


# 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 @@  
1 + <component name="ProjectRunConfigurationManager">  
2 +   <configuration default="false" name="JMHSample\_00\_Reconfigure" type="Application" factoryName="Application" nameIsGe  
3 +     <option name="MAIN\_CLASS\_NAME" value="org.openjdk.jmh.samples.JMHSample\_00\_Reconfigure" />  
4 +     <module name="jmh-samples" />  
5 +     <extension name="coverage">  
6 +       <pattern>  
7 +         <option name="PATTERN" value="org.openjdk.jmh.samples.\*" />  
8 +         <option name="ENABLED" value="true" />  
9 +       </pattern>  
10 +   </extension>  
11 +   <method v="2">  
12 +      <option name="Maven.BeforeRunTask" enabled="true" file="\$PROJECT\_DIR\$/jmh-core/pom.xml" goal="install -DskipTest  
13 +      <option name="Make" enabled="true" />  
14 +    </method>  
15 + </configuration>  
16 + </component> 

▼ 15


■■■■■

.idea/runConfigurations/Remote.xml



...


...

@@ -0,0 +1,15 @@  
1 + <component name="ProjectRunConfigurationManager">  
2 +   <configuration default="false" name="Remote" type="Remote">  
3 +     <option name="USE\_SOCKET\_TRANSPORT" value="true" />  
4 +     <option name="SERVER\_MODE" value="false" />  
5 +     <option name="SHMEM\_ADDRESS" />  
6 +     <option name="HOST" value="localhost" />  
7 +     <option name="PORT" value="5005" />  
8 +     <option name="AUTO\_RESTART" value="false" />  
9 +    <RunnerSettings RunnerId="Debug">  
10 +      <option name="DEBUG\_PORT" value="5005" />  
11 +      <option name="LOCAL" value="false" />  
12 +    </RunnerSettings>  
13 +    <method v="2" />  
14 + </configuration>  
15 + </component> 

▼ 3

■■■■■

README.md



...


...

@@ -0,0 +1,3 @@  
1 + # JMH Fork to support dynamic reconfiguration  
2 +  
3 + 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- <u>Reconfigure</u> </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- <u>Reconfigure</u> </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- <u>Reconfigure</u> </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- <u>Reconfigure</u> </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- <u>Reconfigure</u> </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- <u>Reconfigure</u> </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) {

▼ 2 jmh-core-ct/pom.xml

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

✓ 2 jmh-core-it/pom.xml

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

[illegible]

95	110	
96		-<plugin>
97		-<groupId>com.mycila.maven-license-plugin</groupId>
98		-<artifactId>maven-license-plugin</artifactId>
99		-<executions>
100		-<execution>
101		-<goals>
102		-<goal>format</goal>
103		-</goals>
104		-<phase>process-sources</phase>
105		-<configuration>
106		-<header>file:///\${project.basedir}/../src/license/gpl_cpe/header.txt</header>
107		-<skipExistingHeaders>>true</skipExistingHeaders>
108		-<strictCheck>>true</strictCheck>
109		-<mapping>
110		-<java>PHP</java>
111		-</mapping>
112		-</configuration>
113		-</execution>
114		-</executions>
115		-</plugin>
	111	+<!--<plugin>-->
	112	+<!--<groupId>com.mycila.maven-license-plugin</groupId>-->
	113	+<!--<artifactId>maven-license-plugin</artifactId>-->
	114	+<!--<executions>-->
	115	+<!--<execution>-->
	116	+<!--<goals>-->
	117	+<!--<goal>format</goal>-->
	118	+<!--</goals>-->
	119	+<!--<phase>process-sources</phase>-->
	120	+<!--<configuration>-->
	121	+<!--<header>file:///\${project.basedir}/../src/license/gpl_cpe/header.txt</header>-->
	122	+<!--<skipExistingHeaders>>true</skipExistingHeaders>-->
	123	+<!--<strictCheck>>true</strictCheck>-->
	124	+<!--<mapping>-->
	125	+<!--<java>PHP</java>-->
	126	+<!--</mapping>-->
	127	+<!--</configuration>-->
	128	+<!--</execution>-->
	129	+<!--</executions>-->
	130	+<!--</plugin>-->
116	131	
117	132	<plugin>
118	133	<groupId>org.apache.maven.plugins</groupId>
131	146	<resource>
132	147	<directory>src/main/resources/</directory>
133	148	<filtering>true</filtering>
	149	+<excludes>
	150	+<exclude>pa_linux_amd64</exclude>
	151	+<exclude>pa_darwin_amd64</exclude>
	152	+<exclude>pa_windows_amd64.exe</exclude>
	153	+</excludes>
	154	+</resource>
	155	+<resource>
	156	+<directory>src/main/resources/</directory>
	157	+<filtering>>false</filtering>
	158	+<includes>
	159	+<include>pa_linux_amd64</include>
	160	+<include>pa_darwin_amd64</include>
	161	+<include>pa_windows_amd64.exe</include>
	162	+</includes>
134	163	</resource>
135	164	</resources>
136	165	</build>

▼ 48■■■■■jmh-core/src/main/java/org/openjdk/jmh/annotations/Fork.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
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 <div><div></div><div></div><div></div><div></div><div></div></div> jmh-core/src/main/java/org/openjdk/jmh/annotations/Measurement.java <div></div>		
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 <div><div></div><div></div><div></div><div></div><div></div></div> jmh-core/src/main/java/org/openjdk/jmh/annotations/Mode.java <div></div>		
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 <div><div></div><div></div><div></div><div></div><div></div></div> jmh-core/src/main/java/org/openjdk/jmh/annotations/Reconfigure.java <div></div>		

... ... @@ -0,0 +1,40 @@

1

+ package org.openjdk.jmh.annotations;

2

+

3

+ import java.lang.annotation.\*;

4

+

5

+ /\*\*

6

+ \* <b>Reconfigure annotation allows to set the default reconfigure thresholds for the benchmark.</b>

7

+ \*

8

+ \* <p>This annotation may be put at {@link Benchmark} method to have effect on that

9

+ \* method only, or at the enclosing class instance to have the effect over all

10

+ \* {@link Benchmark} methods in the class. This annotation may be overridden with

11

+ \* the runtime options.</p>

12

+ \*/

13

+ @Inherited

14

+ @Target({ElementType.METHOD, ElementType.TYPE})

15

+ @Retention(RetentionPolicy.RUNTIME)

16

+ public @interface Reconfigure {

17

+

18

+ double BLANK\_THRESHOLD = -1;

19

+

20

+ /\*\*

21

+ \* @return reconfigure mode

22

+ \* @see ReconfigureMode

23

+ \*/

24

+ ReconfigureMode value() default ReconfigureMode.NONE;

25

+

26

+ /\*\*

27

+ \* @return coefficient of variation variability threshold

28

+ \*/

29

+ double covThreshold() default BLANK\_THRESHOLD;

30

+

31

+ /\*\*

32

+ \* @return confidence interval variability threshold

33

+ \*/

34

+ double ciThreshold() default BLANK\_THRESHOLD;

35

+

36

+ /\*\*

37

+ \* @return p value of kullback leibler divergence as variability threshold

38

+ \*/

39

+ double kldThreshold() default BLANK\_THRESHOLD;

40

+ }

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

... ... @@ -0,0 +1,90 @@

1

+ package org.openjdk.jmh.annotations;

2

+

3

+ import java.util.ArrayList;

4

+ import java.util.List;

5

+

6

+ /\*\*

7

+ \* Reconfigure mode.

8

+ \*/

9

+ public enum ReconfigureMode {

10

+

11

+ /\*\*

12

+ \* <p>Internal mode </p>

13

+ \*/

14

+ NONE("none", "none"),

15

+

16

+ /\*\*

17

+ \* <p>coefficient of variation as variability criteria</p>

18

+ \*/

19

+ COV("cov", "coefficient of variation"),

20

+

21

+ /\*\*

22

+ \* <p>confidence interval as variability criteria</p>

23

+ \*/

24

+ CI("ci", "confidence interval"),



```
25 +
26 + /**
27 +  * <p>kullback leibler divergence as variability criteria</p>
28 +  */
29 + DIVERGENCE("kld", "kullback leibler divergence"),
30 +
31 + ;
32 +
33 + private final String shortLabel;
34 + private final String longLabel;
35 +
36 + ReconfigureMode(String shortLabel, String longLabel) {
37 +     this.shortLabel = shortLabel;
38 +     this.longLabel = longLabel;
39 + }
40 +
41 + public String shortLabel() {
42 +     return shortLabel;
43 + }
44 +
45 + public String longLabel() {
46 +     return longLabel;
47 + }
48 +
49 + public static ReconfigureMode deepValueOf(String name) {
50 +     try {
51 +         return ReconfigureMode.valueOf(name);
52 +     } catch (IllegalArgumentException iae) {
53 +         ReconfigureMode inferred = null;
54 +         for (ReconfigureMode type : options()) {
55 +             if (type.shortLabel().startsWith(name)) {
56 +                 if (inferred == null) {
57 +                     inferred = type;
58 +                 } else {
59 +                     throw new IllegalStateException("Unable to parse reconfigure mode, ambiguous prefix given: \""
60 +                         "Known values are " + getKnown());
61 +                 }
62 +             }
63 +         }
64 +         if (inferred != null) {
65 +             return inferred;
66 +         } else {
67 +             throw new IllegalStateException("Unable to parse reconfigure mode: \"" + name + "\"\n" +
68 +                 "Known values are " + getKnown());
69 +         }
70 +     }
71 + }
72 +
73 + public static List<String> getKnown() {
74 +     List<String> res = new ArrayList<>();
75 +     for (ReconfigureMode type : ReconfigureMode.options()) {
76 +         res.add(type.name() + "/" + type.shortLabel());
77 +     }
78 +     return res;
79 + }
80 +
81 + public static ReconfigureMode[] options() {
82 +     return new ReconfigureMode[]{
83 +         COV, CI, DIVERGENCE
84 +     };
85 + }
86 +
87 + public boolean isNone() {
88 +     return equals(ReconfigureMode.NONE);
89 + }
90 + }
```

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

2424 \*/



25	25	<code>package org.openjdk.jmh.annotations;</code>
26	26	
27		<code>- import java.lang.annotation.ElementType;</code>
28		<code>- import java.lang.annotation.Inherited;</code>
29		<code>- import java.lang.annotation.Retention;</code>
30		<code>- import java.lang.annotation.RetentionPolicy;</code>
31		<code>- import java.lang.annotation.Target;</code>
	27	<code>+ import java.lang.annotation.*;</code>
32	28	<code>import java.util.concurrent.TimeUnit;</code>
33	29	
34	30	<code>/**</code>
40	36	<code> *</code>
41	37	<code> * @see Measurement</code>
42	38	<code> */</code>
43		<code>- @Target({ElementType.METHOD,ElementType.TYPE})</code>
	39	<code>+ @Target({ElementType.METHOD, _ElementType.TYPE})</code>
44	40	<code>@Retention(RetentionPolicy.RUNTIME)</code>
45	41	<code>@Inherited</code>
46	42	<code>public @interface Warmup {</code>
49	45	<code>    int BLANK_TIME = -1;</code>
50	46	<code>    int BLANK_BATCHSIZE = -1;</code>
51	47	
52		<code>- /** @return Number of warmup iterations */</code>
	48	<code>+ /**</code>
	49	<code> * @return Number of warmup iterations</code>
	50	<code> */</code>
53	51	<code>int iterations() default BLANK_ITERATIONS;</code>
54	52	
55		<code>- /** @return Time for each warmup iteration */</code>
	53	<code>+ /**</code>
	54	<code> * @return Minimum number of warmup iterations</code>
	55	<code> */</code>
	56	<code>int minIterations() default BLANK_ITERATIONS;</code>
	57	<code>+</code>
	58	<code>/**</code>
	59	<code> * @return Time for each warmup iteration</code>
	60	<code> */</code>
56	61	<code>int time() default BLANK_TIME;</code>
57	62	
58		<code>- /** @return Time unit for warmup iteration duration */</code>
	63	<code>+ /**</code>
	64	<code> * @return Time unit for warmup iteration duration</code>
	65	<code> */</code>
59	66	<code>TimeUnit timeUnit() default TimeUnit.SECONDS;</code>
60	67	
61		<code>- /** @return batch size: number of benchmark method calls per operation */</code>
	68	<code>+ /**</code>
	69	<code> * @return batch size: number of benchmark method calls per operation</code>
	70	<code> */</code>
62	71	<code>int batchSize() default BLANK_BATCHSIZE;</code>
63	72	
64	73	<code>}</code>

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

153	153	<code>group.getGroupThreads(),</code>
154	154	<code>group.getGroupLabels(),</code>
155	155	<code>group.getWarmupIterations(),</code>
	156	<code>+ group.getMinWarmupIterations(),</code>
156	157	<code>group.getWarmupTime(),</code>
157	158	<code>group.getWarmupBatchSize(),</code>
158	159	<code>group.getMeasurementIterations(),</code>
159	160	<code>group.getMeasurementTime(),</code>
160	161	<code>group.getMeasurementBatchSize(),</code>
161	162	<code>group.getForks(),</code>
	163	<code>+ group.getMinForks(),</code>
162	164	<code>group.getWarmupForks(),</code>
	165	<code>+ group.getMinWarmupForks(),</code>
	166	<code>+ group.getReconfigureMode(),</code>
	167	<code>+ group.getReconfigureCovThreshold(),</code>

	168	+	group.getReconfigureCiThreshold(),
	169	+	group.getReconfigureKldThreshold(),
163	170		group.getJvm(),
164	171		group.getJvmArgs(),
165	172		group.getJvmArgsPrepend(),
530	537		generateAverageTime(writer, benchmarkKind, methodGroup, states);
531	538		break;
532	539		case SampleTime:
	540	+	case Reconfigure:
533	541		generateSampleTime(writer, benchmarkKind, methodGroup, states);
534	542		break;
535	543		case SingleShotTime:

▼ 62 ■■■■■ jmh-core/src/main/java/org/openjdk/jmh/generators/core/MethodGroup.java			
25	25		package org.openjdk.jmh.generators.core;
26	26		
27	27		import org.openjdk.jmh.annotations.*;
	28	+	import org.openjdk.jmh.runner.Defaults;
28	29		import org.openjdk.jmh.runner.options.TimeValue;
29	30		import org.openjdk.jmh.util.Optional;
30	31		
172	173		return Optional.none();
173	174		}
174	175		
	176	+	public Optional<Integer> getMinWarmupIterations() {
	177	+	for (Warmup ann : getAll(Warmup.class)) {
	178	+	if (ann.minIterations() != Warmup.BLANK_ITERATIONS) {
	179	+	return Optional.of(ann.minIterations());
	180	+	}
	181	+	}
	182	+	return Optional.none();
	183	+	}
	184	+	
175	185		public Optional<TimeValue> getWarmupTime() {
176	186		for (Warmup ann : getAll(Warmup.class)) {
177	187		if (ann.time() != Warmup.BLANK_TIME) {
226	236		return Optional.none();
227	237		}
228	238		
	239	+	public Optional<Integer> getMinForks() {
	240	+	for (Fork ann : getAll(Fork.class)) {
	241	+	if (ann.minValue() != Fork.BLANK_FORKS) {
	242	+	return Optional.of(ann.minValue());
	243	+	}
	244	+	}
	245	+	return Optional.none();
	246	+	}
	247	+	
229	248		public Optional<Integer> getWarmupForks() {
230	249		for (Fork ann : getAll(Fork.class)) {
231	250		if (ann.warmups() != Fork.BLANK_FORKS) {
235	254		return Optional.none();
236	255		}
237	256		
	257	+	public Optional<Integer> getMinWarmupForks() {
	258	+	for (Fork ann : getAll(Fork.class)) {
	259	+	if (ann.minWarmups() != Fork.BLANK_FORKS) {
	260	+	return Optional.of(ann.minWarmups());
	261	+	}
	262	+	}
	263	+	return Optional.none();
	264	+	}
	265	+	
	266	+	public ReconfigureMode getReconfigureMode() {
	267	+	for (Reconfigure ann : getAll(Reconfigure.class)) {
	268	+	return ann.value();
	269	+	}
	270	+	return Defaults.RECONFIGURE_MODE;
	271	+	}

272	+	
273	+	public Optional<Double> getReconfigureCovThreshold() {
274	+	for (Reconfigure ann : getAll(Reconfigure.class)) {
275	+	if (ann.covThreshold() != Reconfigure.BLANK_THRESHOLD) {
276	+	return Optional.of(ann.covThreshold());
277	+	}
278	+	}
279	+	return Optional.none();
280	+	}
281	+	
282	+	public Optional<Double> getReconfigureCiThreshold() {
283	+	for (Reconfigure ann : getAll(Reconfigure.class)) {
284	+	if (ann.ciThreshold() != Reconfigure.BLANK_THRESHOLD) {
285	+	return Optional.of(ann.ciThreshold());
286	+	}
287	+	}
288	+	return Optional.none();
289	+	}
290	+	
291	+	public Optional<Double> getReconfigureKldThreshold() {
292	+	for (Reconfigure ann : getAll(Reconfigure.class)) {
293	+	if (ann.kldThreshold() != Reconfigure.BLANK_THRESHOLD) {
294	+	return Optional.of(ann.kldThreshold());
295	+	}
296	+	}
297	+	return Optional.none();
298	+	}
299	+	
238	300	public Optional<String> getJvm() {
239	301	for (Fork ann : getAll(Fork.class)) {
240	302	if (!ann.jvm().equals(Fork.BLANK_ARGS)) {

