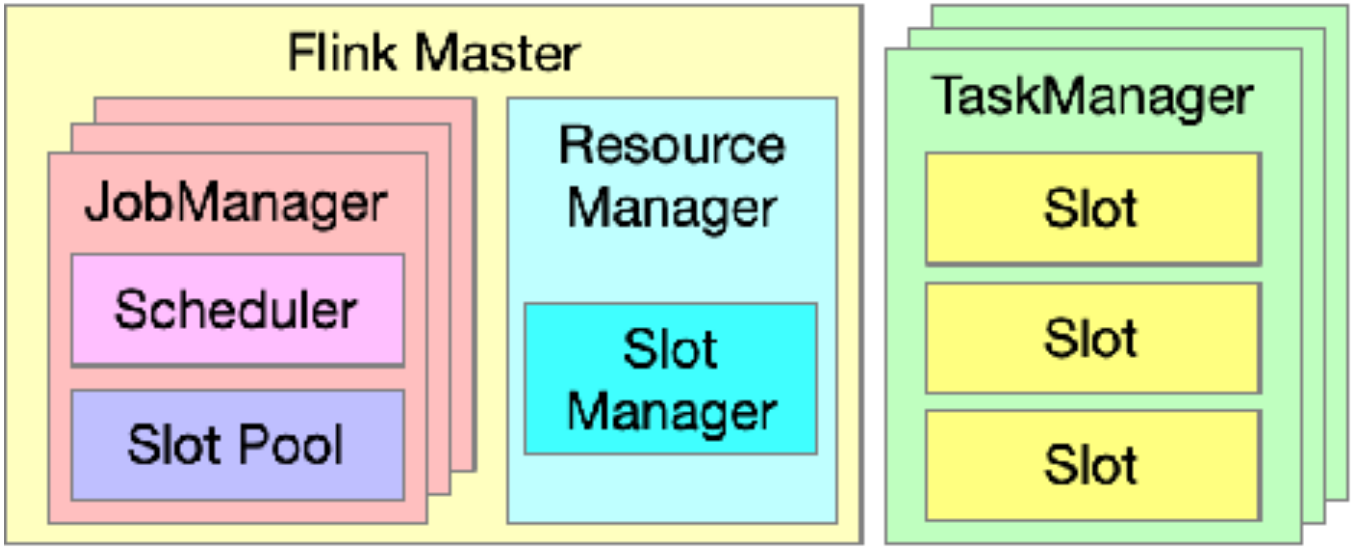
深入解读 Flink 资源管理机制

发布于: 2020 年 06 月 18 日



简介： 文章主要从基本概念、当前机制与策略、未来发展方向等三个方面帮助开发者深入理解 Flink 的资源管理机制。

作者：宋辛童（五藏）

整理：王文杰（Flink 社区志愿者）

摘要：本文根据 Apache Flink 系列直播整理而成，由阿里巴巴高级开发工程师宋辛童分享。文章主要从基本概念、当前机制与策略、未来发展方向等三个方面帮助开发者深入理解 Flink 的资源管理机制。

基本概念

当前机制与策略

未来发展方向

Tips：点击「下方链接」可查看更多数仓系列视频～

https://ververica.cn/developers/flink-training-course-data-warehouse/

1. 基本概念

1.1 相关组件

我们今天介绍的主要是与 Flink 资源管理相关的组件，我们知道一个 Flink Cluster 是由一个 Flink Master 和多个 Task Manager 组成的，Flink Master 和 Task Manager 是进程级组件，其他的组件都是进程内的组件。

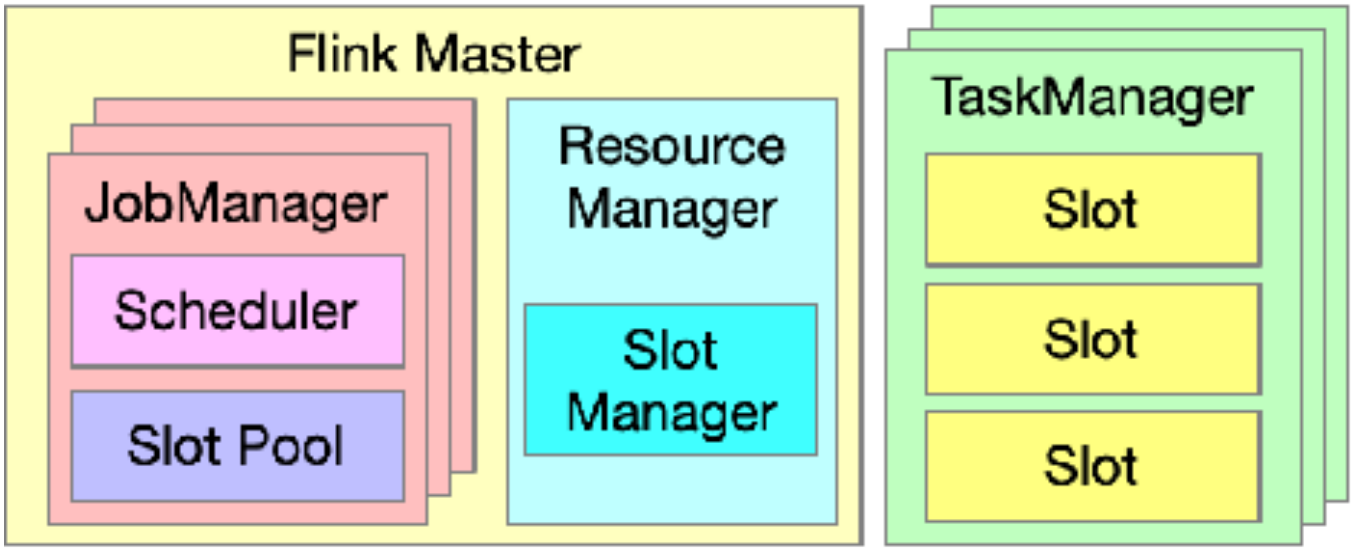


图1. Flink 资源管理相关组件.png

图1. Flink 资源管理相关组件

如图1所示，一个 Flink Master 中有一个 Resource Manager 和多个 Job Manager ，Flink Master 中每一个 Job Manager 单独管理一个具体的 Job ，Job Manager 中的 Scheduler 组件负责调度执行该 Job 的 DAG 中所有 Task ，发出资源请求，即整个资源调度的起点；JobManager 中的 Slot Pool 组件持有分配到该 Job 的所有资源。另外，Flink Master 中唯一的 Resource Manager 负责整个 Flink Cluster 的资源调度以及与外部调度系统对接，这里的外部调度系统指的是 Kubernetes、Mesos、Yarn 等资源管理系统。

