Dependent Function Embedding for Distributed Serverless Edge Computing

IEEE TPDS 2022

DML GROUP MEETING
7.1

OUTLINE

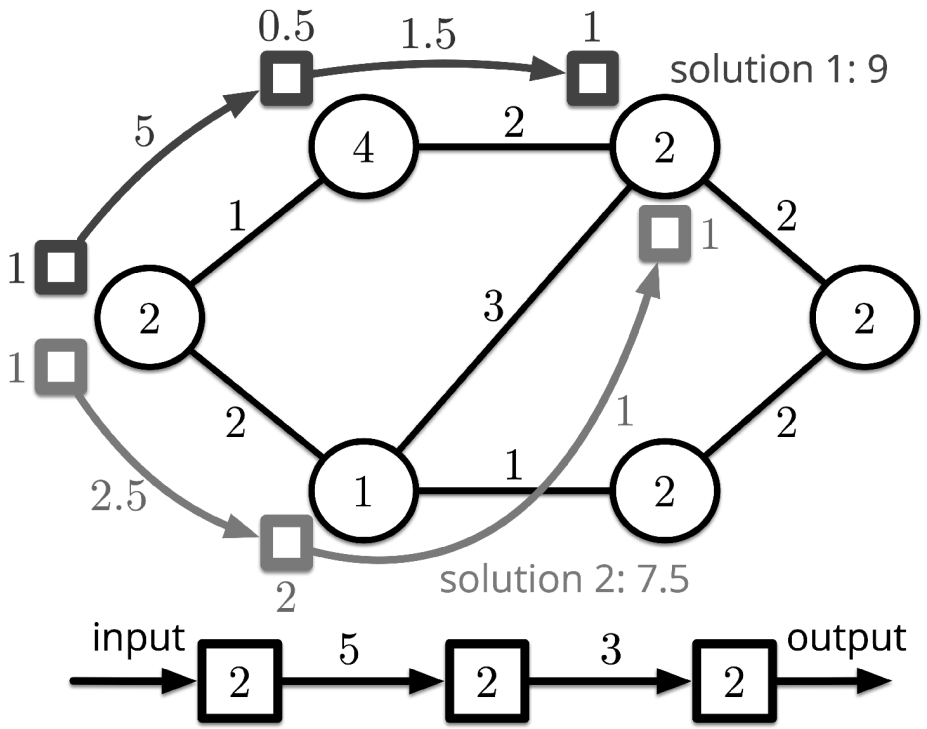
- Introduction
- Background
- System Model
- Algorithm
- Evaluation
- Conclusion

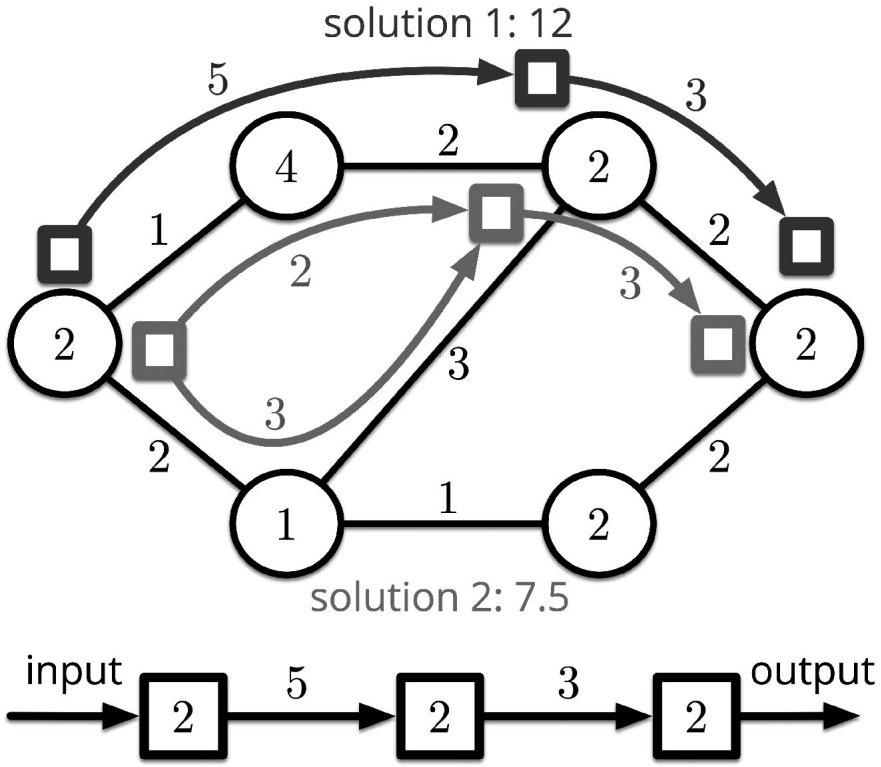
Introduction

- **Serverless**：将单体式应用切分为若干个代码片充当功能函数→**FaaS**
- 无服务器边缘计算范式为这种函数的部署提供了解决方案
- 允许用户在不关注底层基础设施的前提下执行具有差异性的应用程序
- 函数的部署模式和流量路由与映射策略显著影响应用程序的执行时间
- 探索应用程序部署和函数之间的流量路由与管理
- 定义**DPE(Dependent Function Embedding)**算法，求解最佳函数部署策略问题

