

---

# BestConfig Documentation

*Release*

**ljsx**

**Jan 25, 2018**



---

## Contents

---

|          |                          |           |
|----------|--------------------------|-----------|
| <b>1</b> | <b>Contents</b>          | <b>3</b>  |
| 1.1      | QuickStart . . . . .     | 3         |
| 1.2      | FAQ . . . . .            | 11        |
| 1.3      | Use cases . . . . .      | 13        |
| <b>2</b> | <b>Citing BestConfig</b> | <b>21</b> |



**BestConfig** is a system for automatically **finding a best configuration setting within a resource limit** for a deployed system under a given application workload. BestConfig is designed with an extensible architecture to automate the configuration tuning **for general systems**.



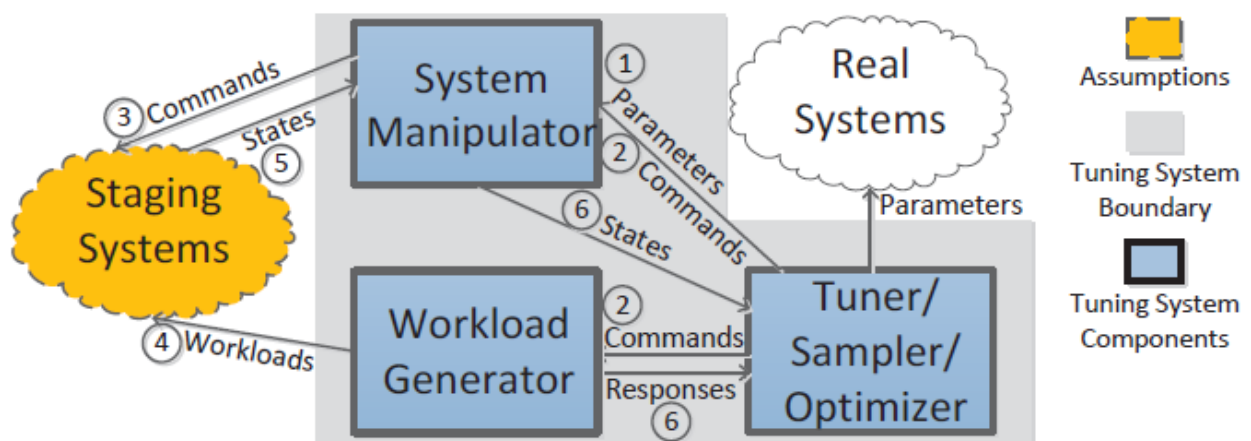
## 1.1 QuickStart

Good tools make system performance tuning quicker, easier and cheaper than if everything is done manually or by experience.

Bestconfig can find better configurations for a specific large-scale system deployed for a given application workload.

