EAIS: Energy-aware adaptive scheduling for DNN inference on high-performance GPU

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

姚春荣

提 纲

研究背景

- **深度学习推理**
  - 低延迟，高并发
  - 具有复杂的应用要求
  - GPU能耗高

如何在推理阶段降低能耗?

提 纲

1 研究背景

2 研究动机

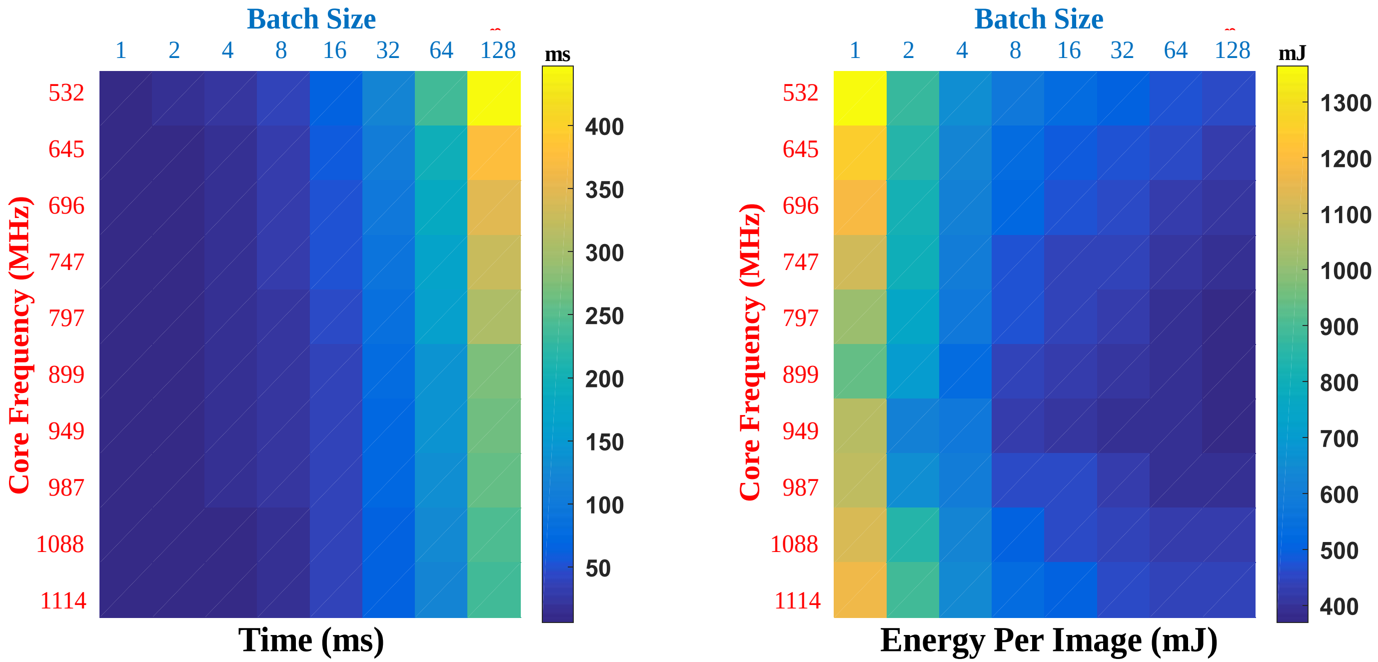
3 调度算法

4 实验结果

5 总结

研究动机

- **Batching & DVFS**
  - Batching提高吞吐量 (Clipper，Nanily)
  - DVFS降低GPU核心频率 (PIT)



如何协调batching和DVFS使得满足延迟要求的同时降低能耗?

