(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 107832129 B
(45)授权公告日 2020.05.19

(21)申请号 201710997546.2
(22)申请日 2017.10.24
(65)同一申请的已公布的文献号
申请公布号 CN 107832129 A
(43)申请公布日 2018.03.23
(73)专利权人 华中科技大学
地址 430074 湖北省武汉市洪山区珞喻路1037号
(72)发明人 吴松 金海 柳密
(74)专利代理机构 华中科技大学专利中心 42201
代理人 李智 曹葆青
(51)Int.Cl.
G06F 9/48(2006.01)

审查员 沈晴

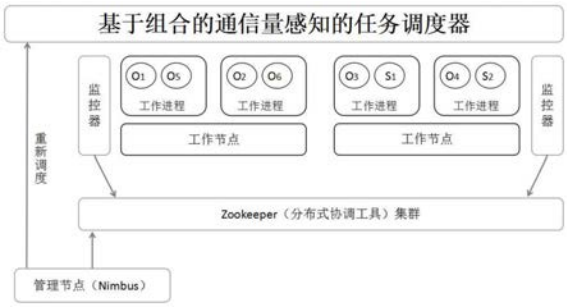
权利要求书2页 说明书7页 附图3页

(54)发明名称

一种面向分布式流计算系统的动态任务调度优化方法

(57)摘要

本发明提供一种面向分布式流计算环境下的任务调度优化方法，在运行时周期性地采集任务之间传输的数据量，通过分组机制，将通信量大的任务划分到一个组，一个组的任务在后续的调度过程中，会被当成一个整体进行调度。本发明的方法有很好的扩展性，当任务拓扑结构非常大的时候，本发明可以采用多层次组合方法，将上一层得到的组进行再次合并，得到更大的组，从而启用更少的调度单元，进一步减少调度规模。当规模足够小的时候，再将这些组调度到进程，降低网络通信开销。进一步地，本发明在调度时还依据负载大小调用进程，从而在确保降低网络通信开销的同时达到负载平衡。



CN 107832129 B

1.一种面向分布式流计算环境下的任务调度优化方法,其特征在于,包括如下步骤:

步骤1、依据当前周期内任务间的通信量预测下一周期内任务间的通信量;具体实现过程为:在采样周期T内统计任务i发送给任务j的消息总数量记为任务i到j的通信量$Traffic_{T(i,j)}$,结合前一周期内任务i到j的通信量$Traffic_{last(i,j)}$预测下一个周期内任务i到j的通信量$Traffic_{next(i,j)}$:

$Traffic_{next(i,j)}=Traffic_{last(i,j)}\times\beta+Traffic_{T(i,j)}\times(1-\beta)$

式中,β是比例伸缩常量;

步骤2、将下一周期内预测通信量最大的两个任务组合为一个任务组;具体实现过程为:

从任务i的所有邻居未访问任务中,选取一个任务i与它之间的预测通信量最大的任务,将其与任务i组合为任务组parentTask,如果任务i的所有邻居任务均已被访问,仅将任务i独立组合为任务组parentTask,完成第一层组合;

遍历每一条数据流,如果数据流的源任务和目标任务属于同一个parentTask,那么这条数据流在下一层组合时忽略;如果数据流的源任务和目标任务不属于同一个parentTask,那么这条数据流将跨越两个不同的parentTask,将这两个paremTask之间的所有数据流的通信量之和作为这两个parentTask在下一层组合时的预测通信量;从parentTask的所有邻居未访问parentTask中,选取一个与它之间的预测通信量最大的parentTask,将其与parentTask组合为下一层次的任务组parentTask,完成第二层组合;重复第二层组合方式,直到最终的任务组数低于阈值taskNumThreshold;

步骤3、启动进程并以任务组为单位进行调度;具体实现方式为:选取当前负载小于进程平均预测负载的进程作为候选进程,从候选进程中选取当前负载最低的调度任务组;

所述进程平均预测负载的计算方式为:获取当前启动的进程总数W,将当前调度的k个任务组在下一周内的预测负载W_load_{nexti}求和得到预测总负载,依据进程总数和预测总负载计算每个进程worker的平均预测负载:

$$W\_load_{average}=\frac{\sum_{i=1}^{i=k}W\_load_{nexti}}{W}。$$

2.根据权利要求1所述的面向分布式流计算环境下的任务调度优化方法,其特征在于,所述在采样周期T内统计任务i发送给任务j的消息总数量的具体实现方式:检测系统中的任务发送队列,记录任务i发送的消息中携带的目标任务j的ID,通过ID更新对应的消息数量。

3.根据权利要求1或2所述面向分布式流计算环境下的任务调度优化方法,其特征在于,使用双层哈希表保存任务i到任务j的通信量,映射关系为任务i的ID、任务j的ID、任务i发送给任务j的消息数量。

4.根据权利要求1所述的面向分布式流计算环境下的任务调度优化方法,其特征在于,所述任务在下一周内的预测负载W_load_{nexti}的获取方式为:

在采样周期T内统计任务i接收和发送的消息总数量W_load_{T}记为任务i的负载W_load_{Ti},结合前一周期内任务i的负载W_load_{Lasti}预测下一周内任务i的负载W_load_{nexti}:

$W\_load_{nexti}=W\_load_{lasti}\times\alpha+W\_load_{Ti}\times(1-\alpha)$

式中,α是比例伸缩常量。

一种面向分布式流计算系统的动态任务调度优化方法

技术领域

[0001] 本发明属于分布式流计算领域，更具体地，涉及一种面向分布式流计算系统的动态任务调度优化方法，用于像Storm这种基于操作符的流计算系统在分布式环境下如何优化任务调度方法来满足负载平衡和降低处理延时，进而提高系统的性能。

背景技术

[0002] 在过去几年中，信息产业以前所未有的高速发展，传统的数据库管理系统不再能够满足大数据所具有的挑战性的要求。根据数据是静态还是动态，我们可以将计算分类为批处理和流处理。在前者中，用于处理的数据应该在计算开始之前实现，而后者用于处理的数据是没有约束的连续流。其中Hadoop是Map-Reduce最具代表性的开源实现，已经成为最流行的离线批处理框架。然而，当数据是无界流时，需要及时处理它，这是Map-Reduce这一类框架几乎不能满足的一个要求。因此，需要一种新的框架。为了满足流处理的要求，很多新的框架被提出来。Storm是当前最流行的分布式流处理引擎之一，能够实时对高吞吐量数据流进行分析。Storm现在被许多公司广泛使用，如Twitter和阿里巴巴。

[0003] 在Storm每个shuffling阶段，在任务之间有大量的tuple(即消息)流传输，这是主要的通信开销。如果通信的任务之间的网络距离尽可能短，则通信开销将显着降低。即使任务的分配如此重要，任务分配问题在Storm中仍然没有解决。Storm中的默认调度程序称为均衡调度器，它主要可以分为两个步骤:第一步，它使用循环策略将执行程序分配给工作程序，然后使用相同的方法将workers分发到集群的节点。这种调度算法的目的是保证负载平衡，而且易于实现，但它有三个明显的缺点。首先，它不考虑任务之间的数据传输，这可能导致高通信延迟，从而导致执行性能降低。第二，它不考虑工作负载平衡。第三，它会让每个topology尽可能多使用群集中的节点，而不考虑负载，这种不必要的分布会进一步加剧通信开销。因此，需要一个能很好解决上述缺陷的调度方法。

发明内容

[0004] 针对分布式流计算环境下通信开销的问题，本发明提出了一种动态的基于组划分的动态任务调度优化方法，该方法能在运行时周期性地采集任务之间传输的数据量，然后利用这些数据关系，在调度的时候能考虑到任务间的精确的通信开销，在对应用层完全透明的前提下，自动选择最佳的调度优化方案，显著地降低网络通信开销。

[0005] 为实现本发明技术目的，本发明采用了以下技术方案：

[0006] 一种面向分布式流计算环境下的任务调度优化方法，包括如下步骤：

[0007] (1)依据当前周期内任务间的通信量预测下一周期内任务间的通信量；

[0008] (2)将下一周期内预测通信量大的两个任务组合为一个任务组；

[0009] (3)启动进程并以任务组为单位进行调度。

[0010] 进一步地，所述步骤(2)的具体实现过程为：

[0011] 从任务i的所有邻居未访问任务中，选取一个任务i与它之间的预测通信量最大的

任务，将其与任务i组合为任务组parentTask，如果任务i的所有邻居任务均已被访问，仅将任务i独立组合为任务组parentTask，完成第一层组合；

[0012] 遍历每一条数据流，如果数据流的源任务和目标任务属于同一个parentTask，那么这条数据流在下一层组合时忽略；如果数据流的源任务和目标任务不属于同一个parentTask，那么这条数据流将跨越两个不同的parentTask，将这两个parentTask之间的所有数据流的通信量之和作为这两个parentTask在下一层组合时的预测通信量；从parentTask的所有邻居未访问parentTask中，选取一个与它之间的预测通信量最大的parentTask，将其与parentTask组合为下一层次的任务组parentTask，完成第二层组合；

[0013] 重复上述组合方式，直到最终的任务组数低于阈值taskNumThreshold。

[0014] 进一步地，所述步骤(1)的具体实现过程为：

[0015] 在采样周期T内统计任务i发送给任务j的消息总数量记为任务i到j的通信量$Traffic_{T(i,j)}$，结合前一周期内任务i到j的通信量$Traffic_{last(i,j)}$预测下一个周期内任务i到j的通信量$Traffic_{next(i,j)}$：

[0016] $Traffic_{next(i,j)}=Traffic_{last(i,j)}\times\beta+Traffic_{T(i,j)}\times(1-\beta)$

