邊緣計算中的最佳化筆記

一、問題背景與挑戰

在階層式邊緣運算架構中，有效管理以下項目極具挑戰：

- 計算排程（Scheduling）
- 資料移動（Data Movement）
- 儲存與快取策略（Caching）
- 延遲與吞吐量的權衡

特別是在 JavaScript 等高階語言中，這些挑戰更為顯著。

核心難題：

- 計算安排與位置選擇
- 資料擷取與轉送策略
- 快取機制設計
- 吞吐量與延遲之間的折衷與最適化

二、兩種代表性解法架構

1. DECO（Distributed Edge COmputing）

- 方法類型：封包導向（Packet-based）
- 目標：聯合最佳化計算位置選擇、請求轉送、資料移動與快取決策
- 特色：強調吞吐量最優（Throughput Optimality）
- 設計者：Kashar Kamran
- 吞吐量指能完成多少筆計算任務（非傳統封包數）

2. LOAM

- 方法類型：流程導向（Flow-based）
- 特色：直接最小化延遲（Latency Optimality）
- 支援更廣泛的快取功能，包含計算結果快取
- 使用 M/M/1 queue 模型分析與優化延遲表現

三、DECO 詳細架構與運作邏輯

運作流程：

- 節點收到請求，但資料在遠端。
- 可以選擇轉送至資料源執行，或節點取得資料後本地執行。
- 執行後資料可快取，降低未來延遲。

核心最佳化挑戰：

- 計算請求轉發、計算位置安排、資料擷取與傳送、快取決策
- 同時平衡計算、通訊與儲存資源

四、系統模型與最佳化策略（DECO）

1. 佇列動態模型

- 計算需求佇列 Y_{n,m,k}(t)：節點 n 對任務 m 和資料類型 k 的需求
- 資料需求佇列：包含資料轉送與快取限制

2. 策略設計原則

計算部署（Computation Placement）

- 每筆請求為 (m, k)
- 排序依據：computation interest - data interest 差值最大者先執行

傳輸排程（Transmission Scheduling）

- 鏈路容量需分配給計算結果與資料傳送
- 使用 normalized backlog 衡量壓力 → 優先處理壓力最大者

快取策略（Caching Policy）

- 根據資料需求數排序，快取前 L 個
- 適用於常用模型或重複資料請求

五、LOAM：延遲最小化設計

設計核心：

- 採流程導向模型（Flow-based）
- 使用 M/M/1 queue 建模延遲
- 延遲作為直接最佳化目標

特性比較：

- 模型：DECO 為封包導向，LOAM 為流程導向
- 延遲：DECO 間接改善，LOAM 直接最小化
- 快取：DECO 為資料快取，LOAM 包含計算結果快取

最佳化問題與解法：

- 非凸最佳化，含混合整數變數
- 結構：次模函數 + 凹函數

GCFW（離線解）

- Gradient-Combining Frank-Wolfe
- 須知所有需求參數
- 理論效能保證比 1/2

GP（線上解）

- Gradient Projection
- 可即時調整，無需預知參數
- 雖無理論保證，但實驗效果佳

六、模擬與實驗結果

測試拓撲：Abilene（Internet2）、GEANT、Falk topology

結果總結：

- GCFW 延遲最低
- GP 效果次佳
- DECO 在處理量與穩定性方面表現卓越

七、總結與應用前景

成功系統需整合：

- 計算部署、資料傳輸、快取決策

應用前景：

- 面向 5G/6G、XR、AR/VR 等高需求應用
- 高延遲敏感場景（<10ms）
- 更需動態、分散、可擴展的邊緣運算管理策略