Mesos Scheduling Mode on Spark

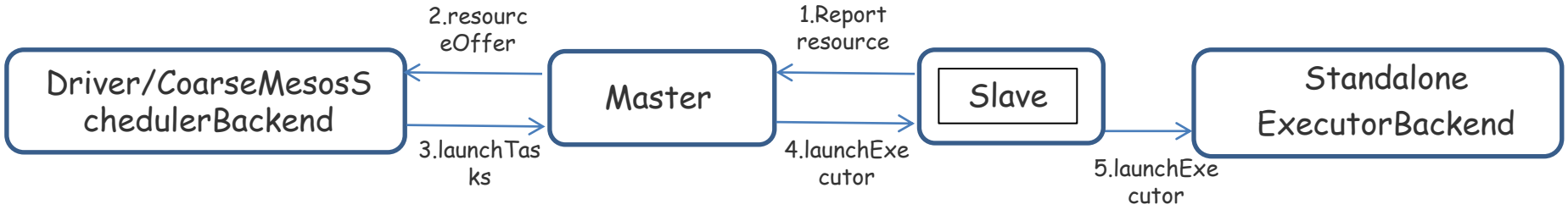
Andrew Xia

Scheduling Type

- Coarse-grained Mode
  - Hold resource until framework/application ended
  - launch only one long-running Spark task on each Mesos slave machine
- Fine-grained Mode
  - Share slave source with other frameworks/applications
  - Each spark task is one Mesos task, overhead in launching each task
  - inappropriate for low-latency applications

Coarse-grained Mode

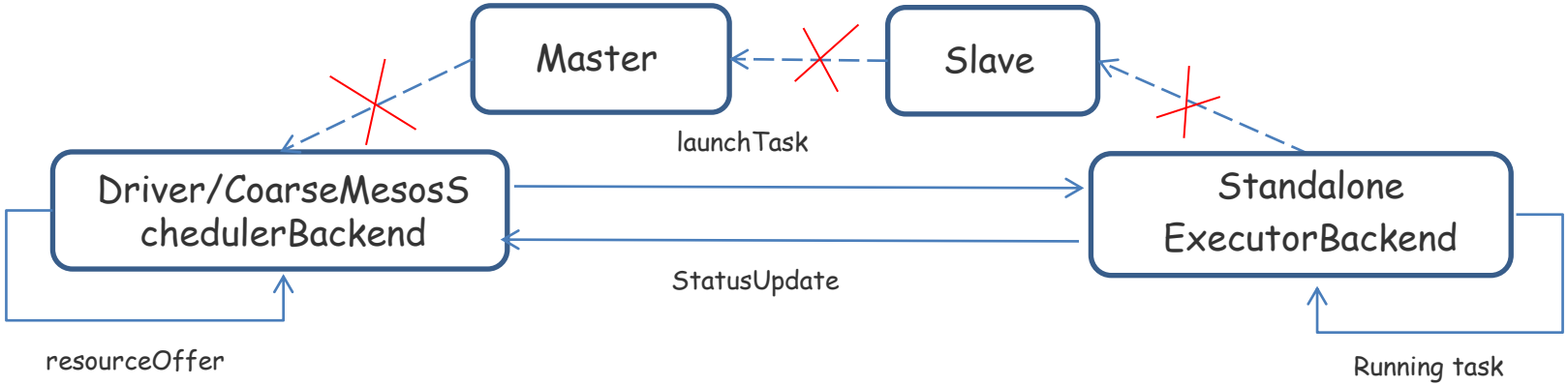
- Launch executor



  - Launch executor when first calling resourceOffer(about 2s)
  - Executor backend same as Standalone mode
  - Driver take into account killing executor backend when application ended