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

25	25	package org.openjdk.jmh.infra;
26	26	
27	27	import org.openjdk.jmh.annotations.Mode;
	28	+ import org.openjdk.jmh.annotations.ReconfigureMode;
28	29	import org.openjdk.jmh.runner.WorkloadParams;
29	30	import org.openjdk.jmh.runner.options.TimeValue;
30	31	import org.openjdk.jmh.util.Utils;
66	67	
67	68	public BenchmarkParams(String benchmark, String generatedTarget, boolean synchIterations,
68	69	int threads, int[] threadGroups, Collection<String> threadGroupLabels,
69	-	int forks, int warmupForks,
	70	+ int forks, int minForks, int warmupForks, int minWarmupForks,
70	71	IterationParams warmup, IterationParams measurement,
71	-	Mode mode, WorkloadParams params,
	72	+ Mode mode, ReconfigureMode reconfigureMode, double reconfigureCovThreshold,
	73	+ double reconfigureCiThreshold, double reconfigureKldThreshold, WorkloadParams params,
72	74	TimeUnit timeUnit, int opsPerInvocation,
73	75	String jvm, Collection<String> jvmArgs,
74	76	String jdkVersion, String vmName, String vmVersion, String jmhVersion,
75	77	TimeValue timeout) {
76	78	super(benchmark, generatedTarget, synchIterations,
77	79	threads, threadGroups, threadGroupLabels,
78	-	forks, warmupForks,
	80	+ forks, minForks, warmupForks, minWarmupForks,
79	81	warmup, measurement,
80	-	mode, params,
	82	+ mode, reconfigureMode, reconfigureCovThreshold,
	83	+ reconfigureCiThreshold, reconfigureKldThreshold, params,
81	84	timeUnit, opsPerInvocation,
82	85	jvm, jvmArgs,
83	86	jdkVersion, vmName, vmVersion, jmhVersion,
91	94	private int markerEnd;
92	95	public BenchmarkParamsL4(String benchmark, String generatedTarget, boolean synchIterations,
93	96	int threads, int[] threadGroups, Collection<String> threadGroupLabels,
94	-	int forks, int warmupForks,
	97	+ int forks, int minForks, int warmupForks, int minWarmupForks,

95	98		IterationParams <b>warmup</b> , IterationParams <b>measurement</b> ,
96		-	Mode <b>mode</b> , WorkloadParams <b>params</b> ,
	99	+	Mode <b>mode</b> , ReconfigureMode <b>reconfigureMode</b> , double <b>reconfigureCovThreshold</b> ,
	100	+	double <b>reconfigureCiThreshold</b> , double <b>reconfigureKldThreshold</b> , WorkloadParams <b>params</b> ,
97	101		TimeUnit <b>timeUnit</b> , int <b>opsPerInvocation</b> ,
98	102		String <b>jvm</b> , Collection<String> <b>jvmArgs</b> ,
99	103		String <b>jdkVersion</b> , String <b>vmName</b> , String <b>vmVersion</b> , String <b>jmhVersion</b> ,
100	104		TimeValue <b>timeout</b> ) {
101	105		super(benchmark, generatedTarget, synchIterations,
102	106		threads, threadGroups, threadGroupLabels,
103		-	forks, warmupForks,
	107	+	forks, <u>minForks</u> , warmupForks, <u>minWarmupForks</u> ,
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 <b>benchmark</b> , String <b>generatedTarget</b> , boolean <b>synchIterations</b> ,
134	139		int <b>threads</b> , int[] <b>threadGroups</b> , Collection<String> <b>threadGroupLabels</b> ,
135		-	int <b>forks</b> , int <b>warmupForks</b> ,
	140	+	int <b>forks</b> , int <u>minForks</u> , int <u>warmupForks</u> , int <u>minWarmupForks</u> ,
136	141		IterationParams <b>warmup</b> , IterationParams <b>measurement</b> ,
137		-	Mode <b>mode</b> , WorkloadParams <b>params</b> ,
	142	+	Mode <b>mode</b> , ReconfigureMode <b>reconfigureMode</b> , double <b>reconfigureCovThreshold</b> ,
	143	+	double <b>reconfigureCiThreshold</b> , double <b>reconfigureKldThreshold</b> , WorkloadParams <b>params</b> ,
138	144		TimeUnit <b>timeUnit</b> , int <b>opsPerInvocation</b> ,
139	145		String <b>jvm</b> , Collection<String> <b>jvmArgs</b> ,
140	146		String <b>jdkVersion</b> , String <b>vmName</b> , String <b>vmVersion</b> , String <b>jmhVersion</b> ,
141	147		TimeValue <b>timeout</b> ) {
142	148		super(benchmark, generatedTarget, synchIterations,
143	149		threads, threadGroups, threadGroupLabels,
144		-	forks, warmupForks,
	150	+	forks, <u>minForks</u> , warmupForks, <u>minWarmupForks</u> ,
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 <b>benchmark</b> , String <b>generatedTarget</b> , boolean <b>synchIterations</b> ,
203	216		int <b>threads</b> , int[] <b>threadGroups</b> , Collection<String> <b>threadGroupLabels</b> ,
204		-	int <b>forks</b> , int <b>warmupForks</b> ,
	217	+	int <b>forks</b> , int <u>minForks</u> , int <u>warmupForks</u> , int <u>minWarmupForks</u> ,
205	218		IterationParams <b>warmup</b> , IterationParams <b>measurement</b> ,
206		-	Mode <b>mode</b> , WorkloadParams <b>params</b> ,
	219	+	Mode <b>mode</b> , ReconfigureMode <b>reconfigureMode</b> , double <b>reconfigureCovThreshold</b> ,
	220	+	double <b>reconfigureCiThreshold</b> , double <b>reconfigureKldThreshold</b> , WorkloadParams <b>params</b> ,
207	221		TimeUnit <b>timeUnit</b> , int <b>opsPerInvocation</b> ,
208	222		String <b>jvm</b> , Collection<String> <b>jvmArgs</b> ,

```
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;
218      232      + this.minForks = minForks;
218      233      this.warmupForks = warmupForks;
218      234      + this.minWarmupForks = minWarmupForks;
219      235      this.warmup = warmup;
220      236      this.measurement = measurement;
221      237      this.mode = mode;
221      238      + this.reconfigureMode = reconfigureMode;
221      239      + this.reconfigureCovThreshold = reconfigureCovThreshold;
221      240      + this.reconfigureCiThreshold = reconfigureCiThreshold;
221      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
291      312      + /**
291      313      +  * @return minimum number of forked VM runs, which we measure
291      314      +  */
291      315      + public int getMinForks() {
291      316      +      return minForks;
291      317      +  }
291      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
298      326      + /**
298      327      +  * @return minimum number of forked VM runs, which we discard from the result
298      328      +  */
298      329      + public int getMinWarmupForks() {
298      330      +      return minWarmupForks;
298      331      +  }
298      332      +
299      333      /**
300      334      * @return benchmark mode
301      335      */
302      336      public Mode getMode() {
303      337      return mode;
304      338  }
305      339
305      340      + /**
305      341      +  * @return reconfigure mode
305      342      +  */
305      343      + public ReconfigureMode getReconfigureMode() {
305      344      +      return reconfigureMode;
305      345      +  }
305      346      +
305      347      + /**
305      348      +  * @return coefficient of variation variability threshold
305      349      +  */
305      350      + public double getReconfigureCovThreshold() {
305      351      +      return reconfigureCovThreshold;
305      352      +  }
305      353      +
305      354      + /**
305      355      +  * @return confidence interval variability threshold
305      356      +  */
305      357      + public double getReconfigureCiThreshold() {
305      358      +      return reconfigureCiThreshold;
305      359      +  }
305      360      +
305      361      + /**
```

	362	+     * @return p value of kullback leibler divergence as variability threshold
	363	+     */
	364	+     public double getReconfigureKldThreshold() {
	365	+         return reconfigureKldThreshold;
	366	+     }
	367	+
	368	+     public double getReconfigureThreshold() {
	369	+         switch (getReconfigureMode()) {
	370	+             case CI:
	371	+                 return getReconfigureCiThreshold();
	372	+             case COV:
	373	+                 return getReconfigureCovThreshold();
	374	+             case DIVERGENCE:
	375	+                 return getReconfigureKldThreshold();
	376	+             default:
	377	+                 throw new IllegalArgumentException("Reconfigure Mode is nod valid");
	378	+         }
	379	+     }
	380	+
306	381	/**
307	382	* @return benchmark name
308	383	*/

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

51	51	Utils.check(IterationParams.class, "type", "count", "timeValue", "batchSize");
52	52	}
53	53	
54		-     public IterationParams(IterationType type, int count, TimeValue time, int batchSize) {
55		-         super(type, count, time, batchSize);
	54	+     public IterationParams(IterationType type, int count, int minCount, TimeValue time, int batchSize) {
	55	+         super(type, count, minCount, time, batchSize);
56	56	}
57	57	}
58	58	
59	59	abstract class IterationParamsL4 extends IterationParamsL3 {
60	60	private static final long serialVersionUID = 9079354621906758255L;
61	61	
62	62	private int markerEnd;
63		-     public IterationParamsL4(IterationType type, int count, TimeValue time, int batchSize) {
64		-         super(type, count, time, batchSize);
	63	+     public IterationParamsL4(IterationType type, int count, int minCount, TimeValue time, int batchSize) {
	64	+         super(type, count, minCount, time, batchSize);
65	65	}
66	66	}
67	67	
85	85	private boolean q161, q162, q163, q164, q165, q166, q167, q168;
86	86	private boolean q171, q172, q173, q174, q175, q176, q177, q178;
87	87	
88		-     public IterationParamsL3(IterationType type, int count, TimeValue time, int batchSize) {
89		-         super(type, count, time, batchSize);
	88	+     public IterationParamsL3(IterationType type, int count, int minCount, TimeValue time, int batchSize) {
	89	+         super(type, count, minCount, time, batchSize);
90	90	}
91	91	}
92	92	
126	126	*/
127	127	protected final int count;
128	128	
	129	+     /**
	130	+         * minimum amount of iterations
	131	+         */
	132	+         protected final int minCount;
	133	+
129	134	/**
130	135	* iteration runtime
131	136	*/
136	141	*/
137	142	protected final int batchSize;
138	143	



139		-	public IterationParamsL2(IterationType type, int count, TimeValue time, int batchSize) {
	144	+	public IterationParamsL2(IterationType type, int count, <u>int minCount</u> , 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()+", "+ <u>getMinCount()</u> +", "+ 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;



```
22 +         this.measurementThresholds = measurementThresholds;
23 +         this.atLeastOneWarning = atLeastOneWarning;
24 +     }
25 +
26 +     public Multimap<BenchmarkParams, BenchmarkResult> getResults() {
27 +         return results;
28 +     }
29 +
30 +     public Pair<List<Double>, List<Double>> getThresholdsPair() {
31 +         return new Pair<List<Double>, List<Double>>(warmupThresholds, measurementThresholds);
32 +     }
33 +
34 +     public boolean hasAtLeastOneWarning() {
35 +         return atLeastOneWarning;
36 +     }
37 + }
```

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

```
...  ... @@ -0,0 +1,30 @@
1 + package org.openjdk.jmh.reconfigure.helper;
2 +
3 + import java.util.ArrayList;
4 + import java.util.List;
5 + import java.util.Map;
6 +
7 + public class HistogramHelper {
8 +     public static List<HistogramItem> toList(Map<Integer, List<HistogramItem>> map) {
9 +         List<HistogramItem> list = new ArrayList<>();
10 +         for (Integer key : map.keySet()) {
11 +             list.addAll(map.get(key));
12 +         }
13 +
14 +         return list;
15 +     }
16 +
17 +     public static List<Double> toArray(List<HistogramItem> input) {
18 +         List<Double> out = new ArrayList<>();
19 +
20 +         for (int i = 0; i < input.size(); i++) {
21 +             HistogramItem item = input.get(i);
22 +
23 +             for (int j = 0; j < item.getCount(); j++) {
24 +                 out.add(item.getValue());
25 +             }
26 +         }
27 +
28 +         return out;
29 +     }
30 + }
```

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

```
...  ... @@ -0,0 +1,45 @@
1 + package org.openjdk.jmh.reconfigure.helper;
2 +
3 + public class HistogramItem {
4 +     private final int fork;
5 +     private final int iteration;
6 +     private final double value;
7 +     private final long count;
8 +
9 +     public HistogramItem(int fork, int iteration, double value, long count) {
10 +         this.fork = fork;
11 +         this.iteration = iteration;
12 +         this.value = value;
13 +         this.count = count;
14 +     }
15 + }
```

```
16 +     public int getFork() {
17 +         return fork;
18 +     }
19 +
20 +     public int getIteration() {
21 +         return iteration;
22 +     }
23 +
24 +     public double getValue() {
25 +         return value;
26 +     }
27 +
28 +     public long getCount() {
29 +         return count;
30 +     }
31 +
32 +     public HistogramItem single() {
33 +         return new HistogramItem(fork, iteration, value, 1);
34 +     }
35 +
36 +     @Override
37 +     public String toString() {
38 +         return "HistogramItem{" +
39 +             "fork=" + fork +
40 +             ", iteration=" + iteration +
41 +             ", value=" + value +
42 +             ", count=" + count +
43 +             '}';
44 +     }
45 + }
```

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

```
...  ... @@ -0,0 +1,17 @@
1 + package org.openjdk.jmh.reconfigure.helper;
2 +
3 + import java.util.List;
4 +
5 + public class ListToArray {
6 +     private static double[] toPrimitive(Double[] array) {
7 +         final double[] result = new double[array.length];
8 +         for (int i = 0; i < array.length; i++) {
9 +             result[i] = array[i];
10 +         }
11 +         return result;
12 +     }
13 +
14 +     public static double[] toPrimitive(List<Double> input) {
15 +         return toPrimitive(input.toArray(new Double[0]));
16 +     }
17 + }
```

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

```
...  ... @@ -0,0 +1,58 @@
1 + package org.openjdk.jmh.reconfigure.helper;
2 +
3 + import org.apache.commons.math3.stat.descriptive.DescriptiveStatistics;
4 +
5 + import java.util.ArrayList;
6 + import java.util.List;
7 +
8 + public class OutlierDetector {
9 +     private double outlierFactor;
10 +     private List<HistogramItem> input;
11 +     private double[] inputRaw;
12 +
13 +     private double min;
14 +     private double max;
```

```
15 +
16 +     private List<HistogramItem> outlier = new ArrayList<>();
17 +     private List<HistogramItem> inlier = new ArrayList<>();
18 +
19 +     public OutlierDetector(double outlierFactor, List<HistogramItem> input) {
20 +         this.outlierFactor = outlierFactor;
21 +         this.input = input;
22 +         this.inputRaw = ListToArray.toPrimitive(HistogramHelper.toArray(input));
23 +     }
24 +
25 +     public void run() {
26 +         DescriptiveStatistics ds = new DescriptiveStatistics(inputRaw);
27 +         double median = ds.getPercentile(50.0);
28 +
29 +         max = outlierFactor * median;
30 +         min = median / outlierFactor;
31 +
32 +         for (int i = 0; i < input.size(); i++) {
33 +             double value = input.get(i).getValue();
34 +
35 +             if (value <= max && value >= min) {
36 +                 inlier.add(input.get(i));
37 +             } else {
38 +                 outlier.add(input.get(i));
39 +             }
40 +         }
41 +     }
42 +
43 +     public double getMin() {
44 +         return min;
45 +     }
46 +
47 +     public double getMax() {
48 +         return max;
49 +     }
50 +
51 +     public List<HistogramItem> getOutlier() {
52 +         return outlier;
53 +     }
54 +
55 +     public List<HistogramItem> getInlier() {
56 +         return inlier;
57 +     }
58 + }
```