Background

- 当前的研究现状主要聚焦于函数放置（将函数放置在满足其QoS的节点上，而不考虑请求过程中的路由）

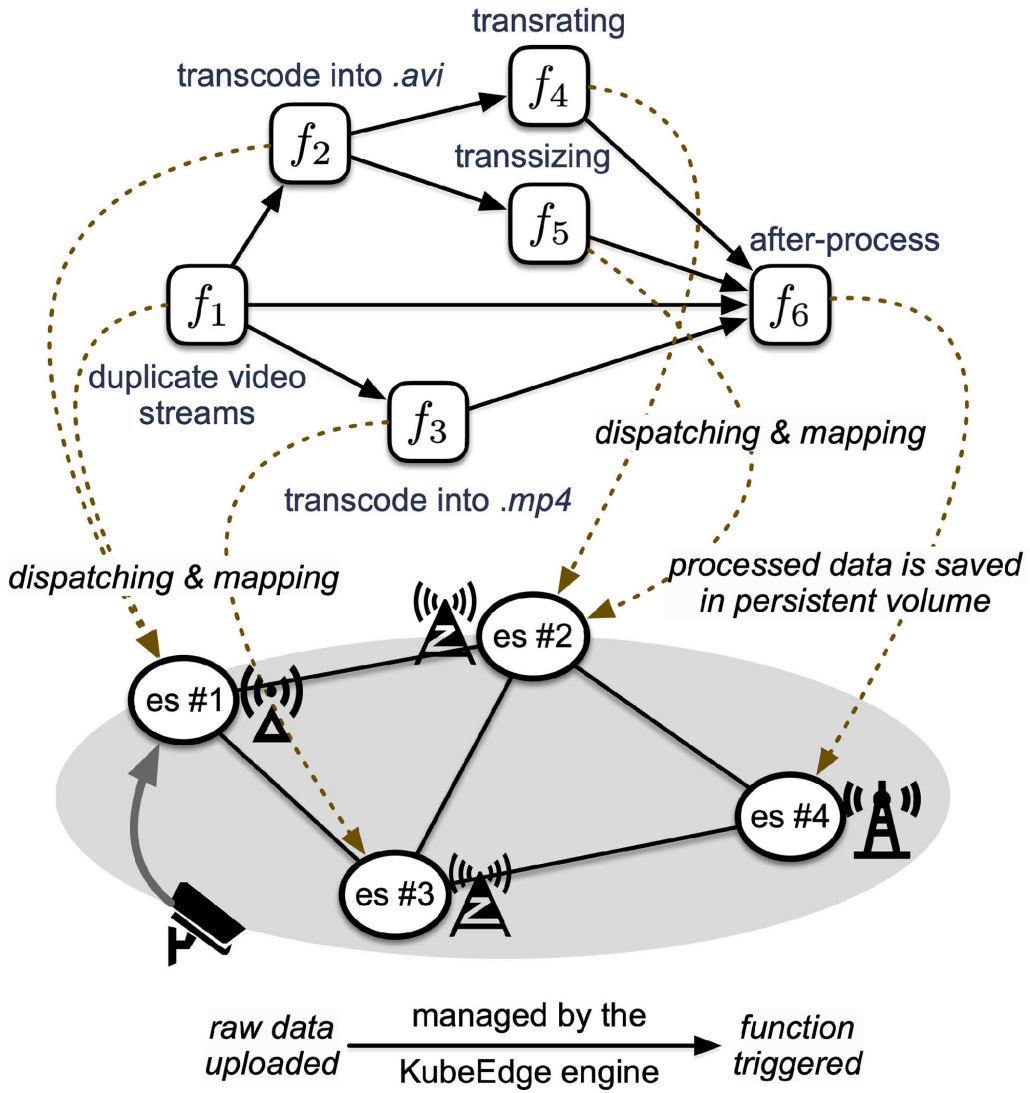




Background

- 选择哪一个服务器放置函数
- 选择最优的流量路由映射策略
- 最小化应用程序任务执行时间

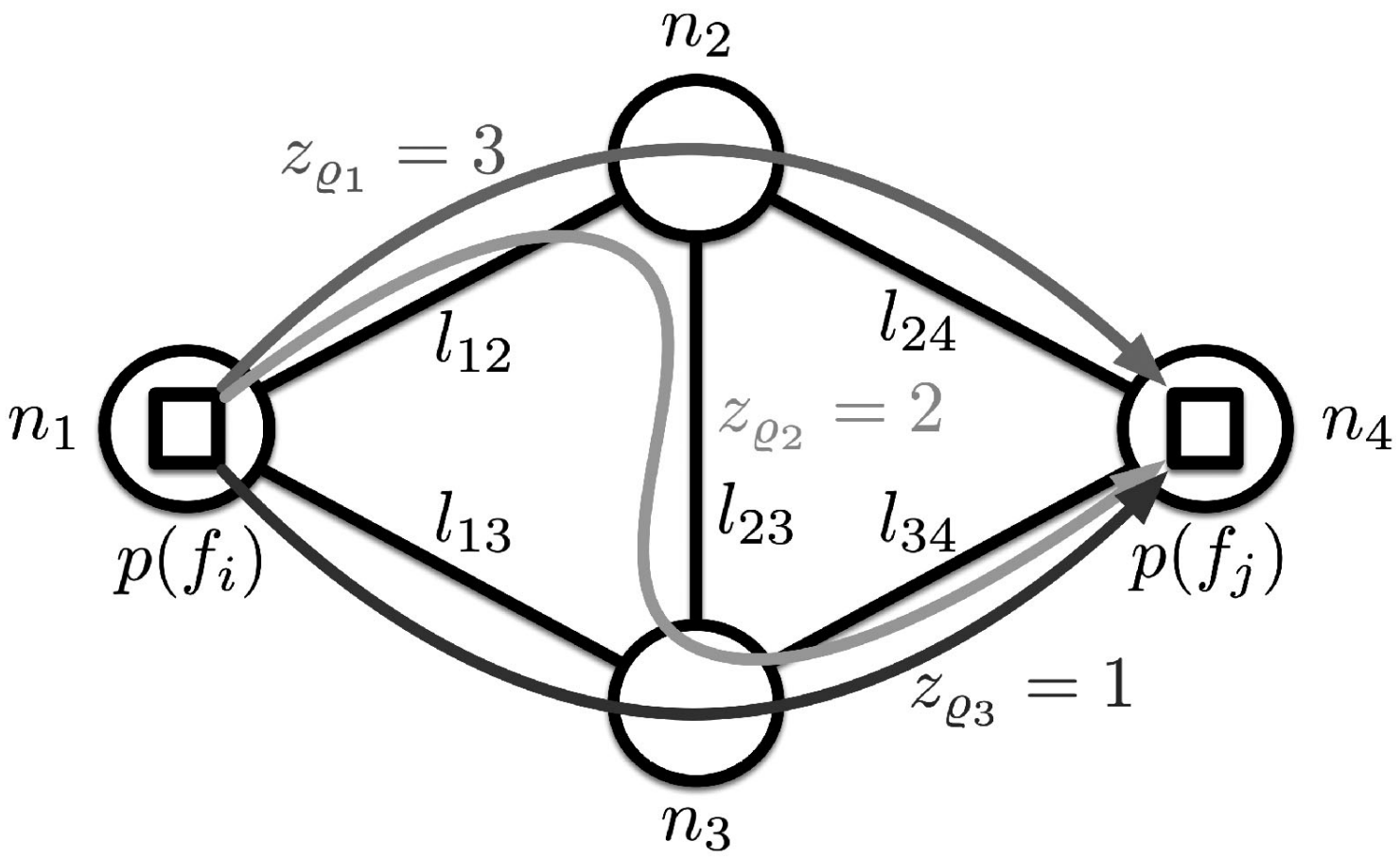
SYSTEM MODEL



Notation	Description
$\mathcal{G} \triangleq (\mathcal{N}, \mathcal{L})$	The graph abstracted from the edge network
$\mathcal{N}$	The set of edge servers
$n_i \in \mathcal{N}$	The ith edge server in $\mathcal{G}$
$\{\psi_n\}_{\forall n \in \mathcal{N}}$	Processing power of each edge server n
$\mathcal{L}$	The set of virtual links
$l_{ij} \in \mathcal{L}$	A virtual link from source n_i to target n_j in $\mathcal{G}$
$\{b_{ij}^{max}\}_{\forall l \in \mathcal{L}}$	Maximum bandwidth of the virtual link l_{ij}
$\mathcal{P}(n_i, n_j)$	The set of simple paths from server n_i to n_j
$(\mathcal{F}, \mathcal{E})$	The DAG abstracted from an IoT application
$\{f\}_{\forall f \in \mathcal{F}}$	The set of dependent functions
$c_i, \forall f_i \in \mathcal{F}$	The number of floating point operations of f_i
$e_{ij}, \forall f_i, f_j \in \mathcal{F}$	The data stream from function f_i to f_j
$s_{ij}, \forall f_i, f_j \in \mathcal{F}$	The data stream size from function f_i to f_j
$p(f) \in \mathcal{N}$	The placement of function $f \in \mathcal{F}$
$\varrho \in \mathcal{P}(n_i, n_j)$	A simple path in the set $\mathcal{P}(n_i, n_j)$
z_ϱ	The data stream size route through ϱ
$T(p(f_i))$	The finish time of f_i when scheduled onto $p(f_i)$
$t(e_{ij})$	The time consumed for transferring s_{ij}
$t(p(f_i))$	The time consumed for processing f_i on $p(f_i)$
ι_n	The earliest idle time of edge server n
$\sigma(\cdot, \cdot)$	The communication start-up cost
b_{mn}^{ϱ}	The bandwidth allocated to z_ϱ on l_{mn}

