Apache spark sizing

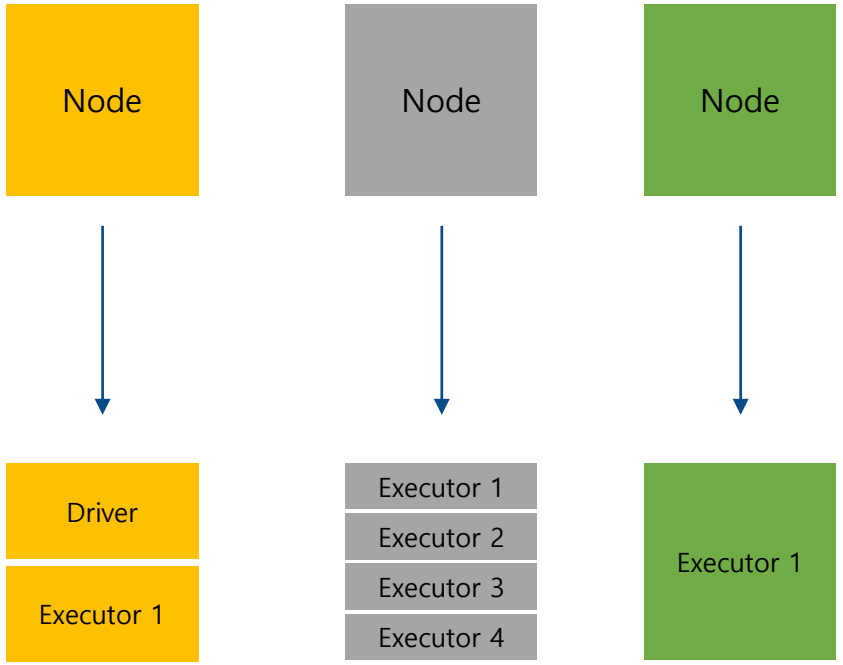
분석환경에 따른 Executors, cores, memories 설정 가이드