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

```
...    ...    @@ -0,0 +1,97 @@
1 + package org.openjdk.jmh.reconfigure.manager;
2 +
3 + import org.openjdk.jmh.infra.BenchmarkParams;
4 + import org.openjdk.jmh.reconfigure.helper.HistogramItem;
5 + import org.openjdk.jmh.reconfigure.statistics.evaluation.StatisticalEvaluation;
6 + import org.openjdk.jmh.results.IterationResult;
7 + import org.openjdk.jmh.runner.format.OutputFormat;
8 +
9 + import java.util.ArrayList;
10 + import java.util.List;
11 +
12 + public class ForkReconfigureManager extends ReconfigureManager {
13 +     private StatisticalEvaluation measurementEvaluation;
14 +
15 +     private List<Double> measurementThresholds = new ArrayList<>();
16 +
17 +     public ForkReconfigureManager(BenchmarkParams benchParams, OutputFormat out) {
18 +         super(benchParams, out);
19 +         warmupEvaluation = StatisticalForkEvaluationFactory.get(benchParams);
20 +         measurementEvaluation = StatisticalForkEvaluationFactory.get(benchParams);
21 +     }
22 + }
```

```
23 +     public void addFork(boolean isWarmup, int fork, List<IterationResult> list) {
24 +         List<HistogramItem> combined = new ArrayList<>();
25 +         for (int iteration = 1; iteration <= list.size(); iteration++) {
26 +             combined.addAll(toHistogramItems(fork, iteration, list.get(iteration - 1)));
27 +         }
28 +
29 +         if (isWarmup) {
30 +             addWarmupFork(combined);
31 +         } else {
32 +             addMeasurementFork(combined);
33 +         }
34 +     }
35 +
36 +     private void addWarmupFork(List<HistogramItem> list) {
37 +         warmupEvaluation.addIteration(list);
38 +     }
39 +
40 +     private void addMeasurementFork(List<HistogramItem> list) {
41 +         measurementEvaluation.addIteration(list);
42 +     }
43 +
44 +     public boolean checkForkThreshold(boolean isWarmup) {
45 +         if (isWarmup) {
46 +             return checkWarmupForkThreshold();
47 +         } else {
48 +             return checkMeasurementForkThreshold();
49 +         }
50 +     }
51 +
52 +     private boolean checkWarmupForkThreshold() {
53 +         int currentWarmupFork = warmupEvaluation.getIterationNumber();
54 +         if (currentWarmupFork < benchParams.getMinWarmupForks()) {
55 +             warmupThresholds.add(null);
56 +             return false;
57 +         } else {
58 +             int maxForks = benchParams.getWarmupForks();
59 +             double value = warmupEvaluation.calculateVariability();
60 +             warmupThresholds.add(value);
61 +             boolean result = warmupEvaluation.stableEnvironment(value);
62 +
63 +             if (currentWarmupFork == maxForks && !result) {
64 +                 printWarning("warmup forks", warmupEvaluation.getThreshold(), value);
65 +             } else if (currentWarmupFork < maxForks && result) {
66 +                 printInfo(currentWarmupFork, maxForks, "warmup forks", value, warmupEvaluation.getThreshold());
67 +             }
68 +
69 +             return result;
70 +         }
71 +     }
72 +
73 +     private boolean checkMeasurementForkThreshold() {
74 +         int currentMeasurementFork = measurementEvaluation.getIterationNumber();
75 +         if (currentMeasurementFork < benchParams.getMinForks()) {
76 +             measurementThresholds.add(null);
77 +             return false;
78 +         } else {
79 +             int maxForks = benchParams.getForks();
80 +             Double value = measurementEvaluation.calculateVariability();
81 +             measurementThresholds.add(value);
82 +             boolean result = measurementEvaluation.stableEnvironment(value);
83 +
84 +             if (currentMeasurementFork == maxForks && !result) {
85 +                 printWarning("measurement forks", measurementEvaluation.getThreshold(), value);
86 +             } else if (currentMeasurementFork < maxForks && result) {
87 +                 printInfo(currentMeasurementFork, maxForks, "measurement forks", value, measurementEvaluation.getThres
88 +             }
89 +
90 +             return result;
91 +         }
92 +     }
```

```
93 +
94 +     public List<Double> getMeasurementThresholds() {
95 +         return measurementThresholds;
96 +     }
97 + }
```

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

```
...  ... @@ -0,0 +1,37 @@
1 + package org.openjdk.jmh.reconfigure.manager;
2 +
3 + import org.openjdk.jmh.infra.BenchmarkParams;
4 + import org.openjdk.jmh.results.IterationResult;
5 + import org.openjdk.jmh.runner.format.OutputFormat;
6 +
7 + public class IterationReconfigureManager extends ReconfigureManager {
8 +     public IterationReconfigureManager(BenchmarkParams benchParams, OutputFormat out) {
9 +         super(benchParams, out);
10 +         warmupEvaluation = StatisticalIterationEvaluationFactory.get(benchParams);
11 +     }
12 +
13 +     public void addWarmupIteration(int iteration, IterationResult ir) {
14 +         warmupEvaluation.addIteration(toHistogramItems(0, iteration, ir));
15 +     }
16 +
17 +     public boolean checkWarmupIterationThreshold() {
18 +         int currentWarmupIteration = warmupEvaluation.getIterationNumber();
19 +         if (currentWarmupIteration < benchParams.getWarmup().getMinCount()) {
20 +             warmupThresholds.add(null);
21 +             return false;
22 +         } else {
23 +             int maxIterations = benchParams.getWarmup().getCount();
24 +             double value = warmupEvaluation.calculateVariability();
25 +             warmupThresholds.add(value);
26 +             boolean result = warmupEvaluation.stableEnvironment(value);
27 +
28 +             if (currentWarmupIteration == maxIterations && !result) {
29 +                 printWarning("warmup iterations", warmupEvaluation.getThreshold(), value);
30 +             } else if (currentWarmupIteration < maxIterations && result) {
31 +                 printInfo(currentWarmupIteration, maxIterations, "warmup iterations", value, warmupEvaluation.getThres
32 +             }
33 +
34 +             return result;
35 +         }
36 +     }
37 + }
```

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

```
...  ... @@ -0,0 +1,62 @@
1 + package org.openjdk.jmh.reconfigure.manager;
2 +
3 + import org.openjdk.jmh.infra.BenchmarkParams;
4 + import org.openjdk.jmh.reconfigure.helper.HistogramItem;
5 + import org.openjdk.jmh.reconfigure.statistics.evaluation.StatisticalEvaluation;
6 + import org.openjdk.jmh.results.IterationResult;
7 + import org.openjdk.jmh.runner.format.OutputFormat;
8 +
9 + import java.util.ArrayList;
10 + import java.util.Iterator;
11 + import java.util.List;
12 + import java.util.Map;
13 +
14 + public abstract class ReconfigureManager {
15 +     protected final BenchmarkParams benchParams;
16 +     private final OutputFormat out;
17 +
18 +     protected StatisticalEvaluation warmupEvaluation;
19 + }
```

```
20 +     protected List<Double> warmupThresholds = new ArrayList<>();
21 +
22 +     private boolean atLeastOneWarning = false;
23 +
24 +     public ReconfigureManager(BenchmarkParams benchParams, OutputFormat out) {
25 +         this.benchParams = benchParams;
26 +         this.out = out;
27 +     }
28 +
29 +     protected List<HistogramItem> toHistogramItems(int fork, int iteration, IterationResult ir) {
30 +         List<HistogramItem> list = new ArrayList<>();
31 +         Iterator<Map.Entry<Double, Long>> iterator = ir.getPrimaryResult().getStatistics().getRawData();
32 +         while (iterator.hasNext()) {
33 +             Map.Entry<Double, Long> entry = iterator.next();
34 +             list.add(new HistogramItem(fork, iteration, entry.getKey(), entry.getValue()));
35 +         }
36 +         return list;
37 +     }
38 +
39 +     protected void printWarning(String type, double threshold, Double value) {
40 +         out.println("");
41 +         out.println("#####");
42 +         out.println(String.format("# WARNING: Maximum number of %s was reached but statistical variability threshold o
43 +         out.println("#####");
44 +         out.println("");
45 +         atLeastOneWarning = true;
46 +     }
47 +
48 +     protected void printInfo(int currentNumber, int maxNumber, String type, Double value, double threshold) {
49 +         out.println("");
50 +         String lessOrGreaterThan = (value < threshold) ? "is less" : "is greater";
51 +         out.println(String.format("# Data collection is stopped after %d of %d %s because value of %.4f " + lessOrGrea
52 +         out.println("");
53 +     }
54 +
55 +     public List<Double> getWarmupThresholds() {
56 +         return warmupThresholds;
57 +     }
58 +
59 +     public boolean hasAtLeastOneWarning() {
60 +         return atLeastOneWarning;
61 +     }
62 + }
```

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

```
...    ...    @@ -0,0 +1,22 @@
1 + package org.openjdk.jmh.reconfigure.manager;
2 +
3 + import org.openjdk.jmh.infra.BenchmarkParams;
4 + import org.openjdk.jmh.reconfigure.statistics.evaluation.CiPercentageEvaluation;
5 + import org.openjdk.jmh.reconfigure.statistics.evaluation.CovEvaluation;
6 + import org.openjdk.jmh.reconfigure.statistics.evaluation.DivergenceEvaluation;
7 + import org.openjdk.jmh.reconfigure.statistics.evaluation.StatisticalEvaluation;
8 +
9 + public class StatisticalForkEvaluationFactory {
10 +     public static StatisticalEvaluation get(BenchmarkParams benchParams) {
11 +         switch (benchParams.getReconfigureMode()) {
12 +             case CI:
13 +                 return CiPercentageEvaluation.getForkInstance(benchParams.getReconfigureCiThreshold());
14 +             case COV:
15 +                 return CovEvaluation.getForkInstance(benchParams.getReconfigureCovThreshold());
16 +             case DIVERGENCE:
17 +                 return DivergenceEvaluation.getForkInstance(benchParams.getReconfigureKldThreshold());
18 +             default:
19 +                 throw new IllegalArgumentException("Reconfigure Mode is nod valid");
20 +         }
21 +     }
22 + }
```

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

...

...

@@ -0,0 +1,22 @@  
1 + package org.openjdk.jmh.reconfigure.manager;  
2 +  
3 + import org.openjdk.jmh.infra.BenchmarkParams;  
4 + import org.openjdk.jmh.reconfigure.statistics.evaluation.CiPercentageEvaluation;  
5 + import org.openjdk.jmh.reconfigure.statistics.evaluation.CovEvaluation;  
6 + import org.openjdk.jmh.reconfigure.statistics.evaluation.DivergenceEvaluation;  
7 + import org.openjdk.jmh.reconfigure.statistics.evaluation.StatisticalEvaluation;  
8 +  
9 + public class StatisticalIterationEvaluationFactory {  
10 + public static StatisticalEvaluation get(BenchmarkParams benchParams) {  
11 + switch (benchParams.getReconfigureMode()) {  
12 + case CI:  
13 + return CiPercentageEvaluation.getIterationInstance(benchParams.getReconfigureCiThreshold());  
14 + case COV:  
15 + return CovEvaluation.getIterationInstance(benchParams.getReconfigureCovThreshold());  
16 + case DIVERGENCE:  
17 + return DivergenceEvaluation.getIterationInstance(benchParams.getReconfigureKldThreshold());  
18 + default:  
19 + throw new IllegalArgumentException("Reconfigure Mode is nod valid");  
20 + }  
21 + }  
22 + }

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

...

...

@@ -0,0 +1,23 @@  
1 + package org.openjdk.jmh.reconfigure.statistics;  
2 +  
3 + import org.apache.commons.math3.stat.descriptive.DescriptiveStatistics;  
4 + import org.openjdk.jmh.reconfigure.helper.HistogramHelper;  
5 + import org.openjdk.jmh.reconfigure.helper.HistogramItem;  
6 + import org.openjdk.jmh.reconfigure.helper.ListToArray;  
7 +  
8 + import java.util.List;  
9 +  
10 + public class COV implements Statistic {  
11 + private List<Double> list;  
12 +  
13 + public COV(List<HistogramItem> list) {  
14 + this.list = HistogramHelper.toArray(list);  
15 + }  
16 +  
17 + @Override  
18 + public double getValue() {  
19 + double[] array = ListToArray.toPrimitive(list);  
20 + DescriptiveStatistics ds = new DescriptiveStatistics(array);  
21 + return ds.getStandardDeviation() / ds.getMean();  
22 + }  
23 + }

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

...

...

@@ -0,0 +1,13 @@  
1 + package org.openjdk.jmh.reconfigure.statistics;  
2 +  
3 + public class ReconfigureConstants {  
4 + public static final double OUTLIER\_FACTOR = 10.0;  
5 + public static final double RANGE\_OUTLIER\_FACTOR = 1.5;  
6 + public static final int SAMPLE\_SIZE = 1000;  
7 + public static final int DIVERGENCE\_NUMBER\_OF\_POINTS = 1000;  
8 + public static final int CI\_BOOTSTRAP\_SIMULATIONS = 1000;  
9 + public static final double CI\_SIGNIFICANCE\_LEVEL = 0.01;  
10 + public static final String CI\_STATISTIC = "mean";  
11 + public static final int CI\_INVOCATION\_SAMPLES = -1;  
12 + public static final int CI\_SAMPLE\_SIZE = 10000;  
13 + }

22 of 59

02.06.20, 16:25



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

...

...

```
@@ -0,0 +1,25 @@
1 + package org.openjdk.jmh.reconfigure.statistics;
2 +
3 + import org.apache.commons.math3.distribution.EnumeratedDistribution;
4 + import org.apache.commons.math3.util.Pair;
5 + import org.openjdk.jmh.reconfigure.helper.HistogramItem;
6 +
7 + import java.util.Arrays;
8 + import java.util.List;
9 + import java.util.stream.Collectors;
10 +
11 + public class Sampler {
12 +     private List<HistogramItem> items;
13 +
14 +     public Sampler(List<HistogramItem> items) {
15 +         this.items = items;
16 +     }
17 +
18 +     public List<HistogramItem> getSample(int size) {
19 +         HistogramItem[] output = new HistogramItem[size];
20 +         List<Pair<HistogramItem, Double>> distributionPairs = items.parallelStream().map(it -> new Pair<>(it.single(),
21 +         EnumeratedDistribution ed = new EnumeratedDistribution<>(distributionPairs);
22 +         ed.sample(size, output);
23 +         return Arrays.asList(output);
24 +     }
25 + }
```

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

...

...

```
@@ -0,0 +1,5 @@
1 + package org.openjdk.jmh.reconfigure.statistics;
2 +
3 + public interface Statistic {
4 +     double getValue();
5 + }
```

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

...

...

```
@@ -0,0 +1,101 @@
1 + package org.openjdk.jmh.reconfigure.statistics.ci;
2 +
3 + import org.apache.commons.io.IOUtils;
4 + import org.openjdk.jmh.reconfigure.helper.HistogramItem;
5 +
6 + import java.io.File;
7 + import java.io.FileWriter;
8 + import java.io.IOException;
9 + import java.nio.charset.Charset;
10 + import java.util.List;
11 +
12 + public class CI {
13 +     protected List<HistogramItem> histogramList;
14 +
15 +     protected String paToolPath;
16 +     protected final int bootstrapSimulations;
17 +     protected final double significanceLevel;
18 +     protected final String statistic;
19 +     protected final int ciInvocationSamples;
20 +
21 +     protected double lower;
22 +     protected double upper;
23 +     protected double statisticMetric;
24 +
25 +     public CI(List<HistogramItem> histogramList, int bootstrapSimulations, double significanceLevel, String statistic,
26 +     this.histogramList = histogramList;
27 +     this.bootstrapSimulations = bootstrapSimulations;
28 +     this.significanceLevel = significanceLevel;
```

```
29 +         this.statistic = statistic;
30 +         this.ciInvocationSamples = ciInvocationSamples;
31 +         executable();
32 +     }
33 +
34 +     private void executable() {
35 +         paToolPath = CIHelper.getInstance().getPath();
36 +     }
37 +
38 +     public void run() {
39 +         String file = getTmpFile(histogramList);
40 +         try {
41 +             Process process = Runtime.getRuntime().exec(paToolPath + " -os -bs " + bootstrapSimulations + " -is " + ci
42 +             String inputString = IOUtils.toString(process.getInputStream(), Charset.defaultCharset());
43 +             String errorString = IOUtils.toString(process.getErrorStream(), Charset.defaultCharset());
44 +             String output = (inputString + "\n" + errorString).trim();
45 +             String line = getFirstLine(output);
46 +             String[] parts = line.split(";");
47 +             statisticMetric = Double.parseDouble(parts[3]);
48 +             lower = Double.parseDouble(parts[4]);
49 +             upper = Double.parseDouble(parts[5]);
50 +         } catch (IOException e) {
51 +             e.printStackTrace();
52 +         } finally {
53 +             new File(file).delete();
54 +         }
55 +     }
56 +
57 +     protected String getFirstLine(String input) {
58 +         String[] lines = input.split("\n");
59 +
60 +         for (int i = 0; i < lines.length; i++) {
61 +             String line = lines[i].trim();
62 +
63 +             if (!line.startsWith("#") && !line.isEmpty()) {
64 +                 return line;
65 +             }
66 +         }
67 +
68 +         return null;
69 +     }
70 +
71 +     protected String getTmpFile(List<HistogramItem> list) {
72 +         try {
73 +             File tmpFile = File.createTempFile("reconfigure", ".csv");
74 +             FileWriter fw = new FileWriter(tmpFile);
75 +
76 +             for (int i = 0; i < list.size(); i++) {
77 +                 HistogramItem hi = list.get(i);
78 +                 fw.append(";;;0;" + hi.getFork() + ";" + hi.getIteration() + ";;;" + hi.getCount() + ";" + hi.getVal
79 +             }
80 +
81 +             fw.flush();
82 +             return tmpFile.getAbsolutePath();
83 +         } catch (IOException e) {
84 +             e.printStackTrace();
85 +         }
86 +
87 +         return null;
88 +     }
89 +
90 +     public double getLower() {
91 +         return lower;
92 +     }
93 +
94 +     public double getUpper() {
95 +         return upper;
96 +     }
97 +
98 +     public double getStatisticMetric() {
```

