



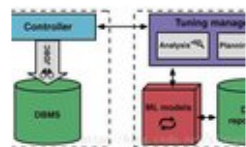
第九届中国系统架构师大会
SYSTEM ARCHITECT CONFERENCE CHINA 2017

技术前沿进展：系统自动化调优

朱好晴



OtterTune来了,DBA怎么办 - CSDN博客



2017年6月6日 - 将OtterTune 所生成的最佳配置与 Tuning 以及 RDS 相关配置进行比较可以发现,MySQL 在延迟水平方面降低了约 60%,而数据吞吐量则在 OtterTune 配置的帮助下提升 22...

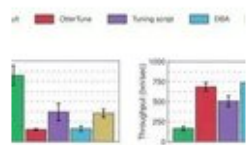
blog.csdn.net/xiangzhi... - 百度快照

厉害|DBA泪奔了!亚马逊用机器学习自动调优数据库管理系统!! - lx...

2017年6月5日 - 导读:最近亚马逊和卡内基梅隆大学一起开发了一套名叫“OtterTune”的机器学习自动化调整DBMS的系统,并公布起设计论文和开源项目,重点解决DBMS长期存在...

www.aixchina.net/Artic... - 百度快照

DBA要失业了?看ML如何自动优化数据库_搜狐科技_搜狐网



2017年6月4日 - 我们在MySQL和Postgre上评估OtterTune的调优能力,通过将OtterTune最佳配置的性能与DBA选择的配置及其他自动调优工具的配置得出的性能做比较。OtterTune是一...

www.sohu.com/a/1460241... - 百度快照

【AI研究室】第1期 | OtterTune来了,DBA真的要失业了么-云栖社区



本期导读 又一个产品借AI上头条了,就是OtterTune!脆弱的人类DBA又要面对威胁了;GoogleAssistant可以在美区AppStore下载了,微软小娜也开启了iOS

<https://m.aliyun.com/yunqi/art...> - 百度快照

运维要失业了?机器学习可自动优化你的数据库管理系统(DBMS)... 搜狐

2017年6月4日 - OtterTune是由卡内基·梅隆大学数据库小组

(<http://db.cs.cmu.edu/projects/autotune/>)的学生和研究人员开发的一种新工具,它能自动为DBMS的配置按钮找...

www.sohu.com/a/1460160... - 百度快照

创新 才能不被淘汰 机器学习时代 运维将何去何从? 搜狐科技 搜狐网

DBA要失业



提纲

- 1 系统自动化调优介绍
- 2 系统调优的难点与挑战
- 3 自动化调优的价值
- 4 应用案例
- 5 前沿技术进展
- 6 如何使用自动化调优

系统自动化调优

- 为了适应不同应用需求，系统在开发时就暴露了大量与部署、应用场景相关的参数
- 这些参数与系统性能紧密相关
 - 需要对系统和应用有资深经验的技术人员来调优
- 系统自动化调优
 - 将这一过程自动化



Application Properties

Property Name	Default
spark.app.name	(none)
spark.driver.cores	1

Runtime Environment

Property Name

spark.driver.extraClassPath

Shuffle Behavior

Property Name

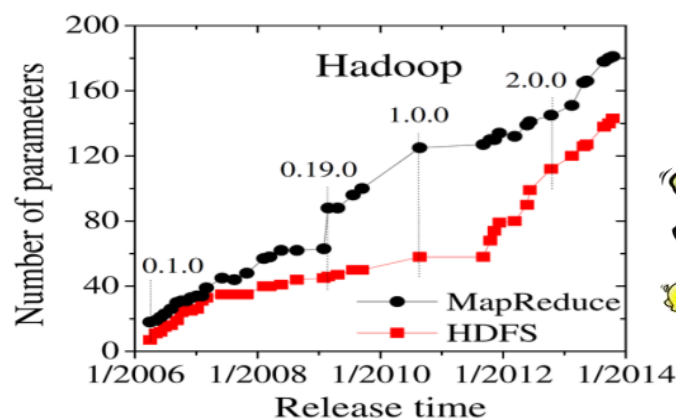
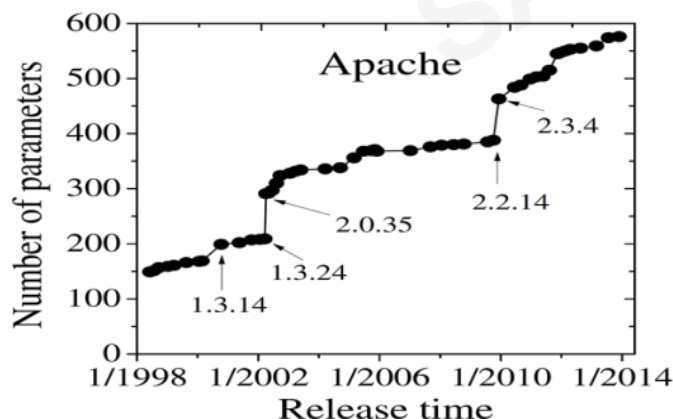
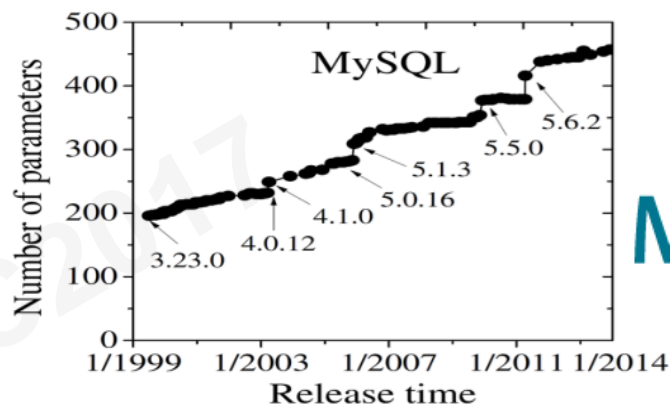
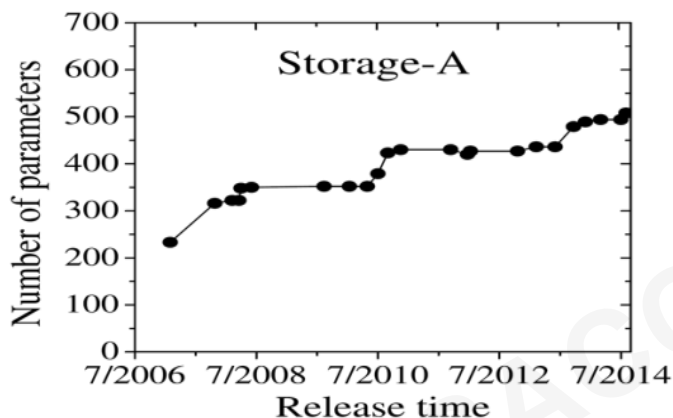
spark.reducer.maxSizeInFlight

系统调优的挑战与难点

- 系统参数个数越来越多
 - 适应更多的部署环境和应用场景
- 涉及的系统越来越多
 - 满足不同的应用负载需求
- 参数设置与系统、应用紧密相关
 - 性能曲线复杂多变