[0017] 式中，β是比例伸缩常量。

[0018] 进一步地，所述在采样周期T内统计任务i发送给任务j的消息总数量的具体实现方式：检测系统中的任务发送队列，记录任务i发送的消息中携带的目标任务j的ID，通过ID更新对应的消息数量。

[0019] 进一步地，使用双层哈希表保存任务i到任务j的通信量，映射关系为任务i的ID、任务j的ID、任务i发送给任务j的消息数量。

[0020] 进一步地，所述步骤(3)的具体实现方式为：选取当前负载小于进程平均预测负载的进程作为候选进程，从候选进程中选取当前负载最低的调度任务组；

[0021] 所述进程平均预测负载的计算方式为：获取当前启动的进程总数W，将当前调度的k个任务组在下一周内的预测负载W_load_{nexti}求和得到预测总负载，依据进程总数和预测总负载计算每个进程worker的平均预测负载：

[0022] $$W\_load_{average}=\frac{\sum_{i=1}^{i=k}W\_load_{nexti}}{W}。$$

[0023] 进一步地，所述任务在下一周内的预测负载W_load_{nexti}的获取方式为：

[0024] 在采样周期T内统计任务i接收和发送的消息总数量W_load_{T}记为任务i的负载W_load_{Ti}，结合前一周期内任务i的负载W_load_{Lasti}预测下一周内任务i的负载W_load_{nexti}：

[0025] $W\_load_{nexti}=W\_load_{lasti}\times\alpha+W\_load_{Ti}\times(1-\alpha)$

[0026] 式中，α是比例伸缩常量。

[0027] 本发明通过在运行时周期性地采集任务之间传输的数据量，通过分组机制，将通信量大的任务划分到一个组，一个组的任务在后续的调度过程中，会被当成一个整体进行调度。本发明的方法有很好的扩展性，当任务拓扑结构非常大的时候，本发明可以采用多层次的算法，将上一步骤得到的组进行合并，得到更大的组，更少的调度单元，进一步减少调度规模。当规模足够小的时候，再将这些组调度到worker进程，降低网络通信开销。进一步

地，本发明在调度时还依据负载大小调用进程，从而在确保降低网络通信开销的同时达到负载平衡。

[0028] 综上所述，本发明面向分布式流计算系统的任务调度优化方法具有以下效果和优点：

[0029] (1)本发明通过优化任务调度机制来减少通信开销和达到负载平衡，在维持对应用层透明性的前提下，能够有效地提高storm系统的负载平衡和降低处理延时，进而提高应用程序的性能。

[0030] (2)本发明仅仅需要对任务的调度策略做出修改，以模块的形式植入storm系统，不需要修改用户应用层的API和系统其它部分。因此，它是轻量级的，应用非常方便。

[0031] (3)本发明采用实时负载和通信开销的精确分析和动态的任务调度方法来优化系统性能，主要的额外开销在于实时信息的采集和任务特征的周期性预测。但是，任务调度的时间在总的作业运行时间里面占据比例很小。因此，当作业任务的数目增多时，本发明仍然能够自动调节，正常工作，具有很高的可扩展性。

附图说明

[0032] 图1基于操作符的流计算系统任务的拓扑及分布式环境下部署的示意图；

[0033] 图2是分布式流计算环境下的任务组合方法流程图；

[0034] 图3是基于任务组的调度的示意图；

[0035] 图4是基于任务组调度后的结果和Storm默认的调度方式的实验对比；

[0036] 图5是本发明实例中模块周期采集器monitor和动态调度器在Storm系统中的角色；

[0037] 图6是本发明实例中任务特征周期性获取的模块和任务负载和节点容量感知的负载均衡模块的细化流程图。

具体实施方式

[0038] 为了使本发明的目的、技术方案及优点更加清楚明白，以下结合附图和实例对本发明作进一步的详细说明。

[0039] 图1基于操作符的流计算系统任务的拓扑及分布式环境下部署的示意图，一个流计算的作业是一个有向无环图(DAG)，图中每一个节点都是一个逻辑计算单元，每个计算单元只用负责整个作业的一小部分，当一个事件从源头开始，走一遍图中的数据流，就完成了处理。在大数据环境下，由于数据量很大，所以对应的逻辑的计算单元数量也很大。与此同时，硬件的规模也很大，作业往往是部署在大量的计算节点(机器)上。流计算系统就需要将这些逻辑计算单元一一分配到对应的计算节点上，每个逻辑计算单元最终都是运行在进程里面而一个节点上可能还会开启多个并发的进程，所以还需要将计算单元分配到具体的进程。

[0040] 本发明通过在运行时周期性地采集任务之间传输的数据量，通过分组机制，将通信量大的任务划分到一个组，一个组的任务在后续的调度过程中，会被当成一个整体进行调度。本发明的方法有很好的扩展性，当任务拓扑结构非常大的时候，本发明可以采用多层次的算法，将上一步骤得到的组进行合并，得到更大的组，更少的调度单元，进一步减少调