99 +       return statisticMetric;

100 +     }

101 + }

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

...   ...   @@ -0,0 +1,88 @@

1   + package org.openjdk.jmh.reconfigure.statistics.ci;

2   +

3   + import org.apache.commons.io.FileUtils;

4   + import org.apache.commons.lang3.SystemUtils;

5   +

6   + import java.io.File;

7   + import java.io.IOException;

8   + import java.io.InputStream;

9   + import java.net.URL;

10   + import java.nio.file.Paths;

11   +

12   + public class CIHelper {

13   +     private static CIHelper instance;

14   +     private File tempFile;

15   +

16   +     private CIHelper() {

17   +         copyExecutableToTmpFolder();

18   +     }

19   +

20   +     public static CIHelper getInstance() {

21   +         if (CIHelper.instance == null) {

22   +             CIHelper.instance = new CIHelper();

23   +         }

24   +         return CIHelper.instance;

25   +     }

26   +

27   +     public String getPath() {

28   +         return tempFile.getAbsolutePath();

29   +     }

30   +

31   +     private void copyExecutableToTmpFolder() {

32   +         String executableName = executableName();

33   +         tempFile = Paths.get(System.getProperty("java.io.tmpdir"), executableName).toFile();

34   +

35   +     ClassLoader classLoader = getClass().getClassLoader();

36   +

37   +     URL resource = classLoader.getResource(executableName());

38   +     if (resource == null) {

39   +         throw new IllegalArgumentException("file is not found!");

40   +     } else {

41   +         if (resource.getProtocol().equals("jar")) {

42   +             copyExecutableToTmpFolderJar(resource);

43   +         } else {

44   +             copyExecutableToTmpFolderFile(resource);

45   +         }

46   +     }

47   +

48   +     tempFile.setExecutable(true, true);

49   +     }

50   +

51   +     private void copyExecutableToTmpFolderFile(URL resource) {

52   +         try {

53   +             File executable = new File(resource.getFile());

54   +             FileUtils.copyFile(executable, tempFile);

55   +         } catch (IOException e) {

56   +             e.printStackTrace();

57   +         }

58   +     }

59   +

60   +     private void copyExecutableToTmpFolderJar(URL resource) {

61   +         try {

62   +             InputStream in = resource.openStream();

63   +             FileUtils.copyInputStreamToFile(in, tempFile);

```
64 +         } catch (IOException e) {
65 +             e.printStackTrace();
66 +         }
67 +     }
68 +
69 +     private boolean isWindows() {
70 +         return SystemUtils.IS_OS_WINDOWS;
71 +     }
72 +
73 +     private boolean isMacOS() {
74 +         return SystemUtils.IS_OS_MAC_OSX;
75 +     }
76 +
77 +     private String executableName() {
78 +         String e;
79 +         if (isWindows()) {
80 +             e = "pa_windows_amd64.exe";
81 +         } else if (isMacOS()) {
82 +             e = "pa_darwin_amd64";
83 +         } else {
84 +             e = "pa_linux_amd64";
85 +         }
86 +         return e;
87 +     }
88 + }
```

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

```
...  ... @@ -0,0 +1,32 @@
1 + package org.openjdk.jmh.reconfigure.statistics.ci;
2 +
3 + import org.openjdk.jmh.reconfigure.helper.HistogramItem;
4 + import org.openjdk.jmh.reconfigure.statistics.Statistic;
5 +
6 + import java.util.List;
7 +
8 + import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.*;
9 +
10 + public class CiPercentage implements Statistic {
11 +     private List<HistogramItem> histogramList;
12 +     private int ciBootstrapSimulations = CI_BOOTSTRAP_SIMULATIONS;
13 +     private double ciSignificanceLevel = CI_SIGNIFICANCE_LEVEL;
14 +     private String ciStatistics = CI_STATISTIC;
15 +     private int ciInvocationSamples = CI_INVOCATION_SAMPLES;
16 +
17 +     public CiPercentage(List<HistogramItem> histogramList) {
18 +         this.histogramList = histogramList;
19 +     }
20 +
21 +     public CiPercentage(List<HistogramItem> histogramList, int ciBootstrapSimulations) {
22 +         this.histogramList = histogramList;
23 +         this.ciBootstrapSimulations = ciBootstrapSimulations;
24 +     }
25 +
26 +     @Override
27 +     public double getValue() {
28 +         CI ci = new CI(histogramList, ciBootstrapSimulations, ciSignificanceLevel, ciStatistics, ciInvocationSamples);
29 +         ci.run();
30 +         return (ci.getUpper() - ci.getLower()) / ci.getStatisticMetric();
31 +     }
32 + }
```

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

```
...  ... @@ -0,0 +1,72 @@
1 + package org.openjdk.jmh.reconfigure.statistics.divergence;
2 +
3 + import org.apache.commons.math3.util.Pair;
4 + import org.apache.commons.math3.stat.descriptive.DescriptiveStatistics;
```

```
5 + import org.openjdk.jmh.reconfigure.helper.ListToArray;
6 + import org.openjdk.jmh.reconfigure.statistics.Statistic;
7 +
8 + import java.util.ArrayList;
9 + import java.util.List;
10 +
11 + import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.DIVERGENCE_NUMBER_OF_POINTS;
12 + import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.RANGE_OUTLIER_FACTOR;
13 +
14 + public class Divergence implements Statistic {
15 +     private List<Double> before;
16 +     private List<Double> after;
17 +
18 +     public Divergence(List<Double> before, List<Double> after) {
19 +         this.before = before;
20 +         this.after = after;
21 +     }
22 +
23 +     @Override
24 +     public double getValue() {
25 +         Pair<Double, Double> range = getRange();
26 +         double min = range.getFirst();
27 +         double max = range.getSecond();
28 +
29 +         if (min == max) {
30 +             return 1.0;
31 +         }
32 +
33 +         List<Double> y = new ArrayList<>();
34 +         double step = (max - min) / (DIVERGENCE_NUMBER_OF_POINTS - 1);
35 +         for (int i = 0; i < DIVERGENCE_NUMBER_OF_POINTS; i++) {
36 +             y.add(min + i * step);
37 +         }
38 +
39 +         double[] pdfBefore = ProbabilityDensityFunction.estimate(before, y);
40 +         double[] pdfAfter = ProbabilityDensityFunction.estimate(after, y);
41 +
42 +         double kldBefore = KullbackLeiblerDivergence.continuous(pdfBefore, pdfAfter, step);
43 +         double kldAfter = KullbackLeiblerDivergence.continuous(pdfAfter, pdfBefore, step);
44 +
45 +         return Math.pow(2.0, -kldBefore) * Math.pow(2.0, -kldAfter);
46 +     }
47 +
48 +     private Pair<Double, Double> getRange() {
49 +         Pair<Double, Double> rangeBefore = getRangeDistribution(before);
50 +         Pair<Double, Double> rangeAfter = getRangeDistribution(after);
51 +
52 +         double min = Math.min(rangeBefore.getFirst(), rangeAfter.getFirst());
53 +         double max = Math.max(rangeBefore.getSecond(), rangeAfter.getSecond());
54 +
55 +         return new Pair<Double, Double>(min, max);
56 +     }
57 +
58 +     private Pair<Double, Double> getRangeDistribution(List<Double> list) {
59 +         DescriptiveStatistics ds = new DescriptiveStatistics(ListToArray.toPrimitive(list));
60 +         double q1 = ds.getPercentile(25.0);
61 +         double q3 = ds.getPercentile(75.0);
62 +         double iqr = q3 - q1;
63 +
64 +         double max = q3 + RANGE_OUTLIER_FACTOR * iqr;
65 +         double min = q1 - RANGE_OUTLIER_FACTOR * iqr;
66 +
67 +         if (min < 0) {
68 +             min = 0.0;
69 +         }
70 +         return new Pair<Double, Double>(min, max);
71 +     }
72 + }
```



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

...

...

@@ -0,0 +1,21 @@  
1 + package org.openjdk.jmh.reconfigure.statistics.divergence;  
2 +  
3 + public class KullbackLeiblerDivergence {  
4 + public static double continuous(double[] x, double[] y, double width) {  
5 + boolean intersection = false;  
6 + double kl = 0.0;  
7 +  
8 + for (int i = 0; i < x.length; i++) {  
9 + if (x[i] != 0.0 && y[i] != 0.0) {  
10 + intersection = true;  
11 + kl += x[i] \* Math.log(x[i] / y[i]) \* width;  
12 + }  
13 + }  
14 +  
15 + if (intersection) {  
16 + return kl;  
17 + } else {  
18 + return Double.POSITIVE\_INFINITY;  
19 + }  
20 + }  
21 + }

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

...

...

@@ -0,0 +1,20 @@  
1 + package org.openjdk.jmh.reconfigure.statistics.divergence;  
2 +  
3 + import org.openjdk.jmh.reconfigure.helper.ListToArray;  
4 + import smile.stat.distribution.KernelDensity;  
5 +  
6 + import java.util.ArrayList;  
7 + import java.util.List;  
8 +  
9 + public class ProbabilityDensityFunction {  
10 + public static double[] estimate(List<Double> sample, List<Double> y) {  
11 + KernelDensity kd = new KernelDensity(ListToArray.toPrimitive(sample));  
12 + List<Double> result = new ArrayList<>();  
13 +  
14 + for (double value : y) {  
15 + result.add(kd.p(value));  
16 + }  
17 +  
18 + return ListToArray.toPrimitive(result);  
19 + }  
20 + }

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

...


...

@@ -0,0 +1,89 @@  
1 + package org.openjdk.jmh.reconfigure.statistics.evaluation;  
2 +  
3 + import org.openjdk.jmh.reconfigure.helper.HistogramItem;  
4 + import org.openjdk.jmh.reconfigure.helper.OutlierDetector;  
5 + import org.openjdk.jmh.reconfigure.statistics.Sampler;  
6 + import org.openjdk.jmh.reconfigure.statistics.ci.CiPercentage;  
7 +  
8 + import java.util.\*;  
9 +  
10 + import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.\*;  
11 +  
12 + public class CiPercentageEvaluation implements StatisticalEvaluation {  
13 + private double threshold;  
14 + private double historySize;  
15 +  
16 + private List<HistogramItem> allMeasurements = new ArrayList<>();  
17 + private Map<Integer, List<HistogramItem>> sampleInIteration = new HashMap<>();

```
18 +     private Map<Integer, Double> ciPercentagePerIteration = new HashMap<>();
19 +
20 +     private CiPercentageEvaluation(double threshold, int historySize) {
21 +         this.threshold = threshold;
22 +         this.historySize = historySize;
23 +     }
24 +
25 +     public static CiPercentageEvaluation getIterationInstance(double threshold) {
26 +         return new CiPercentageEvaluation(threshold, 5);
27 +     }
28 +
29 +     public static CiPercentageEvaluation getForkInstance(double threshold) {
30 +         return new CiPercentageEvaluation(threshold, 2);
31 +     }
32 +
33 +     @Override
34 +     public void addIteration(List<HistogramItem> list) {
35 +         OutlierDetector od = new OutlierDetector(OUTLIER_FACTOR, list);
36 +         od.run();
37 +         List<HistogramItem> sample = new Sampler(od.getInlier()).getSample(SAMPLE_SIZE);
38 +
39 +         int iteration = sampleInIteration.size() + 1;
40 +         allMeasurements.addAll(sample);
41 +
42 +         List<HistogramItem> sampleUntil = new Sampler(allMeasurements).getSample(SAMPLE_SIZE);
43 +         sampleInIteration.put(iteration, sampleUntil);
44 +     }
45 +
46 +     @Override
47 +     public double getThreshold() {
48 +         return threshold;
49 +     }
50 +
51 +     @Override
52 +     public Double calculateVariability() {
53 +         if (sampleInIteration.size() < historySize) {
54 +             return null;
55 +         } else {
56 +             List<Double> deltas = new ArrayList<>();
57 +             int currentIteration = sampleInIteration.size();
58 +             double currentCiPercentage = getCiPercentageOfIteration(currentIteration);
59 +
60 +             for (int i = 1; i <= historySize - 1; i++) {
61 +                 double ciPercentage = getCiPercentageOfIteration(currentIteration - i);
62 +                 double delta = Math.abs(ciPercentage - currentCiPercentage);
63 +                 deltas.add(delta);
64 +             }
65 +
66 +             return Collections.max(deltas);
67 +         }
68 +     }
69 +
70 +     public double getCiPercentageOfIteration(int iteration) {
71 +         if (ciPercentagePerIteration.get(iteration) == null) {
72 +             double ciPercentage = new CiPercentage(sampleInIteration.get(iteration)).getValue();
73 +             ciPercentagePerIteration.put(iteration, ciPercentage);
74 +             return ciPercentage;
75 +         } else {
76 +             return ciPercentagePerIteration.get(iteration);
77 +         }
78 +     }
79 +
80 +     @Override
81 +     public int getIterationNumber() {
82 +         return sampleInIteration.size();
83 +     }
84 +
85 +     @Override
86 +     public boolean stableEnvironment(Double value) {
87 +         return value != null && value < threshold;
```



88 + }

89 + } 

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

...  
1 + package org.openjdk.jmh.reconfigure.statistics.evaluation;  
2 +  
3 + import org.openjdk.jmh.reconfigure.helper.HistogramItem;  
4 + import org.openjdk.jmh.reconfigure.helper.OutlierDetector;  
5 + import org.openjdk.jmh.reconfigure.statistics.COV;  
6 + import org.openjdk.jmh.reconfigure.statistics.Sampler;  
7 +  
8 + import java.util.\*;  
9 +  
10 + import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.OUTLIER\_FACTOR;  
11 + import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.SAMPLE\_SIZE;  
12 +  
13 + public class CovEvaluation implements StatisticalEvaluation {  
14 + private double threshold;  
15 + private double historySize;  
16 +  
17 + private Map<Integer, List<HistogramItem>> samplePerIteration = new HashMap<>();  
18 + private Map<Integer, Double> covPerIteration = new HashMap<>();  
19 +  
20 + private CovEvaluation(double threshold, int historySize) {  
21 + this.threshold = threshold;  
22 + this.historySize = historySize;  
23 + }  
24 +  
25 + public static CovEvaluation getIterationInstance(double threshold) {  
26 + return new CovEvaluation(threshold, 5);  
27 + }  
28 +  
29 + public static CovEvaluation getForkInstance(double threshold) {  
30 + return new CovEvaluation(threshold, 2);  
31 + }  
32 +  
33 + @Override  
34 + public void addIteration(List<HistogramItem> list) {  
35 + OutlierDetector od = new OutlierDetector(OUTLIER\_FACTOR, list);  
36 + od.run();  
37 + List<HistogramItem> sample = new Sampler(od.getInlier()).getSample(SAMPLE\_SIZE);  
38 +  
39 + int iteration = samplePerIteration.size() + 1;  
40 + samplePerIteration.put(iteration, sample);  
41 + }  
42 +  
43 + @Override  
44 + public double getThreshold() {  
45 + return threshold;  
46 + }  
47 +  
48 + @Override  
49 + public Double calculateVariability() {  
50 + if (samplePerIteration.size() < historySize) {  
51 + return null;  
52 + } else {  
53 + List<Double> deltas = new ArrayList<>();  
54 + int currentIteration = samplePerIteration.size();  
55 + double currentCov = getCovOfIteration(currentIteration);  
56 +  
57 + for (int i = 1; i <= historySize - 1; i++) {  
58 + double cov = getCovOfIteration(currentIteration - i);  
59 + double delta = Math.abs(cov - currentCov);  
60 + deltas.add(delta);  
61 + }  
62 +  
63 + return Collections.max(deltas);  
64 + }  
...

```
65 +     }
66 +
67 +     public double getCovOfIteration(int iteration) {
68 +         if (covPerIteration.get(iteration) == null) {
69 +             List<HistogramItem> all = new ArrayList<>();
70 +             for (int i = 1; i <= iteration; i++) {
71 +                 all.addAll(samplePerIteration.get(i));
72 +             }
73 +
74 +             double cov = new COV(all).getValue();
75 +             covPerIteration.put(iteration, cov);
76 +             return cov;
77 +         } else {
78 +             return covPerIteration.get(iteration);
79 +         }
80 +     }
81 +
82 +     @Override
83 +     public int getIterationNumber() {
84 +         return samplePerIteration.size();
85 +     }
86 +
87 +     @Override
88 +     public boolean stableEnvironment(Double value) {
89 +         return value != null && value < threshold;
90 +     }
91 + }
```

