

Annotation

1 report exported from PdfHighlights.com on 5/27/2018 6:02:44 AM

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

C:\Users\Navroz\Documents\ed-etc\Hadoop\Spark\mastering-apache-spark--jacek-lackowski.pdf

15

15

17

20

20

20

21

24

28

147

148

150

150

151

152

153

255

260

277

278

278

279

280

280

281

281

285

285

288

289

293

296

297

297

297

298

298

300

300

301

301

302

303

304

304

339

339

339

340

340

341

343

344

344

344

344

345

346

355

356

367

370

371

376

377

377

379

380

381

412

412

413

414

414

414

415

415

415

417

418

419

420

421

422

424

424

425

425

429

429

442

444

444

445

445

447

448

449

450

450

458

459

461

624

885

894

1128

1147

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Line

Line

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Underline

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Underline

Highlight: Underline

Highlight: Highlight

Highlight: Underline

Rectangle

Highlight: Underline

Rectangle

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Popup: Comment: heartbeatReceiverRef is initialized when the Executor is created:

```
private val heartbeatReceiverRef = RpcUtils.makeDriverRef(HeartbeatReceiver.ENDPOINT_NAME, conf, env.rpcEnv)  
This occurs BEFORE running the private method startDriverHeartbeat()
```

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Popup: Comment: java.util.concurrent.ScheduledThreadPoolExecutor

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Decorative

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Underline

Highlight: Underline

Highlight: Underline

Highlight: Highlight

Highlight: Underline

Highlight: Highlight

Highlight: Underline

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Underline

Highlight: Highlight

Highlight: Highlight

Highlight: Underline

Highlight: Underline

Highlight: Underline

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Underline

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Underline

Rectangle

Highlight: Highlight

Highlight: Underline

Highlight: Underline

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Underline

Rectangle

Highlight: Underline

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Free Text: namely Scala, Python, Java & R
c# module too Also for .NET?

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Line

Line

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Line

Line

Highlight: Highlight

Line

Line

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Underline

Highlight: Highlight

Highlight: Highlight

Highlight: Underline

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Underline

Highlight: Underline

Rectangle

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Underline

Highlight: Underline

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Underline

Highlight: Highlight

Highlight: Underline

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Rectangle

Rectangle

res1: Long = 30

This creates an RDD of 30 partitions and gloms it. The count of array objects in the new RDD, containing data from each partition, is also 30.

glom could be used as a quick way to put all of an RDD's elements into a single array. You could first repartition the RDD into one partition and then call glom. The result is an RDD with a single array element containing all of the RDD's previous elements. Of course, this applies only to RDDs small enough so that all of their elements fit into a single partition.

from-SSpark-in_Action

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

-
Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

.....o.....o.....o.....

Highlight: Underline

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Underline

Highlight: Underline

Highlight: Underline

Highlight: Underline

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Line

Line

Highlight: Highlight

Rectangle

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Underline

Highlight: Highlight

Highlight: Underline

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

- - - -

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Line

Rectangle

Line

Line

Rectangle

Highlight: Highlight

Line

Line

Line

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Underline

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Underline

Highlight: Highlight

Highlight: Highlight

Highlight: Underline

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

o o - o o -

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Line

Line

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Highlight: Highlight

Highlight: Highlight

Rectangle

Rectangle

Highlight: Highlight

Highlight: Highlight

Apache Spark

Apache Spark is an open-source distributed general-purpose cluster framework with (mostly) in-memory data processing engine that can do machine learning and graph processing on large volumes of data at rest (or in motion (streaming processing)) with rich concise high-level APIs for the languages: Scala, Python, Java, R, and SQL.

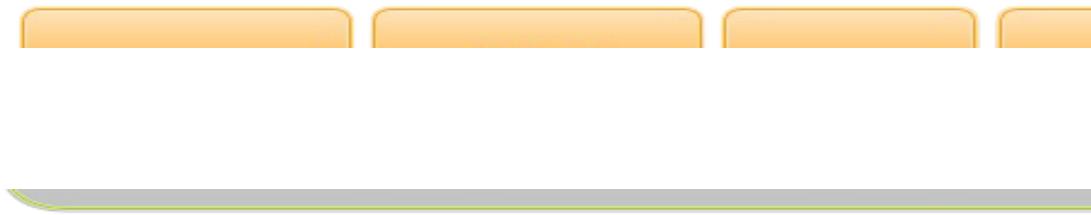


Figure 1. The Spark Platform

You could also describe Spark as a distributed, data processing engine for streaming modes featuring SQL queries, graph processing, and machine

In contrast to Hadoop's two-stage disk-based MapReduce computation engine, the multi-stage (mostly) in-memory computing engine allows for running most of the time in memory, and hence most of the time provides better performance for certain e.g. iterative algorithms or interactive data mining (read [Spark officially supports large-scale sorting](#)).

Spark aims at speed, ease of use, extensibility and interactive analytics.

Using Spark Application Frameworks, Spark simplifies access to machine predictive analytics at scale.

Spark is mainly written in [Scala](#), but provides developer API for languages and [R](#).

Note	Microsoft's Mobius project provides C# API for Spark "enabling implementation of Spark driver program and data processing of languages supported in the .NET framework like C# or F#."
------	--

If you have large amounts of data that requires low latency processing than MapReduce program cannot provide, Spark is a viable alternative.

- Access any data type across any data source.
- Huge demand for storage and data processing.

The Apache Spark project is an umbrella for [SQL](#) (with Datasets), [stream learning](#) (pipelines) and [graph](#) processing engines built atop Spark Core. all in a single application using a consistent API.

Spark runs locally as well as in clusters, on-premises or in cloud. It runs on YARN, Apache Mesos, standalone or in the cloud (Amazon EC2 or IBM Bluemix).

Spark can access data from many [data sources](#).

Apache Spark's Streaming and SQL programming models with MLlib and easier for developers and data scientists to build applications that exploit big data and graph analytics.

At a high level, any Spark application creates **RDDs** out of some input, runs [transformations](#) of these RDDs to some other form (shape), and finally performs [actions](#) to collect or store data. Not much, huh?

You can look at Spark from programmer's, data engineer's and administrator's point of view. And to be honest, all three types of people will spend quite a lot of their time working with Spark.

Spark combines batch, interactive, and streaming workloads under one roof.

Spark supports **near real-time streaming workloads** via [Spark Streaming](#) framework.

ETL workloads and Analytics workloads are different, however Spark attempts to provide a unified platform for a wide variety of workloads.

Sources

Spark can read from many types of data sources — relational, NoSQL, file-based, using many types of data formats - Parquet, Avro, CSV, JSON.

Both, input and output data sources, allow programmers and data engineers to interact with the platform with the large amount of data that is read from or saved to for analysis.

feels like. Time, personal preference, operating system you work on are all factors that influence what is right at a time (and using a hammer can be a reasonable choice).

Spark embraces many concepts in a single unified development and runtime environment:

- Machine learning that is so tool- and feature-rich in Python, e.g. SciKit-learn can be used by Scala developers (as Pipeline API in Spark MLlib or calling Python code from Scala).
- DataFrames from R are available in Scala, Java, Python, R APIs.
- Single node computations in machine learning algorithms are migrated to distributed versions in Spark MLlib.

This single platform gives plenty of opportunities for Python, Scala, Java, C/C++ programmers as well as data engineers (SparkR) and scientists (using pythonic APIs).

Low-level Optimizations

Apache Spark uses a **directed acyclic graph (DAG) of computation stages DAG**. It postpones any processing until really required for actions. Spark gives plenty of opportunities to induce low-level optimizations (so users have do more).

Mind the proverb less is more.

Overview of Apache Spark

Spark supports diverse workloads, but successfully targets low-latency iterations are often used in **Machine Learning and graph algorithms**.

Many Machine Learning algorithms require plenty of iterations before the optimal, like logistic regression. The same applies to graph algorithms to find nodes and edges when needed. Such computations can increase their performance if the interim partial results are stored in memory or at very fast solid state drives.

Spark can cache intermediate data in memory for faster model building and training. This means that the data is loaded to memory (as an initial step), reusing it multiple times instead of reading from disk, which can lead to performance slowdowns.

Also, graph algorithms can traverse graphs one connection per iteration without having to store the entire graph in memory.

Less disk access and network can make a huge difference when you need to process large amounts of data even when it is a BIG Data.

In the no-so-long-ago times, when the most prevalent distributed computing system was [Hadoop MapReduce](#), you could reuse a data between computation (even after you've written it to an external storage like [Hadoop Distributed Filesystem](#)) but it can cost you a lot of time to compute even very basic multi-stage computations because it suffers from IO (and perhaps network) overhead.

One of the many motivations to build Spark was to have a framework that makes data reuse easier.

Spark cuts it out in a way to keep as much data as possible in memory and reuse it until a job is finished. It doesn't matter how many stages belong to a job. What matters is the available memory and how effective you are in using Spark API (so far).

The less network and disk IO, the better performance, and Spark tries hard to minimize both.

Caution

I'm new to Machine Learning as a discipline and Spark MLlib mistakes in this document are considered a norm (not an ex

Spark MLlib is a module (a library / an extension) of Apache Spark to provide machine learning algorithms on top of Spark's RDD abstraction. Its goal is development and usage of large scale machine learning.

You can find the following types of machine learning algorithms in MLlib:

- Classification
- Regression
- Frequent itemsets (via FP-growth Algorithm)
- Recommendation
- Feature extraction and selection
- Clustering
- Statistics
- Linear Algebra

You can also do the following using MLlib:

- Model import and export
- Pipelines

Note

There are two libraries for Machine Learning in Spark MLlib:
`org.apache.spark.mllib` for RDD-based Machine Learning API under
`org.apache.spark.ml` for DataFrame-based Machine Pipelines.

Machine Learning uses large datasets to identify (infer) patterns and make predictions). Automated decision making is what makes Machine Learning. You can teach a system from a dataset and let the system act by itself to

The amount of data (measured in TB or PB) is what makes Spark MLlib efficient since a human could not possibly extract much value from the dataset in a reasonable time.

Observation

An **observation** is used to learn about or evaluate (i.e. draw conclusions) observed item's target value.

Spark models observations as rows in a `DataFrame`.

Feature

A **feature** (aka *dimension* or *variable*) is an attribute of an observation. It is a **variable**.

Spark models features as columns in a `DataFrame` (one per feature or a set of them).

Note Ultimately, it is up to an algorithm to expect one or many features.

ML Pipelines (`spark.ml`)

The **ML Pipeline API** (aka **Spark ML** or **spark.ml**) due to the package the API allows users quickly and easily assemble and configure practical distributed Machine Learning pipelines (aka workflows) by standardizing the APIs for different Machine Learning algorithms.

Note Both `scikit-learn` and `GraphLab` have the concept of **pipelines** in their system.

Over Structured Data on Massive Scale

Like Apache Spark in general, **Spark SQL** in particular is all about distributed computations on massive scale.

The primary difference between Spark SQL's and the "bare" Spark Core's models is the framework for loading, querying and persisting structured and data using **structured queries** that can be expressed using good ol' **SQL** custom high-level SQL-like, declarative, type-safe **Dataset API** called **Structured DSL**.

Tip

You can find more information about Spark SQL in my [Mastering DataFrames notebook](#).

Datasets

Spark Structured Streaming is a new computation model introduced in Spark 2.0 for building end-to-end streaming applications termed as **continuous applications**.

Structured streaming offers a high-level declarative streaming API built on top of DStreams (inside Spark SQL's engine) for continuous incremental execution of structured data processing pipelines.

Tip

You can find more information about Spark Structured Streaming in my [notebook titled "Spark Structured Streaming"](#).

Spark Shell — spark-shell shell script

Spark shell is an interactive environment where you can learn how to make Apache Spark quickly and conveniently.

Tip

Spark shell is particularly helpful for fast interactive prototyping.

Under the covers, **Spark shell** is a standalone Spark application written in environment with auto-completion (using `TAB` key) where you can **run** and get familiar with the features of Spark (that help you in developing your own Spark applications). It is a very convenient tool to explore the many things

with immediate feedback. It is one of the many reasons why Spark is so popular for processing datasets of any size.

There are variants of Spark shell for different languages: `spark-shell` for Python and `sparkR` for R.

Note This document (and the book in general) uses `spark-shell` for Python.

You can start Spark shell using `spark-shell` script.

```
$ ./bin/spark-shell  
scala>
```

`spark-shell` is an extension of Scala REPL with automatic instantiation of `spark` (and `SparkContext` as `sc`).

```
scala> :type spark  
org.apache.spark.sql.SparkSession  
  
// Learn the current version of Spark in use  
scala> spark.version  
res0: String = 2.1.0-SNAPSHOT
```

`spark-shell` also imports Scala SQL's implicits and `sql` method.

```
scala> :imports  
1) import spark.implicits._          (59 terms, 38 are implicit)  
2) import spark.sql                 (1 terms)
```

Spark Shell — spark-shell shell script

Note

When you execute `spark-shell` you actually execute [Spark submit command](#):

```
org.apache.spark.deploy.SparkSubmit --class  
org.apache.spark.repl.Main --name Spark shell  
shell
```

Set `SPARK_PRINT_LAUNCH_COMMAND` to see the entire command to refer to [Print Launch Command of Spark Scripts](#).

Using Spark shell

```
org.apache.spark.sql.SparkSession
```

Besides, there is also `sc` value created which is an instance of [SparkContext](#):

```
scala> :type sc  
org.apache.spark.SparkContext
```

To close Spark shell, you press `ctrl+D` or type in `:q` (or any subset of

1	show at <console>:24	2016/09/29 14:01:14	87 ms	0/1 (1 failed)
---	----------------------	---------------------	-------	----------------

Figure 1. Welcome page - Jobs page

Every `SparkContext` launches its own instance of Web UI which is available at `http://[driver]:4040` by default (the port can be changed using `spark.ui.port`, it will increase if this port is already taken (until an open port is found)).

web UI comes with the following tabs (which may not all be visible at once, created on demand, e.g. `Streaming` tab):

1. [Jobs](#)
2. [Stages](#)
3. [Storage](#) with RDD size and memory use
4. [Environment](#)
5. [Executors](#)
6. [SQL](#)

Tip

You can use the web UI after the application has finished by persisting the `EventLoggingListener` and using `Spark History Server`.

Web UI — Spark Application's Web Console

Note

All the information that is displayed in web UI is available thanks to `JobProgressListener` and other `SparkListeners`. One could say that web layer is built on top of Spark listeners.

Settings

SparkLauncher — Launching Spark Applications Programmatically

```
import org.apache.spark.launcher.SparkLauncher

val command = new SparkLauncher()
.setAppResource("SparkPi")
.setVerbose(true)

val appHandle = command.startApplication()
```

Spark Architecture

Spark uses a **master/worker architecture**. There is a **driver** that talks to a **coordinator called master** that manages **workers** in which **executors** run.