Task Manager 负责 Task 的执行，其中的 Slot 是 Task Manager 资源的一个子集，也是 Flink 资源管理的基本单位，Slot 的概念贯穿资源调度过程的始终。

1.2 逻辑层级

介绍完相关组件，我们需要了解一下这些组件之间的逻辑关系，共分如下为4层。

Operator

算子是最基本的数据处理单元

Task

Flink Runtime 中真正去进行调度的最小单位

由一系列算子链式组合而成（chained operators）

（Note：如果两个 Operator 属于同一个 Task，那么不会出现一个 Operator 已经开始运行另一个 Operator 还没被调度的情况。）

Job

对应一个 Job Graph

Flink Cluster

1 Flink Master + N Task Managers

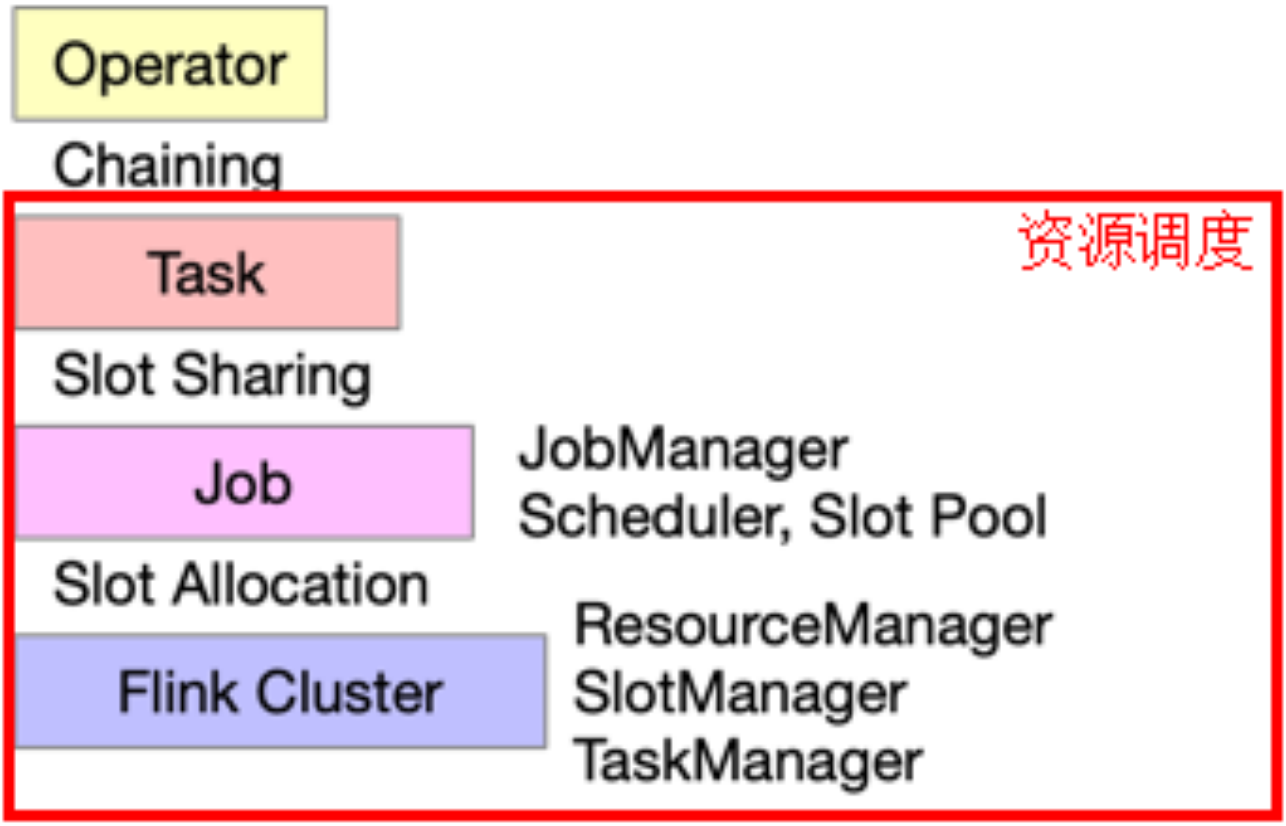


图2. 组件的逻辑层级.png

图2. 组件的逻辑层级

资源调度的范畴，实际上是图2红框内的内容。刚刚介绍的与资源调度相关的组件中，JobManager、Secheduler 和 Slot Pool 对应于 Job 级别，Resource Manager、Slot Manager 和 Task Manager 对应于 Flink Cluster 级别。

在 Operator 和 Task 中间的 Chaining 是指如何用 Operator 组成 Task 。在 Task 和 Job 之间的 Slot Sharing 是指多个 Task 如何共享一个 Slot 资源，这种情况不会发生在跨作业的情况中。在 Flink Cluster 和 Job 之间的 Slot Allocation 是指 Flink Cluster 中的 Slot 是怎样分配给不同的 Job 。

1.3 两层资源调度模型

Flink 的资源调度是一个经典的两层模型，其中从 Cluster 到 Job 的分配过程是由 Slot Manager 来完成，Job 内部分配给 Task 资源的过程则是由 Scheduler 来完成。如图3，Scheduler 向 Slot Pool 发出 Slot Request（资源请求），Slot Pool 如果不能满足该资源需求则会进一步请求 Resource Manager，具体来满足该请求的组件是 Slot Manager。