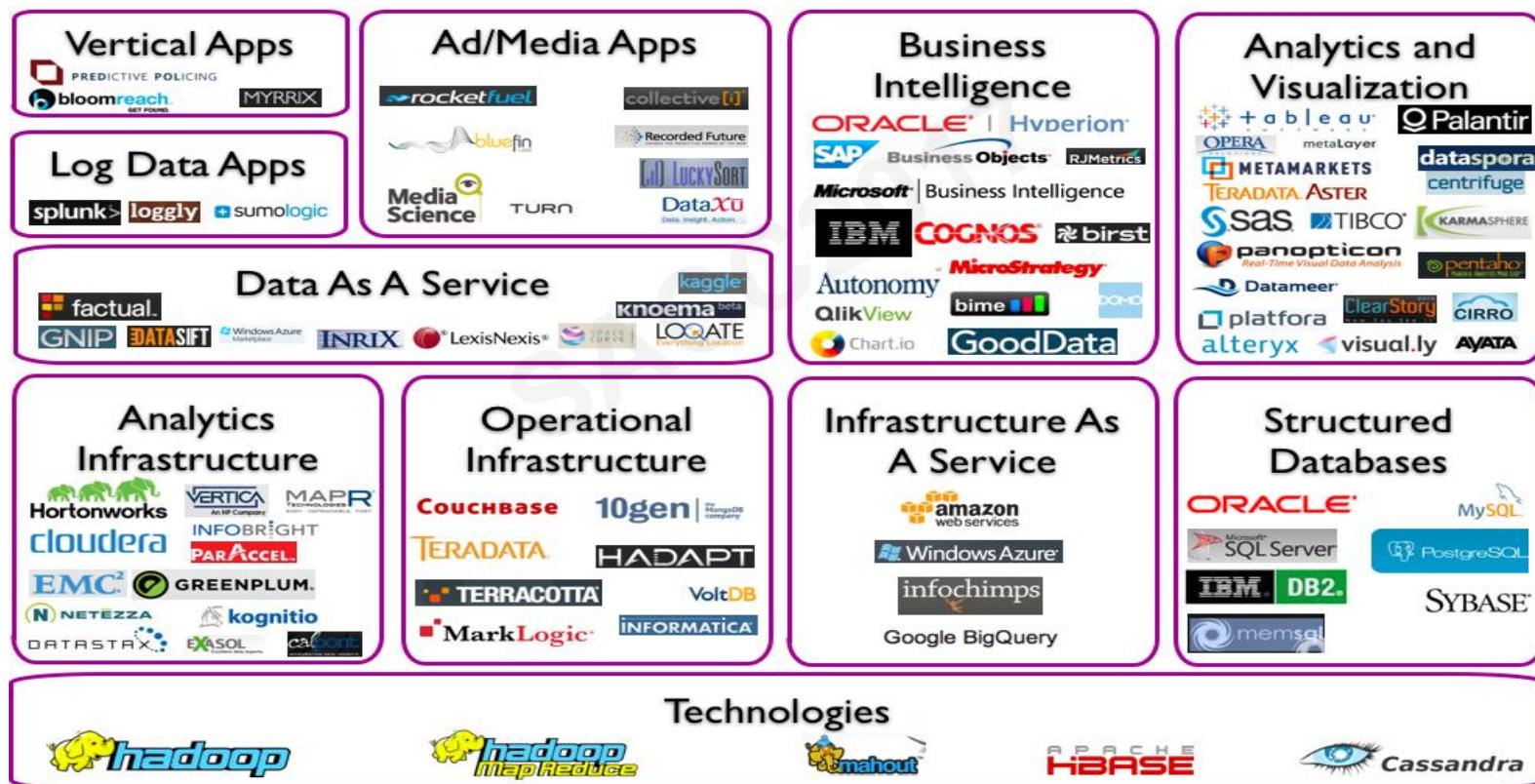
系统调优的挑战与难点

- 系统参数个数越来越多



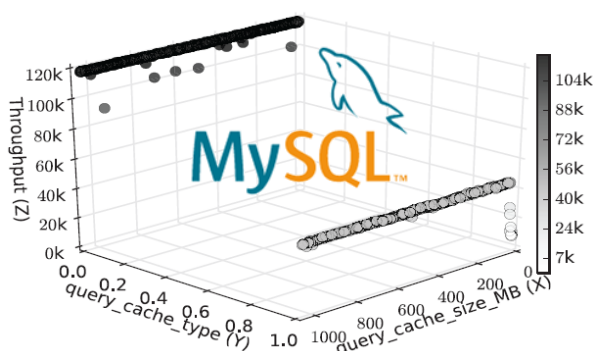
系统调优的挑战与难点

- 涉及的系统越来越多

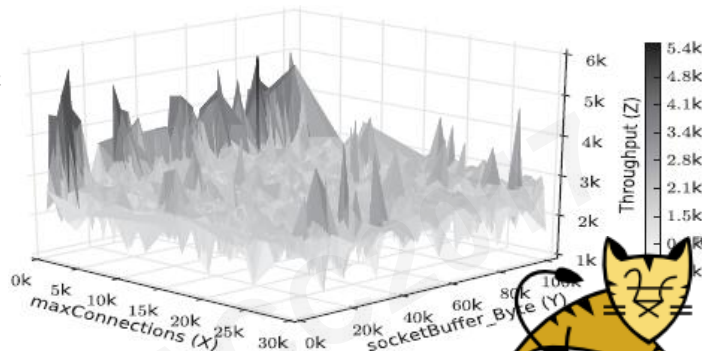


系统调优的挑战与难点

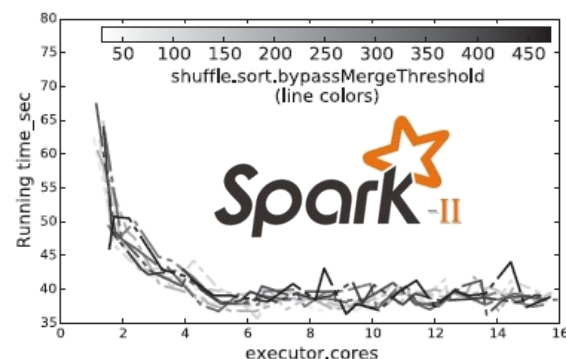
- 参数设置与系统、应用紧密相关
 - 不同系统导致不同的复杂性能曲面



(a) MySQL: uniform read

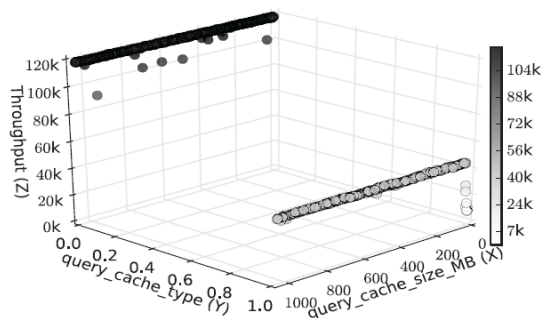


(b) Tomcat: default JVM settings

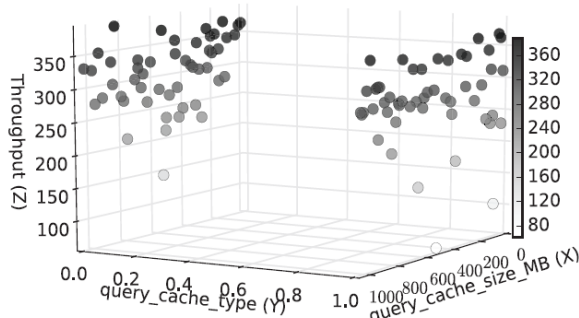


(c) Spark: standalone

- 不同应用负载导致不同性能曲面



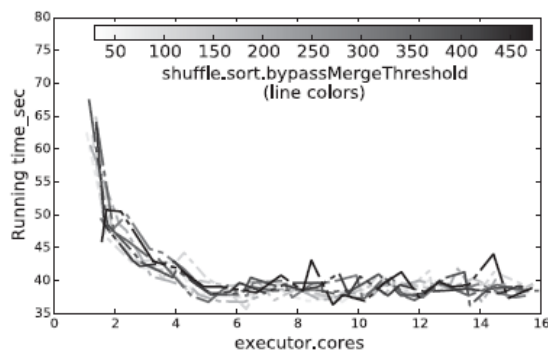
(a) MySQL: uniform read



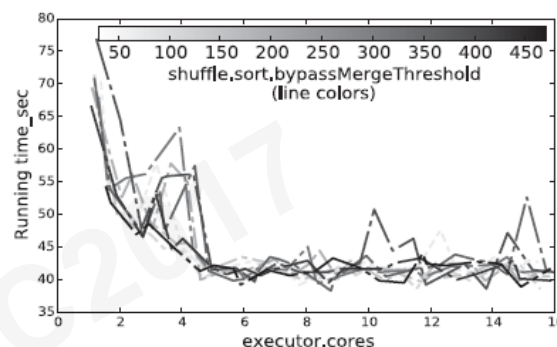
(d) MySQL: zipfian read-write

系统调优的挑战与难点

- 参数设置与系统、应用紧密相关
 - 不同**硬件部署环境**导致不同性能曲面



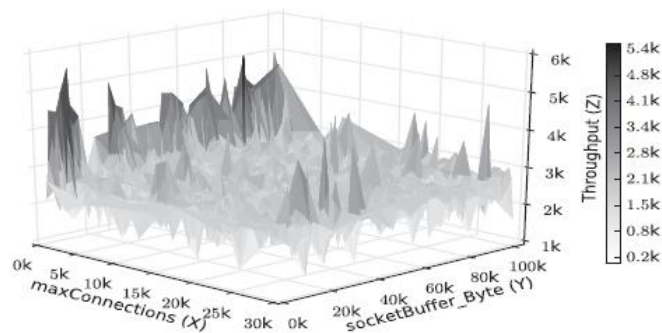
(c) Spark: standalone



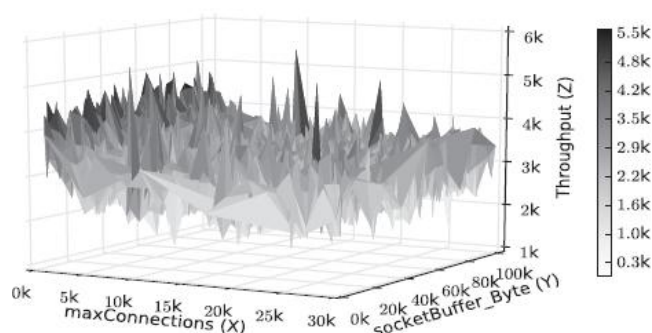
(f) Spark: cluster



- 不同**软件部署环境**导致不同性能曲面



(b) Tomcat: default JVM settings



(e) Tomcat: tuned JVM settings

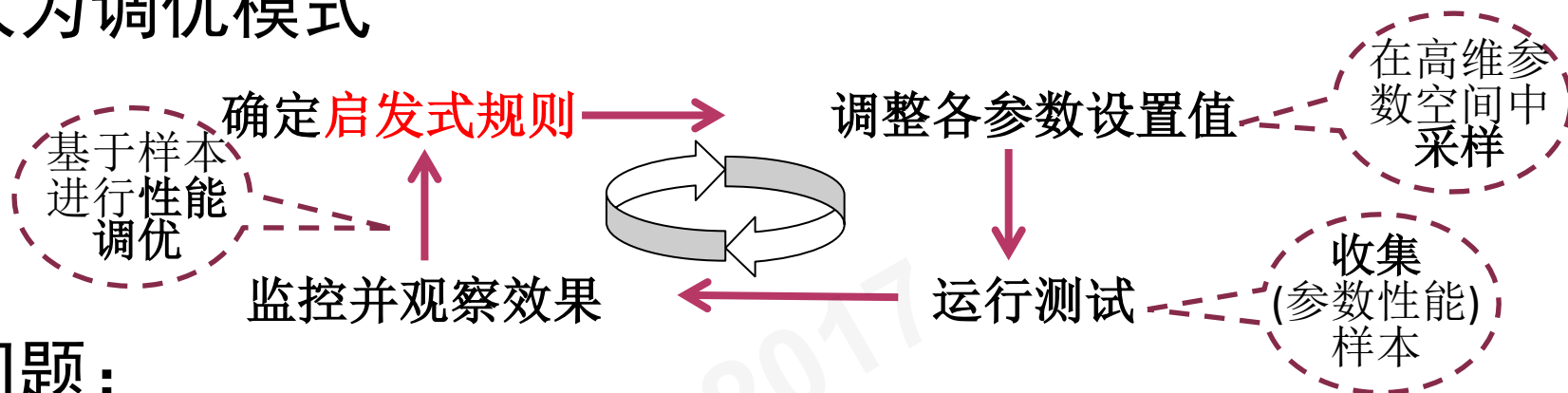


系统调优的挑战与难点

- 系统参数个数越来越多
- 涉及的系统越来越多
- 参数设置与系统、应用紧密相关
 - 系统、应用、硬件部署环境、软件部署环境等都能导致不同的性能曲面
 - ➔ 需要不同的系统性能调优路径

系统调优的挑战与难点