SYSTEM MODEL

Application as a DAG



SYSTEM MODEL

Dependent Function Embedding

$$\sum_{\varrho \in P\left(p(f_i), p(f_j)\right)} z_\varrho = s_{ij}$$

一条路径 e_{ij} 被拆分为多个子路径（由 ϱ 表示），z_ϱ 表示流过 ϱ 的数据量大小

SYSTEM MODEL

Involution Function of Finish Time

$$T\left(p(f_j)\right) = \max\left\{\iota_{p(f_j)}, max_{\forall i:e_{ij}\in\mathcal{E}}\left(T(p(f_i)) + t(e_{ij})\right)\right\} + t(p(f_j)) \quad (1)$$

$$T(p(f_i)) = \iota_{p(f_i)} + t(p(f_i)), \qquad f_i \in \mathcal{F}_{entry} \quad (2)$$

$$t(e_{ij}) \triangleq \sigma(f_i, f_j) + max_{\varrho\in P\left(p(f_i),p(f_j)\right)} \sum_{l_{mn}\in\varrho} \frac{z_\varrho}{b_{mn}^{\varrho}} \quad (3)$$

$$t\left(p(f_j)\right) \triangleq \frac{c_j}{\psi_{p(f_j)}} \quad (4)$$

SYSTEM MODEL

Problem Formulation

在所有的函数都完成了调度操作后，总任务的最大执行时间为**最后一个出口函数的结束时间**。
目标是通过找到 $p \triangleq \{p(f)\}_{\forall f \in \mathcal{F}}$ 和 $z \triangleq \left\{z_\varrho | \forall \varrho \in \mathcal{P}\left(p(f_i), p\left(f_j\right)\right)\right\}_{e_{ij} \in \mathcal{E}}$ 的最优值，进而求出DAG的最小总任务执行时间。

对 **依赖函数嵌入问题** 的公式化表示为：

$$\mathrm{P}: \quad min_{p,z} max_{f \in \mathcal{F}_{exit}} \mathrm{T}\left(p(f)\right)$$

$$s.t. \quad \sum_{\varrho \in P\left(p(f_i), p(f_j)\right)} z_\varrho = s_{ij}\,, \qquad \forall e_{ij} \in \mathcal{E}, \qquad (5)$$

$$z \geq 0 \qquad (6)$$

ALGORITHM

Finding Optimal Substructure

考虑到前后函数之间的依赖关系，函数的最优部署和数据流的最优（分解）映射不能同时满足。然而，我们可以根据其最优子结构一步一步地得到最优解。

我们用$T^*\left(p(f)\right)$表示：当把函数f部署到边缘服务器$p(f)$上时，函数f的最早（运行）完成时间。基于(1)，$\forall f_i \in \mathcal{F} \backslash \mathcal{F}_{entry}$，有：

$$T^*\left(p(f_j)\right) = max\left\{max_{\forall i:e_{ij}\in\mathcal{E}}\left[min_{p(f_i),\{z_\varrho\}_{\forall \varrho\in\mathcal{P}_{ij}}}\left(T^*\left(p(f_i)\right)+t\left(e_{ij}\right)\right)\right], \iota_{p(f_j)}\right\}+t\left(p(f_j)\right) \quad (7)$$

此外，对于所有的入口函数$f_i \in \mathcal{F}_{entry}$，$T^*\left(p(f_i)\right)$可以由(2)立刻计算得到。

ALGORITHM

Finding Optimal Substructure

在(7)中，对于有e_{ij}存在的函数对(f_i, f_j)，定义子问题P_{sub}：

$$P_{sub}: min_{p(f_i),\{z_\varrho\}_{\forall \varrho \in \mathcal{P}_{ij}}} \Phi_{ij} \triangleq \left(T^*\left(p(f_i)\right) + t\left(e_{ij}\right)\right)$$

$$s.t. \quad \sum_{\varrho \in \mathcal{P}_{ij}} z_\varrho = s_{ij}, \qquad (8)$$

$$z_\varrho \geq 0, \forall \varrho \in \mathcal{P}_{ij} \qquad (9)$$

在P_{sub}中，我们需要决定在哪里部署函数f_i，以及e_{ij}的映射方式。通过解决P_{sub}，我们可以根据(7)，得到最早结束时间$T^*\left(p(f_j)\right)$。这样，通过按拓扑顺序计算每个函数的最早（执行）完成时间来最优地求解 P。

ALGORITHM

Optimal Data Splitting

为了得到P_{sub}的最优解，首先固定入口函数f_i的部署位置，即 $p(f_i)$，然后集中关注数据流e_{ij}的最佳映射方式。为了得到$t(e_{ij})$的最小值，我们将矩阵 A 定义如下：

$$\mathbf{A} \triangleq \operatorname{diag}\left(\sum_{l_{mn}\in\varrho_1}\frac{1}{b_{mn}^{\varrho_1}}, \sum_{l_{mn}\in\varrho_2}\frac{1}{b_{mn}^{\varrho_2}}, \cdots, \sum_{l_{mn}\in\varrho_{|\mathcal{P}_{ij}|}}\frac{1}{b_{mn}^{\varrho_{|\mathcal{P}_{ij}|}}}\right)$$

矩阵A的对角元素均是正实数。需要确定的变量表示为：$z_{ij} \triangleq \left[z_{\varrho_1}, z_{\varrho_2}, \cdots, z_{\varrho_{|\mathcal{P}_{ij}|}}\right]^T \in \mathbb{R}^{|\mathcal{P}_{ij}|}$，因此，$P_{sub}$被转换成：

$$P_{norm}: \ min_{z_{ij}} \left\| Az_{ij} \right\|_{\infty}$$

$$s.t. \begin{cases} 1^T z_{ij} = s_{ij} \\ z_{ij} \geq 0 \end{cases} \qquad (10)$$