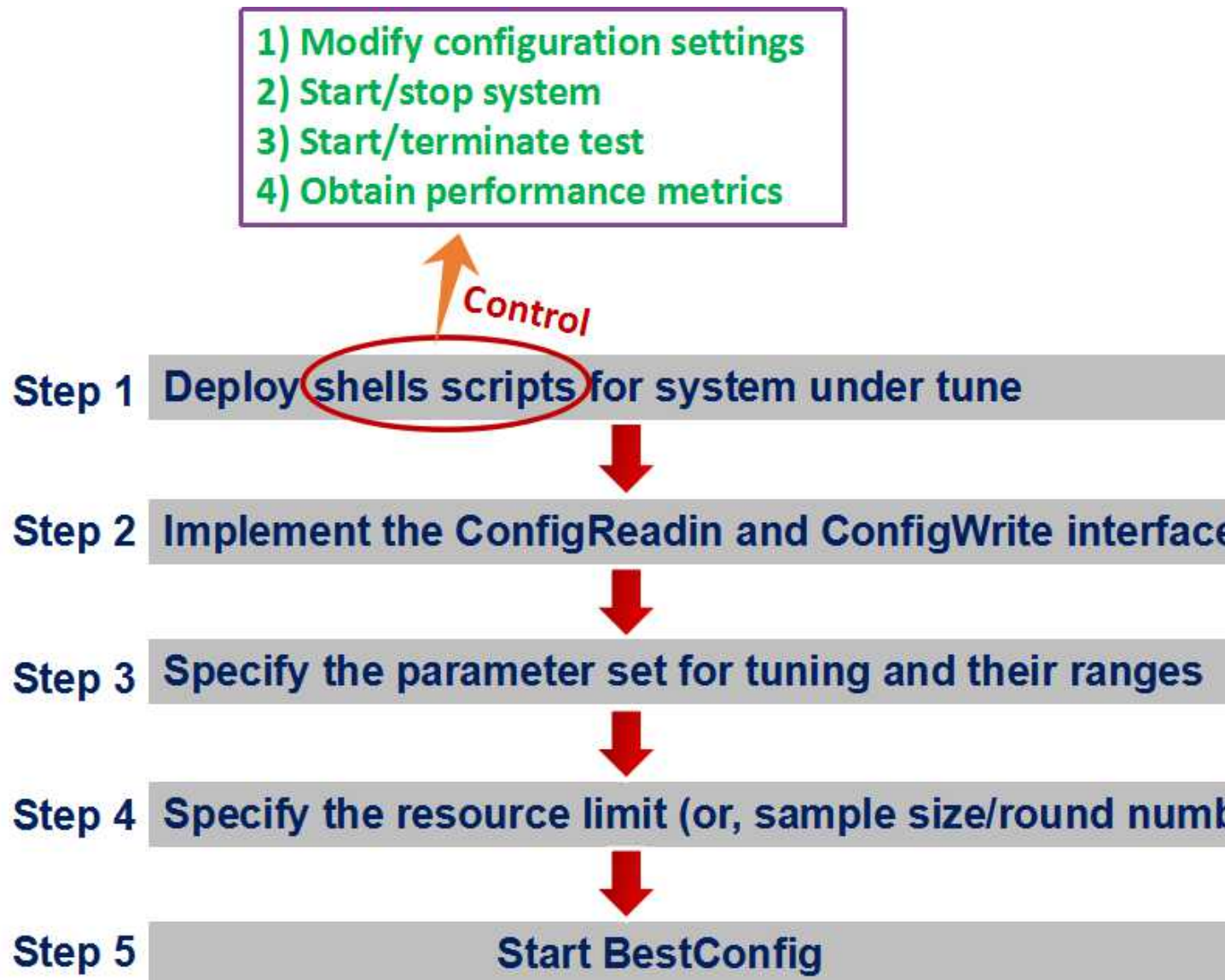
- *Overview*
- *BestConfig Tuning – Taking Spark as the example SUT*
- *Implementing your own sampling/tuning algorithms for BestConfig*

### 1.1.1 Overview



Here, “deployment environment” refers to the actual running environment of your applications, while “staging environment” is some environment that is almost the same as the deployment environment but where tests are run without

interfering the actual application.



The detailed method of using BestConfig to tune practical system is as the following, which can be showed by a case of spark tuning.

### 1.1.2 BestConfig Tuning – Taking Spark as the example SUT

#### Step 1. Deploy shells scripts for system under tune

There are 9 shell scripts in BestConfig and they are classified into two groups.

1. One group consists of 5 shell scripts. They are start.sh, isStart.sh, stop.sh, isClosed.sh and terminateSystem.sh and deployed on the system under tune.

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

```

#!/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

```

```

#!/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

```

```

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

```

```

#!/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

```

### 3. Identical shell scripts on master and worker node

```

#!/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

```

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

```
#!/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
```

2. The other group consists of 4 shell scripts. They are startTest.sh, getTestResult.sh, terminateTest.sh and isFinished.sh and deployed on the test node.

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

```
#!/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
```

```
#!/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
```

```

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 [ $duration == "Duration(s)" ];
        then
            ((i++))
        else
            ((i++))
            result=`echo "$result + $duration"|bc`
            if [ $i == 3 ]; then
                break
            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

```

```

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

```

## Step 2. Implement the ConfigReadin and ConfigWrite interfaces

As for spark tuning, we need to implement the ConfigReadin and ConfigWrite interfaces as `SparkConfigReadin` and `SparkConfigWrite`.