데이터엔지니어스랩

2022.12

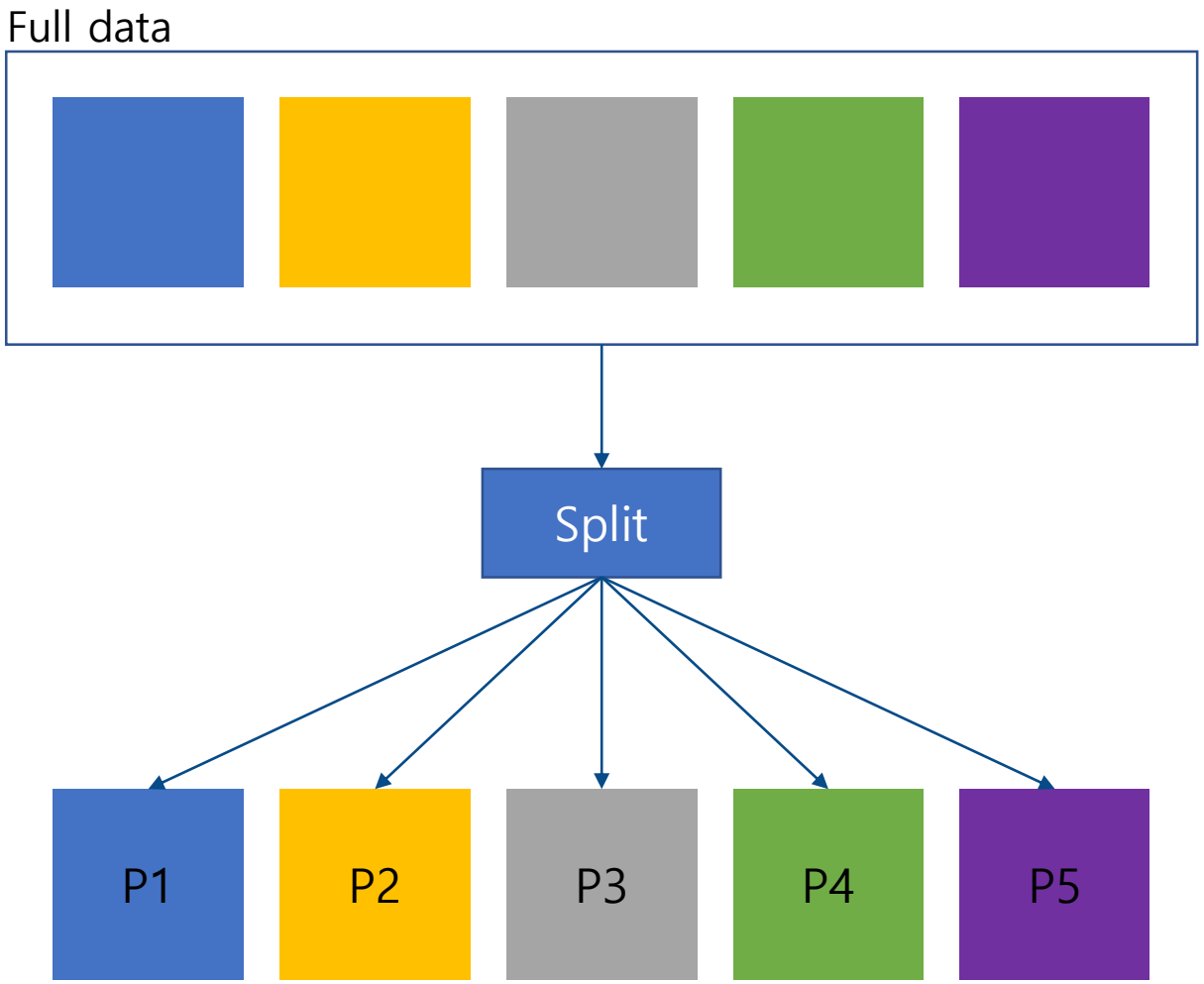
이진호

Node 란?



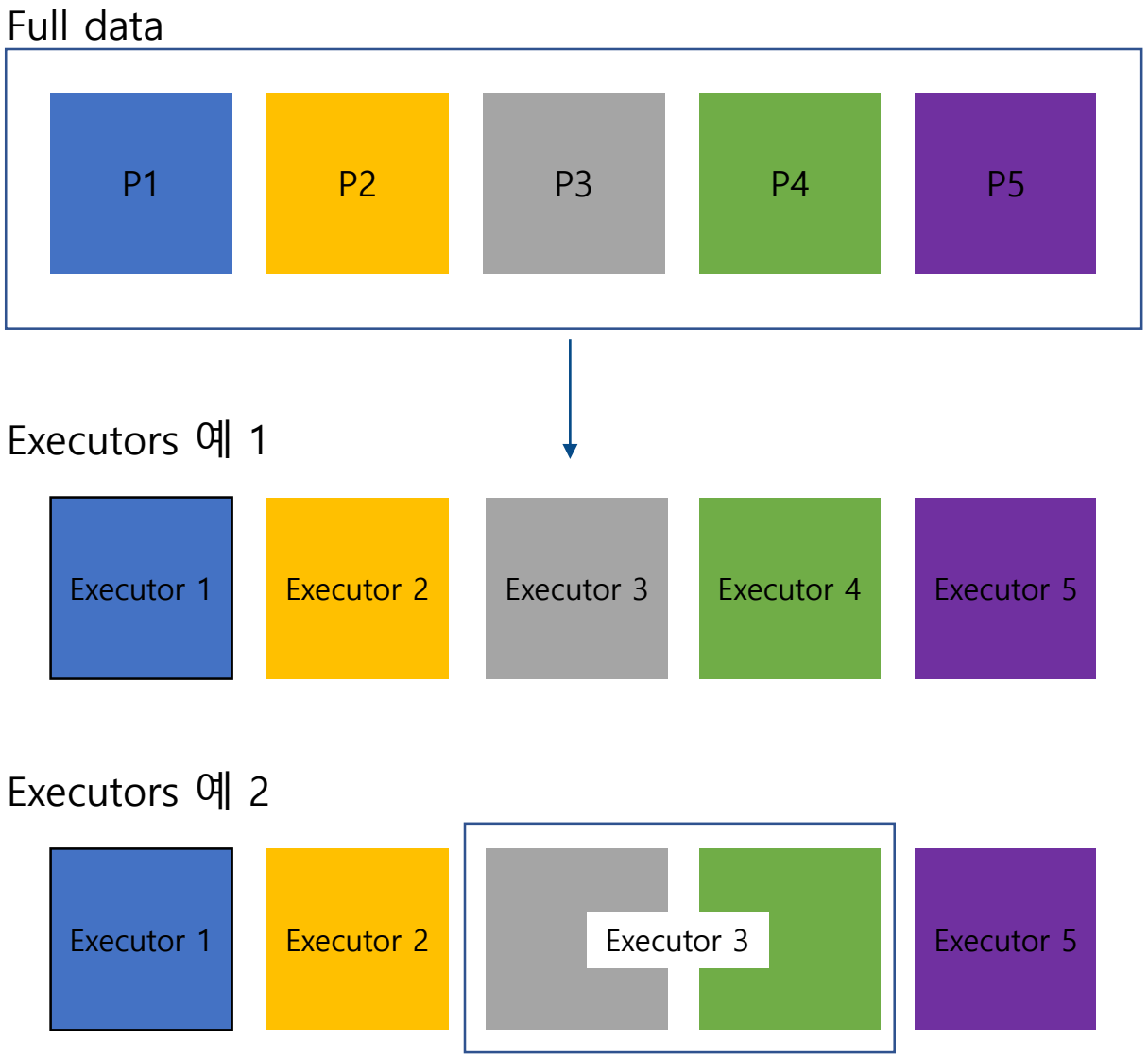
- Node 는 일반적으로 Spark 어플리케이션을 돌리기 위한 컴퓨터를 뜻함
- Master node(Driver), Worker node(executor) 라고도 함
- 1개의 node는 1개의 executor 혹은 여러 개의 executors 를 가짐

