sealuzh / jmh Private

forked from stewue/jmh

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base: 1.21-release ← compare: 1.21-reconfigure ✓ Able to merge. These branches can be automatically merged.

Create pull request Discuss and review the changes in this comparison with others.

Commits 42 | Files changed 85 | Commit comments 0 | 3 contributors

Showing 85 changed files with 2,677 additions and 236 deletions. Unified Split

16 .idea/runConfigurations/JMHSample_00_Reconfigure.xml

```
@@ -0,0 +1,16 @@
+ <component name="ProjectRunConfigurationManager">
+   <configuration default="false" name="JMHSample_00_Reconfigure" type="Application" factoryName="Application" nameIsGe
+     <option name="MAIN_CLASS_NAME" value="org.openjdk.jmh.samples.JMHSample_00_Reconfigure" />
+     <module name="jmh-samples" />
+     <extension name="coverage">
+       <pattern>
+         <option name="PATTERN" value="org.openjdk.jmh.samples.*" />
+         <option name="ENABLED" value="true" />
+       </pattern>
+     </extension>
+     <method v="2">
+       <option name="Maven.BeforeRunTask" enabled="true" file="$PROJECT_DIR$/jmh-core/pom.xml" goal="install -DskipTest
+       <option name="Make" enabled="true" />
+     </method>
+   </configuration>
+ </component>
```

15 .idea/runConfigurations/Remote.xml

```
@@ -0,0 +1,15 @@
+ <component name="ProjectRunConfigurationManager">
+   <configuration default="false" name="Remote" type="Remote">
+     <option name="USE_SOCKET_TRANSPORT" value="true" />
+     <option name="SERVER_MODE" value="false" />
+     <option name="SHMEM_ADDRESS" />
+     <option name="HOST" value="localhost" />
+     <option name="PORT" value="5005" />
+     <option name="AUTO_RESTART" value="false" />
+     <RunnerSettings RunnerId="Debug">
+       <option name="DEBUG_PORT" value="5005" />
+       <option name="LOCAL" value="false" />
+     </RunnerSettings>
+     <method v="2" />
+   </configuration>
+ </component>
```

3 README.md

```
@@ -0,0 +1,3 @@
+ # JMH Fork to support dynamic reconfiguration
+
+ The execution configuration is dynamically determined with different stoppage criteria.
```

2 jmh-archetypes/jmh-groovy-benchmark-archetype/pom.xml

```
5   5       <parent>
6   6           <groupId>org.openjdk.jmh</groupId>
7   7           <artifactId>jmh-archetypes</artifactId>
8   -           <version>1.21</version>
    8   +       <version>1.21-Reconfigure</version>
9   9       </parent>
10  10
11  11      <artifactId>jmh-groovy-benchmark-archetype</artifactId>
```

2 jmh-archetypes/jmh-java-benchmark-archetype/pom.xml

```
30  30      <parent>
31  31          <groupId>org.openjdk.jmh</groupId>
32  32          <artifactId>jmh-archetypes</artifactId>
33  -           <version>1.21</version>
    33  +       <version>1.21-Reconfigure</version>
34  34      </parent>
35  35
36  36      <artifactId>jmh-java-benchmark-archetype</artifactId>
```

2 jmh-archetypes/jmh-kotlin-benchmark-archetype/pom.xml

```
5   5       <parent>
6   6           <groupId>org.openjdk.jmh</groupId>
7   7           <artifactId>jmh-archetypes</artifactId>
8   -           <version>1.21</version>
    8   +       <version>1.21-Reconfigure</version>
9   9       </parent>
10  10
11  11      <artifactId>jmh-kotlin-benchmark-archetype</artifactId>
```

2 jmh-archetypes/jmh-scala-benchmark-archetype/pom.xml

```
5   5       <parent>
6   6           <groupId>org.openjdk.jmh</groupId>
7   7           <artifactId>jmh-archetypes</artifactId>
8   -           <version>1.21</version>
    8   +       <version>1.21-Reconfigure</version>
9   9       </parent>
10  10
11  11      <artifactId>jmh-scala-benchmark-archetype</artifactId>
```

2 jmh-archetypes/pom.xml

```
30  30      <parent>
31  31          <groupId>org.openjdk.jmh</groupId>
32  32          <artifactId>jmh-parent</artifactId>
33  -           <version>1.21</version>
    33  +       <version>1.21-Reconfigure</version>
34  34      </parent>
35  35
36  36      <name>JMH Archetypes</name>
```

2 jmh-core-benchmarks/pom.xml

```
30  30      <parent>
31  31          <groupId>org.openjdk.jmh</groupId>
32  32          <artifactId>jmh-parent</artifactId>
33  -           <version>1.21</version>
    33  +       <version>1.21-Reconfigure</version>
34  34      </parent>
35  35
36  36      <name>JMH Core Benchmarks</name>
```

2 jmh-core-benchmarks/src/main/java/org/openjdk/jmh/validation/IterationScoresFormatter.java

```
71  71      }
72  72
73  73      @Override
74  -       public void endRun(Collection<RunResult> result) {
```

```
+     public void endRun(Collection<RunResult> result, boolean atLeastOneWarning) {

      }

```

2 jmh-core-ct/pom.xml

```
      <parent>
          <groupId>org.openjdk.jmh</groupId>
          <artifactId>jmh-parent</artifactId>
-         <version>1.21</version>
+         <version>1.21-Reconfigure</version>
      </parent>

      <name>JMH Core Compilation Tests</name>
```

2 jmh-core-it/pom.xml

```
      <parent>
          <groupId>org.openjdk.jmh</groupId>
          <artifactId>jmh-parent</artifactId>
-         <version>1.21</version>
+         <version>1.21-Reconfigure</version>
      </parent>

      <name>JMH Core Integration Tests</name>
```

75 jmh-core/pom.xml

```
      <parent>
          <groupId>org.openjdk.jmh</groupId>
          <artifactId>jmh-parent</artifactId>
-         <version>1.21</version>
+         <version>1.21-Reconfigure</version>
      </parent>

      <name>JMH Core</name>
              <groupId>org.apache.commons</groupId>
              <artifactId>commons-math3</artifactId>
          </dependency>
+         <dependency>
+             <groupId>commons-io</groupId>
+             <artifactId>commons-io</artifactId>
+             <version>2.6</version>
+         </dependency>
+         <dependency>
+             <groupId>org.apache.commons</groupId>
+             <artifactId>commons-lang3</artifactId>
+             <version>3.8.1</version>
+         </dependency>
+         <dependency>
+             <groupId>com.github.haifengl</groupId>
+             <artifactId>smile-math</artifactId>
+             <version>1.5.3</version>
+         </dependency>
      </dependencies>

      <properties>
                  <artifactId>maven-compiler-plugin</artifactId>
                  <configuration>
                      <compilerVersion>1.7</compilerVersion>
-                     <source>1.7</source>
-                     <target>1.7</target>
+                     <source>8</source>
+                     <target>8</target>
                      <compilerArgument>-proc:none</compilerArgument>
                  </configuration>
              </plugin>
                  </configuration>
              </plugin>
```

```
-            <plugin>
-                <groupId>com.mycila.maven-license-plugin</groupId>
-                <artifactId>maven-license-plugin</artifactId>
-                <executions>
-                    <execution>
-                        <goals>
-                            <goal>format</goal>
-                        </goals>
-                        <phase>process-sources</phase>
-                        <configuration>
-                            <header>file:///${project.basedir}/../src/license/gpl_cpe/header.txt</header>
-                            <skipExistingHeaders>true</skipExistingHeaders>
-                            <strictCheck>true</strictCheck>
-                            <mapping>
-                                <java>PHP</java>
-                            </mapping>
-                        </configuration>
-                    </execution>
-                </executions>
-            </plugin>
+ <!--            <plugin>-->
+ <!--                <groupId>com.mycila.maven-license-plugin</groupId>-->
+ <!--                <artifactId>maven-license-plugin</artifactId>-->
+ <!--                <executions>-->
+ <!--                    <execution>-->
+ <!--                        <goals>-->
+ <!--                            <goal>format</goal>-->
+ <!--                        </goals>-->
+ <!--                        <phase>process-sources</phase>-->
+ <!--                        <configuration>-->
+ <!--                            <header>file:///${project.basedir}/../src/license/gpl_cpe/header.txt</header>-->
+ <!--                            <skipExistingHeaders>true</skipExistingHeaders>-->
+ <!--                            <strictCheck>true</strictCheck>-->
+ <!--                            <mapping>-->
+ <!--                                <java>PHP</java>-->
+ <!--                            </mapping>-->
+ <!--                        </configuration>-->
+ <!--                    </execution>-->
+ <!--                </executions>-->
+ <!--            </plugin>-->

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
            <resource>
                <directory>src/main/resources/</directory>
                <filtering>true</filtering>
+               <excludes>
+                   <exclude>pa_linux_amd64</exclude>
+                   <exclude>pa_darwin_amd64</exclude>
+                   <exclude>pa_windows_amd64.exe</exclude>
+               </excludes>
+           </resource>
+           <resource>
+               <directory>src/main/resources/</directory>
+               <filtering>false</filtering>
+               <includes>
+                   <include>pa_linux_amd64</include>
+                   <include>pa_darwin_amd64</include>
+                   <include>pa_windows_amd64.exe</include>
+               </includes>
            </resource>
        </resources>
    </build>
```

48 jmh-core/src/main/java/org/openjdk/jmh/annotations/Fork.java

```
 */
package org.openjdk.jmh.annotations;

```

```
27      - import java.lang.annotation.ElementType;
28      - import java.lang.annotation.Inherited;
29      - import java.lang.annotation.Retention;
30      - import java.lang.annotation.RetentionPolicy;
31      - import java.lang.annotation.Target;
    27  + import java.lang.annotation.*;
32  28
33  29    /**
34  30     * <b>Fork annotation allows to set the default forking parameters for the benchmark.</b>
39  35     * the runtime options.</p>
40  36     */
41  37    @Inherited
42      - @Target({ElementType.METHOD,ElementType.TYPE})
    38  + @Target({ElementType.METHOD, ElementType.TYPE})
43  39    @Retention(RetentionPolicy.RUNTIME)
44  40    public @interface Fork {
45  41
46  42        int BLANK_FORKS = -1;
47  43
48  44        String BLANK_ARGS = "blank_blank_blank_2014";
49  45
50      -     /** @return number of times harness should fork, zero means "no fork" */
    46  +     /**
    47  +      * @return number of times harness should fork, zero means "no fork"
    48  +      */
51  49        int value() default BLANK_FORKS;
52  50
53      -     /** @return number of times harness should fork and ignore the results */
    51  +     /**
    52  +      * @return minimum number of times harness should fork
    53  +      */
    54  +     int minValue() default BLANK_FORKS;
    55  +
    56  +     /**
    57  +      * @return number of times harness should fork and ignore the results
    58  +      */
54  59        int warmups() default BLANK_FORKS;
55  60
56      -     /** @return JVM executable to run with */
    61  +     /**
    62  +      * @return minimum number of times harness should fork and ignore the results
    63  +      */
    64  +     int minWarmups() default BLANK_FORKS;
    65  +
    66  +     /**
    67  +      * @return JVM executable to run with
    68  +      */
57  69        String jvm() default BLANK_ARGS;
58  70
59      -     /** @return JVM arguments to replace in the command line */
60      -     String[] jvmArgs() default { BLANK_ARGS };
    71  +     /**
    72  +      * @return JVM arguments to replace in the command line
    73  +      */
    74  +     String[] jvmArgs() default {BLANK_ARGS};
61  75
62      -     /** @return JVM arguments to prepend in the command line */
63      -     String[] jvmArgsPrepend() default { BLANK_ARGS };
    76  +     /**
    77  +      * @return JVM arguments to prepend in the command line
    78  +      */
    79  +     String[] jvmArgsPrepend() default {BLANK_ARGS};
64  80
65      -     /** @return JVM arguments to append in the command line */
66      -     String[] jvmArgsAppend() default { BLANK_ARGS };
    81  +     /**
    82  +      * @return JVM arguments to append in the command line
    83  +      */
    84  +     String[] jvmArgsAppend() default {BLANK_ARGS};
67  85
```

```
68  86   }
```

24 jmh-core/src/main/java/org/openjdk/jmh/annotations/Measurement.java

```
24  24    */
25  25   package org.openjdk.jmh.annotations;
26  26
27      - import java.lang.annotation.ElementType;
28      - import java.lang.annotation.Inherited;
29      - import java.lang.annotation.Retention;
30      - import java.lang.annotation.RetentionPolicy;
31      - import java.lang.annotation.Target;
    27  + import java.lang.annotation.*;
32  28   import java.util.concurrent.TimeUnit;
33  29
34  30   /**
43  39    * @see Warmup
44  40    */
45  41   @Inherited
46      - @Target({ElementType.METHOD,ElementType.TYPE})
    42  + @Target({ElementType.METHOD, ElementType.TYPE})
47  43   @Retention(RetentionPolicy.RUNTIME)
48  44   public @interface Measurement {
49  45
50  46       int BLANK_ITERATIONS = -1;
51  47       int BLANK_TIME = -1;
52  48       int BLANK_BATCHSIZE = -1;
53  49
54      -     /** @return Number of measurement iterations */
    50  +     /**
    51  +      * @return Number of measurement iterations
    52  +      */
55  53       int iterations() default BLANK_ITERATIONS;
56  54
57      -     /** @return Time of each measurement iteration */
    55  +     /**
    56  +      * @return Time of each measurement iteration
    57  +      */
58  58       int time() default BLANK_TIME;
59  59
60      -     /** @return Time unit for measurement iteration duration */
    60  +     /**
    61  +      * @return Time unit for measurement iteration duration
    62  +      */
61  63       TimeUnit timeUnit() default TimeUnit.SECONDS;
62  64
63      -     /** @return Batch size: number of benchmark method calls per operation */
    65  +     /**
    66  +      * @return Batch size: number of benchmark method calls per operation
    67  +      */
64  68       int batchSize() default BLANK_BATCHSIZE;
65  69
66  70   }
```

5 jmh-core/src/main/java/org/openjdk/jmh/annotations/Mode.java

```
61  61        */
62  62       SampleTime("sample", "Sampling time"),
63  63
    64  +     /**
    65  +      * <p>Dynamic reconfigure: runs the {@link #SampleTime} mode but dynamically decide then to stop the data collecti
    66  +      */
    67  +     Reconfigure("re", "Dynamic reconfigure"),
    68  +
64  69       /**
65  70        * <p>Single shot time: measures the time for a single operation.</p>
66  71        *
```

40 jmh-core/src/main/java/org/openjdk/jmh/annotations/Reconfigure.java

```
@@ -0,0 +1,40 @@
+ package org.openjdk.jmh.annotations;
+
+ import java.lang.annotation.*;
+
+ /**
+  * <b>Reconfigure annotation allows to set the default reconfigure thresholds for the benchmark.</b>
+  *
+  * <p>This annotation may be put at {@link Benchmark} method to have effect on that
+  * method only, or at the enclosing class instance to have the effect over all
+  * {@link Benchmark} methods in the class. This annotation may be overridden with
+  * the runtime options.</p>
+  */
+ @Inherited
+ @Target({ElementType.METHOD, ElementType.TYPE})
+ @Retention(RetentionPolicy.RUNTIME)
+ public @interface Reconfigure {
+
+     double BLANK_THRESHOLD = -1;
+
+     /**
+      * @return reconfigure mode
+      * @see ReconfigureMode
+      */
+     ReconfigureMode value() default ReconfigureMode.NONE;
+
+     /**
+      * @return coefficient of variation variability threshold
+      */
+     double covThreshold() default BLANK_THRESHOLD;
+
+     /**
+      * @return confidence interval variability threshold
+      */
+     double ciThreshold() default BLANK_THRESHOLD;
+
+     /**
+      * @return p value of kullback leibler divergence as variability threshold
+      */
+     double kldThreshold() default BLANK_THRESHOLD;
+ }
```

90 jmh-core/src/main/java/org/openjdk/jmh/annotations/ReconfigureMode.java

```
@@ -0,0 +1,90 @@
+ package org.openjdk.jmh.annotations;
+
+ import java.util.ArrayList;
+ import java.util.List;
+
+ /**
+  * Reconfigure mode.
+  */
+ public enum ReconfigureMode {
+
+     /**
+      * <p>Internal mode </p>
+      */
+     NONE("none", "none"),
+
+     /**
+      * <p>coefficient of variation as variability criteria</p>
+      */
+     COV("cov", "coefficient of variation"),
+
+     /**
+      * <p>confidence interval as variability criteria</p>
+      */
+     CI("ci", "confidence interval"),
```

```
+
+     /**
+      * <p>kullback leibler divergence as variability criteria</p>
+      */
+     DIVERGENCE("kld", "kullback leibler divergence"),
+
+     ;
+
+     private final String shortLabel;
+     private final String longLabel;
+
+     ReconfigureMode(String shortLabel, String longLabel) {
+         this.shortLabel = shortLabel;
+         this.longLabel = longLabel;
+     }
+
+     public String shortLabel() {
+         return shortLabel;
+     }
+
+     public String longLabel() {
+         return longLabel;
+     }
+
+     public static ReconfigureMode deepValueOf(String name) {
+         try {
+             return ReconfigureMode.valueOf(name);
+         } catch (IllegalArgumentException iae) {
+             ReconfigureMode inferred = null;
+             for (ReconfigureMode type : options()) {
+                 if (type.shortLabel().startsWith(name)) {
+                     if (inferred == null) {
+                         inferred = type;
+                     } else {
+                         throw new IllegalStateException("Unable to parse reconfigure mode, ambiguous prefix given: \""
+                                 "Known values are " + getKnown());
+                     }
+                 }
+             }
+             if (inferred != null) {
+                 return inferred;
+             } else {
+                 throw new IllegalStateException("Unable to parse reconfigure mode: \"" + name + "\"\n" +
+                         "Known values are " + getKnown());
+             }
+         }
+     }
+
+     public static List<String> getKnown() {
+         List<String> res = new ArrayList<>();
+         for (ReconfigureMode type : ReconfigureMode.options()) {
+             res.add(type.name() + "/" + type.shortLabel());
+         }
+         return res;
+     }
+
+     public static ReconfigureMode[] options() {
+         return new ReconfigureMode[]{
+                 COV, CI, DIVERGENCE
+         };
+     }
+
+     public boolean isNone() {
+         return equals(ReconfigureMode.NONE);
+     }
+ }
```

29 jmh-core/src/main/java/org/openjdk/jmh/annotations/Warmup.java

```
24  24     */
```

```
 package org.openjdk.jmh.annotations;

- import java.lang.annotation.ElementType;
- import java.lang.annotation.Inherited;
- import java.lang.annotation.Retention;
- import java.lang.annotation.RetentionPolicy;
- import java.lang.annotation.Target;
+ import java.lang.annotation.*;
 import java.util.concurrent.TimeUnit;

 /**
  *
  * @see Measurement
  */
- @Target({ElementType.METHOD,ElementType.TYPE})
+ @Target({ElementType.METHOD, ElementType.TYPE})
 @Retention(RetentionPolicy.RUNTIME)
 @Inherited
 public @interface Warmup {
     int BLANK_TIME = -1;
     int BLANK_BATCHSIZE = -1;

-    /** @return Number of warmup iterations */
+    /**
+     * @return Number of warmup iterations
+     */
     int iterations() default BLANK_ITERATIONS;

-    /** @return Time for each warmup iteration */
+    /**
+     * @return Minimum number of warmup iterations
+     */
+    int minIterations() default BLANK_ITERATIONS;
+
+    /**
+     * @return Time for each warmup iteration
+     */
     int time() default BLANK_TIME;

-    /** @return Time unit for warmup iteration duration */
+    /**
+     * @return Time unit for warmup iteration duration
+     */
     TimeUnit timeUnit() default TimeUnit.SECONDS;

-    /** @return batch size: number of benchmark method calls per operation */
+    /**
+     * @return batch size: number of benchmark method calls per operation
+     */
     int batchSize() default BLANK_BATCHSIZE;

 }
```