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

```
...  ... @@ -0,0 +1,119 @@
1 + package org.openjdk.jmh.reconfigure.statistics.evaluation;
2 +
3 + import org.openjdk.jmh.reconfigure.helper.HistogramHelper;
4 + import org.openjdk.jmh.reconfigure.helper.HistogramItem;
5 + import org.openjdk.jmh.reconfigure.helper.OutlierDetector;
6 + import org.openjdk.jmh.reconfigure.statistics.Sampler;
7 + import org.openjdk.jmh.reconfigure.statistics.divergence.Divergence;
8 +
9 + import java.io.OutputStream;
10 + import java.io.PrintStream;
11 + import java.util.*;
12 +
13 + import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.OUTLIER_FACTOR;
14 + import static org.openjdk.jmh.reconfigure.statistics.ReconfigureConstants.SAMPLE_SIZE;
15 +
16 + public class DivergenceEvaluation implements StatisticalEvaluation {
17 +     private double threshold;
18 +     private double historySize;
19 +
20 +     private List<Double> allMeasurements = new ArrayList<>();
21 +     private Map<Integer, List<Double>> sampleUntilIteration = new HashMap<>();
22 +     private Map<Integer, Double> pValuePerIteration = new HashMap<>();
23 +
24 +     private DivergenceEvaluation(double threshold, int historySize) {
25 +         this.threshold = threshold;
26 +         this.historySize = historySize;
27 +         disableSystemErr();
28 +     }
29 +
30 +     public static DivergenceEvaluation getIterationInstance(double threshold) {
31 +         return new DivergenceEvaluation(threshold, 6);
32 +     }
33 +
34 +     public static DivergenceEvaluation getForkInstance(double threshold) {
35 +         return new DivergenceEvaluation(threshold, 2);
36 +     }
37 +
38 +     @Override
39 +     public void addIteration(List<HistogramItem> list) {
```

```
40 +         OutlierDetector od = new OutlierDetector(OUTLIER_FACTOR, list);
41 +         od.run();
42 +         List<HistogramItem> sample = new Sampler(od.getInlier()).getSample(SAMPLE_SIZE);
43 +
44 +         int iteration = sampleUntilIteration.size() + 1;
45 +         List<Double> newValues = HistogramHelper.toArray(sample);
46 +         allMeasurements.addAll(newValues);
47 +
48 +         List<Double> sampleUntil = getSample(allMeasurements);
49 +         sampleUntilIteration.put(iteration, sampleUntil);
50 +     }
51 +
52 +     @Override
53 +     public double getThreshold() {
54 +         return threshold;
55 +     }
56 +
57 +     @Override
58 +     public Double calculateVariability() {
59 +         if (sampleUntilIteration.size() < historySize) {
60 +             return null;
61 +         } else {
62 +             List<Double> pvalues = new ArrayList<>();
63 +             int currentIteration = sampleUntilIteration.size();
64 +
65 +             for (int i = 0; i <= historySize - 2; i++) {
66 +                 Double pvalue = getPValueOfIteration(currentIteration - i);
67 +                 pvalues.add(pvalue);
68 +             }
69 +
70 +             return pvalues.stream().mapToDouble(it -> it).average().orElse(0.0);
71 +         }
72 +     }
73 +
74 +     public Double getPValueOfIteration(int iteration) {
75 +         if (pValuePerIteration.get(iteration) == null) {
76 +             List<Double> currentSample = sampleUntilIteration.get(iteration);
77 +             List<Double> previousSample = sampleUntilIteration.get(iteration - 1);
78 +             if (currentSample == null || previousSample == null) {
79 +                 return null;
80 +             } else {
81 +                 double pValue = new Divergence(currentSample, previousSample).getValue();
82 +                 pValuePerIteration.put(iteration, pValue);
83 +                 return pValue;
84 +             }
85 +         } else {
86 +             return pValuePerIteration.get(iteration);
87 +         }
88 +     }
89 +
90 +     private List<Double> getSample(List<Double> list) {
91 +         Random random = new Random();
92 +         List<Double> sample = new ArrayList<>();
93 +
94 +         for (int i = 0; i < SAMPLE_SIZE; i++) {
95 +             Double d = list.get(random.nextInt(list.size()));
96 +             sample.add(d);
97 +         }
98 +
99 +         Collections.sort(sample);
100 +         return sample;
101 +     }
102 +
103 +     private void disableSystemErr() {
104 +         System.setErr(new PrintStream(new OutputStream() {
105 +             public void write(int b) {
106 +             }
107 +         }));
108 +     }
109 + }
```

```
110 + @Override
111 + public int getIterationNumber() {
112 +     return sampleUntilIteration.size();
113 + }
114 +
115 + @Override
116 + public boolean stableEnvironment(Double value) {
117 +     return value != null && value > threshold;
118 + }
119 + }
```

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

```
...    ... @@ -0,0 +1,17 @@
      1 + package org.openjdk.jmh.reconfigure.statistics.evaluation;
      2 +
      3 + import org.openjdk.jmh.reconfigure.helper.HistogramItem;
      4 +
      5 + import java.util.List;
      6 +
      7 + public interface StatisticalEvaluation {
      8 +     void addIteration(List<HistogramItem> list);
      9 +
     10 +     double getThreshold();
     11 +
     12 +     Double calculateVariability();
     13 +
     14 +     int getIterationNumber();
     15 +
     16 +     boolean stableEnvironment(Double value);
     17 + }
```

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

```
25    25     package org.openjdk.jmh.results;
26    26
27    27     import java.io.Serializable;
28    28 + import java.util.List;
29    29
30    30     public class BenchmarkResultMetaData implements Serializable {
31    31
32    32         private final long stopTime;
33    33         private final long warmupOps;
34    34         private final long measurementOps;
35    35 + private final List<Double> warmupThresholds;
36    36 + private final boolean atLeastOneWarning;
37    37
38    38 - public BenchmarkResultMetaData(long warmupTime, long measurementTime, long stopTime, long warmupOps, long measurementOps,
39    39 + public BenchmarkResultMetaData(long warmupTime, long measurementTime, long stopTime, long warmupOps, long measurementOps,
40    40         this.startTime = Long.MIN_VALUE;
41    41         this.warmupTime = warmupTime;
42    42         this.measurementTime = measurementTime;
43    43         this.stopTime = stopTime;
44    44         this.warmupOps = warmupOps;
45    45         this.measurementOps = measurementOps;
46    46 +         this.warmupThresholds = warmupThresholds;
47    47 +         this.atLeastOneWarning = atLeastOneWarning;
48    48     }
49    49
50    50
51    51
52    52     public long getStartTime() {
53    53         return warmupOps;
54    54     }
55    55
56    56
57    57
58    58
59    59
60    60
61    61
62    62
63    63
64    64
65    65
66    66
67    67
68    68
69    69 + public List<Double> getWarmupThresholds() {
70    70 +     return warmupThresholds;
71    71 + }
72    72 +
73    73 + public boolean hasAtLeastOneWarning() {
74    74 +     return atLeastOneWarning;
75    75 + }
```

		86	+
74		87	public void adjustStart(long startTime) {
75		88	this.startTime = startTime;
76		89	}

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

27	27	import org.openjdk.jmh.infra.BenchmarkParams;
28	28	
29	29	import java.io.Serializable;
30		- import java.util.ArrayList;
31		- import java.util.Collection;
32		- import java.util.Comparator;
33		- import java.util.Map;
	30	+ import java.util.*;
34	31	
35	32	/**
36	33	* Complete run result.
42	39	
43	40	private final Collection<BenchmarkResult> benchmarkResults;
44	41	private final BenchmarkParams params;
	42	+ private final List<Double> warmupThresholds;
	43	+ private final List<Double> measurementThresholds;
45	44	
46		- public RunResult(BenchmarkParams params, Collection<BenchmarkResult> data) {
	45	+ public RunResult(BenchmarkParams params, Collection<BenchmarkResult> data, <u>List&lt;Double&gt; warmupThresholds, List&lt;Dou</u>
47	46	this.benchmarkResults = data;
48	47	this.params = params;
	48	+     this.warmupThresholds = warmupThresholds;
	49	+     this.measurementThresholds = measurementThresholds;
49	50	}
50	51	
51	52	public Collection<BenchmarkResult> getBenchmarkResults() {
92	93	return params;
93	94	}
94	95	
	96	+ public List<Double> getWarmupThresholds() {
	97	+     return warmupThresholds;
	98	+ }
	99	+
	100	+ public List<Double> getMeasurementThresholds() {
	101	+     return measurementThresholds;
	102	+ }
	103	+
95	104	public static final Comparator<RunResult> DEFAULT_SORT_COMPARATOR = new Comparator<RunResult>() {
96	105	@Override
97	106	public int compare(RunResult o1, RunResult o2) {
98	107	return o1.params.compareTo(o2.params);
99	108	}
100	109	};
101		-
102	110	}

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

24	24	*/
25	25	package org.openjdk.jmh.results.format;
26	26	
	27	+ import org.openjdk.jmh.annotations.Mode;
27	28	import org.openjdk.jmh.infra.BenchmarkParams;
28	29	import org.openjdk.jmh.results.BenchmarkResult;
29	30	import org.openjdk.jmh.results.IterationResult;
37	38	import java.io.StringWriter;
38	39	import java.util.ArrayList;
39	40	import java.util.Collection;
	41	+ import java.util.List;
40	42	import java.util.Map;
41	43	
42	44	class JSONResultFormat implements ResultFormat {
60	62	pw.println("[");

```
61      63      for (RunResult runResult : results) {
62      64      BenchmarkParams params = runResult.getParams();
63      65      +      boolean isReconfigureMode = params.getMode().equals(Mode.Reconfigure);
64      66
65      67      if (first) {
66      68      first = false;
67      69
68      70      pw.println("\"mode\" : \"" + params.getMode().shortLabel() + "\",");
69      71      pw.println("\"threads\" : " + params.getThreads() + ",");
70      72      pw.println("\"forks\" : " + params.getForks() + ",");
71      73
72      74      +      if (isReconfigureMode) {
73      75      +      pw.println("\"minForks\" : " + params.getMinForks() + ",");
74      76      +      }
75      77      pw.println("\"warmupForks\" : " + params.getWarmupForks() + ",");
76      78      +      if (isReconfigureMode) {
77      79      +      pw.println("\"minWarmupForks\" : " + params.getMinWarmupForks() + ",");
78      80      +      }
79      81      pw.println("\"jvm\" : " + toJsonString(params.getJvm()) + ",");
80      82      // if empty, write an empty array.
81      83      pw.println("\"jvmArgs\" : []");
82      84      pw.println("\"vmName\" : " + toJsonString(params.getVmName()) + ",");
83      85      pw.println("\"vmVersion\" : " + toJsonString(params.getVmVersion()) + ",");
84      86      pw.println("\"warmupIterations\" : " + params.getWarmup().getCount() + ",");
85      87      +      if (isReconfigureMode) {
86      88      +      pw.println("\"minWarmupIterations\" : " + params.getWarmup().getMinCount() + ",");
87      89      +      }
88      90      pw.println("\"warmupTime\" : \"" + params.getWarmup().getTime() + "\",");
89      91      pw.println("\"warmupBatchSize\" : " + params.getWarmup().getBatchSize() + ",");
90      92      pw.println("\"measurementIterations\" : " + params.getMeasurement().getCount() + ",");
91      93      pw.println("},");
92      94      }
93      95
94      96      +      if (isReconfigureMode) {
95      97      +      pw.println("\"reconfigureMode\" : " + toJsonString(params.getReconfigureMode().shortLabel()) + ",");
96      98      +      pw.println("\"reconfigureThreshold\" : " + params.getReconfigureThreshold() + ",");
97      99      +
98      100      +      pw.println("\"thresholds\" : {}");
99      101      +
100     102      +      pw.println("\"warmupForks\" : ");
101     103      +      Collection<String> warmupForkThresholds = toListOfStrings(runResult.getWarmupThresholds());
102     104      +      pw.println(printMultiple(warmupForkThresholds, "[", "]"));
103     105      +      pw.println(",");
104     106      +
105     107      +      pw.println("\"measurementForks\" : ");
106     108      +      Collection<String> measurementForkThresholds = toListOfStrings(runResult.getMeasurementThresholds());
107     109      +      pw.println(printMultiple(measurementForkThresholds, "[", "]"));
108     110      +      pw.println(",");
109     111      +
110     112      +      Collection<String> warmupIterationList = new ArrayList<>();
111     113      +      for (BenchmarkResult benchmarkResult : runResult.getBenchmarkResults()) {
112     114      +      Collection<String> warmupIterationThresholds = toListOfStrings(benchmarkResult.getMetadata().getWa
113     115      +      warmupIterationList.add(printMultiple(warmupIterationThresholds, "[", "]"));
114     116      +      }
115     117      +
116     118      +      pw.println("\"warmupIterations\" : {}");
117     119      +      pw.println(printMultiple(warmupIterationList, "[", "]"));
118     120      +      pw.println("},");
119     121      +
120     122      +      pw.println("},");
121     123      +
122     124      +      pw.println("\"warnings\" : {}");
123     125      +      pw.println(tidy(getThresholdWarnings(runResult)));
124     126      +      pw.println("},");
125     127      +      }
126     128      +
127     129      Result primaryResult = runResult.getPrimaryResult();
128     130      pw.println("\"primaryMetric\" : {}");
129     131      pw.println("\"score\" : " + emit(primaryResult.getScore()) + ",");
130     132
131     133
132     134
133     135
134     136
135     137
136     138
137     139
138     140
139     141
140     142
141     143
142     144
143     145
144     146
145     147
146     148
147     149
148     150
149     151
150     152
151     153
152     154
153     155
154     156
155     157
156     158
157     159
158     160
159     161
160     162
161     163
162     164
163     165
164     166
165     167
166     168
167     169
168     170
169     171
170     172
171     173
172     174
173     175
174     176
175     177
176     178
177     179
178     180
179     181
180     182
181     183
182     184
183     185
184     186
185     187
186     188
187     189
188     190
189     191
190     192
191     193
192     194
193     195
194     196
195     197
196     198
197     199
198     200
199     201
200     202
201     203
202     204
203     205
204     206
205     207
206     208
207     209
208     210
209     211
210     212
211     213
212     214
213     215
214     216
215     217
216     218
217     219
218     220
219     221
220     222
221     223
222     224
223     225
224     226
225     227
226     228
227     229
228     230
229     231
230     232
231     233
232     234
233     235
234     236
235     237
236     238
237     239
238     240
239     241
240     242
241     243
242     244
243     245
244     246
245     247
246     248
247     249
248     250
249     251
250     252
251     253
252     254
253     255
254     256
255     257
256     258
257     259
258     260
259     261
260     262
261     263
262     264
263     265
264     266
265     267
266     268
267     269
268     270
269     271
270     272
271     273
272     274
273     275
274     276
275     277
276     278
277     279
278     280
279     281
280     282
281     283
282     284
283     285
284     286
285     287
286     288
287     289
288     290
289     291
290     292
291     293
292     294
293     295
294     296
295     297
296     298
297     299
298     300
299     301
300     302
301     303
302     304
303     305
304     306
305     307
306     308
307     309
308     310
309     311
310     312
311     313
312     314
313     315
314     316
315     317
316     318
317     319
318     320
319     321
320     322
321     323
322     324
323     325
324     326
325     327
326     328
327     329
328     330
329     331
330     332
331     333
332     334
333     335
334     336
335     337
336     338
337     339
338     340
339     341
340     342
341     343
342     344
343     345
344     346
345     347
346     348
347     349
348     350
349     351
350     352
351     353
352     354
353     355
354     356
355     357
356     358
357     359
358     360
359     361
360     362
361     363
362     364
363     365
364     366
365     367
366     368
367     369
368     370
369     371
370     372
371     373
372     374
373     375
374     376
375     377
376     378
377     379
378     380
379     381
380     382
381     383
382     384
383     385
384     386
385     387
386     388
387     389
388     390
389     391
390     392
391     393
392     394
393     395
394     396
395     397
396     398
397     399
398     400
399     401
400     402
401     403
402     404
403     405
404     406
405     407
406     408
407     409
408     410
409     411
410     412
411     413
412     414
413     415
414     416
415     417
416     418
417     419
418     420
419     421
420     422
421     423
422     424
423     425
424     426
425     427
426     428
427     429
428     430
429     431
430     432
431     433
432     434
433     435
434     436
435     437
436     438
437     439
438     440
439     441
440     442
441     443
442     444
443     445
444     446
445     447
446     448
447     449
448     450
449     451
450     452
451     453
452     454
453     455
454     456
455     457
456     458
457     459
458     460
459     461
460     462
461     463
462     464
463     465
464     466
465     467
466     468
467     469
468     470
469     471
470     472
471     473
472     474
473     475
474     476
475     477
476     478
477     479
478     480
479     481
480     482
481     483
482     484
483     485
484     486
485     487
486     488
487     489
488     490
489     491
490     492
491     493
492     494
493     495
494     496
495     497
496     498
497     499
498     500
499     501
500     502
501     503
502     504
503     505
504     506
505     507
506     508
507     509
508     510
509     511
510     512
511     513
512     514
513     515
514     516
515     517
516     518
517     519
518     520
519     521
520     522
521     523
522     524
523     525
524     526
525     527
526     528
527     529
528     530
529     531
530     532
531     533
532     534
533     535
534     536
535     537
536     538
537     539
538     540
539     541
540     542
541     543
542     544
543     545
544     546
545     547
546     548
547     549
548     550
549     551
550     552
551     553
552     554
553     555
554     556
555     557
556     558
557     559
558     560
559     561
560     562
561     563
562     564
563     565
564     566
565     567
566     568
567     569
568     570
569     571
570     572
571     573
572     574
573     575
574     576
575     577
576     578
577     579
578     580
579     581
580     582
581     583
582     584
583     585
584     586
585     587
586     588
587     589
588     590
589     591
590     592
591     593
592     594
593     595
594     596
595     597
596     598
597     599
598     600
599     601
600     602
601     603
602     604
603     605
604     606
605     607
606     608
607     609
608     610
609     611
610     612
611     613
612     614
613     615
614     616
615     617
616     618
617     619
618     620
619     621
620     622
621     623
622     624
623     625
624     626
625     627
626     628
627     629
628     630
629     631
630     632
631     633
632     634
633     635
634     636
635     637
636     638
637     639
638     640
639     641
640     642
641     643
642     644
643     645
644     646
645     647
646     648
647     649
648     650
649     651
650     652
651     653
652     654
653     655
654     656
655     657
656     658
657     659
658     660
659     661
660     662
661     663
662     664
663     665
664     666
665     667
666     668
667     669
668     670
669     671
670     672
671     673
672     674
673     675
674     676
675     677
676     678
677     679
678     680
679     681
680     682
681     683
682     684
683     685
684     686
685     687
686     688
687     689
688     690
689     691
690     692
691     693
692     694
693     695
694     696
695     697
696     698
697     699
698     700
699     701
700     702
701     703
702     704
703     705
704     706
705     707
706     708
707     709
708     710
709     711
710     712
711     713
712     714
713     715
714     716
715     717
716     718
717     719
718     720
719     721
720     722
721     723
722     724
723     725
724     726
725     727
726     728
727     729
728     730
729     731
730     732
731     733
732     734
733     735
734     736
735     737
736     738
737     739
738     740
739     741
740     742
741     743
742     744
743     745
744     746
745     747
746     748
747     749
748     750
749     751
750     752
751     753
752     754
753     755
754     756
755     757
756     758
757     759
758     760
759     761
760     762
761     763
762     764
763     765
764     766
765     767
766     768
767     769
768     770
769     771
770     772
771     773
772     774
773     775
774     776
775     777
776     778
777     779
778     780
779     781
780     782
781     783
782     784
783     785
784     786
785     787
786     788
787     789
788     790
789     791
790     792
791     793
792     794
793     795
794     796
795     797
796     798
797     799
798     800
799     801
800     802
801     803
802     804
803     805
804     806
805     807
806     808
807     809
808     810
809     811
810     812
811     813
812     814
813     815
814     816
815     817
816     818
817     819
818     820
819     821
820     822
821     823
822     824
823     825
824     826
825     827
826     828
827     829
828     830
829     831
830     832
831     833
832     834
833     835
834     836
835     837
836     838
837     839
838     840
839     841
840     842
841     843
842     844
843     845
844     846
845     847
846     848
847     849
848     850
849     851
850     852
851     853
852     854
853     855
854     856
855     857
856     858
857     859
858     860
859     861
860     862
861     863
862     864
863     865
864     866
865     867
866     868
867     869
868     870
869     871
870     872
871     873
872     874
873     875
874     876
875     877
876     878
877     879
878     880
879     881
880     882
881     883
882     884
883     885
884     886
885     887
886     888
887     889
888     890
889     891
890     892
891     893
892     894
893     895
894     896
895     897
896     898
897     899
898     900
899     901
900     902
901     903
902     904
903     905
904     906
905     907
906     908
907     909
908     910
909     911
910     912
911     913
912     914
913     915
914     916
915     917
916     918
917     919
918     920
919     921
920     922
921     923
922     924
923     925
924     926
925     927
926     928
927     929
928     930
929     931
930     932
931     933
932     934
933     935
934     936
935     937
936     938
937     939
938     940
939     941
940     942
941     943
942     944
943     945
944     946
945     947
946     948
947     949
948     950
949     951
950     952
951     953
952     954
953     955
954     956
955     957
956     958
957     959
958     960
959     961
960     962
961     963
962     964
963     965
964     966
965     967
966     968
967     969
968     970
969     971
970     972
971     973
972     974
973     975
974     976
975     977
976     978
977     979
978     980
979     981
980     982
981     983
982     984
983     985
984     986
985     987
986     988
987     989
988     990
989     991
990     992
991     993
992     994
993     995
994     996
995     997
996     998
997     999
998     1000
999     1001
1000    1002
1001    1003
1002    1004
1003    1005
1004    1006
1005    1007
1006    1008
1007    1009
1008    1010
1009    1011
1010    1012
1011    1013
1012    1014
1013    1015
1014    1016
1015    1017
1016    1018
1017    1019
1018    1020
1019    1021
1020    1022
1021    1023
1022    1024
1023    1025
1024    1026
1025    1027
1026    1028
1027    1029
1028    1030
1029    1031
1030    1032
1031    1033
1032    1034
1033    1035
1034    1036
1035    1037
1036    1038
1037    1039
1038    1040
1039    1041
1040    1042
1041    1043
1042    1044
1043    1045
1044    1046
1045    1047
1046    1048
1047    1049
1048    1050
1049    1051
1050    1052
1051    1053
1052    1054
1053    1055
1054    1056
1055    1057
1056    1058
1057    1059
1058    1060
1059    1061
1060    1062
1061    1063
1062    1064
1063    1065
1064    1066
1065    1067
1066    1068
1067    1069
1068    1070
1069    1071
1070    1072
1071    1073
1072    1074
1073    1075
1074    1076
1075    1077
1076    1078
1077    1079
1078    1080
1079    1081
1080    1082
1081    1083
1082    1084
1083    1085
1084    1086
1085    1087
1086    1088
1087    1089
1088    1090
1089    1091
1090    1092
1091    1093
1092    1094
1093    1095
1094    1096
1095    1097
1096    1098
1097    1099
1098    1100
1099    1101
1100    1102
1101    1103
1102    1104
1103    1105
1104    1106
1105    1107
1106    1108
1107    1109
1108    1110
1109    1111
1110    1112
1111    1113
1112    1114
1113    1115
1114    1116
1115    1117
1116    1118
1117    1119
1118    1120
1119    1121
1120    1122
1121    1123
1122    1124
1123    1125
1124    1126
1125    1127
1126    1128
1127    1129
1128    1130
1129    1131
1130    1132
1131    1133
1132    1134
1133    1135
1134    1136
1135    1137
1136    1138
1137    1139
1138    1140
1139    1141
1140    1142
1141    1143
1142    1144
1143    1145
1144    1146
1145    1147
1146    1148
1147    1149
1148    1150
1149    1151
1150    1152
1151    1153
1152    1154
1153    1155
1154    1156
1155    1157
1156    1158
1157    1159
1158    1160
1159    1161
1160    1162
1161    1163
1162    1164
1163    1165
1164    1166
1165    1167
1166    1168
1167    1169
1168    1170
1169    1171
1170    1172
1171    1173
1172    1174
1173    1175
1174    1176
1175    1177
1176    1178
1177    1179
1178    1180
1179    1181
1180    1182
1181    1183
1182    1184
1183    1185
1184    1186
1185    1187
1186    1188
1187    1189
1188    1190
1189    1191
1190    1192
1191    1193
1192    1194
1193    1195
1194    1196
1195    1197
1196    1198
1197    1199
1198    1200
1199    1201
1200    1202
1201    1203
1202    1204
1203    1205
1204    1206
1205    1207
1206    1208
1207    1209
1208    1210
1209    1211
1210    1212
1211    1213
1212    1214
1213    1215
1214    1216
1215    1217
1216    1218
1217    1219
1218    1220
1219    1221
1220    1222
1221    1223
1222    1224
1223    1225
1224    1226
1225    1227
1226    1228
1227    1229
1228    1230
1229    1231
1230    1232
1231    1233
1232    1234
1233    1235
1234    1236
1235    1237
1236    1238
1237    1239
1238    1240
1239    1241
1240    1242
1241    1243
1242    1244
1243    1245
1244    1246
1245    1247
1246    1248
1247    1249
1248    1250
1249    1251
1250    1252
1251    1253
1252    1254
1253    1255
1254    1256
1255    1257
1256    1258
1257    1259
1258    1260
1259    1261
1260    1262
1261    1263
1262    1264
1263    1265
1264    1266
1265    1267
1266    1268
1267    1269
1268    1270
1269    1271
1270    1272
1271    1273
1272    1274
1273    1275
1274    1276
1275    1277