- 人为调优模式



- 问题：

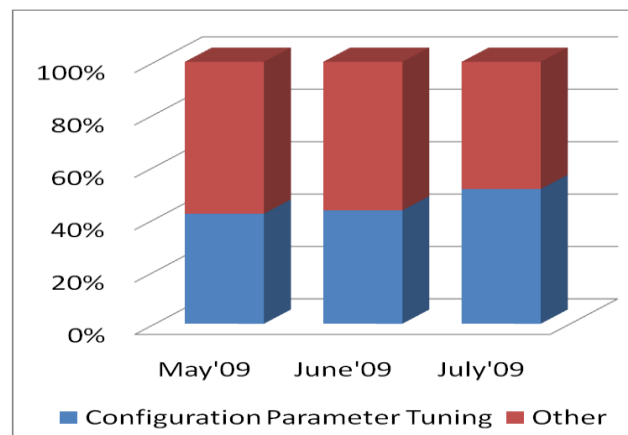
- 启发式规则需要大量专业知识与实践经验支持

- 一旦启发式规则错误，所有动作都是错的

- 调优过程耗时耗力

- 不断重复进行实验直到找到为止

或者，不调了、忍受着？



系统调优的痛点

- 企业痛点

- 大公司

- 聘请调优专家成本高
 - 系统调优耗时长
 - 培养优秀调优人员周期长、开销大

- 中小公司

- 支付高昂调优专家费用，性价比低
 - 系统性能不高，导致硬件开销大
 - 难以迅速上手新系统，导致业务受限

系统自动化调优的价值

- 辛苦的调优劳动

- ➔ 坐等系统运行到其最优性能

1. 仅调整系统参数值，即可使性能最大提升11倍

2. 节省人力开销

- ➔ 将5人半年的工作减少为机器2天

3. 减少对硬件的需求

- ➔ 从每26个虚拟机中去掉1个

4. 更公平地测试和比较系统性能

5. 确定系统瓶颈

- ➔ 分组件组合调优

系统自动化调优：应用案例

- Cloud+应用
 - 云端虚拟机上部署Tomcat服务器
 - 云端物理机使用的是ARMv8架构CPU
 - 虚拟机配置8核，4核用于网络，4核用于处理
 - 应用负载使得网络4核满载，处理4核利用率80%
- 企业专家认为无法再进行性能调优了
- 使用自动化调优工具**BestConfig**优化性能后

Metrics	Default	BestConfig	Improvement
Txns/seconds	978	1018	4.07% ↑
Hits/seconds	3235	3620	11.91% ↑
Passed Txns	3184598	3381644	6.19% ↑
Failed Txns	165	144	12.73% ↓
Errors	37	34	8.11% ↓

同一应用负载、同一系统
部署条件下：
使得可以从每26个虚拟机的
需求中减去1个

系统自动化调优：前沿进展

- BestConfig

BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning

- ACM SoCC 2017
- 可面向各种系统进行自动化参数调优，使性能最优
- 如Spark、Hadoop、MySQL、Hive、Casandra、Tomcat等
- 甚至JVM！！