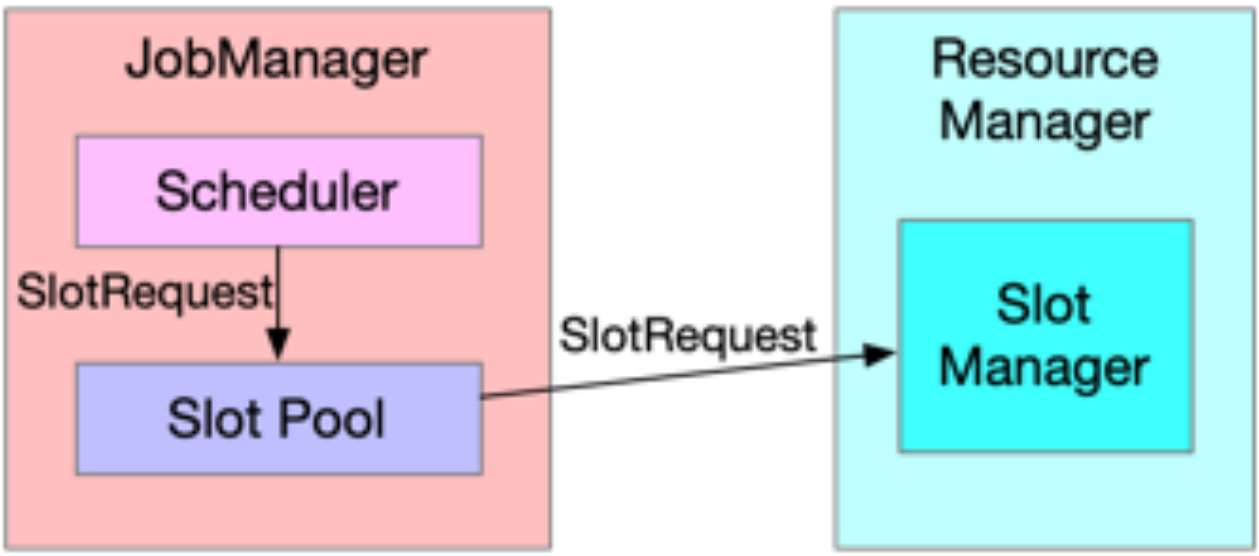


图3. 两层资源调度模型.png

图3. 两层资源调度模型

Task 对 Slot 进行复用有两种方式：

Slot Caching

批作业

流作业的 Failover

多个 task 先后/轮流使用 slot 资源

Slot Sharing

多个 Task 在满足一定条件下可同时共享同一个 Slot 资源

2. 当前机制与策略

截至 Flink 1.10 版本，Flink 当前的资源管理机制与策略是怎样的？以下将详细说明。

2.1 Task Manager 有哪些资源?

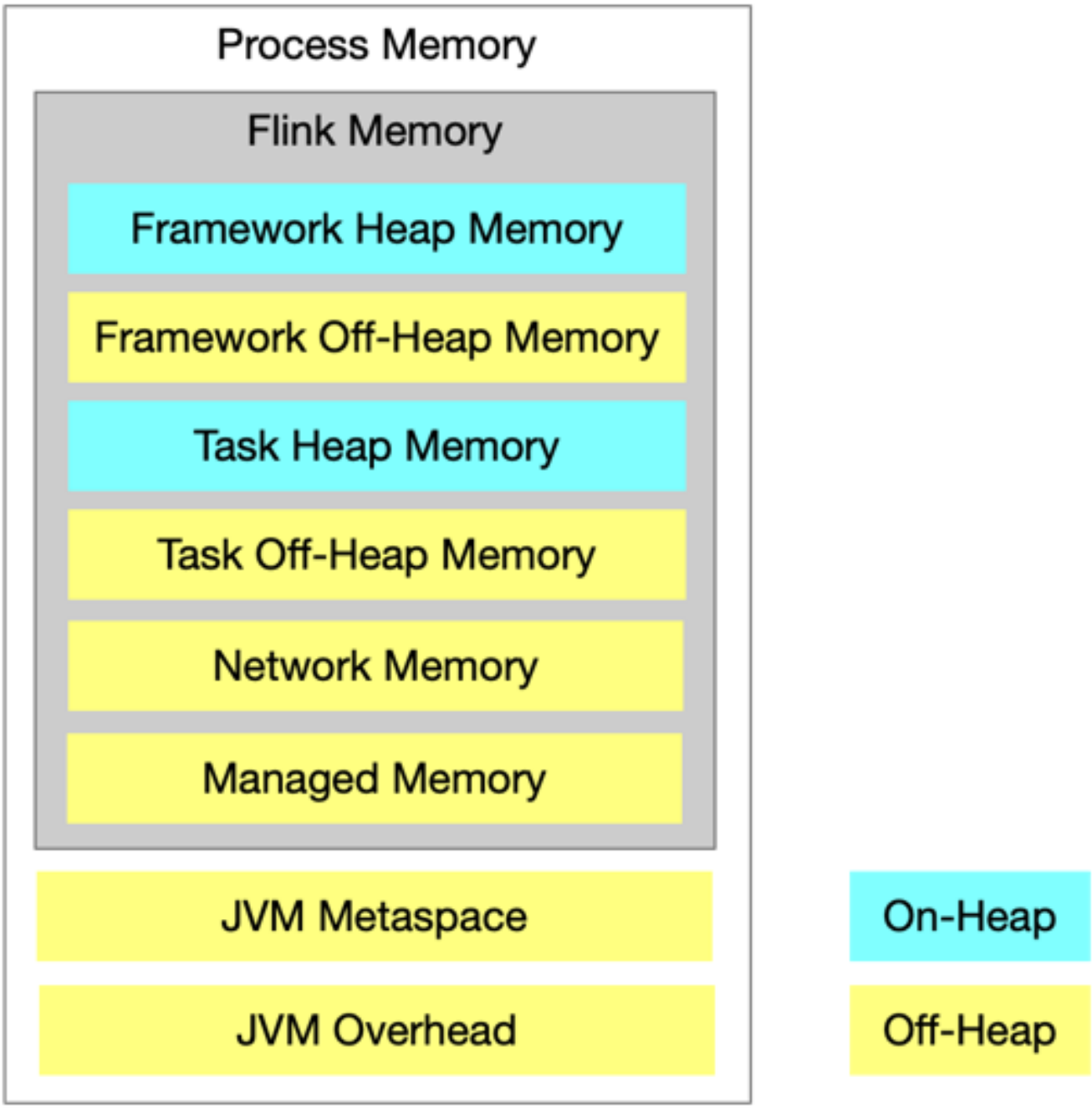


图4. Task Manager 资源组成.png

图4. Task Manager 资源组成

资源类型

内存

CPU

其他扩展资源

GPU（FLIP–108，在 Flink 1.11 版本完成）

TM 资源由配置决定

Standalone 部署模式下，TM 资源可能不同

其他部署模式下，所有 TM 资源均相同

2.2 Slot 有哪些资源?

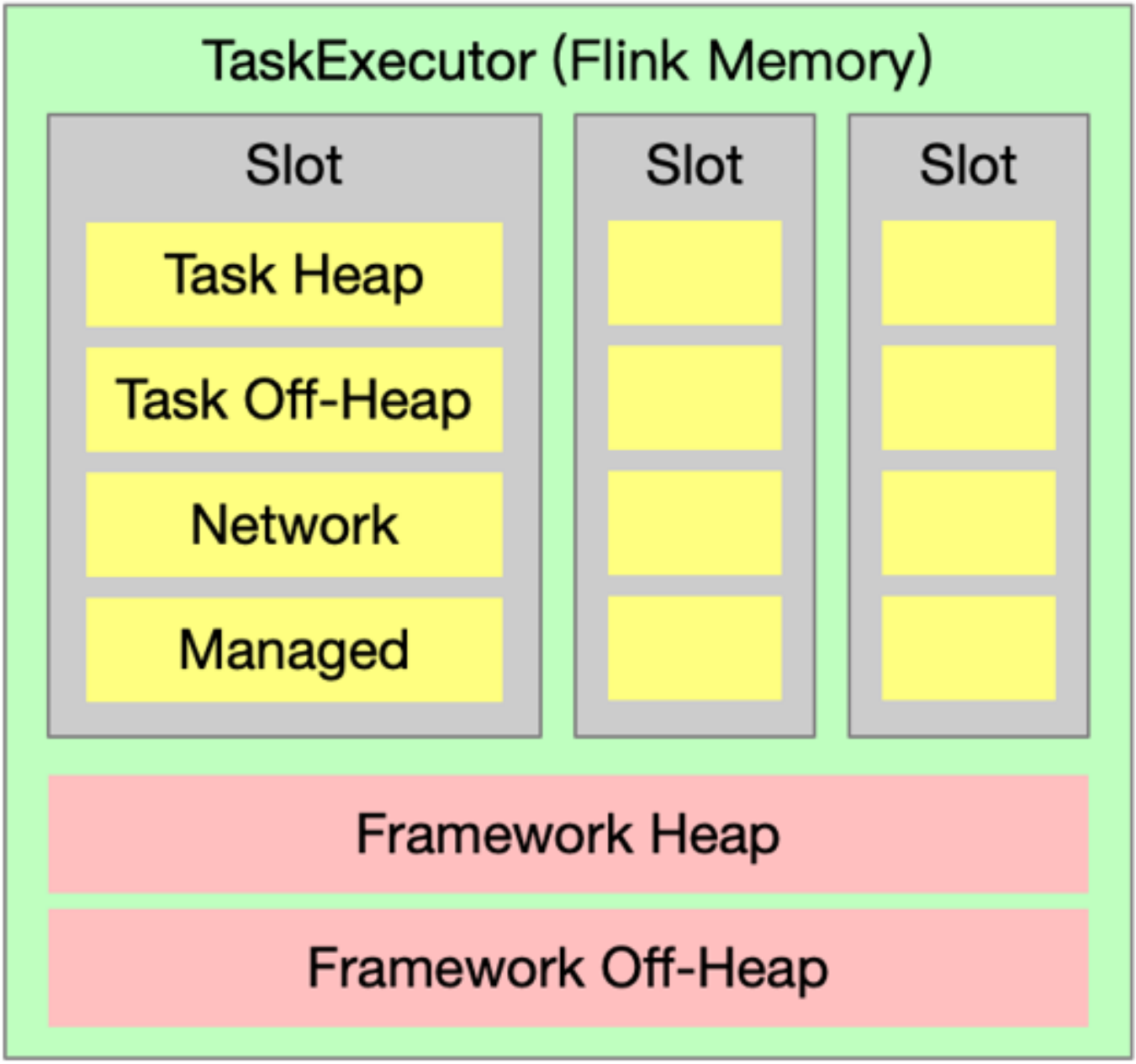


图5. Slot资源组成.png

图5. Slot资源组成

Task Manager 中有固定数量的 Slot ，Slot 的具体数量由配置决定。同一 Task Manager 上 Slot 之间没有差别，每一个 Slot 都一样大，即资源一样多。

2.3 Flink Cluster 有多少 Task Manager ?

Standalone 部署模式

在 Standalone 部署模式下，Task Manager 的数量是固定的，如果是 start-cluster.sh 脚本来启动集群，可以通过修改以下文件中的配置来决定 TM 的数量；也可以通过手动执行 taskmanager.sh

脚本来启动一个 TM 。

```
<FLINK_DIR>/conf/slaves
```

Active Resource Manager 部署模式

Kubernetes，Yarn，Mesos

由 SlotManager / ResourceManager 按需动态决定

当前 Slot 数量不能满足新的 Slot Request 时，申请并启动新的 TaskManager

TaskManager 空闲一段时间后，超时则释放

Note：On-Yarn 部署模式不再支持指定固定数量的 TM ，即以下命令参数已经失效。

```
yarn-session.sh -n <num>
flink run -yn <num>
```