Next, we need to compile `SparkConfigReadin` and `SparkconfigWrite` to bytecodes. Then the location(path) of compiled bytecodes need to be added to classpath of BestConfig project.

```

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.clusterInterfaceImpl
  - SparkConfigReadin.java
  - SparkConfigWrite.java

## Step 3. Specify the parameter set for tuning and their ranges

1. An example of defaultConfig.yaml (specifying the parameters for tuning)

```

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

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]"

```

#### Step 4. Specify the resource limit and things about the tuning environment (or, sample size/round number)

1. bestconf.properties

```

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

```

```

InitialSampleSetSize=24
RRSMaxRounds=1

```

**initial size of sample set**  
**maximum round times of RRS algrithom**

2. SUTconfig.properties

|                                                                         |                                                               |
|-------------------------------------------------------------------------|---------------------------------------------------------------|
| <code>systemName=Spark</code>                                           | target system to tune                                         |
| <code>configFileStyle=yaml</code>                                       | type of configuration file                                    |
| <code>numServers=1</code>                                               |                                                               |
| <code>server0=IP0</code>                                                | ip, user and password of remote node                          |
| <code>username0=root</code>                                             |                                                               |
| <code>password0=1jx123</code>                                           |                                                               |
| <code>localDataPath=data</code>                                         | directory of configuration files of BestConfig                |
| <code>shellsPath=/opt/spark</code>                                      | the path of shell scripts on system under test                |
| <code>remoteConfigFilePath=/opt/HiBench-master/conf/workloads/ml</code> | config file path of system under test                         |
| <code>interfacePath=cn.ict.zyq.bestConf.cluster.InterfaceImpl</code>    | directory of interface implementation file                    |
| <code>sutStartTimeoutInSec=300</code>                                   | time out of system start-up                                   |
| <code>testDurationTimeoutInSec=1800</code>                              | time out of test                                              |
| <code>maxRoundConnection=60</code>                                      | maximum number of connections permitted for remote connection |
| <code>targetTestErrorNum=10</code>                                      | maximum number of errors for a certain test                   |
| <code>maxConsecutiveFailedSysStarts=10</code>                           | maximum number of consecutive failure of system start-up      |
| <code>sshReconnectWaitingTimeInSec=10</code>                            | waiting time of ssh reconnection                              |
| <code>maxConnectionRetries=10</code>                                    | maximum number of ssh retry connections                       |
| <code>performanceType=3</code>                                          | user-defined performance model                                |
| <code>targetTestServer=IP1</code>                                       |                                                               |
| <code>targetTestUsername=root</code>                                    | ip, user and password of test node                            |
| <code>targetTestPassword=*</code>                                       |                                                               |
| <code>targetTestPath=/opt/spark</code>                                  | path of shell scripts on test node                            |

## Step 5. Start BestConfig

Now, you can start BestConfig. BestConfig will automatically run the tuning process without any requirement for user interferences, until the tuning process ends due to resource exhaustion or unhandlable environment errors.

BestConfig will output the best configuration setting into files once the tuning is done.

- (1). `cd bestconf-master`
- (3). `ant run`

### 1.1.3 Implementing your own sampling/tuning algorithms for BestConfig

You can also choose to extend and tailor BestConfig for your specific use cases using your own sampling/tuning algorithms.

1. **To implement your own sampling algorithms** → **Extend the** abstract class of `ConfigSampler`

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

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

2. **To implement your own tuning algorithms** → **Implement the** interface of `Optimization`

`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
```

```
public static void startOneTest(int method, String preLoadDataPath){
    BestConf bestconf = new BestConf();
    Optimization opt;
    switch(method){
    case 0:
    default:
        opt = new RBSoDDSOptimization(bestconf, BestConf.InitialSampleSetSize, BestConf.RRSMaxRounds);
        break;
    }
}
```

## 1.2 FAQ

**Q1. When I tried to start bestconfig by executing start.sh, an exception of “can’t find the class of bestconf” occurred?**

You need to move bestconf.jar to deploy directory. And remember to move the data directory to the deploy directory.

**Q2. Why an exception of “connection refused” occurred?**

You need to modify the ssh\_config file located in /etc/ssh/ directory. Set the value of “PermitRootLogin” to yes and then restart the ssh service.

**Q3. In production environment, the scale of hadoop cluster is considerably large, for example, the number of nodes in hadoop cluster may reach 1500, so the overhead of starting and stopping the cluster is significantly huge. In that case, is BestConfig still applicable?**

No problem. This just depends on how long you can put up with the tuning process. If you can bear a long waiting time, then you can use Bestconfig to obtain a best configuration setting.