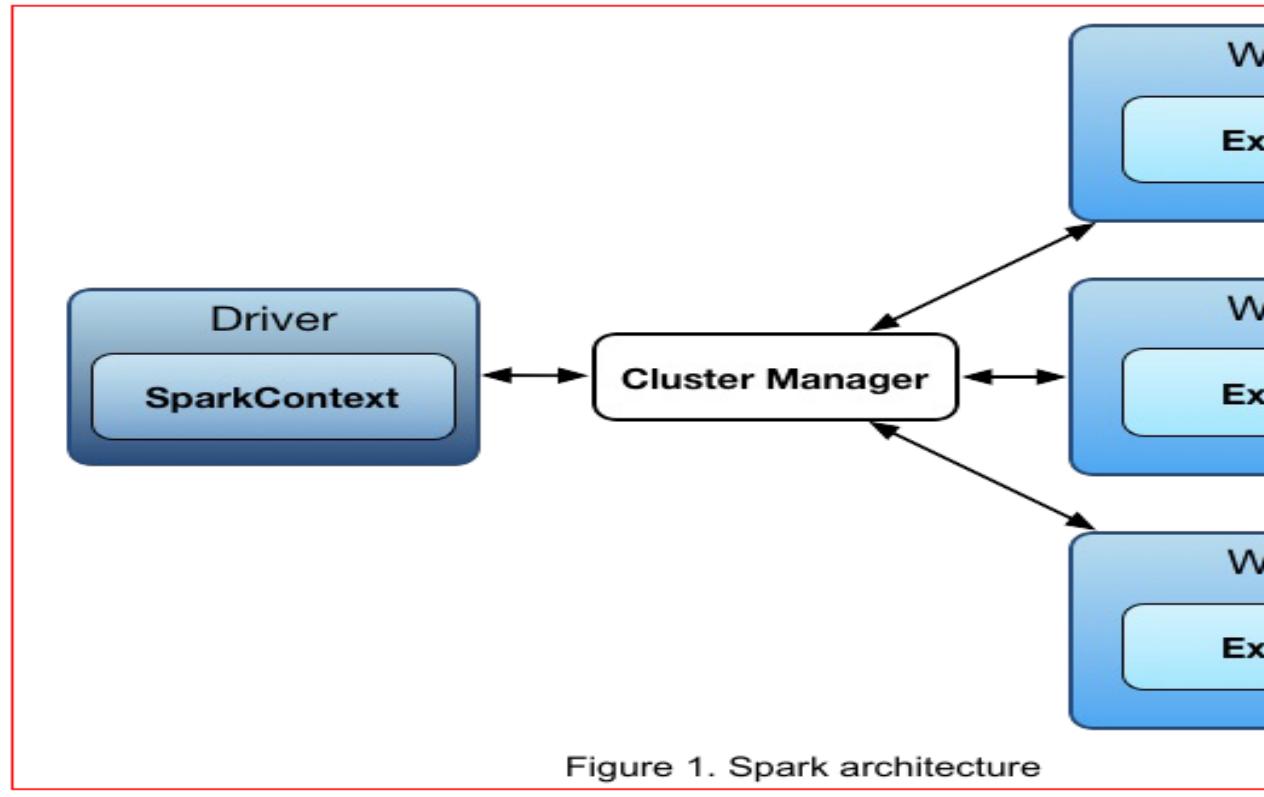


Figure 1. Spark architecture

The driver and the executors run in their own Java processes. You can run them in the same (*horizontal cluster*) or separate machines (*vertical cluster*) or in a mixed configuration.

Driver

A **Spark driver** (aka an application's driver process) is a JVM process **SparkContext** for a Spark application. It is the **master node** in a Spark app. It is the **cockpit of jobs and tasks execution** (using **DAGScheduler** and **TaskHosts**) **Web UI** for the environment.

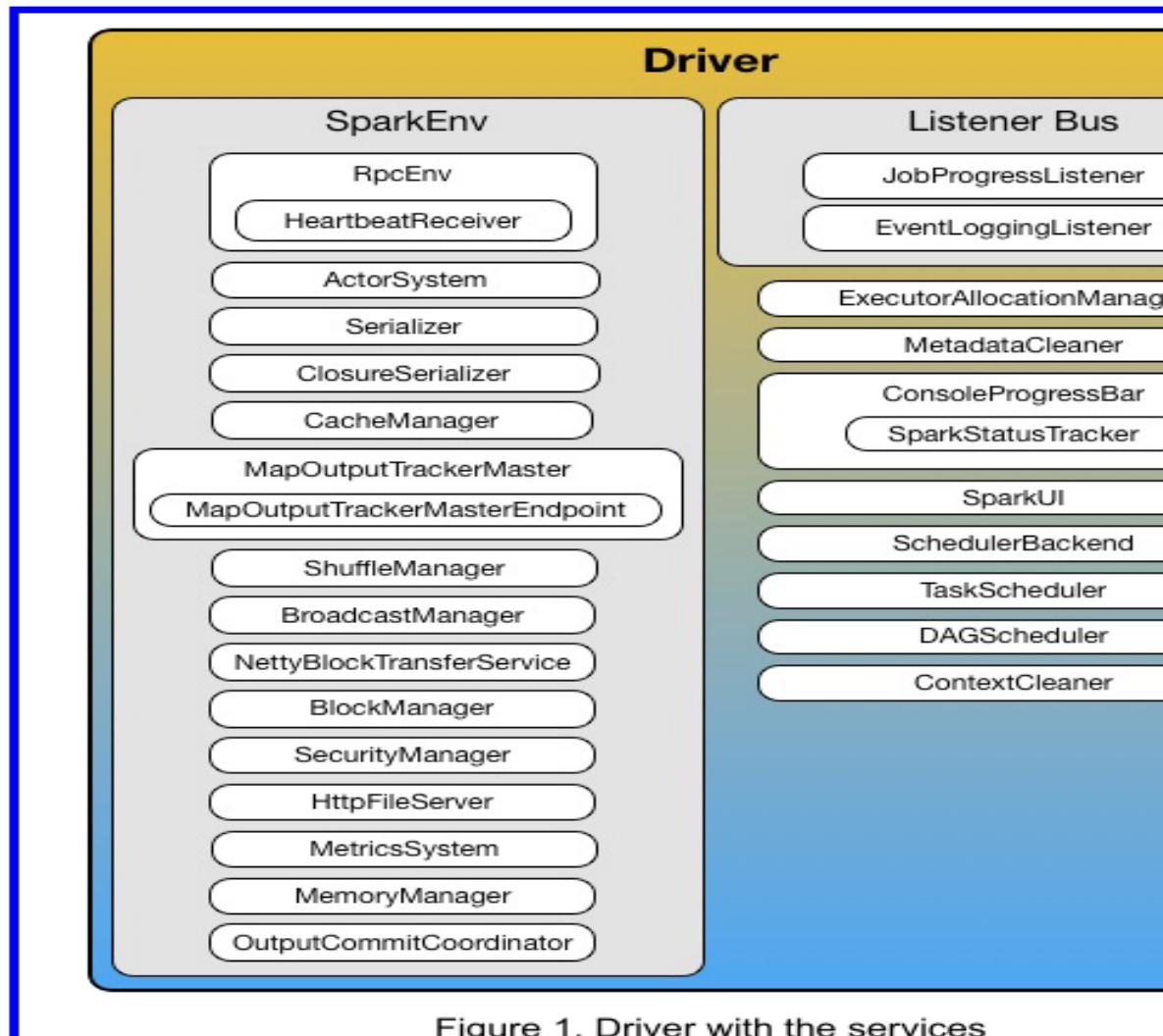


Figure 1. Driver with the services

It splits a Spark application into tasks and schedules them to run on executors.

A driver is where the task scheduler lives and spawns tasks across workers.

A driver coordinates workers and overall execution of tasks.

Driver

Driver requires the additional services (beside the common ones like `ShuffleService`, `MemoryManager`, `BlockTransferService`, `BroadcastManager`, `CacheManager`)

- Listener Bus
- RPC Environment
- `MapOutputTrackerMaster` with the name **MapOutputTracker**
- `BlockManagerMaster` with the name **BlockManagerMaster**
- `HttpFileServer`
- `MetricsSystem` with the name **driver**
- `OutputCommitCoordinator` with the endpoint's name **OutputCommitCoordinator**

Caution

`FIXME` Diagram of `RpcEnv` for a driver (and later executors) should be in the notes about `RpcEnv`?

- High-level control flow of work
- Your Spark application runs as long as the Spark driver.
 - Once the driver terminates, so does your Spark application.
- Creates `SparkContext`, `RDD's, and executes transformations and actions
- Launches tasks

Driver's Memory

Executor

Executor is a distributed agent that is responsible for executing tasks.

Executor is created when:

- CoarseGrainedExecutorBackend receives RegisteredExecutor messages (Standalone and YARN)
- Spark on Mesos's MesosExecutorBackend does registered
- LocalEndpoint is created (for local mode)

Executor typically runs for the entire lifetime of a Spark application which allocation of executors (but you could also opt in for dynamic allocation)

Note Executors are managed exclusively by executor backends.

Executors reports heartbeat and partial metrics for active tasks to Heartbeat Endpoint on the driver.

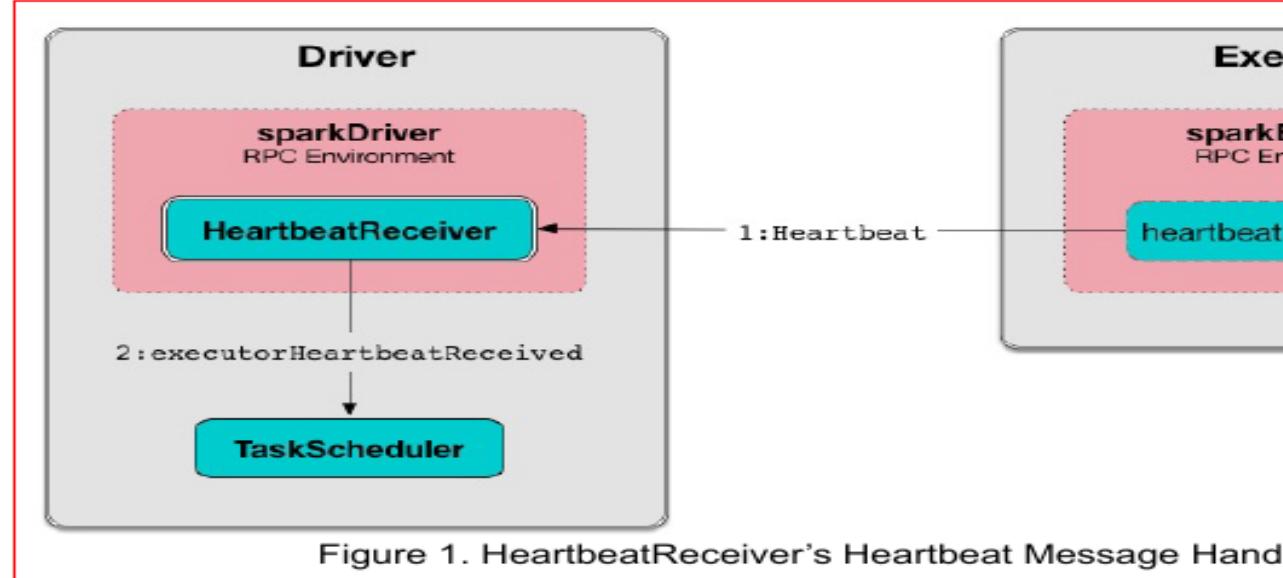


Figure 1. HeartbeatReceiver's Heartbeat Message Hand

Executors provide in-memory storage for RDDs that are cached in Spark Block Manager).

When an executor starts it first registers with the driver and communicates execute tasks.

Executor

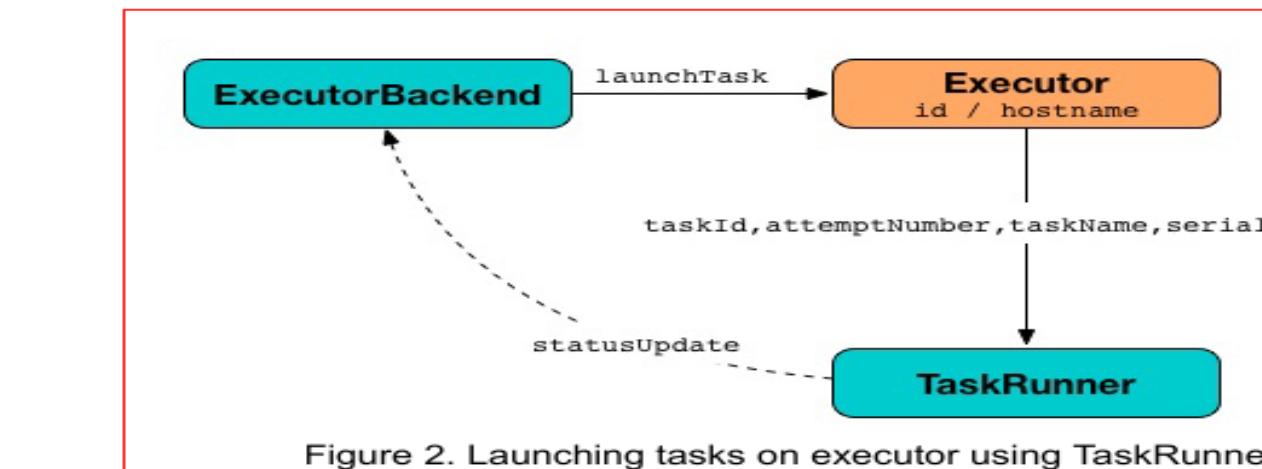


Figure 2. Launching tasks on executor using TaskRunner

Executor offers are described by executor id and the host on which an executor runs (see [Resource Offers](#) in this document).

Executors can run multiple tasks over its lifetime, both in parallel and sequentially. Executors track **running tasks** (by their task ids in **runningTasks** internal registry). See the [Tasks](#) section.

Executors use a **Executor task launch worker thread pool** for launching tasks.

Executors send **metrics** (and heartbeats) using the **internal heartbeater - Heartbeat Thread**.

It is recommended to have as many executors as data nodes and as many cores as you can get from the cluster.

Executors are described by their **id**, **hostname**, **environment** (as defined in [Environment](#)), **classpath** (and, less importantly, and more for internal optimization, whether it's **local** or **cluster mode**).

Caution

FIXME How many cores are assigned per executor?

`id`), and `BlockManagerId` is sent to `HeartbeatReceiver RPC endpoint` (with `spark.executor.heartbeatInterval` timeout).

Caution

FIXME When is `heartbeatReceiverRef` created? 

If the response `requests to reregister BlockManager`, you should see the following:

Tip

Read about TaskMetrics in [TaskMetrics](#).

Reporting Heartbeat and Partial Metrics for Active Driver — `reportHeartBeat` Internal Method

```
reportHeartBeat(): Unit
```

`reportHeartBeat` collects TaskRunners for currently running tasks (aka active) and their tasks deserialized (i.e. either ready for execution or already started).