2.4 Cluster -> Job 资源调度的过程

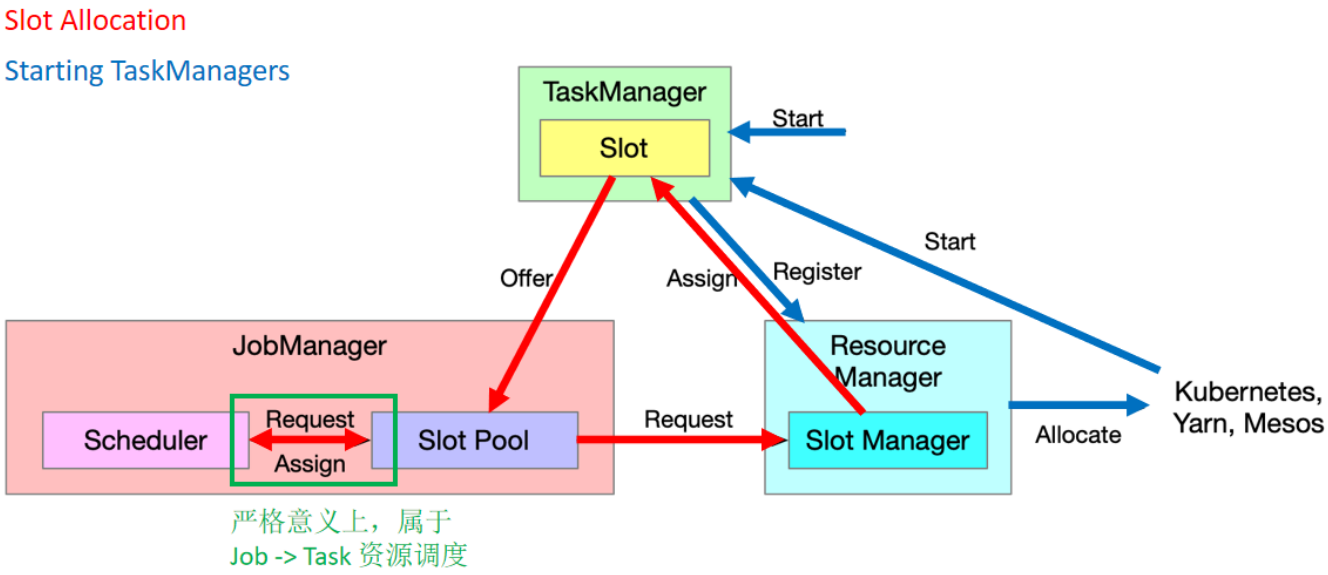


图6. Cluster 到 Job 的资源调度过程.png

图6. Cluster 到 Job 的资源调度过程

如图6，Cluster 到 Job 的资源调度过程中主要包含两个过程。

Slot Allocation（图6中红色箭头）

Scheduler 向 Slot Pool 发送请求，如果 Slot 资源足够则直接分配，如果 Slot 资源不够，则由 Slot Pool 再向 Slot Manager发送请求（此时即为 Job 向 Cluster 请求资源），如果 Slot

Manager 判断集群当中有足够的资源可以满足需求，那么就会向 Task Manager 发送 Assign 指令，Task Manager 就会提供 Slot 给 Slot Pool，Slot Pool 再去满足 Scheduler 的资源请求。

Starting TaskManagers（图6中蓝色箭头）

在 Active Resource Manager 资源部署模式下，当 Resource Manager 判定 Flink Cluster 中没有足够的资源去满足需求时，它会进一步去底层的资源调度系统请求资源，由调度系统把新的 Task Manager 启动起来，并且 TaskManager 向 Resource Manager 注册，则完成了新 Slot 的补充。

2.5 Job -> Task 资源调度的过程

Scheduler

根据 Execution Graph 和 Task 的执行状态，决定接下来要调度的 Task

发起 SlotRequest

决定 Task / Slot 之间的分配

Slot Sharing

Slot Sharing Group 中的任务可共用Slot

默认所有节点在一个 Slot Sharing Group 中

一个 Slot 中相同任务只能有一个

优点

运行一个作业所需的 Slot 数量为最大并发数

相对负载均衡

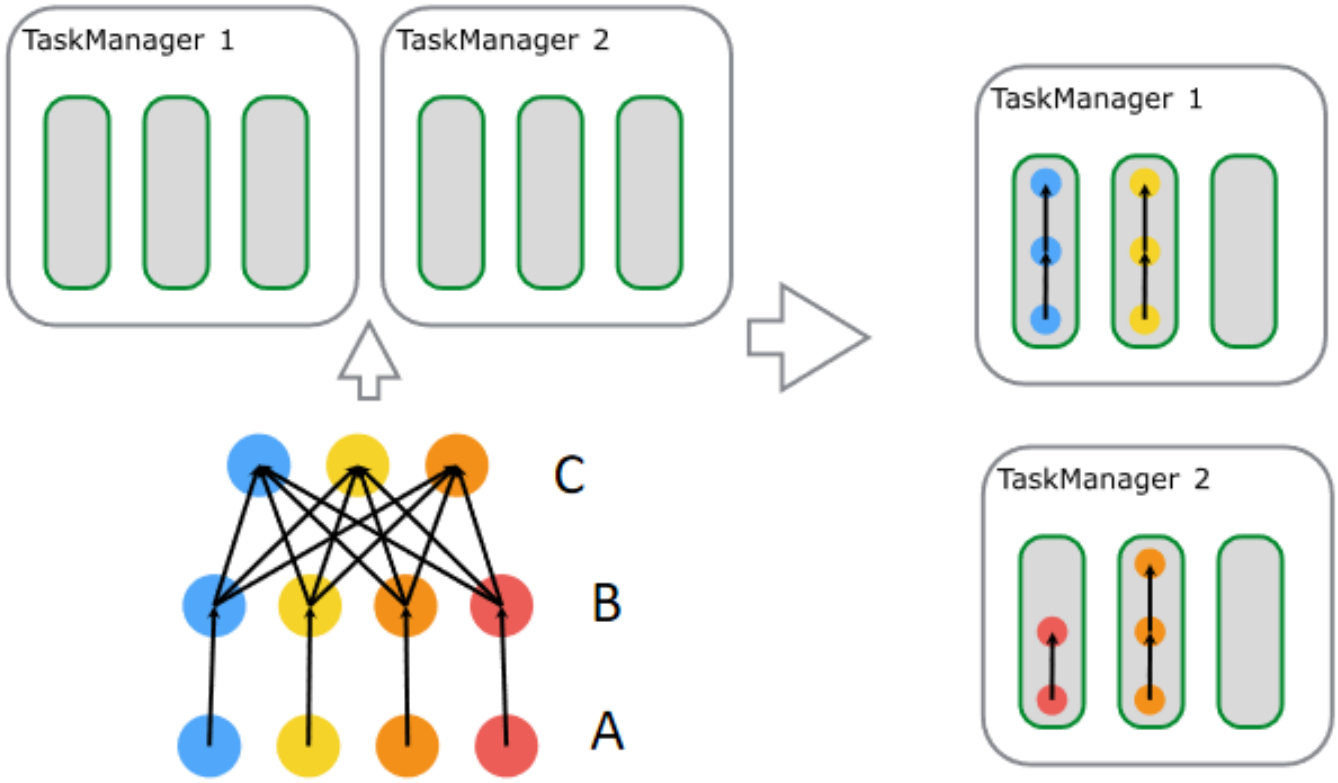


图7. Job 到 Task 资源调度过程.png

图7. Job 到 Task 资源调度过程

Slot Sharing 过程如图7所示（每一行分别是一个 task 的多个并发，自下而上分别是 A、B、C），A、B、C 的并行度分别是4、4、3，这些 Task 属于同一个 Slot Sharing Group 中，所以不同的 Task 可以放在相同的 Slot 中运行，如图7右侧所示，有3个 Slot 放入了 ABC，而第四个 Slot 放入了 AB 。通过以上过程我们可以很容易推算出这个 Job 需要的 Slot 数是4，也是最大并发数。