Spark Partitions



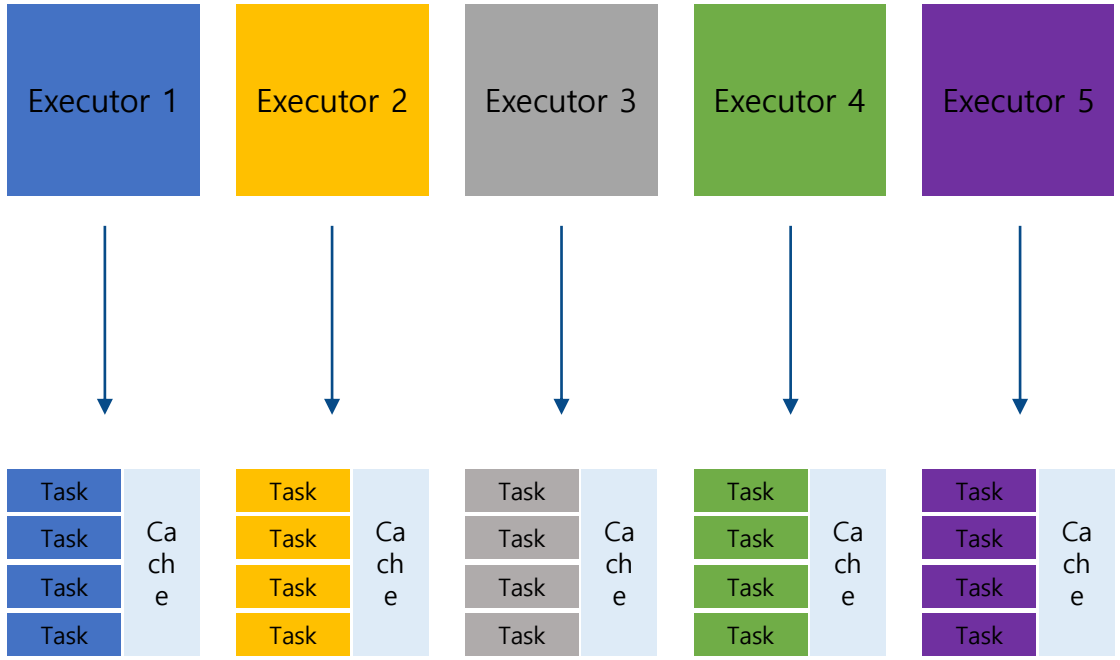
- Spark 는 데이터를 파티션 단위로 분할해서 병렬 처리함
- 그림 에서는 Full data 를 5개의 partitions 로 분할한 모습
- 관련 설정: spark.default.parallelism
  이값은 spark 실행 환경에 따라 변경됨

