JMH-HBASE-25913

Default currentTime()

This test simply invokes EnvironmentEdgeManager.currentTime.

```
BenchDefaultClock.testCurrentTime                            1644514           79.890 ± 1.845
BenchDefaultClock.testCurrentTime·p0.00                                          20.000
BenchDefaultClock.testCurrentTime·p0.50                                          58.000
BenchDefaultClock.testCurrentTime·p0.90                                          63.000
BenchDefaultClock.testCurrentTime·p0.95                                          67.000
BenchDefaultClock.testCurrentTime·p0.99                                          77.000
BenchDefaultClock.testCurrentTime·p0.999                                       1941.700
BenchDefaultClock.testCurrentTime·p0.9999                                     30240.000
BenchDefaultClock.testCurrentTime·p1.00                                      164096.000

      ns  percent  samples  top
  ----------  -------  -------  ---
 22890000000   52.51%        2289  mach_absolute_time
  6510000000   14.93%  651  __commpage_gettimeofday_internal
  4120000000   9.45%  412  BenchDefaultClock_testCurrentTime_jmhStub
  3680000000   8.44%  368  os::javaTimeMillis()
  3140000000   7.20%  314  [unknown]
   890000000   2.04%   89  org.apache.hadoop.hbase.util.BaseEnvironmentEdge$1.currentTime
   320000000   0.73%   32  os::javaTimeNanos()
   290000000   0.67%   29  __commpage_gettimeofday
   230000000   0.53%   23  org.apache.hadoop.hbase.util.EnvironmentEdgeManager.currentTime
   150000000   0.34%   15  org.apache.hadoop.hbase.util.EnvironmentEdgeManager.getDelegate
```

BenchIncrementAdvancingClock

This test invokes IncrementAdvancingClock's currentTime() method, which uses an atomic long to ensure we never return a time that jumps backwards. currentTime() and currentTimeAdvancing() operate on the same atomic for consistency.

```
  public long currentTime() {
        return lastTime.updateAndGet(x -> {
               long now = System.currentTimeMillis();
               if (now > x) {
                      return update(now);
               }
               return x;
        });
  }

  // Broken out to inlinable method for subclassing and instrumentation.
  protected long update(long now) {
        // System clock has moved ahead of our notion of current tick. Move us forward to match.
        // We do that just by returning the current time, which was passed in to us as 'n'.
        return now;
  }
```

```
BenchIncrementAdvancingClock.testCurrentTime                          1458044       81.112 ±  3.364
BenchIncrementAdvancingClock.testCurrentTime·p0.00                                  23.000
BenchIncrementAdvancingClock.testCurrentTime·p0.50                                  59.000
BenchIncrementAdvancingClock.testCurrentTime·p0.90                                  63.000
BenchIncrementAdvancingClock.testCurrentTime·p0.95                                  66.000
```

```
BenchIncrementAdvancingClock.testCurrentTime·p0.99                    85.000
BenchIncrementAdvancingClock.testCurrentTime·p0.999                 1301.550
BenchIncrementAdvancingClock.testCurrentTime·p0.9999               30393.024
BenchIncrementAdvancingClock.testCurrentTime·p1.00                901120.000
```