Note

TaskRunner has task deserialized when it runs the task.

For every running task, `reportHeartBeat` takes its TaskMetrics and:

- Requests ShuffleRead metrics to be merged
- Sets jvmGCTime metrics

`reportHeartBeat` then records the latest values of internal and external accumulators for every task.

Note

Internal accumulators are a task's metrics while external accumulators are the Spark application's accumulators that a user has created.

`reportHeartBeat` sends a blocking Heartbeat message to `HeartbeatReceiver` (running on the driver). `reportHeartBeat` uses `spark.executor.heartbeatInterval` timeout.

Note

A Heartbeat message contains the executor identifier, the accumulated updates, and the identifier of the `BlockManager`.

Note

`reportHeartBeat` uses `SparkEnv` to access the current `BlockManager`.

Executor

If the response (from HeartbeatReceiver endpoint) is to re-register the BlockManager to re-register (which will register the blocks the BlockManager the driver).

```
INFO Told to re-register on heartbeat
```

Note

HeartbeatResponse requests BlockManager to re-register when TaskScheduler or HeartbeatReceiver know nothing about the executors.

When posting the Heartbeat was successful, reportHeartBeat resets the internal counter.

In case of a non-fatal exception, you should see the following WARN message (followed by the stack trace).

```
WARN Issue communicating with driver in heartbeater
```

Every failure reportHeartBeat increments heartbeat failures up to spark.executor.heartbeat.maxFailures Spark property. When the heartbeat reaches the maximum, you should see the following ERROR message in the logs terminates with the error code: 56 .

```
ERROR Exit as unable to send heartbeats to driver more than [HEARTBEAT] times
```

Note

reportHeartBeat is used when Executor schedules reporting partial metrics for active tasks to the driver (that happens every spark.executor.heartbeatInterval Spark property).

heartbeater — Heartbeat Sender Thread

heartbeater is a daemon ScheduledThreadPoolExecutor with a single thread.

The name of the thread pool is driver-heartbeater.

Coarse-Grained Executors

TaskRunner

TaskRunner is a thread of execution that manages a single individual task.

TaskRunner is created exclusively when Executor is requested to launch a task.

Worker

Anatomy of Spark Application

Every Spark application starts from creating **SparkContext**.

Note

Without **SparkContext** **no computation** (as a Spark job) can be

Note

A **Spark application** is an instance of **SparkContext**. Or, put it di
context **constitutes** a Spark application.

A Spark application is uniquely identified by a pair of the **application** and **application id**s.

For it to work, you have to **create a Spark configuration using** **SparkConf** **and** **SparkContext constructor**.

```
package pl.japila.spark
```

5. Execute `count` action

Tip

Spark shell creates a Spark context and SQL context for you at

When a Spark application starts (using **spark-submit script** or as a standa
connects to **Spark master** as described by **master URL**. It is part of **Spark**
initialization.

Your Spark application

Figure 1. Submitting Spark application to master using master

Note	Your Spark application can run locally or on the cluster which is managed by a cluster manager and the deploy mode (<code>--deploy-mode</code>). Refer to Deploy Modes .
------	--

You can then [create RDDs](#), [transform them to other RDDs](#) and ultimately [reduce them](#). You can also [cache interim RDDs](#) to speed up data processing.

After all the data processing is completed, the Spark application finishes by calling [stop](#) on the [Spark context](#).

Mandatory Settings - `spark.master` and `spark.app.name`

There are [two mandatory settings](#) of any Spark application that have to be specified when you run the application. These are `spark.master` and `spark.app.name`. This means that this Spark application could be run — `spark-submit` [spark.app.name](#).

Spark Properties

Every user program starts with creating an instance of `SparkConf` that holds the configuration properties for the application. This configuration includes the [URL to connect to](#) (`spark.master`), the [name for your Spark application](#) (`spark.app.name`) which is displayed in [web UI](#) and becomes `spark.app.name` and other Spark properties.

SparkConf — Programmable Configuration for Spark Applications

proper runs. The `instance of` `SparkConf` `can be used to create` `SparkContext`.

Tip

Start `Spark shell` with `--conf spark.logConf=true` to log the effective configuration as INFO when `SparkContext` is started.

```
$ ./bin/spark-shell --conf spark.logConf=true
...
15/10/19 17:13:49 INFO SparkContext: Running Spark version 1.6.3
15/10/19 17:13:49 INFO SparkContext: Spark configuration:
spark.app.name=Spark shell
spark.home=/Users/jacek/dev/oss/spark
spark.jars=
spark.logConf=true
spark.master=local[*]
spark.repl.class.uri=http://10.5.10.20:64055
spark.submit.deployMode=client
...
```

Use `sc.getConf.toDebugString` to have a richer output once Spark finished initializing.

You can query for the values of Spark properties in `Spark shell` as follows:

There are the following places where a Spark application looks for Spark properties (in order of importance from the least important to the most important):

- `conf/spark-defaults.conf` - the configuration file with the default Spark properties. [Read `spark-defaults.conf`.](#)
- `--conf` or `-c` - the command-line option used by `spark-submit` (and other tools that use `spark-submit` or `spark-class` under the covers, e.g. `spark-shell`)
- `SparkConf`

Default Configuration

SparkConf — Programmable Configuration for Spark Applications

```
import org.apache.spark.SparkConf  
val conf = new SparkConf
```

It simply loads `spark.*` system properties.

You can use `conf.toDebugString` or `conf.getAll` to have the `spark.*` system properties loaded printed out.

```
scala> conf.getAll  
res0: Array[(String, String)] = Array((spark.app.name,Spark shell), (spark.master,local[*]), (spark.submit.deployMode,client))  
  
scala> conf.toDebugString  
res1: String =  
spark.app.name=Spark shell  
spark.jars=  
  
spark.master=local[*]  
spark.submit.deployMode=client
```

```
scala> println(conf.toDebugString)  
spark.app.name=Spark shell  
spark.jars=  
spark.master=local[*]
```

Method

```
getAppId: String
```

getAppId gives spark.app.id Spark property or reports NoSuchElementException

Note

getAppId is used when:

- NettyBlockTransferService is initialized (and creates a NettyBlockRpcServer as well as saves the identifier for later).
- Executor is created (in non-local mode and requests BlockManager).

Settings

SparkConf — Programmable Configuration for Spark Applications

Table 1. Spark Properties

Spark Property	Default Value	Desc
<code>spark.master</code>		Master URL
<code>spark.app.id</code>	<code>TaskScheduler.applicationId()</code>	Unique identifier for application that can uniquely identify it.
<code>spark.app.name</code>		Set when <code>SparkContext</code> is created (right after <code>TaskScheduler</code> is initialized). It actually gives the application name.

Spark properties are the means of tuning the execution environment for applications.

The default Spark properties file is `$SPARK_HOME/conf/spark-defaults.conf` overriden using `spark-submit`'s `--properties-file` command-line option

Table 1. Environment Variables

Environment Variable	Default Value	Description
<code>SPARK_CONF_DIR</code>	<code> \${SPARK_HOME}/conf</code>	Spark's configuration directory (<code>conf</code>)

Tip Read the official documentation of Apache Spark on [Spark Configuration](#).

spark-defaults.conf — Default Spark Properties

`spark-defaults.conf` (under `SPARK_CONF_DIR` or `$SPARK_HOME/conf`) is the file with the Spark properties of your Spark applications.

Note `spark-defaults.conf` is loaded by `AbstractCommandBuilder's loadPropertiesFile internal method`.

Calculating Path of Default Spark Properties — `Utils.getDefaultPropertiesFile` method

```
getDefaultPropertiesFile(env: Map[String, String] = sys.env): String
```

`getDefaultPropertiesFile` calculates the absolute path to `spark-defaults.conf` file that can be either in directory specified by `SPARK_CONF_DIR` environment variable or `$SPARK_HOME/conf` directory.

Note `getDefaultPropertiesFile` is a part of `private[spark] org.apache.spark.util.Utils` object.

Environment Variables

Deploy Mode

Deploy mode specifies the location of where driver executes in the deployment environment.

Deploy mode can be one of the following options:

- `client` (default) - the driver runs on the machine that the Spark application launched.
- `cluster` - the driver runs on a random node in a cluster.

Note `cluster` deploy mode is only available for non-local cluster deployment.

You can control the deploy mode of a Spark application using `spark-submit` command-line option or `spark.submit.deployMode` Spark property.

Note `spark.submit.deployMode` setting can be `client` or `cluster`.

Client Deploy Mode

Caution

FIXME

spark.submit.deployMode

`spark.submit.deployMode` (default: `client`) can be `client` or `cluster`.

SparkContext — Entry Point to Spark

SparkContext (aka **Spark context**) is the heart of a Spark application.

Note You could also assume that a SparkContext instance is a Spark master.

Spark context sets up internal services and establishes a connection to a environment.

Once a SparkContext is created you can use it to create RDDs, accumulate broadcast variables, access Spark services and run jobs (until SparkContext is closed).

A Spark context is essentially a client of Spark's execution environment and master of your Spark application (don't get confused with the other meaning of master, though).

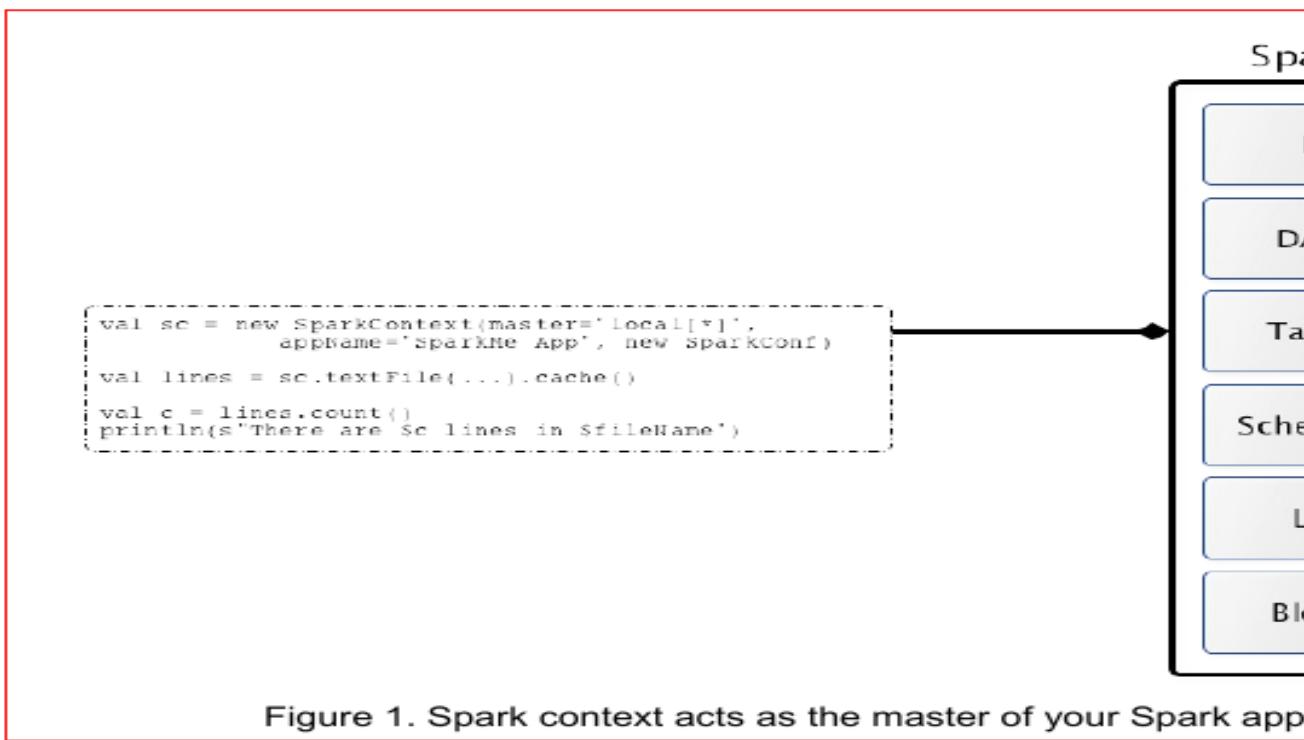


Figure 1. Spark context acts as the master of your Spark application. SparkContext offers the following functions:

- Getting current status of a Spark application

Refer to [Logging](#).

Removing RDD Blocks from BlockManagerMaster — `unpersistRDD` Internal Method

```
unpersistRDD(rddId: Int, blocking: Boolean = true): Unit
```

`unpersistRDD` **requests** `BlockManagerMaster` **to remove the blocks for the rddId**).

Note `unpersistRDD` uses `SparkEnv` **to access the current `BlockManager`**, turn used to **access the current `BlockManagerMaster`**.

SparkContext

`unpersistRDD` **removes** `rddId` **from `persistentRdds` registry**.

In the end, `unpersistRDD` **posts a `SparkListenerUnpersistRDD` (with `rddId`)** `LiveListenerBus Event Bus`.

Note `unpersistRDD` **is used when:**

- `ContextCleaner` **does `doCleanupRDD`**
- `SparkContext` **unpersists an RDD** (i.e. marks an RDD as removable)

Unique Identifier of Spark Application — `applicationId`

Caution

FIXME

Accessing persistent RDDs — `getPersistent` Method

```
getPersistentRDDs: Map[Int, RDD[_]]
```

`getPersistentRDDs` returns the collection of RDDs that have marked them persistent via `cache`.

Internally, `getPersistentRDDs` returns `persistentRdds` internal registry.

Cancelling Job — `cancelJob` Method

```
cancelJob(jobId: Int)
```

`cancelJob` **requests** DAGScheduler **to cancel a Spark job.**

Cancelling Stage — `cancelStage` Methods

```
cancelStage(stageId: Int): Unit  
cancelStage(stageId: Int, reason: String): Unit
```

`cancelStage` **simply requests** DAGScheduler **to cancel a Spark stage (with reason).**

Note `cancelStage` is used when StagesTab handles a kill request (from web UI).

Programmable Dynamic Allocation

SparkContext **offers the following methods as the developer API for dynamic executors:**

- `requestExecutors`
- `killExecutors`
- `requestTotalExecutors`
- `(private!) getExecutorIds`

Requesting New Executors — `requestExecutors`

```
requestExecutors(numAdditionalExecutors: Int): Boolean
```

`requestExecutors` **requests** numAdditionalExecutors **executors from CoarseGrainedSchedulerBackend.**

Requesting to Kill Executors — `killExecutors`

Requesting Total Executors — `requestTotalExecutors` Method

```
requestTotalExecutors(  
    numExecutors: Int,  
    localityAwareTasks: Int,  
    hostToLocalTaskCount: Map[String, Int]): Boolean
```

`requestTotalExecutors` is a `private[spark]` method that requests the executors from a coarse-grained scheduler backend.