8 jmh-core/src/main/java/org/openjdk/jmh/generators/core/BenchmarkGenerator.java

```
                                group.getGroupThreads(),
                                group.getGroupLabels(),
                                group.getWarmupIterations(),
+                               group.getMinWarmupIterations(),
                                group.getWarmupTime(),
                                group.getWarmupBatchSize(),
                                group.getMeasurementIterations(),
                                group.getMeasurementTime(),
                                group.getMeasurementBatchSize(),
                                group.getForks(),
+                               group.getMinForks(),
                                group.getWarmupForks(),
+                               group.getMinWarmupForks(),
+                               group.getReconfigureMode(),
+                               group.getReconfigureCovThreshold(),
```

```
+                                   group.getReconfigureCiThreshold(),
+                                   group.getReconfigureKldThreshold(),
                                    group.getJvm(),
                                    group.getJvmArgs(),
                                    group.getJvmArgsPrepend(),
                        generateAverageTime(writer, benchmarkKind, methodGroup, states);
                        break;
                    case SampleTime:
+                   case Reconfigure:
                        generateSampleTime(writer, benchmarkKind, methodGroup, states);
                        break;
                    case SingleShotTime:
```

62 jmh-core/src/main/java/org/openjdk/jmh/generators/core/MethodGroup.java

```
  package org.openjdk.jmh.generators.core;

  import org.openjdk.jmh.annotations.*;
+ import org.openjdk.jmh.runner.Defaults;
  import org.openjdk.jmh.runner.options.TimeValue;
  import org.openjdk.jmh.util.Optional;

        return Optional.none();
    }

+   public Optional<Integer> getMinWarmupIterations() {
+       for (Warmup ann : getAll(Warmup.class)) {
+           if (ann.minIterations() != Warmup.BLANK_ITERATIONS) {
+               return Optional.of(ann.minIterations());
+           }
+       }
+       return Optional.none();
+   }
+
    public Optional<TimeValue> getWarmupTime() {
        for (Warmup ann : getAll(Warmup.class)) {
            if (ann.time() != Warmup.BLANK_TIME) {
        return Optional.none();
    }

+   public Optional<Integer> getMinForks() {
+       for (Fork ann : getAll(Fork.class)) {
+           if (ann.minValue() != Fork.BLANK_FORKS) {
+               return Optional.of(ann.minValue());
+           }
+       }
+       return Optional.none();
+   }
+
    public Optional<Integer> getWarmupForks() {
        for (Fork ann : getAll(Fork.class)) {
            if (ann.warmups() != Fork.BLANK_FORKS) {
        return Optional.none();
    }

+   public Optional<Integer> getMinWarmupForks() {
+       for (Fork ann : getAll(Fork.class)) {
+           if (ann.minWarmups() != Fork.BLANK_FORKS) {
+               return Optional.of(ann.minWarmups());
+           }
+       }
+       return Optional.none();
+   }
+
+   public ReconfigureMode getReconfigureMode() {
+       for (Reconfigure ann : getAll(Reconfigure.class)) {
+           return ann.value();
+       }
+       return Defaults.RECONFIGURE_MODE;
+   }
```

```
+
+     public Optional<Double> getReconfigureCovThreshold() {
+         for (Reconfigure ann : getAll(Reconfigure.class)) {
+             if (ann.covThreshold() != Reconfigure.BLANK_THRESHOLD) {
+                 return Optional.of(ann.covThreshold());
+             }
+         }
+         return Optional.none();
+     }
+
+     public Optional<Double> getReconfigureCiThreshold() {
+         for (Reconfigure ann : getAll(Reconfigure.class)) {
+             if (ann.ciThreshold() != Reconfigure.BLANK_THRESHOLD) {
+                 return Optional.of(ann.ciThreshold());
+             }
+         }
+         return Optional.none();
+     }
+
+     public Optional<Double> getReconfigureKldThreshold() {
+         for (Reconfigure ann : getAll(Reconfigure.class)) {
+             if (ann.kldThreshold() != Reconfigure.BLANK_THRESHOLD) {
+                 return Optional.of(ann.kldThreshold());
+             }
+         }
+         return Optional.none();
+     }
+
      public Optional<String> getJvm() {
          for (Fork ann : getAll(Fork.class)) {
              if (!ann.jvm().equals(Fork.BLANK_ARGS)) {
```

103 jmh-core/src/main/java/org/openjdk/jmh/infra/BenchmarkParams.java

```
  package org.openjdk.jmh.infra;

  import org.openjdk.jmh.annotations.Mode;
+ import org.openjdk.jmh.annotations.ReconfigureMode;
  import org.openjdk.jmh.runner.WorkloadParams;
  import org.openjdk.jmh.runner.options.TimeValue;
  import org.openjdk.jmh.util.Utils;

      public BenchmarkParams(String benchmark, String generatedTarget, boolean synchIterations,
                             int threads, int[] threadGroups, Collection<String> threadGroupLabels,
-                            int forks, int warmupForks,
+                            int forks, int minForks, int warmupForks, int minWarmupForks,
                             IterationParams warmup, IterationParams measurement,
-                            Mode mode, WorkloadParams params,
+                            Mode mode, ReconfigureMode reconfigureMode, double reconfigureCovThreshold,
+                            double reconfigureCiThreshold,  double reconfigureKldThreshold, WorkloadParams params,
                             TimeUnit timeUnit, int opsPerInvocation,
                             String jvm, Collection<String> jvmArgs,
                             String jdkVersion, String vmName, String vmVersion, String jmhVersion,
                             TimeValue timeout) {
          super(benchmark, generatedTarget, synchIterations,
                  threads, threadGroups, threadGroupLabels,
-                 forks, warmupForks,
+                 forks, minForks, warmupForks, minWarmupForks,
                  warmup, measurement,
-                 mode, params,
+                 mode, reconfigureMode, reconfigureCovThreshold,
+                 reconfigureCiThreshold,  reconfigureKldThreshold, params,
                  timeUnit, opsPerInvocation,
                  jvm, jvmArgs,
                  jdkVersion, vmName, vmVersion, jmhVersion,
      private int markerEnd;
      public BenchmarkParamsL4(String benchmark, String generatedTarget, boolean synchIterations,
                               int threads, int[] threadGroups, Collection<String> threadGroupLabels,
-                              int forks, int warmupForks,
+                              int forks, int minForks, int warmupForks, int minWarmupForks,
```

```
95  98                                  IterationParams warmup, IterationParams measurement,
96     -                                Mode mode, WorkloadParams params,
    99 +                                Mode mode, ReconfigureMode reconfigureMode, double reconfigureCovThreshold,
   100 +                                double reconfigureCiThreshold,  double reconfigureKldThreshold, WorkloadParams params,
97  101                                  TimeUnit timeUnit, int opsPerInvocation,
98  102                                  String jvm, Collection<String> jvmArgs,
99  103                                  String jdkVersion, String vmName, String vmVersion, String jmhVersion,
100 104                                  TimeValue timeout) {
101 105             super(benchmark, generatedTarget, synchIterations,
102 106                     threads, threadGroups, threadGroupLabels,
103    -                    forks, warmupForks,
   107 +                    forks, minForks, warmupForks, minWarmupForks,
104 108                     warmup, measurement,
105    -                    mode, params,
   109 +                    mode, reconfigureMode, reconfigureCovThreshold,
   110 +                    reconfigureCiThreshold,  reconfigureKldThreshold, params,
106 111                     timeUnit, opsPerInvocation,
107 112                     jvm, jvmArgs,
108 113                     jdkVersion, vmName, vmVersion, jmhVersion,
132 137
133 138         public BenchmarkParamsL3(String benchmark, String generatedTarget, boolean synchIterations,
134 139                                  int threads, int[] threadGroups, Collection<String> threadGroupLabels,
135    -                                int forks, int warmupForks,
   140 +                                int forks, int minForks, int warmupForks, int minWarmupForks,
136 141                                  IterationParams warmup, IterationParams measurement,
137    -                                Mode mode, WorkloadParams params,
   142 +                                Mode mode, ReconfigureMode reconfigureMode, double reconfigureCovThreshold,
   143 +                                double reconfigureCiThreshold,  double reconfigureKldThreshold, WorkloadParams params,
138 144                                  TimeUnit timeUnit, int opsPerInvocation,
139 145                                  String jvm, Collection<String> jvmArgs,
140 146                                  String jdkVersion, String vmName, String vmVersion, String jmhVersion,
141 147                                  TimeValue timeout) {
142 148             super(benchmark, generatedTarget, synchIterations,
143 149                     threads, threadGroups, threadGroupLabels,
144    -                    forks, warmupForks,
   150 +                    forks, minForks, warmupForks, minWarmupForks,
145 151                     warmup, measurement,
146    -                    mode, params,
   152 +                    mode, reconfigureMode, reconfigureCovThreshold,
   153 +                    reconfigureCiThreshold,  reconfigureKldThreshold, params,
147 154                     timeUnit, opsPerInvocation,
148 155                     jvm, jvmArgs,
149 156                     jdkVersion, vmName, vmVersion, jmhVersion,
184 191         protected final int[] threadGroups;
185 192         protected final Collection<String> threadGroupLabels;
186 193         protected final int forks;
   194 + ,      protected final int minForks;
187 195         protected final int warmupForks;
   196 + ,      protected final int minWarmupForks;
188 197         protected final IterationParams warmup;
189 198         protected final IterationParams measurement;
190 199         protected final Mode mode;
   200 + ,      protected final ReconfigureMode reconfigureMode;
   201 + ,      protected final double reconfigureCovThreshold;
   202 + ,      protected final double reconfigureCiThreshold;
   203 + ,      protected final double reconfigureKldThreshold;
191 204         protected final WorkloadParams params;
192 205         protected final TimeUnit timeUnit;
193 206         protected final int opsPerInvocation;
201 214
202 215         public BenchmarkParamsL2(String benchmark, String generatedTarget, boolean synchIterations,
203 216                                  int threads, int[] threadGroups, Collection<String> threadGroupLabels,
204    -                                int forks, int warmupForks,
   217 +                                int forks, int minForks, int warmupForks, int minWarmupForks,
205 218                                  IterationParams warmup, IterationParams measurement,
206    -                                Mode mode, WorkloadParams params,
   219 +                                Mode mode, ReconfigureMode reconfigureMode, double reconfigureCovThreshold,
   220 +                                double reconfigureCiThreshold,  double reconfigureKldThreshold, WorkloadParams params,
207 221                                  TimeUnit timeUnit, int opsPerInvocation,
208 222                                  String jvm, Collection<String> jvmArgs,
```

```
209 223                                 String jdkVersion, String vmName, String vmVersion, String jmhVersion,
215 229         this.threadGroups = threadGroups;
216 230         this.threadGroupLabels = threadGroupLabels;
217 231         this.forks = forks;
    232 +       this.minForks = minForks;
218 233         this.warmupForks = warmupForks;
    234 +       this.minWarmupForks = minWarmupForks;
219 235         this.warmup = warmup;
220 236         this.measurement = measurement;
221 237         this.mode = mode;
    238 +       this.reconfigureMode = reconfigureMode;
    239 +       this.reconfigureCovThreshold = reconfigureCovThreshold;
    240 +       this.reconfigureCiThreshold = reconfigureCiThreshold;
    241 +       this.reconfigureKldThreshold = reconfigureKldThreshold;
222 242         this.params = params;
223 243         this.timeUnit = timeUnit;
224 244         this.opsPerInvocation = opsPerInvocation;
289 309         return forks;
290 310     }
291 311
    312 +   /**
    313 +    * @return minimum number of forked VM runs, which we measure
    314 +    */
    315 +   public int getMinForks() {
    316 +       return minForks;
    317 +   }
    318 +
292 319     /**
293 320      * @return number of forked VM runs, which we discard from the result
294 321      */
295 322     public int getWarmupForks() {
296 323         return warmupForks;
297 324     }
298 325
    326 +   /**
    327 +    * @return minimum number of forked VM runs, which we discard from the result
    328 +    */
    329 +   public int getMinWarmupForks() {
    330 +       return minWarmupForks;
    331 +   }
    332 +
299 333     /**
300 334      * @return benchmark mode
301 335      */
302 336     public Mode getMode() {
303 337         return mode;
304 338     }
305 339
    340 +   /**
    341 +    * @return reconfigure mode
    342 +    */
    343 +   public ReconfigureMode getReconfigureMode() {
    344 +       return reconfigureMode;
    345 +   }
    346 +
    347 +   /**
    348 +    * @return coefficient of variation variability threshold
    349 +    */
    350 +   public double getReconfigureCovThreshold() {
    351 +       return reconfigureCovThreshold;
    352 +   }
    353 +
    354 +   /**
    355 +    * @return confidence interval variability threshold
    356 +    */
    357 +   public double getReconfigureCiThreshold() {
    358 +       return reconfigureCiThreshold;
    359 +   }
    360 +
    361 +   /**
```

```
+      * @return p value of kullback leibler divergence as variability threshold
+      */
+     public double getReconfigureKldThreshold() {
+         return reconfigureKldThreshold;
+     }
+
+     public double getReconfigureThreshold() {
+         switch (getReconfigureMode()) {
+             case CI:
+                 return getReconfigureCiThreshold();
+             case COV:
+                 return getReconfigureCovThreshold();
+             case DIVERGENCE:
+                 return getReconfigureKldThreshold();
+             default:
+                 throw new IllegalArgumentException("Reconfigure Mode is nod valid");
+         }
+     }
+
      /**
       * @return benchmark name
       */
```

32 jmh-core/src/main/java/org/openjdk/jmh/infra/IterationParams.java

```
          Utils.check(IterationParams.class, "type", "count", "timeValue", "batchSize");
      }

-     public IterationParams(IterationType type, int count, TimeValue time, int batchSize) {
-         super(type, count, time, batchSize);
+     public IterationParams(IterationType type, int count, int minCount, TimeValue time, int batchSize) {
+         super(type, count, minCount, time, batchSize);
      }
  }

  abstract class IterationParamsL4 extends IterationParamsL3 {
      private static final long serialVersionUID = 9079354621906758255L;

      private int markerEnd;
-     public IterationParamsL4(IterationType type, int count, TimeValue time, int batchSize) {
-         super(type, count, time, batchSize);
+     public IterationParamsL4(IterationType type, int count, int minCount, TimeValue time, int batchSize) {
+         super(type, count, minCount, time, batchSize);
      }
  }

      private boolean q161, q162, q163, q164, q165, q166, q167, q168;
      private boolean q171, q172, q173, q174, q175, q176, q177, q178;

-     public IterationParamsL3(IterationType type, int count, TimeValue time, int batchSize) {
-         super(type, count, time, batchSize);
+     public IterationParamsL3(IterationType type, int count, int minCount, TimeValue time, int batchSize) {
+         super(type, count, minCount, time, batchSize);
      }
  }

       */
      protected final int count;

+     /**
+      * minimum amount of iterations
+      */
+     protected final int minCount;
+
      /**
       * iteration runtime
       */
       */
      protected final int batchSize;
```

```
139      -     public IterationParamsL2(IterationType type, int count, TimeValue time, int batchSize) {
         144   +     public IterationParamsL2(IterationType type, int count, int minCount, TimeValue time, int batchSize) {
140      145           this.type = type;
141      146           this.count = count;
         147   +         this.minCount = minCount;
142      148           this.timeValue = time;
143      149           this.batchSize = batchSize;
144      150       }
159      165           return count;
160      166       }
161      167
         168   +     /**
         169   +      * Minimum number of iterations.
         170   +      * @return number of iterations of given type.
         171   +      */
         172   +     public int getMinCount() {
         173   +         return minCount;
         174   +     }
         175   +
162      176       /**
163      177        * Time for iteration.
164      178        * @return time
183      197           IterationParams that = (IterationParams) o;
184      198
185      199           if (count != that.count) return false;
         200   +         if (minCount != that.minCount) return false;
186      201           if (batchSize != that.batchSize) return false;
187      202           if (timeValue != null ? !timeValue.equals(that.timeValue) : that.timeValue != null) return false;
188      203
192      207       @Override
193      208       public int hashCode() {
194      209           int result = count;
         210   +         result = 31 * result + minCount;
195      211           result = 31 * result + batchSize;
196      212           result = 31 * result + (timeValue != null ? timeValue.hashCode() : 0);
197      213           return result;
198      214       }
199      215
200      216       @Override
201      217       public String toString() {
202      -         return "IterationParams("+ getCount()+", "+ getTime()+", "+ getBatchSize()+")";
         218   +         return "IterationParams("+ getCount()+", "+ getMinCount()+", "+ getTime()+", "+ getBatchSize()+")";
203      219       }
204      220
205      221   }
```

37 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/helper/BenchmarkMetaData.java

```
...  ...  @@ -0,0 +1,37 @@
         1   + package org.openjdk.jmh.reconfigure.helper;
         2   +
         3   + import org.apache.commons.math3.util.Pair;
         4   + import org.openjdk.jmh.infra.BenchmarkParams;
         5   + import org.openjdk.jmh.results.BenchmarkResult;
         6   + import org.openjdk.jmh.util.Multimap;
         7   +
         8   + import java.util.List;
         9   +
         10  + public class BenchmarkMetaData {
         11  +     private Multimap<BenchmarkParams, BenchmarkResult> results;
         12  +     private List<Double> warmupThresholds;
         13  +     private List<Double> measurementThresholds;
         14  +     private boolean atLeastOneWarning = false;
         15  +
         16  +     public BenchmarkMetaData() {
         17  +     }
         18  +
         19  +     public BenchmarkMetaData(Multimap<BenchmarkParams, BenchmarkResult> results, List<Double> warmupThresholds, List<D
         20  +         this.results = results;
         21  +         this.warmupThresholds = warmupThresholds;
```

```
+         this.measurementThresholds = measurementThresholds;
+         this.atLeastOneWarning = atLeastOneWarning;
+     }
+
+     public Multimap<BenchmarkParams, BenchmarkResult> getResults() {
+         return results;
+     }
+
+     public Pair<List<Double>, List<Double>> getThresholdsPair() {
+         return new Pair<List<Double>, List<Double>>(warmupThresholds, measurementThresholds);
+     }
+
+     public boolean hasAtLeastOneWarning() {
+         return atLeastOneWarning;
+     }
+ }
```

30 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/helper/HistogramHelper.java

```
@@ -0,0 +1,30 @@
+ package org.openjdk.jmh.reconfigure.helper;
+
+ import java.util.ArrayList;
+ import java.util.List;
+ import java.util.Map;
+
+ public class HistogramHelper {
+     public static List<HistogramItem> toList(Map<Integer, List<HistogramItem>> map) {
+         List<HistogramItem> list = new ArrayList<>();
+         for (Integer key : map.keySet()) {
+             list.addAll(map.get(key));
+         }
+
+         return list;
+     }
+
+     public static List<Double> toArray(List<HistogramItem> input) {
+         List<Double> out = new ArrayList<>();
+
+         for (int i = 0; i < input.size(); i++) {
+             HistogramItem item = input.get(i);
+
+             for (int j = 0; j < item.getCount(); j++) {
+                 out.add(item.getValue());
+             }
+         }
+
+         return out;
+     }
+ }
```

