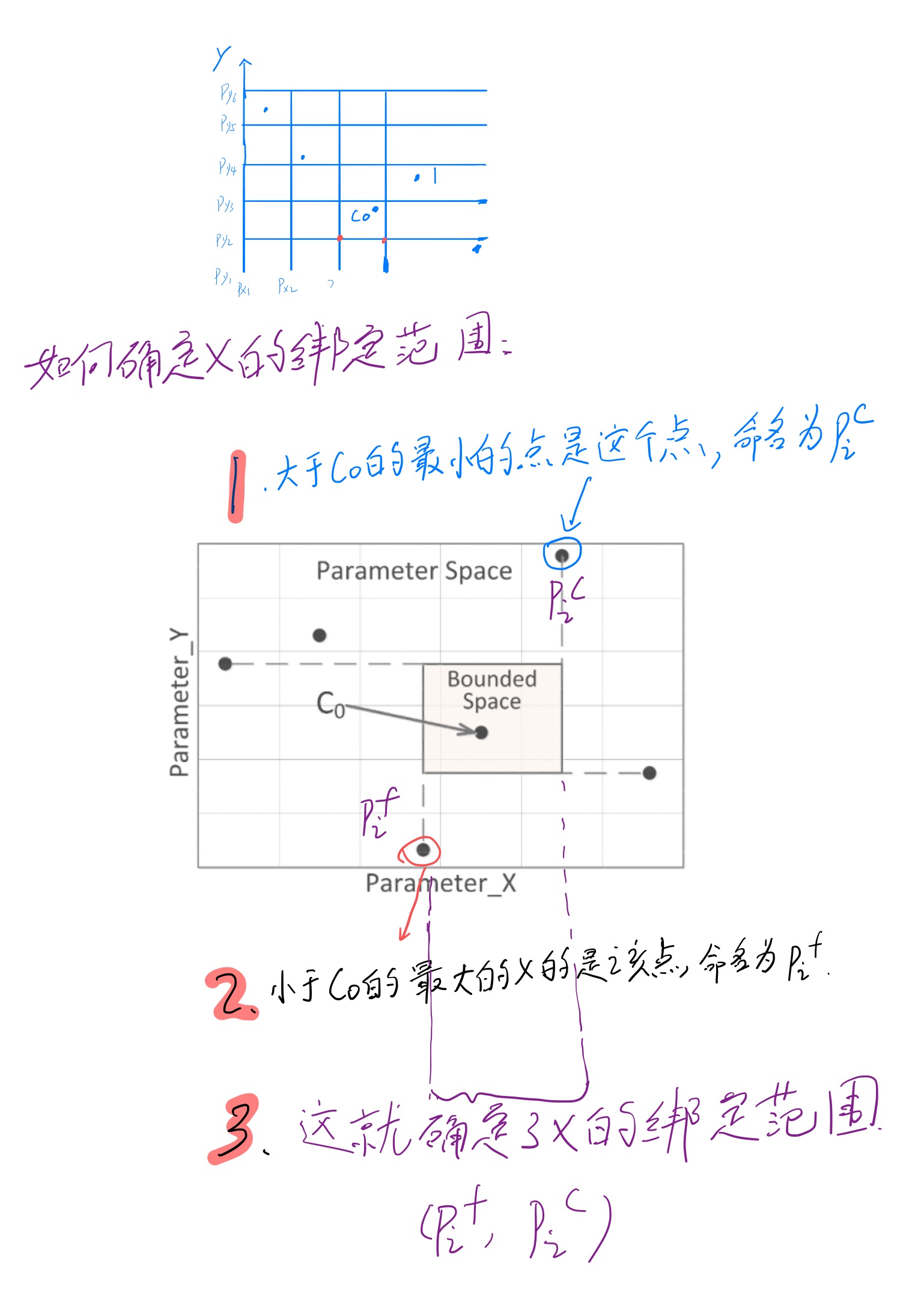
}

1. BestConfig调优算法RBS

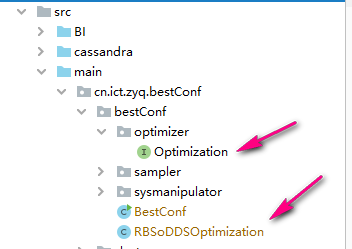
RBS（Recursive Bound and Search）是BestConfig的优化算法。它不是一个机器学习算法。在DDS划分的值区间的基础上，在区域内进行最优性能点的搜索。其思想是：给定一个连续的性能表面，我们有很高的可能性发现周围其他点在样本集中具有相似或更好的性能。换句话说，即给定一个初始样本集，RBS找到点C0有最好的表现。然后，它要求在C0周围的有界空间中采样另一组点，有很大的可能性会找到另一点(例如C1 )有更好的表现。 在C1周围的有界空间中再次采样，可以得到C2。可以递归地执行这个绑定步骤，直到在示例集中找不到性能更好的点。

那怎么确定边界呢？

对于每个参数pi，RBS 找到样本集中最大值pif 且小于C0，同样，也在样本集中找到最小值pic且大于C0，到此即可确定pi的绑定空间为（pif,pic)。下图解释了其绑定机制。

当然，笔者一开始看的时候，不知道他在说啥，又大于又小于的，也不知道啥意思。下图的笔记，笔者给出了如何确定x绑定范围的更详细的解释，y绑定也是同样的原理。x和y都绑定之后，也就确定了整个Bounded Space。

其接口及实验类如下图所示，Optimization为优化接口，RBSoDDSOptimization 是实现Optimization的文件。代码相对于DDS来说非常简单，读者可自行参考RBSoDDSOptimization.java。



1. BestConfig与SUT交互

BestConfig与SUT交互依赖系统操作模块。BestConf 采用 shell 脚本的方式驱动系统。下面的SUT以Tomcat为例。

若想获取SUT的性能，需要让目标系统发送请求，并获得性能数据。在Tomcat的测试环境下，BestConfig依赖脚本驱动Jmeter，包括负载的启动、终止、负载状态、负载的性能结果等，该部分已在第4节中讨论，请参考本文第4节。

BestConfig除了操控负载生成工具之外，还需要与SUT交互，如启动SUT、停止SUT等。在Tomcat环境下，包含以下几个操作SUT的脚本，如下：

isClosed.sh SUT是否关闭

isStart.sh SUT是否启动

start.sh 启动SUT

stop.sh 停止SUT，

terminateSystem.sh 终止SUT，通过 linux 的 kill 命令实现

其中，start.sh的内容如下，即启动tomcat，其他脚本内容与之类似，请读者执行阅读。