$T^*(p(f_i))$和σ_{ij}这两个常数并不会改变子问题的最优解z_{ij}^*。P_{norm}是一个无穷范数的最小化问题（尽可能地使z_{ij}小，求向量$\mathbb{R}^{n*1}$的最大值——选择一条时延最长的子路径作为$t(e_{ij})$带入(7)中求得最优解）。

ALGORITHM

Optimal Data Splitting

通过引入非负松弛变量 $\tau \in \mathbb{R}$ 和 $y \in \mathbb{R}^{|\mathcal{P}_{ij}|}$，$P_{norm}$可以被转换成以下的标准形式:

$$P'_{slack}: \min_{z'_{ij} \triangleq \left[z_{ij}^T, y^T\right]^T} \tau$$

$$s.t. \begin{cases} \sum_{\varrho \in \mathcal{P}_{ij}} z_\varrho = s_{ij} \\ \boldsymbol{A} z_{ij} + y = \tau \cdot 1 \\ z'_{ij} \geq 0 \end{cases}$$

P'_{slack}可通过单纯形法和内点法求得子问题的最优解。然而，当图$\mathcal{G}$的规模尺度增加时，这些标准方法可能是不可取的，因为单纯形方法具有指数级的复杂度，而内点法在最坏的情况下至少是$O\left(\left|z'_{ij}\right|^{3.5}\right)$。但我们可以直接利用最优值 z^*_{ij} 的解析表达式来求解子问题。结果由下面的定理证之。

ALGORITHM

Optimal Data Splitting

定理1. P_{norm} 的最优目标值为：

$$\min_{z_{ij}} \left\| Az_{ij} \right\|_{\infty} = \frac{S_{ij}}{\sum_{k=1}^{\left|\mathcal{P}_{ij}\right|} 1/A_{k,k}} \qquad (11)$$

当且仅当：

$$A_{u,u} z_{ij}^{(u)} = A_{v,v} z_{ij}^{(v)}, 1 \le u \ne v \le \left|\mathcal{P}_{ij}\right| \qquad (12)$$

其中 $z_{ij}^{(u)}$ 是向量 z_{ij} 的第 u 个元素，$A_{u,u}$ 是对角矩阵 A 的第 u 个对角元素。

ALGORITHM

Optimal Data Splitting

获取最优数据拆分和映射的方法可总结为Algorithm 1(OSM)，如图所示：

从第一行到第五行，可以得出算法OSM通过并行求解$|\mathcal{N}|$次P_{sub}来求解P_{norm}，每一次都有不同的$p(f_i)$，该过程可以并行执行，因为不存在相互耦合。在第四行中，直接使用解析解(11)获得P_{sub}的目标值。最耗费时间的操作在第四行到第六行，因为其分别至少要遍历一次所有的边缘服务器和简单路径。

算法OSM的计算复杂度：$O\left(max\{|\mathcal{N}|, |\mathcal{P}_{ij}|\}\right)$

Algorithm 1. Optimal Stream Mapping (OSM)

Input: $\mathcal{G}$, the function pair (f_i, f_j), and $p(f_j)$
Output: The optimal $\mathbf{\Phi}_{ij}^{\star}$, $p^{\star}(f_i)$, and $z_{ij}^{\star}$

1 **for** *each* $m \in \mathcal{N}$ **do in parallel**
2 $\quad p(f_i) \leftarrow m$
3 $\quad$ /* Obtain the mth optimal $\mathbf{\Phi}_{ij}$ by (11) */
4 $\quad \mathbf{\Phi}_{ij}^{(m)} \leftarrow \frac{s_{ij}}{\sum_k 1/A_{k,k}^{(m)}} + T^{\star}\left(p(f_i)\right)$
5 **end for**
6 $p^{\star}(f_i) \leftarrow \arg\min_{m \in \mathcal{N}} \mathbf{\Phi}_{ij}^{(m)}$
7 Calculate $z_{ij}^{\star}$ by (10) and (12) with $\mathbf{A} = \mathbf{A}^{(p^{\star}(f_i))}$

ALGORITHM

Dynamic Programming-Based Embedding

将OSM和动态规划相结合，得到算法DPE。
该算法循环从非入口函数出发。$\forall f_j \in \mathcal{F} \backslash \mathcal{F}_{entry}$，DPE首先将其部署 $p(f_j)$ 作为边缘服务器n(line 3)，然后，从第四行到第十二行，对于函数对 $(f_i, f_j), e_{ij} \in \mathcal{E}$ DPE通过调用OSM求解子问题 $P_{sub}^{(i)}$ 作为返回值。如果 $p^*(f_i)$ 已经提前确定，DPE将会跳过 f_i 并直接处理下一个前驱 f_i' (line 5 ~ line 7)。最后DPE根据13行 $\left\{P_{sub}^{(i)}\right\}_{\forall e_{ij} \in \mathcal{E}}$ 的计算结果更新函数 f_j 的结束时间。
当所有函数对应的结束时间被计算得出之后，DAG的**全局最优任务执行时间**可由下式计算得出：

$$\max_{f \in \mathcal{F}_{exit}} \operatorname{argmin}_{p^*} T^*\left(p^*(f)\right)$$