Coarse-grained Mode

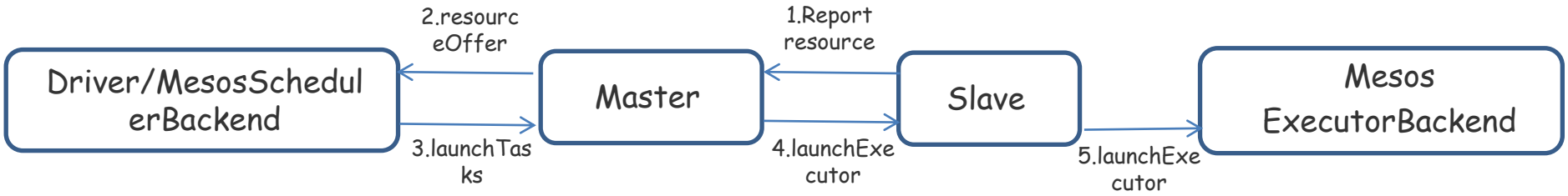
- Task Scheduling & resources alloc



  - Scheduling process is similar with standalone mode
  - Generally Executor will always hold resource even if no task run
    - Cpu: min(Spark.core.max , cores of slave)
    - Mem: SPARK_MEM < mem of slave
  - Executor do not share usage status of resource with master and slave, master always know that it have allocate all slave resource to executor and "***Master has no resource now!***"

Fine-grained Mode

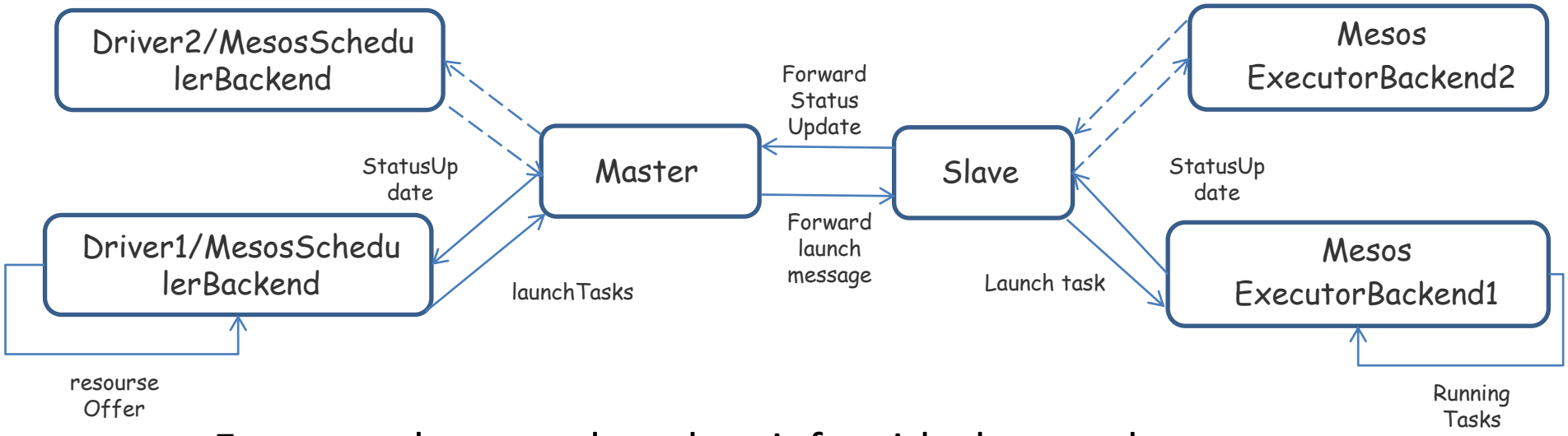
- Launch executor



  - Launch process is similar with Coarse-grained mode
  - Executor is extends with MesosExecutor, and different from StandaloneExecutorBackend

Fine-grained Mode

- Task scheduling & resource alloc



  - Executor shares task update info with slave and master
  - Executor does not have static resource, dynamically changed when tasks launch and end
  - Master manager all resource and allocate resource between driver1 and driver2 by FIFO or FAIR, "***Master knows everything now***"