45 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/helper/HistogramItem.java

```
@@ -0,0 +1,45 @@
+ package org.openjdk.jmh.reconfigure.helper;
+
+ public class HistogramItem {
+     private final int fork;
+     private final int iteration;
+     private final double value;
+     private final long count;
+
+     public HistogramItem(int fork, int iteration, double value, long count) {
+         this.fork = fork;
+         this.iteration = iteration;
+         this.value = value;
+         this.count = count;
+     }
+
```

```
+     public int getFork() {
+         return fork;
+     }
+
+     public int getIteration() {
+         return iteration;
+     }
+
+     public double getValue() {
+         return value;
+     }
+
+     public long getCount() {
+         return count;
+     }
+
+     public HistogramItem single() {
+         return new HistogramItem(fork, iteration, value, 1);
+     }
+
+     @Override
+     public String toString() {
+         return "HistogramItem{" +
+                 "fork=" + fork +
+                 ", iteration=" + iteration +
+                 ", value=" + value +
+                 ", count=" + count +
+                 '}';
+     }
+ }
```

17 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/helper/ListToArray.java

```
@@ -0,0 +1,17 @@
+ package org.openjdk.jmh.reconfigure.helper;
+
+ import java.util.List;
+
+ public class ListToArray {
+     private static double[] toPrimitive(Double[] array) {
+         final double[] result = new double[array.length];
+         for (int i = 0; i < array.length; i++) {
+             result[i] = array[i];
+         }
+         return result;
+     }
+
+     public static double[] toPrimitive(List<Double> input) {
+         return toPrimitive(input.toArray(new Double[0]));
+     }
+ }
```

58 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/helper/OutlierDetector.java

```
@@ -0,0 +1,58 @@
+ package org.openjdk.jmh.reconfigure.helper;
+
+ import org.apache.commons.math3.stat.descriptive.DescriptiveStatistics;
+
+ import java.util.ArrayList;
+ import java.util.List;
+
+ public class OutlierDetector {
+     private double outlierFactor;
+     private List<HistogramItem> input;
+     private double[] inputRaw;
+
+     private double min;
+     private double max;
```

```
+
+     private List<HistogramItem> outlier = new ArrayList<>();
+     private List<HistogramItem> inlier = new ArrayList<>();
+
+     public OutlierDetector(double outlierFactor, List<HistogramItem> input) {
+         this.outlierFactor = outlierFactor;
+         this.input = input;
+         this.inputRaw = ListToArray.toPrimitive(HistogramHelper.toArray(input));
+     }
+
+     public void run() {
+         DescriptiveStatistics ds = new DescriptiveStatistics(inputRaw);
+         double median = ds.getPercentile(50.0);
+
+         max = outlierFactor * median;
+         min = median / outlierFactor;
+
+         for (int i = 0; i < input.size(); i++) {
+             double value = input.get(i).getValue();
+
+             if (value <= max && value >= min) {
+                 inlier.add(input.get(i));
+             } else {
+                 outlier.add(input.get(i));
+             }
+         }
+     }
+
+     public double getMin() {
+         return min;
+     }
+
+     public double getMax() {
+         return max;
+     }
+
+     public List<HistogramItem> getOutlier() {
+         return outlier;
+     }
+
+     public List<Histogram Item> getInlier() {
+         return inlier;
+     }
+ }
```

97 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/manager/ForkReconfigureManager.java

```
@@ -0,0 +1,97 @@
+ package org.openjdk.jmh.reconfigure.manager;
+
+ import org.openjdk.jmh.infra.BenchmarkParams;
+ import org.openjdk.jmh.reconfigure.helper.HistogramItem;
+ import org.openjdk.jmh.reconfigure.statistics.evaluation.StatisticalEvaluation;
+ import org.openjdk.jmh.results.IterationResult;
+ import org.openjdk.jmh.runner.format.OutputFormat;
+
+ import java.util.ArrayList;
+ import java.util.List;
+
+ public class ForkReconfigureManager extends ReconfigureManager {
+     private StatisticalEvaluation measurementEvaluation;
+
+     private List<Double> measurementThresholds = new ArrayList<>();
+
+     public ForkReconfigureManager(BenchmarkParams benchParams, OutputFormat out) {
+         super(benchParams, out);
+         warmupEvaluation = StatisticalForkEvaluationFactory.get(benchParams);
+         measurementEvaluation = StatisticalForkEvaluationFactory.get(benchParams);
+     }
+
```

```
+     public void addFork(boolean isWarmup, int fork, List<IterationResult> list) {
+         List<HistogramItem> combined = new ArrayList<>();
+         for (int iteration = 1; iteration <= list.size(); iteration++) {
+             combined.addAll(toHistogramItems(fork, iteration, list.get(iteration - 1)));
+         }
+
+         if (isWarmup) {
+             addWarmupFork(combined);
+         } else {
+             addMeasurementFork(combined);
+         }
+     }
+
+     private void addWarmupFork(List<HistogramItem> list) {
+         warmupEvaluation.addIteration(list);
+     }
+
+     private void addMeasurementFork(List<HistogramItem> list) {
+         measurementEvaluation.addIteration(list);
+     }
+
+     public boolean checkForkThreshold(boolean isWarmup) {
+         if (isWarmup) {
+             return checkWarmupForkThreshold();
+         } else {
+             return checkMeasurementForkThreshold();
+         }
+     }
+
+     private boolean checkWarmupForkThreshold() {
+         int currentWarmupFork = warmupEvaluation.getIterationNumber();
+         if (currentWarmupFork < benchParams.getMinWarmupForks()) {
+             warmupThresholds.add(null);
+             return false;
+         } else {
+             int maxForks = benchParams.getWarmupForks();
+             double value = warmupEvaluation.calculateVariability();
+             warmupThresholds.add(value);
+             boolean result = warmupEvaluation.stableEnvironment(value);
+
+             if (currentWarmupFork == maxForks && !result) {
+                 printWarning("warmup forks", warmupEvaluation.getThreshold(), value);
+             } else if (currentWarmupFork < maxForks && result) {
+                 printInfo(currentWarmupFork, maxForks, "warmup forks", value, warmupEvaluation.getThreshold());
+             }
+
+             return result;
+         }
+     }
+
+     private boolean checkMeasurementForkThreshold() {
+         int currentMeasurementFork = measurementEvaluation.getIterationNumber();
+         if (currentMeasurementFork < benchParams.getMinForks()) {
+             measurementThresholds.add(null);
+             return false;
+         } else {
+             int maxForks = benchParams.getForks();
+             Double value = measurementEvaluation.calculateVariability();
+             measurementThresholds.add(value);
+             boolean result = measurementEvaluation.stableEnvironment(value);
+
+             if (currentMeasurementFork == maxForks && !result) {
+                 printWarning("measurement forks", measurementEvaluation.getThreshold(), value);
+             } else if (currentMeasurementFork < maxForks && result) {
+                 printInfo(currentMeasurementFork, maxForks, "measurement forks", value, measurementEvaluation.getThres
+             }
+
+             return result;
+         }
+     }
```

```
+
+     public List<Double> getMeasurementThresholds() {
+         return measurementThresholds;
+     }
+ }
```

37 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/manager/IterationReconfigureManager.java

```
@@ -0,0 +1,37 @@
+ package org.openjdk.jmh.reconfigure.manager;
+
+ import org.openjdk.jmh.infra.BenchmarkParams;
+ import org.openjdk.jmh.results.IterationResult;
+ import org.openjdk.jmh.runner.format.OutputFormat;
+
+ public class IterationReconfigureManager extends ReconfigureManager {
+     public IterationReconfigureManager(BenchmarkParams benchParams, OutputFormat out) {
+         super(benchParams, out);
+         warmupEvaluation = StatisticalIterationEvaluationFactory.get(benchParams);
+     }
+
+     public void addWarmupIteration(int iteration, IterationResult ir) {
+         warmupEvaluation.addIteration(toHistogramItems(0, iteration, ir));
+     }
+
+     public boolean checkWarmupIterationThreshold() {
+         int currentWarmupIteration = warmupEvaluation.getIterationNumber();
+         if (currentWarmupIteration < benchParams.getWarmup().getMinCount()) {
+             warmupThresholds.add(null);
+             return false;
+         } else {
+             int maxIterations = benchParams.getWarmup().getCount();
+             double value = warmupEvaluation.calculateVariability();
+             warmupThresholds.add(value);
+             boolean result = warmupEvaluation.stableEnvironment(value);
+
+             if (currentWarmupIteration == maxIterations && !result) {
+                 printWarning("warmup iterations", warmupEvaluation.getThreshold(), value);
+             } else if (currentWarmupIteration < maxIterations && result) {
+                 printInfo(currentWarmupIteration, maxIterations, "warmup iterations", value, warmupEvaluation.getThres
+             }
+
+             return result;
+         }
+     }
+ }
```

62 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/manager/ReconfigureManager.java

```
@@ -0,0 +1,62 @@
+ package org.openjdk.jmh.reconfigure.manager;
+
+ import org.openjdk.jmh.infra.BenchmarkParams;
+ import org.openjdk.jmh.reconfigure.helper.HistogramItem;
+ import org.openjdk.jmh.reconfigure.statistics.evaluation.StatisticalEvaluation;
+ import org.openjdk.jmh.results.IterationResult;
+ import org.openjdk.jmh.runner.format.OutputFormat;
+
+ import java.util.ArrayList;
+ import java.util.Iterator;
+ import java.util.List;
+ import java.util.Map;
+
+ public abstract class ReconfigureManager {
+     protected final BenchmarkParams benchParams;
+     private final OutputFormat out;
+
+     protected StatisticalEvaluation warmupEvaluation;
+
```

```
+     protected List<Double> warmupThresholds = new ArrayList<>();
+
+     private boolean atLeastOneWarning = false;
+
+     public ReconfigureManager(BenchmarkParams benchParams, OutputFormat out) {
+         this.benchParams = benchParams;
+         this.out = out;
+     }
+
+     protected List<HistogramItem> toHistogramItems(int fork, int iteration, IterationResult ir) {
+         List<HistogramItem> list = new ArrayList<>();
+         Iterator<Map.Entry<Double, Long>> iterator = ir.getPrimaryResult().getStatistics().getRawData();
+         while (iterator.hasNext()) {
+             Map.Entry<Double, Long> entry = iterator.next();
+             list.add(new HistogramItem(fork, iteration, entry.getKey(), entry.getValue()));
+         }
+         return list;
+     }
+
+     protected void printWarning(String type, double threshold, Double value) {
+         out.println("");
+         out.println("##########");
+         out.println(String.format("# WARNING: Maximum number of %s was reached but statistical variability threshold o
+         out.println("##########");
+         out.println("");
+         atLeastOneWarning = true;
+     }
+
+     protected void printInfo(int currentNumber, int maxNumber, String type, Double value, double threshold) {
+         out.println("");
+         String lessOrGreaterThan = (value < threshold) ? "is less" : "is greater";
+         out.println(String.format("# Data collection is stopped after %d of %d %s because value of %.4f " + lessOrGrea
+         out.println("");
+     }
+
+     public List<Double> getWarmupThresholds() {
+         return warmupThresholds;
+     }
+
+     public boolean hasAtLeastOneWarning() {
+         return atLeastOneWarning;
+     }
+ }
```

22 ...e/src/main/java/org/openjdk/jmh/reconfigure/manager/StatisticalForkEvaluationFactory.java

```
@@ -0,0 +1,22 @@
+ package org.openjdk.jmh.reconfigure.manager;
+
+ import org.openjdk.jmh.infra.BenchmarkParams;
+ import org.openjdk.jmh.reconfigure.statistics.evaluation.CiPercentageEvaluation;
+ import org.openjdk.jmh.reconfigure.statistics.evaluation.CovEvaluation;
+ import org.openjdk.jmh.reconfigure.statistics.evaluation.DivergenceEvaluation;
+ import org.openjdk.jmh.reconfigure.statistics.evaluation.StatisticalEvaluation;
+
+ public class StatisticalForkEvaluationFactory {
+     public static StatisticalEvaluation get(BenchmarkParams benchParams) {
+         switch (benchParams.getReconfigureMode()) {
+             case CI:
+                 return CiPercentageEvaluation.getForkInstance(benchParams.getReconfigureCiThreshold());
+             case COV:
+                 return CovEvaluation.getForkInstance(benchParams.getReconfigureCovThreshold());
+             case DIVERGENCE:
+                 return DivergenceEvaluation.getForkInstance(benchParams.getReconfigureKldThreshold());
+             default:
+                 throw new IllegalArgumentException("Reconfigure Mode is nod valid");
+         }
+     }
+ }
```

22 .../main/java/org/openjdk/jmh/reconfigure/manager/StatisticalIterationEvaluationFactory.java

```
@@ -0,0 +1,22 @@
+ package org.openjdk.jmh.reconfigure.manager;
+
+ import org.openjdk.jmh.infra.BenchmarkParams;
+ import org.openjdk.jmh.reconfigure.statistics.evaluation.CiPercentageEvaluation;
+ import org.openjdk.jmh.reconfigure.statistics.evaluation.CovEvaluation;
+ import org.openjdk.jmh.reconfigure.statistics.evaluation.DivergenceEvaluation;
+ import org.openjdk.jmh.reconfigure.statistics.evaluation.StatisticalEvaluation;
+
+ public class StatisticalIterationEvaluationFactory {
+     public static StatisticalEvaluation get(BenchmarkParams benchParams) {
+         switch (benchParams.getReconfigureMode()) {
+             case CI:
+                 return CiPercentageEvaluation.getIterationInstance(benchParams.getReconfigureCiThreshold());
+             case COV:
+                 return CovEvaluation.getIterationInstance(benchParams.getReconfigureCovThreshold());
+             case DIVERGENCE:
+                 return DivergenceEvaluation.getIterationInstance(benchParams.getReconfigureKldThreshold());
+             default:
+                 throw new IllegalArgumentException("Reconfigure Mode is nod valid");
+         }
+     }
+ }
```

23 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/statistics/COV.java

```
@@ -0,0 +1,23 @@
+ package org.openjdk.jmh.reconfigure.statistics;
+
+ import org.apache.commons.math3.stat.descriptive.DescriptiveStatistics;
+ import org.openjdk.jmh.reconfigure.helper.HistogramHelper;
+ import org.openjdk.jmh.reconfigure.helper.HistogramItem;
+ import org.openjdk.jmh.reconfigure.helper.ListToArray;
+
+ import java.util.List;
+
+ public class COV implements Statistic {
+     private List<Double> list;
+
+     public COV(List<HistogramItem> list) {
+         this.list = HistogramHelper.toArray(list);
+     }
+
+     @Override
+     public double getValue() {
+         double[] array = ListToArray.toPrimitive(list);
+         DescriptiveStatistics ds = new DescriptiveStatistics(array);
+         return ds.getStandardDeviation() / ds.getMean();
+     }
+ }
```

13 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/statistics/ReconfigureConstants.java

```
@@ -0,0 +1,13 @@
+ package org.openjdk.jmh.reconfigure.statistics;
+
+ public class ReconfigureConstants {
+     public static final double OUTLIER_FACTOR = 10.0;
+     public static final double RANGE_OUTLIER_FACTOR = 1.5;
+     public static final int SAMPLE_SIZE = 1000;
+     public static final int DIVERGENCE_NUMBER_OF_POINTS = 1000;
+     public static final int CI_BOOTSTRAP_SIMULATIONS = 1000;
+     public static final double CI_SIGNIFICANCE_LEVEL = 0.01;
+     public static final String CI_STATISTIC = "mean";
+     public static final int CI_INVOCATION_SAMPLES = -1;
+     public static final int CI_SAMPLE_SIZE = 10000;
+ }
```

25 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/statistics/Sampler.java

```
@@ -0,0 +1,25 @@
+ package org.openjdk.jmh.reconfigure.statistics;
+
+ import org.apache.commons.math3.distribution.EnumeratedDistribution;
+ import org.apache.commons.math3.util.Pair;
+ import org.openjdk.jmh.reconfigure.helper.HistogramItem;
+
+ import java.util.Arrays;
+ import java.util.List;
+ import java.util.stream.Collectors;
+
+ public class Sampler {
+     private List<HistogramItem> items;
+
+     public Sampler(List<HistogramItem> items) {
+         this.items = items;
+     }
+
+     public List<HistogramItem> getSample(int size) {
+         HistogramItem[] output = new HistogramItem[size];
+         List<Pair<HistogramItem, Double>> distributionPairs = items.parallelStream().map(it -> new Pair<>(it.single(),
+         EnumeratedDistribution ed = new EnumeratedDistribution<>(distributionPairs);
+         ed.sample(size, output);
+         return Arrays.asList(output);
+     }
+ }
```

5 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/statistics/Statistic.java

```
@@ -0,0 +1,5 @@
+ package org.openjdk.jmh.reconfigure.statistics;
+
+ public interface Statistic {
+     double getValue();
+ }
```

101 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/statistics/ci/CI.java

```
@@ -0,0 +1,101 @@
+ package org.openjdk.jmh.reconfigure.statistics.ci;
+
+ import org.apache.commons.io.IOUtils;
+ import org.openjdk.jmh.reconfigure.helper.HistogramItem;
+
+ import java.io.File;
+ import java.io.FileWriter;
+ import java.io.IOException;
+ import java.nio.charset.Charset;
+ import java.util.List;
+
+ public class CI {
+     protected List<HistogramItem> histogramList;
+
+     protected String paToolPath;
+     protected final int bootstrapSimulations;
+     protected final double significanceLevel;
+     protected final String statistic;
+     protected final int ciInvocationSamples;
+
+     protected double lower;
+     protected double upper;
+     protected double statisticMetric;
+
+     public CI(List<HistogramItem> histogramList, int bootstrapSimulations, double significanceLevel, String statistic,
+         this.histogramList = histogramList;
+         this.bootstrapSimulations = bootstrapSimulations;
+         this.significanceLevel = significanceLevel;
```

```
+        this.statistic = statistic;
+        this.ciInvocationSamples = ciInvocationSamples;
+        executable();
+    }
+
+    private void executable() {
+        paToolPath = CIHelper.getInstance().getPath();
+    }
+
+    public void run() {
+        String file = getTmpFile(histogramList);
+        try {
+            Process process = Runtime.getRuntime().exec(paToolPath + " -os -bs " + bootstrapSimulations + " -is " + ci
+            String inputString = IOUtils.toString(process.getInputStream(), Charset.defaultCharset());
+            String errorString = IOUtils.toString(process.getErrorStream(), Charset.defaultCharset());
+            String output = (inputString + "\n" + errorString).trim();
+            String line = getFirstLine(output);
+            String[] parts = line.split(";");
+            statisticMetric = Double.parseDouble(parts[3]);
+            lower = Double.parseDouble(parts[4]);
+            upper = Double.parseDouble(parts[5]);
+        } catch (IOException e) {
+            e.printStackTrace();
+        } finally {
+            new File(file).delete();
+        }
+    }
+
+    protected String getFirstLine(String input) {
+        String[] lines = input.split("\n");
+
+        for (int i = 0; i < lines.length; i++) {
+            String line = lines[i].trim();
+
+            if (!line.startsWith("#") && !line.isEmpty()) {
+                return line;
+            }
+        }
+
+        return null;
+    }
+
+    protected String getTmpFile(List<HistogramItem> list) {
+        try {
+            File tmpFile = File.createTempFile("reconfigure", ".csv");
+            FileWriter fw = new FileWriter(tmpFile);
+
+            for (int i = 0; i < list.size(); i++) {
+                HistogramItem hi = list.get(i);
+                fw.append(";;;;;0;" + hi.getFork() + ";" + hi.getIteration() + ";;;" + hi.getCount() + ";" + hi.getVal
+            }
+
+            fw.flush();
+            return tmpFile.getAbsolutePath();
+        } catch (IOException e) {
+            e.printStackTrace();
+        }
+
+        return null;
+    }
+
+    public double getLower() {
+        return lower;
+    }
+
+    public double getUpper() {
+        return upper;
+    }
+
+    public double getStatisticMetric() {
```

```
+         return statisticMetric;
+     }
+ }
```