2.6 资源调优

通过以上介绍的机制，我们容易发现，Flink 所采用的是自顶向下的资源管理，我们所配置的是 Job 整体的资源，而 Flink 通过 Slot Sharing 机制控制 Slot 的数量和负载均衡，通过调整 Task Manager / Slot 的资源，以适应一个 Slot Sharing Group 的资源需求。Flink 的资源管理配置简单，易用性强，适合拓扑结构简单或规模较小的作业。

3. 未来发展方向

3.1 细粒度资源管理

■ Slot Sharing 的局限性

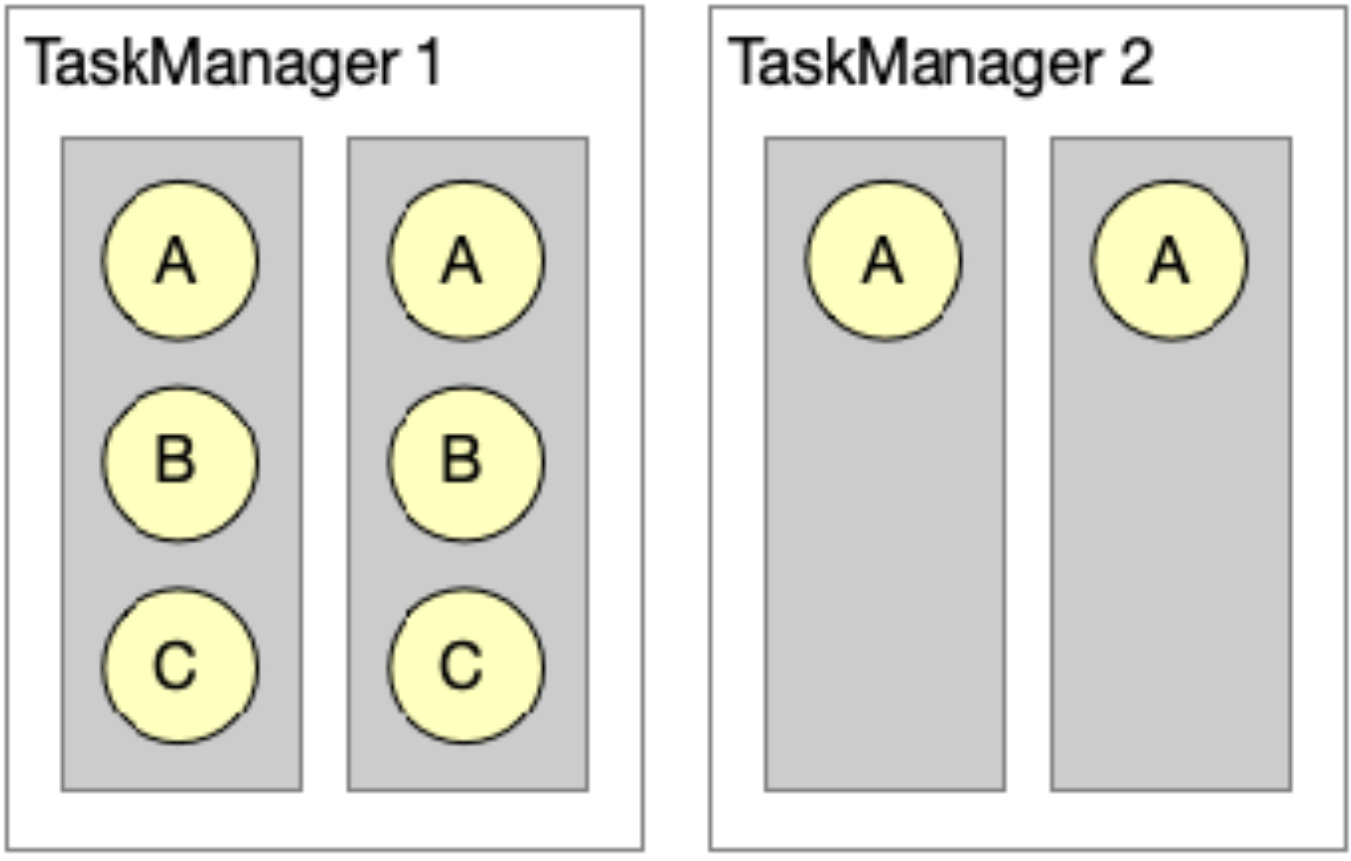


图8. Slot Sharing的局限性.png

图8. Slot Sharing的局限性

资源利用率非最优

通过 Slot Sharing 机制我们可以看到，对资源的利用率不是最优的，因为我们是按照最大并发数来配置 Slot 的资源，这样就会造成如图8所示的部分资源被浪费。