```



```
154 +         case Reconfigure:
108 155             pw.println("\rawDataHistogram" :");
109 156             pw.println(getRawData(runResult, true));
110 157             break;
182 229         return sb.toString();
183 230     }
184 231
232 + private String getThresholdWarnings(RunResult runResult) {
233 +     BenchmarkParams params = runResult.getParams();
234 +     double threshold = params.getReconfigureThreshold();
235 +
236 +     StringBuilder sb = new StringBuilder();
237 +
238 +     boolean warmupForkHasWarning = false;
239 +     if (runResult.getWarmupThresholds().size() > 0) {
240 +         Double lastWarmupForkItem = runResult.getWarmupThresholds().get(runResult.getWarmupThresholds().size() - 1);
241 +         warmupForkHasWarning = runResult.getWarmupThresholds().size() == params.getMinWarmupForks() && lastWarmupForkItem > threshold;
242 +     }
243 +     sb.append("\warmupForks" : " + warmupForkHasWarning + ",");
244 +
245 +     boolean measurementForkJasWarning = false;
246 +     if (runResult.getMeasurementThresholds().size() > 0) {
247 +         Double lastMeasurementForkItem = runResult.getMeasurementThresholds().get(runResult.getMeasurementThresholds().size() - 1);
248 +         measurementForkJasWarning = runResult.getMeasurementThresholds().size() == params.getMinForks() && lastMeasurementForkItem > threshold;
249 +     }
250 +     sb.append("\measurementForks" : " + measurementForkJasWarning + ",");
251 +
252 +     Collection<String> warmupIterationList = new ArrayList<>();
253 +     for (BenchmarkResult benchmarkResult : runResult.getBenchmarkResults()) {
254 +         List<Double> warmupThresholds = benchmarkResult.getMetadata().getWarmupThresholds();
255 +         if (warmupThresholds.size() > 0) {
256 +             Double lastWarmupIterationItem = warmupThresholds.get(warmupThresholds.size() - 1);
257 +             boolean warmupIterationHasWarning = warmupThresholds.size() == params.getWarmup().getCount() && lastWarmupIterationItem > threshold;
258 +             warmupIterationList.add(warmupIterationHasWarning ? "true" : "false");
259 +         }
260 +     }
261 +
262 +     sb.append("\warmupIterations" : ");
263 +     sb.append(printMultiple(warmupIterationList, "[", "]"));
264 +     sb.append(",");
265 +
266 +     boolean totalHasWarning = warmupForkHasWarning || measurementForkJasWarning || warmupIterationList.contains("true");
267 +     sb.append("\total" : " + totalHasWarning);
268 +
269 +     return sb.toString();
270 + }
271 +
185 272 private String emitParams(BenchmarkParams params) {
186 273     StringBuilder sb = new StringBuilder();
187 274     boolean isFirst = true;
257 344     }
258 345     switch (c) {
259 346         // use & as escape character to escape the tidying
260 -         case '&': sb.append("&"); break;
261 +         case '&':
262 +             sb.append("&");
263 +             break;
261 350         // we cannot escape to \\ since this would create sequences interpreted by the tidying
262 -         case '\\': sb.append("&/"); break;
263 -         case "'": sb.append("&'"); break;
261 +         case '\\':
262 +             sb.append("&/");
263 +             break;
264 +         case "'":
265 +             sb.append("&'");
266 +             break;
264 357         // escape spacial chars for the tidying formatting below that might appear in a string
265 -         case ',': sb.append("&,""); break;
266 -         case '[': sb.append("&[""); break;
267 -         case ']': sb.append("&]"); break;
```



268		-	case '<': sb.append("&-"); break;
269		-	case '>': sb.append("&="); break;
270		-	case ';': sb.append("&:"); break;
271		-	case '{': sb.append("&("); break;
272		-	case '}': sb.append("&)"); break;
273		-	default: sb.append(c);
	358	+	case ',':
	359	+	sb.append(",");
	360	+	break;
	361	+	case '[':
	362	+	sb.append("<");
	363	+	break;
	364	+	case ']':
	365	+	sb.append(">");
	366	+	break;
	367	+	case '<':
	368	+	sb.append("&-");
	369	+	break;
	370	+	case '>':
	371	+	sb.append("&=");
	372	+	break;
	373	+	case ';':
	374	+	sb.append("&:");
	375	+	break;
	376	+	case '{':
	377	+	sb.append("&(");
	378	+	break;
	379	+	case '}':
	380	+	sb.append("&)");
	381	+	break;
	382	+	default:
	383	+	sb.append(c);
274	384		}
275	385		}
276	386		sb.append("\\");
356	466		}
357	467		}
358	468		
	469	+	private static List<String> toListOfStrings(List<Double> doubles) {
	470	+	List<String> strings = new ArrayList<>();
	471	+	for (Double threshold : doubles) {
	472	+	if (threshold == null) {
	473	+	strings.add(null);
	474	+	} else {
	475	+	strings.add(threshold.toString());
	476	+	}
	477	+	}
	478	+	return strings;
	479	+	}
359	480		}

▼ 13 ■■■■ jmh-core/src/main/java/org/openjdk/jmh/runner/BaseRunner.java 📄			
27	27		import org.openjdk.jmh.annotations.Mode;
28	28		import org.openjdk.jmh.infra.BenchmarkParams;
29	29		import org.openjdk.jmh.infra.IterationParams;
	30	+	import org.openjdk.jmh.reconfigure.manager.IterationReconfigureManager;
30	31		import org.openjdk.jmh.results.BenchmarkResult;
31	32		import org.openjdk.jmh.results.BenchmarkResultMetaData;
32	33		import org.openjdk.jmh.results.IterationResult;
35	36		import org.openjdk.jmh.util.Multimap;
36	37		import org.openjdk.jmh.util.TreeMultimap;
37	38		import org.openjdk.jmh.util.Utills;
38		-	import org.openjdk.jmh.util.Version;
39	39		
40	40		import java.lang.management.GarbageCollectorMXBean;
41	41		import java.lang.management.ManagementFactory;
249	249		long allWarmup = 0;
250	250		long allMeasurement = 0;
251	251		

	252	+	IterationReconfigureManager irm = new IterationReconfigureManager(benchParams, out);
	253	+	
252	254		// warmup
253	255		IterationParams wp = benchParams.getWarmup();
254	256		for (int i = 1; i <= wp.getCount(); i++) {
263	265		out.iterationResult(benchParams, wp, i, ir);
264	266		
265	267		allWarmup += ir.getMetadata().getAllOps();
	268	+	
	269	+	if (benchParams.getMode().equals(Mode.Reconfigure)) {
	270	+	irm.addWarmupIteration(i, ir);
	271	+	if (irm.checkWarmupIterationThreshold()) {
	272	+	break;
	273	+	}
	274	+	}
266	275		}
267	276		
268	277		long measurementTime = System.currentTimeMillis();
293	302		
294	303		BenchmarkResultMetaData md = new BenchmarkResultMetaData(
295	304		warmupTime, measurementTime, stopTime,
296		-	allWarmup, allMeasurement);
	305	+	allWarmup, allMeasurement, <u>irm.getWarmupThresholds(), irm.hasAtLeastOneWarning()</u> );
297	306		
298	307		if (acceptor != null) {
299	308		acceptor.acceptMeta(md);

▼ 118 jmh-core/src/main/java/org/openjdk/jmh/runner/BenchmarkListEntry.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.runner.options.TimeValue;
29	30		import org.openjdk.jmh.util.Optional;
30	31		import org.openjdk.jmh.util.lines.TestLineReader;
45	46		private final Optional<Collection<String>> threadGroupLabels;
46	47		private final Optional<Integer> threads;
47	48		private final Optional<Integer> warmupIterations;
	49	+	private final Optional<Integer> minWarmupIterations;
48	50		private final Optional<TimeValue> warmupTime;
49	51		private final Optional<Integer> warmupBatchSize;
50	52		private final Optional<Integer> measurementIterations;
51	53		private final Optional<TimeValue> measurementTime;
52	54		private final Optional<Integer> measurementBatchSize;
53	55		private final Optional<Integer> forks;
	56	+	private final Optional<Integer> minForks;
54	57		private final Optional<Integer> warmupForks;
	58	+	private final Optional<Integer> minWarmupForks;
	59	+	private final ReconfigureMode reconfigureMode;
	60	+	private final Optional<Double> reconfigureCovThreshold;
	61	+	private final Optional<Double> reconfigureCiThreshold;
	62	+	private final Optional<Double> reconfigureKldThreshold;
55	63		private final Optional<String> jvm;
56	64		private final Optional<Collection<String>> jvmArgs;
57	65		private final Optional<Collection<String>> jvmArgsPrepend;
65	73		
66	74		public BenchmarkListEntry(String userClassQName, String generatedClassQName, String method, Mode mode,
67	75		Optional<Integer> threads, int[] threadGroups, Optional<Collection<String>> threadGroupL
68		-	Optional<Integer> warmupIterations, Optional<TimeValue> warmupTime, Optional<Integer> wa
	76	+	Optional<Integer> warmupIterations, Optional<Integer> minWarmupIterations, Optional<Time
69	77		Optional<Integer> measurementIterations, Optional<TimeValue> measurementTime, Optional<I
70		-	Optional<Integer> forks, Optional<Integer> warmupForks,
	78	+	Optional<Integer> forks, Optional<Integer> minForks, Optional<Integer> warmupForks, Opti
	79	+	ReconfigureMode reconfigureMode, Optional<Double> reconfigureCovThreshold, Optional<Doub
71	80		Optional<String> jvm, Optional<Collection<String>> jvmArgs, Optional<Collection<String>>
72	81		Optional<Map<String, String[]>> params, Optional<TimeUnit> tu, Optional<Integer> opsPerI
73	82		Optional<TimeValue> timeout) {
79	88		this.threads = threads;
80	89		this.threadGroupLabels = threadGroupLabels;