度规模。当规模足够小的时候，再将这些组调度到worker进程，降低网络通信开销。

[0041] 请参见本发明图2，本发明方法步骤如下：

[0042] (1)依据当前周期内任务间的通信量预测下一个周期内任务间的通信量；

[0043] (2)将通信量大的两个任务组合为一个任务组；

[0044] (3)启动worker进程以任务组为单位进行调度。

[0045] 所述步骤(1)中预测通信量的具体实现过程为：

[0046] 在采样周期T内统计任务i发送给任务j的消息tuple总数量记为任务i到j的通信量$Traffic_{T(i,j)}$，结合前一周期内任务i到j的通信量$Traffic_{last(i,j)}$预测下一个周期内任务i到j的通信量$Traffic_{next(i,j)}$：

[0047] $Traffic_{next(i,j)}=Traffic_{last(i,j)}\times\beta+Traffic_{T(i,j)}\times(1-\beta)$

[0048] 式中，β是比例伸缩常量，经验值。

[0049] 所述任务i发送给任务j的消息总数量的获取方式：检测系统中的任务发送队列，记录任务i发送的消息中携带的目标任务j的ID，通过ID更新消息数量。

[0050] 按照一种优选方式，使用双层哈希表保存任务i到任务j的通信量，映射关系为任务i的ID、任务j的ID、任务i发送给任务j的消息数量。

[0051] 所述步骤(2)将通信量大的两个任务组合为一个任务组的具体实现方式为：

[0052] 遍历每一个任务，找出与该任务通信量最大的任务并将它们组合到一个组，以此类推直到所有任务都组合完成，完成第一层组合，当任务规模很大的时候，可能还需要进一步进行下一层组合。通过这种组合既能将任务间的通信负载都集中到进程内部又能减少后续的任务调度规模，从而降低了任务调度的开销。

[0053] 使用来自监控器的运行时信息，调度程序将首先根据数据流对任务进行分组。组合阶段尝试将相互通信的任务组合为parentTask(组合后的任务)。属于同一组的任务将被分配给同一个worker进程。组合后的任务将以相同的方式进行分组，直到任务数量小于taskNumThreshold，这个阈值为经验值，可根据试验结果调整。

[0054] 具体步骤如下，对于每个任务：首先获得所有的邻居任务，按照它们之间的通信量顺序排序。然后，获取第一个未访问的邻居任务，并将其组合为parentTask。如果该任务所有邻居任务之前已经被访问过，它将自己独自组合为parentTask。一旦将两个任务进行了组合，则可以忽略它们之间的通信开销。

[0055] 任务组合完成后，接下来就是数据流的处理。遍历每一条数据流，如果数据流的源任务和目标任务属于同一个parentTask，那么这条数据流在下一层组合时就可以忽略，如果数据流的源任务和目标任务不属于同一个parentTask，那么这条数据流将跨越两个不同的parentTask，在下一层组合过程中，将不能忽略，每两个parentTask之间的所有数据流的通信量之和就是这两个parentTask在下一层组合时的通信量。

[0056] 重复这样的组合方式，直到最终的任务数低于阈值taskNumThreshold。

[0057] 通过上述的组合方式，其实就是在原始的任务和组合建立的各层parentTask之间建立了树形关系。当对组合后的parentTask进行调度完成后，通过这个树关系，就可以得到原始任务的调度结果。

[0058] 所述步骤(3)启动worker进程以任务组为单位进行调度的具体实现方式为：初始化，记录当前集群里的所有拥有空闲slot(每一个slot都对应一个进程，一个节点有多少个

slot,就最多能启动多少个worker进程)的节点。优先将worker进程调度到负载低的节点的slot里;进程worker位置确定后,启动进程worker以任务组为单位进行调度。

[0059] 按照本发明的一种较佳实施方式,在调度时还依据负载大小调用进程,从而在确保降低网络通信开销的同时达到负载平衡。具体实现方式为:

[0060] 选取当前负载小于进程平均预测负载的进程作为候选进程,从候选进程中选取当前负载最低的调度任务组;

[0061] 所述进程平均预测负载的计算方式为:获取当前启动的进程总数W,将当前调度的任务组中的k个任务在下一周内的预测负载W_load_{nexti}求和得到预测总负载,依据进程总数和预测总负载计算每个进程worker的平均预测负载:

[0062] $$W\_load_{average}=\frac{\sum_{i=1}^{i=k}W\_load_{nexti}}{W}。$$

[0063] 所述任务在下一周内的预测负载W_load_{nexti}的获取方式为:在采样周期T内统计任务i接收和发送的消息总数量W_load_T记为任务i的负载W_load_{Ti},结合前一周期内任务i的负载W_load_{Lasti}预测下一周内任务i的负载W_load_i:

[0064] $W\_load_{nexti}=W\_load_{lasti}\times\alpha+W\_load_{Ti}\times(1-\alpha)$。