Spark Executors



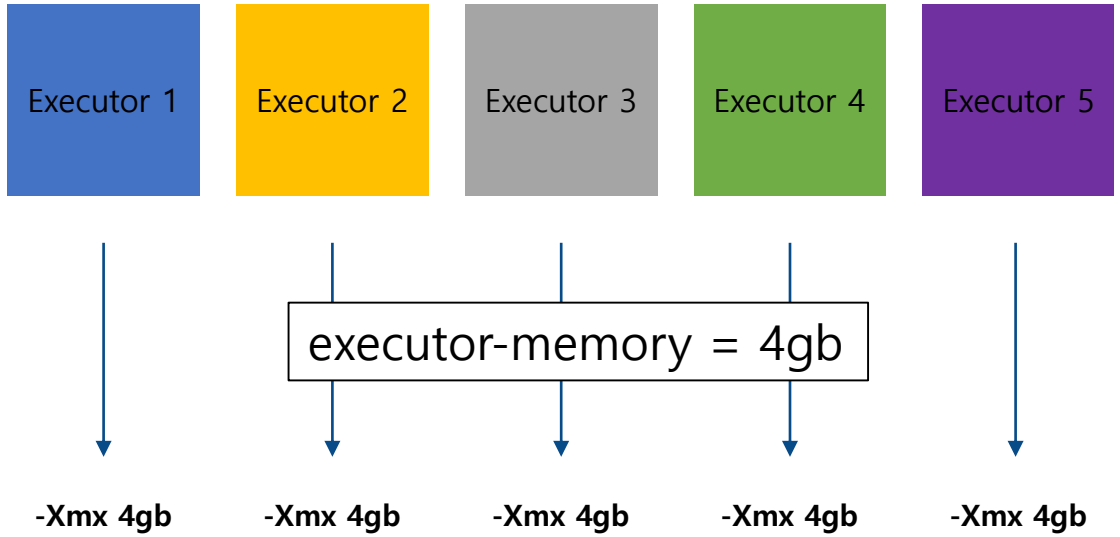
- Executor는 파티션 데이터를 처리하는 virtual machine 을 의미
- 1 Executor = 1 JVM (Java Virtual Machine)
- 각각의 파티션을 Executor 에서 동시 처리함
- 예2 처럼 하나의 executor 가 여러 개의 파티션 처리도 가능
- 단, 1개의 파티션을 여러 개의 Executor 에서 처리는 불가능

Executor cores = Spark tasks



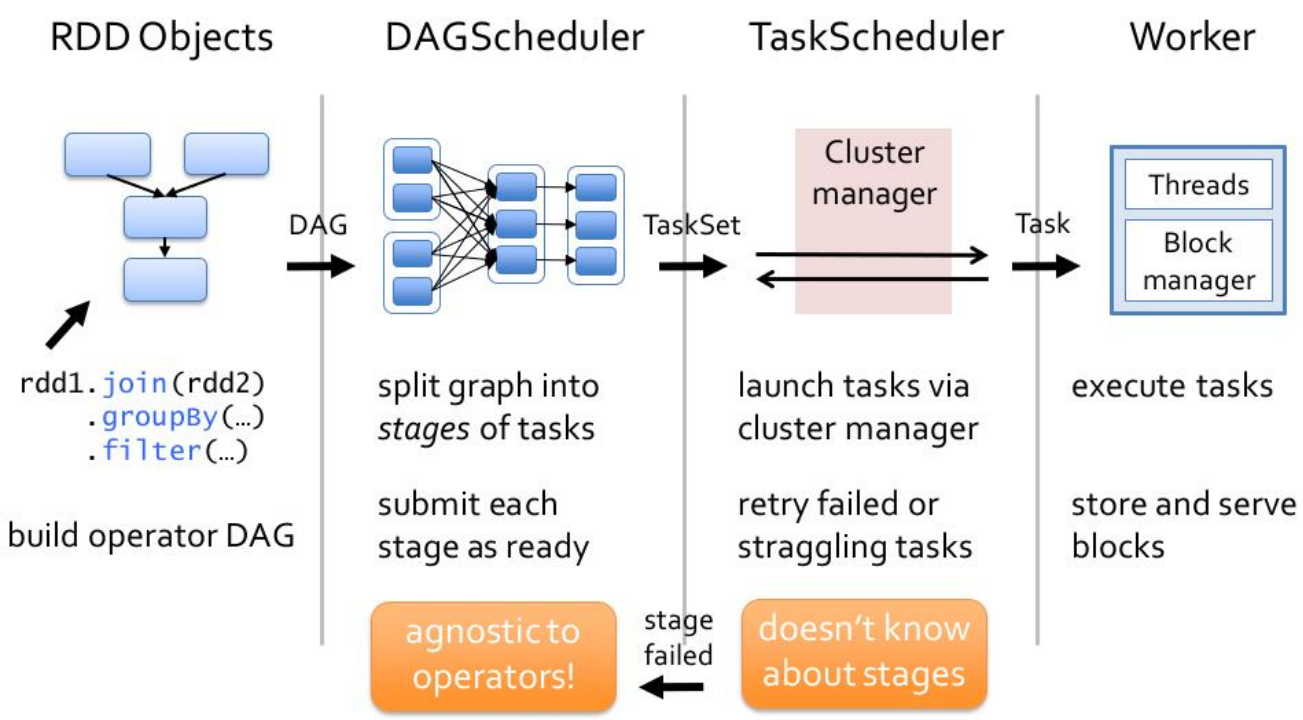
- Core 수는 일반적으로 1개의 executor 에서 동시처리 가능한 spark tasks 의 수
- executor-cores = 4 이면 1개의 Executor 에서 처리하는 동시 task는 4개를 의미
- Spark Job 실행시 모든 executor 가 동일한 cores 수를 가진다 (config 설정이나 submit 시 파라미터로 설정)
- 각 executor 는 tasks 와 cache 영역을 가짐