**Q4. Can BestConfig only be used for software system tuning? Can it also be able to tune hardware system?**

Now, BestConfig can tune both software and hardware systems. Given the set of parameters of system (software / hardware) under tune and their valid ranges, then BestConfig is able to generate a best configuration setting.

**Q5. Can BestConfig only be used to tune a certain system such as MySQL or Hadoop? Is it able to tune other systems such as database systems or big data systems.**

Yes! BestConfig is a general system tuning tool and able to tune the widely used systems including MySQL, Tomcat, JVM, Hadoop(Hive), Spark, Cassandra, etc.

### Q6. Is BestConfig tuning process based on the online real system and workload?

No, the tuning process of BestConfig is run on the staging environment which is a mirror of the production environment, using the same actual deployment settings (e.g. hardware, clustering, software, etc.).

It is mainly for a final test, using live data, of the system before production. And, implementing the real application workload in the workload generator is possible for the system in the staging environment, e.g., by log replay.

When the tuning process on the staging systems ends, the best configuration setting is obtained. Then we apply the best configuration setting to the real system. In this way, samples can be collected without affecting applications on the real system deployment.

### Q7. The workloads generated by large number of users on the online systems are constantly changing. Can the workloads applied in the staging environment exactly simulate the real online workloads?

The application workloads can be classified into two types. One is periodic workload, the other is non-periodic workload. As for periodic workload, we can apply log replay to implement the real application workload. As for non-periodic workload, we can use benchmark to simulate a workload that highly similar to real application workload.

### Q8. How to build BestConfig project from source?

It's easy for users to build BestConfig project from source. You only need to import the whole Bestconfig project into eclipse and then build it.

### Q9. How to use BestConfig?

You can use BestConfig by the following steps.

Step 1. Download the latest release of BestConfig and then unzip it. After that, enter the directory of bestconf-master/deploy.

Step 2. Set up a system for tuning. In the project, we offer deployable examples for 6 systems, including Spark, Hive+Hadoop, Cassandra, MySQL, and Tomcat. We also specify the workload generators to be used for tuning the systems.

The detailed steps for setting up BestConfig for you own systems are presented in QuickStart.

Step 3. Run BestConfig.

As for linux system, firstly, you need to update all system and deployment related scripts accordingly and move them to the correct path on the servers. Next, you need to move the system-specific jar file to lib. (For example, move deploy/4BI/bestconfBI.jar to deploy/lib). At last, you enter the directory of deploy and run the shell of "start.sh".

### Q10. Why did I encounter a system exception of “java.lang.ClassNotFoundException” when I tried to run BestConfig for tomcat system?

Firstly, you need to implement the ConfigReadin and ConfigWrite interfaces as TomcatConfigReadin and TomcatConfigWrite.

Next, you need to compile TomcatConfigReadin and TomcatconfigWrite to bytecodes. After that, the location(path) of compiled bytecodes need to be added to classpath of BestConfig project.

### Q11. How can I get the detailed information of of BestConfig’s implementation mechanism?

You can read the following two papers to get all the information that you need.

[1] Yuqing ZHU, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, Yingchun Yang. BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning. Proceedings of the ACM Symposium on Cloud Computing 2017 (SoCC’17) [[pdf\\_Socc2017](#)] [[slides\\_Socc2017](#)]

[2] Yuqing ZHU, Jianxun Liu, Mengying Guo, Yungang Bao. ACTS in Need: Automatic Configuration Tuning with Scalability. Proceedings of the 8th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys’17) [[pdf\\_APSys2017](#)]