This test invokes IncrementAdvancingClock's currentTimeAdvancing() method, which uses an atomic long to ensure we always return a time that is at least one tick ahead of the last time we returned, even if the system clock has not advanced yet. currentTime() and currentTimeAdvancing() operate on the same atomic for consistency.

```
  public long currentTimeAdvancing() throws InterruptedException {
        return lastTime.updateAndGet(x -> {
               long now = System.currentTimeMillis();
               if (now > x) {
                      return update(now);
               }
               return advance(x);
        });
  }

  // Broken out to inlinable method for subclassing and instrumentation.
  protected long advance(long last) {
        // System clock hasn't moved forward. Increment our notion of current tick to keep
        // the time advancing.
        return last + 1;
  }

  // Broken out to inlinable method for subclassing and instrumentation.
  protected long update(long now) {
        // System clock has moved ahead of our notion of current tick. Move us forward to match.
        // We do that just by returning the current time, which was passed in to us as 'n'.
        return now;
  }
```

```
BenchIncrementAdvancingClock.testCurrentTimeAdvancing            1531621     99.374 ± 21.543
BenchIncrementAdvancingClock.testCurrentTimeAdvancing·p0.00                  16.000
BenchIncrementAdvancingClock.testCurrentTimeAdvancing·p0.50                  55.000
BenchIncrementAdvancingClock.testCurrentTimeAdvancing·p0.90                  59.000
BenchIncrementAdvancingClock.testCurrentTimeAdvancing·p0.95                  61.000
BenchIncrementAdvancingClock.testCurrentTimeAdvancing·p0.99                  77.000
BenchIncrementAdvancingClock.testCurrentTimeAdvancing·p0.999               1012.000
BenchIncrementAdvancingClock.testCurrentTimeAdvancing·p0.9999             29978.810
BenchIncrementAdvancingClock.testCurrentTimeAdvancing·p1.00              999360.000

testCurrentTimeAdvancing:

       ns  percent  samples  top
  ----------  -------  -------  ---
 19880000000   46.75%        1988  mach_absolute_time
  5470000000   12.86%  547  java.util.concurrent.atomic.AtomicLong.compareAndSet
  5360000000   12.61%  536  __commpage_gettimeofday_internal
  3440000000   8.09%  344  os::javaTimeMillis()
  3060000000   7.20%  306  BenchIncrementAdvancingClock_testCurrentTimeAdvancing_jmhStub
  2460000000   5.79%  246  [unknown]
  1060000000   2.49%  106  java.util.concurrent.atomic.AtomicLong.get
   350000000   0.82%   35  java.util.concurrent.atomic.AtomicLong.updateAndGet
   340000000   0.80%   34  org.apache.hadoop.hbase.util.IncrementAdvancingClock.currentTimeAdvancing
   330000000   0.78%   33  __commpage_gettimeofday
   310000000   0.73%   31  org.apache.hadoop.hbase.util.IncrementAdvancingClock.lambda$currentTimeAdvancing$1
   230000000   0.54%   23  os::javaTimeNanos()
    50000000   0.12%   5  __psynch_cvwait
```

BenchSpinAdvancingClock

This test invokes SpinAdvancingClock's currentTime() method, which uses an atomic long to ensure we never return a time that jumps backwards. currentTime() and currentTimeAdvancing() operate on the same atomic for consistency.

```
  public long currentTime() {
        return lastTime.updateAndGet(x -> {
                long now = System.currentTimeMillis();
                if (now > x) {
                        return update(now);
                }
                return x;
        });
  }

  // Broken out to inlinable method for subclassing for instrumentation.
  protected long update(long now) {
        return now;
  }
```

```
BenchSpinAdvancingClock.testCurrentTime                          1467450             74.391 ±   2.199
BenchSpinAdvancingClock.testCurrentTime·p0.00                                 26.000
BenchSpinAdvancingClock.testCurrentTime·p0.50                                             59.000
BenchSpinAdvancingClock.testCurrentTime·p0.90                                             64.000
BenchSpinAdvancingClock.testCurrentTime·p0.95                                             66.000
BenchSpinAdvancingClock.testCurrentTime·p0.99                                             85.000
BenchSpinAdvancingClock.testCurrentTime·p0.999                               199.000
BenchSpinAdvancingClock.testCurrentTime·p0.9999                            26472.157
BenchSpinAdvancingClock.testCurrentTime·p1.00                             396288.000
```

This test invokes SpinAdvancingClock's currentTimeAdvancing() method, which will not return until the system clock has advanced, spinning around currentTimeMillis. As expected the method requires at least the time for the clock to tick over on the development host, approximately 1 millisecond. currentTime() and currentTimeAdvancing() operate on the same atomic for consistency.

```
  public long currentTimeAdvancing() throws InterruptedException {
        long now;
        while (true) {
                now = System.currentTimeMillis();
                if (now > lastTime.get()) {
                        final long updateTime = now;
                        return lastTime.updateAndGet(x -> update(updateTime));
                }
                spin();
        }
  }

  // Broken out to inlinable method for subclassing for instrumentation.
  protected void spin() { }

  // Broken out to inlinable method for subclassing for instrumentation.
  protected long update(long now) {
        return now;
  }
```

```
BenchSpinAdvancingClock.testCurrentTimeAdvancing                        50022    999285.286 ± 159.648
BenchSpinAdvancingClock.testCurrentTimeAdvancing·p0.00                    299.000
BenchSpinAdvancingClock.testCurrentTimeAdvancing·p0.50                 999424.000
BenchSpinAdvancingClock.testCurrentTimeAdvancing·p0.90                 999424.000
```

```
BenchSpinAdvancingClock.testCurrentTimeAdvancing·p0.95                    999424.000
BenchSpinAdvancingClock.testCurrentTimeAdvancing·p0.99                   1011712.000
BenchSpinAdvancingClock.testCurrentTimeAdvancing·p0.999                  1021952.000
BenchSpinAdvancingClock.testCurrentTimeAdvancing·p0.9999                 1043434.803
BenchSpinAdvancingClock.testCurrentTimeAdvancing·p1.00                   1067008.000

testCurrentTimeAdvancing:

       ns  percent  samples  top
  ----------  -------  -------  ---
 26520000000   60.98%        2652  mach_absolute_time
  7310000000   16.81%  731  __commpage_gettimeofday_internal
  4480000000   10.30%  448  os::javaTimeMillis()
  3240000000   7.45%  324  [unknown]
   470000000   1.08%   47  org.apache.hadoop.hbase.util.SpinAdvancingClock$$Lambda$1.2123188209.get$Lambda
   440000000   1.01%   44  org.apache.hadoop.hbase.util.SpinAdvancingClock.currentTimeAdvancing
   370000000   0.85%   37  __commpage_gettimeofday
    60000000   0.14%         6  __psynch_cvwait
```