[0065] α是一个比例伸缩常量,经验值,决定了下个周期负载的预测更依赖于最新的周期采集值还是以前的旧值。

[0066] 任务的负载还可以用cup时间来衡量,但是CPU时间表示负载的一个前提是集群中所有节点的地CPU频率一致。虽然一般情况下这个假设是成立的,但是依然依赖底层硬件,所有本发明选择更优化的方式,即任务负载,通过任务负载来衡量负载就可以完全不依赖底层的平台,只跟当前的应用有关。

[0067] 实施例:

[0068] 本发明方法应用于storm流计算系统。

[0069] 步骤(1)预测通信量和负载。将预测结果保存到zookeeper(分布式协调工具,负责storm集群nimbus和supervisor之间的数据共享和通信)。nimbus检测到更新后,就会读取最新的预测值。

[0070] 保存到zookeeper的具体过程如下:用哈希表保存每个任务的负载值,映射关系为任务ID到其负载值,对应哈希表1;用双层哈希表保存任务间的通信量,映射关系为每一个任务ID发送给其他所有任务的tuple数量,对应哈希表2;利用Kryo反序列化工具将上述的两张哈希表序列化并保存到zookeeper指定的目录,zookeeper会检测到数据的更新,然后通知观察者做出回应,如图6所示;

[0071]

Task_ID	W_load
ID_1	L_1
…	…
ID_k	L_k

[0072] 哈希表1:负载

[0073]

Task_ID	内层哈希表	
	ID	Traffic
ID_1	ID_2	T(1, 2)
	ID_3	T(1, 3)

[0074] 哈希表2:通信量

[0075] 调用调度器,利用本发明的分组算法进行分组,基本原则就是讲通信量大的任务划分到一个组,看做一个整体进行调度,在降低了网络开销的同时,减少了调度单元。

[0076] 步骤(2)基于组的划分过程主要包括以下几个子步骤:

[0077] (2.1)遍历哈希表2,每个任务又对应一个内部哈希表,内部哈希表记录的是该任务发送给其他任务的数据量,所以接着一个内循环遍历内哈希表,一个源任务一个目标任务以及他们之间的数据量就构成了一个加权边,而每个任务都是一个节点,节点的权重在哈希表1里面可以查询到;

[0078] (2.2)但是上一步骤会出现节点的重复添加,导致错误,所以,构建拓扑的时候还需要一个额外的表,记录已经添加的节点,每一遍历到一个节点,如果已经添加就跳过,如果没有就添加到拓扑里面。

[0079] (2.3)此时全局拓扑图已经建立,开始划分组。

[0080] 图2为实例分组示意图,在如图所示的拓扑图中,任务(操作符operator,图中任务代号0都是operator的缩写)之间的通信量已经量化为图中所示的数字。接下来就是组合过程,任务05的所有邻居里面,与任务03的通信量最大,两者组成了0305。任务S1的所有邻居里面,与01的通信量最大,于是组成了S101。这时,04只剩一个未访问的邻居06,所以组合成了0406。而S2,02均无未被访问的邻居,所以就独自组成了S2,02。由于该拓扑实例的规模很小,所以只需要一层组合就可以了,接下来就只需要将这些组合后的任务调度到对应的worker即可。

[0081] 如图3所示将通信量的任务划分到一个组,作为整体进行调度。

[0082] 如果是部署到4个worker:首先会随机地把其中的四个任务放到四个worker,还剩一个,放到当前负载最低的那个,所以S2,02最后在一个worker。

[0083] 如果是部署到三个worker:首先会把其中的三个任务放到三个worker,还剩两个任务,一次放到当前任务负载最低的那个worker,最终的效果就是如图所示,虽然最终的结果并不是完全平衡的,但是是该场景下做能达到的最好的平衡了。

[0084] 如果是部署到两个worker,过程和上面一样。

[0085] 步骤(3)任务负载和节点容量感知的负载均衡包括如下子步骤:

[0086] (3.1)初始化步骤:将当前集群里的所有拥有空闲slot的worker node记录下来,然后根据他们的空闲slot进行降序排序,即,优先将worker进程调度到负载低的节点的slot里;

[0087] (3.2)worker位置确定了后就是,启动这些worker;

[0088] (3.3)接下来就是将任务调度到这些worker,每个任务都有预测的负载值,当已知启动的worker数和总的负载量,那么每个worker的平均负载也就可以求出了;

[0089] $$W\_load_{average}=\frac{\sum_{i=1}^{i=k}W\_load_{nexti}}{W}$$

[0090] 其中W为当前worker的总数,k为组合后的任务组数。

[0091] (3.4)选取当前负载小于进程平均预测负载的进程作为候选进程,每次都将任务调度到目前负载最低的候选进程worker;