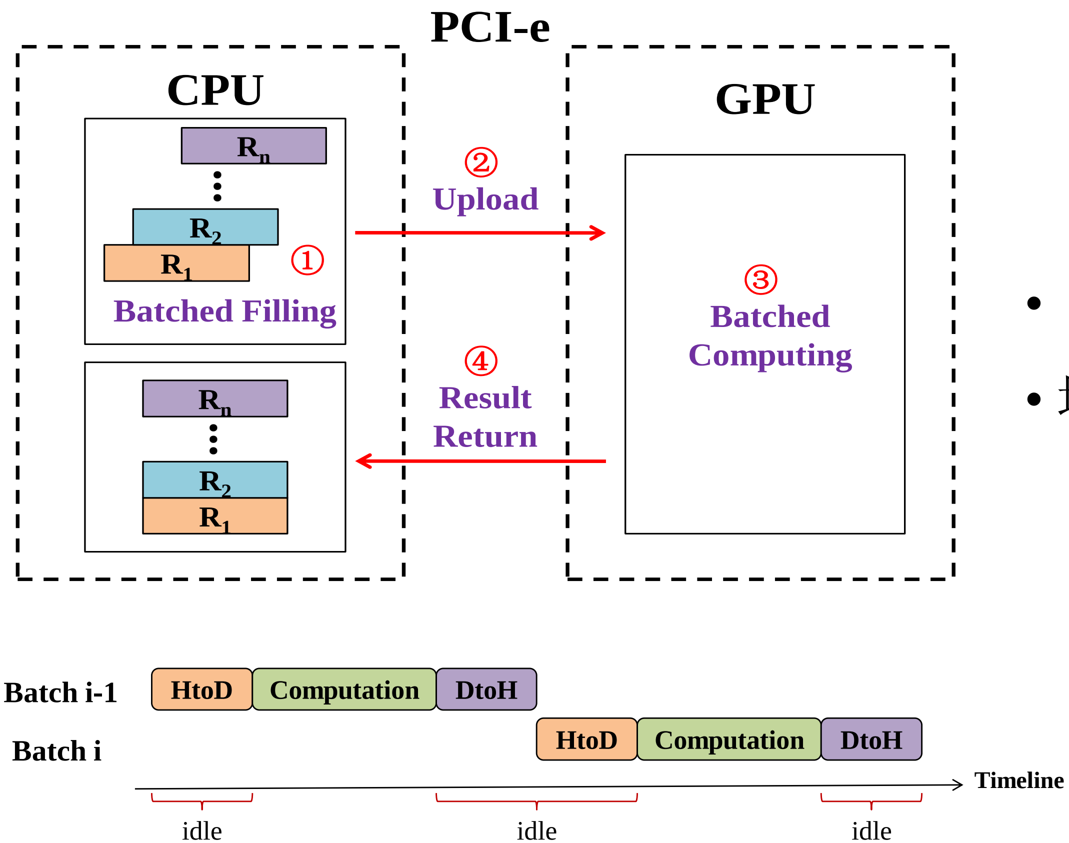
研究动机

- **挑战**
  - 配置空间大
    - 离线状态下考虑batch size和core frequency的多种组合。

      （GPU V100具有187个core frequency级别。计算能力达到15.7 TFLOPS，这就使得ResNet-50的batch size最大可达到1024）
    - 不同GPU和DNN表现出的性能不同。
  - 负载波动
    - 在实际推理服务中，请求到达率是随机且突发。

研究动机

➢ batching机制



- GPU没有被充分利用；
- 增大GPU固定的能耗开销；

研究动机

- 目的

在推理调度阶段，满足延迟目标Service-Level Objectives (SLOs)，并最小化推理能耗。

$$\min \sum_{i=1} E_{batch}{}^{[i]} \quad \textbf{s.t.} \quad L^{[i]} \le L_{SLO}, i \in \mathbb{N}_+$$

- 方法

提出一种能量感知的自适应调度框架EAIS。EAIS由DNN inference profiler，asynchronous execution strategy，energy-aware scheduler组成。