每个函数的最优嵌入可以从 z^* 和 p^* 中得出。

Algorithm 2. DP-Based Embedding (DPE)

Input: $\mathcal{G}$ and $(\mathcal{F}, \mathcal{E})$
Output: Optimal value and corresponding solution

```
1  for j = |F_entry| + 1 to |F| do
2    for each n ∈ N do
3      p(f_j) ← n // Fix the placement of f_j
4      for each f_i ∈ {f_i | e_ij exists} do
5        if p*(f_i) has been decided then
6          continue
7        end if
8        if f_i ∈ F_entry then
9          ∀p(f_i) ∈ N, update T*(p(f_i)) by (2)
10       end if
11       Obtain the optimal Φ*_ij, p*(f_i), and z*_ij by calling OSM
12     end for
13     Update T*(p(f_j)) by (7)
14   end for
15 end for
16 return max_{f∈F_exit} arg min_{p*} T*(p*(f)), z*, and p*
```

ALGORITHM

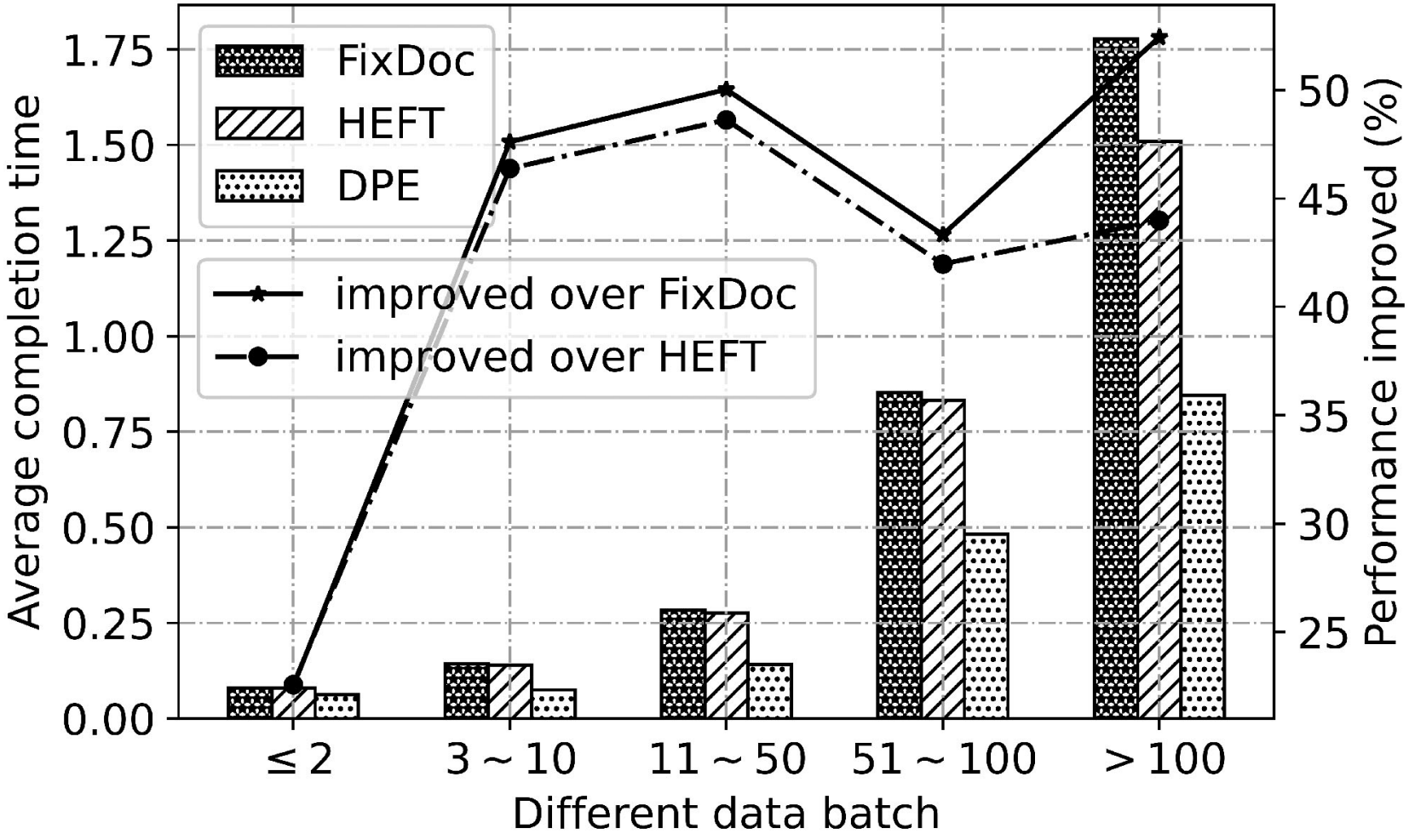
Extending to Multiple DAGs

DPE可扩展到多个DAG执行。