## 1.3 Use cases

### 1.3.1 BestConfig for Hadoop + Hive

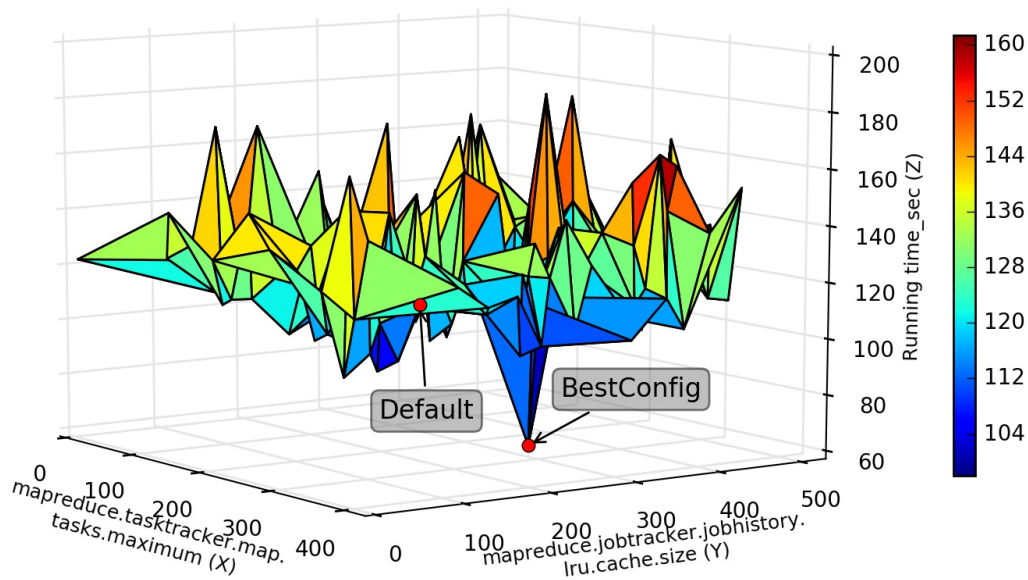
#### Experimental Settings

We executed Bestconfig for the Hadoop cluster with 4 nodes. The Hadoop cluster consists of 1 master node and 3 slave nodes. All nodes used in our experiment are shown below.

| Node    | OS     | CPU                                     | Memory |
|---------|--------|-----------------------------------------|--------|
| Master  | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |
| Slave 1 | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |
| Slave 2 | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |
| Slave 3 | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |

#### Performance Surface

We use [HiBench](#) that is a widely adopted benchmark tools in the workload generator for Spark to generate the target workload. Figure 1 plot the highly differed performance surfaces for Hadoop+Hive Join workload.



## Test Results

The test results of Hadoop under Join workload [hadoopJoin.arff](#).

The test results of Hadoop under Pagerank workload [hadoopPageRank.arff](#).

The test results of Hadoop under Join workload with 500 samples [join-trainingBestConf.arff](#) and [join-BestConfig.arff](#).

## Interface Impl

The source files of [HadoopConfigReadin](#) and [HadoopConfigWrite](#) implement the interfaces of [ConfigReadin](#) and [ConfigWrite](#) respectively.