Executor memory



- 1개의 executor 에서 가지는 Max JVM heap memory 를 뜻함
- executor-memory = 4GB 이면 각 executor 의 heap memory 는 4gb 로 세팅됨
- JVM 이므로 GC overhead 영향을 받는다.
- 일반적으로 Node 에 속한 executors 메모리 합계를 node의 60% ~ 70% 메모리를 권장.
- JVM 이므로 memory 세팅 golden rule 을 따름. 각 executor memory 값은 max 30GB 를 넘기지 않는다. (초과시 GC overhead hell 을 만날 수 있다)
- ref: https://www.elastic.co/guide/en/elasticsearch/reference/current/advanced-configuration.html#set-jvm-heap-size

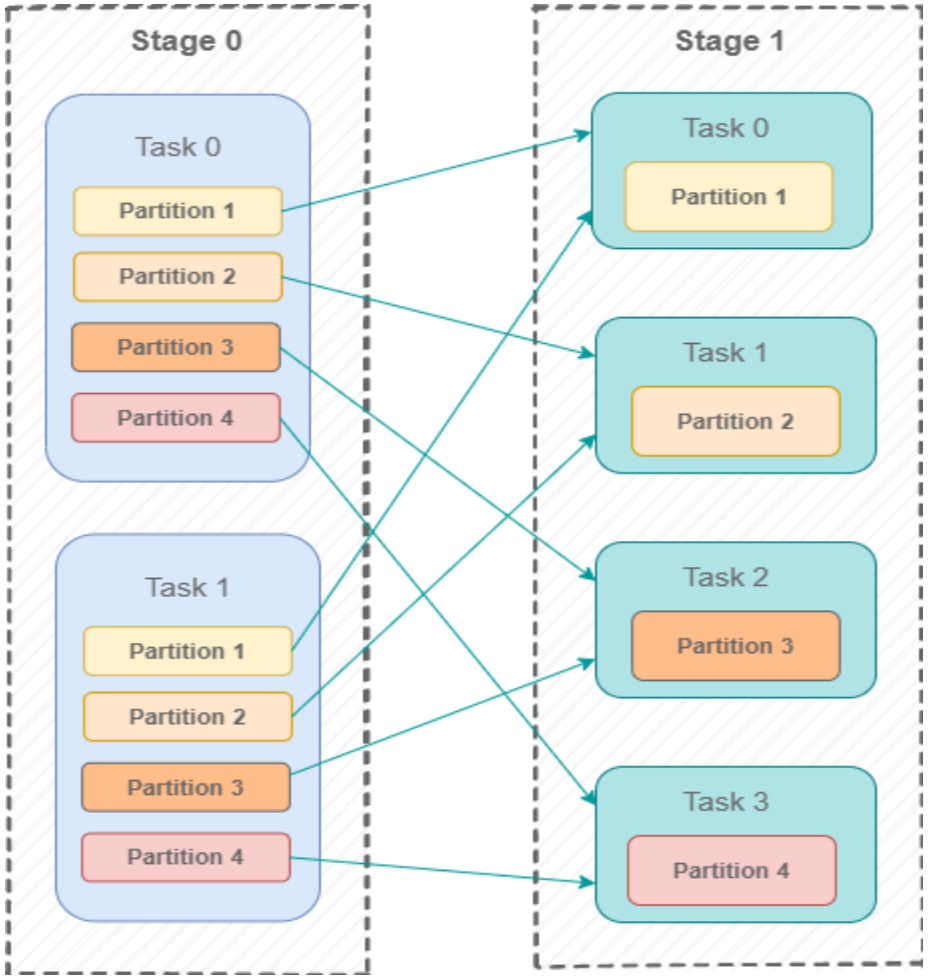
DAG 및 Stage, tasks



각 Tasks 들을 Stage 단위로 구조화 하여 DAG 형식으로 스케쥴링하여 실행

- 앞서 정해진 여러 개의 tasks 를 Spark executors 에서 절차적으로 수행하기 위해 논리적 단계인 Stage 로 그룹핑
- 각 Stage 를 DAG(Directed Acyclic Graph) 이용, 전체 tasks 을 스케쥴링 한다.
- 작성된 DAG 는 클러스터 매니저(일반적으로 마스터) 에게 전송됨
- 클러스터 매니저는 DAG 스케쥴링에 따라 각 Stages 를 executors 에게 실행 명령

Partiton shuffles

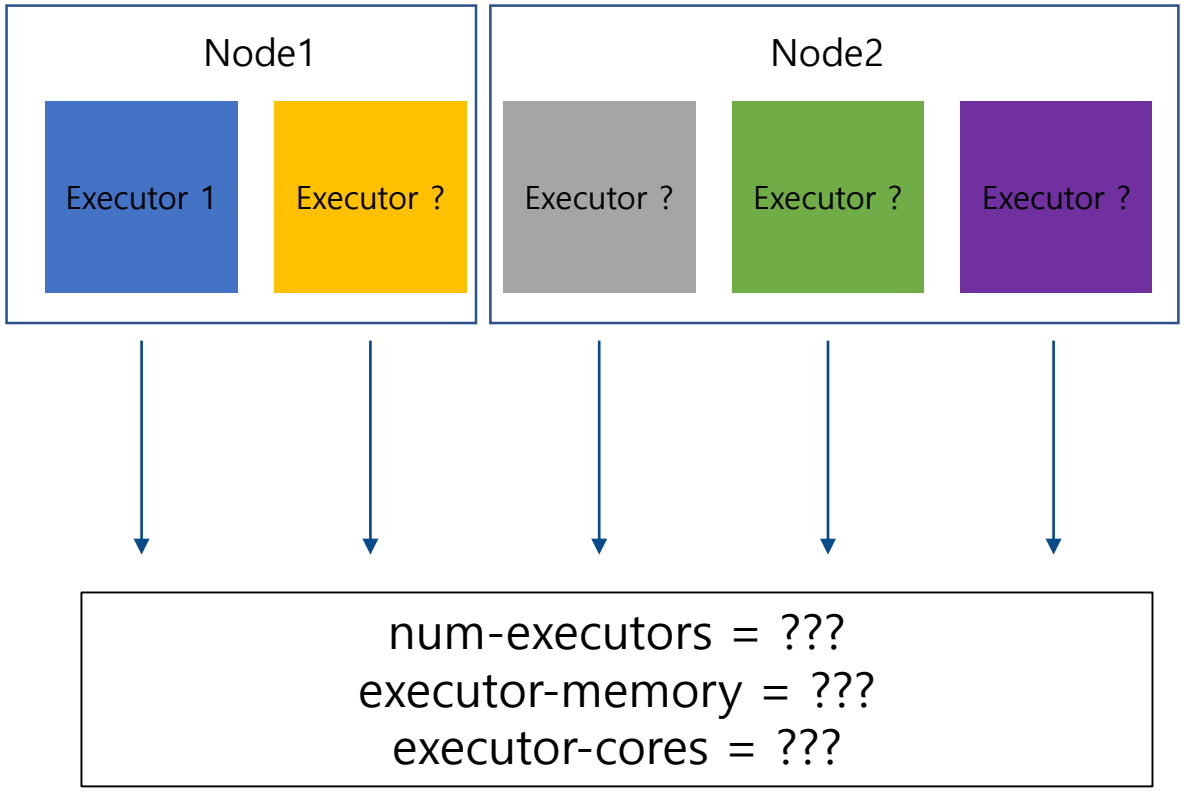


각 executor 의 stage 에서 파티션 셔플 job 수행

(https://medium.com/bild-journal/a-hitchhikers-guide-to-spark-s-aqe-exploring-dynamically-coalescing-shuffle-partitions-8d4913015050)