88 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/statistics/ci/CIHelper.java

```
@@ -0,0 +1,88 @@
+ package org.openjdk.jmh.reconfigure.statistics.ci;
+
+ import org.apache.commons.io.FileUtils;
+ import org.apache.commons.lang3.SystemUtils;
+
+ import java.io.File;
+ import java.io.IOException;
+ import java.io.InputStream;
+ import java.net.URL;
+ import java.nio.file.Paths;
+
+ public class CIHelper {
+     private static CIHelper instance;
+     private File tempFile;
+
+     private CIHelper() {
+         copyExecutableToTmpFolder();
+     }
+
+     public static CIHelper getInstance() {
+         if (CIHelper.instance == null) {
+             CIHelper.instance = new CIHelper();
+         }
+         return CIHelper.instance;
+     }
+
+     public String getPath() {
+         return tempFile.getAbsolutePath();
+     }
+
+     private void copyExecutableToTmpFolder() {
+         String executableName = executableName();
+         tempFile = Paths.get(System.getProperty("java.io.tmpdir"), executableName).toFile();
+
+         ClassLoader classLoader = getClass().getClassLoader();
+
+         URL resource = classLoader.getResource(executableName());
+         if (resource == null) {
+             throw new IllegalArgumentException("file is not found!");
+         } else {
+             if (resource.getProtocol().equals("jar")) {
+                 copyExecutableToTmpFolderJar(resource);
+             } else {
+                 copyExecutableToTmpFolderFile(resource);
+             }
+         }
+
+         tempFile.setExecutable(true, true);
+     }
+
+     private void copyExecutableToTmpFolderFile(URL resource) {
+         try {
+             File executable = new File(resource.getFile());
+             FileUtils.copyFile(executable, tempFile);
+         } catch (IOException e) {
+             e.printStackTrace();
+         }
+     }
+
+     private void copyExecutableToTmpFolderJar(URL resource) {
+         try {
+             InputStream in = resource.openStream();
+             FileUtils.copyInputStreamToFile(in, tempFile);
```

```
+         } catch (IOException e) {
+             e.printStackTrace();
+         }
+     }
+
+     private boolean isWindows() {
+         return SystemUtils.IS_OS_WINDOWS;
+     }
+
+     private boolean isMacOS() {
+         return SystemUtils.IS_OS_MAC_OSX;
+     }
+
+     private String executableName() {
+         String e;
+         if (isWindows()) {
+             e = "pa_windows_amd64.exe";
+         } else if (isMacOS()) {
+             e = "pa_darwin_amd64";
+         } else {
+             e = "pa_linux_amd64";
+         }
+         return e;
+     }
+ }
```

32 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/statistics/ci/CiPercentage.java

```
@@ -0,0 +1,32 @@
+ package org.openjdk.jmh.reconfigure.statistics.ci;
+
+ import org.openjdk.jmh.reconfigure.helper.HistogramItem;
+ import org.openjdk.jmh.reconfigure.statistics.Statistic;
+
+ import java.util.List;
+
+ import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.*;
+
+ public class CiPercentage implements Statistic {
+     private List<HistogramItem> histogramList;
+     private int ciBootstrapSimulations = CI_BOOTSTRAP_SIMULATIONS;
+     private double ciSignificanceLevel = CI_SIGNIFICANCE_LEVEL;
+     private String ciStatistics = CI_STATISTIC;
+     private int ciInvocationSamples = CI_INVOCATION_SAMPLES;
+
+     public CiPercentage(List<HistogramItem> histogramList) {
+         this.histogramList = histogramList;
+     }
+
+     public CiPercentage(List<HistogramItem> histogramList, int ciBootstrapSimulations) {
+         this.histogramList = histogramList;
+         this.ciBootstrapSimulations = ciBootstrapSimulations;
+     }
+
+     @Override
+     public double getValue() {
+         CI ci = new CI(histogramList, ciBootstrapSimulations, ciSignificanceLevel, ciStatistics, ciInvocationSamples);
+         ci.run();
+         return (ci.getUpper() - ci.getLower()) / ci.getStatisticMetric();
+     }
+ }
```

72 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/statistics/divergence/Divergence.java

```
@@ -0,0 +1,72 @@
+ package org.openjdk.jmh.reconfigure.statistics.divergence;
+
+ import org.apache.commons.math3.util.Pair;
+ import org.apache.commons.math3.stat.descriptive.DescriptiveStatistics;
```

```
+ import org.openjdk.jmh.reconfigure.helper.ListToArray;
+ import org.openjdk.jmh.reconfigure.statistics.Statistic;
+
+ import java.util.ArrayList;
+ import java.util.List;
+
+ import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.DIVERGENCE_NUMBER_OF_POINTS;
+ import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.RANGE_OUTLIER_FACTOR;
+
+ public class Divergence implements Statistic {
+     private List<Double> before;
+     private List<Double> after;
+
+     public Divergence(List<Double> before, List<Double> after) {
+         this.before = before;
+         this.after = after;
+     }
+
+     @Override
+     public double getValue() {
+         Pair<Double, Double> range = getRange();
+         double min = range.getFirst();
+         double max = range.getSecond();
+
+         if (min == max) {
+             return 1.0;
+         }
+
+         List<Double> y = new ArrayList<>();
+         double step = (max - min) / (DIVERGENCE_NUMBER_OF_POINTS - 1);
+         for (int i = 0; i < DIVERGENCE_NUMBER_OF_POINTS; i++) {
+             y.add(min + i * step);
+         }
+
+         double[] pdfBefore = ProbabilityDensityFunction.estimate(before, y);
+         double[] pdfAfter = ProbabilityDensityFunction.estimate(after, y);
+
+         double kldBefore = KullbackLeiblerDivergence.continuous(pdfBefore, pdfAfter, step);
+         double kldAfter = KullbackLeiblerDivergence.continuous(pdfAfter, pdfBefore, step);
+
+         return Math.pow(2.0, -kldBefore) * Math.pow(2.0, -kldAfter);
+     }
+
+     private Pair<Double, Double> getRange() {
+         Pair<Double, Double> rangeBefore = getRangeDistribution(before);
+         Pair<Double, Double> rangeAfter = getRangeDistribution(after);
+
+         double min = Math.min(rangeBefore.getFirst(), rangeAfter.getFirst());
+         double max = Math.max(rangeBefore.getSecond(), rangeAfter.getSecond());
+
+         return new Pair<Double, Double>(min, max);
+     }
+
+     private Pair<Double, Double> getRangeDistribution(List<Double> list) {
+         DescriptiveStatistics ds = new DescriptiveStatistics(ListToArray.toPrimitive(list));
+         double q1 = ds.getPercentile(25.0);
+         double q3 = ds.getPercentile(75.0);
+         double iqr = q3 - q1;
+
+         double max = q3 + RANGE_OUTLIER_FACTOR * iqr;
+         double min = q1 - RANGE_OUTLIER_FACTOR * iqr;
+
+         if (min < 0) {
+             min = 0.0;
+         }
+         return new Pair<Double, Double>(min, max);
+     }
+ }
```

21 ...ain/java/org/openjdk/jmh/reconfigure/statistics/divergence/KullbackLeiblerDivergence.java

```
@@ -0,0 +1,21 @@
+ package org.openjdk.jmh.reconfigure.statistics.divergence;
+
+ public class KullbackLeiblerDivergence {
+     public static double continuous(double[] x, double[] y, double width) {
+         boolean intersection = false;
+         double kl = 0.0;
+
+         for (int i = 0; i < x.length; i++) {
+             if (x[i] != 0.0 && y[i] != 0.0) {
+                 intersection = true;
+                 kl += x[i] * Math.log(x[i] / y[i]) * width;
+             }
+         }
+
+         if (intersection) {
+             return kl;
+         } else {
+             return Double.POSITIVE_INFINITY;
+         }
+     }
+ }
```

20 ...in/java/org/openjdk/jmh/reconfigure/statistics/divergence/ProbabilityDensityFunction.java

```
@@ -0,0 +1,20 @@
+ package org.openjdk.jmh.reconfigure.statistics.divergence;
+
+ import org.openjdk.jmh.reconfigure.helper.ListToArray;
+ import smile.stat.distribution.KernelDensity;
+
+ import java.util.ArrayList;
+ import java.util.List;
+
+ public class ProbabilityDensityFunction {
+     public static double[] estimate(List<Double> sample, List<Double> y) {
+         KernelDensity kd = new KernelDensity(ListToArray.toPrimitive(sample));
+         List<Double> result = new ArrayList<>();
+
+         for (double value : y) {
+             result.add(kd.p(value));
+         }
+
+         return ListToArray.toPrimitive(result);
+     }
+ }
```

89 ...c/main/java/org/openjdk/jmh/reconfigure/statistics/evaluation/CiPercentageEvaluation.java

```
@@ -0,0 +1,89 @@
+ package org.openjdk.jmh.reconfigure.statistics.evaluation;
+
+ import org.openjdk.jmh.reconfigure.helper.HistogramItem;
+ import org.openjdk.jmh.reconfigure.helper.OutlierDetector;
+ import org.openjdk.jmh.reconfigure.statistics.Sampler;
+ import org.openjdk.jmh.reconfigure.statistics.ci.CiPercentage;
+
+ import java.util.*;
+
+ import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.*;
+
+ public class CiPercentageEvaluation implements StatisticalEvaluation {
+     private double threshold;
+     private double historySize;
+
+     private List<HistogramItem> allMeasurements = new ArrayList<>();
+     private Map<Integer, List<HistogramItem>> sampleInIteration = new HashMap<>();
```

```
+     private Map<Integer, Double> ciPercentagePerIteration = new HashMap<>();
+
+     private CiPercentageEvaluation(double threshold, int historySize) {
+         this.threshold = threshold;
+         this.historySize = historySize;
+     }
+
+     public static CiPercentageEvaluation getIterationInstance(double threshold) {
+         return new CiPercentageEvaluation(threshold, 5);
+     }
+
+     public static CiPercentageEvaluation getForkInstance(double threshold) {
+         return new CiPercentageEvaluation(threshold, 2);
+     }
+
+     @Override
+     public void addIteration(List<HistogramItem> list) {
+         OutlierDetector od = new OutlierDetector(OUTLIER_FACTOR, list);
+         od.run();
+         List<HistogramItem> sample = new Sampler(od.getInlier()).getSample(SAMPLE_SIZE);
+
+         int iteration = sampleInIteration.size() + 1;
+         allMeasurements.addAll(sample);
+
+         List<HistogramItem> sampleUntil = new Sampler(allMeasurements).getSample(SAMPLE_SIZE);
+         sampleInIteration.put(iteration, sampleUntil);
+     }
+
+     @Override
+     public double getThreshold() {
+         return threshold;
+     }
+
+     @Override
+     public Double calculateVariability() {
+         if (sampleInIteration.size() < historySize) {
+             return null;
+         } else {
+             List<Double> deltas = new ArrayList<>();
+             int currentIteration = sampleInIteration.size();
+             double currentCiPercentage = getCiPercentageOfIteration(currentIteration);
+
+             for (int i = 1; i <= historySize - 1; i++) {
+                 double ciPercentage = getCiPercentageOfIteration(currentIteration - i);
+                 double delta = Math.abs(ciPercentage - currentCiPercentage);
+                 deltas.add(delta);
+             }
+
+             return Collections.max(deltas);
+         }
+     }
+
+     public double getCiPercentageOfIteration(int iteration) {
+         if (ciPercentagePerIteration.get(iteration) == null) {
+             double ciPercentage = new CiPercentage(sampleInIteration.get(iteration)).getValue();
+             ciPercentagePerIteration.put(iteration, ciPercentage);
+             return ciPercentage;
+         } else {
+             return ciPercentagePerIteration.get(iteration);
+         }
+     }
+
+     @Override
+     public int getIterationNumber() {
+         return sampleInIteration.size();
+     }
+
+     @Override
+     public boolean stableEnvironment(Double value) {
+         return value != null && value < threshold;
```

```
+     }
+ }
```

91 jmh-core/src/main/java/org/openjdk/jmh/reconfigure/statistics/evaluation/CovEvaluation.java

```
@@ -0,0 +1,91 @@
+ package org.openjdk.jmh.reconfigure.statistics.evaluation;
+
+ import org.openjdk.jmh.reconfigure.helper.HistogramItem;
+ import org.openjdk.jmh.reconfigure.helper.OutlierDetector;
+ import org.openjdk.jmh.reconfigure.statistics.COV;
+ import org.openjdk.jmh.reconfigure.statistics.Sampler;
+
+ import java.util.*;
+
+ import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.OUTLIER_FACTOR;
+ import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.SAMPLE_SIZE;
+
+ public class CovEvaluation implements StatisticalEvaluation {
+     private double threshold;
+     private double historySize;
+
+     private Map<Integer, List<HistogramItem>> samplePerIteration = new HashMap<>();
+     private Map<Integer, Double> covPerIteration = new HashMap<>();
+
+     private CovEvaluation(double threshold, int historySize) {
+         this.threshold = threshold;
+         this.historySize = historySize;
+     }
+
+     public static CovEvaluation getIterationInstance(double threshold) {
+         return new CovEvaluation(threshold, 5);
+     }
+
+     public static CovEvaluation getForkInstance(double threshold) {
+         return new CovEvaluation(threshold, 2);
+     }
+
+     @Override
+     public void addIteration(List<HistogramItem> list) {
+         OutlierDetector od = new OutlierDetector(OUTLIER_FACTOR, list);
+         od.run();
+         List<HistogramItem> sample = new Sampler(od.getInlier()).getSample(SAMPLE_SIZE);
+
+         int iteration = samplePerIteration.size() + 1;
+         samplePerIteration.put(iteration, sample);
+     }
+
+     @Override
+     public double getThreshold() {
+         return threshold;
+     }
+
+     @Override
+     public Double calculateVariability() {
+         if (samplePerIteration.size() < historySize) {
+             return null;
+         } else {
+             List<Double> deltas = new ArrayList<>();
+             int currentIteration = samplePerIteration.size();
+             double currentCov = getCovOfIteration(currentIteration);
+
+             for (int i = 1; i <= historySize - 1; i++) {
+                 double cov = getCovOfIteration(currentIteration - i);
+                 double delta = Math.abs(cov - currentCov);
+                 deltas.add(delta);
+             }
+
+             return Collections.max(deltas);
+         }
```

```
+     }
+
+     public double getCovOfIteration(int iteration) {
+         if (covPerIteration.get(iteration) == null) {
+             List<HistogramItem> all = new ArrayList<>();
+             for (int i = 1; i <= iteration; i++) {
+                 all.addAll(samplePerIteration.get(i));
+             }
+
+             double cov = new COV(all).getValue();
+             covPerIteration.put(iteration, cov);
+             return cov;
+         } else {
+             return covPerIteration.get(iteration);
+         }
+     }
+
+     @Override
+     public int getIterationNumber() {
+         return samplePerIteration.size();
+     }
+
+     @Override
+     public boolean stableEnvironment(Double value) {
+         return value != null && value < threshold;
+     }
+ }
```

119 ...src/main/java/org/openjdk/jmh/reconfigure/statistics/evaluation/DivergenceEvaluation.java

```
@@ -0,0 +1,119 @@
+ package org.openjdk.jmh.reconfigure.statistics.evaluation;
+
+ import org.openjdk.jmh.reconfigure.helper.HistogramHelper;
+ import org.openjdk.jmh.reconfigure.helper.HistogramItem;
+ import org.openjdk.jmh.reconfigure.helper.OutlierDetector;
+ import org.openjdk.jmh.reconfigure.statistics.Sampler;
+ import org.openjdk.jmh.reconfigure.statistics.divergence.Divergence;
+
+ import java.io.OutputStream;
+ import java.io.PrintStream;
+ import java.util.*;
+
+ import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.OUTLIER_FACTOR;
+ import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.SAMPLE_SIZE;
+
+ public class DivergenceEvaluation implements StatisticalEvaluation {
+     private double threshold;
+     private double historySize;
+
+     private List<Double> allMeasurements = new ArrayList<>();
+     private Map<Integer, List<Double>> sampleUntilIteration = new HashMap<>();
+     private Map<Integer, Double> pValuePerIteration = new HashMap<>();
+
+     private DivergenceEvaluation(double threshold, int historySize) {
+         this.threshold = threshold;
+         this.historySize = historySize;
+         disableSystemErr();
+     }
+
+     public static DivergenceEvaluation getIterationInstance(double threshold) {
+         return new DivergenceEvaluation(threshold, 6);
+     }
+
+     public static DivergenceEvaluation getForkInstance(double threshold) {
+         return new DivergenceEvaluation(threshold, 2);
+     }
+
+     @Override
+     public void addIteration(List<HistogramItem> list) {
```

```
+        OutlierDetector od = new OutlierDetector(OUTLIER_FACTOR, list);
+        od.run();
+        List<HistogramItem> sample = new Sampler(od.getInlier()).getSample(SAMPLE_SIZE);
+
+        int iteration = sampleUntilIteration.size() + 1;
+        List<Double> newValues = HistogramHelper.toArray(sample);
+        allMeasurements.addAll(newValues);
+
+        List<Double> sampleUntil = getSample(allMeasurements);
+        sampleUntilIteration.put(iteration, sampleUntil);
+    }
+
+    @Override
+    public double getThreshold() {
+        return threshold;
+    }
+
+    @Override
+    public Double calculateVariability() {
+        if (sampleUntilIteration.size() < historySize) {
+            return null;
+        } else {
+            List<Double> pvalues = new ArrayList<>();
+            int currentIteration = sampleUntilIteration.size();
+
+            for (int i = 0; i <= historySize - 2; i++) {
+                Double pvalue = getPValueOfIteration(currentIteration - i);
+                pvalues.add(pvalue);
+            }
+
+            return pvalues.stream().mapToDouble(it -> it).average().orElse(0.0);
+        }
+    }
+
+    public Double getPValueOfIteration(int iteration) {
+        if (pValuePerIteration.get(iteration) == null) {
+            List<Double> currentSample = sampleUntilIteration.get(iteration);
+            List<Double> previousSample = sampleUntilIteration.get(iteration - 1);
+            if (currentSample == null || previousSample == null) {
+                return null;
+            } else {
+                double pValue = new Divergence(currentSample, previousSample).getValue();
+                pValuePerIteration.put(iteration, pValue);
+                return pValue;
+            }
+        } else {
+            return pValuePerIteration.get(iteration);
+        }
+    }
+
+    private List<Double> getSample(List<Double> list) {
+        Random random = new Random();
+        List<Double> sample = new ArrayList<>();
+
+        for (int i = 0; i < SAMPLE_SIZE; i++) {
+            Double d = list.get(random.nextInt(list.size()));
+            sample.add(d);
+        }
+
+        Collections.sort(sample);
+        return sample;
+    }
+
+    private void disableSystemErr() {
+        System.setErr(new PrintStream(new OutputStream() {
+            public void write(int b) {
+            }
+        }));
+    }
+
```

```
+     @Override
+     public int getIterationNumber() {
+         return sampleUntilIteration.size();
+     }
+
+     @Override
+     public boolean stableEnvironment(Double value) {
+         return value != null && value > threshold;
+     }
+ }
```