[0092] 该调度流程在系统中所处的位置见图5。初步的实验结果在图4,如图所示,本发明的组合调度机制可以将消息的的平均处理延时降低40.74%。

[0093] 此外,以上所述仅为本发明的较佳实施例,不能用以限制本发明,凡是在本发明的精神和原则内做出的相应修改和替换,均应该包含在本发明的保护范围之内。

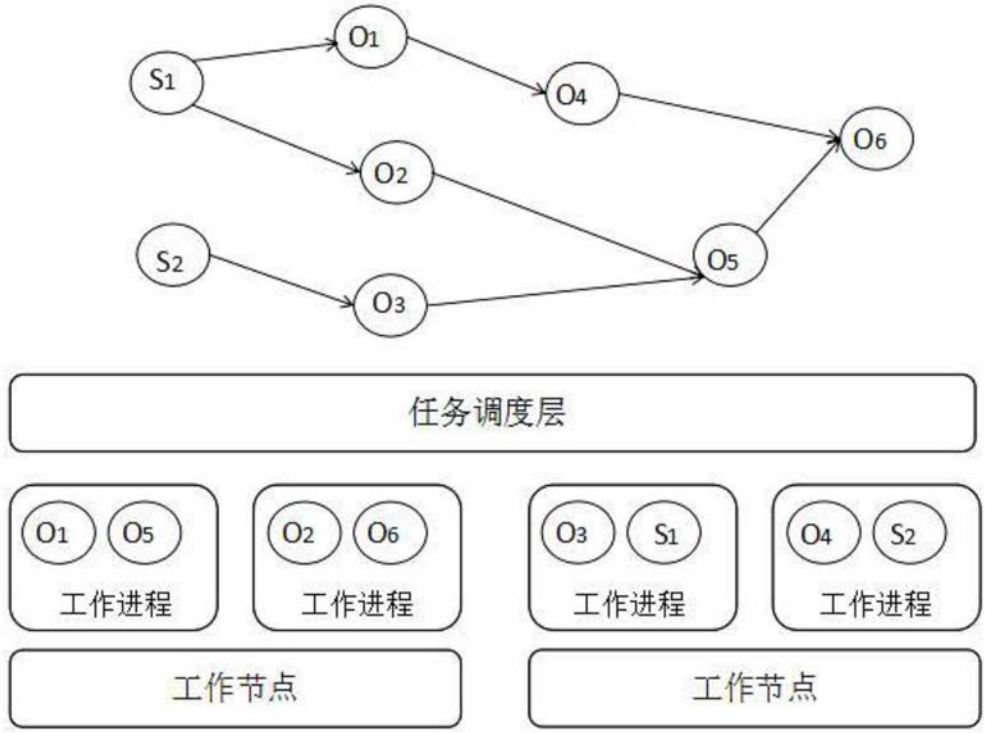

O1
S1
O4
O6
O2
S2
O5
O3
任务调度层
O1
O5
工作进程
O2
O6
工作进程
O3
S1
工作进程
O4
S2
工作进程
工作节点
工作节点

图1

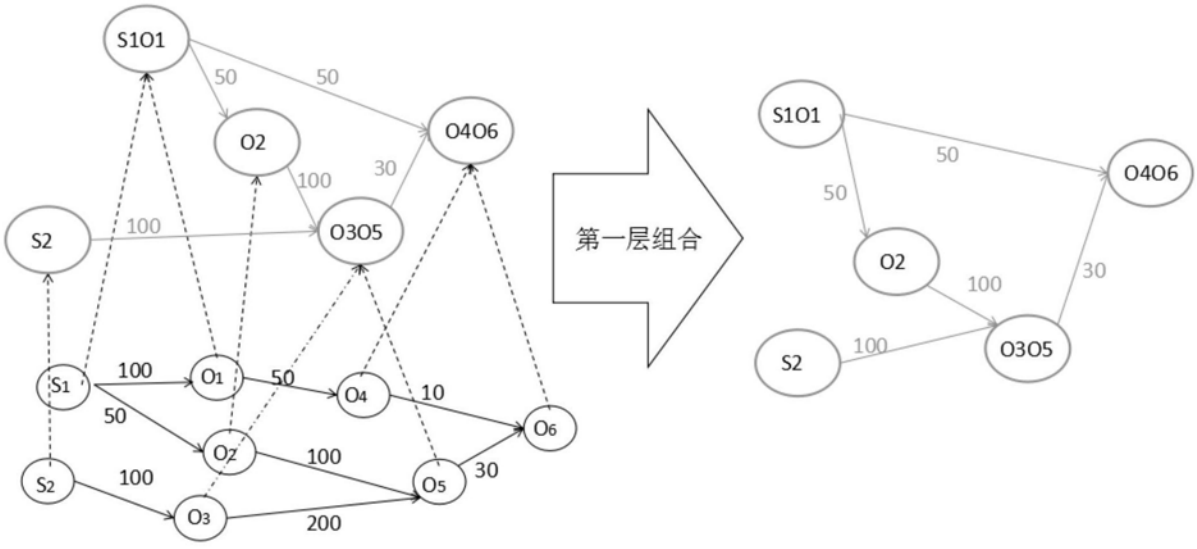

S1O1
50
50
O2
O4O6
100
30
S2
100
O3O5
S1
100
O1
50
O4
10
O6
50
O2
100
S2
100
O5
30
O3
200
第一层组合
S1O1
50
O4O6
50
O2
100
30
S2
100
O3O5