Note	It works for coarse-grained scheduler backends only.
------	--

When called for other scheduler backends you should see the following W in the logs:

WARN Requesting executors is only supported in coarse-grained mode

Getting Executor Ids — `getExecutorIds` Method

SparkContext

Getting Existing or Creating New SparkContext — `getOrCreate` Methods

```
getOrCreate(): SparkContext
getOrCreate(conf: SparkConf): SparkContext
```

`getOrCreate` methods allow you to get the existing `SparkContext` or create a new one.

```
import org.apache.spark.SparkContext
val sc = SparkContext.getOrCreate()

// Using an explicit SparkConf object
import org.apache.spark.SparkConf
val conf = new SparkConf()
  .setMaster("local[*]")
  .setAppName("SparkMe App")
val sc = SparkContext.getOrCreate(conf)
```

The no-param `getOrCreate` method requires that the two mandatory `SparkConf` and `application name` - are specified using `spark-submit`.

Constructors

```
SparkContext()
SparkContext(conf: SparkConf)
SparkContext(master: String, appName: String, conf: SparkConf)
SparkContext(
  master: String,
  appName: String,
  sparkHome: String = null,
  jars: Seq[String] = Nil,
  environment: Map[String, String] = Map())
```

You can create a `SparkContext` instance using the four constructors.

```
INFO SparkContext: Running Spark version 2.0.0-SNAPSHOT
```

Note

[Only one](#) `SparkContext` may be running in a [single JVM](#) (check [2243 Support multiple SparkContexts in the same JVM](#)). Sharing `SparkContext` in the JVM is the solution to share data within Spark relying on other means of data sharing using external data stor

Accessing Current SparkEnv — `env` Method

Submitting Jobs Asynchronously — `submitJob`

```
submitJob[T, U, R](  
    rdd: RDD[T],  
    processPartition: Iterator[T] => U,  
    partitions: Seq[Int],  
    resultHandler: (Int, U) => Unit,  
    resultFunc: => R): SimpleFutureAction[R]
```

`submitJob` submits a job in an asynchronous, non-blocking way to DAGScheduler.

It cleans the `processPartition` input function argument and returns an instance of `SimpleFutureAction` that holds the `JobWalker` instance.

Caution

[FIXME](#) What are `resultFunc` ?

SparkContext

It is used in:

- `AsyncRDDActions` methods
- Spark Streaming for `ReceiverTrackerEndpoint.startReceiver`

Spark Configuration

SparkContext and RDDs

You use a Spark context to create RDDs (see [Creating RDD](#)).

When an RDD is created, it belongs to and is completely owned by the SparkContext it originated from. RDDs can't by design be shared between SparkContexts.

SparkContext

Creating RDD — `parallelize` Method

SparkContext allows you to create many different RDDs from input sources.

- Scala's collections, i.e. `sc.parallelize(0 to 100)`
- local or remote filesystems, i.e. `sc.textFile("README.md")`

SparkContext

- Any Hadoop InputSource using `sc.newAPIHadoopFile`

Read [Creating RDDs in RDD - Resilient Distributed Dataset.](#)

Caution	FIXME
<code>unpersist removes an RDD from the master's Block Manager (calls removeRDD on the master's BlockManager)</code> <code>blocking: Boolean)) and the internal persistentRdds mapping.</code>	
<code>It finally posts SparkListenerUnpersistRDD message to listenerBus .</code>	

Setting Checkpoint Directory — `setCheckpointDir`

SparkContext

```
longAccumulator: LongAccumulator
longAccumulator(name: String): LongAccumulator
doubleAccumulator: DoubleAccumulator
doubleAccumulator(name: String): DoubleAccumulator
collectionAccumulator[T]: CollectionAccumulator[T]
collectionAccumulator[T](name: String): CollectionAccumulator[T]
```

You can use `longAccumulator`, `doubleAccumulator` or `collectionAccumulator` to register accumulators for simple and collection values.

`longAccumulator` returns `LongAccumulator` with the zero value `0`.

`doubleAccumulator` returns `DoubleAccumulator` with the zero value `0.0`.

`collectionAccumulator` returns `CollectionAccumulator` with the zero value `java.util.List[T]`.

```
scala> val acc = sc.longAccumulator
acc: org.apache.spark.util.LongAccumulator = LongAccumulator(id: 0, name: null)

scala> val counter = sc.longAccumulator("counter")
counter: org.apache.spark.util.LongAccumulator = LongAccumulator(id: 1, name: counter), value: 0

scala> counter.value
res0: Long = 0

scala> sc.parallelize(0 to 9).foreach(n => counter.add(n))

scala> counter.value
res3: Long = 45
```

Creating Broadcast Variable — `broadcast` Method

```
broadcast[T](value: T): Broadcast[T]
```

`broadcast` method creates a broadcast variable. It is a shared memory with broadcast blocks) on the driver and later on all Spark executors.

```
val sc: SparkContext = ???  
sc.broadcast("hello")
```

```
hello: org.apache.spark.broadcast.Broadcast[String] = Broadcast(0)
```

Spark transfers the value to Spark executors once, and tasks can share it without repetitive network transmissions when the broadcast variable is used multiple times.

Figure 4. Broadcasting a value to executors

Internally, `broadcast` requests the current `BroadcastManager` to create a broadcast variable.

Note

The current `BroadcastManager` is available using `SparkEnv.broadcastManager` attribute and is always `BroadcastManager` (with few internal code changes to reflect where it runs, i.e. inside the driver or executor).

You should see the following INFO message in the logs:

```
INFO SparkContext: Created broadcast [id] from [callSite]
```

If `contextCleaner` is defined, the new broadcast variable is registered for

Note

Spark does not support broadcasting RDDs.

```
scala> sc.broadcast(sc.range(0, 10))
java.lang.IllegalArgumentException: requirement failed: Can
at scala.Predef$.require(Predef.scala:224)
```

Running Job Synchronously — `runJob` Method

RDD actions run jobs using one of `runJob` methods.

SparkContext

```
runJob[T, U](  
    rdd: RDD[T],  
    func: (TaskContext, Iterator[T]) => U,  
    partitions: Seq[Int],  
    resultHandler: (Int, U) => Unit)  
runJob[T, U](  
    rdd: RDD[T],  
    func: (TaskContext, Iterator[T]) => U,  
    partitions: Seq[Int]): Array[U]  
runJob[T, U](  
    rdd: RDD[T],  
    func: Iterator[T] => U,  
    partitions: Seq[Int]): Array[U]  
runJob[T, U](rdd: RDD[T], func: (TaskContext, Iterator[T]) => U): Array[U]  
runJob[T, U](rdd: RDD[T], func: Iterator[T] => U): Array[U]  
runJob[T, U](  
    rdd: RDD[T],  
    processPartition: (TaskContext, Iterator[T]) => U,  
    resultHandler: (Int, U) => Unit)  
runJob[T, U: ClassTag](  
    rdd: RDD[T],  
    processPartition: Iterator[T] => U,  
    resultHandler: (Int, U) => Unit)
```

runJob **executes a function on one or many partitions of a RDD (in a SparkContext) to produce a collection of values per partition.**

Note

runJob can only work when a SparkContext is *not stopped*.

Internally, runJob first makes sure that the SparkContext is *not stopped*. see the following IllegalStateException exception in the logs:

```
... 48 elided
```

runJob then calculates the call site and cleans a func closure.

You should see the following INFO message in the logs:

```
INFO: SparkRoulette: RDD is ready for dependency.
[toString]
```

runJob requests DAGScheduler to run a job.

Tip	runJob just prepares input parameters for DAGScheduler to run a job.
-----	--

After DAGScheduler is done and the job has finished, runJob stops consequently and performs RDD checkpointing of rdd.

Tip	For some actions, e.g. first() and lookup(), there is no need to partition the partitions of the RDD in a job. And Spark knows it.
-----	--

```
// RDD to work with
```

Tip	Read TaskContext.
-----	-------------------

Running a job is essentially executing a func function on all or a subset of rdd RDD and returning the result as an array (with elements being the results of each partition).

Figure 6. Executing action

Stopping SparkContext — stop Method

```
stop(): Unit
```

stop **stops the** SparkContext .

Internally, stop **enables** stopped **internal flag**. If already stopped, you should see the following INFO message in the logs:

```
INFO SparkContext: SparkContext already stopped.
```

stop **then does the following:**

1. **Removes** `_shutdownHookRef` **from** `ShutdownHookManager`
2. **Posts a** `SparkListenerApplicationEnd` **(to** `LiveListenerBus` **Event Bus)**
3. **Stops web UI**
4. **Requests** `MetricSystem` **to report metrics** **(from all registered sinks)**
5. **Stops** `ContextCleaner`
6. **Requests** `ExecutorAllocationManager` **to stop**

SparkContext

7. If `LiveListenerBus` was started, requests `LiveListenerBus` to stop
8. Requests `EventLoggingListener` to stop
9. Requests `DAGScheduler` to stop
10. Requests `RpcEnv` to stop `HeartbeatReceiver` endpoint
11. Requests `ConsoleProgressBar` to stop
12. Clears the reference to `TaskScheduler`, i.e. `_taskScheduler` is `null`
13. Requests `sparkEnv` to stop and clears `SparkEnv`
14. Clears `SPARK_YARN_MODE` flag
15. Clears an active `sparkContext`

Ultimately, you should see the following INFO message in the logs:

```
INFO SparkContext: Successfully stopped SparkContext
```

HeartbeatReceiver RPC Endpoint

HeartbeatReceiver is a ThreadSafeRpcEndpoint registered on the driver HeartbeatReceiver.

HeartbeatReceiver receives Heartbeat messages from executors that Spark mechanism to receive accumulator updates (with task metrics and a Spark accumulators) and pass them along to TaskScheduler .

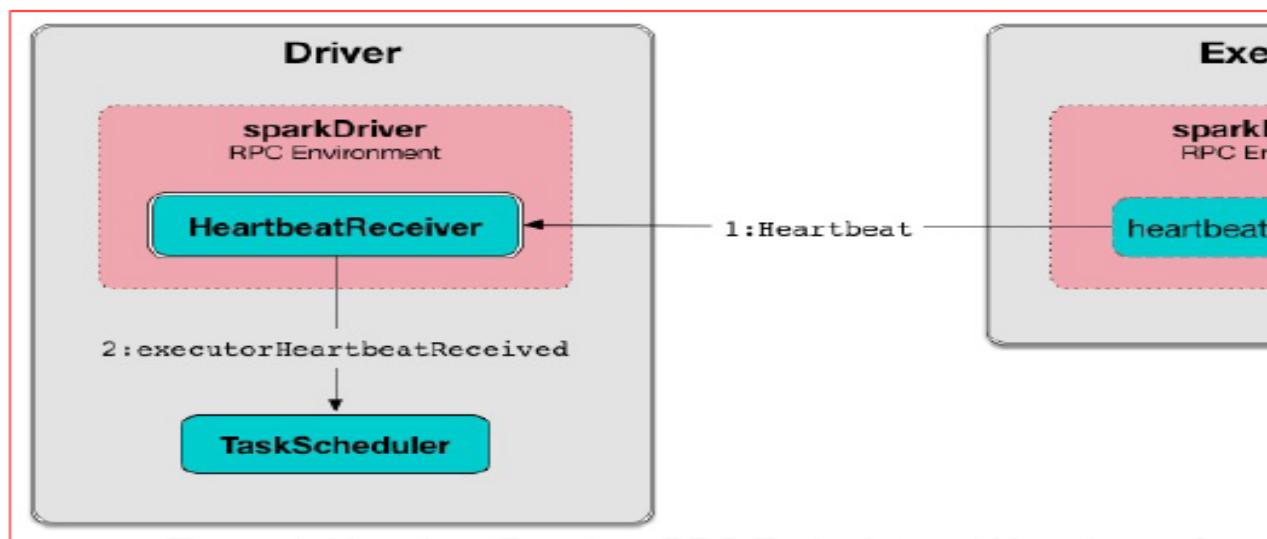


Figure 1. HeartbeatReceiver RPC Endpoint and Heartbeats from

Note	HeartbeatReceiver is registered immediately after a Spark application started, i.e. when SparkContext is created.
------	---

HeartbeatReceiver is a SparkListener to get notified when a new executor longer available in a Spark application. HeartbeatReceiver tracks executor (executorLastSeen registry) to handle Heartbeat and ExpireDeadHosts messages for executors that are assigned to the Spark application.

HeartbeatReceiver RPC Endpoint

Table 1. HeartbeatReceiver RPC Endpoint's Messages (in alphabetical order)

Message	Description
ExecutorRemoved	Posted when <code>HeartbeatReceiver</code> is notified that the executor is no longer available (to a Spark application).
ExecutorRegistered	Posted when <code>HeartbeatReceiver</code> is notified that the executor has been registered (with a Spark application).
ExpireDeadHosts	FIXME
Heartbeat	Posted when <code>Executor</code> informs that it is alive and provides task metrics.
TaskSchedulerIsSet	Posted when <code>SparkContext</code> informs that the TaskScheduler is available.

Table 2. HeartbeatReceiver's Internal Registries and Counters

Name	Description
executorLastSeen	Executor ids and the timestamps of when the last heartbeat was received.
scheduler	TaskScheduler

Enable DEBUG or TRACE logging levels for `org.apache.spark.HeartbeatReceiver` to see what happens inside.

Refer to Logging.

Creating HeartbeatReceiver Instance

HeartbeatReceiver takes the following when created:

- SparkContext
- Clock

HeartbeatReceiver registers itself as a SparkListener.

HeartbeatReceiver initializes the internal registries and counters.

Heartbeat

```
Heartbeat(executorId: String,  
          accumUpdates: Array[(Long, Seq[AccumulatorV2[_, _]])],  
          blockManagerId: BlockManagerId)
```

When received, `HeartbeatReceiver` finds the `executorId` executor (in `executorLastSeen` registry).

When the executor is found, `HeartbeatReceiver` updates the time the heartbeat was received (in `executorLastSeen`).

Note

`HeartbeatReceiver` uses the internal `Clock` to know the current time.

`HeartbeatReceiver` then submits an asynchronous task to notify `TaskScheduler` that a heartbeat was received from the executor (using `TaskScheduler` internal registry). `HeartbeatReceiver` posts a `HeartbeatResponse` back to the executor (with information about `TaskScheduler` whether the executor has been registered already or not so it need to re-register).

If however the executor was not found (in `executorLastSeen` registry), i.e. not registered before, you should see the following DEBUG message in the logs:

Note

The example uses Spark in [local mode](#), but the initialization with [cluster modes](#) would follow similar steps.

Creating `SparkContext` instance starts by setting the internal `allowMultipleContexts` with the value of `spark.driver.allowMultipleContexts` and marking this `SparkContext` as partially constructed. It makes sure that no other thread is creating a `SparkContext` instance in this JVM. It does so by synchronizing on `SPARK_CONTEXT_CONSTRUCTOR` using the internal atomic reference `activeContext` (that eventually has a `SparkContext` instance).