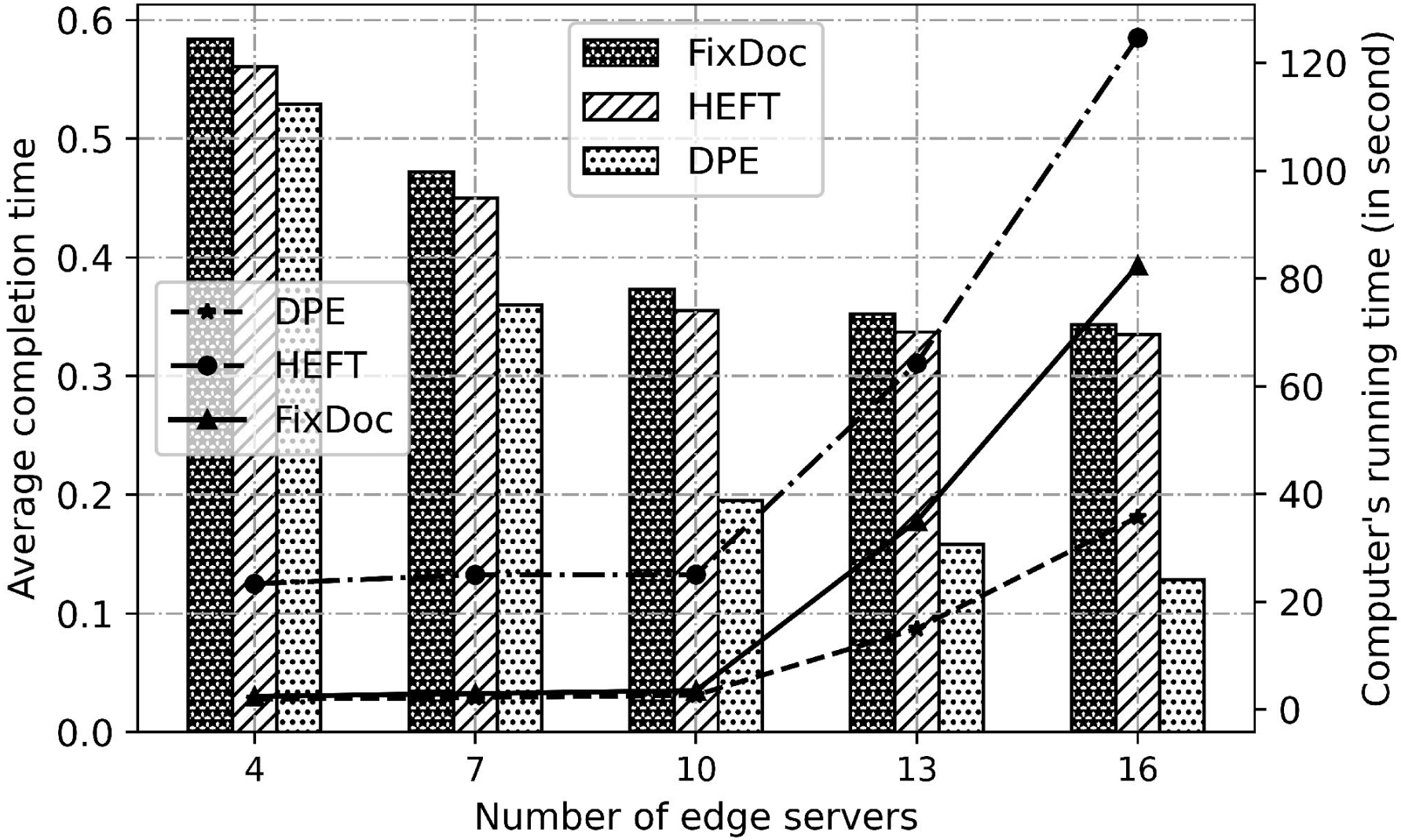
要做到这一点，只需将所有的DAG连接到一个增强的DAG中。

特别地，如果一个DAG有多于一个的入口函数，我们添加一个虚拟的头函数 f_h 和若干条有向边 $\{e_{hi}|\forall f_i \in \mathcal{F}_{entry}\}$，这样的函数$f_h$所需的浮点操作次数为零，且所有流经简单路径$s_{hi}$的数据量大小为零。同样，如果一个DAG有多于一个的出口函数，我们添加一个伪尾函数和相应的有向边。在此基础上，我们在DAG之前的（虚拟）尾函数和之后的DAG的（虚拟）头函数之间添加了一条有向边。将增广的DAG作为DPE的输入，得到嵌入结果。

EVALUATION


FixDoc
HEFT
DPE
improved over FixDoc
improved over HEFT
Average completion time
Performance improved (%)
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
50
45
40
35
30
25
≤ 2
3 ~ 10
11 ~ 50
51 ~ 100
> 100
Different data batch

EVALUATION


FixDoc
HEFT
DPE
DPE
HEFT
FixDoc
Average completion time
Computer's running time (in second)
Number of edge servers
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
20
40
60
80
100
120
4
7
10
13
16