Some compares

- Coarse-grained has lower latency than fine-grained mode
- Fine-grained allocate resource more efficiency than Coarse-grained mode
- ***How do we choose these two modes?***

Test environment

- Client/Mesos master/Mesos slave in three different machine
  - Cpu:24
  - Mem:48G
- Test application

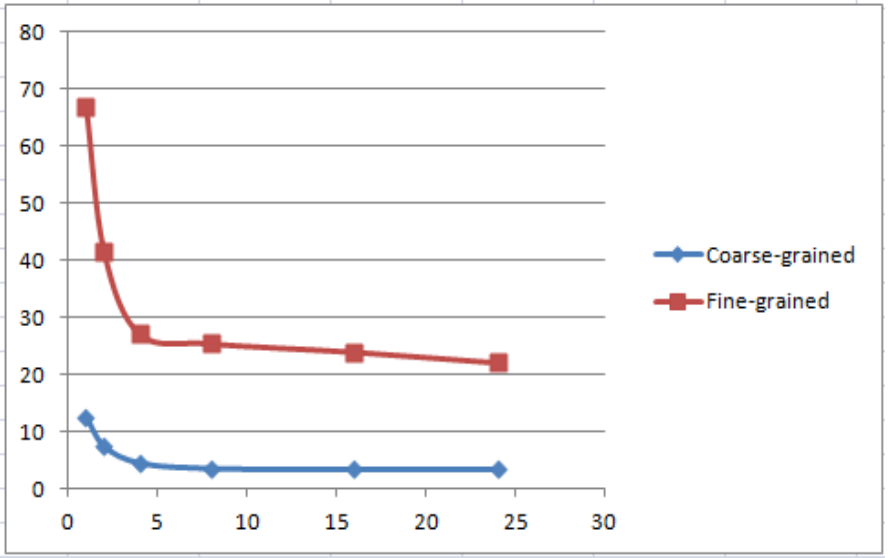
  sc = new sparkContext(…….)

  val count = sc.parallelize(1 to 1000, 1000).map {i => 1}.reduce(_ + _)

“App runtime” versus “cores”

- Set slave resource “core” to 1,2,8,16,24
- Set task number 1000
- Each task run time ≈3ms
- Run test application using Coarse and fine mode
- X is core and Y is application run time(s)(actual time)

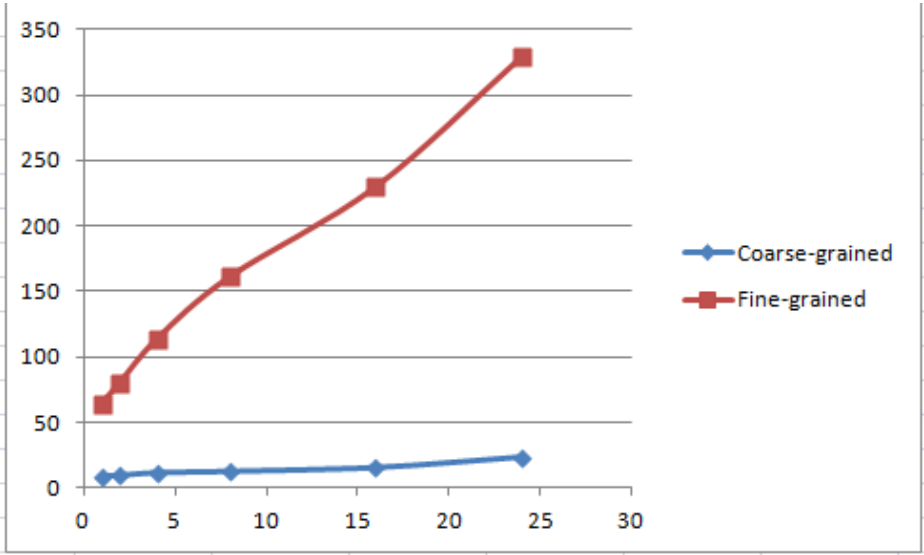
	Coarse-grained	Fine-grained
1	12.544	67.077
2	7.45	41.64
4	4.56	27.1
8	3.62	25.4
16	3.46	23.93
24	3.43	22.07