Note

The entire code of `SparkContext` that creates a fully-working `SparkContext` instance is between two statements:

```
SparkContext.markPartiallyConstructed(this, allowMultipleContexts)
// the SparkContext code goes here
SparkContext.setActiveContext(this, allowMultipleContexts)
```

`startTime` is set to the current time in milliseconds.

`stopped` internal flag is set to `false`.

RDD — Resilient Distributed Dataset

Resilient Distributed Dataset (aka **RDD**) is the primary data abstraction and the **core of Spark** (that I often refer to as "Spark Core").

The origins of RDD

The original paper that gave birth to the concept of RDD is [Resilient Distributed Fault-Tolerant Abstraction for In-Memory Cluster Computing](#) by Matei Zaharia et al.

A RDD is a resilient and distributed collection of records spread over one or more nodes.

Note

One could compare RDDs to collections in Scala, i.e. a RDD is a collection of records spread over many JVMs while a Scala collection lives on a single JVM.

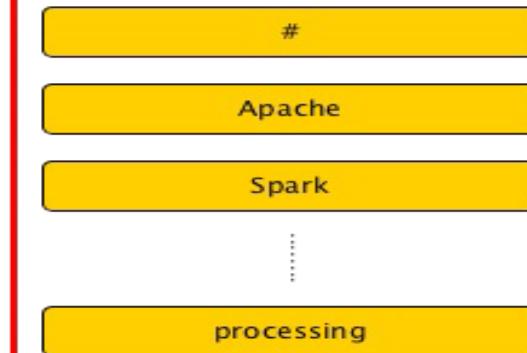
Using RDD Spark hides data partitioning and so distribution that in turn allows us to design parallel computational framework with a higher-level programming interface than four mainstream programming languages.

namely Scala, Python, Java, C# module too A

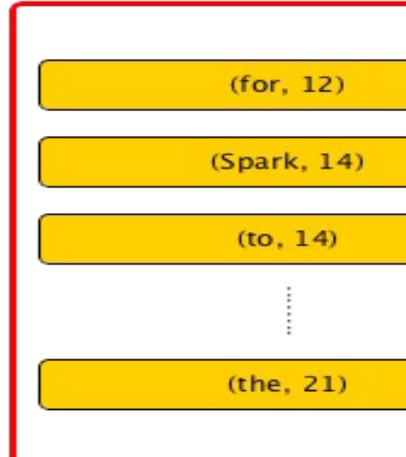
The features of RDDs (decomposing the name):

- **Resilient**, i.e. fault-tolerant with the help of [RDD lineage graph](#) and [partitioner](#) that can recompute missing or damaged partitions due to node failures.
- **Distributed** with data residing on multiple nodes in a [cluster](#).
- **Dataset** is a collection of [partitioned data](#) with primitive values or value pairs (tuples) or other objects (that represent records of the data you work with).

RDD of Strings



RDD of Pairs



RDD — Resilient Distributed Dataset

From the scaladoc of `org.apache.spark.rdd.RDD`:

A Resilient Distributed Dataset (RDD), the basic abstraction in Spark. It is an immutable, partitioned collection of elements that can be operated on in parallel.

From the original paper about RDD - Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing:

Resilient Distributed Datasets (RDDs) are a distributed memory abstraction that programmers perform in-memory computations on large clusters in a fault-tolerant manner.

Beside the above traits (that are directly embedded in the name of the data type RDD) it has the following additional traits:

- **In-Memory**, i.e. data inside RDD is stored in memory as much (size) as possible.
- **Immutable or Read-Only**, i.e. it does not change once created and can only be transformed using transformations to new RDDs.
- **Lazy evaluated**, i.e. the data inside RDD is not available or transformed until it is executed that triggers the execution.
- **Cacheable**, i.e. you can hold all the data in a persistent "storage" like memory (the most preferred) or disk (the least preferred due to access speed).
- **Parallel**, i.e. process data in parallel.
- **Typed** — RDD records have types, e.g. `Long` in `RDD[Long]` or `(Int, String)`.
- **Partitioned** — records are partitioned (split into logical partitions) and distributed across nodes in a cluster.
- **Location-Stickiness** — RDD can define placement preferences to co-locate data (as close to the records as possible).

Note

Preferred location (aka *locality preferences* or *placement preferences*) is information about the locations of RDD records that the `DAGScheduler` uses to place computing partitions on to have them close to the data as possible.

RDD — Resilient Distributed Dataset

partitions - logical chunks (parts) of data. The logical division is for processing, internally it is not divided whatsoever. Each partition comprises of **records**.

distributed and partitioned RDD

Figure 2. RDDs

Partitions are the units of parallelism. You can control the number of partitions using **repartition** or **coalesce** transformations. Spark tries to be as close to HDFS or Cassandra, and partitions.

RDDs support two kinds of operations:

- **transformations** - lazy operations that return another RDD.
- **actions** - operations that trigger computation and return values.

The motivation to create RDD were (after the authors) two types of applications computing frameworks handle inefficiently:

- **iterative algorithms** in machine learning and graph computations.
- **interactive data mining tools** as ad-hoc queries on the same dataset.

The goal is to reuse intermediate in-memory results across multiple data-processing workloads with no need for copying large amounts of data over the network.

Technically, RDDs follow the contract defined by the five main intrinsic properties:

- List of parent RDDs that are the dependencies of the RDD.
- An array of partitions that a dataset is divided to.
- A compute function to do a computation on partitions.

RDD — Resilient Distributed Dataset

- An optional Partitioner that defines how keys are hashed, and the partitioning of key-value RDDs)
- Optional preferred locations (aka locality info), i.e. hosts for a partition where records live or are the closest to read from.

This RDD abstraction supports an expressive set of operations without having to implement a scheduler for each one.

An RDD is a named (by `name`) and uniquely identified (by `id`) entity in a SparkContext (available as `context` property).

RDDs live in one and only one `SparkContext` that creates a logical boundary between them.

Note

RDDs cannot be shared between `SparkContexts` (see `SparkContext.broadcast`).

An RDD can optionally have a friendly name accessible using `name` that is used for logging.

```
    res: String = (o) Friendly name ParallelCollectionRDD[2] at parallel
```

24 []

RDDs are a container of instructions on how to materialize big (arrays of) and how to split it into partitions so Spark (using executors) can hold some.

In general data distribution can help executing processing in parallel so a chunk of data that it could eventually keep in memory.

Spark does jobs in parallel, and RDDs are split into partitions to be processed parallel. Inside a partition, data is processed sequentially.

RDD — Resilient Distributed Dataset

Saving partitions results in part-files instead of one single file (unless there's one partition).

checkpointRDD Internal Method

Caution

FIXME

persist Methods

```
persist(): this.type  
persist(newLevel: StorageLevel): this.type
```

Refer to [Persisting RDD — persist Methods](#).

persist Internal Method

```
persist(newLevel: StorageLevel, allowOverride: Boolean): this.type
```

Caution

FIXME

Note

[persist](#) is used when `RDD` is requested to [persist](#) itself and [local checkpointing](#).

RDD Contract

```
abstract class RDD[T] {  
    def compute(split: Partition, context: TaskContext): Iterator[T]  
    def getPartitions: Array[Partition]  
    def getDependencies: Seq[Dependency[_]]  
    def getPreferredLocations(split: Partition): Seq[String] = Nil  
    val partitioner: Option[Partitioner] = None  
}
```

Note

RDD is an abstract class in Scala.

Table 1. RDD Contract

Method	Description
compute	Used exclusively when RDD computes a value (possibly by reading from a checkpoint).
getPartitions	Used exclusively when RDD is requested for its partitions (called only once as the value is cached).
getDependencies	Used when RDD is requested for its dependencies (called only once as the value is cached).
getPreferredLocations	Defines placement preferences of a partition. Used exclusively when RDD is requested for its preferred locations of a partition.
partitioner	Defines the Partitioner of a RDD.

Types of RDDs

There are some of the most interesting types of RDDs:

- `ParallelCollectionRDD`
- `CoGroupedRDD`
- `HadoopRDD` is an RDD that provides core functionality for reading data using the older MapReduce API. The most notable use case is the reading of `SparkContext.textFile`.
- `MapPartitionsRDD` - a result of calling operations like `map`, `flatMap`, `mapPartitions`, etc.
- `CoalescedRDD` - a result of `repartition` or `coalesce` transformations.
- `ShuffledRDD` - a result of shuffling, e.g. after `repartition` or `coalesce` transformations.
- `PipedRDD` - an RDD created by piping elements to a forked external process.
- `PairRDD` (implicit conversion by `PairRDDFunctions`) that is an RDD of pairs that is a result of `groupByKey` and `join` operations.
- `DoubleRDD` (implicit conversion as `org.apache.spark.rdd.DoubleRDDFunctions`) that is an RDD of doubles.
- `SequenceFileRDD` (implicit conversion as `org.apache.spark.rdd.SequenceFileRDDFunctions`) that is an RDD that reads from SequenceFile.

Appropriate operations of a given RDD type are automatically available on the right type, e.g. `RDD[(Int, Int)]`, through implicit conversion in Scala.

Transformations

A **transformation** is a lazy operation on a RDD that returns another RDD. These operations include `flatMap`, `filter`, `reduceByKey`, `join`, `cogroup`, etc.

Tip

Go in-depth in the section [Transformations](#).

Actions

An **action** is an operation that triggers execution of **RDD transformations** (to a Spark driver - the user program).

Tip

Go in-depth in the section [Actions](#).

SparkContext.parallelize

One way to create a RDD is with `SparkContext.parallelize` method. It accepts of elements as shown below (`sc` is a `SparkContext` instance):

```
scala> val rdd = sc.parallelize(1 to 1000)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallel:25
```

Given the reason to use Spark to process more data than your own laptop, `SparkContext.parallelize` is mainly used to learn Spark in the Spark shell. `SparkContext.parallelize` requires all the data to be available on a single `Spark driver` - that eventually hits the limits of your laptop.

SparkContext.makeRDD

Caution

[FIXME](#) What's the use case for `makeRDD` ?

```
scala> sc.makeRDD(0 to 1000)
res0: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[1] at make:25
```

SparkContext.textFile

One of the easiest ways to create an RDD is to use `SparkContext.textFile`.

You can use the local `README.md` file (and then `flatMap` over the lines in the `RDD` of words).

```
scala> val words = sc.textFile("README.md").flatMap(_.split("\\w+"))
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[27] at flat
:24
```

Note

You `cache` it so the computation is not performed every time you access `words`.

Creating RDDs from Input

Execute the following Spark job and you will see how the number of partitions changes.

```
ints.repartition(2).count
```

Note

position in the owning RDD.

Computing Partition (in TaskContext) — `compute`

```
compute(split: Partition, context: TaskContext): Iterator[T]
```

The abstract `compute` method computes the input `split` partition in the produce a collection of values (of type `T`).

`compute` is implemented by any type of RDD in Spark and is called every are requested unless RDD is cached or checkpointed (and the records can external storage, but this time closer to the compute node).

When an RDD is cached, for specified storage levels (i.e. all but `NONE`) requested to get or compute partitions.

Note

`compute` method runs on the driver.

Defining Placement Preferences of RDD Partitions — `preferredLocations` — `Final Method`

RDD — Resilient Distributed Dataset

```
preferredLocations(split: Partition): Seq[String]
```

`preferredLocations` **requests** `CheckpointRDD` **for placement preferences (checkpointed)** or **calculates them itself**.

Note

`preferredLocations` is a template method that uses `getPreferredLocations`. Custom RDDs can override to specify placement preferences for themselves. `getPreferredLocations` defines no placement preferences by default.

Note

`preferredLocations` is mainly used when `DAGScheduler` computes locations for missing partitions.

The other usages are to define the locations by custom RDDs,

- (Spark Core) `BlockRDD`, `CoalescedRDD`, `HadoopRDD`, `NewParallelCollectionRDD`, `ReliableCheckpointRDD`, `ShuffledRDD`
- (Spark SQL) `KafkaSourceRDD`, `ShuffledRowRDD`, `FileScanRDD`, `StateStoreRDD`
- (Spark Streaming) `KafkaRDD`, `WriteAheadLogBackedBlockRDD`

Getting Number of Partitions — `getNumPartitions` Method

```
getNumPartitions: Int
```

`getNumPartitions` gives the number of partitions of a RDD.

RDD Lineage — Logical Execution Plan

RDD Lineage (aka *RDD operator graph* or *RDD dependency graph*) is a graph of parent RDDs of a RDD. It is built as a result of applying transformations to RDDs, which creates a logical execution plan.

Note

The execution DAG or physical execution plan is the DAG of RDD lineage.

Note

The following diagram uses `cartesian` or `zip` for learning purposes. You may use other operators to build a RDD graph.

r00

r01

```
val r00 = sc.parallelize(0 to 9)
val r01 = sc.parallelize(0 to 90 by 10)
val r10 = r00 cartesian r01
val r11 = r00.map(n => (n, n))
val r12 = r00 zip r01
val r13 = r01.keyBy(_ / 20)
val r20 = Seq(r11, r12, r13).foldLeft(r10)(_ union _)
```

A RDD lineage graph is hence a graph of what transformations need to be action has been called.

You can learn about a RDD lineage graph using `RDD.toDebugString` method.

Logical Execution Plan

Logical Execution Plan starts with the earliest RDDs (those with no dependencies or reference cached data) and ends with the RDD that produces the action that has been called to execute.

ParallelCollectionRDD

ParallelCollectionRDD is an RDD of a collection of elements with optional locationPrefs.

ParallelCollectionRDD is the result of `SparkContext.parallelize` and `SparkContext.makeRDD` methods.

The data collection is split on to numSlices slices.

It uses ParallelCollectionPartition.

- flatMap
- filter
- glom
- mapPartitions

Operators - Transformations and Actions

RDDs have two types of operations: transformations and actions.

Note

Operators are also called operations.

Transformations

Transformations are lazy operations on a RDD that create one or many
map , filter , reduceByKey , join , cogroup , randomSplit .

```
transformation: RDD => RDD  
transformation: RDD => Seq[RDD]
```

In other words, transformations are *functions* that take a RDD as the input
or many RDDs as the output. They do not change the input RDD (since RDDs
are immutable and hence cannot be modified), but always produce one or more
RDDs by applying the computations they represent.

By applying transformations you incrementally build a RDD lineage with all
of the final RDD(s).

Transformations are lazy, i.e. are not executed immediately. Only after calling
transformations executed.

After executing a transformation, the result RDD(s) will always be different
and can be smaller (e.g. filter , count , distinct , sample), bigger (e.g.
union , cartesian) or the same size (e.g. map).

Caution

There are transformations that may trigger jobs, e.g. sortBy
etc.

There are two kinds of transformations:

- narrow transformations
- wide transformations

Narrow Transformations

Narrow transformations are the result of `map`, `filter` and such that is a single partition only, i.e. it is self-sustained.

An output RDD has partitions with records that originate from a single partition of the input RDD. Only a limited subset of partitions used to calculate the result.

Spark groups narrow transformations as a stage which is called pipelining.