BenchYieldAdvancingClock

This test invokes YieldAdvancingClock's currentTime() method, which uses an atomic long to ensure we never return a time that jumps backwards. currentTime() and currentTimeAdvancing() operate on the same atomic for consistency.

```
  public class YieldAdvancingClock extends SpinAdvancingClock

BenchYieldAdvancingClock.testCurrentTime                        1568327             68.681 ±    1.350
BenchYieldAdvancingClock.testCurrentTime·p0.00                                      26.000
BenchYieldAdvancingClock.testCurrentTime·p0.50                                      56.000
BenchYieldAdvancingClock.testCurrentTime·p0.90                                      59.000
BenchYieldAdvancingClock.testCurrentTime·p0.95                                      59.000
BenchYieldAdvancingClock.testCurrentTime·p0.99                                      62.000
BenchYieldAdvancingClock.testCurrentTime·p0.999                                    130.000
BenchYieldAdvancingClock.testCurrentTime·p0.9999                                 25088.000
BenchYieldAdvancingClock.testCurrentTime·p1.00                                   72704.000
```

This test invokes YieldAdvancingClock's currentTimeAdvancing() method, which will not return until the system clock has advanced, spinning around currentTimeMillis with a 100ns sleep at each loop iteration. As expected the method requires at least the time for the clock to tick over on the development host, approximately 1 millisecond. There is additional overhead for the thread sleep. currentTime() and currentTimeAdvancing() operate on the same atomic for consistency.

```
  public class YieldAdvancingClock extends SpinAdvancingClock

  // Broken out to inlinable method for subclassing and instrumentation.
  @Override
  protected void spin() {
        try {
               Thread.sleep(0, 100);  // black magic to yield on all platforms
        } catch (InterruptedException e) {
               // Restore thread interrupt status
               Thread.currentThread().interrupt();
        }
  }

BenchYieldAdvancingClock.testCurrentTimeAdvancing                 38545    1295586.692 ± 1486.865
BenchYieldAdvancingClock.testCurrentTimeAdvancing·p0.00                             159.000
BenchYieldAdvancingClock.testCurrentTimeAdvancing·p0.50                         1318912.000
BenchYieldAdvancingClock.testCurrentTimeAdvancing·p0.90                         1331200.000
BenchYieldAdvancingClock.testCurrentTimeAdvancing·p0.95                         1345536.000
```

```
BenchYieldAdvancingClock.testCurrentTimeAdvancing·p0.99                  1372160.000
BenchYieldAdvancingClock.testCurrentTimeAdvancing·p0.999                 1402880.000
BenchYieldAdvancingClock.testCurrentTimeAdvancing·p0.9999                2603560.960
BenchYieldAdvancingClock.testCurrentTimeAdvancing·p1.00                  3158016.000

testCurrentTimeAdvancing:

       ns  percent  samples  top
  ----------  -------  -------  ---
   740000000   63.79%        74  __psynch_cvwait
    70000000   6.03%          7  __gettimeofday
    40000000   3.45%          4  java_lang_Thread::set_thread_status(oopDesc*,
java_lang_Thread::ThreadStatus)
    30000000   2.59%          3  java.lang.Thread.sleep
    30000000   2.59%          3  JavaThread::handle_special_suspend_equivalent_condition()
    20000000   1.72%          2  os::PlatformEvent::park(long)
    10000000   0.86%          1  os::sleep(Thread*, long, bool)
    10000000   0.86%          1  BenchYieldAdvancingClock_testCurrentTimeAdvancing_jmhStub
```