17 ...rc/main/java/org/openjdk/jmh/reconfigure/statistics/evaluation/StatisticalEvaluation.java

```
@@ -0,0 +1,17 @@
+ package org.openjdk.jmh.reconfigure.statistics.evaluation;
+
+ import org.openjdk.jmh.reconfigure.helper.HistogramItem;
+
+ import java.util.List;
+
+ public interface StatisticalEvaluation {
+     void addIteration(List<HistogramItem> list);
+
+     double getThreshold();
+
+     Double calculateVariability();
+
+     int getIterationNumber();
+
+     boolean stableEnvironment(Double value);
+ }
```

15 jmh-core/src/main/java/org/openjdk/jmh/results/BenchmarkResultMetaData.java

```
  package org.openjdk.jmh.results;

  import java.io.Serializable;
+ import java.util.List;

  public class BenchmarkResultMetaData implements Serializable {

      private final long stopTime;
      private final long warmupOps;
      private final long measurementOps;
+     private final List<Double> warmupThresholds;
+     private final boolean atLeastOneWarning;

-     public BenchmarkResultMetaData(long warmupTime, long measurementTime, long stopTime, long warmupOps, long measurem
+     public BenchmarkResultMetaData(long warmupTime, long measurementTime, long stopTime, long warmupOps, long measurem
          this.startTime = Long.MIN_VALUE;
          this.warmupTime = warmupTime;
          this.measurementTime = measurementTime;
          this.stopTime = stopTime;
          this.warmupOps = warmupOps;
          this.measurementOps = measurementOps;
+         this.warmupThresholds = warmupThresholds;
+         this.atLeastOneWarning = atLeastOneWarning;
      }

      public long getStartTime() {
          return warmupOps;
      }

+     public List<Double> getWarmupThresholds() {
+         return warmupThresholds;
+     }
+
+     public boolean hasAtLeastOneWarning() {
+         return atLeastOneWarning;
+     }
```

```
+
    public void adjustStart(long startTime) {
        this.startTime = startTime;
    }
```

20 jmh-core/src/main/java/org/openjdk/jmh/results/RunResult.java

```
  import org.openjdk.jmh.infra.BenchmarkParams;

  import java.io.Serializable;
- import java.util.ArrayList;
- import java.util.Collection;
- import java.util.Comparator;
- import java.util.Map;
+ import java.util.*;

  /**
   * Complete run result.

      private final Collection<BenchmarkResult> benchmarkResults;
      private final BenchmarkParams params;
+     private final List<Double> warmupThresholds;
+     private final List<Double> measurementThresholds;

-     public RunResult(BenchmarkParams params, Collection<BenchmarkResult> data) {
+     public RunResult(BenchmarkParams params, Collection<BenchmarkResult> data, List<Double> warmupThresholds, List<Dou
          this.benchmarkResults = data;
          this.params = params;
+         this.warmupThresholds = warmupThresholds;
+         this.measurementThresholds = measurementThresholds;
      }

      public Collection<BenchmarkResult> getBenchmarkResults() {
          return params;
      }

+     public List<Double> getWarmupThresholds() {
+         return warmupThresholds;
+     }
+
+     public List<Double> getMeasurementThresholds() {
+         return measurementThresholds;
+     }
+
      public static final Comparator<RunResult> DEFAULT_SORT_COMPARATOR = new Comparator<RunResult>() {
          @Override
          public int compare(RunResult o1, RunResult o2) {
              return o1.params.compareTo(o2.params);
          }
      };
-
  }
```

145 jmh-core/src/main/java/org/openjdk/jmh/results/format/JSONResultFormat.java

```
   */
  package org.openjdk.jmh.results.format;

+ import org.openjdk.jmh.annotations.Mode;
  import org.openjdk.jmh.infra.BenchmarkParams;
  import org.openjdk.jmh.results.BenchmarkResult;
  import org.openjdk.jmh.results.IterationResult;
  import java.io.StringWriter;
  import java.util.ArrayList;
  import java.util.Collection;
+ import java.util.List;
  import java.util.Map;

  class JSONResultFormat implements ResultFormat {
          pw.println("[");
```

```
61  63          for (RunResult runResult : results) {
62  64              BenchmarkParams params = runResult.getParams();
    65 +            boolean isReconfigureMode = params.getMode().equals(Mode.Reconfigure);
63  66
64  67              if (first) {
65  68                  first = false;
74  77              pw.println("\"mode\" : \"" + params.getMode().shortLabel() + "\",");
75  78              pw.println("\"threads\" : " + params.getThreads() + ",");
76  79              pw.println("\"forks\" : " + params.getForks() + ",");
    80 +            if (isReconfigureMode) {
    81 +                pw.println("\"minForks\" : " + params.getMinForks() + ",");
    82 +            }
    83 +            pw.println("\"warmupForks\" : " + params.getWarmupForks() + ",");
    84 +            if (isReconfigureMode) {
    85 +                pw.println("\"minWarmupForks\" : " + params.getMinWarmupForks() + ",");
    86 +            }
77  87              pw.println("\"jvm\" : " + toJsonString(params.getJvm()) + ",");
78  88              // if empty, write an empty array.
79  89              pw.println("\"jvmArgs\" : [");
83  93              pw.println("\"vmName\" : " + toJsonString(params.getVmName()) + ",");
84  94              pw.println("\"vmVersion\" : " + toJsonString(params.getVmVersion()) + ",");
85  95              pw.println("\"warmupIterations\" : " + params.getWarmup().getCount() + ",");
    96 +            if (isReconfigureMode) {
    97 +                pw.println("\"minWarmupIterations\" : " + params.getWarmup().getMinCount() + ",");
    98 +            }
86  99              pw.println("\"warmupTime\" : \"" + params.getWarmup().getTime() + "\",");
87  100             pw.println("\"warmupBatchSize\" : " + params.getWarmup().getBatchSize() + ",");
88  101             pw.println("\"measurementIterations\" : " + params.getMeasurement().getCount() + ",");
95  108                 pw.println("},");
96  109             }
97  110
    111 +           if (isReconfigureMode) {
    112 +               pw.println("\"reconfigureMode\" : " + toJsonString(params.getReconfigureMode().shortLabel()) + ",");
    113 +               pw.println("\"reconfigureThreshold\" : " + params.getReconfigureThreshold() + ",");
    114 +
    115 +               pw.println("\"thresholds\" : {");
    116 +
    117 +               pw.println("\"warmupForks\" : ");
    118 +               Collection<String> warmupForkThresholds = toListOfStrings(runResult.getWarmupThresholds());
    119 +               pw.println(printMultiple(warmupForkThresholds, "[", "]"));
    120 +               pw.println(",");
    121 +
    122 +               pw.println("\"measurementForks\" : ");
    123 +               Collection<String> measurementForkThresholds = toListOfStrings(runResult.getMeasurementThresholds());
    124 +               pw.println(printMultiple(measurementForkThresholds, "[", "]"));
    125 +               pw.println(",");
    126 +
    127 +               Collection<String> warmupIterationList = new ArrayList<>();
    128 +               for (BenchmarkResult benchmarkResult : runResult.getBenchmarkResults()) {
    129 +                   Collection<String> warmupIterationThresholds = toListOfStrings(benchmarkResult.getMetadata().getWa
    130 +                   warmupIterationList.add(printMultiple(warmupIterationThresholds, "[", "]"));
    131 +               }
    132 +
    133 +               pw.println("\"warmupIterations\" : {");
    134 +               pw.println(printMultiple(warmupIterationList, "[", "]"));
    135 +               pw.println("},");
    136 +
    137 +               pw.println("},");
    138 +
    139 +               pw.println("\"warnings\" : {");
    140 +               pw.println(tidy(getThresholdWarnings(runResult)));
    141 +               pw.println("},");
    142 +           }
    143 +
98  144             Result primaryResult = runResult.getPrimaryResult();
99  145             pw.println("\"primaryMetric\" : {");
100 146             pw.println("\"score\" : " + emit(primaryResult.getScore()) + ",");
105 151
106 152             switch (params.getMode()) {
107 153                 case SampleTime:
```

```
+                 case Reconfigure:
                      pw.println("\"rawDataHistogram\" :");
                      pw.println(getRawData(runResult, true));
                      break;
          return sb.toString();
      }

+     private String getThresholdWarnings(RunResult runResult) {
+         BenchmarkParams params = runResult.getParams();
+         double threshold = params.getReconfigureThreshold();
+
+         StringBuilder sb = new StringBuilder();
+
+         boolean warmupForkHasWarning = false;
+         if (runResult.getWarmupThresholds().size() > 0) {
+             Double lastWarmupForkItem = runResult.getWarmupThresholds().get(runResult.getWarmupThresholds().size() - 1
+             warmupForkHasWarning = runResult.getWarmupThresholds().size() == params.getMinWarmupForks() && lastWarmupF
+         }
+         sb.append("\"warmupForks\" : " + warmupForkHasWarning + ",");
+
+         boolean measurementForkJasWarning = false;
+         if (runResult.getMeasurementThresholds().size() > 0) {
+             Double lastMeasurementForkItem = runResult.getMeasurementThresholds().get(runResult.getMeasurementThreshol
+             measurementForkJasWarning = runResult.getMeasurementThresholds().size() == params.getMinForks() && lastMea
+         }
+         sb.append("\"measurementForks\" : " + measurementForkJasWarning + ",");
+
+         Collection<String> warmupIterationList = new ArrayList<>();
+         for (BenchmarkResult benchmarkResult : runResult.getBenchmarkResults()) {
+             List<Double> warmupThresholds = benchmarkResult.getMetadata().getWarmupThresholds();
+             if (warmupThresholds.size() > 0) {
+                 Double lastWarmupIterationItem = warmupThresholds.get(warmupThresholds.size() - 1);
+                 boolean warmupIterationHasWarning = warmupThresholds.size() == params.getWarmup().getCount() && lastWa
+                 warmupIterationList.add(warmupIterationHasWarning ? "true" : "false");
+             }
+         }
+
+         sb.append("\"warmupIterations\" : ");
+         sb.append(printMultiple(warmupIterationList, "[", "]"));
+         sb.append(",");
+
+         boolean totalHasWarning = warmupForkHasWarning || measurementForkJasWarning || warmupIterationList.contains("t
+         sb.append("\"total\" : " + totalHasWarning);
+
+         return sb.toString();
+     }
+
      private String emitParams(BenchmarkParams params) {
          StringBuilder sb = new StringBuilder();
          boolean isFirst = true;
              }
              switch (c) {
                  // use & as escape character to escape the tidying
-                 case '&': sb.append("&&"); break;
+                 case '&':
+                     sb.append("&&");
+                     break;
                  // we cannot escape to \\\\ since this would create sequences interpreted by the tidying
-                 case '\\': sb.append("&/"); break;
-                 case '"': sb.append("&'"); break;
+                 case '\\':
+                     sb.append("&/");
+                     break;
+                 case '"':
+                     sb.append("&'");
+                     break;
                  // escape spacial chars for the tidying formatting below that might appear in a string
-                 case ',': sb.append(";"); break;
-                 case '[': sb.append("<"); break;
-                 case ']': sb.append(">"); break;
```

```
-                 case '<': sb.append("&-"); break;
-                 case '>': sb.append("&="); break;
-                 case ';': sb.append("&:"); break;
-                 case '{': sb.append("&("); break;
-                 case '}': sb.append("&)"); break;
-                 default: sb.append(c);
+                 case ',':
+                     sb.append(";");
+                     break;
+                 case '[':
+                     sb.append("<");
+                     break;
+                 case ']':
+                     sb.append(">");
+                     break;
+                 case '<':
+                     sb.append("&-");
+                     break;
+                 case '>':
+                     sb.append("&=");
+                     break;
+                 case ';':
+                     sb.append("&:");
+                     break;
+                 case '{':
+                     sb.append("&(");
+                     break;
+                 case '}':
+                     sb.append("&)");
+                     break;
+                 default:
+                     sb.append(c);
                 }
             }
             sb.append("\"");
             }
         }

+     private static List<String> toListOfStrings(List<Double> doubles) {
+         List<String> strings = new ArrayList<>();
+         for (Double threshold : doubles) {
+             if (threshold == null) {
+                 strings.add(null);
+             } else {
+                 strings.add(threshold.toString());
+             }
+         }
+         return strings;
+     }
   }
```

13 jmh-core/src/main/java/org/openjdk/jmh/runner/BaseRunner.java

```
  import org.openjdk.jmh.annotations.Mode;
  import org.openjdk.jmh.infra.BenchmarkParams;
  import org.openjdk.jmh.infra.IterationParams;
+ import org.openjdk.jmh.reconfigure.manager.IterationReconfigureManager;
  import org.openjdk.jmh.results.BenchmarkResult;
  import org.openjdk.jmh.results.BenchmarkResultMetaData;
  import org.openjdk.jmh.results.IterationResult;
  import org.openjdk.jmh.util.Multimap;
  import org.openjdk.jmh.util.TreeMultimap;
  import org.openjdk.jmh.util.Utils;
- import org.openjdk.jmh.util.Version;

  import java.lang.management.GarbageCollectorMXBean;
  import java.lang.management.ManagementFactory;
          long allWarmup = 0;
          long allMeasurement = 0;

```

```
+        IterationReconfigureManager irm = new IterationReconfigureManager(benchParams, out);
+
         // warmup
         IterationParams wp = benchParams.getWarmup();
         for (int i = 1; i <= wp.getCount(); i++) {
             out.iterationResult(benchParams, wp, i, ir);

             allWarmup += ir.getMetadata().getAllOps();
+
+            if (benchParams.getMode().equals(Mode.Reconfigure)) {
+                irm.addWarmupIteration(i, ir);
+                if (irm.checkWarmupIterationThreshold()) {
+                    break;
+                }
+            }
         }

         long measurementTime = System.currentTimeMillis();

         BenchmarkResultMetaData md = new BenchmarkResultMetaData(
                 warmupTime, measurementTime, stopTime,
-                allWarmup, allMeasurement);
+                allWarmup, allMeasurement, irm.getWarmupThresholds(), irm.hasAtLeastOneWarning());

         if (acceptor != null) {
             acceptor.acceptMeta(md);
```