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

	151	+	<code>this.params</code>	<code>= reader.nextOptionalParamCollection();</code>
	152	+	<code>this.tu</code>	<code>= reader.nextOptionalTimeUnit();</code>
	153	+	<code>this.opsPerInvocation</code>	<code>= reader.nextOptionalInt();</code>
	154	+	<code>this.timeout</code>	<code>= reader.nextOptionalTimeValue();</code>
132	155		<code>}</code>	
133	156			
134	157		<code>public String toLine() {</code>	
142	165		<code>writer.putIntArray(threadGroups);</code>	
143	166		<code>writer.putOptionalStringCollection(threadGroupLabels);</code>	
144	167		<code>writer.putOptionalInt(warmupIterations);</code>	
	168	+	<code>writer.putOptionalInt(minWarmupIterations);</code>	
145	169		<code>writer.putOptionalTimeValue(warmupTime);</code>	
146	170		<code>writer.putOptionalInt(warmupBatchSize);</code>	
147	171		<code>writer.putOptionalInt(measurementIterations);</code>	
148	172		<code>writer.putOptionalTimeValue(measurementTime);</code>	
149	173		<code>writer.putOptionalInt(measurementBatchSize);</code>	
150	174		<code>writer.putOptionalInt(forks);</code>	
	175	+	<code>writer.putOptionalInt(minForks);</code>	
151	176		<code>writer.putOptionalInt(warmupForks);</code>	
	177	+	<code>writer.putOptionalInt(minWarmupForks);</code>	
	178	+	<code>writer.putString(reconfigureMode.toString());</code>	
	179	+	<code>writer.putOptionalDouble(reconfigureCovThreshold);</code>	
	180	+	<code>writer.putOptionalDouble(reconfigureCiThreshold);</code>	
	181	+	<code>writer.putOptionalDouble(reconfigureKldThreshold);</code>	
152	182		<code>writer.putOptionalString(jvm);</code>	
153	183		<code>writer.putOptionalStringCollection(jvmArgs);</code>	
154	184		<code>writer.putOptionalStringCollection(jvmArgsPrepend);</code>	
164	194		<code>public BenchmarkListEntry cloneWith(Mode mode) {</code>	
165	195		<code>return new BenchmarkListEntry(userClassQName, generatedClassQName, method, mode,</code>	
166	196		<code>threads, threadGroups, threadGroupLabels,</code>	
167		-	<code>warmupIterations, warmupTime, warmupBatchSize,</code>	
	197	+	<code><u>minWarmupIterations</u>, warmupIterations, warmupTime, warmupBatchSize,</code>	
168	198		<code>measurementIterations, measurementTime, measurementBatchSize,</code>	
169		-	<code>forks, warmupForks,</code>	
	199	+	<code>forks, minForks, warmupForks, minWarmupForks,</code>	
	200	+	<code>reconfigureMode, reconfigureCovThreshold, reconfigureCiThreshold, reconfigureKldThreshold,</code>	
170	201		<code>jvm, jvmArgs, jvmArgsPrepend, jvmArgsAppend,</code>	
171	202		<code>params, tu, opsPerInvocation,</code>	
172	203		<code>timeout);</code>	
175	206		<code>public BenchmarkListEntry cloneWith(WorkloadParams p) {</code>	
176	207		<code>BenchmarkListEntry br = new BenchmarkListEntry(userClassQName, generatedClassQName, method, mode,</code>	
177	208		<code>threads, threadGroups, threadGroupLabels,</code>	
178		-	<code>warmupIterations, warmupTime, warmupBatchSize,</code>	
	209	+	<code><u>minWarmupIterations</u>, warmupIterations, warmupTime, warmupBatchSize,</code>	
179	210		<code>measurementIterations, measurementTime, measurementBatchSize,</code>	
180		-	<code>forks, warmupForks,</code>	
	211	+	<code>forks, minForks, warmupForks, minWarmupForks,</code>	
	212	+	<code>reconfigureMode, reconfigureCovThreshold, reconfigureCiThreshold, reconfigureKldThreshold,</code>	
181	213		<code>jvm, jvmArgs, jvmArgsPrepend, jvmArgsAppend,</code>	
182	214		<code>params, tu, opsPerInvocation,</code>	
183	215		<code>timeout);</code>	
269	301		<code>return warmupIterations;</code>	
270	302		<code>}</code>	
271	303			
	304	+	<code>public Optional&lt;Integer&gt; getMinWarmupIterations() {</code>	
	305	+	<code>return minWarmupIterations;</code>	
	306	+	<code>}</code>	
	307	+		
272	308		<code>public Optional&lt;Integer&gt; getWarmupBatchSize() {</code>	
273	309		<code>return warmupBatchSize;</code>	
274	310		<code>}</code>	
289	325		<code>return forks;</code>	
290	326		<code>}</code>	
291	327			
	328	+	<code>public Optional&lt;Integer&gt; getMinForks() {</code>	
	329	+	<code>return minForks;</code>	
	330	+	<code>}</code>	
	331	+		
292	332		<code>public Optional&lt;Integer&gt; getWarmupForks() {</code>	
293	333		<code>return warmupForks;</code>	

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.

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, <u>boolean atLeastOneWarning</u> );
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

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	+	/**

	346	+	* p value of kullback leibler divergence as variability threshold
	347	+	* @param value threshold
	348	+	* @return builder
	349	+	* @see org.openjdk.jmh.annotations.Reconfigure
	350	+	* @see org.openjdk.jmh.runner.Defaults#RECONFIGURE_KLD_THRESHOLD
	351	+	*/
	352	+	ChainedOptionsBuilder reconfigureKldThreshold(double value);
290	353		/**
291	354		* Forked JVM to use.
292	355		*

▼ 89 jmh-core/src/main/java/org/openjdk/jmh/runner/options/CommandLineOptions.java			
26	26		
27	27		import joptsimple.*;
28	28		import org.openjdk.jmh.annotations.Mode;
	29	+	import org.openjdk.jmh.annotations.ReconfigureMode;
29	30		import org.openjdk.jmh.profile.ProfilerFactory;
30	31		import org.openjdk.jmh.results.format.ResultFormatType;
31	32		import org.openjdk.jmh.runner.Defaults;
49	50		private final Optional<TimeValue> runTime;
50	51		private final Optional<Integer> batchSize;
51	52		private final Optional<Integer> warmupIterations;
	53	+	private final Optional<Integer> minWarmupIterations;
52	54		private final Optional<TimeValue> warmupTime;
53	55		private final Optional<Integer> warmupBatchSize;
54	56		private final List<Mode> benchMode = new ArrayList<>();
63	65		private final Optional<Integer> opsPerInvocation;
64	66		private final List<String> regexps = new ArrayList<>();
65	67		private final Optional<Integer> fork;
	68	+	private final Optional<Integer> minFork;
66	69		private final Optional<Integer> warmupFork;
	70	+	private final Optional<Integer> minWarmupFork;
	71	+	private final Optional<ReconfigureMode> reconfigureMode;
	72	+	private final Optional<Double> reconfigureCovThreshold;
	73	+	private final Optional<Double> reconfigureCiThreshold;
	74	+	private final Optional<Double> reconfigureKldThreshold;
67	75		private final Optional<String> output;
68	76		private final Optional<String> result;
69	77		private final Optional<ResultFormatType> resultFormat;
114	122		Defaults.WARMUP_ITERATIONS + " for all other modes)")
115	123		.withRequiredArg().withValuesConvertedBy(IntegerValueConverter.NON_NEGATIVE).describedAs("int");
116	124		
	125	+	OptionSpec<Integer> optMinWarmupCount = parser.accepts("mwi", "Minimum number of warmup iterations to do. Warm
	126	+	iterations are not counted towards the benchmark score. " +
	127	+	"(default: " + Defaults.MIN_WARMUP_ITERATIONS + ")")
	128	+	.withRequiredArg().withValuesConvertedBy(IntegerValueConverter.NON_NEGATIVE).describedAs("int");
	129	+	
117	130		OptionSpec<Integer> optWarmupBatchSize = parser.accepts("wbs", "Warmup batch size: number of benchmark " +
118	131		method calls per operation. Some benchmark modes may ignore this setting. " +
119	132		"(default: " + Defaults.WARMUP_BATCHSIZE + ")")
171	184		"(default: " + Defaults.MEASUREMENT_FORKS + ")")
172	185		.withRequiredArg().withValuesConvertedBy(IntegerValueConverter.NON_NEGATIVE).describedAs("int");
173	186		
	187	+	OptionSpec<Integer> optMinForks = parser.accepts("mf", "How many times to fork a single benchmark at least." +
	188	+	"(default: " + Defaults.MIN_MEASUREMENT_FORKS + ")")
	189	+	.withRequiredArg().withValuesConvertedBy(IntegerValueConverter.NON_NEGATIVE).describedAs("int");
	190	+	
174	191		OptionSpec<Integer> optWarmupForks = parser.accepts("wf", "How many warmup forks to make for a single benchmar
175	192		All iterations within the warmup fork are not counted towards the benchmark score. Use 0 to disable "
176	193		warmup forks. " +
177	194		"(default: " + Defaults.WARMUP_FORKS + ")")
178	195		.withRequiredArg().withValuesConvertedBy(IntegerValueConverter.NON_NEGATIVE).describedAs("int");
179	196		
	197	+	OptionSpec<Integer> optMinWarmupForks = parser.accepts("mwf", "How many warmup forks to make for a single benc
	198	+	"(default: " + Defaults.MIN_WARMUP_FORKS + ")")
	199	+	.withRequiredArg().withValuesConvertedBy(IntegerValueConverter.NON_NEGATIVE).describedAs("int");
	200	+	
	201	+	OptionSpec<String> optReconfigureMode = parser.accepts("rm", "Reconfigure mode. Available modes are: " + Recon
	202	+	"(default: " + Defaults.RECONFIGURE_MODE + ")")

```
203 +         .withRequiredArg().ofType(String.class).withValuesSeparatedBy(',').describedAs("mode");
204 +
205 +         OptionSpec<Double> optReconfigureCovThreshold = parser.accepts("rcov", "coefficient of variation variability t
206 +         "(default: " + Defaults.RECONFIGURE_COV_THRESHOLD + ")")
207 +         .withRequiredArg().withValuesConvertedBy(DoubleValueConverter.NON_NEGATIVE).describedAs("double");
208 +
209 +         OptionSpec<Double> optReconfigureCiThreshold = parser.accepts("rci", "confidence interval variability threshol
210 +         "(default: " + Defaults.RECONFIGURE_CI_THRESHOLD + ")")
211 +         .withRequiredArg().withValuesConvertedBy(DoubleValueConverter.NON_NEGATIVE).describedAs("double");
212 +
213 +         OptionSpec<Double> optReconfigureKldThreshold = parser.accepts("rkld", "p value of kullback leibler divergence
214 +         "(default: " + Defaults.RECONFIGURE_KLD_THRESHOLD + ")")
215 +         .withRequiredArg().withValuesConvertedBy(DoubleValueConverter.PROBABILITY).describedAs("double");
216 +
180 217         OptionSpec<String> optOutput = parser.accepts("o", "Redirect human-readable output to a given file.")
181 218         .withRequiredArg().ofType(String.class).describedAs("filename");
182 219
319 356         batchSize = toOptional(optMeasureBatchSize, set);
320 357         runTime = toOptional(optMeasureTime, set);
321 358         warmupIterations = toOptional(optWarmupCount, set);
322 359 +         minWarmupIterations = toOptional(optMinWarmupCount, set);
323 360         warmupBatchSize = toOptional(optWarmupBatchSize, set);
324 361         warmupTime = toOptional(optWarmupTime, set);
327 362         timeout = toOptional(optTimeoutTime, set);
328 365         gcEachIteration = toOptional(optGC, set);
329 366         failOnError = toOptional(optFOE, set);
330 367         fork = toOptional(optForks, set);
331 368 +         minFork = toOptional(optMinForks, set);
332 369         warmupFork = toOptional(optWarmupForks, set);
333 370 +         minWarmupFork = toOptional(optMinWarmupForks, set);
334 371         output = toOptional(optOutput, set);
335 372         result = toOptional(optOutputResults, set);
336 373
337 374 +         if (set.has(optReconfigureMode)) {
338 375 +             try {
339 376 +                 reconfigureMode = Optional.of(ReconfigureMode.deepValueOf(optReconfigureMode.value(set)));
340 377 +             } catch (IllegalArgumentException iae) {
341 378 +                 throw new CommandLineOptionException(iae.getMessage(), iae);
342 379 +             }
343 380 +         } else {
344 381 +             reconfigureMode = Optional.none();
345 382 +         }
346 383 +
347 384 +         reconfigureCovThreshold = toOptional(optReconfigureCovThreshold, set);
348 385 +         reconfigureCiThreshold = toOptional(optReconfigureCiThreshold, set);
349 386 +         reconfigureKldThreshold = toOptional(optReconfigureKldThreshold, set);
350 387 +
351 388         if (set.has(optBenchmarkMode)) {
352 389             try {
353 390                 List<Mode> modes = new ArrayList<>();
354 391
548 602         return fork;
549 603     }
550 604
551 605 +     @Override
552 606 +     public Optional<Integer> getMinForkCount() {
553 607 +         return minFork;
554 608 +     }
555 609 +
556 610     @Override
557 611     public Optional<Integer> getWarmupForkCount() {
558 612         return warmupFork;
559 613     }
560 614
561 615 +     @Override
562 616 +     public Optional<Integer> getMinWarmupForkCount() {
563 617 +         return minWarmupFork;
564 618 +     }
565 619 +
566 620     @Override
567 621     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");

```
30 +     }
31 +
32 +     if (newValue < minValue || newValue > maxValue) {
33 +         String message = "The given value " + value + " should be greater or equal than " + minValue + " and less
34 +         throw new ValueConversionException(message);
35 +     }
36 +     return newValue;
37 + }
38 +
39 + @Override
40 + public Class<Double> valueType() {
41 +     return TO_DOUBLE_CONVERTER.valueType();
42 + }
43 +
44 + @Override
45 + public String valuePattern() {
46 +     return "double";
47 + }
48 + }
```

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