- OtterTune

Automatic Database Management System Tuning Through Large-scale Machine Learning

- SIGMOD 2017
- 受到阿里数据库团队的热切关注
- 仅面向数据库进行参数调优，如MySQL、PostgreSQL

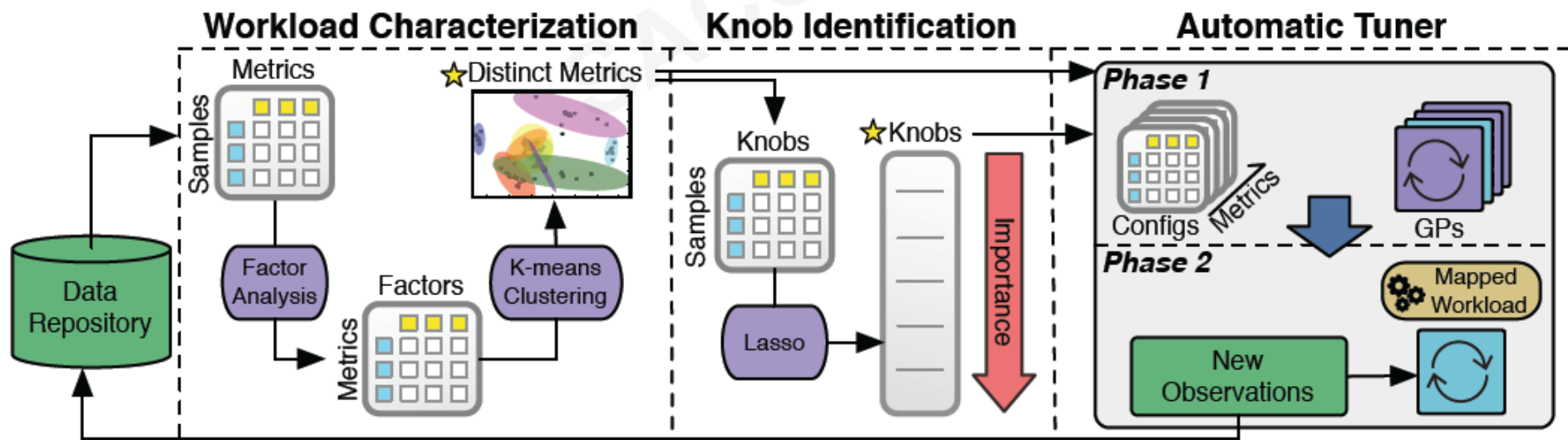
系统自动化调优：前沿进展

- OtterTune

- 卡耐基梅隆大学数据库团队研发

- 关键技术：

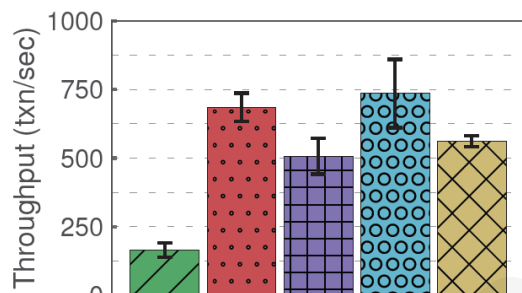
- 性能特征项匹配
- 关键参数发现
- 光滑性能曲面自动优化



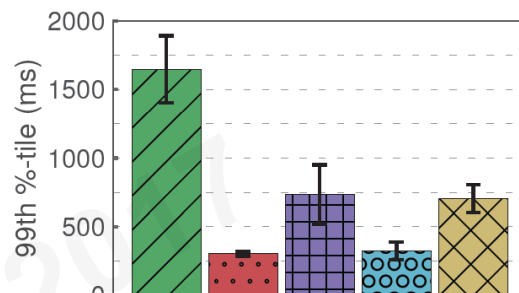
系统自动化调优：前沿进展

- OtterTune

- MySQL调优结果

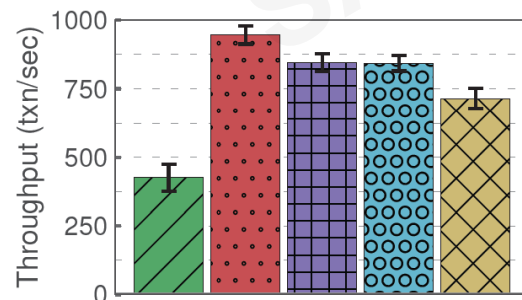


(a) TPC-C (Throughput)

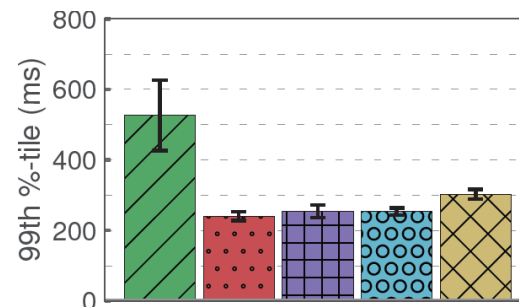


(b) TPC-C (99%-tile Latency)

- PostgreSQL调优结果



(a) TPC-C (Throughput)



(b) TPC-C (99%-tile Latency)



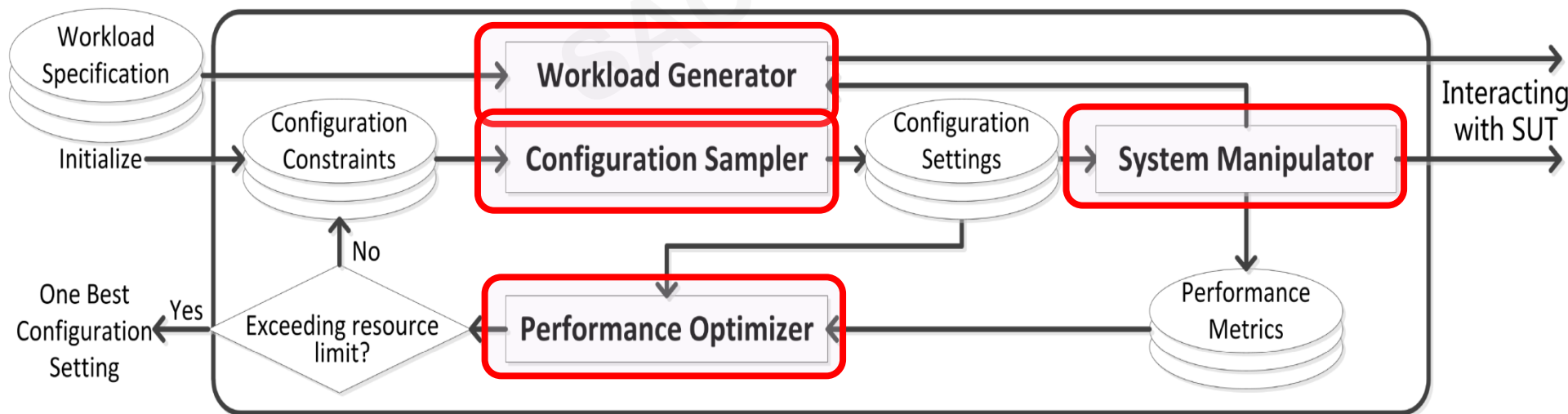
系统自动化调优：前沿进展

- BestConfig

- 中科院计算所, 先进计算机系统研究中心
- 关键技术:

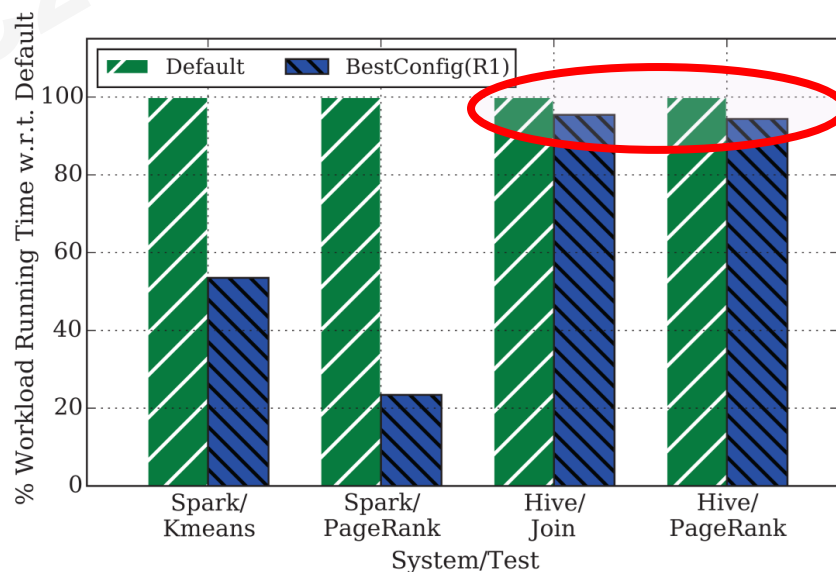
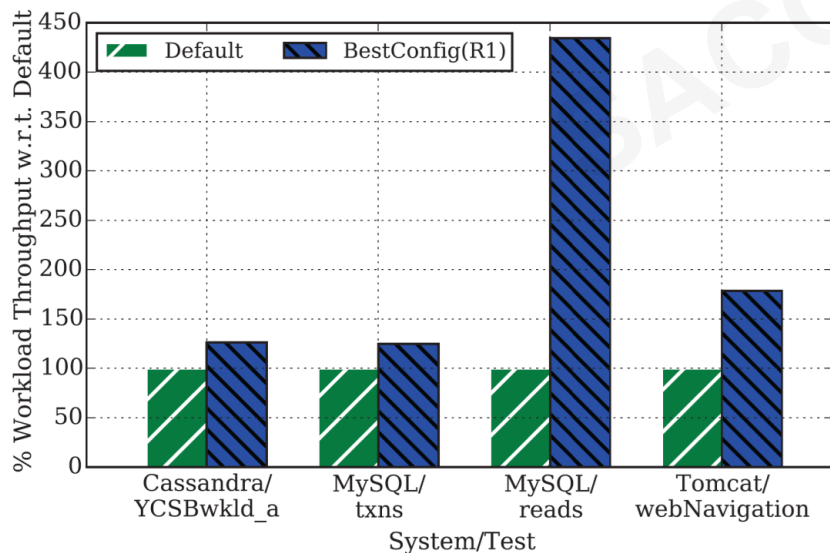
- 高可扩展调优系统架构
- 分割差异化采样算法
- 递归限定查找算法

大数据系统
前沿探索小组
研发



系统自动化调优：前沿进展

- BestConfig
 - 已应用于6个常见系统及JVM
 - 最多调优了109个参数



系统自动化调优：前沿进展

- BestConfig
 - 参数较多，增加调优可进行测试次数：100 → 500
 - Hadoop+Hive：性能提升2倍！！

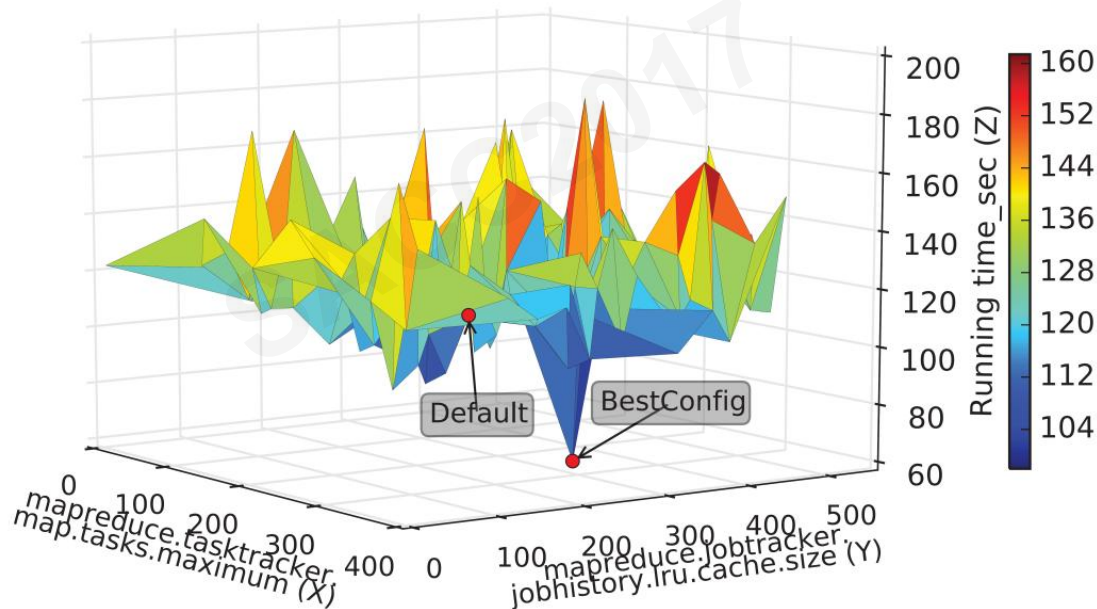


Figure 6: BestConfig reduces 50% running time of HiBench-Join on Hive+Hadoop within 500 tests.

系统自动化调优：前沿进展

- BestConfig
 - 发现了些你不知道的事情：网上一些调优规则是错的！
 - “MySQL的`thread_cache_size`值不要设置到超过200”
 - 使`thread_cache_size=11987`，性能也可以很好，甚至更好！！！！

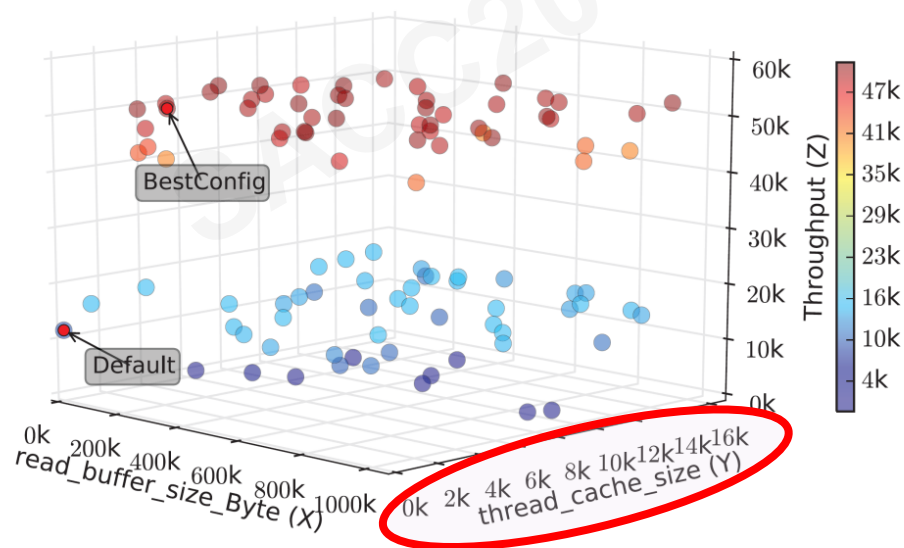


Figure 7: Throughputs for varied `thread_cache_size` of MySQL, invalidating the manual tuning guideline.