提 纲

1 研究背景

2 研究动机

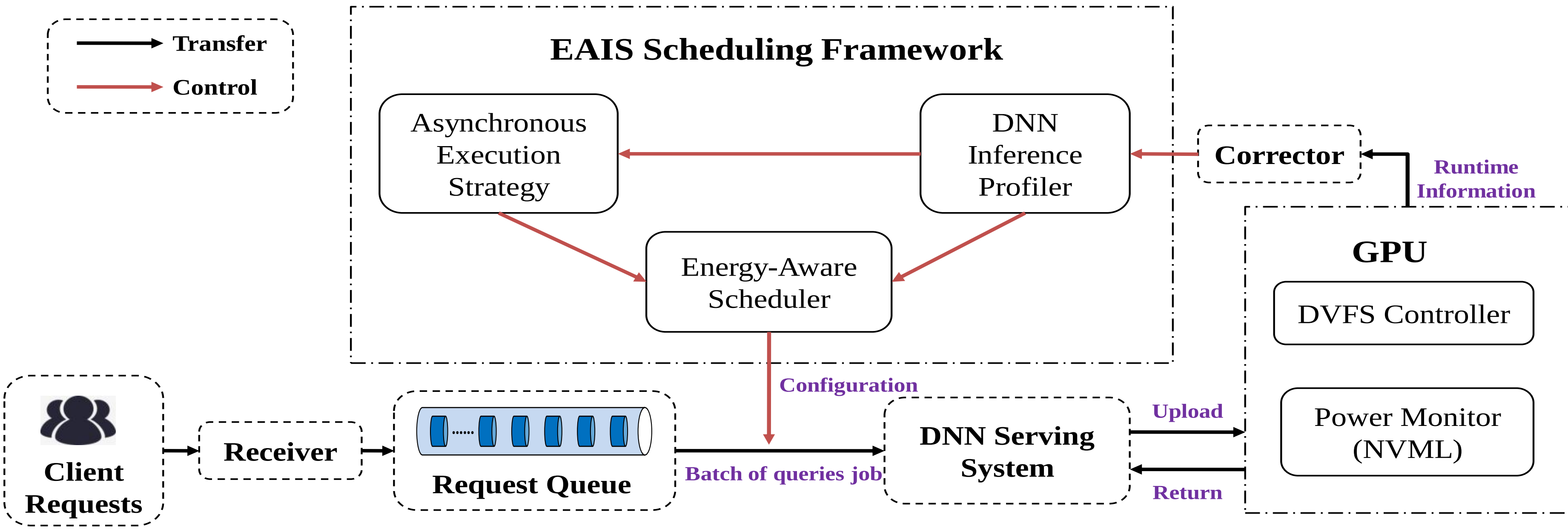
3 调度算法

4 实验结果

5 总结

调度算法

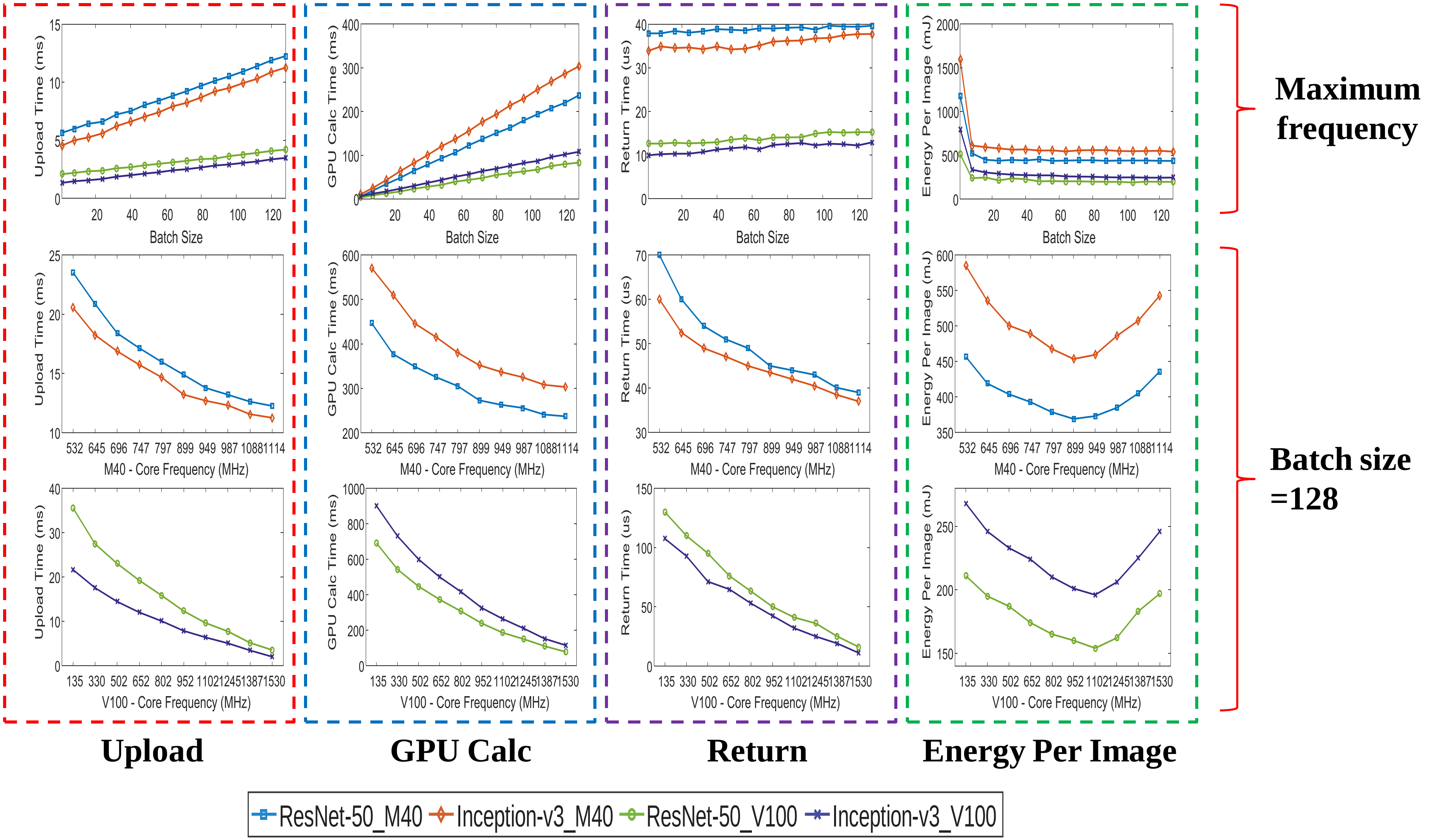
● 调度框架



- DNN Inference Profiler：提供关于DNN模型的性能特征的有效信息。
- Asynchronous Execution Strategy：使数据上传与GPU计算重叠。
- Energy-Aware Scheduler：自适应批调度，使能耗最低。

调度算法

● **DNN Inference Profiler**



调度算法

Algorithm 1 Using the fitting method to complete the data

Input:

The sampling data of upload, GPU calculation, and energy per image, $\tilde{T}_{upload}$, $\tilde{T}_{calculate}$, $\tilde{E}_{image}$;

Output:

Fitted value, $\hat{T}_{upload}$, $\hat{T}_{calculate}$, $\hat{E}_{image}$;

Energy efficiency, $\hat{EE}$;

for each $b_i \in B$ **do**

  Upload, GPU calculation and energy discrete data on core frequencies, $\tilde{T}_{upload,i}(f)$, $\tilde{T}_{calculate,i}(f)$, $\tilde{E}_{image,i}(f)$;

  $\tilde{T}_{upload,i}(f)$ and $\tilde{T}_{calculate,i}(f)$ are fitted by Rational function curve, $\hat{T}_{upload,i}(f)$, $\hat{T}_{calculate,i}(f)$;

  $\tilde{E}_{image,i}(f)$ is fitted by Fourier curve, $\hat{E}_{image,i}(f)$;

end for

Update F;

for each $f_j \in F$ **do**

  Upload, GPU calculation and energy discrete data on batch sizes, $\tilde{T}_{upload,j}(b)$, $\tilde{T}_{calculate,j}(b)$, $\tilde{E}_{image,j}(b)$;

  $\tilde{T}_{upload,j}(b)$ and $\tilde{T}_{calculate,j}(b)$ are fitted by least squares curve, $\hat{T}_{upload,j}(b)$, $\hat{T}_{calculate,j}(b)$;

  $\tilde{E}_{image,j}(b)$ is fitted by Power function curve, $\hat{E}_{image,j}(b)$;