## Download

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

## 1.3.2 BestConfig for Spark

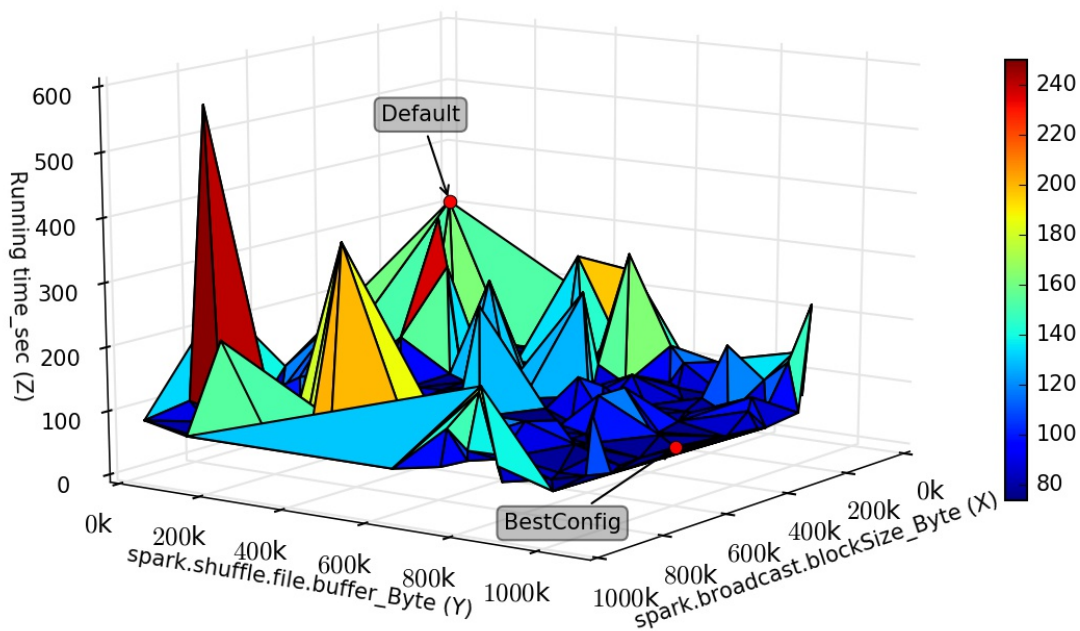
### Experimental Settings

We executed Bestconfig for the spark cluster with 4 nodes. The spark cluster consists of 1 master node and 3 slave nodes. All nodes used in our experiment are shown below.

| Node    | OS     | CPU                                     | Memory |
|---------|--------|-----------------------------------------|--------|
| Master  | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |
| Slave 1 | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |
| Slave 2 | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |
| Slave 3 | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |

## Performance Surface

We use [HiBench](#) that is a widely adopted benchmark tools in the workload generator for Spark to generate the target workload. Figure 1 plot the highly differed performance surfaces for Spark Pagerank workload.



## Test Results

The test result of Spark pagerank workload [pagerank](#). The test result of Spark kmeans workload [kmeans](#).

## Interface Impl

The source files of [SparkConfigReadin](#) and [SparkConfigWrite](#) implement the interfaces of [ConfigReadin](#) and [ConfigWrite](#) respectively.

## Download

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

### 1.3.3 BestConfig for Cassandra

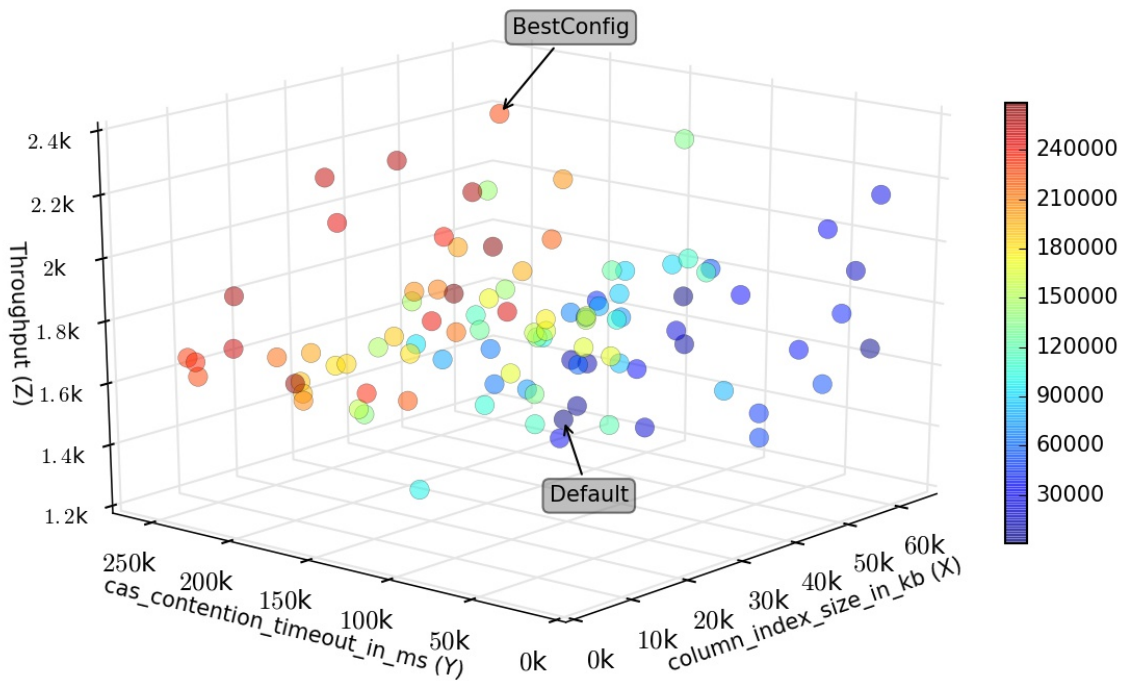
#### Experimental Settings

We executed Bestconfig for the spark cluster with 4 nodes. The spark cluster consists of 1 master node and 3 slave nodes. All nodes used in our experiment are shown below.