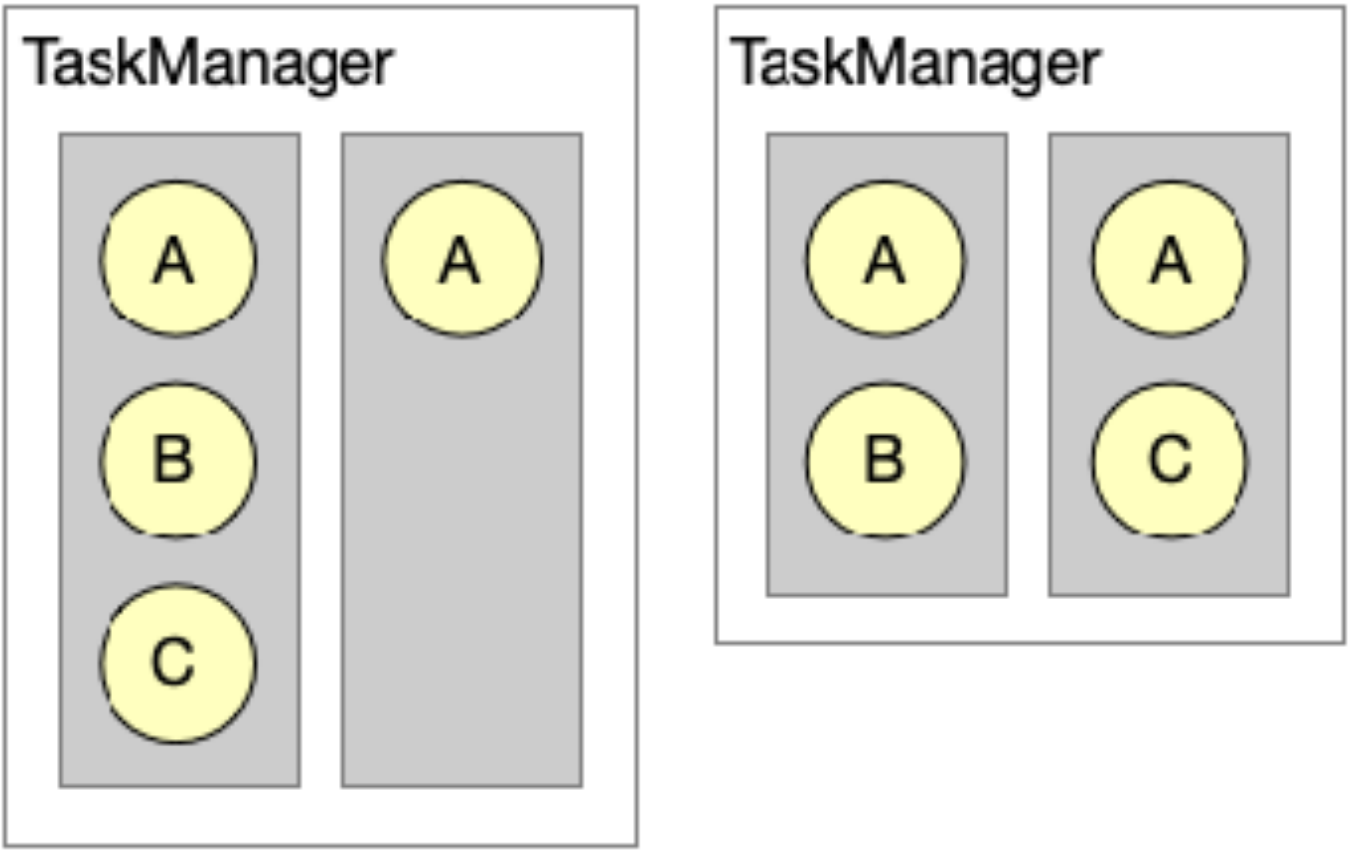


图9. Slot Sharing 的不确定性.png

不确定性

如图9所示，A 的并发度是2，BC 的并发度是1，图9中的两种分配方式均满足 Slot Sharing 机制的要求，这样就可能会出现如下情况：我们在测试的时候出现的是上图右边这种 Slot 资源配置情况，我们进行了调优配置好了 Slot 的大小，但是我们真正提交作业到生产环境中确是上图左边的情况，这样就会造成资源不够用，进而导致作业无法执行。

■ 细粒度资源管理

基于以上 Slot Sharing 机制的局限性，我们提出了细粒度资源管理的概念。

当算子的资源需求是已知的，可以通过经验性的预估、半自动化或自动化的工具来衡量 Slot 的资源大小。

每一个 Task 独占一个 Slot 来进行资源调度。

3.2 动态 Slot 切分

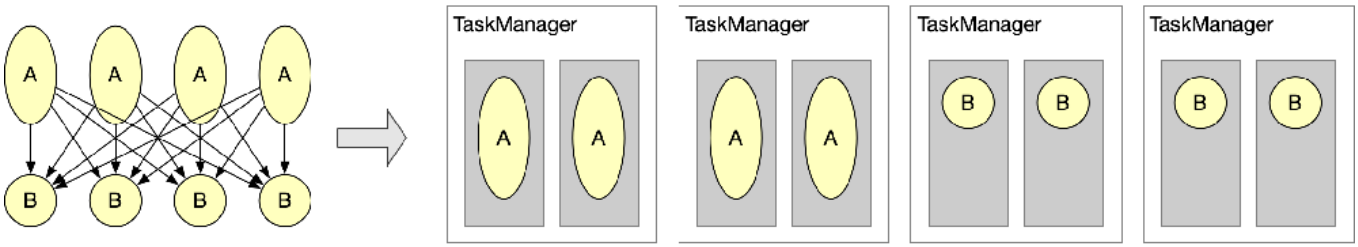


图10. 静态 Slot 分配.png

图10. 静态 Slot 分配

如图10所示，我们用圆圈的大小来表示该任务所需资源的多少，如果不采用 Slot Sharing Group 机制，现有的 Flink 资源管理机制要求 Slot 的大小必须一致，所以我们可以得到右侧这样的 Slot 资源配置，四个 Task Manager。

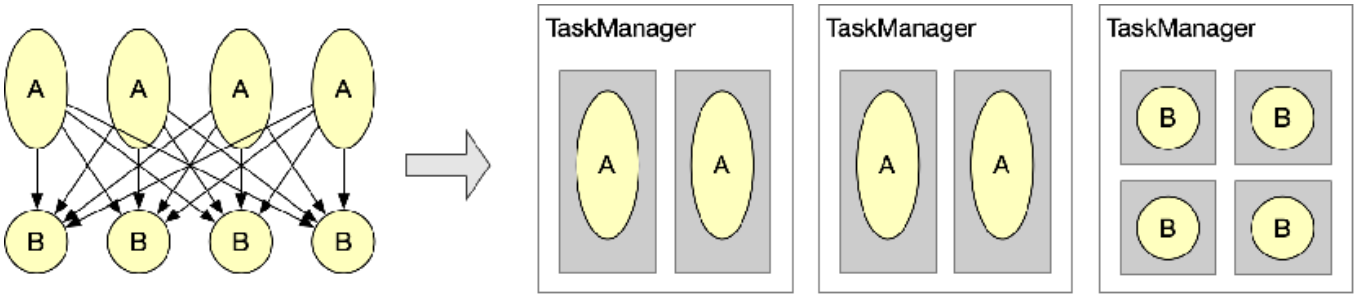


图11. 动态 Slot 切分.png

图11. 动态 Slot 切分

如果我们可以根据不同任务动态的决定每个 Slot 的大小，我们就可以将 Task Manager 切分成如图11所示的情况，仅需要三个 Task Manager。

