Spark Executor parallelism with Partition

By Mohit Kumar

Spark Internals:ControllingParallelism:ex-7

```
        .repartition(partition)
        .map(x -> fastTask(x))
        .map(x -> ConcurrentContext.getInstance().executeAsync(x))
        .mapPartitions(new FlatMapFunction<Iterator<CompletableFuture<Long>>, Long>() {
            @Override
            public Iterator<Long> call(Iterator<CompletableFuture<Long>> t) throws Exception {
                return ConcurrentContext.getInstance().awaitBatch(t).iterator();
            }
        })
        .foreach(t->System.out.println("Finishing:"+t));
}

static class ConcurrentContext {
    private static final ConcurrentContext INSTANCE = new ConcurrentContext();
    private final int BatchSize = 4;
    private final Executor executor = Executors.newFixedThreadPool(BatchSize, new ThreadFactory() {
        @Override
        public Thread newThread(Runnable r) {
            Thread t = new Thread(r);
            t.setDaemon(true);
            return t;
        }
    });
    public static ConcurrentContext getInstance() {
        return INSTANCE;
    }
    public CompletableFuture<Long> executeAsync(long x) {
        return CompletableFuture.supplyAsync(() -> slowTask(x),executor);
    }
    public List<Long> awaitBatch(Iterator<CompletableFuture<Long>> futureit) {
        Stream<CompletableFuture<Long>> stream2 = StreamSupport
                .stream(Spliterators.
                        spliteratorUnknownSize(futureit, Spliterator.CONCURRENT), true);
        List<Long> vals=stream2
                .parallel()
                .map(CompletableFuture::join)
                .collect(Collectors.toList());
        return vals;
    }
```

1. Programmatic executor
2. Future executed on executor
3. Enables stream construction from an iterator. IO.I transforms are built on this.
4. Parallel stream reduction within a locality.

Spark Internals:ControllingParallelism:ex-7

```
Executor task launch worker for task 2:78:FastTask:48
Executor task launch worker for task 2:78:FastTask:50
Thread-4:81:SlowTaskEnd:2
Thread-5:82:SlowTaskEnd:1
Thread-4:81:SlowTaskStart:6
Thread-7:84:SlowTaskEnd:3
Thread-6:83:SlowTaskEnd:4
Thread-7:84:SlowTaskStart:5
Thread-5:82:SlowTaskStart:8
Thread-6:83:SlowTaskStart:7
Thread-4:81:SlowTaskEnd:6
Thread-4:81:SlowTaskStart:9
Thread-7:84:SlowTaskEnd:5
Thread-5:82:SlowTaskEnd:8
Thread-6:83:SlowTaskEnd:7
Thread-5:82:SlowTaskStart:11
Thread-7:84:SlowTaskStart:10
Thread-6:83:SlowTaskStart:12
Thread-4:81:SlowTaskEnd:9
Thread-4:81:SlowTaskStart:13
Thread-7:84:SlowTaskEnd:10
Thread-6:83:SlowTaskEnd:12
Thread-5:82:SlowTaskEnd:11
Thread-6:83:SlowTaskStart:15
Thread-7:84:SlowTaskStart:14
Thread-5:82:SlowTaskStart:16
Thread-4:81:SlowTaskEnd:13
Thread-4:81:SlowTaskStart:17
```

Notice only 4 are working at a time

Unfortunately we are blocking for every batch, which in turn is causing scheduler delays and deserialization time.