Wide Transformations

Wide transformations are the result of `groupByKey` and `reduceByKey`. They compute the records in a single partition may reside in many partitions of the output RDD.

Transformations

Note

Wide transformations are also called shuffle transformations as they may not depend on a shuffle.

All of the tuples with the same key must end up in the same partition, produced by the same task. To satisfy these operations, Spark must execute RDD shuffle, moving data across cluster and results in a new stage with a new set of partitions.

map

Actions

Actions are [RDD operations](#) that produce non-RDD values. They materialize the RDD in the Spark program. In other words, a RDD operation that returns a value of a type `RDD[T]` [is an action](#).

```
action: RDD => a value
```

Note

Actions are synchronous. You can use [AsyncRDDActions](#) to run them in a separate thread while calling actions.

They trigger execution of [RDD transformations](#) to return values. Simply put, they evaluate the [RDD lineage graph](#).

You can think of actions as a valve and until action is fired, [the data to be returned](#) is [not yet available even in the pipes, i.e. transformations](#). Only actions can materialize the entire [data pipeline with real data](#).

Actions are one of two ways to send data from [executors to the driver](#) (the other way is [accumulators](#)).

Actions in [org.apache.spark.rdd.RDD](#):

- `treeReduce`

Actions run jobs using [SparkContext.runJob](#) or directly [DAGScheduler.runJob](#).

```
scala> words.count  (1)
res0: Long = 500
```

1. `words` is an RDD of `String`.

Tip

You should cache RDDs you work with when you want to execute actions on it for a better performance. Refer to [RDD Caching and Persistence](#).

Before calling an action, Spark does closure/function cleaning (using `ClosureCleaner`) to make it ready for serialization and sending over the wire to executors. It may throw a `SparkException` if the computation cannot be cleaned.

Note

Spark uses `ClosureCleaner` to clean closures.

Asynchronous RDD Actions

RDD Caching and Persistence

Caching or persistence are optimisation techniques for (iterative and interactive) computations. They help saving interim partial results so they can be reused across stages. These interim results as RDDs are thus kept in memory (default) or stored in storages like disk and/or replicated.

RDDs can be cached using `cache` operation. They can also be persisted using `persist` operation.

The difference between `cache` and `persist` operations is purely syntactic. `cache` is a synonym of `persist` or `persist(MEMORY_ONLY)`, i.e. `cache` is merely `persist` at default storage level `MEMORY_ONLY`.

Note

Due to the very small and purely syntactic difference between `cache` and `persist` operations of RDDs the two terms are often used interchangeably.

RDDs can also be `unpersisted` to remove RDD from a permanent storage and/or disk.

Caching RDD — `cache` Method

```
cache(): this.type = persist()
```

`cache` is a synonym of `persist` with `MEMORY_ONLY` storage level.

Persisting RDD — `persist` Methods

```
persist(): this.type  
persist(newLevel: StorageLevel): this.type
```

`persist` marks a RDD for persistence using `newLevel` storage level.

You can only change the storage level once or `persist` reports an `UnsupportedOperationException`:

```
Cannot change storage level of an RDD after it was already assigned a
```

Caching and Persistence

Note

You can pretend to change the storage level of an RDD with all storage level only if the storage level is the same as it is current.

If the RDD is marked as persistent the first time, the RDD is registered to available) and `SparkContext`.

The internal `storageLevel` attribute is set to the input `newLevel` `storageLevel`.

Unpersisting RDDs (Clearing Blocks) — `unpersist` Method

```
unpersist(blocking: Boolean = true): this.type
```

When called, `unpersist` prints the following INFO message to the logs:

- `MEMORY_AND_DISK_SER_2`
- `OFF_HEAP`

You can check out the storage level using `getStorageLevel()` operation.

```
val lines = sc.textFile("README.md")  
  
scala> lines.getStorageLevel  
res0: org.apache.spark.storage.StorageLevel = StorageLevel(disk=false,  
ffheap=false, deserialized=false, replication=1)
```

From the official documentation about Broadcast Variables:

Broadcast variables allow the programmer to keep a read-only variable in machine rather than shipping a copy of it with tasks.

And later in the document:

Explicitly creating broadcast variables is only useful when tasks across need the same data or when caching the data in deserialized form is im

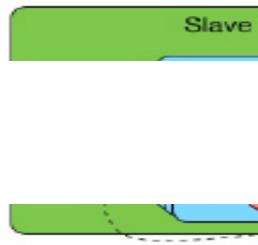


Figure 1. Broadcasting a value to executors

To use a broadcast value in a Spark transformation you have to create it from `SparkContext.broadcast`, and then use `value` method to access the shared data. See the [Introductory Example](#) section.

The Broadcast feature in Spark uses `SparkContext` to create broadcast variables. It uses `BroadcastManager` and `ContextCleaner` to manage their lifecycle.

Broadcast variables

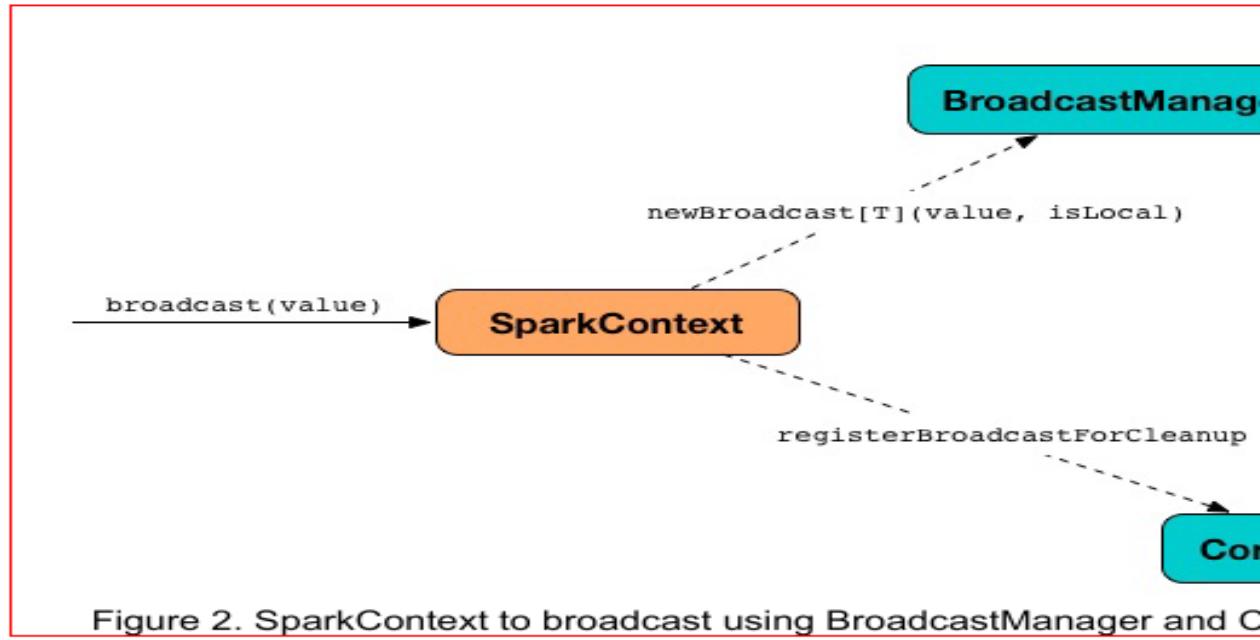


Figure 2. `SparkContext` to broadcast using `BroadcastManager` and `CompletionHandler`

Tip

Not only can Spark developers use broadcast variables for efficient distribution, but Spark itself uses them quite often. A very notable example is when [Spark distributes tasks to executors for their execution](#). That's how the `mapPartitions` operation works.

Table 1. Broadcast API

Method Name	Description
<code>id</code>	The unique identifier
<code>value</code>	The value
<code>unpersist</code>	Asynchronously deletes cached copies on the executors.
<code>destroy</code>	Destroys all data and metadata related to variable.
<code>toString</code>	The string representation

Lifecycle of Broadcast Variable

You can create a broadcast variable of type `T` using `SparkContext.broadcast`.

```
scala> val b = sc.broadcast(1)
b: org.apache.spark.broadcast.Broadcast[Int] = Broadcast@0
```

Enable DEBUG logging level for org.apache.spark.storage.BlockManager

```
DEBUG BlockManager: Putting block broadcast_0_piece0 without replicati
```

After creating an instance of a broadcast variable, you can then reference the **value** method.

```
scala> b.value  
res0: Int = 1
```

Note

value method is the only way to access the value of a broadcast variable.

With DEBUG logging level enabled, you should see the following message:

```
ized, 1 replicas)
```

When you are done with a broadcast variable, you should **destroy** it to release memory.

```
scala> b.destroy
```

With DEBUG logging level enabled, you should see the following message:

```
value: T
```

value returns the value of a broadcast variable. You **can only access the value after the broadcast has been destroyed** after which you will see the following **SparkException** exception:

```
org.apache.spark.SparkException: Attempted to use Broadcast(0) after it was destroyed  
(destroy at <console>:27)
```

```
... 48 elided
```

Internally, `value` makes sure that the broadcast variable is valid, i.e. `described` is called, and, if so, calls the abstract `getValue` method.

Note

`getValue` is abstracted and broadcast variable implementation has to provide a concrete behaviour.

Refer to [TorrentBroadcast](#).

Unpersisting Broadcast Variable — `unpersist` Method

```
unpersist(): Unit  
unpersist(blocking: Boolean): Unit
```

Destroying Broadcast Variable — `destroy` Method

Introduction

Broadcast is part of Spark that is responsible for broadcasting information a cluster.

You use broadcast variable to implement **map-side join**, i.e. a join using lookup tables are distributed across nodes in a cluster using `broadcast` inside `map` (to do the join implicitly).

When you broadcast a value, it is copied to executors only once (while it is times for tasks otherwise). It means that broadcast can help to get your Spark faster if you have a large value to use in tasks or there are more tasks than executors.

It appears that a Spark idiom emerges that uses `broadcast` with `collectAsMap` for broadcast. When an RDD is `map` over to a smaller dataset (column record-wise), `collectAsMap`, and `broadcast`, using the very big RDD to map the broadcast RDDs is computationally faster.

```
(acMap.value.get(a).get, otherMap.value.get(c).get)  
}.collect
```

Use large broadcasted HashMaps over RDDs whenever possible and learn key to lookup necessary data as demonstrated above.

Spark comes with a BitTorrent implementation.

It is not enabled by default.

Broadcast Contract

Accumulators

Accumulators are variables that are "added" to through an associative "add" operation. They act as a container for accumulating partial values across tasks (running on executors). They are designed to be used safely and efficiently in distributed Spark computations and are meant for distributed counter task metrics).

You can create built-in accumulators for longs, doubles, or collections or named accumulators using the `SparkContext.register` methods. You can create a named accumulator with or without a name, but only named accumulators are displayed in web UI for a given stage).

Accumulators	Accumulable	Value

7	7	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms
---	---	---	---------	---------------	--------------------	---------------------	-------

Figure 1. Accumulators in the Spark UI

Accumulator are write-only variables for executors. They can be added to read by the driver only.

```
executor1: accumulator.add(incByExecutor1)
executor2: accumulator.add(incByExecutor2)

driver:   println(accumulator.value)
```

Accumulators are not thread-safe. They do not really have to since the DAGScheduler.updateAccumulators method that the driver uses to update accumulators after a task completes (successfully or with a failure) is only single thread that runs scheduling loop. Beside that, they are write-only data workers that have their own local accumulator reference whereas accessing accumulator is only allowed by the driver.

Accumulators

Accumulators are serializable so they can safely be referenced in the code executors and then safely send over the wire for execution.

```
val counter = sc.longAccumulator("counter")
sc.parallelize(1 to 9).foreach(x => counter.add(x))
```

Internally, longAccumulator, doubleAccumulator, and collectionAccumulator are the built-in typed accumulators and call `SparkContext.register`.

Tip

Read the official documentation about [Accumulators](#).

Caution

FIXME

AccumulatorV2

```
abstract class AccumulatorV2[IN, OUT]
```

AccumulatorV2 **parameterized class represents an accumulator that accu**
values to produce OUT result.

Registering Accumulator — register Method

```
register(  
  sc: SparkContext,  
  name: Option[String] = None,  
  countFailedValues: Boolean = false): Unit
```

register **creates a AccumulatorMetadata metadata object for the accum**
unique identifier) that is then used to register the accumulator with.

Accumulators

In the end, `register` registers the accumulator for cleanup (only when defined in the `SparkContext`).

`register` reports a `IllegalStateException` if `metadata` is already defined that `register` was called already).

Cannot register an Accumulator twice.

Note `register` is a `private[spark]` method.

Note

`register` is used when:

- `SparkContext` registers accumulators
- `TaskMetrics` registers the internal accumulators
- `SQLMetrics` creates metrics.

AccumulatorMetadata

`AccumulatorMetadata` is a container object with the metadata of an accumulator.

- `Accumulator ID`
- `(optional) name`
- `Flag whether to include the latest value of an accumulator on failure`

Note

`countFailedValues` is used exclusively when `Task` collects the accumulators (irrespective of task status — a success or a failure).

Named Accumulators

```
val counter = sc.longAccumulator("counter")
```

AccumableInfo

AccumableInfo contains information about a task's local updates to an accumulator.

- id of the accumulator
- optional name of the accumulator

Accumulators

- optional partial update to the accumulator from a task
- value
- whether or not it is internal
- whether or not to countFailedValues to the final value of the accumulator tasks
- optional metadata

AccumableInfo is used to transfer accumulator updates from executors during an executor heartbeat or when a task finishes.

Create an representation of this with the provided values.

```
....
```

AccumulatorContext

AccumulatorContext is a private[spark] internal object used to track accumulators. Spark itself uses an internal originals lookup table. Spark uses the AccumulatorContext object to register and unregister accumulators.

The originals lookup table maps accumulator identifier to the accumulated value. Every accumulator has its own unique accumulator id that is assigned using nextId counter.

register Method

Caution

FIXME

AccumulatorContext.SQL_ACCUM_IDENTIFIER

AccumulatorContext.SQL_ACCUM_IDENTIFIER is an internal identifier for Spark accumulators. The value is sql and Spark uses it to distinguish Spark SQL from others.

Caution**FIXME**

When `SparkEnv` is created (either for the driver or executors), it instantiates a `SerializerManager` that is then used to create a `BlockManager`.

`SerializerManager` automatically selects the "best" serializer for shuffle batches either by `KryoSerializer` when a RDD's types are known to be compatible or by the `default Serializer`.

The common idiom in Spark's code is to access the current `SerializerManager` via `SparkEnv`.

```
SparkEnv.get.serializerManager
```

Note

`SerializerManager` was introduced in [SPARK-13926](#).

Caution**FIXME**

`SerializerManager` will automatically pick a Kryo serializer for ShuffledRDDs if the key and/or value, and/or combiner types are primitives, arrays of primitives, or strings.