系统自动化调优：前沿进展

- BestConfig vs. OtterTune

	BestConfig	OtterTune
可调系统	通用，各种系统 ✓	数据库专用
当前最大性能提升结果	11倍 ✓	6倍
前期准备	无需 ✓	参数性能样本数据库
调优终止条件设定	按时间、次数等 ✓	长时间或人为观察
开源代码实现程度	调优整过程、全自动 ✓	仅算法代码
取样算法	全覆盖、可扩展 ✓	普通随机算法
调优算法	不依赖特定性能曲面 ✓	面向光滑曲面
样本个数需求	100以上即可 ✓	1000+

系统自动化调优的价值

- 辛苦的调优劳动

- ➔ 坐等系统运行到其最优性能

1. 仅调整系统参数值，即可使性能最大提升11倍

2. 节省人力开销

- ➔ 将5人半年的工作减少为机器2天

3. 减少对硬件的需求

- ➔ 从每26个虚拟机中去掉1个

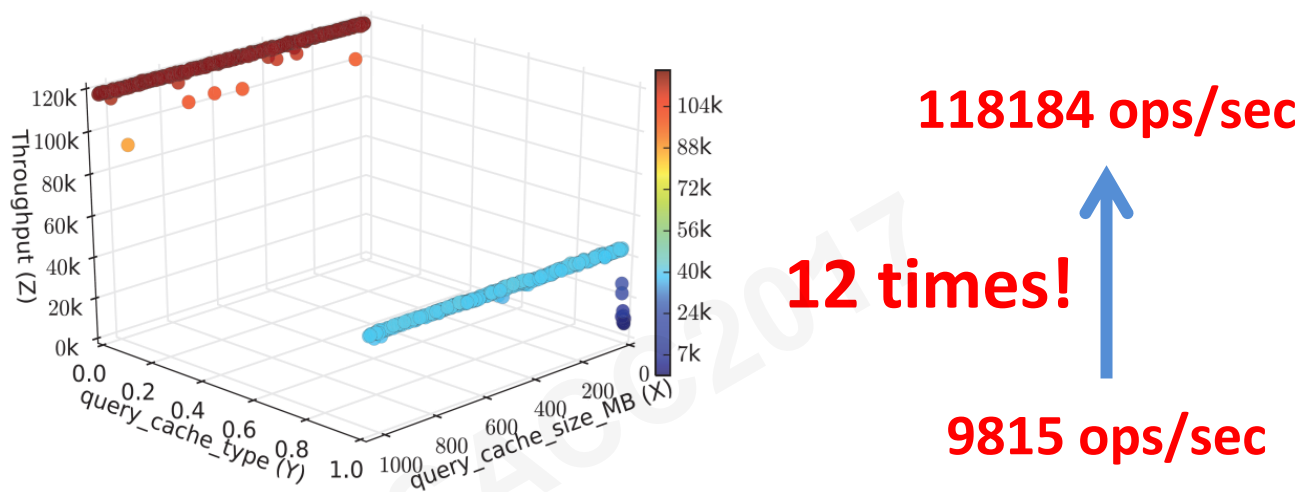
4. 更公平地测试和比较系统性能

5. 确定系统瓶颈

- ➔ 分组件组合调优

系统自动化调优的价值

- 仅调整系统参数值，即可使性能最大提升11倍



(a) MySQL under uniform reads

- 节省人力开销
 - 5人半年
 - ➔ 机器，自动化调优：2天！

- 人工调优极为困难
- 时间、人力成本开销极大

系统自动化调优的价值

- 减少对硬件的需求
 - 云端虚拟机上、系统性能自动化调优

Metrics	Default	BestConfig	Improvement
Txns/seconds	978	1018	4.07% ↑
Hits/seconds	3235	3620	11.91% ↑
Passed Txns	3184598	3381644	6.19% ↑
Failed Txns	165	144	12.73% ↓
Errors	37	34	8.11% ↓

同一应用负载、同一系统
部署条件下：
使得可以从每**26**个虚拟机的
需求中减去**1**个

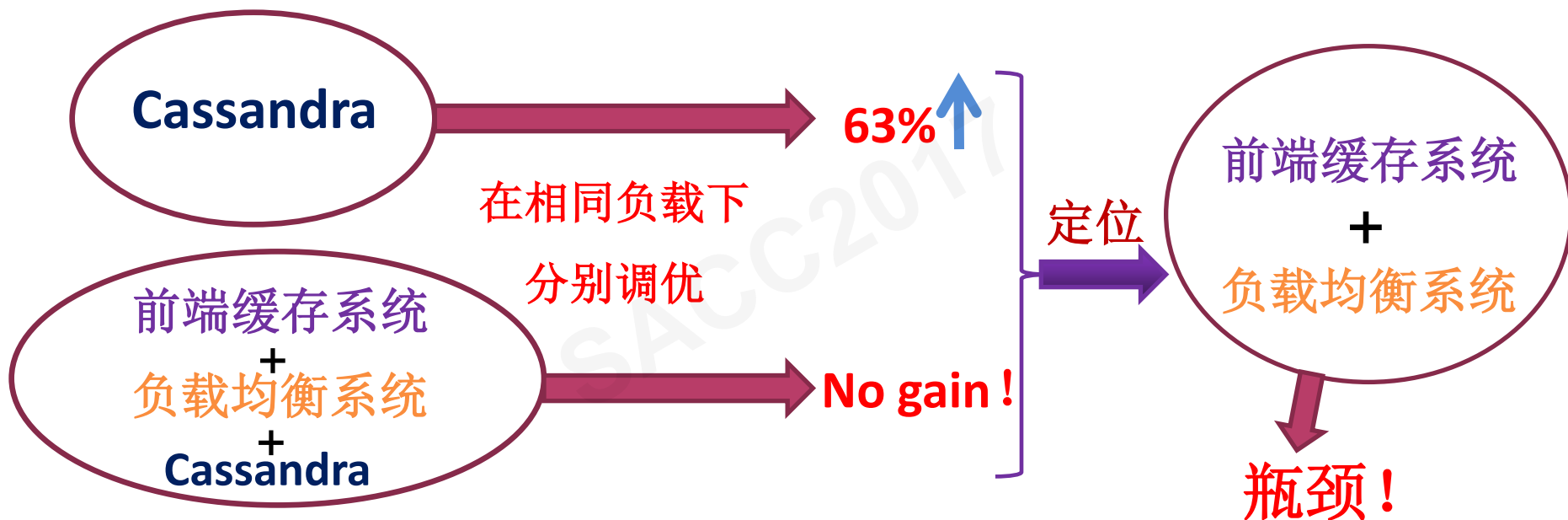
- 更公平地测试和比较系统性能

System	Throughput (调优前)	Throughput (调优后)
Cassandra	12199 ops/sec	16015 ops/sec
MySQL	9815 ops/sec	118184 ops/sec

能够为系统测
试与比较提供
更公平的起点

系统自动化调优的好处

- 分组件组合调优，确定系统瓶颈



系统自动化调优的价值

- 辛苦的调优劳动

➔ 坐等系统运行到其最优性能

1. 仅调整系统参数值，即可使性能
2. 节省人力开销
3. 减少硬件开销
4. 更公平地测试和比较系统性能
5. 确定系统瓶颈

如何使用BestConfig 进行自动化调优?