| Node        | OS     | CPU                                     | Memory |
|-------------|--------|-----------------------------------------|--------|
| Cassandra 1 | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |
| Cassandra 2 | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |
| Cassandra 3 | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |
| YCSB        | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |

#### Performance Surface

We use [YCSB](#) that is a widely adopted benchmark tools in the workload generator for Cassandra to generate the target workload. Currently, the workload adopted in our test is workoad, and we set recorecount to 17000000 and operationcount to 720000. Figure 1 is the scatter plot of performance for Cassandra under YCSB workload.



#### Test Results

The test result of Cassandra under YCSB workload is [cassandraYcsba.arff](#).

## Interface Impl

The source files of `CassandraConfigReadin` and `CassandraConfigWrite` implement the interfaces of `ConfigReadin` and `ConfigWrite` respectively.

## Download

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

## 1.3.4 BestConfig for MySQL

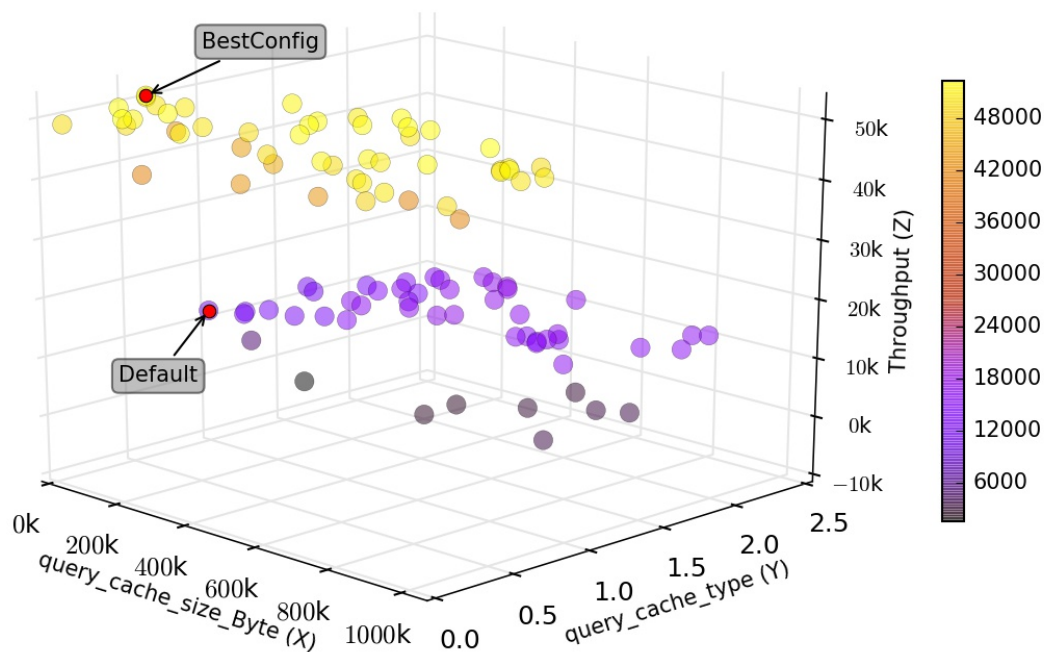
### Experimental Settings

We executed Bestconfig for the MySQL system, and we applied sysbench to test the performance of MySQL. All nodes used in our experiment are shown below.

| Node     | OS     | CPU                                     | Memory |
|----------|--------|-----------------------------------------|--------|
| MySQL    | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |
| Sysbench | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |

### Performance Surface

We use `Sysbench` that is a widely adopted benchmark tools in the workload generator for MySQL to generate the target workload. Currently, the test type in our experiment is `oltp` and the test mode is `simple`, and we set `num-threads` to 16, `oltp-table-size` to 10000000, and `max-time` to 300. Figure 1 is the scatter plot of performance for MySQL under OLTP simple test mode.



### Test Results

The result of MySQL under the zipfian read-write workload [MySQL\\_zipfian\\_readwrite.arff](#). The result of MySQL under OLTP simple test mode [MySQL\\_OLTP\\_simple.arff](#).