#!/bin/bash

export CATALINA\_HOME=/root/apache-tomcat-8.5.9

path=$CATALINA\_HOME/startresults.txt

rm -f $path

cd $CATALINA\_HOME

bin/startup.sh > $path

1. 总结

BestConfig可以对任何系统进行调优，前提是要自己实现他的负载控制脚本和SUT控制脚本，也就是以下几个脚本，前四个用于BestConfig控制Benchmark工具，后五个用于控制SUT。

getTestResult.sh 获取测试结果，主要是吞吐量

isFinished.sh 测试是否完成

startTest.sh 开始测试

terminateTest.sh 终止测试

isClosed.sh SUT是否关闭

isStart.sh SUT是否启动

start.sh 启动SUT

stop.sh 停止SUT，

terminateSystem.sh 终止SUT，通过 linux 的 kill 命令实现

其提出的DDS搜索算法既保证了高维空间参数的少，又保证了其参数范围覆盖的广泛性，加上RBS调优算法，可以保证在性能样本稀少的情况下，也能找到性能最高的点，达到系统调优的目的。

BestConfig度量性能的指标只有两个：一个是Latency（处理时间）、另外一个是OPS（吞吐量），源码中证明如下：

public class AutoTestAdjust implements ClusterManager {

public double getPerformanceByType(int type) {

double performance = 0.0;

switch (type) {

case 1: { //indicates latency

performance = systemPerformance.getPerformanceOfLatency();

break;

}

case 2: { //indicates throughput

performance = systemPerformance.getPerformanceOfThroughput();

System.out.println("throughput is " + performance);

break;

}

default:

break;

}

return performance;

}  
}

BestConfig操作SUT都是通过shell脚本，这个比较有意思，暴力且简单。