118 jmh-core/src/main/java/org/openjdk/jmh/runner/BenchmarkListEntry.java

```
  package org.openjdk.jmh.runner;

  import org.openjdk.jmh.annotations.Mode;
+ import org.openjdk.jmh.annotations.ReconfigureMode;
  import org.openjdk.jmh.runner.options.TimeValue;
  import org.openjdk.jmh.util.Optional;
  import org.openjdk.jmh.util.lines.TestLineReader;
      private final Optional<Collection<String>> threadGroupLabels;
      private final Optional<Integer> threads;
      private final Optional<Integer> warmupIterations;
+     private final Optional<Integer> minWarmupIterations;
      private final Optional<TimeValue> warmupTime;
      private final Optional<Integer> warmupBatchSize;
      private final Optional<Integer> measurementIterations;
      private final Optional<TimeValue> measurementTime;
      private final Optional<Integer> measurementBatchSize;
      private final Optional<Integer> forks;
+     private final Optional<Integer> minForks;
      private final Optional<Integer> warmupForks;
+     private final Optional<Integer> minWarmupForks;
+     private final ReconfigureMode reconfigureMode;
+     private final Optional<Double> reconfigureCovThreshold;
+     private final Optional<Double> reconfigureCiThreshold;
+     private final Optional<Double> reconfigureKldThreshold;
      private final Optional<String> jvm;
      private final Optional<Collection<String>> jvmArgs;
      private final Optional<Collection<String>> jvmArgsPrepend;

      public BenchmarkListEntry(String userClassQName, String generatedClassQName, String method, Mode mode,
                                Optional<Integer> threads, int[] threadGroups, Optional<Collection<String>> threadGroupL
-                               Optional<Integer> warmupIterations, Optional<TimeValue> warmupTime, Optional<Integer> wa
+                               Optional<Integer> warmupIterations, Optional<Integer> minWarmupIterations, Optional<Time
                                Optional<Integer> measurementIterations, Optional<TimeValue> measurementTime, Optional<I
-                               Optional<Integer> forks, Optional<Integer> warmupForks,
+                               Optional<Integer> forks, Optional<Integer> minForks, Optional<Integer> warmupForks, Opti
+                               ReconfigureMode reconfigureMode, Optional<Double> reconfigureCovThreshold, Optional<Doub
                                Optional<String> jvm, Optional<Collection<String>> jvmArgs, Optional<Collection<String>>
                                Optional<Map<String, String[]>> params, Optional<TimeUnit> tu, Optional<Integer> opsPerI
                                Optional<TimeValue> timeout) {
          this.threads = threads;
          this.threadGroupLabels = threadGroupLabels;
```

```
81  90          this.warmupIterations = warmupIterations;
    91  +       this.minWarmupIterations = minWarmupIterations;
82  92          this.warmupTime = warmupTime;
83  93          this.warmupBatchSize = warmupBatchSize;
84  94          this.measurementIterations = measurementIterations;
85  95          this.measurementTime = measurementTime;
86  96          this.measurementBatchSize = measurementBatchSize;
87  97          this.forks = forks;
    98  +       this.minForks = minForks;
88  99          this.warmupForks = warmupForks;
    100 +       this.minWarmupForks = minWarmupForks;
    101 +       this.reconfigureMode = reconfigureMode;
    102 +       this.reconfigureCovThreshold = reconfigureCovThreshold;
    103 +       this.reconfigureCiThreshold = reconfigureCiThreshold;
    104 +       this.reconfigureKldThreshold = reconfigureKldThreshold;
89  105         this.jvm = jvm;
90  106         this.jvmArgs = jvmArgs;
91  107         this.jvmArgsPrepend = jvmArgsPrepend;
106 122             throw new IllegalStateException("Unable to parse the line: " + line);
107 123         }
108 124
109     -       this.userClassQName         = reader.nextString();
110     -       this.generatedClassQName    = reader.nextString();
111     -       this.method                 = reader.nextString();
112     -       this.mode                   = Mode.deepValueOf(reader.nextString());
113     -       this.threads                = reader.nextOptionalInt();
114     -       this.threadGroups           = reader.nextIntArray();
115     -       this.threadGroupLabels      = reader.nextOptionalStringCollection();
116     -       this.warmupIterations       = reader.nextOptionalInt();
117     -       this.warmupTime             = reader.nextOptionalTimeValue();
118     -       this.warmupBatchSize        = reader.nextOptionalInt();
119     -       this.measurementIterations  = reader.nextOptionalInt();
120     -       this.measurementTime        = reader.nextOptionalTimeValue();
121     -       this.measurementBatchSize   = reader.nextOptionalInt();
122     -       this.forks                  = reader.nextOptionalInt();
123     -       this.warmupForks            = reader.nextOptionalInt();
124     -       this.jvm                    = reader.nextOptionalString();
125     -       this.jvmArgs                = reader.nextOptionalStringCollection();
126     -       this.jvmArgsPrepend         = reader.nextOptionalStringCollection();
127     -       this.jvmArgsAppend          = reader.nextOptionalStringCollection();
128     -       this.params                 = reader.nextOptionalParamCollection();
129     -       this.tu                     = reader.nextOptionalTimeUnit();
130     -       this.opsPerInvocation       = reader.nextOptionalInt();
131     -       this.timeout                = reader.nextOptionalTimeValue();
    125 +       this.userClassQName             = reader.nextString();
    126 +       this.generatedClassQName        = reader.nextString();
    127 +       this.method                     = reader.nextString();
    128 +       this.mode                       = Mode.deepValueOf(reader.nextString());
    129 +       this.threads                    = reader.nextOptionalInt();
    130 +       this.threadGroups               = reader.nextIntArray();
    131 +       this.threadGroupLabels          = reader.nextOptionalStringCollection();
    132 +       this.warmupIterations           = reader.nextOptionalInt();
    133 +       this.minWarmupIterations        = reader.nextOptionalInt();
    134 +       this.warmupTime                 = reader.nextOptionalTimeValue();
    135 +       this.warmupBatchSize            = reader.nextOptionalInt();
    136 +       this.measurementIterations      = reader.nextOptionalInt();
    137 +       this.measurementTime            = reader.nextOptionalTimeValue();
    138 +       this.measurementBatchSize       = reader.nextOptionalInt();
    139 +       this.forks                      = reader.nextOptionalInt();
    140 +       this.minForks                   = reader.nextOptionalInt();
    141 +       this.warmupForks                = reader.nextOptionalInt();
    142 +       this.minWarmupForks             = reader.nextOptionalInt();
    143 +       this.reconfigureMode            = ReconfigureMode.deepValueOf(reader.nextString());
    144 +       this.reconfigureCovThreshold    = reader.nextOptionalDouble();
    145 +       this.reconfigureCiThreshold     = reader.nextOptionalDouble();
    146 +       this.reconfigureKldThreshold    = reader.nextOptionalDouble();
    147 +       this.jvm                        = reader.nextOptionalString();
    148 +       this.jvmArgs                    = reader.nextOptionalStringCollection();
    149 +       this.jvmArgsPrepend             = reader.nextOptionalStringCollection();
    150 +       this.jvmArgsAppend              = reader.nextOptionalStringCollection();
```

```
+        this.params                     = reader.nextOptionalParamCollection();
+        this.tu                         = reader.nextOptionalTimeUnit();
+        this.opsPerInvocation           = reader.nextOptionalInt();
+        this.timeout                    = reader.nextOptionalTimeValue();
     }

     public String toLine() {
         writer.putIntArray(threadGroups);
         writer.putOptionalStringCollection(threadGroupLabels);
         writer.putOptionalInt(warmupIterations);
+        writer.putOptionalInt(minWarmupIterations);
         writer.putOptionalTimeValue(warmupTime);
         writer.putOptionalInt(warmupBatchSize);
         writer.putOptionalInt(measurementIterations);
         writer.putOptionalTimeValue(measurementTime);
         writer.putOptionalInt(measurementBatchSize);
         writer.putOptionalInt(forks);
+        writer.putOptionalInt(minForks);
         writer.putOptionalInt(warmupForks);
+        writer.putOptionalInt(minWarmupForks);
+        writer.putString(reconfigureMode.toString());
+        writer.putOptionalDouble(reconfigureCovThreshold);
+        writer.putOptionalDouble(reconfigureCiThreshold);
+        writer.putOptionalDouble(reconfigureKldThreshold);
         writer.putOptionalString(jvm);
         writer.putOptionalStringCollection(jvmArgs);
         writer.putOptionalStringCollection(jvmArgsPrepend);
     public BenchmarkListEntry cloneWith(Mode mode) {
         return new BenchmarkListEntry(userClassQName, generatedClassQName, method, mode,
                 threads, threadGroups, threadGroupLabels,
-                warmupIterations, warmupTime, warmupBatchSize,
+                minWarmupIterations, warmupIterations, warmupTime, warmupBatchSize,
                 measurementIterations, measurementTime, measurementBatchSize,
-                forks, warmupForks,
+                forks, minForks, warmupForks, minWarmupForks,
+                reconfigureMode, reconfigureCovThreshold, reconfigureCiThreshold,  reconfigureKldThreshold,
                 jvm, jvmArgs, jvmArgsPrepend, jvmArgsAppend,
                 params, tu, opsPerInvocation,
                 timeout);
     public BenchmarkListEntry cloneWith(WorkloadParams p) {
         BenchmarkListEntry br = new BenchmarkListEntry(userClassQName, generatedClassQName, method, mode,
                 threads, threadGroups, threadGroupLabels,
-                warmupIterations, warmupTime, warmupBatchSize,
+                minWarmupIterations, warmupIterations, warmupTime, warmupBatchSize,
                 measurementIterations, measurementTime, measurementBatchSize,
-                forks, warmupForks,
+                forks, minForks, warmupForks, minWarmupForks,
+                reconfigureMode, reconfigureCovThreshold, reconfigureCiThreshold,  reconfigureKldThreshold,
                 jvm, jvmArgs, jvmArgsPrepend, jvmArgsAppend,
                 params, tu, opsPerInvocation,
                 timeout);
         return warmupIterations;
     }

+    public Optional<Integer> getMinWarmupIterations() {
+        return minWarmupIterations;
+    }
+
     public Optional<Integer> getWarmupBatchSize() {
         return warmupBatchSize;
     }
         return forks;
     }

+    public Optional<Integer> getMinForks() {
+        return minForks;
+    }
+
     public Optional<Integer> getWarmupForks() {
         return warmupForks;
```

```
294 334         }
295 335
    336 +       public Optional<Integer> getMinWarmupForks() {
    337 +           return minWarmupForks;
    338 +       }
    339 +
    340 +       public ReconfigureMode getReconfigureMode() {
    341 +           return reconfigureMode;
    342 +       }
    343 +
    344 +       public Optional<Double> getReconfigureCovThreshold() {
    345 +           return reconfigureCovThreshold;
    346 +       }
    347 +
    348 +       public Optional<Double> getReconfigureCiThreshold() {
    349 +           return reconfigureCiThreshold;
    350 +       }
    351 +
    352 +       public Optional<Double> getReconfigureKldThreshold() {
    353 +           return reconfigureKldThreshold;
    354 +       }
    355 +
296 356         public Optional<String> getJvm() {
297 357             return jvm;
298 358         }
```

48 jmh-core/src/main/java/org/openjdk/jmh/runner/Defaults.java

```
25  25  package org.openjdk.jmh.runner;
26  26
27  27  import org.openjdk.jmh.annotations.Mode;
    28 + import org.openjdk.jmh.annotations.ReconfigureMode;
28  29  import org.openjdk.jmh.results.format.ResultFormatType;
29  30  import org.openjdk.jmh.runner.options.TimeValue;
30  31  import org.openjdk.jmh.runner.options.VerboseMode;
40  41      /**
41  42       * Number of warmup iterations.
42  43       */
43     -    public static final int WARMUP_ITERATIONS = 5;
    44 +    public static final int WARMUP_ITERATIONS = 50;
    45 +
    46 +    /**
    47 +     * Minimum number of warmup iterations.
    48 +     */
    49 +    public static final int MIN_WARMUP_ITERATIONS = 5;
    50 +
    51 +    /**
    52 +     * Minimum number of warmup iterations for divergence mode.
    53 +     */
    54 +    public static final int MIN_WARMUP_ITERATIONS_DIVERGENCE = 6;
44  55
45  56      /**
46  57       * Number of warmup iterations in {@link org.openjdk.jmh.annotations.Mode#SingleShotTime} mode.
55  66      /**
56  67       * The duration of warmup iterations.
57  68       */
58     -    public static final TimeValue WARMUP_TIME = TimeValue.seconds(10);
    69 +    public static final TimeValue WARMUP_TIME = TimeValue.seconds(1);
59  70
60  71      /**
61  72       * Number of measurement iterations.
62  73       */
63     -    public static final int MEASUREMENT_ITERATIONS = 5;
    74 +    public static final int MEASUREMENT_ITERATIONS = 50;
64  75
65  76      /**
66  77       * Number of measurement iterations in {@link org.openjdk.jmh.annotations.Mode#SingleShotTime} mode.
75  86      /**
76  87       * The duration of measurement iterations.
77  88       */
```

```
78      -     public static final TimeValue MEASUREMENT_TIME = TimeValue.seconds(10);
        89    +     public static final TimeValue MEASUREMENT_TIME = TimeValue.seconds(1);
79      90
80      91          /**
81      92           * Number of measurement threads.
87      98           */
88      99          public static final int MEASUREMENT_FORKS = 5;
89      100
        101   +     /**
        102   +      * Minimum number of forks in which we measure the workload.
        103   +      */
        104   +     public static final int MIN_MEASUREMENT_FORKS = 2;
        105   +
90      106         /**
91      107          * Number of warmup forks we discard.
92      108          */
93      109         public static final int WARMUP_FORKS = 0;
94      110
        111   +     /**
        112   +      * Minimum number of warmup forks we discard.
        113   +      */
        114   +     public static final int MIN_WARMUP_FORKS = 0;
        115   +
95      116         /**
96      117          * Should JMH fail on benchmark error?
97      118          */
152     173          */
153     174         public static final String INCLUDE_BENCHMARKS = ".*";
154     175
        176   +     /**
        177   +      * Default reconfigure mode.
        178   +      */
        179   +     public static final ReconfigureMode RECONFIGURE_MODE = ReconfigureMode.DIVERGENCE;
        180   +
        181   +     /**
        182   +      * coefficient of variation variability threshold.
        183   +      */
        184   +     public static final double RECONFIGURE_COV_THRESHOLD = 0.01;
        185   +
        186   +     /**
        187   +      * confidence interval variability threshold.
        188   +      */
        189   +     public static final double RECONFIGURE_CI_THRESHOLD = 0.03;
        190   +
        191   +     /**
        192   +      * p value of kullback leibler divergence as variability threshold.
        193   +      */
        194   +     public static final double RECONFIGURE_KLD_THRESHOLD = 0.99;
155     195     }
```

159 jmh-core/src/main/java/org/openjdk/jmh/runner/Runner.java

Load diff

Large diffs are not rendered by default.

2 jmh-core/src/main/java/org/openjdk/jmh/runner/format/OutputFormat.java

```
79      79          * Format for end-of-benchmark.
80      80          * @param result benchmark results
81      81          */
82          -     void endRun(Collection<RunResult> result);
        82    +     void endRun(Collection<RunResult> result, boolean atLeastOneWarning);
83      83
84      84          /* ------------- RAW OUTPUT METHODS ------------------- */
85      85
```

2 jmh-core/src/main/java/org/openjdk/jmh/runner/format/SilentFormat.java

```
48  48      }
49  49
50  50      @Override
51      -   public void endRun(Collection<RunResult> results) {
    51  +   public void endRun(Collection<RunResult> results, boolean atLeastOneWarning) {
52  52      }
53  53
54  54      @Override
```

22 jmh-core/src/main/java/org/openjdk/jmh/runner/format/TextReportFormat.java

```
25  25  package org.openjdk.jmh.runner.format;
26  26
27  27  import org.openjdk.jmh.annotations.Mode;
    28  + import org.openjdk.jmh.annotations.ReconfigureMode;
28  29  import org.openjdk.jmh.infra.BenchmarkParams;
29  30  import org.openjdk.jmh.infra.IterationParams;
30  31  import org.openjdk.jmh.results.BenchmarkResult;
139 140             }
140 141             out.println("# Parameters: (" + s + ")");
141 142         }
    143 +
    144 +       if(params.getMode().equals(Mode.Reconfigure)){
    145 +           out.println("# Reconfigure mode: " + params.getReconfigureMode().longLabel());
    146 +
    147 +           if(params.getReconfigureMode().equals(ReconfigureMode.COV)){
    148 +               out.println("# COV threshold: " + params.getReconfigureCovThreshold());
    149 +           }else if(params.getReconfigureMode().equals(ReconfigureMode.CI)){
    150 +               out.println("# CI threshold: " + params.getReconfigureCiThreshold());
    151 +           }else{
    152 +               out.println("# Kullback leibler divergence p value threshold: " + params.getReconfigureKldThreshold())
    153 +           }
    154 +       }
142 155     }
143 156
144 157     @Override
230 243     }
231 244
232 245     @Override
233     -   public void endRun(Collection<RunResult> runResults) {
    246 +   public void endRun(Collection<RunResult> runResults, boolean atLeastOneWarning) {
234 247         out.println("REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on");
235 248         out.println("why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial");
236 249         out.println("experiments, perform baseline and negative tests that provide experimental control, make sure");
237 250         out.println("the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
238 251         out.println("Do not assume the numbers tell you what you want them to tell.");
239 252         out.println("");
240 253
    254 +       if (atLeastOneWarning) {
    255 +           out.println("##########");
    256 +           out.println("At least one warning exists from the dynamic reconfiguration!");
    257 +           out.println("##########");
    258 +           out.println("");
    259 +       }
    260 +
241 261         ResultFormatFactory.getInstance(ResultFormatType.TEXT, out).writeOut(runResults);
242 262     }
243 263
```

63 jmh-core/src/main/java/org/openjdk/jmh/runner/options/ChainedOptionsBuilder.java

```
25  25  package org.openjdk.jmh.runner.options;
26  26
27  27  import org.openjdk.jmh.annotations.Mode;
    28  + import org.openjdk.jmh.annotations.ReconfigureMode;
28  29  import org.openjdk.jmh.profile.Profiler;
29  30  import org.openjdk.jmh.results.format.ResultFormatType;
30  31
```

```
179 180         */
180 181        ChainedOptionsBuilder warmupIterations(int value);
181 182
    183 +      /**
    184 +       * How many warmup iterations to do at least?
    185 +       * @param value flag
    186 +       * @return builder
    187 +       * @see org.openjdk.jmh.annotations.Warmup
    188 +       * @see org.openjdk.jmh.runner.Defaults#MIN_WARMUP_ITERATIONS
    189 +       */
    190 +      ChainedOptionsBuilder minWarmupIterations(int value);
    191 +
182 192        /**
183 193         * How large warmup batchSize should be?
184 194         * @param value batch size
278 288         */
279 289        ChainedOptionsBuilder forks(int value);
280 290
    291 +      /**
    292 +       * Minimum number of forks to use in the run
    293 +       * @param value minimum number of forks
    294 +       * @return builder
    295 +       * @see org.openjdk.jmh.annotations.Fork
    296 +       * @see org.openjdk.jmh.runner.Defaults#MIN_MEASUREMENT_FORKS
    297 +       */
    298 +      ChainedOptionsBuilder minForks(int value);
    299 +
281 300        /**
282 301         * Number of ignored forks
283 302         * @param value number of ignored forks
287 306         */
288 307        ChainedOptionsBuilder warmupForks(int value);
289 308
    309 +      /**
    310 +       * Minimum number of ignored forks
    311 +       * @param value minimum value number of ignored forks
    312 +       * @return builder
    313 +       * @see org.openjdk.jmh.annotations.Fork
    314 +       * @see org.openjdk.jmh.runner.Defaults#MIN_WARMUP_FORKS
    315 +       */
    316 +      ChainedOptionsBuilder minWarmupForks(int value);
    317 +
    318 +      /**
    319 +       * Reconfigure mode
    320 +       * @param mode reconfigure mode
    321 +       * @return builder
    322 +       * @see org.openjdk.jmh.annotations.Reconfigure
    323 +       * @see org.openjdk.jmh.runner.Defaults#RECONFIGURE_MODE
    324 +       */
    325 +      ChainedOptionsBuilder reconfigureMode(ReconfigureMode mode);
    326 +
    327 +      /**
    328 +       * coefficient of variation variability threshold
    329 +       * @param value threshold
    330 +       * @return builder
    331 +       * @see org.openjdk.jmh.annotations.Reconfigure
    332 +       * @see org.openjdk.jmh.runner.Defaults#RECONFIGURE_COV_THRESHOLD
    333 +       */
    334 +      ChainedOptionsBuilder reconfigureCovThreshold(double value);
    335 +
    336 +      /**
    337 +       * confidence interval variability threshold
    338 +       * @param value threshold
    339 +       * @return builder
    340 +       * @see org.openjdk.jmh.annotations.Reconfigure
    341 +       * @see org.openjdk.jmh.runner.Defaults#RECONFIGURE_CI_THRESHOLD
    342 +       */
    343 +      ChainedOptionsBuilder reconfigureCiThreshold(double value);
    344 +
    345 +      /**
```

```
+      * p value of kullback leibler divergence as variability threshold
+      * @param value threshold
+      * @return builder
+      * @see org.openjdk.jmh.annotations.Reconfigure
+      * @see org.openjdk.jmh.runner.Defaults#RECONFIGURE_KLD_THRESHOLD
+      */
+     ChainedOptionsBuilder reconfigureKldThreshold(double value);
       /**
        * Forked JVM to use.
        *
```