图2

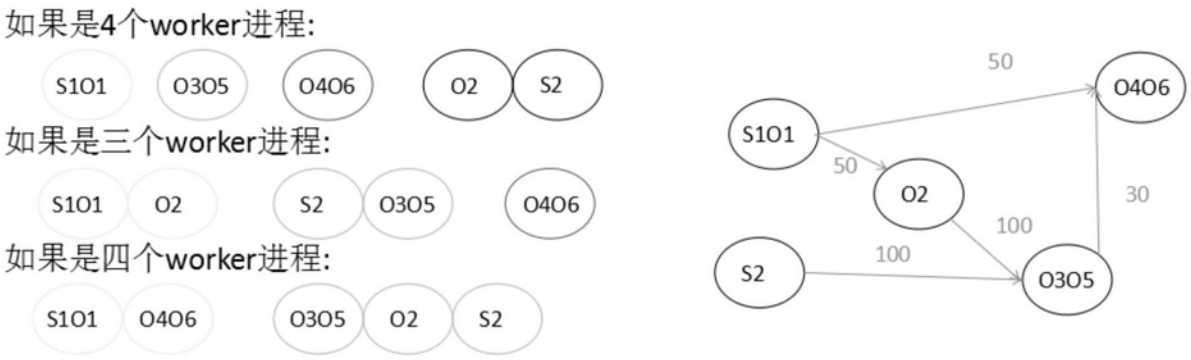

如果是4个worker进程:
S1O1
O3O5
O4O6
O2
S2
如果是三个worker进程:
S1O1
O2
S2
O3O5
O4O6
如果是四个worker进程:
S1O1
O4O6
O3O5
O2
S2
50
50
100
100
30

图3

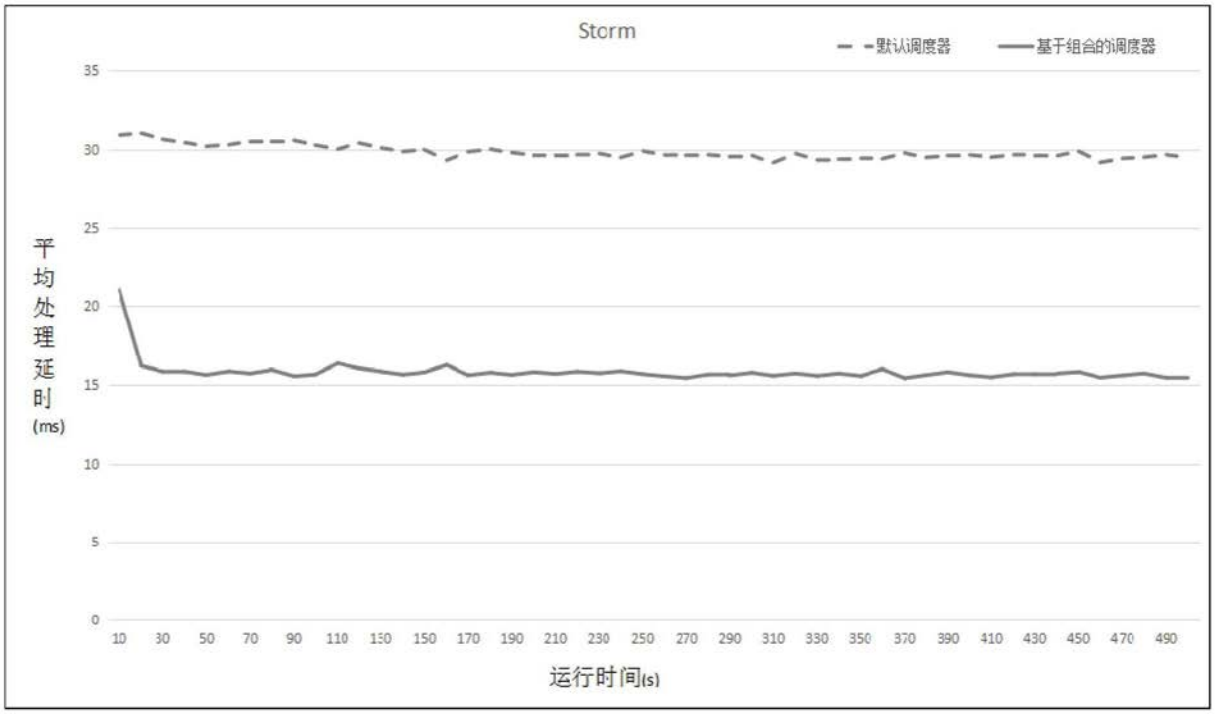

Storm
默认调度器
基于组合的调度器
平均处理延时(ms)
运行时间(s)