EVALUATION

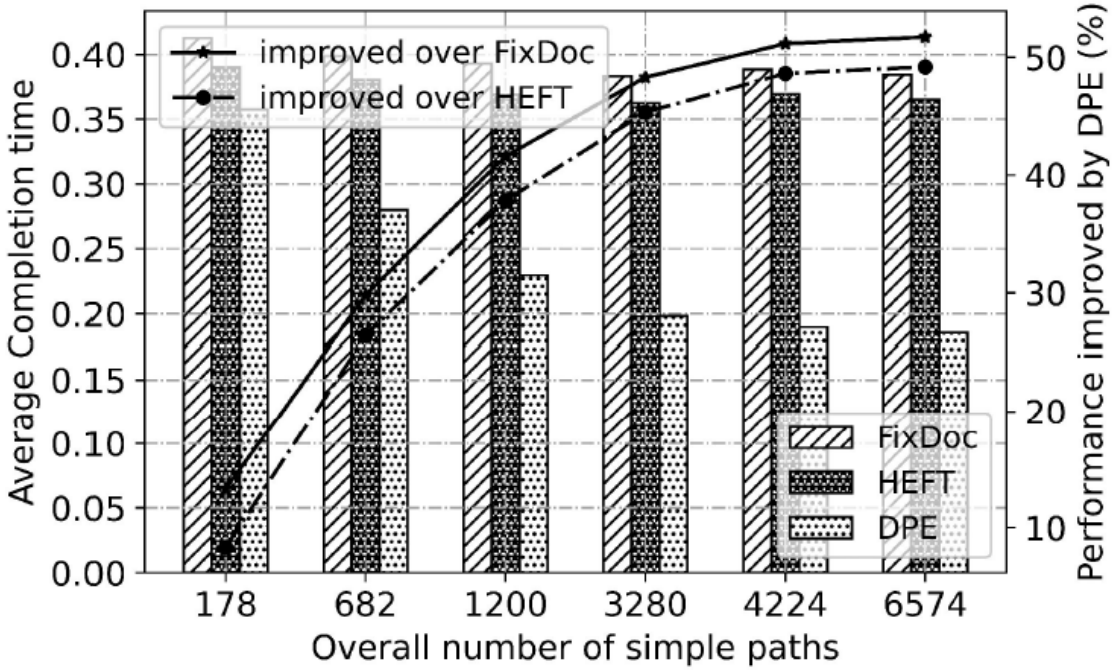


Fig. 10. Average completion time under different sparsities of $\mathcal{G}$.

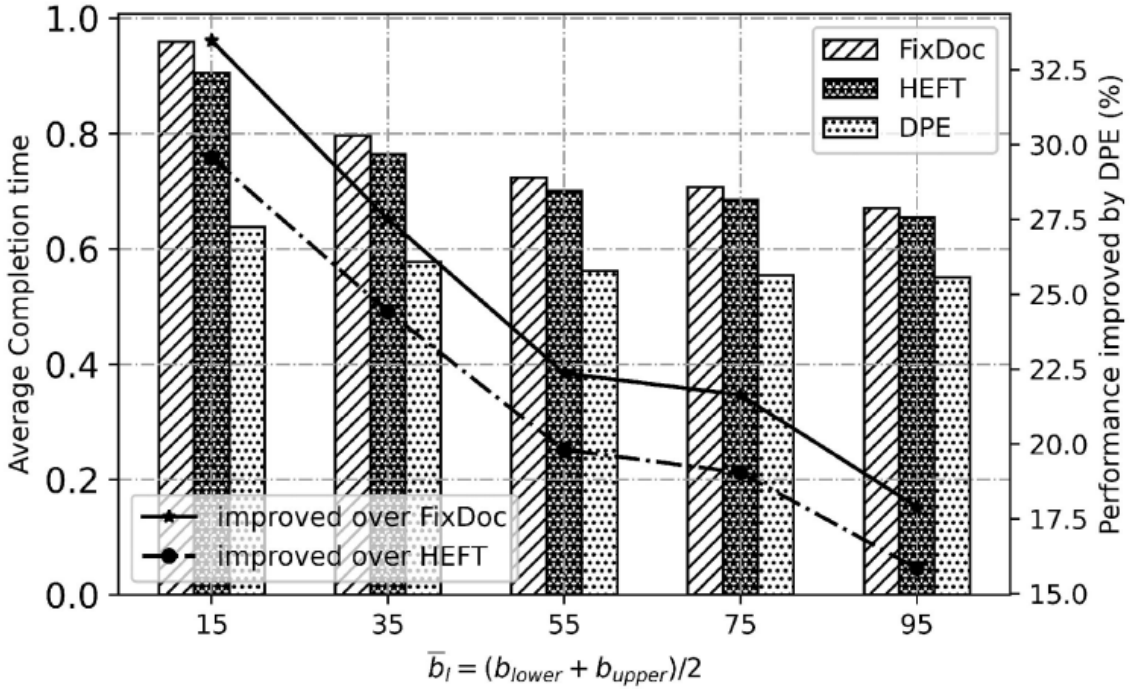


Fig. 12. Average completion time under different throughputs of links.

CONCLUSION

- 对无服务器边缘计算中的最优函数嵌入问题开展了研究
- 首创性提出将函数部署策略和流量映射与数据拆分结合考虑，有效缩短应用程序的执行时间
- 通过所提出的DPE算法从理论上验证最优化策略的可行性

- 只考虑离线场景，未考虑在线场景
- 未考虑到函数的并发执行的场景
- ~~做好向微服务部署策略场景的迁移适配~~

Thank you !