动态 Slot 切分（FLIP-56）

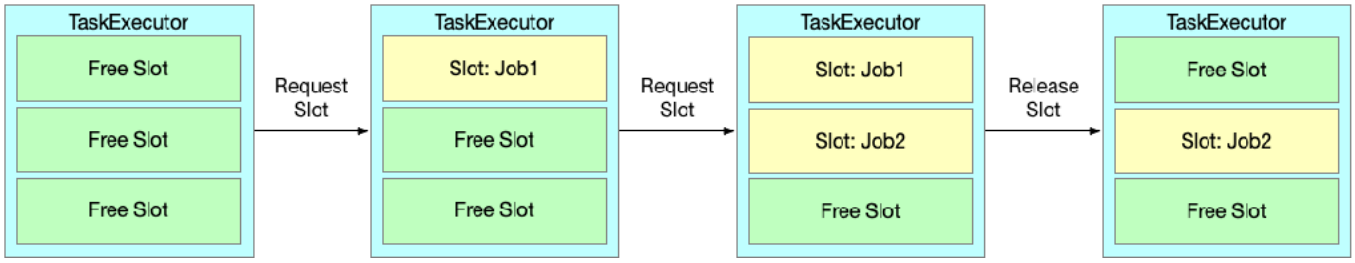


图12. 静态 Slot 划分.png

图12. 静态 Slot 划分

如图12所示，这是当前静态的固定大小的 Task Manager 的管理方式，随着任务的执行，Slot 只能简单的被占用或者被释放，而不能进行更多额外调整。

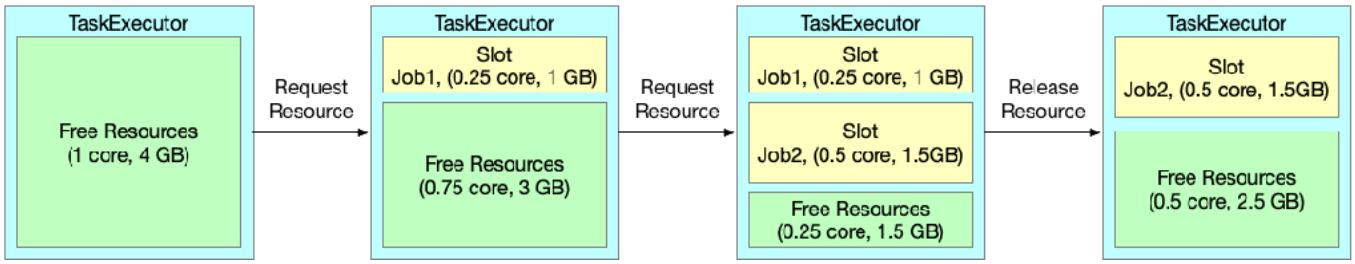


图13. 动态 Slot 划分.png

图13. 动态 Slot 划分

如图13所示，每一个 Task Manager 启动之后是一整块的资源，每接收一个资源请求时，都可以根据该请求动态的切分出一个 Slot 提供给它。但这也是有缺陷的，因为不管我们怎样切分，都经常会出现一小部分资源被浪费的情况，这也是我们常说的资源碎片问题。

3.3 碎片化问题

针对上述提到的资源碎片问题，我们提出了一个解决方案，可以根据 Slot Request 资源需求定制 Task Manager 资源，当前Flink 1.10 中每一个 Task Manager 都是一致的，但是在细粒度的资源管理中，已知资源需求时，完全可以定制 Task Manager，从理论上讲是完全可以彻底杜绝资源碎片问题。

这样做的代价是需要延长作业的调度时间，要想定制 Task Manager 就必须要等收到 Slot Request 后才可以，启动 Task Manager 的过程是比较耗时的。另一方面，可能会导致 Task Manager 比较难复用，很有可能需要释放掉旧的 Task Manager 而启动新的，这也会耗费很多时间。

在不同的应用场景下也可使用不同的方案：

Streaming（流处理）

一次调度，长期运行

提高资源利用率的收益较高

适合采用定制 Task Manager 资源的调度策略

Batch（批处理，尤其是短查询）

频繁调度，Task 运行时间短

对调度延迟敏感

适合采用非定制的 Task Manager 资源的调度策略

3.4 易用性问题

与现有的资源调优相反，细粒度资源管理下的资源调优是自底向上的资源管理，我们不再是需要配置 Job 的整体资源，而是需要用户去配置每个 Task 具体的资源需求，我们需要把 Task 的资源配置尽可能的接近其实际的资源需求，来提高资源利用率。但是同样带来的问题是，配置难度高。所以更适用于拓扑复杂或规模较大的作业。

与当前的资源调优相比，两种机制并不是孰优孰劣的关系，而是可以针对不同的场景需求适配不同的调优策略，在社区看来，两种策略均有存在的价值。

3.5 资源调度策略插件化（FLINK-14106）

不管是当前静态的资源管理机制，还是细粒度资源管理机制都要求调度策略针对不同的场景来进行不同的变化。目前 Flink 1.11 中调度策略插件化的开发工作已经完成。

资源调度策略

Task Manager 的数量

何时申请/释放 Task Manager

Task Manager 的资源大小

Slot Request 与 Task Manager 资源之间的适配

通过这三个资源调度策略，我们可以得到如下优势：

解决流处理和批处理的不同资源调度策略需求

满足用户对于细粒度、非细粒度资源管理的不同选择

未来更多资源调度策略带来的可能性

例如：Spark 根据负载弹性伸缩集群的策略

随着 Flink 支持越来越多的应用场景，灵活的资源调度策略对于保障高性能及资源效率至关重要，我们欢迎更多 Flink 爱好者和开发者加入我们社区，携手共进。

作者介绍：

宋辛童（五藏），阿里巴巴高级开发工程师。2018 年博士毕业于北京大学网络与信息系统研究所，后加入阿里巴巴实时计算团队，主要负责 Apache Flink 及阿里巴巴企业版本 Blink 中资源调度与管理机制的研发工作。

弹性计算 资源调度 Kubernetes 负载均衡 调度 Apache 流计算 开发者 异构计算 容器