### Interface Impl

The source files of [MySQLConfigReadin](#) and [MySQLConfigWrite](#) implement the interfaces of [ConfigReadin](#) and [ConfigWrite](#) respectively.

### Download

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

## 1.3.5 BestConfig for Tomcat Server

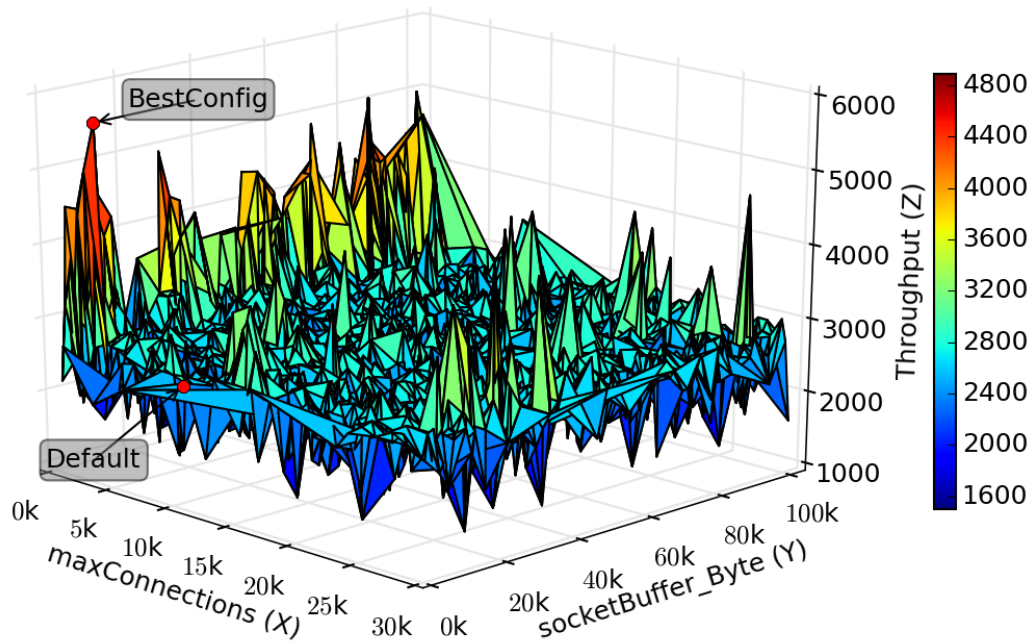
### Experimental Settings

We executed Bestconfig for the Tomcat server, and we applied sysbench to test the performance of Tomcat server. All nodes used in our experiment are shown below.

| Node          | OS     | CPU                                     | Memory |
|---------------|--------|-----------------------------------------|--------|
| Tomcat Server | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |
| JMeter        | CentOS | 16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 32GB   |

### Performance Surface

We use [JMeter](#) that is a widely adopted benchmark tools in the workload generator for Tomcat to generate the target workload.



## Test Results

All the test results of Tomcat under different workloads -> [Tomcat\\_Results](#).

## Script files

Script files for Tomcat node Scripts files for JMeter node

## Interface Impl

The source files of `TomcatConfigReadin` and `TomcatConfigWrite` implement the interfaces of `ConfigReadin` and `ConfigWrite` respectively.

## Download

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



## CHAPTER 2

---

### Citing BestConfig

---

Please cite:

```
@inproceedings{Zhu:2017:BTP:3127479.3128605,  
  author = {Zhu, Yuqing and Liu, Jianxun and Guo, Mengying and Bao, Yungang and  
    Ma, Wenlong and Liu, Zhuoyue and Song, Kunpeng and Yang, Yingchun},  
  title = {BestConfig: Tapping the Performance Potential of Systems via Automatic  
    Configuration Tuning},  
  booktitle = {Proceedings of the 2017 Symposium on Cloud Computing},  
  series = {SoCC '17},  
  year = {2017},  
  isbn = {978-1-4503-5028-0},  
  location = {Santa Clara, California},  
  pages = {338--350},  
  numpages = {13},  
  url = {http://doi.acm.org/10.1145/3127479.3128605},  
  acmid = {3128605},  
  publisher = {ACM},  
  address = {New York, NY, USA},  
  keywords = {ACT, automatic configuration tuning, performance optimization},  
}
```