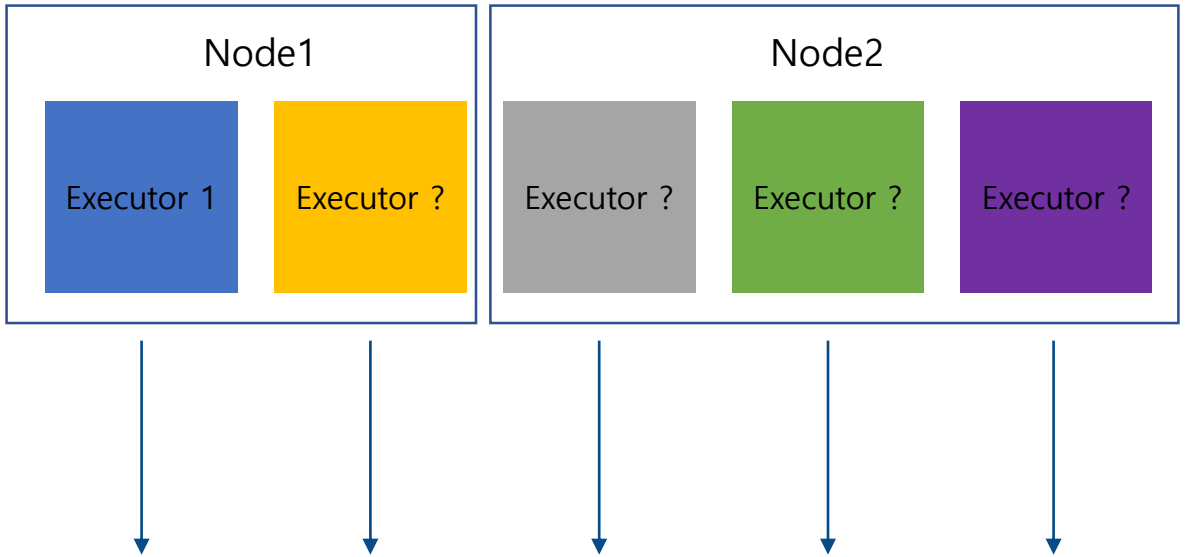
- 파티션의 merge, sort, collect, count 등의 크로스 파티션 연산을 수행할때 일어남
- Spark 연산에서 일반적으로 가장 많은 costs 가 발생하는 구간.
- Shuffle 에 필요한 자체 연산도 하지만, Executors 들끼리 데이터를 주고 받으니 네트워크 직렬화가 필연적으로 발생 (serialization/deserialization)
- 파티셔닝 셔플을 얼마나 효율적으로 하느냐에 따라 성능 격차 존재.

Deep dives into spark sizing (1/3)



- 모든 환경에 최적화된 설정은 없음. 아래의 가이드 기준에 따라 늘리고 줄여가며 최적 값을 찾아야 합니다.
- 분석하고자 하는 데이터 용량이 1TB 라고 해서 total executors memory 를 1TB 로 할 필요는 없다. 어차피 파티셔닝 할때 필요한 부분만 로드 하고 나머지는 다음 stage 를 위해 남겨둠 (Lazy)
- 같은 노드에 있는 executors 들끼리도 데이터를 주고 받는데 비용이 발생함. (직렬/역직렬화)
- 따라서 executor 메모리를 너무 적게 주면 OOM 에러가 생길수 있음. 자원이 허용된다면 executor-memory 는 최소 4GB 이상을 권장.
- Executors 개수는 분석 환경의 vCpu cores 개수에 따라 유동적으로 세팅

