Dedas: Online Task Dispatching and Scheduling with Bandwidth Constraint in Edge Computing

Jiaying Meng, Haisheng Tan, Chao Xu, Wanli Cao, Liuyan Liu, Bojie Li

问题的提出：

如何有效地调度和调度边缘服务器的任务是至关重要的，这涉及到以下两个基本问题：

•Task dispatching：终端设备可以根据传播延迟、占用带宽、服务器处理能力等因素，选择远程云或附近某个边缘服务器来加载任务。

•Task scheduling:：将任务加载到目的服务器时，需要根据数据量、网络环境和任务截止时间来分配带宽和确定传输开始时间，这是一个网络带宽资源调度问题。此外，每个边缘服务器都要确定其排队任务的处理顺序，这是一个计算资源调度问题。

以往关于边缘计算的研究大多只考虑边缘服务器上计算资源的调度，而不考虑网络带宽。然而，网络和计算资源的调度对于减少错过最后期限的任务数量都很重要。

目标：

共同考虑网络带宽和计算资源的管理，最大限度地增加满足截止日期任务的数量，同时减少任务的平均完成时间。

算法思想：

提出了一种基于 Dedas 的实时调度算法。调度算法将新到达的任务贪婪地插入队列，并决定是否在必要时替换现有的任务以满足新的截止日期。调度算法尝试在每台服务器上处理新任务，并选择产生最佳调度的任务。

模型：

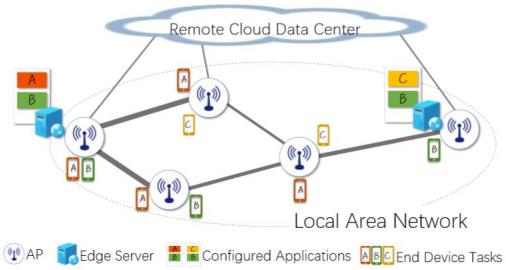

Fig. 1: An example of our model

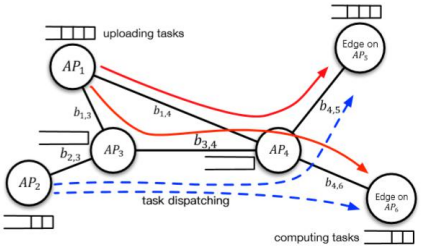

Fig. 3: System Model: the arrowed curves denote the tasks dispatching, and queues represent the uploading/computing tasks.

边缘服务器部署在接入点上。边缘服务器的计算和存储能力相对有限，因此每个边缘服务器只能配置和处理一个子集的任务。每个 AP 都有一个调度模块和一个网络资源调度模块。

1) Networking Model:

边缘计算网络建模为一个无向图 G = (V,E)。vi 表示一个 AP，edge(vi，vj)表示 vi 和 vj 通信链路 $e_{i,j}$，$e_{i,j}$ 具有传播延迟 $l_{i,j}$ 和带宽 $b_{i,j}$。$P_{i,j}$ 表示根据 vi 和 vj 之间的传播延迟计算的最短路径。

2) Edge Server Model

边缘服务器部署在 AP 上，每个 AP 最多只能部署一个边缘服务器。

K 个边缘服务器。每个边缘服务器 k 都有一个存储容量 Sk
M 种应用程序，每个应用程序 m 都需要配置 sm 存储资源
边缘服务器 k 上配置的应用程序的数量受以下约束：$\sum s_m \leq S_k, \forall k \in \{1,2,...,K\}.$
只有当边缘服务器配置了与任务对应的应用程序时，才能处理此类任务。
服务器一次只能处理一个任务，并且允许抢占。

3) Task Model:

每个任务 R 将在时间 aR 到达最近的 ap，然后被转发到服务器进行处理。
任务携带数据 γR 数据　截止日期 dR
每个任务 r 都要求特定的应用程序 m 为其提供服务
pR,k 来表示任务 R 在边缘服务器 k 上的计算时间
将距离的 vi AP 最近的任务称为（vi，ar，γr，dr，m，pr，k）th 任务
当任务被分派时，我们需要为它找到一个边缘服务器，并通过为每对 AP 预先确定的最短路径路由该任务。当任务通过中间节点时，该节点需要为此任务分配一些带宽，并且带宽可能会随时间而变化。

4) Cloud Model:

假设任务到达云后可以立即处理，而无需等待
云上任务 R 的处理时间用 pR，cloud 表示
AP 和云服务器之间存在固定的延迟 L
每个 AP 和云之间的带宽是 B.