Method

```
getSerializer(keyClassTag: ClassTag[_], valueClassTag: ClassTag[_]): S
```

getSerializer **selects the "best" Serializer given the input types for keys RDD).**

getSerializer **returns KryoSerializer when the types of keys and values with Kryo or the default Serializer .**

Note The default Serializer is defined when `SerializerManager` is

Note getSerializer is used when `ShuffledRDD` returns the single-e dependency list (with `shuffleDependency`).

Settings

System

`MemoryManager` is an abstract base **memory manager** to manage shared execution and storage.

Execution memory is used for computation in shuffles, joins, sorts and a

Storage memory is used for caching and propagating internal data across cluster.

A `MemoryManager` is created when `SparkEnv` is created (one per JVM) and two possible implementations:

1. `UnifiedMemoryManager` — the default memory manager since Spark
2. `StaticMemoryManager` (legacy)

Note

`org.apache.spark.memory.MemoryManager` is a private[spark] S

Spark.

MemoryManager Contract

Every `MemoryManager` obeys the following contract:

- `maxOnHeapStorageMemory`
- `acquireStorageMemory`

`acquireStorageMemory`

Caution

FIXME

maxOnHeapStorageMemory Attribute

maxOnHeapStorageMemory: Long

maxOnHeapStorageMemory is the total amount of memory available for storage. It can vary over time.

Caution

FIXME Where is this used?

It is used in [MemoryStore](#) to ??? and [BlockManager](#) to ???

releaseAllExecutionMemoryForTask

tungstenMemoryMode

tungstenMemoryMode informs others whether Spark works in OFF_HEAP or mode.

It uses `spark.memory.offHeap.enabled` (default: false), `spark.memory.offHeapSize`, and `org.apache.spark.unsafe.Platform.unaligned` before OFF_HEAP is

Caution

FIXME Describe `org.apache.spark.unsafe.Platform.unaligned`

SparkEnv — Spark Runtime Environment

Spark Runtime Environment (`SparkEnv`) is the runtime environment with services that interact with each other to establish a distributed computing Spark application.

Spark Runtime Environment is represented by a `SparkEnv` object that holds runtime services for a running Spark application with separate environments and executors.

The idiomatic way in Spark to access the current `SparkEnv` when on the console is to use `get` method.

```
import org.apache.spark._  
scala> SparkEnv.get  
res0: org.apache.spark.SparkEnv = org.apache.spark.SparkEnv@49322d04
```

SparkEnv — Spark Runtime Environment

Table 1. SparkEnv Services

Property	Service	Description
rpcEnv	RpcEnv	
serializer	Serializer	
closureSerializer	Serializer	
serializerManager	SerializerManager	
mapOutputTracker	MapOutputTracker	
shuffleManager	ShuffleManager	
broadcastManager	BroadcastManager	
blockManager	BlockManager	
securityManager	SecurityManager	
metricsSystem	MetricsSystem	
memoryManager	MemoryManager	
outputCommitCoordinator	OutputCommitCoordinator	

Table 2. SparkEnv's Internal Properties

Name	Initial Value	Description
isStopped	Disabled, i.e. false	Used to mark SparkEnv as stopped.
driverTmpDir		

Enable INFO or DEBUG logging level for org.apache.spark.SparkEnv to see what happens inside.

Creating "Base" SparkEnv — `create` Method

```
create(  
    conf: SparkConf,  
    executorId: String,  
    hostname: String,  
    port: Int,  
    isDriver: Boolean,  
    isLocal: Boolean,  
    numUsableCores: Int,  
    listenerBus: LiveListenerBus = null,  
    mockOutputCommitCoordinator: Option[OutputCommitCoordinator] = None)
```

`create` is a **internal helper method to create a "base" SparkEnv** regardless environment, i.e. a driver or an executor.

Table 3. `create`'s Input Arguments and Their Usage

Input Argument	Usage
<code>bindAddress</code>	Used to create <code>RpcEnv</code> and <code>NettyBlockTransferService</code>
<code>advertiseAddress</code>	Used to create <code>RpcEnv</code> and <code>NettyBlockTransferService</code>
<code>numUsableCores</code>	Used to create <code>MemoryManager</code> , <code>NettyBlockTransferService</code> and <code>BlockManager</code>

When executed, `create` creates a `Serializer` (based on `spark.serializer`). You should see the following `DEBUG` message in the logs:

```
DEBUG SparkEnv: Using serializer: [serializer]
```

It creates another `Serializer` (based on `spark.closure.serializer`).

It creates a `ShuffleManager` based on `spark.shuffle.manager` Spark property.

It creates a `MemoryManager` based on `spark.memory.useLegacyMode` setting. If `useLegacyMode` is `false`, it creates a `UnifiedMemoryManager` being the default and `numcores` is the input `numUsableCores`.

`create` creates a `NettyBlockTransferService`. It uses `spark.driver.blockManager.port` for the port on the driver and `spark.blockManager.port` for the port on executor.

Caution

FIXME A picture with `SparkEnv`, `NettyBlockTransferService` and `BlockManager` is "armed".

SparkEnv — Spark Runtime Environment

`create` creates a **BlockManagerMaster** object with the `BlockManagerMaster` reference (by registering or looking it up by name and `BlockManagerMaster` input `SparkConf`, and the input `isDriver` flag.

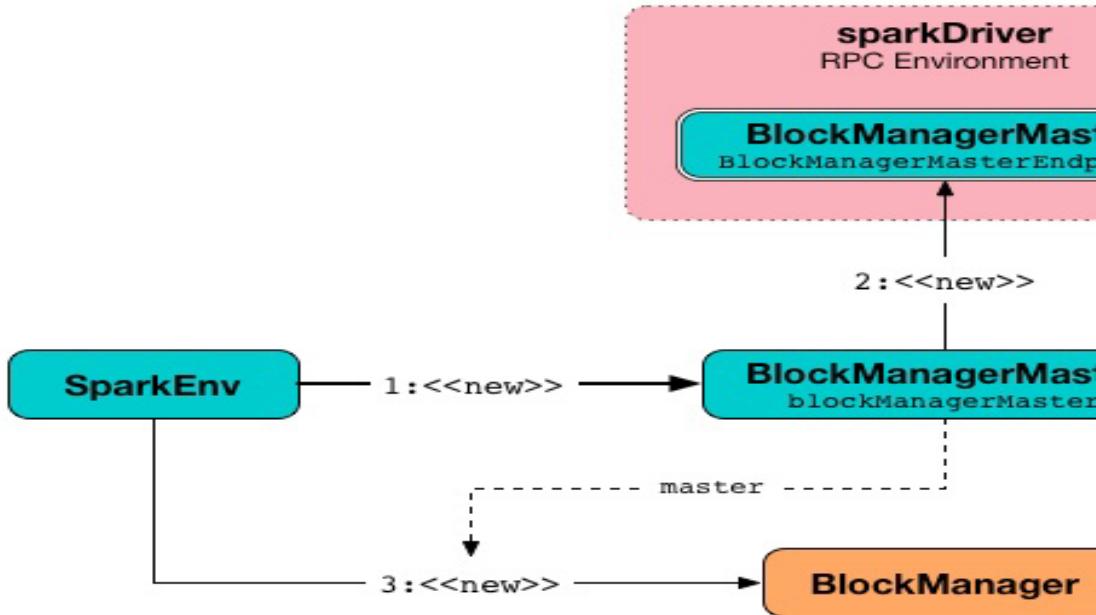
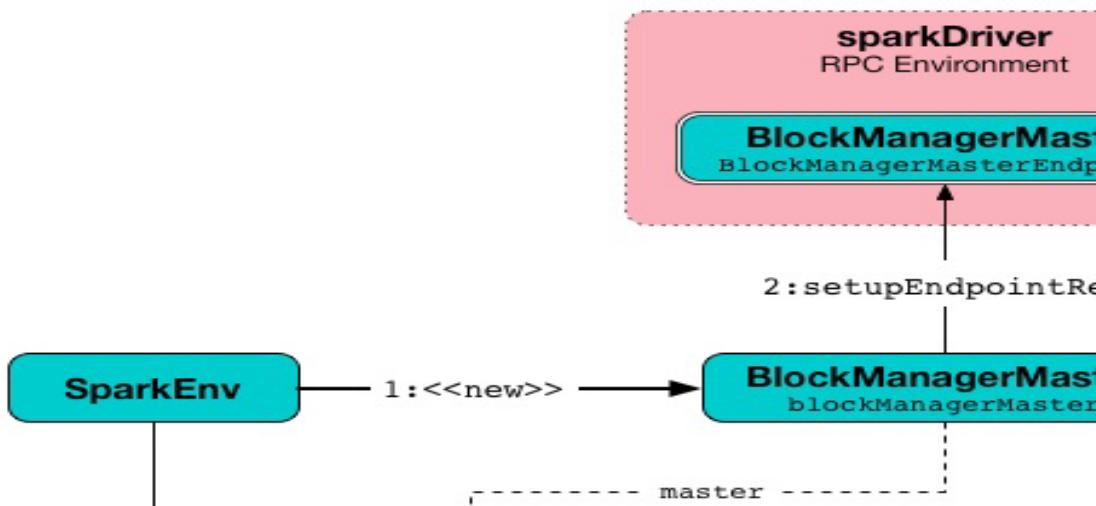


Figure 1. Creating BlockManager for the Driver

Note

`create` registers the **BlockManagerMaster** RPC endpoint for looks it up for executors.



SparkEnv — Spark Runtime Environment

`create` creates a `BroadcastManager`.

`create` creates a `MapOutputTrackerMaster` or `MapOutputTrackerWorker` executors, respectively.

Note The choice of the real implementation of `MapOutputTracker` is based on whether the input `executorId` is **driver** or not.

`create` registers or looks up `RpcEndpoint` as `MapOutputTracker`. It registers `MapOutputTrackerMasterEndpoint` on the driver and creates a RPC endpoint on executors. The RPC endpoint reference gets assigned as the `MapOutputTracker` endpoint.

Caution

FIXME

It creates a `CacheManager`.

It creates a `MetricsSystem` for a driver and a worker separately.

It initializes `userFiles` temporary directory used for downloading dependencies while this is the executor's current working directory for an executor.

An `OutputCommitCoordinator` is created.

Note `create` is called by `createDriverEnv` and `createExecutorEnv`.

Registering or Looking up RPC Endpoint by Name — `registerOrLookupEndpoint` Method

```
registerOrLookupEndpoint(name: String, endpointCreator: => RpcEndpoint)
```

`registerOrLookupEndpoint` registers or looks up a RPC endpoint by name.

If called from the driver, you should see the following INFO message in the logs:

```
INFO SparkEnv: Registering [name]
```

And the RPC endpoint is registered in the RPC environment.

SparkEnv — Spark Runtime Environment

```
createDriverEnv(  
    conf: SparkConf,  
    isLocal: Boolean,  
    listenerBus: LiveListenerBus,  
    numCores: Int,  
    mockOutputCommitCoordinator: Option[OutputCommitCoordinator] = None)
```

`createDriverEnv` creates a `SparkEnv` execution environment for the driver.

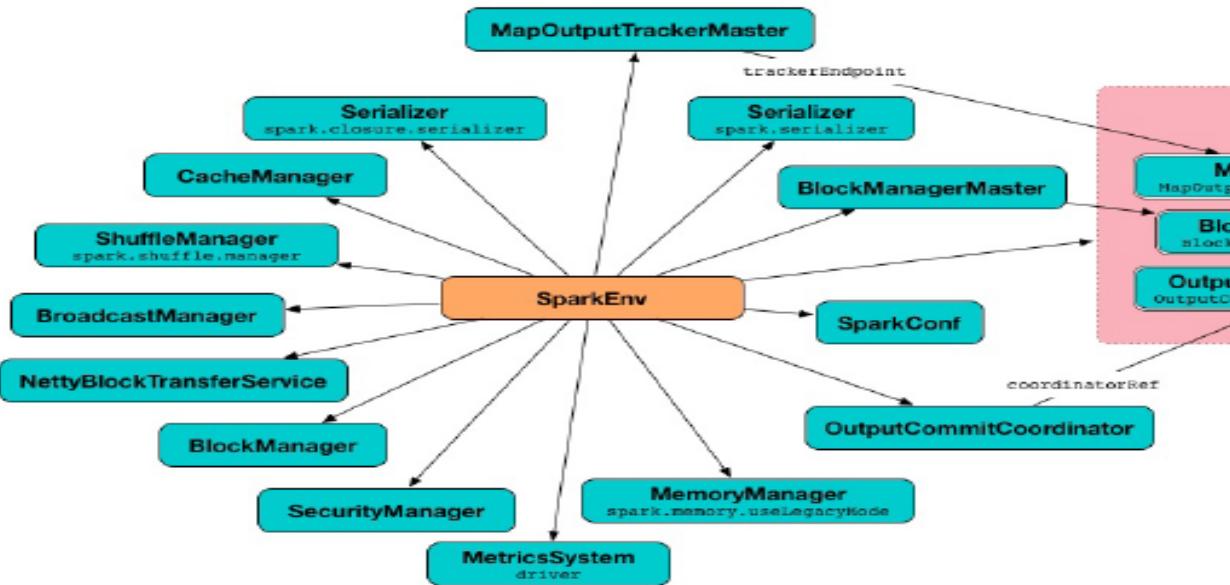


Figure 3. Spark Environment for driver

`createDriverEnv` accepts an instance of `SparkConf`, whether it runs in local mode or `LiveListenerBus`, the number of cores to use for execution in local mode or `isLocal`, and a `OutputCommitCoordinator` (default: none).

`createDriverEnv` ensures that `spark.driver.host` and `spark.driver.port` settings are set correctly.

It then passes the call straight on to the `create helper method` (with `isDriver` enabled, and the input parameters).

Note

`createDriverEnv` is exclusively used by `SparkContext` to create the `SparkEnv` (while a `SparkContext` is being created for the driver).

Creating SparkEnv for Executor — `createExecutorEnv`

SparkEnv — Spark Runtime Environment

```
createExecutorEnv(  
    conf: SparkConf,  
    executorId: String,  
    hostname: String,  
    port: Int,  
    numCores: Int,  
    ioEncryptionKey: Option[Array[Byte]],  
    isLocal: Boolean): SparkEnv
```

`createExecutorEnv` creates an executor's (execution) environment that execution environment for an executor.