89 jmh-core/src/main/java/org/openjdk/jmh/runner/options/CommandLineOptions.java

```
 import joptsimple.*;
 import org.openjdk.jmh.annotations.Mode;
+import org.openjdk.jmh.annotations.ReconfigureMode;
 import org.openjdk.jmh.profile.ProfilerFactory;
 import org.openjdk.jmh.results.format.ResultFormatType;
 import org.openjdk.jmh.runner.Defaults;
     private final Optional<TimeValue> runTime;
     private final Optional<Integer> batchSize;
     private final Optional<Integer> warmupIterations;
+    private final Optional<Integer> minWarmupIterations;
     private final Optional<TimeValue> warmupTime;
     private final Optional<Integer> warmupBatchSize;
     private final List<Mode> benchMode = new ArrayList<>();
     private final Optional<Integer> opsPerInvocation;
     private final List<String> regexps = new ArrayList<>();
     private final Optional<Integer> fork;
+    private final Optional<Integer> minFork;
     private final Optional<Integer> warmupFork;
+    private final Optional<Integer> minWarmupFork;
+    private final Optional<ReconfigureMode> reconfigureMode;
+    private final Optional<Double> reconfigureCovThreshold;
+    private final Optional<Double> reconfigureCiThreshold;
+    private final Optional<Double> reconfigureKldThreshold;
     private final Optional<String> output;
     private final Optional<String> result;
     private final Optional<ResultFormatType> resultFormat;
                 Defaults.WARMUP_ITERATIONS + " for all other modes)")
                 .withRequiredArg().withValuesConvertedBy(IntegerValueConverter.NON_NEGATIVE).describedAs("int");

+        OptionSpec<Integer> optMinWarmupCount = parser.accepts("mwi", "Minimum number of warmup iterations to do. Warm
+                "iterations are not counted towards the benchmark score. " +
+                "(default: " + Defaults.MIN_WARMUP_ITERATIONS + ")")
+                .withRequiredArg().withValuesConvertedBy(IntegerValueConverter.NON_NEGATIVE).describedAs("int");
+
         OptionSpec<Integer> optWarmupBatchSize = parser.accepts("wbs", "Warmup batch size: number of benchmark " +
                 "method calls per operation. Some benchmark modes may ignore this setting. " +
                 "(default: " + Defaults.WARMUP_BATCHSIZE + ")")
                 "(default: " + Defaults.MEASUREMENT_FORKS + ")")
                 .withRequiredArg().withValuesConvertedBy(IntegerValueConverter.NON_NEGATIVE).describedAs("int");

+        OptionSpec<Integer> optMinForks = parser.accepts("mf", "How many times to fork a single benchmark at least." +
+                "(default: " + Defaults.MIN_MEASUREMENT_FORKS + ")")
+                .withRequiredArg().withValuesConvertedBy(IntegerValueConverter.NON_NEGATIVE).describedAs("int");
+
         OptionSpec<Integer> optWarmupForks = parser.accepts("wf", "How many warmup forks to make for a single benchmar
                 "All iterations within the warmup fork are not counted towards the benchmark score. Use 0 to disable "
                 "warmup forks. " +
                 "(default: " + Defaults.WARMUP_FORKS + ")")
                 .withRequiredArg().withValuesConvertedBy(IntegerValueConverter.NON_NEGATIVE).describedAs("int");

+        OptionSpec<Integer> optMinWarmupForks = parser.accepts("mwf", "How many warmup forks to make for a single benc
+                "(default: " + Defaults.MIN_WARMUP_FORKS + ")")
+                .withRequiredArg().withValuesConvertedBy(IntegerValueConverter.NON_NEGATIVE).describedAs("int");
+
+        OptionSpec<String> optReconfigureMode = parser.accepts("rm", "Reconfigure mode. Available modes are: " + Recon
+                "(default: " + Defaults.RECONFIGURE_MODE + ")")
```

```
+                .withRequiredArg().ofType(String.class).withValuesSeparatedBy(',').describedAs("mode");
+
+        OptionSpec<Double> optReconfigureCovThreshold = parser.accepts("rcov", "coefficient of variation variability t
+                "(default: " + Defaults.RECONFIGURE_COV_THRESHOLD + ")")
+                .withRequiredArg().withValuesConvertedBy(DoubleValueConverter.NON_NEGATIVE).describedAs("double");
+
+        OptionSpec<Double> optReconfigureCiThreshold = parser.accepts("rci", "confidence interval variability threshol
+                "(default: " + Defaults.RECONFIGURE_CI_THRESHOLD + ")")
+                .withRequiredArg().withValuesConvertedBy(DoubleValueConverter.NON_NEGATIVE).describedAs("double");
+
+        OptionSpec<Double> optReconfigureKldThreshold = parser.accepts("rkld", "p value of kullback leibler divergence
+                "(default: " + Defaults.RECONFIGURE_KLD_THRESHOLD + ")")
+                .withRequiredArg().withValuesConvertedBy(DoubleValueConverter.PROBABILITY).describedAs("double");
+
         OptionSpec<String> optOutput = parser.accepts("o", "Redirect human-readable output to a given file.")
                 .withRequiredArg().ofType(String.class).describedAs("filename");

             batchSize = toOptional(optMeasureBatchSize, set);
             runTime = toOptional(optMeasureTime, set);
             warmupIterations = toOptional(optWarmupCount, set);
+            minWarmupIterations = toOptional(optMinWarmupCount, set);
             warmupBatchSize = toOptional(optWarmupBatchSize, set);
             warmupTime = toOptional(optWarmupTime, set);
             timeout = toOptional(optTimeoutTime, set);
             gcEachIteration = toOptional(optGC, set);
             failOnError = toOptional(optFOE, set);
             fork = toOptional(optForks, set);
+            minFork = toOptional(optMinForks, set);
             warmupFork = toOptional(optWarmupForks, set);
+            minWarmupFork = toOptional(optMinWarmupForks, set);
             output = toOptional(optOutput, set);
             result = toOptional(optOutputResults, set);

+            if (set.has(optReconfigureMode)) {
+                try {
+                    reconfigureMode = Optional.of(ReconfigureMode.deepValueOf(optReconfigureMode.value(set)));
+                } catch (IllegalArgumentException iae) {
+                    throw new CommandLineOptionException(iae.getMessage(), iae);
+                }
+            } else {
+                reconfigureMode = Optional.none();
+            }
+
+            reconfigureCovThreshold = toOptional(optReconfigureCovThreshold, set);
+            reconfigureCiThreshold = toOptional(optReconfigureCiThreshold, set);
+            reconfigureKldThreshold = toOptional(optReconfigureKldThreshold, set);
+
             if (set.has(optBenchmarkMode)) {
                 try {
                     List<Mode> modes = new ArrayList<>();
         return fork;
     }

+    @Override
+    public Optional<Integer> getMinForkCount() {
+        return minFork;
+    }
+
     @Override
     public Optional<Integer> getWarmupForkCount() {
         return warmupFork;
     }

+    @Override
+    public Optional<Integer> getMinWarmupForkCount() {
+        return minWarmupFork;
+    }
+
     @Override
     public Optional<String> getOutput() {
```

```
558  622            return output;
593  657            return warmupIterations;
594  658        }
595  659
     660 +      @Override
     661 +      public Optional<Integer> getMinWarmupIterations() {
     662 +          return minWarmupIterations;
     663 +      }
     664 +
596  665        @Override
597  666        public Optional<Integer> getWarmupBatchSize() {
598  667            return warmupBatchSize;
599  668        }
600  669
     670 +      @Override
     671 +      public Optional<ReconfigureMode> getReconfigureMode() {
     672 +          return reconfigureMode;
     673 +      }
     674 +
     675 +      @Override
     676 +      public Optional<Double> getReconfigureCovThreshold() {
     677 +          return reconfigureCovThreshold;
     678 +      }
     679 +
     680 +      @Override
     681 +      public Optional<Double> getReconfigureCiThreshold() {
     682 +          return reconfigureCiThreshold;
     683 +      }
     684 +
     685 +      @Override
     686 +      public Optional<Double> getReconfigureKldThreshold() {
     687 +          return reconfigureKldThreshold;
     688 +      }
     689 +
601  690        @Override
602  691        public Optional<Integer> getThreads() {
603  692            return threads;
```

48 jmh-core/src/main/java/org/openjdk/jmh/runner/options/DoubleValueConverter.java

```
...  ...  @@ -0,0 +1,48 @@
     1  + package org.openjdk.jmh.runner.options;
     2  +
     3  + import joptsimple.ValueConversionException;
     4  + import joptsimple.ValueConverter;
     5  + import joptsimple.internal.Reflection;
     6  +
     7  + /**
     8  +  * Converts option value from {@link String} to {@link Double} and makes sure the value exceeds given minimal and maxi
     9  +  */
     10 + public class DoubleValueConverter implements ValueConverter<Double> {
     11 +     private final static ValueConverter<Double> TO_DOUBLE_CONVERTER = Reflection.findConverter(double.class);
     12 +
     13 +     public final static DoubleValueConverter PROBABILITY = new DoubleValueConverter(0, 1);
     14 +     public final static DoubleValueConverter NON_NEGATIVE = new DoubleValueConverter(0, Double.MAX_VALUE);
     15 +
     16 +     private final double minValue;
     17 +     private final double maxValue;
     18 +
     19 +     public DoubleValueConverter(double minValue, double maxValue) {
     20 +         this.minValue = minValue;
     21 +         this.maxValue = maxValue;
     22 +     }
     23 +
     24 +     @Override
     25 +     public Double convert(String value) {
     26 +         Double newValue = TO_DOUBLE_CONVERTER.convert(value);
     27 +         if (newValue == null) {
     28 +             // should not get here
     29 +             throw new ValueConversionException("value should not be null");
```

```
+         }
+
+         if (newValue < minValue || newValue > maxValue) {
+             String message = "The given value " + value + " should be greater or equal than " + minValue + " and less
+             throw new ValueConversionException(message);
+         }
+         return newValue;
+     }
+
+     @Override
+     public Class<Double> valueType() {
+         return TO_DOUBLE_CONVERTER.valueType();
+     }
+
+     @Override
+     public String valuePattern() {
+         return "double";
+     }
+ }
```

50 jmh-core/src/main/java/org/openjdk/jmh/runner/options/Options.java

```
 package org.openjdk.jmh.runner.options;

 import org.openjdk.jmh.annotations.Mode;
+ import org.openjdk.jmh.annotations.ReconfigureMode;
 import org.openjdk.jmh.results.format.ResultFormatType;
 import org.openjdk.jmh.util.Optional;

      */
     Optional<Integer> getWarmupIterations();

+     /**
+      * Minimum number of warmup iterations
+      * @return minimum number of warmup iterations
+      * @see org.openjdk.jmh.annotations.Warmup
+      */
+     Optional<Integer> getMinWarmupIterations();
+
     /**
      * The duration for warmup iterations
      * @return duration
      */
     Optional<Integer> getForkCount();

+     /**
+      * Minimum fork count
+      * @return minimum fork count
+      * @see org.openjdk.jmh.annotations.Fork
+      */
+     Optional<Integer> getMinForkCount();
+
     /**
      * Number of initial forks to ignore the results for
      * @return initial fork count; 0, to disable
      * @see org.openjdk.jmh.annotations.Fork
      */
     Optional<Integer> getWarmupForkCount();

+     /**
+      * Minimum number of initial forks to ignore the results for
+      * @return minimum initial fork count
+      * @see org.openjdk.jmh.annotations.Fork
+      */
+     Optional<Integer> getMinWarmupForkCount();
+
+     /**
+      * reconfigure mode
+      * @return reconfigure mode
+      * @see org.openjdk.jmh.annotations.ReconfigureMode
```

```
+      */
+     Optional<ReconfigureMode> getReconfigureMode();
+
+     /**
+      * coefficient of variation variability threshold
+      * @return threshold
+      * @see org.openjdk.jmh.annotations.Reconfigure
+      */
+     Optional<Double> getReconfigureCovThreshold();
+
+     /**
+      * confidence interval variability threshold
+      * @return threshold
+      * @see org.openjdk.jmh.annotations.Reconfigure
+      */
+     Optional<Double> getReconfigureCiThreshold();
+
+     /**
+      * p value of kullback leibler divergence as variability threshold
+      * @return threshold
+      * @see org.openjdk.jmh.annotations.Reconfigure
+      */
+     Optional<Double> getReconfigureKldThreshold();
+
      /**
       * JVM executable to use for forks
       * @return path to JVM executable
```

162 jmh-core/src/main/java/org/openjdk/jmh/runner/options/OptionsBuilder.java

```
  package org.openjdk.jmh.runner.options;

  import org.openjdk.jmh.annotations.Mode;
+ import org.openjdk.jmh.annotations.ReconfigureMode;
  import org.openjdk.jmh.annotations.Threads;
  import org.openjdk.jmh.profile.Profiler;
  import org.openjdk.jmh.results.format.ResultFormatType;
          throw new IllegalArgumentException(message);
      }

+     private static void checkGreaterOrEqual(double value, double minValue, String s) {
+         if (value >= minValue) {
+             return;
+         }
+         String message = s + " (" + value + ") should be greater or equal than " + minValue;
+         throw new IllegalArgumentException(message);
+     }
+
+     private static void checkLessOrEqual(double value, double minValue, String s) {
+         if (value <= minValue) {
+             return;
+         }
+         String message = s + " (" + value + ") should be less or equal than " + minValue;
+         throw new IllegalArgumentException(message);
+     }
+
      // ------------------------------------------------------------------------

      private final List<String> regexps = new ArrayList<>();

      // ------------------------------------------------------------------------

+     private Optional<Integer> minWarmupIterations = Optional.none();
+
+     @Override
+     public ChainedOptionsBuilder minWarmupIterations(int value) {
+         checkGreaterOrEqual(value, 0, "Minimum warmup iterations");
+         this.minWarmupIterations = Optional.of(value);
+         return this;
+     }
```

```
+
+     @Override
+     public Optional<Integer> getMinWarmupIterations() {
+         if (otherOptions != null) {
+             return minWarmupIterations.orAnother(otherOptions.getMinWarmupIterations());
+         } else {
+             return minWarmupIterations;
+         }
+     }
+
+     // ------------------------------------------------------------------------
+
      private Optional<Integer> warmupBatchSize = Optional.none();

      @Override
          }
      }

-
      // ------------------------------------------------------------------------

      private final EnumSet<Mode> benchModes = EnumSet.noneOf(Mode.class);

      // ------------------------------------------------------------------------

+     private Optional<Integer> minForks = Optional.none();
+
+     @Override
+     public ChainedOptionsBuilder minForks(int value) {
+         checkGreaterOrEqual(value, 0, "Minimum forks");
+         this.minForks = Optional.of(value);
+         return this;
+     }
+
+     @Override
+     public Optional<Integer> getMinForkCount() {
+         if (otherOptions != null) {
+             return minForks.orAnother(otherOptions.getMinForkCount());
+         } else {
+             return minForks;
+         }
+     }
+
+     // ------------------------------------------------------------------------
+
      private Optional<Integer> warmupForks = Optional.none();

      @Override

      // ------------------------------------------------------------------------

+     private Optional<Integer> minWarmupForks = Optional.none();
+
+     @Override
+     public ChainedOptionsBuilder minWarmupForks(int value) {
+         checkGreaterOrEqual(value, 0, "Minimum warmup forks");
+         this.minWarmupForks = Optional.of(value);
+         return this;
+     }
+
+     @Override
+     public Optional<Integer> getMinWarmupForkCount() {
+         if (otherOptions != null) {
+             return minWarmupForks.orAnother(otherOptions.getMinWarmupForkCount());
+         } else {
+             return minWarmupForks;
+         }
+     }
+
+     // ------------------------------------------------------------------------
```

```
+
+     private Optional<ReconfigureMode> reconfigureMode = Optional.none();
+
+     @Override
+     public ChainedOptionsBuilder reconfigureMode(ReconfigureMode value) {
+         if(value.equals(ReconfigureMode.NONE)){
+             throw new IllegalArgumentException("None is as reconfigure mode not allowed");
+         }
+         this.reconfigureMode = Optional.of(value);
+         return this;
+     }
+
+     @Override
+     public Optional<ReconfigureMode> getReconfigureMode() {
+         if (otherOptions != null) {
+             return reconfigureMode.orAnother(otherOptions.getReconfigureMode());
+         } else {
+             return reconfigureMode;
+         }
+     }
+
+     // ------------------------------------------------------------------------
+
+     private Optional<Double> reconfigureCovThreshold = Optional.none();
+
+     @Override
+     public ChainedOptionsBuilder reconfigureCovThreshold(double value) {
+         checkGreaterOrEqual(value, 0, "reconfigure cov threshold");
+         checkLessOrEqual(value, 1, "reconfigure cov threshold");
+         this.reconfigureCovThreshold = Optional.of(value);
+         return this;
+     }
+
+     @Override
+     public Optional<Double> getReconfigureCovThreshold() {
+         if (otherOptions != null) {
+             return reconfigureCovThreshold.orAnother(otherOptions.getReconfigureCovThreshold());
+         } else {
+             return reconfigureCovThreshold;
+         }
+     }
+
+     // ------------------------------------------------------------------------
+
+     private Optional<Double> reconfigureCiThreshold = Optional.none();
+
+     @Override
+     public ChainedOptionsBuilder reconfigureCiThreshold(double value) {
+         checkGreaterOrEqual(value, 0, "reconfigure ci threshold");
+         this.reconfigureCiThreshold = Optional.of(value);
+         return this;
+     }
+
+     @Override
+     public Optional<Double> getReconfigureCiThreshold() {
+         if (otherOptions != null) {
+             return reconfigureCiThreshold.orAnother(otherOptions.getReconfigureCiThreshold());
+         } else {
+             return reconfigureCiThreshold;
+         }
+     }
+
+     // ------------------------------------------------------------------------
+
+     private Optional<Double> reconfigureKldThreshold = Optional.none();
+
+     @Override
+     public ChainedOptionsBuilder reconfigureKldThreshold(double value) {
+         checkGreaterOrEqual(value, 0, "reconfigure kld threshold");
+         checkLessOrEqual(value, 1, "reconfigure kld threshold");
```

```
+         this.reconfigureKldThreshold = Optional.of(value);
+         return this;
+     }
+
+     @Override
+     public Optional<Double> getReconfigureKldThreshold() {
+         if (otherOptions != null) {
+             return reconfigureKldThreshold.orAnother(otherOptions.getReconfigureKldThreshold());
+         } else {
+             return reconfigureKldThreshold;
+         }
+     }
+
+     // ------------------------------------------------------------------------
+
      private Optional<String> jvmBinary = Optional.none();

      @Override
```

1 jmh-core/src/main/java/org/openjdk/jmh/util/lines/Constants.java

```
      public static final char TAG_EMPTY_OPTIONAL     = 'E';
      public static final char TAG_STRING             = 'S';
      public static final char TAG_INT                = 'I';
+     public static final char TAG_DOUBLE             = 'D';
      public static final char TAG_TIMEVALUE          = 'T';
      public static final char TAG_STRING_COLLECTION  = 'L';
      public static final char TAG_INT_ARRAY          = 'A';
```

11 jmh-core/src/main/java/org/openjdk/jmh/util/lines/TestLineReader.java

```
          }
      }

+     public Optional<Double> nextOptionalDouble() {
+         char tag = readChar();
+         if (tag == Constants.TAG_EMPTY_OPTIONAL) {
+             return Optional.none();
+         } else if (tag == TAG_DOUBLE) {
+             return Optional.of(Double.valueOf(readString()));
+         } else {
+             throw error("unexpected tag = " + tag);
+         }
+     }
+
      public Optional<String> nextOptionalString() {
          char tag = readChar();
          if (tag == Constants.TAG_EMPTY_OPTIONAL) {
```

9 jmh-core/src/main/java/org/openjdk/jmh/util/lines/TestLineWriter.java

```
          }
      }

+     public void putOptionalDouble(Optional<Double> opt) {
+         if (!opt.hasValue()) {
+             appendTag(TAG_EMPTY_OPTIONAL);
+         } else {
+             appendTag(TAG_DOUBLE);
+             appendWithLen(String.valueOf(opt.get()));
+         }
+     }
+
      public void putOptionalString(Optional<String> opt) {
          if (!opt.hasValue()) {
              appendTag(TAG_EMPTY_OPTIONAL);
```

BIN +2.58 MB jmh-core/src/main/resources/pa_darwin_amd64

Binary file not shown.

BIN +2.57 MB `jmh-core/src/main/resources/pa_linux_amd64`

Binary file not shown.

BIN +2.67 MB `jmh-core/src/main/resources/pa_windows_amd64.exe`

Binary file not shown.