```
25 25 package org.openjdk.jmh.runner.options;
26 26
27 27 import org.openjdk.jmh.annotations.Mode;
28 28 + import org.openjdk.jmh.annotations.ReconfigureMode;
29 29 import org.openjdk.jmh.results.format.ResultFormatType;
30 30 import org.openjdk.jmh.util.Optional;
31 31
118 119     */
119 120     Optional<Integer> getWarmupIterations();
120 121
122 122 + /**
123 123 +     * Minimum number of warmup iterations
124 124 +     * @return minimum number of warmup iterations
125 125 +     * @see org.openjdk.jmh.annotations.Warmup
126 126 +     */
127 127 + Optional<Integer> getMinWarmupIterations();
128 128 +
129 129 /**
130 130     * The duration for warmup iterations
131 131     * @return duration
194 202     */
195 203     Optional<Integer> getForkCount();
196 204
205 205 + /**
206 206 +     * Minimum fork count
207 207 +     * @return minimum fork count
208 208 +     * @see org.openjdk.jmh.annotations.Fork
209 209 +     */
210 210 + Optional<Integer> getMinForkCount();
211 211 +
197 212 /**
198 213     * Number of initial forks to ignore the results for
199 214     * @return initial fork count; 0, to disable
200 215     * @see org.openjdk.jmh.annotations.Fork
201 216     */
202 217     Optional<Integer> getWarmupForkCount();
203 218
219 219 + /**
220 220 +     * Minimum number of initial forks to ignore the results for
221 221 +     * @return minimum initial fork count
222 222 +     * @see org.openjdk.jmh.annotations.Fork
223 223 +     */
224 224 + Optional<Integer> getMinWarmupForkCount();
225 225 +
226 226 + /**
227 227 +     * reconfigure mode
228 228 +     * @return reconfigure mode
229 229 +     * @see org.openjdk.jmh.annotations.ReconfigureMode
```

	230	+     */
	231	+     Optional<ReconfigureMode> getReconfigureMode();
	232	+
	233	+     /**
	234	+     * coefficient of variation variability threshold
	235	+     * @return threshold
	236	+     * @see org.openjdk.jmh.annotations.Reconfigure
	237	+     */
	238	+     Optional<Double> getReconfigureCovThreshold();
	239	+
	240	+     /**
	241	+     * confidence interval variability threshold
	242	+     * @return threshold
	243	+     * @see org.openjdk.jmh.annotations.Reconfigure
	244	+     */
	245	+     Optional<Double> getReconfigureCiThreshold();
	246	+
	247	+     /**
	248	+     * p value of kullback leibler divergence as variability threshold
	249	+     * @return threshold
	250	+     * @see org.openjdk.jmh.annotations.Reconfigure
	251	+     */
	252	+     Optional<Double> getReconfigureKldThreshold();
	253	+
204	254	/**
205	255	* JVM executable to use for forks
206	256	* @return path to JVM executable

▼ 162 ■■■■ jmh-core/src/main/java/org/openjdk/jmh/runner/options/OptionsBuilder.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.annotations.Threads;
29	30	import org.openjdk.jmh.profile.Profiler;
30	31	import org.openjdk.jmh.results.format.ResultFormatType;
70	71	throw new IllegalArgumentException(message);
71	72	}
72	73	
	74	+ private static void checkGreaterOrEqual(double value, double minValue, String s) {
	75	+     if (value >= minValue) {
	76	+         return;
	77	+     }
	78	+     String message = s + " (" + value + ") should be greater or equal than " + minValue;
	79	+     throw new IllegalArgumentException(message);
	80	+     }
	81	+
	82	+ private static void checkLessOrEqual(double value, double minValue, String s) {
	83	+     if (value <= minValue) {
	84	+         return;
	85	+     }
	86	+     String message = s + " (" + value + ") should be less or equal than " + minValue;
	87	+     throw new IllegalArgumentException(message);
	88	+     }
	89	+
73	90	// -----
74	91	
75	92	private final List<String> regexps = new ArrayList<>();
354	371	
355	372	// -----
356	373	
	374	+ private Optional<Integer> minWarmupIterations = Optional.none();
	375	+
	376	+ @Override
	377	+ public ChainedOptionsBuilder minWarmupIterations(int value) {
	378	+     checkGreaterOrEqual(value, 0, "Minimum warmup iterations");
	379	+     this.minWarmupIterations = Optional.of(value);
	380	+     return this;
	381	+     }



	382	+	
	383	+	@Override
	384	+	public Optional<Integer> getMinWarmupIterations() {
	385	+	if (otherOptions != null) {
	386	+	return minWarmupIterations.orAnother(otherOptions.getMinWarmupIterations());
	387	+	} else {
	388	+	return minWarmupIterations;
	389	+	}
	390	+	}
	391	+	
	392	+	// -----
	393	+	
357	394		private Optional<Integer> warmupBatchSize = Optional.none();
358	395		
359	396		@Override
489	526		}
490	527		}
491	528		
492		-	
493	529		// -----
494	530		
495	531		private final EnumSet<Mode> benchModes = EnumSet.noneOf(Mode.class);
570	606		
571	607		// -----
572	608		
	609	+	private Optional<Integer> minForks = Optional.none();
	610	+	
	611	+	@Override
	612	+	public ChainedOptionsBuilder minForks(int value) {
	613	+	checkGreaterOrEqual(value, 0, "Minimum forks");
	614	+	this.minForks = Optional.of(value);
	615	+	return this;
	616	+	}
	617	+	
	618	+	@Override
	619	+	public Optional<Integer> getMinForkCount() {
	620	+	if (otherOptions != null) {
	621	+	return minForks.orAnother(otherOptions.getMinForkCount());
	622	+	} else {
	623	+	return minForks;
	624	+	}
	625	+	}
	626	+	
	627	+	// -----
	628	+	
573	629		private Optional<Integer> warmupForks = Optional.none();
574	630		
575	631		@Override
590	646		
591	647		// -----
592	648		
	649	+	private Optional<Integer> minWarmupForks = Optional.none();
	650	+	
	651	+	@Override
	652	+	public ChainedOptionsBuilder minWarmupForks(int value) {
	653	+	checkGreaterOrEqual(value, 0, "Minimum warmup forks");
	654	+	this.minWarmupForks = Optional.of(value);
	655	+	return this;
	656	+	}
	657	+	
	658	+	@Override
	659	+	public Optional<Integer> getMinWarmupForkCount() {
	660	+	if (otherOptions != null) {
	661	+	return minWarmupForks.orAnother(otherOptions.getMinWarmupForkCount());
	662	+	} else {
	663	+	return minWarmupForks;
	664	+	}
	665	+	}
	666	+	
	667	+	// -----



```
668 +
669 +     private Optional<ReconfigureMode> reconfigureMode = Optional.none();
670 +
671 +     @Override
672 +     public ChainedOptionsBuilder reconfigureMode(ReconfigureMode value) {
673 +         if(value.equals(ReconfigureMode.NONE)){
674 +             throw new IllegalArgumentException("None is as reconfigure mode not allowed");
675 +         }
676 +         this.reconfigureMode = Optional.of(value);
677 +         return this;
678 +     }
679 +
680 +     @Override
681 +     public Optional<ReconfigureMode> getReconfigureMode() {
682 +         if (otherOptions != null) {
683 +             return reconfigureMode.orAnother(otherOptions.getReconfigureMode());
684 +         } else {
685 +             return reconfigureMode;
686 +         }
687 +     }
688 +
689 +     // -----
690 +
691 +     private Optional<Double> reconfigureCovThreshold = Optional.none();
692 +
693 +     @Override
694 +     public ChainedOptionsBuilder reconfigureCovThreshold(double value) {
695 +         checkGreaterOrEqual(value, 0, "reconfigure cov threshold");
696 +         checkLessOrEqual(value, 1, "reconfigure cov threshold");
697 +         this.reconfigureCovThreshold = Optional.of(value);
698 +         return this;
699 +     }
700 +
701 +     @Override
702 +     public Optional<Double> getReconfigureCovThreshold() {
703 +         if (otherOptions != null) {
704 +             return reconfigureCovThreshold.orAnother(otherOptions.getReconfigureCovThreshold());
705 +         } else {
706 +             return reconfigureCovThreshold;
707 +         }
708 +     }
709 +
710 +     // -----
711 +
712 +     private Optional<Double> reconfigureCiThreshold = Optional.none();
713 +
714 +     @Override
715 +     public ChainedOptionsBuilder reconfigureCiThreshold(double value) {
716 +         checkGreaterOrEqual(value, 0, "reconfigure ci threshold");
717 +         this.reconfigureCiThreshold = Optional.of(value);
718 +         return this;
719 +     }
720 +
721 +     @Override
722 +     public Optional<Double> getReconfigureCiThreshold() {
723 +         if (otherOptions != null) {
724 +             return reconfigureCiThreshold.orAnother(otherOptions.getReconfigureCiThreshold());
725 +         } else {
726 +             return reconfigureCiThreshold;
727 +         }
728 +     }
729 +
730 +     // -----
731 +
732 +     private Optional<Double> reconfigureKldThreshold = Optional.none();
733 +
734 +     @Override
735 +     public ChainedOptionsBuilder reconfigureKldThreshold(double value) {
736 +         checkGreaterOrEqual(value, 0, "reconfigure kld threshold");
737 +         checkLessOrEqual(value, 1, "reconfigure kld threshold");
```

	738	+	this.reconfigureKldThreshold = Optional.of(value);
	739	+	return this;
	740	+	}
	741	+	
	742	+	@Override
	743	+	public Optional<Double> getReconfigureKldThreshold() {
	744	+	if (otherOptions != null) {
	745	+	return reconfigureKldThreshold.orAnother(otherOptions.getReconfigureKldThreshold());
	746	+	} else {
	747	+	return reconfigureKldThreshold;
	748	+	}
	749	+	}
	750	+	
	751	+	// -----
	752	+	
593	753		private Optional<String> jvmBinary = Optional.none();
594	754		
595	755		@Override

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

31	31		public static final char TAG_EMPTY_OPTIONAL = 'E';
32	32		public static final char TAG_STRING = 'S';
33	33		public static final char TAG_INT = 'I';
	34	+	public static final char TAG_DOUBLE = 'D';
34	35		public static final char TAG_TIMEVALUE = 'T';
35	36		public static final char TAG_STRING_COLLECTION = 'L';
36	37		public static final char TAG_INT_ARRAY = 'A';

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

100	100		}
101	101		}
102	102		
	103	+	public Optional<Double> nextOptionalDouble() {
	104	+	char tag = readChar();
	105	+	if (tag == Constants.TAG_EMPTY_OPTIONAL) {
	106	+	return Optional.none();
	107	+	} else if (tag == TAG_DOUBLE) {
	108	+	return Optional.of(Double.valueOf(readString()));
	109	+	} else {
	110	+	throw error("unexpected tag = " + tag);
	111	+	}
	112	+	}
	113	+	
103	114		public Optional<String> nextOptionalString() {
104	115		char tag = readChar();
105	116		if (tag == Constants.TAG_EMPTY_OPTIONAL) {

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

72	72		}
73	73		}
74	74		
	75	+	public void putOptionalDouble(Optional<Double> opt) {
	76	+	if (!opt.hasValue()) {
	77	+	appendTag(TAG_EMPTY_OPTIONAL);
	78	+	} else {
	79	+	appendTag(TAG_DOUBLE);
	80	+	appendWithLen(String.valueOf(opt.get()));
	81	+	}
	82	+	}
	83	+	
75	84		public void putOptionalString(Optional<String> opt) {
76	85		if (!opt.hasValue()) {
77	86		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), <u>null, null</u> );
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), <u>null, null</u> );
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), <u>null, null</u> );
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), <u>null, null</u> );
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, <u>TimeUnit.SECONDS</u> , 1,
59		-	Utils.getCurrentJvm(), Collections.<String>emptyList(),
	55	+	1, 1, <u>1, 1,</u>
	56	+	new IterationParams(IterationType.WARMUP, 1, <u>1,</u> TimeValue.seconds(1), 1),
	57	+	new IterationParams(IterationType.MEASUREMENT, 1, <u>1,</u> TimeValue.seconds(1), 1),
	58	+	Mode.Throughput, null, <u>1,1,1,</u> null,
	59	+	<u>TimeUnit.SECONDS</u> , 1, <u>Utils.getCurrentJvm()</u> , 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, <u>1,</u> 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	+
	77	int threads = r.nextInt(1000);
	78	+
	79	int threadGroups = r.nextInt(1000);
	80	+
	81	int forks = r.nextInt(1000);
	82	+
	83	int warmupForks = r.nextInt(1000);
	84	+
	85	+
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, <u>TimeUnit.SECONDS</u> , 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, <u>1,1,1</u> , null,
	68	<u>TimeUnit.SECONDS</u> , 1, <u>Utils.getCurrentJvm()</u> , 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, <u>TimeUnit.SECONDS, 1,</u>
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	+	<u>TimeUnit.SECONDS, 1, Utils.getCurrentJvm(),</u>
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(),

53	54		Optional.<Integer>none(),
	55	+	Optional.<Integer>none(),
54	56		Optional.<TimeValue>none(),
55	57		Optional.<Integer>none(),
56	58		Optional.<Integer>none(),
57	59		Optional.<TimeValue>none(),
58	60		Optional.<Integer>none(),
59	61		Optional.<Integer>none(),
60	62		Optional.<Integer>none(),
	63	+	Optional.<Integer>none(),
	64	+	Optional.<Integer>none(),
	65	+	ReconfigureMode.CI,
	66	+	Optional.<Double>none(),
	67	+	Optional.<Double>none(),
	68	+	Optional.<Double>none(),
61	69		Optional.<String>none(),
62	70		Optional.<Collection<String>>none(),
63	71		Optional.<Collection<String>>none(),

▼ 5 jmh-core/src/test/resources/org/openjdk/jmh/results/format/output-golden.json			
5	5		"mode" : "thrpt",
6	6		"threads" : 80,
7	7		"forks" : 828,
	8	+	"warmupForks" : 55,
8	9		"jvm" : "javadummy",
9	10		"jvmArgs" : [
10	11		],
228	229		"mode" : "thrpt",
229	230		"threads" : 900,
230	231		"forks" : 364,
	232	+	"warmupForks" : 275,
231	233		"jvm" : "javadummy",
232	234		"jvmArgs" : [
233	235		],
334	336		"mode" : "thrpt",
335	337		"threads" : 466,
336	338		"forks" : 677,
	339	+	"warmupForks" : 750,
337	340		"jvm" : "javadummy",
338	341		"jvmArgs" : [
339	342		],
502	505		"mode" : "thrpt",
503	506		"threads" : 968,
504	507		"forks" : 581,
	508	+	"warmupForks" : 856,
505	509		"jvm" : "javadummy",
506	510		"jvmArgs" : [
507	511		],
702	706		"mode" : "thrpt",
703	707		"threads" : 739,
704	708		"forks" : 670,
	709	+	"warmupForks" : 338,
705	710		"jvm" : "javadummy",
706	711		"jvmArgs" : [
707	712		],

▼ 2 jmh-generator-annprocess/pom.xml			
31	31		<parent>
32	32		<groupId>org.openjdk.jmh</groupId>
33	33		<artifactId>jmh-parent</artifactId>
34		-	<version>1.21</version>
	34	+	<version>1.21-Reconfigure</version>
35	35		</parent>
36	36		
37	37		<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- <u>Reconfigure</u> </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- <u>Reconfigure</u> </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- <u>Reconfigure</u> </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- <u>Reconfigure</u> </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




```
26 + * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
27 + * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
28 + * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
29 + * THE POSSIBILITY OF SUCH DAMAGE.
30 + */
31 + package org.openjdk.jmh.samples;
32 +
33 + import org.openjdk.jmh.annotations.*;
34 + import org.openjdk.jmh.results.format.ResultFormatType;
35 + import org.openjdk.jmh.runner.Runner;
36 + import org.openjdk.jmh.runner.RunnerException;
37 + import org.openjdk.jmh.runner.options.Options;
38 + import org.openjdk.jmh.runner.options.OptionsBuilder;
39 +
40 + import java.util.concurrent.TimeUnit;
41 +
42 + public class JMHSample_00_Reconfigure {
43 +
44 +     @Benchmark
45 +     @Reconfigure(value = ReconfigureMode.DIVERGENCE, covThreshold = 0.07, ciThreshold = 0.08, kldThreshold = 0.09)
46 +     @Warmup(minIterations = 11, time = 1, timeUnit = TimeUnit.MICROSECONDS)
47 +     @Measurement(time = 1, timeUnit = TimeUnit.MICROSECONDS)
48 +     @BenchmarkMode(Mode.Reconfigure)
49 +     @Fork(minWarmups = 7, minValue = 9)
50 +     public void wellHelloThere() {
51 +         // this method was intentionally left blank.
52 +     }
53 +
54 +     public static void main(String[] args) throws RunnerException {
55 +         Options opt = new OptionsBuilder()
56 +             .include(JMHSample_00_Reconfigure.class.getSimpleName())
57 +             .result("D:\\out.json")
58 +             .resultFormat(ResultFormatType.JSON)
59 +             .build();
60 +
61 +         new Runner(opt).run();
62 +     }
63 +
64 + }
```

▼ 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>

124	-	<quiet>>true</quiet>
125	-	</configuration>
126	-	<executions>
127	-	<execution>
128	-	<id>attach-javadoc</id>
129	-	<phase>verify</phase>
130	-	<goals>
131	-	<goal>jar</goal>
132	-	</goals>
133	-	</execution>
134	-	</executions>
135	-	</plugin>
	120	+ <!--
	121	+ <!--
	122	+ <!--
	123	+ <!--
	124	+ <!--
	125	+ <!--
	126	+ <!--
	127	+ <!--
	128	+ <!--
	129	+ <!--
	130	+ <!--
	131	+ <!--
	132	+ <!--
	133	+ <!--
	134	+ <!--
	135	+ <!--
136	136	<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>--->
137	137	<!-- Add sources and javadoc to eclipse project files when available. -->
138	-	<plugin>
139	-	<groupId>org.apache.maven.plugins</groupId>
140	-	<artifactId>maven-eclipse-plugin</artifactId>
141	-	<configuration>
142	-	<downloadSources>>true</downloadSources>
143	-	<downloadJavadocs>>true</downloadJavadocs>
144	-	</configuration>
145	-	</plugin>
	138	+ <!--
	139	+ <!--
	140	+ <!--
	141	+ <!--
	142	+ <!--
	143	+ <!--
	144	+ <!--
	145	+ <!--
146	146	<plugin>--->
		<groupId>org.apache.maven.plugins</groupId>--->
		<artifactId>maven-eclipse-plugin</artifactId>--->
		<configuration>--->
		<downloadSources>>true</downloadSources>--->
		<downloadJavadocs>>true</downloadJavadocs>--->
		</configuration>--->
		</plugin>--->
147	147	<plugin>
148	148	<groupId>org.apache.maven.plugins</groupId>