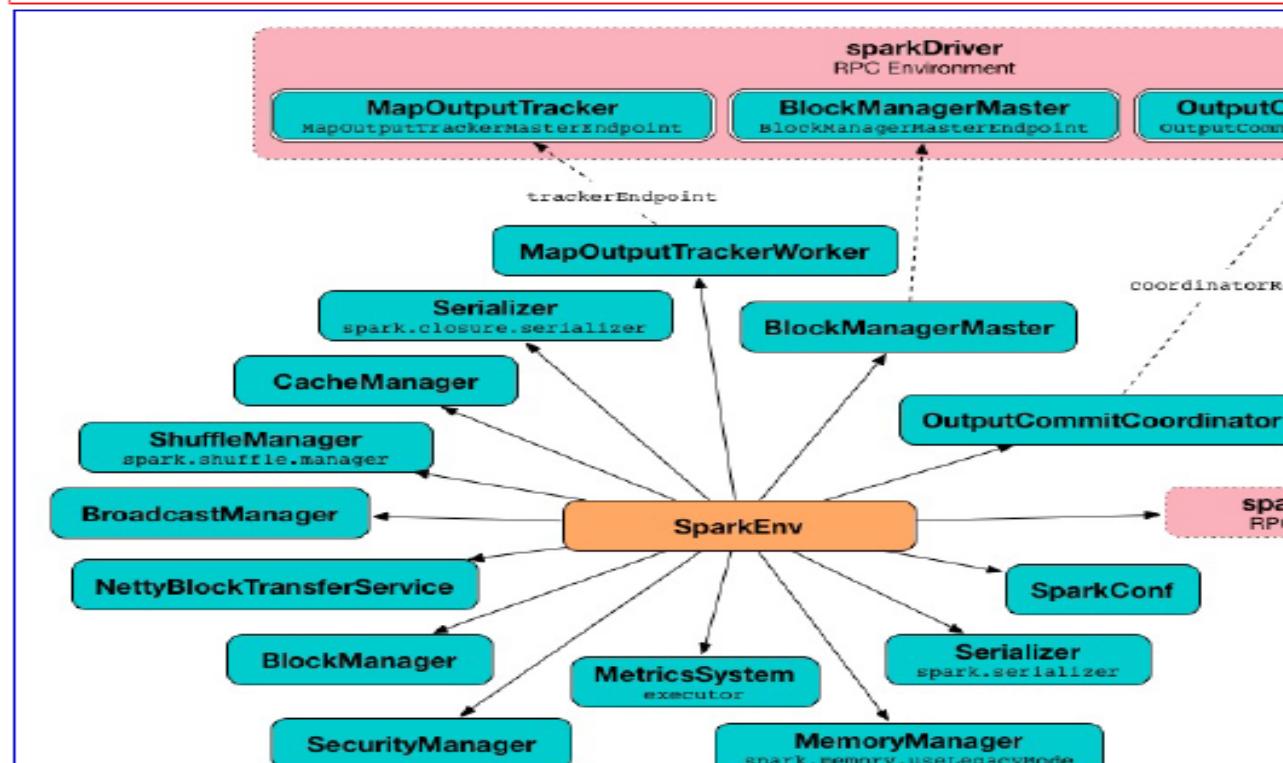


Figure 4. Spark Environment for executor

Note	<code>createExecutorEnv</code> is a <code>private[spark]</code> method.
------	---

`createExecutorEnv` simply creates the base `SparkEnv` (passing in all the info) and sets it as the current `SparkEnv`.

Note	The number of cores <code>numCores</code> is configured using <code>--cores</code> command-line option of <code>CoarseGrainedExecutorBackend</code> and is specific to a cluster.
------	---

Getting Current SparkEnv — `get` Method

```
get: SparkEnv
```

`get` returns the current `SparkEnv`:

```
import org.apache.spark._  
scala> SparkEnv.get  
res0: org.apache.spark.SparkEnv = org.apache.spark.SparkEnv@49322d04
```

Stopping SparkEnv — `stop` Method

Note

The introduction that follows was highly influenced by the Scala `org.apache.spark.scheduler.DAGScheduler`. As `DAGScheduler` class it does not appear in the official API documentation. You are encouraged to read the sources and only then read this and the afterwards.

Introduction

DAGScheduler is the scheduling layer of Apache Spark that implements **scheduling**. It transforms a **logical execution plan** (i.e. `RDD` lineage or `using RDD transformations`) to a **physical execution plan** (`using stages`)

```
(2) MapPartitionsRDD[62] at repartition at <console>:27 □  
| CoalescedRDD[61] at repartition at <console>:27 □  
| ShuffledRDD[60] at repartition at <console>:27 □
```

DAGScheduler — Stage-Oriented Scheduler

After an `action` has been called, `SparkContext` hands over a logical plan to that it in turn translates to a set of stages that are submitted as `TaskSets` (`ExecutionModel`).

DAGSche

Figure 2. Executing action leads to new ResultStage and ActiveJob in DAGScheduler. The fundamental concepts of `DAGScheduler` are **jobs** and **stages** (refer to `Job` and `Stage` respectively) that it tracks through `internal registries and counters`.

`DAGScheduler` works solely on the driver and is created as part of `SparkContext` initialization (right after `TaskScheduler` and `SchedulerBackend` are ready).

```
DAGSchedulerEvent  
eventProc
```

Figure 3. DAGScheduler as created by SparkContext with other

DAGScheduler does three things in Spark (thorough explanations follow):

- Computes an **execution DAG**, i.e. DAG of stages, for a job.
- Determines the **preferred locations** to run each task on.
- Handles failures due to **shuffle output files** being lost.

DAGScheduler computes a directed acyclic graph (DAG) of stages for each of which RDDs and stage outputs are materialized, and finds a minimal set. It then submits stages to TaskScheduler.

In addition to coming up with the execution DAG, DAGScheduler also determines preferred locations to run each task on, based on the current cache status information to TaskScheduler.

DAGScheduler tracks which RDDs are cached (or persisted) to avoid "recomputation", i.e. redoing the map side of a shuffle. DAGScheduler remembers what Shuffles have already produced output files (that are stored in BlockManagers).

DAGScheduler is only interested in cache location coordinates, i.e. host and partition of a RDD.

Caution

FIXME: A diagram, please

DAGScheduler — Stage-Oriented Scheduler

Furthermore, it handles failures due to shuffle output files being lost, in which stages may need to be resubmitted. Failures within a stage that are not caused by file loss are handled by the TaskScheduler itself, which will retry each task multiple times before cancelling the whole stage.

DAGScheduler uses an event queue architecture, in which a thread can read DAGSchedulerEvent events, e.g. a new job or stage being submitted, that reads and executes sequentially. See the section Internal Event Loop - data loop.

DAGScheduler runs stages in topological order.

Table 1. DAGScheduler's Internal Properties

Name	Initial Value	Description
	Refer to Logging.	

DAGScheduler uses SparkContext, TaskScheduler, LiveListenerBus, MapOutputTracker, and BlockManager for its services. However, at the very minimum, DAGScheduler only depends on SparkContext (and requests SparkContext for the other services).

DAGScheduler reports metrics about its execution (refer to the section Metrics).

Creating DAGScheduler Instance

DAGScheduler takes the following when created:

- `SparkContext`
- `TaskScheduler`
- `LiveListenerBus`
- `MapOutputTrackerMaster`
- `BlockManagerMaster`
- `SparkEnv`
- `Clock (defaults to SystemClock)`

DAGScheduler initializes the internal registries and counters.

DAGScheduler — Stage-Oriented Scheduler

DAGScheduler sets itself in the given TaskScheduler and in the end starts Event Bus.

Note

DAGScheduler can reference all the services through a single Service or without specifying explicit TaskScheduler.

LiveListenerBus Event Bus for SparkListenerEvents

executorHeartbeatReceived Method

```
executorHeartbeatReceived(  
    execId: String,  
    accumUpdates: Array[(Long, Int, Int, Seq[AccumulableInfo])],  
    blockManagerId: BlockManagerId): Boolean
```

executorHeartbeatReceived **posts a SparkListenerExecutorMetricsUpdate** and **informs BlockManagerMaster** that **blockManagerId** **block manager is BlockManagerHeartbeat**).

Note

executorHeartbeatReceived is called when TaskSchedulerImpl executorHeartbeatReceived .

Cleaning Up After ActiveJob and Independent S

cleanUpStateForJobAndIndependentSt

Note

`markMapStageJobAsFinished` is used in [handleMapStageSubmission](#)
[handleTaskCompletion](#).

Submitting Job — `submitJob` method

```
submitJob[T, U](  
    rdd: RDD[T],  
    func: (TaskContext, Iterator[T]) => U,  
    partitions: Seq[Int],  
    callSite: CallSite,  
    resultHandler: (Int, U) => Unit,  
    properties: Properties): JobWaiter[U]
```

`submitJob` creates a `JobWaiter` and posts a `JobSubmitted` event.

```
----- handleJobSubmit
```

Figure 4. DAGScheduler.submitJob

Internally, `submitJob` does the following:

1. Checks whether `partitions` reference available partitions of the input `rdd`.
2. Increments `nextJobId` internal job counter.
3. Returns a 0-task `JobWaiter` when the number of `partitions` is zero.
4. Posts a `JobSubmitted` event and returns a `JobWaiter`.

You may see a `IllegalArgumentException` thrown when the input `partitions` not in the input `rdd`:

```
Attempting to access a non-existent partition: [p]. Total number of partitions]
```

Note `submitJob` is called when `SparkContext` submits a job and DAGScheduler handles it.

Note `submitJob` assumes that the partitions of a RDD are indexed from 0 to `partitions - 1` in sequential order.

Submitting ShuffleDependency for Execution

`submitMaster` Method

Caution

FIXME

Running Job — `runJob` Method

```
runJob[T, U](  
    rdd: RDD[T],  
    func: (TaskContext, Iterator[T]) => U,  
    partitions: Seq[Int],  
    callSite: CallSite,  
    resultHandler: (Int, U) => Unit,  
    properties: Properties): Unit
```

`runJob` submits an action job to the `DAGScheduler` and waits for a result.

Internally, `runJob` executes `submitJob` and then waits until a result comes.

When the job succeeds, you should see the following INFO message in the logs:

```
INFO Job [jobId] failed: [callSite], took [time] s
```

Note

`runJob` is used when `SparkContext` runs a job.

Finding or Creating New ShuffleMapStages for a Partition

ShuffleDependency — getOrCreateShuffleMapStage Internal Method

```
getOrCreateShuffleMapStage(  
    shuffleDep: ShuffleDependency[_, _, _],  
    firstJobId: Int): ShuffleMapStage
```

`getOrCreateShuffleMapStage` finds or creates the `ShuffleMapStage` for the `ShuffleDependency`.

Internally, `getOrCreateShuffleMapStage` finds the `shuffleDependency` in `shuffledDependencies` internal registry and returns one when found.

If no `shuffleDependency` was available, `getOrCreateShuffleMapStage` finds `shuffle dependencies` and creates corresponding `shuffleMapStage` stages the input `shuffleDep`).

Note	All the new <code>shuffleMapStage</code> stages are associated with the input <code>shuffleDep</code> .
------	---

Note	<code>getOrCreateShuffleMapStage</code> is used when <code>DAGScheduler</code> finds missing direct parent <code>ShuffleMapStages</code> (for <code>ShuffleDependency</code> in <code>RDD</code>), <code>getMissingParentStages</code> (for <code>ShuffleDependencies</code>), is run when <code>ShuffleDependency</code> was submitted, and checks if a stage depends on it.
------	---

Creating ShuffleMapStage for ShuffleDependency — `createShuffleMapStage` Method

```
createShuffleMapStage(  
    shuffleDep: ShuffleDependency[_, _, _],  
    jobId: Int): ShuffleMapStage
```

`createShuffleMapStage` creates a `ShuffleMapStage` for the input `ShuffleDependency`. It uses `jobId` (of a `ActiveJob`) possibly copying shuffle map output locations from previous job to avoid recomputing records.

DAGScheduler — Stage-Oriented Scheduler

Note

When a `ShuffleMapStage` is created, the `id` is generated (using internal counter), `rdd` is from `ShuffleDependency`, `numTasks` is partitions in the RDD, all parents are looked up (and possibly jobId is given, callSite is the creationSite of the RDD, and the input `ShuffleDependency`).

Internally, `createShuffleMapStage` first finds or creates missing parent stages of the associated RDD.

Note

`ShuffleDependency` is associated with exactly one `RDD[Product`

`createShuffleMapStage` creates a `ShuffleMapStage` (with the stage id from internal counter).

Note

The RDD of the new `shuffleMapStage` is from the input `Shuffle`

`createShuffleMapStage` registers the `shuffleMapStage` in `stageIdToStage` `shuffleIdToMapStage` internal registries.

`createShuffleMapStage` calls `updateJobIdStageIdMaps`.

If `MapOutputTrackerMaster` tracks the input `ShuffleDependency` (because it already computed it), `createShuffleMapStage` requests the serialized shuffle outputs, deserializes them and registers with the new `ShuffleMapStage`.

Note

`MapOutputTrackerMaster` was defined when `DAGScheduler` wa

Driver

```
createShuffleMapStage returns the new ShuffleMapStage .
```

Note	createShuffleMapStage is executed only when DAGScheduler finds parent ShuffleMapStage stages for a ShuffleDependency .
------	--

Clearing Cache of RDD Block Locations

`clearCacheLocations` Internal Method

Caution

FIXME

Allocating Memory Block for Tungsten Consumers — `allocatePage` Method

```
MemoryBlock allocatePage(long size, MemoryConsumer consumer)
```

Note	It only handles Tungsten Consumers , i.e. <code>MemoryConsumers tungstenMemoryMode</code> mode.
------	--

`allocatePage` allocates a block of memory (aka *page*) smaller than `MAXIMUM_PAGE_SIZE_BYTES` maximum size.

It checks `size` against the internal `MAXIMUM_PAGE_SIZE_BYTES` maximum size, if `size` is larger than the maximum size, the following `IllegalArgumentException` is thrown:

Solution

Now we know the available endpoints in the environment, let's move on to the next exercise [Developing RPC Environment](#).

RPC Environment (aka `RpcEnv`) is an environment for `RpcEndpoints` to handle messages. A RPC Environment manages the entire lifecycle of `RpcEndpoints`.

- registers (sets up) endpoints (by name or uri)
- routes incoming messages to them
- stops them

A RPC Environment is defined by the **name**, **host**, and **port**. It can also be configured with a **security manager**.

You can create a RPC Environment using `RpcEnv.create` factory methods.

The only implementation of RPC Environment is **Netty-based implementation**.

Tip

Read up [RpcEnv — RPC Environment](#) on the concept of RPC Environment in Spark.

The class `org.apache.spark.rpc.netty.NettyRpcEnv` is the implementation of Netty - "an asynchronous event-driven network application framework for building maintainable, high performance protocol servers & clients".

You can use [sbt](#) or [Maven](#) as the build command.

Using sbt as the build tool

The build command with sbt as the build tool is as follows:

```
./build/sbt -Phadoop-2.7,yarn,mesos,hive,hive-thriftserver -DskipTests
```

Using Java 8 to build Spark using sbt takes ca 10 minutes.

Starting standalone Master and workers on Windows

Windows 7 users can use [spark-class](#) to start [Spark Standalone](#) as there are scripts for the Windows platform.

```
$ ./bin/spark-class org.apache.spark.deploy.master.Master -h localhost
```

```
$ ./bin/spark-class org.apache.spark.deploy.worker.Worker spark://localhost:7077
```