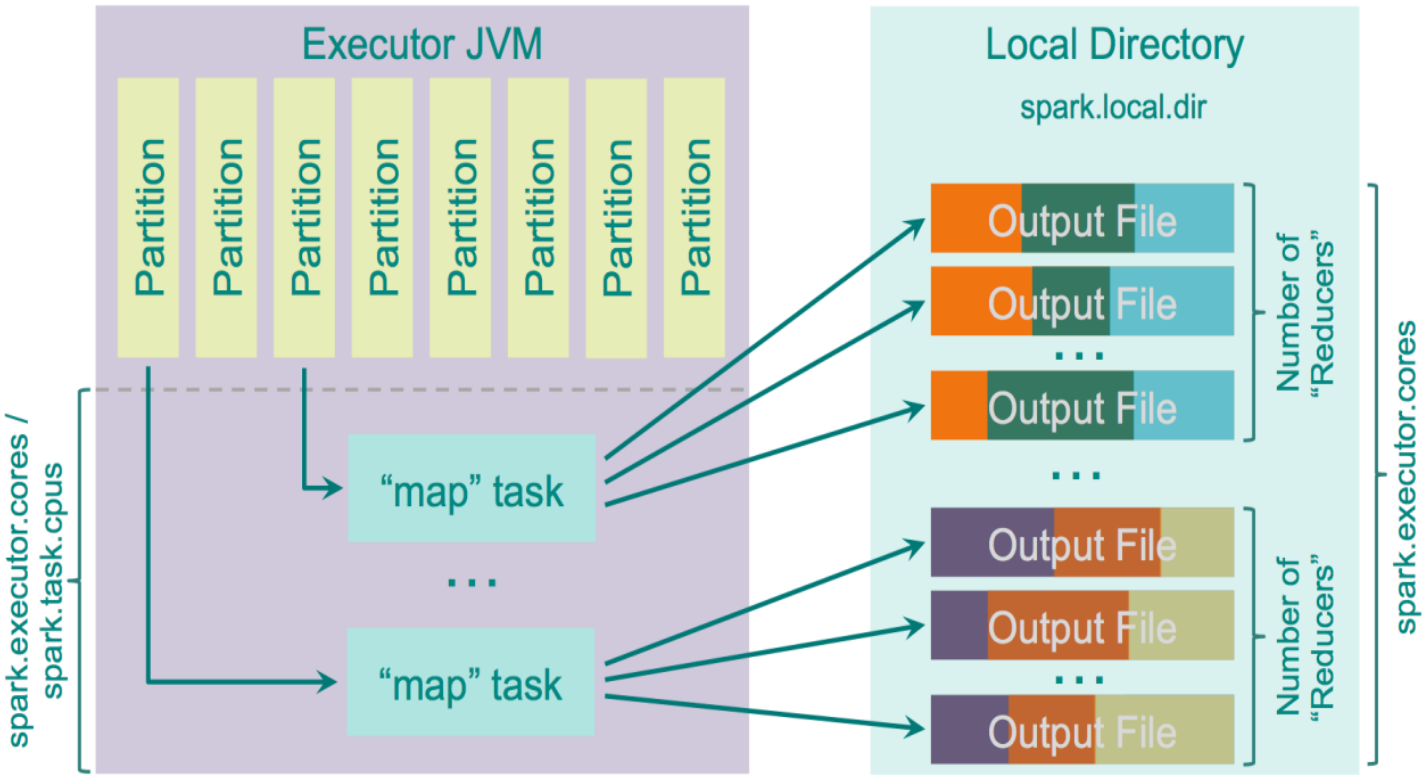
Deep dives into spark sizing (2/3)



num-executors = **total cpu cores 에 종속적**
executor-memory = **4GB 이상 30GB 이하**
executor-cores = **4~5 개 권장**

- 예: 사용 가능한 Total cpu cores 가 96 이면 tasks 를 4개로 분할하면 executors 개수가 됨.
- Total vCpu 중 일부는 Cluster manager, Master 나 Driver 노드를 위해 남겨둔다.
- 96 cores – 4 = (92 / 4) = (**23 executors**)
- 여러 벤치마크 결과를 토대로 하나의 executor 에서 5개 이상 cores 를 사용하는 것은 성능향상에 크게 도움이 안된다.
- Advanced analytics with Spark 의 저자인 Sandy Ryza 는 executor 당 5개의 cores 를 최대치로 봐야 한다고 제안. (https://blog.cloudera.com/how-to-tune-your-apache-spark-jobs-part-2/)

Deep dives into spark sizing (3/3)



대표적 transform 인 map 단계에서의 파티셔닝 처리 로직

- Spark 성능향상을 이야기할때 파티션 튜닝을 빼놓을수 없음.
- Spark 파티션 튜닝 = 파티션수 조정을 의미. (spark.default.parallelism 을 조정하거나 repartition, coalesce 함수로 변경)
- Spark transformation 은 narrow 방식과 wide 방식으로 나뉘는데 그중 wide 방식을 쓸때 명시적으로 파티션수 조정이 가능
- 기본값인 spark.default.parallelism 은 [executors x cores] 값으로 지정되는데 이값은 최적값이 아니고 최소값
- total cpu가 96인데 default.parallelism 이 92로 되어있다면? 4개의 cpu 는 놀게된다 (state idle)
- default.parallelism 보다 조금씩 늘려가며 최적값을 찾는게 일반적으로 성능향상에 도움이 됨.

End of documents