图4

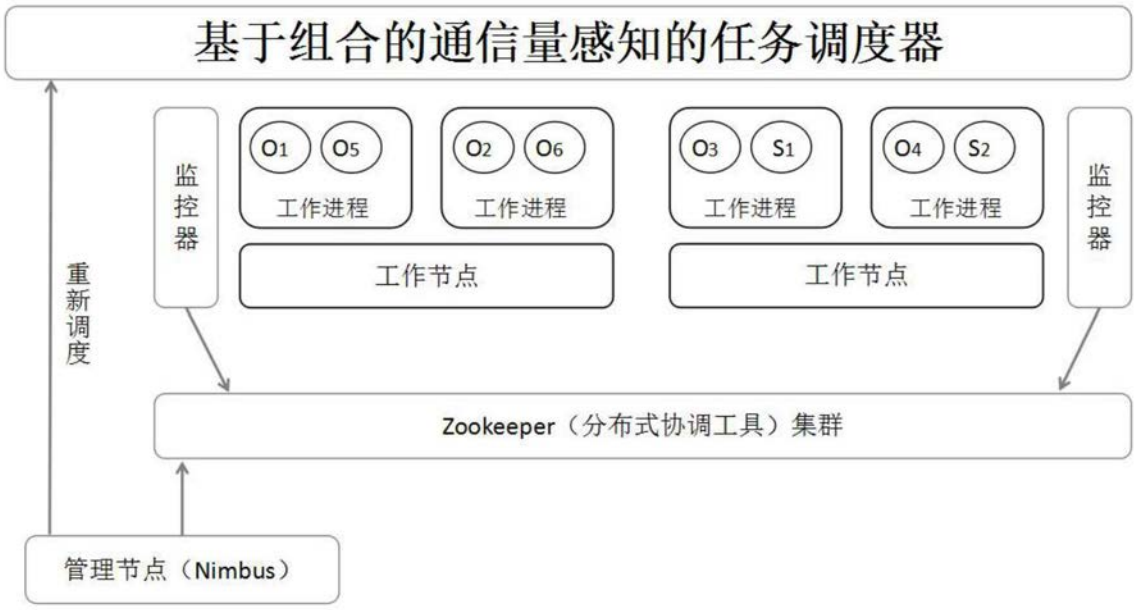

基于组合的通信量感知的任务调度器
监控器
O1
O5
工作进程
O2
O6
工作进程
O3
S1
工作进程
O4
S2
工作进程
监控器
重新调度
工作节点
工作节点
Zookeeper（分布式协调工具）集群
管理节点（Nimbus）

图5

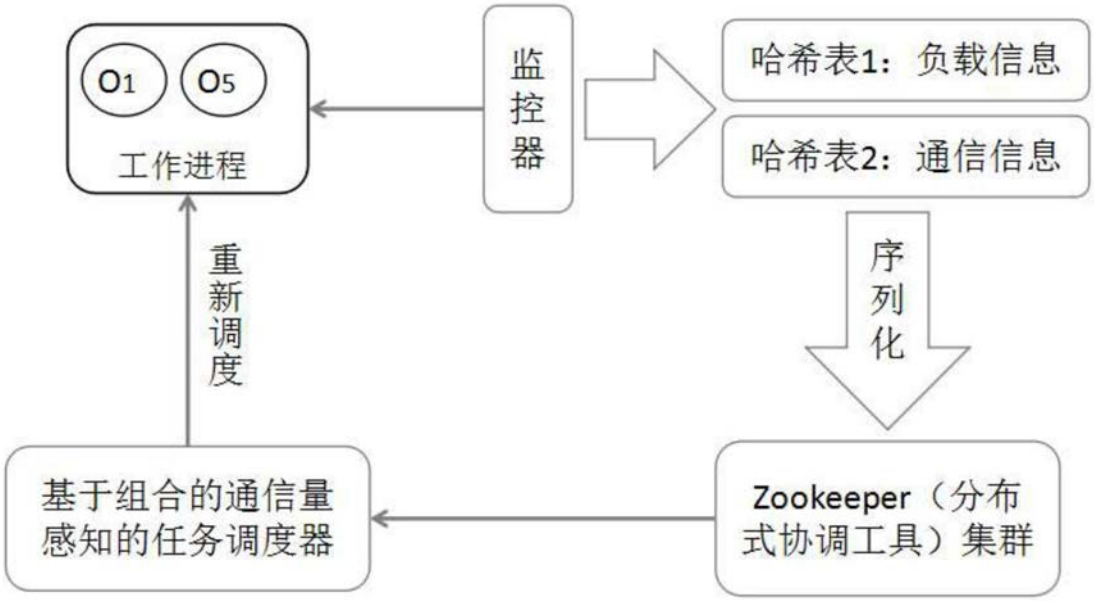

O1
O5
工作进程
监控器
哈希表1：负载信息
哈希表2：通信信息
序列化
重新调度
基于组合的通信量感知的任务调度器
Zookeeper（分布式协调工具）集群

图6