end for

- 在给定的batch size时

有理数曲线拟合

傅里叶曲线拟合

- 在给定的核心频率时

最小二乘曲线拟合

幂函数曲线拟合

调度算法

- **Asynchronous Execution Strategy**

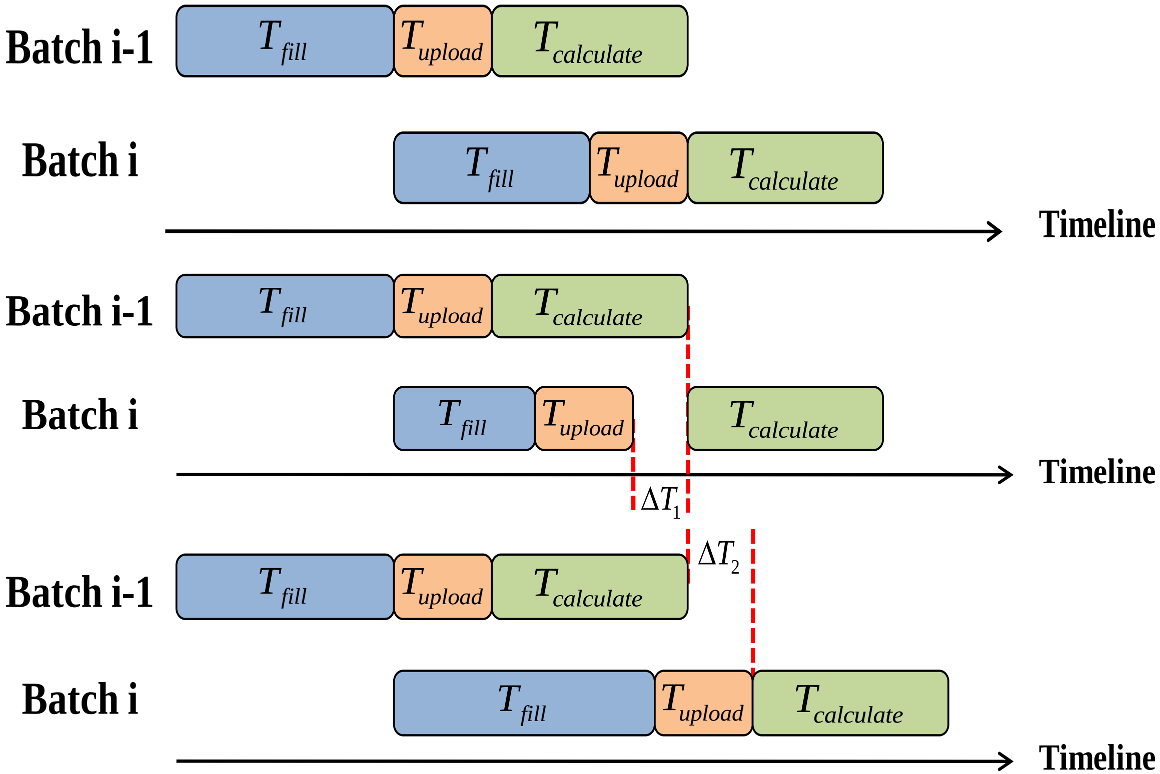
  - batch 填充

$$T_{fill} = \sum_{k=1} \frac{n_k}{R_{request,k}} \qquad b = \sum_{k=1} n_k$$

  - 请求速率稳定时

$$T_{fill} = \frac{b}{R_{request}}$$

调度算法



$$L^{[i]} = \begin{cases} T_{fill}{}^{[i]} + T_{upload}{}^{[i]} + T_{calculate}{}^{[i]}, i = 1 \\ \max[(T_{upload}{}^{[i-1]} + T_{calculate}{}^{[i-1]}), (T_{fill}{}^{[i]} + T_{upload}{}^{[i]})] + T_{calculate}{}^{[i]}, i \geq 2 \end{cases}$$

$$f_{calculate}{}^{[i-1]} = f_{upload}{}^{[i]}$$

调度算法

➢ 满足延迟要求的最大请求速率和batch size

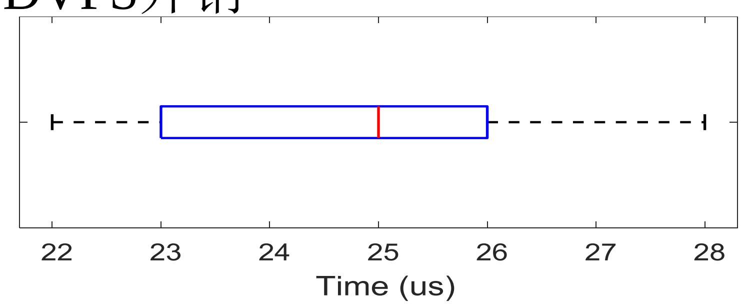
$$\begin{cases} T_{fill} \geq T_{calculate} \\ T_{fill} + T_{upload} + T_{calculate} \leq L_{SLO} \end{cases}$$