SACC2017

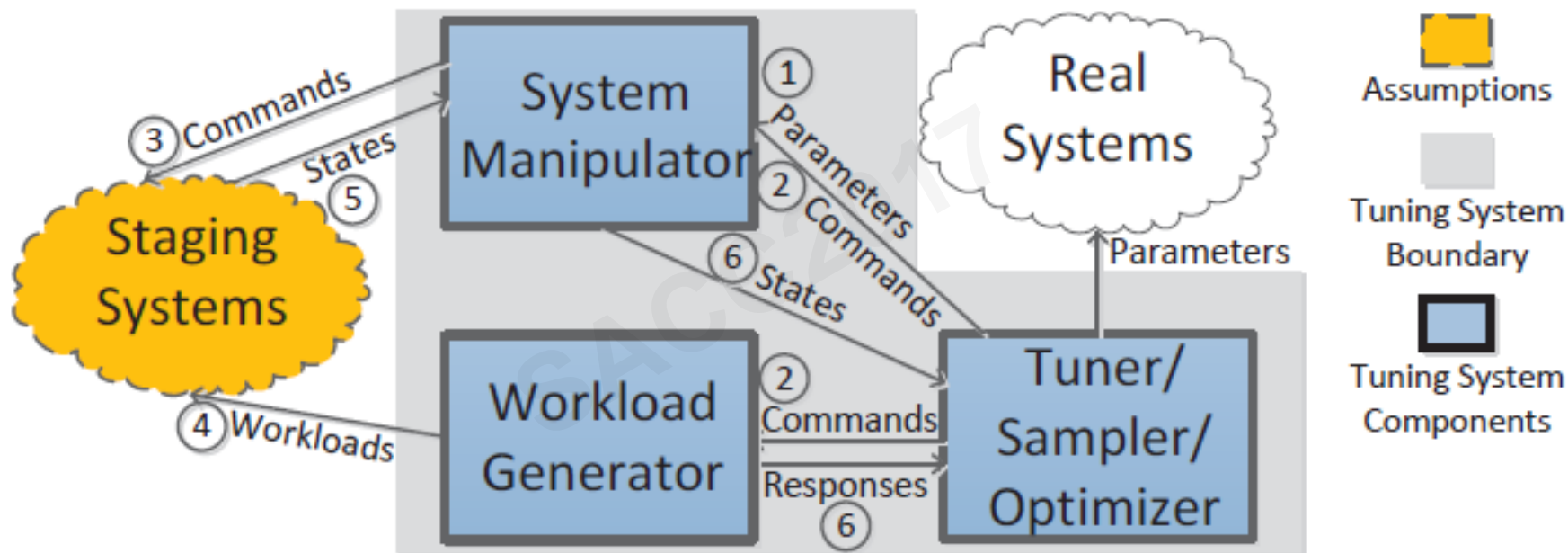
使用BestConfig

- 环境
 - 实际应用部署环境，或
 - 应用准上线环境
- 前提条件
 - 待调优系统或集群节点的IP地址，登录名和密码
 - 开放远程SSH登录系统的权限
 - 下载BestConfig源码

<https://github.com/zhuyuqing/bestconf>

使用BestConfig

- 部署架构



使用BestConfig

- 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

控制

1. 在待调节系统或集群节点上部署好调优所需 **shell脚本**



2. 将调优工具部署到启动调优程序的机器节点



3. 修改调优工具的配置文件，以符合实际待调系统需求



4. 启动调优过程，生成最佳参数配置文件






使用BestConfig

- 以Spark为例 
 - Shell脚本

- 待调系统相关脚本

- start.sh isStart.sh





- stop.sh isClosed.sh terminateSystem.sh

	stop.sh	155 Bytes
	start.sh	150 Bytes
	terminateSystem.sh	32 Bytes
	isClosed.sh	268 Bytes
	isStart.sh	304 Bytes

- 测试节点相关脚本

- startTest.sh getTestResult.sh

- terminateTest.sh isFinished.sh

	terminateTest.sh	452 Bytes
	startTest.sh	205 Bytes
	isFinished.sh	278 Bytes
	getTestResult.sh	870 Bytes

使用BestConfig

- 以Spark为例 
 - Shell脚本——待调系统相关脚本

部署到Spark的Master节点和Worker节点的start.sh 和stop.sh 两个脚本有所不同

- 部署到Master节点的脚本内容 (start.sh 和 stop.sh)

start.sh

```
#!/bin/bash
path=/opt/spark/startresults.txt
rm -f /opt/spark/startresults.txt;
touch /opt/spark/startresults.txt;
cd /opt/hadoop-2.6.5/hadoop-2.6.5
sbin/start-all.sh >& $path
bin/hadoop dfsadmin -safemode leave
cd /opt/hadoop-2.6.5/spark/spark-1.6.1
sbin/start-all.sh >> $path
echo ok >> $path
```

stop.sh

```
#!/bin/bash
path=/opt/spark/stopresults.txt
rm -f /opt/spark/stopresults.txt
touch /opt/spark/stopresults.txt
cd /opt/hadoop-2.6.5/hadoop-2.6.5
sbin/stop-all.sh >& $path
cd /opt/hadoop-2.6.5/spark/spark-1.6.1
sbin/stop-all.sh >> $path && rm -rf /opt/hadoop-2.6.5/hadoop-2.6.5/logs/* && echo ok >> $path
```

使用BestConfig

- 以Spark为例 
 - Shell脚本——待调系统相关脚本

部署到Spark的Master节点和Worker节点的start.sh 和stop.sh 两个脚本有所不同

- 部署到Worker节点的脚本内容 (start.sh 和 stop.sh)

start.sh

```
#!/bin/bash
path=/opt/spark/startresults.txt
rm -f /opt/spark/startresults.txt;
touch /opt/spark/startresults.txt;
echo ok >& $path
echo ok >> $path
```

stop.sh

```
#!/bin/bash
path=/opt/spark/stopresults.txt
rm -rf /opt/hadoop-2.6.5/dfs/data/* >& $path
rm -rf /opt/hadoop-2.6.5/hadoop-2.6.5/logs/* && echo ok >> $path
```

使用BestConfig

- 以Spark为例 
 - Shell脚本——待调系统相关脚本
 - Worker 和 Master节点相同的脚本内容
(isStart.sh, terminateSystem.sh 和 isClosed.sh)

isStart.sh

```
#!/bin/bash
resultFile=/opt/spark/startresults.txt
rm -f /opt/spark/startresult.txt
tail -n 2 $resultFile > /opt/spark/startresult.txt

while read line
do
    if [ "$line" == "ok" ];
    then
        echo "ok"
    else
        echo "not ok!"
    fi
done < /opt/spark/startresult.txt
```

使用BestConfig

- 以Spark为例 
 - Shell脚本——待调系统相关脚本
 - Worker 和 Master节点相同的脚本内容
(isStart.sh, terminateSystem.sh 和 isClosed.sh)

terminateSystem.sh

```
cd /opt/spark
./stop.sh
echo ok
```

isClosed.sh

```
#!/bin/bash
resultFile=/opt/spark/stopresults.txt
rm -f ./stopresult.txt
tail -n 2 $resultFile > ./stopresult.txt

while read line
do
    if [ "$line" == "ok" ];
    then
        echo "ok"
    else
        echo "not ok!"
    fi
done < ./stopresult.txt
```

使用BestConfig

- 以Spark为例  Spark-2
 - Shell脚本——测试节点相关脚本

startTest.sh

```
#!/bin/bash
path=/opt/spark/testresults.txt
rm -f /opt/HiBench-master/report/hibench.report
rm -f /opt/spark/testresults.txt
touch /opt/spark/testresults.txt
cd /opt/HiBench-master
bin/run-all.sh >& $path
```

isFinished.sh

```
#!/bin/bash
resultFile=/opt/spark/testresults.txt
rm -f ./testresult.txt
tail -n 2 $resultFile > ./testresult.txt

while read line
do
    if [ "$line" == "Run all done!" ];
    then
        echo "ok"
    else
        echo "not ok!"
    fi
done < ./testresult.txt
```