“Scheduling latency” versus “cores”

- Latency time
  - 1. time of driver to send “launch Task” message
  - 2. time of driver to receive “status Update” message
  - Latency time = “2” – “1”(scheduling time + scheduling waiting time + task running time)
  - X is cores and Y is latency time(ms)(actual time)

	Coarse-grained	Fine-grained
1	8.5	64
2	9	80
4	11	113
8	12	161
16	15	230
24	23	330



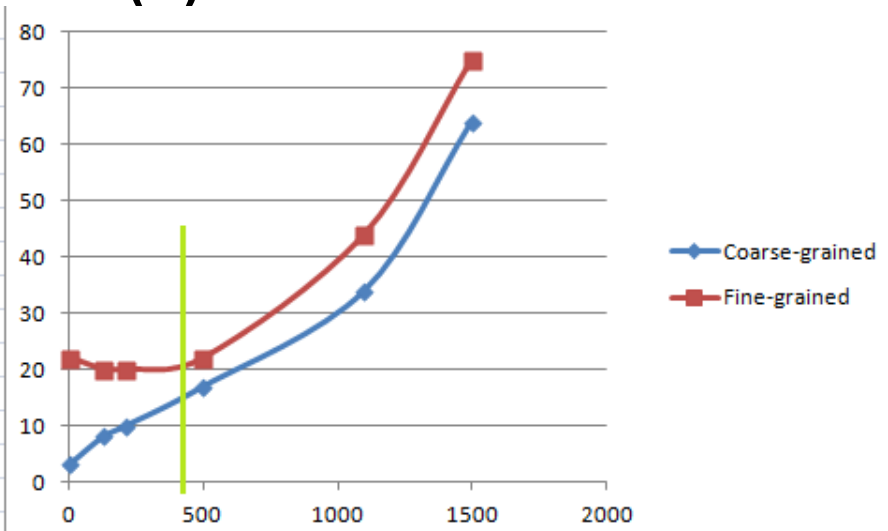
Analysis

- For core is 1
  - Fine Mode, latency = 64ms (scheduling time + task running Time), as now system has only 1 core, no any waiting latency
- For core is 2
  - Fine Mode, latency = 80ms (scheduling time + scheduling waiting time+ task runtime)
  - Fine mode, wait 1 core to status update need 80 -64 =16ms(scheduling waiting time)
- For core is n
  - We predicate latency time is 64+16*(cores -1)
  - We predicate runtime is (latency time*(task number/cores))
- Compared with actual time, the predicate time is somewhat accurate

“App runtime” versus “task runtime”

- Cores resource is 24
- Application contains 1000 tasks
- X is run time of each tasks(ms)
  - I construct logic to satisfy various task run time
- Y is run time of application(s)

	Coarse-grained	Fine-grained
5	3.34	22
125	8.05	20
210	10	20
500	17	22
1100	34	44
1500	64	75



Analysis

- Exist an “threshold” (other cores has same law)
  - If task runtime < threshold, coarse-grained app runtime is better than fine-grained(2x~7X). Hotspot of fine-grained is scheduling latency
  - If task runtime > threshold, distance of coarse-grained and fine is a static time. Hotspot of fine-grained is task runtime
- How to calculate threshold
  - Threshold ≈ 64+16*(cores-1)
  - For cores=24, threshold is 432ms
  - For cores=96(4 machine, 24 cores each machine), threshold ≈ 64+ 16*(96-1) = 1.58s

Conclude

- If task runtime of spark app << threshold, we could choose coarse-grained mode, and benefit by short app runtime.
- If task runtime of spark app > threshold, we could choose fine-grained mode, benefit by resource share.
- App runtime of **Standalone mode** is similar with Mesos **coarse-grained mode**