问题定义：

上传延迟 δ↑是传播延迟和通信延迟的总和：$\delta\uparrow_R = \sum_{e_{i,j}\in\mathcal{P}_{R,k}} l_{i,j} + \frac{\gamma_R}{\min_{e_{i,j}\in\mathcal{P}_{R,k}} b_{i,j}}$
下载延迟 δ↓ = 传播延迟
等待延迟 wR = wi,R + wk,R.（等待发送的时间 + 计算时间）
任务从发布到完成的总时间ϕR = δ↑R + δ↓R + wR + pR,k
任务 R 的完成时间 CR = aR + wR + δ↑R + pR,k + δ↓R
如果将任务 r 分派到云服务器 ，则 CR = aR + wcloud,R + $\mathcal{L}\cdot 2 + \frac{\gamma_R}{\mathcal{B}} + p_{R,cloud}.$
最大限度地增加任务的数量，同时减少任务的平均完成时间

算法：

当任务到达时，我们首先调用调度策略来确定目标服务器，然后调用调度策略来确定网络上传和服务器计算的任务顺序

A. Scheduling Policy

将新任务插入到最小化计划成本函数的位置。如果所产生的调度是不可行的，我们根据成本函数替换现有任务，产生具有相同数量的完成任务和较低成本的调度。
成本函数重新考虑了两个度量：a）资源的完工时间 b）空闲时间
ACT(Q'schedule) 表示 Q'schedule 的动作（成本）

为了最小化资源碎片，尝试所有可能的插入位置（队列的尾部或在任何现有任务之前）检查可行性和计算每个调度的成本函数。（1-7）
如果有几个可行的**插入**位置，我们选择**成本最低**的位置（8-10）
如果不可能插入新的任务，我们尝试用最小的成本**替换现有的任务**并选择可行的调度。(11-12) (Line 7 - 10 in Alg. 2).

Algorithm 1: Dedas: Scheduling Policy

Input: The original scheduling sequence $Q_{schedule}$ and newly dispatched task R
Output: New scheduling sequence Q_{best}

$ACT_{best} \leftarrow \infty$; $Q_{best} \leftarrow \emptyset$;
$Q'_{schedule} \leftarrow$ Insert R into the tail of the $Q_{schedule}$;
if $Q'_{schedule}$ *is feasible* **then**
  $Q_{best} \leftarrow Q'_{schedule}$;
  $ACT_{best} \leftarrow \text{ACT}(Q'_{schedule})$;
for *every task* R' *in* $Q_{schedule}$ **do**
  $Q'_{schedule} \leftarrow$ Insert R before R' in $Q_{schedule}$;
  if $Q'_{schedule}$ *is feasible* **and** $ACT(Q'_{schedule}) < ACT_{best}$ **then**
    $Q_{best} \leftarrow Q'_{schedule}$;
    $ACT_{best} \leftarrow \text{ACT}(Q'_{schedule})$;
if $Q_{best} = \emptyset$, *(i.e. none of* $Q'_{schedule}$ *is feasible)* **then**
  $Q_{best} \leftarrow$ Choose an appropriate task to replace (Algorithm 2);

Algorithm 2: Dedas: Choose a task to replace

Input: New task R, current schedule $Q_{schedule}$
Output: Best schedule Q_{best} after replacement

$Q_{best} \leftarrow Q_{schedule}$;
$ACT_{best} \leftarrow$ ACT of $Q_{schedule}$;
for *every task* R' *in* $Q_{schedule}$ **do**
  if R' *and* R *has at least one overlapping links in transmission path or share a server* k **then**
    if R' *has started processing* **and** ($\gamma_R > \gamma_{R'}$ **or** $p_{R,k} > p_{R',k}$) **then**
      continue;
    $Q'_{schedule} \leftarrow$ Replace R' with R in $Q_{schedule}$;
    if $Q'_{schedule}$ *is feasible* **and** $ACT(Q'_{schedule}) < ACT_{best}$ **then**
      $Q_{best} \leftarrow Q'_{schedule}$;
      $ACT_{best} \leftarrow \text{ACT}(Q'_{schedule})$;

两类任务不考虑替换：
新任务 R 与不共享资源的任务 R’不能替换，因为不能产生可行的调度(Line 4 in Alg. 2)
为了替换部分处理的任务 R’，新任务 R 不应该比 R’的剩余部分具有更多的流量和计算时间 (Line 5 in Alg. 2).
如果不能将新任务插入到计划中，而不能通过替换现有任务而产生更好的计划，则将被**丢弃**，我们保留原来的计划。

B. Dispatching Policy

尝试依次将任务 R 分派到每个候选服务器（包括云服务器和边缘服务器），并调用调度策略来计算每个分派选择的调度及其 ACT。如果有多个可行的调度选择，则选择具有完成任务的最大数量和最小动作的服务器。

Algorithm 3: Dedas: Dispatching Policy

Input: task R, the set of candidate server K' (including the cloud), the information of the system
Output: the target server k_{target} to dispatch R

Target server: $k_{target} \leftarrow$ cloud;
Number of completed tasks: $N_{best} \leftarrow 0$;
ACT: $ACT_{best} \leftarrow \infty$;
for *every server* k *in* K' **do**
  if *the server* k *is cloud* **then**
    $\delta \downarrow_R \leftarrow \mathcal{L} + p_{R,cloud}$; $p_{R,cloud} \leftarrow 0$;
  Try to dispatch task R to server k;
  $N, ACT \leftarrow$ Schedule this task R according to Algorithm 1;
  if $N > N_{best}$ **or** ($N = N_{best}$ **and** $ACT < ACT_{best}$) **then**
    if *task* R *meets deadline* **then**
      $k_{target} \leftarrow k$;
    else
      $k_{target} \leftarrow$ cloud;
    $N_{best} \leftarrow N$; $ACT_{best} \leftarrow ACT$;

实验：
从五个方面评估 dedas 的性能
1) Task arrival density
2) Task distribution
3) Computing and uploading time ratio
4) Number of Edge Servers
5) Resource Utilization Ratio