8 `jmh-core/src/test/java/org/openjdk/jmh/results/ResultAggregationTest.java`

```
57  57          Assert.assertEquals(1, br.getSecondaryResults().get("bench").getSampleCount());
58  58          Assert.assertEquals(2, br.getIterationResults().size());
59  59
60      -       RunResult rr = new RunResult(null, Arrays.asList(br, br));
    60  +       RunResult rr = new RunResult(null, Arrays.asList(br, br), null, null);
61  61          Assert.assertEquals(20000.0, rr.getPrimaryResult().getScore());
62  62          Assert.assertEquals(10000.0, rr.getSecondaryResults().get("sec").getScore());
63  63          Assert.assertEquals(3000.0, rr.getSecondaryResults().get("bench").getScore());
91  91          Assert.assertEquals(1, br.getSecondaryResults().get("bench").getSampleCount());
92  92          Assert.assertEquals(2, br.getIterationResults().size());
93  93
94      -       RunResult rr = new RunResult(null, Arrays.asList(br, br));
    94  +       RunResult rr = new RunResult(null, Arrays.asList(br, br), null, null);
95  95          Assert.assertEquals(10000.0, rr.getPrimaryResult().getScore());
96  96          Assert.assertEquals(5000.0, rr.getSecondaryResults().get("sec").getScore());
97  97          Assert.assertEquals(3000.0, rr.getSecondaryResults().get("bench").getScore());
134 134         Assert.assertEquals(1, br.getSecondaryResults().get("bench").getSampleCount());
135 135         Assert.assertEquals(2, br.getIterationResults().size());
136 136
137     -       RunResult rr = new RunResult(null, Arrays.asList(br, br));
    137 +       RunResult rr = new RunResult(null, Arrays.asList(br, br), null, null);
138 138         Assert.assertEquals(10000.0, rr.getPrimaryResult().getScore());
139 139         Assert.assertEquals(5000.0, rr.getSecondaryResults().get("sec").getScore());
140 140         Assert.assertEquals(3000.0, rr.getSecondaryResults().get("bench").getScore());
168 168         Assert.assertEquals(1, br.getSecondaryResults().get("bench").getSampleCount());
169 169         Assert.assertEquals(2, br.getIterationResults().size());
170 170
171     -       RunResult rr = new RunResult(null, Arrays.asList(br, br));
    171 +       RunResult rr = new RunResult(null, Arrays.asList(br, br), null, null);
172 172         Assert.assertEquals(10000.0, rr.getPrimaryResult().getScore());
173 173         Assert.assertEquals(5000.0, rr.getSecondaryResults().get("sec").getScore());
174 174         Assert.assertEquals(3000.0, rr.getSecondaryResults().get("bench").getScore());
```

12 `jmh-core/src/test/java/org/openjdk/jmh/results/TestAggregateResult.java`

```
52  52          result = new IterationResult(
53  53                  new BenchmarkParams("blah", "blah", false,
54  54                          1, new int[]{1}, Collections.<String>emptyList(),
55      -                       1, 1,
56      -                       new IterationParams(IterationType.WARMUP, 1, TimeValue.seconds(1), 1),
57      -                       new IterationParams(IterationType.MEASUREMENT, 1, TimeValue.seconds(1), 1),
58      -                       Mode.Throughput, null, TimeUnit.SECONDS, 1,
59      -                       Utils.getCurrentJvm(), Collections.<String>emptyList(),
    55  +                       1, 1, 1, 1,
    56  +                       new IterationParams(IterationType.WARMUP, 1, 1, TimeValue.seconds(1), 1),
    57  +                       new IterationParams(IterationType.MEASUREMENT, 1, 1, TimeValue.seconds(1), 1),
    58  +                       Mode.Throughput, null, 1,1,1, null,
    59  +                       TimeUnit.SECONDS, 1, Utils.getCurrentJvm(), Collections.<String>emptyList(),
60  60                          System.getProperty("java.version"), System.getProperty("java.vm.name"), System.getProperty("ja
61  61                          TimeValue.days(1)),
62      -                   new IterationParams(IterationType.MEASUREMENT, 1, TimeValue.days(1), 1),
    62  +                   new IterationParams(IterationType.MEASUREMENT, 1, 1, TimeValue.days(1), 1),
63  63                      null
64  64          );
65  65          for (double d : values) {
```

30 `jmh-core/src/test/java/org/openjdk/jmh/results/format/ResultFormatTest.java`

```
72  72              for (int p = 0; p < 5; p++) {
```

```
73  73                  ps.put("param" + p, "value" + p, p);
74  74              }
    75 +
    76 +            int threads = r.nextInt(1000);
    77 +            int threadGroups = r.nextInt(1000);
    78 +            int forks = r.nextInt(1000);
    79 +            int warmupForks = r.nextInt(1000);
    80 +            int countWarmup = r.nextInt(1000);
    81 +            int warmupTime = r.nextInt(1000);
    82 +            int countMeasurement = r.nextInt(1000);
    83 +            int measurementTime = r.nextInt(1000);
    84 +
75  85              BenchmarkParams params = new BenchmarkParams(
76  86                      "benchmark_" + b,
77  87                      JSONResultFormat.class.getName() + ".benchmark_" + b + "_" + Mode.Throughput,
78  88                      false,
79     -                    r.nextInt(1000),
80     -                    new int[]{ r.nextInt(1000) },
    89 +                    threads,
    90 +                    new int[]{ threadGroups },
81  91                      Collections.<String>emptyList(),
82     -                    r.nextInt(1000),
83     -                    r.nextInt(1000),
84     -                    new IterationParams(IterationType.WARMUP,      r.nextInt(1000), TimeValue.seconds(r.nextInt(1000))
85     -                    new IterationParams(IterationType.MEASUREMENT, r.nextInt(1000), TimeValue.seconds(r.nextInt(1000))
    92 +                    forks,
    93 +                    forks,
    94 +                    warmupForks,
    95 +                    warmupForks,
    96 +                    new IterationParams(IterationType.WARMUP, countWarmup, countWarmup, TimeValue.seconds(warmupTime),
    97 +                    new IterationParams(IterationType.MEASUREMENT, countMeasurement, countMeasurement, TimeValue.secon
86  98                      Mode.Throughput,
    99 +                    null,
    100 +                   1,
    101 +                   1,
    102 +                   1,
87  103                     ps,
88  104                     TimeUnit.SECONDS, 1,
89  105                     JVM_DUMMY,
106 122                 }
107 123                 benchmarkResults.add(new BenchmarkResult(params, iterResults));
108 124             }
109    -            results.add(new RunResult(params, benchmarkResults));
    125 +            results.add(new RunResult(params, benchmarkResults, null, null));
110 126         }
111 127         return results;
112 128     }
```

30 jmh-core/src/test/java/org/openjdk/jmh/runner/RunnerTest.java

```
61  61          Runner blade = new Runner(new OptionsBuilder());
62  62          BenchmarkParams bp = new BenchmarkParams("Foo", "bar", false,
63  63                  1, new int[]{1}, Collections.<String>emptyList(),
64     -                1, 1,
65     -                new IterationParams(IterationType.WARMUP,      1, TimeValue.seconds(1), 1),
66     -                new IterationParams(IterationType.MEASUREMENT, 1, TimeValue.seconds(1), 1),
67     -                Mode.Throughput, null, TimeUnit.SECONDS, 1,
68     -                Utils.getCurrentJvm(), Collections.<String>emptyList(),
    64 +                1, 1, 1,1,
    65 +                new IterationParams(IterationType.WARMUP,      1, 1, TimeValue.seconds(1), 1),
    66 +                new IterationParams(IterationType.MEASUREMENT, 1, 1, TimeValue.seconds(1), 1),
    67 +                Mode.Throughput, null, 1,1,1, null,
    68 +                TimeUnit.SECONDS, 1, Utils.getCurrentJvm(), Collections.<String>emptyList(),
69  69                  System.getProperty("java.version"), System.getProperty("java.vm.name"), System.getProperty("java.vm.ve
70  70                  TimeValue.days(1));
71  71          List<String> command = blade.getForkedMainCommand(bp, Collections.<ExternalProfiler>emptyList(), DUMMY_HOST, D
93  93          Runner blade = new Runner(new OptionsBuilder().build());
94  94          BenchmarkParams bp = new BenchmarkParams("Foo", "bar", false,
95  95                  1, new int[]{1}, Collections.<String>emptyList(),
96     -                1, 1,
```

```
97  -                 new IterationParams(IterationType.WARMUP,      1, TimeValue.seconds(1), 1),
98  -                 new IterationParams(IterationType.MEASUREMENT, 1, TimeValue.seconds(1), 1),
99  -                 Mode.Throughput, null, TimeUnit.SECONDS, 1,
100 -                 Utils.getCurrentJvm(), Collections.singletonList(CompilerHints.XX_COMPILE_COMMAND_FILE + tempHints),
    96 +              1, 1, 1, 1,
    97 +              new IterationParams(IterationType.WARMUP,      1, 1, TimeValue.seconds(1), 1),
    98 +              new IterationParams(IterationType.MEASUREMENT, 1, 1, TimeValue.seconds(1), 1),
    99 +              Mode.Throughput, null, 1,1,1, null, TimeUnit.SECONDS,
    100 +             1, Utils.getCurrentJvm(), Collections.singletonList(CompilerHints.XX_COMPILE_COMMAND_FILE + tempHints)
101 101               System.getProperty("java.version"), System.getProperty("java.vm.name"), System.getProperty("java.vm.ve
102 102               TimeValue.days(1));
103 103           List<String> command = blade.getForkedMainCommand(bp, Collections.<ExternalProfiler>emptyList(), DUMMY_HOST, D
129 129           Runner blade = new Runner(new OptionsBuilder().build());
130 130           BenchmarkParams bp = new BenchmarkParams("Foo", "bar", false,
131 131               1, new int[]{1}, Collections.<String>emptyList(),
132 -                 1, 1,
133 -                 new IterationParams(IterationType.WARMUP,      1, TimeValue.seconds(1), 1),
134 -                 new IterationParams(IterationType.MEASUREMENT, 1, TimeValue.seconds(1), 1),
135 -                 Mode.Throughput, null, TimeUnit.SECONDS, 1,
136 -                 Utils.getCurrentJvm(),
    132 +             1, 1, 1, 1,
    133 +             new IterationParams(IterationType.WARMUP,      1, 1, TimeValue.seconds(1), 1),
    134 +             new IterationParams(IterationType.MEASUREMENT, 1, 1, TimeValue.seconds(1), 1),
    135 +             Mode.Throughput, null, 1,1,1,null,
    136 +             TimeUnit.SECONDS, 1, Utils.getCurrentJvm(),
137 137               Arrays.asList(CompilerHints.XX_COMPILE_COMMAND_FILE + tempHints1, CompilerHints.XX_COMPILE_COMMAND_FIL
138 138               System.getProperty("java.version"), System.getProperty("java.vm.name"), System.getProperty("java.vm.ve
139 139               TimeValue.days(1));
```

8 jmh-core/src/test/java/org/openjdk/jmh/runner/TestBenchmarkList.java

```
27  27  import org.junit.BeforeClass;
28  28  import org.junit.Test;
29  29  import org.openjdk.jmh.annotations.Mode;
    30 + import org.openjdk.jmh.annotations.ReconfigureMode;
30  31  import org.openjdk.jmh.runner.format.OutputFormat;
31  32  import org.openjdk.jmh.runner.format.OutputFormatFactory;
32  33  import org.openjdk.jmh.runner.options.TimeValue;
57  58                  new int[]{1},
58  59                  Optional.<Collection<String>>none(),
59  60                  Optional.<Integer>none(),
    61 +                Optional.<Integer>none(),
60  62                  Optional.<TimeValue>none(),
61  63                  Optional.<Integer>none(),
62  64                  Optional.<Integer>none(),
63  65                  Optional.<TimeValue>none(),
64  66                  Optional.<Integer>none(),
65  67                  Optional.<Integer>none(),
66  68                  Optional.<Integer>none(),
    69 +                Optional.<Integer>none(),
    70 +                Optional.<Integer>none(),
    71 +                ReconfigureMode.CI,
    72 +                Optional.<Double>none(),
    73 +                Optional.<Double>none(),
    74 +                Optional.<Double>none(),
67  75                  Optional.<String>none(),
68  76                  Optional.<Collection<String>>none(),
69  77                  Optional.<Collection<String>>none(),
```

8 jmh-core/src/test/java/org/openjdk/jmh/runner/TestBenchmarkListEncoding.java

```
26  26
27  27  import org.junit.Test;
28  28  import org.openjdk.jmh.annotations.Mode;
    29 + import org.openjdk.jmh.annotations.ReconfigureMode;
29  30  import org.openjdk.jmh.runner.options.TimeValue;
30  31  import org.openjdk.jmh.util.Optional;
31  32
51  52                  new int[]{1},
52  53                  Optional.<Collection<String>>none(),
```

```
                    Optional.<Integer>none(),
+                   Optional.<Integer>none(),
                    Optional.<TimeValue>none(),
                    Optional.<Integer>none(),
                    Optional.<Integer>none(),
                    Optional.<TimeValue>none(),
                    Optional.<Integer>none(),
                    Optional.<Integer>none(),
                    Optional.<Integer>none(),
+                   Optional.<Integer>none(),
+                   Optional.<Integer>none(),
+                   ReconfigureMode.CI,
+                   Optional.<Double>none(),
+                   Optional.<Double>none(),
+                   Optional.<Double>none(),
                    Optional.<String>none(),
                    Optional.<Collection<String>>none(),
                    Optional.<Collection<String>>none(),
```

5 jmh-core/src/test/resources/org/openjdk/jmh/results/format/output-golden.json

```
            "mode" : "thrpt",
            "threads" : 80,
            "forks" : 828,
+           "warmupForks" : 55,
            "jvm" : "javadummy",
            "jvmArgs" : [
            ],
            "mode" : "thrpt",
            "threads" : 900,
            "forks" : 364,
+           "warmupForks" : 275,
            "jvm" : "javadummy",
            "jvmArgs" : [
            ],
            "mode" : "thrpt",
            "threads" : 466,
            "forks" : 677,
+           "warmupForks" : 750,
            "jvm" : "javadummy",
            "jvmArgs" : [
            ],
            "mode" : "thrpt",
            "threads" : 968,
            "forks" : 581,
+           "warmupForks" : 856,
            "jvm" : "javadummy",
            "jvmArgs" : [
            ],
            "mode" : "thrpt",
            "threads" : 739,
            "forks" : 670,
+           "warmupForks" : 338,
            "jvm" : "javadummy",
            "jvmArgs" : [
            ],
```

2 jmh-generator-annprocess/pom.xml

```
        <parent>
            <groupId>org.openjdk.jmh</groupId>
            <artifactId>jmh-parent</artifactId>
-           <version>1.21</version>
+           <version>1.21-Reconfigure</version>
        </parent>

        <name>JMH Generators: Annotation Processors</name>
```

2 jmh-generator-asm/pom.xml

```
30  30      <parent>
31  31          <groupId>org.openjdk.jmh</groupId>
32  32          <artifactId>jmh-parent</artifactId>
33      -       <version>1.21</version>
    33  +       <version>1.21-Reconfigure</version>
34  34      </parent>
35  35
36  36      <name>JMH Generators: ASM</name>
```

2 jmh-generator-bytecode/pom.xml

```
30  30      <parent>
31  31          <groupId>org.openjdk.jmh</groupId>
32  32          <artifactId>jmh-parent</artifactId>
33      -       <version>1.21</version>
    33  +       <version>1.21-Reconfigure</version>
34  34      </parent>
35  35
36  36      <name>JMH Generators: Bytecode</name>
```

2 jmh-generator-reflection/pom.xml

```
30  30      <parent>
31  31          <groupId>org.openjdk.jmh</groupId>
32  32          <artifactId>jmh-parent</artifactId>
33      -       <version>1.21</version>
    33  +       <version>1.21-Reconfigure</version>
34  34      </parent>
35  35
36  36      <name>JMH Generators: Reflection</name>
```

2 jmh-samples/pom.xml

```
36  36      <parent>
37  37          <groupId>org.openjdk.jmh</groupId>
38  38          <artifactId>jmh-parent</artifactId>
39      -       <version>1.21</version>
    39  +       <version>1.21-Reconfigure</version>
40  40      </parent>
41  41
42  42      <name>JMH Samples</name>
```

64 jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_00_Reconfigure.java

```
...  ...  @@ -0,0 +1,64 @@
1    + /*
2    +  * Copyright (c) 2015, Oracle America, Inc.
3    +  * All rights reserved.
4    +  *
5    +  * Redistribution and use in source and binary forms, with or without
6    +  * modification, are permitted provided that the following conditions are met:
7    +  *
8    +  *  * Redistributions of source code must retain the above copyright notice,
9    +  *    this list of conditions and the following disclaimer.
10   +  *
11   +  *  * Redistributions in binary form must reproduce the above copyright
12   +  *    notice, this list of conditions and the following disclaimer in the
13   +  *    documentation and/or other materials provided with the distribution.
14   +  *
15   +  *  * Neither the name of Oracle nor the names of its contributors may be used
16   +  *    to endorse or promote products derived from this software without
17   +  *    specific prior written permission.
18   +  *
19   +  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20   +  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21   +  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22   +  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
23   +  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
24   +  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
25   +  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
```

```
+  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
+  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
+  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
+  * THE POSSIBILITY OF SUCH DAMAGE.
+  */
+ package org.openjdk.jmh.samples;
+
+ import org.openjdk.jmh.annotations.*;
+ import org.openjdk.jmh.results.format.ResultFormatType;
+ import org.openjdk.jmh.runner.Runner;
+ import org.openjdk.jmh.runner.RunnerException;
+ import org.openjdk.jmh.runner.options.Options;
+ import org.openjdk.jmh.runner.options.OptionsBuilder;
+
+ import java.util.concurrent.TimeUnit;
+
+ public class JMHSample_00_Reconfigure {
+
+     @Benchmark
+     @Reconfigure(value = ReconfigureMode.DIVERGENCE, covThreshold = 0.07, ciThreshold = 0.08, kldThreshold = 0.09)
+     @Warmup(minIterations = 11, time = 1, timeUnit = TimeUnit.MICROSECONDS)
+     @Measurement(time = 1, timeUnit = TimeUnit.MICROSECONDS)
+     @BenchmarkMode(Mode.Reconfigure)
+     @Fork(minWarmups = 7, minValue = 9)
+     public void wellHelloThere() {
+         // this method was intentionally left blank.
+     }
+
+     public static void main(String[] args) throws RunnerException {
+         Options opt = new OptionsBuilder()
+                 .include(JMHSample_00_Reconfigure.class.getSimpleName())
+                 .result("D:\\out.json")
+                 .resultFormat(ResultFormatType.JSON)
+                 .build();
+
+         new Runner(opt).run();
+     }
+
+ }
```

BIN +2.58 MB jmh-samples/src/main/resources/pa_darwin_amd64

Binary file not shown.

BIN +2.57 MB jmh-samples/src/main/resources/pa_linux_amd64

Binary file not shown.

BIN +2.67 MB jmh-samples/src/main/resources/pa_windows_amd64.exe

Binary file not shown.

50 pom.xml

```
30  30      <groupId>org.openjdk.jmh</groupId>
31  31      <artifactId>jmh-parent</artifactId>
32  32      <packaging>pom</packaging>
33      -   <version>1.21</version>
    33  +   <version>1.21-Reconfigure</version>
34  34      <name>Java Microbenchmark Harness Parent</name>
35  35
36  36      <description>
117 117             </plugin>
118 118
119 119             <!-- Create javadoc jar -->
120     -           <plugin>
121     -               <groupId>org.apache.maven.plugins</groupId>
122     -               <artifactId>maven-javadoc-plugin</artifactId>
123     -               <configuration>
```

```
-                    <quiet>true</quiet>
-                </configuration>
-                <executions>
-                    <execution>
-                        <id>attach-javadoc</id>
-                        <phase>verify</phase>
-                        <goals>
-                            <goal>jar</goal>
-                        </goals>
-                    </execution>
-                </executions>
-            </plugin>
+ <!--            <plugin>-->
+ <!--                <groupId>org.apache.maven.plugins</groupId>-->
+ <!--                <artifactId>maven-javadoc-plugin</artifactId>-->
+ <!--                <configuration>-->
+ <!--                    <quiet>true</quiet>-->
+ <!--                </configuration>-->
+ <!--                <executions>-->
+ <!--                    <execution>-->
+ <!--                        <id>attach-javadoc</id>-->
+ <!--                        <phase>verify</phase>-->
+ <!--                        <goals>-->
+ <!--                            <goal>jar</goal>-->
+ <!--                        </goals>-->
+ <!--                    </execution>-->
+ <!--                </executions>-->
+ <!--            </plugin>-->

            <!-- Add sources and javadoc to eclipse project files when available. -->
-           <plugin>
-               <groupId>org.apache.maven.plugins</groupId>
-               <artifactId>maven-eclipse-plugin</artifactId>
-               <configuration>
-                   <downloadSources>true</downloadSources>
-                   <downloadJavadocs>true</downloadJavadocs>
-               </configuration>
-           </plugin>
+ <!--            <plugin>-->
+ <!--                <groupId>org.apache.maven.plugins</groupId>-->
+ <!--                <artifactId>maven-eclipse-plugin</artifactId>-->
+ <!--                <configuration>-->
+ <!--                    <downloadSources>true</downloadSources>-->
+ <!--                    <downloadJavadocs>true</downloadJavadocs>-->
+ <!--                </configuration>-->
+ <!--            </plugin>-->

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
```