使用BestConfig

- 以Spark为例  Spark-11
 - Shell脚本——测试节点相关脚本

getTestResult.sh

```
resultFile=/opt/HiBench-master/report/hibench.report
minduration=0
i=0
result=0
if [ -f "$resultFile" ]; then
cat $resultFile > /opt/spark/testresult.txt
while read line
do
    if [ -n "$line" ]; then
        duration=`echo $line|awk -F ' ' '{print $5}'|tr -d ' '`
        if [ $duration == "Duration(s)" ]; then
            then
                ((i++))
            else
                ((i++))
                result=`echo "$result + $duration"|bc`
                if [ $i == 3 ]; then
                    break
                fi
            fi
        fi
    fi
done < /opt/spark/testresult.txt
if [ $i == 3 ]; then
    num2=10000
    num3=`echo "scale=10;$num2/$result"|bc`
    echo $num3
else
    echo "error"
fi
else
    echo "not exist"
fi
```

使用BestConfig

- 以Spark为例  Spark-11
 - Shell脚本——测试节点相关脚本

terminateTest.sh

```
pidprepare=`pgrep prepare`  
pidrun=`pgrep run`  
pidrunall=`pgrep run-all`  
echo $pidprepare  
echo $pidrun  
echo $pidrunall  
if [ -n "$pidprepare" ];  
then  
    echo "kill prepare"  
    kill -9 $pidprepare && kill -9 $pidprepare && echo yes  
fi  
if [ -n "$pidrunall" ];  
then  
    echo "kill runall"  
    kill -9 $pidrunall && kill -9 $pidrunall && echo yes  
fi  
if [ -n "$pidrun" ];  
then  
    echo "kill run"  
    kill -9 $pidrun && kill -9 $pidrun && echo yes  
fi  
echo ok
```


使用BestConfig

- 以Spark为例 
 - 接口实现——配置文件读写

```
public interface ConfigReadin {  
    void initial(String server, String username, String password, String localPath, String remotePath);  
    void downLoadConfigFile(String fileName);  
    HashMap modifyConfigFile(HashMap hm, String filepath);  
    HashMap modifyConfigFile(HashMap hm);  
}  
  
public interface ConfigWrite {  
    void initial(String server, String username, String password, String localPath, String remotePath);  
    void uploadConfigFile();  
    void writetoConfigfile(HashMap hm);  
}
```

- 📁 src/spark
 - 📁 cn.ict.zyq.bestConf.cluster.InterfaceImpl
 - 📄 SparkConfigReadin.java
 - 📄 SparkConfigWrite.java

使用BestConfig

- 以Spark为例 
 - 接口实现——配置文件目录

data

bestconf.properties

defaultConfig.yaml

defaultConfig.yaml_range

SUTconfig.properties

配置调优算法和样本集

待调参数的默认取值

待调参数的取值范围

待调系统和测试相关配置

使用BestConfig

- 以Spark为例 
 - 接口实现——配置文件目录

- **bestconf.properties**

```
configPath=data/SUTconfig.properties  
yamlPath=data/defaultConfig.yaml
```

```
InitialSampleSetSize=24    设置初始样本集大小  
RRSMaxRounds=1            设置算法的最大轮数
```

使用BestConfig

- 以Spark为例  Spark-II
 - 接口实现——配置文件目录

- defaultConfig.yaml （Spark待调配置参数列表）

```
Size.spark.reducer.maxSizeInFlight: 49152
Size.spark.shuffle.file.buffer: 32
spark.shuffle.sort.bypassMergeThreshold: 200
Time.spark.speculation.interval: 100
spark.speculation.multiplier: 1.5
spark.speculation.quantile: 0.75
Size.spark.broadcast.blockSize: 4096
Type.spark.io.compression.codec: 2
Size.spark.io.compression.lz4.blockSize: 32
Size.spark.io.compression.snappy.blockSize: 32
Bool.spark.kryo.referenceTracking: 1
Size.spark.kryoserializer.buffer.max: 65536
```

使用BestConfig

- 以Spark为例  Spark-II
 - 接口实现——配置文件目录

- defaultConfig.yaml_range（Spark待调配置参数范围）

```
Size.spark.reducer.maxSizeInFlight: "[2048,1048576]"
Size.spark.shuffle.file.buffer: "[2,1024]"
spark.shuffle.sort.bypassMergeThreshold: "[10,1000]"
Time.spark.speculation.interval: "[10,2000]"
spark.speculation.multiplier: "[1,5]"
spark.speculation.quantile: "[0,1]"
Size.spark.broadcast.blockSize: "[1024,131072]"
Type.spark.io.compression.codec: "[0,3]"
Size.spark.io.compression.lz4.blockSize: "[2,1024]"
Size.spark.io.compression.snappy.blockSize: "[2,1024]"
Bool.spark.kryo.referenceTracking: "[0,1]"
Size.spark.kryoserializer.buffer.max: "[8192,1048576]"
```

使用BestConfig

- 以Spark为例 
 - 接口实现——配置文件目录

- **SUTconfig.properties**

```
systemName=Spark
configFileStyle=yaml
numServers=1
server0=IPO
username0=root
password0=ljx123
localDataPath=data
shellsPath=/opt/spark
remoteConfigFilePath=/opt/HiBench-master/conf/workloads/ml
interfacePath=cn.ict.zyq.bestConf.cluster.InterfaceImpl
sutStartTimeoutInSec=300
testDurationTimeoutInSec=1800
maxRoundConnection=60
targetTestErrorNum=10
```

设置待调系统名字, yaml 是配置文件的类型

设置远程服务器IP, 用户名及密码

调优程序的配置文件目录

shell脚本在待调系统上的目录

待调系统的配置文件目录

接口文件所在程序目录

待调系统启动超时设置

测试超时设置

连接远程系统的最大失败次数

测试失败的最大次数

使用BestConfig

- 以Spark为例  Spark-II
 - 接口实现——配置文件目录

- **SUTconfig.properties**

<code>maxConsecutiveFailedSysStarts=10</code>	系统最大连续启动失败次数
<code>sshReconnectWaitingtimeInSec=10</code>	ssh重新连接等待时间
<code>maxConnectionRetries=10</code>	ssh连接最大重试次数
<code>performanceType=3</code>	自定义性能模型
<code>targetTestServer=IP1</code>	远程测试节点的IP, 用户名, 密码
<code>targetTestUsername=root</code>	
<code>targetTestPassword=*</code>	
<code>targetTestPath=/opt/spark</code>	测试脚本所在目录

使用和扩展BestConfig

- 使用BestConfig
 - Shell脚本
 - 待调系统相关
 - 测试节点（应用负载）相关
 - 配置文件目录
 - 待调参数相关——未来可从待测系统中自动抽取
 - 调优流程、待调系统相关配置文件
- 扩展BestConfig
 - 扩展采样算法
 - 扩展优化算法

扩展BestConfig

- 扩展采样算法
 - 继承ConfigSampler抽象类

bestconf / src / main / cn / ict / zyq / bestConf / bestConf / sampler / ConfigSampler.java

```
abstract Instances sampleMultiDimContinuous(ArrayList<Attribute> atts, int sampleSetSize, boolean useMid);
```

- 扩展优化算法
 - 实现Optimization接口

bestconf / src / main / cn / ict / zyq / bestConf / bestConf / optimizer / Optimization.java

```
public interface Optimization {  
    public void optimize(String preLoadDataPath);  
}
```

bestconf / src / main / cn / ict / zyq / bestConf / bestConf / BestConf.java

```
316 public static void startOneTest(int method, String preLoadDataPath){  
317     BestConf bestconf = new BestConf();  
318     Optimization opt;  
319     switch(method){  
320     case 0:  
321     default:  
322         opt = new RBSoDDSOptimization(bestconf, BestConf.InitialSampleSetSize, BestConf.RRSMaXrounds);  
323         break;  
324     }
```

融入自动化、智能化潮流， 让它为你服务

~~DBA要失业~~



朱好晴



THANKS

<https://github.com/zhuyuqing/bestconf>