其中，

$$\begin{cases} T_{fill} = \dfrac{b}{R_{request}} \\ T_{upload} = \alpha_{upload} b + \beta_{upload} \\ T_{calculate} = \alpha_{calculate} b + \beta_{calculate} \end{cases}$$

$$\Rightarrow R_{request} \leq \frac{L_{SLO} - \beta_{upload} - 2\beta_{calculate}}{(L_{SLO} - \beta_{upload})\alpha_{calculate} + \alpha_{upload}\beta_{calculate}} \qquad b \leq \frac{L_{SLO} - \beta_{upload} - 2\beta_{calculate}}{\alpha_{upload} + 2\alpha_{calculate}}$$

➢ DVFS开销



DVFS的时间开销为23~26μs

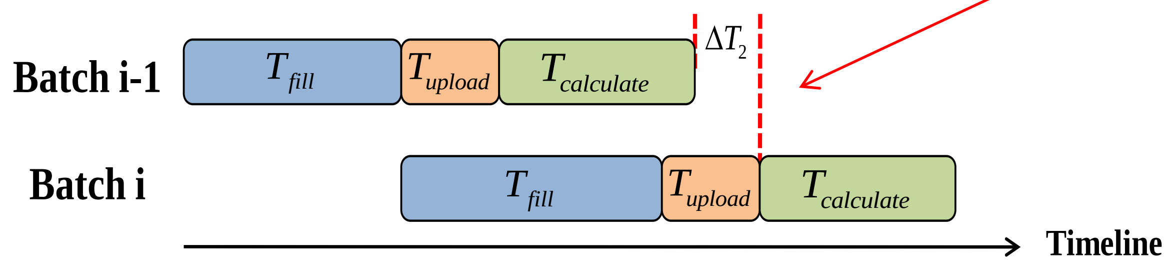
GPU M40的核心频率从532MHz变成595MHz

调度算法

- **Energy-Aware Scheduler**

目标： $\min \sum_{i=1} E_{batch}{}^{[i]}$ **s.t.** $L^{[i]} \le L_{SLO}, i \in \mathbb{N}_+$

$$E_{batch}{}^{[i]} = \begin{cases} E_{image}{}^{[i]} b^{[i]} \\ E_{image}{}^{[i]} b^{[i]} + (T_{fill}{}^{[i]} + T_{upload}{}^{[i]} - T_{upload}{}^{[i-1]} - T_{calculate}{}^{[i-1]}) P_{static} \end{cases}$$



贪心标准： $EE = \dfrac{Throughput}{\bar{P}}$ 其中， $\begin{cases} Throughput = \dfrac{b}{t} \\ \bar{P} = \dfrac{E_{batch}}{t} \end{cases} \Rightarrow EE = \dfrac{b}{E_{batch}} = \dfrac{1}{E_{image}}$

调度算法

Algorithm 2 Adaptive-batching scheduling

Input:

Request queue, q;

Latency SLO, L_{SLO};

Output:

Total energy consumption, E_{total};

batch index, $i = 1$;
while True **do**
  if $i = 1$ **then**
    $b^{[i]}, f^{[i]} = \underset{b,f}{argmax}(EE^{[i]}) s.t. T_{fill}{}^{[i]} + T_{upload}{}^{[i]} + T_{calculate}{}^{[i]} \leq L_{SLO}$;

批次i=1

  else
    $b^{[i]}, f^{[i]} = \underset{b,f}{argmax}(EE^{[i]}) s.t. max[(T_{upload}{}^{[i-1]} + T_{calculate}{}^{[i-1]}), (T_{fill}{}^{[i]} + T_{upload}{}^{[i]})] + T_{calculate}{}^{[i]} \leq L_{SLO}$;

批次i>1

    if $b^{[i]} < 1$ **then**
      return 0;
    else if $b^{[i]} < len(q)$ **then**
      $b^{[i]} = q.size()$;
      $f^{[i]} = \underset{f}{argmax} EE^{[i]}(b^{[i]} = q)$;
      return 1;

最后的批次

    end if
  end if
  $E_{batch} \leftarrow b^{[i]}, f^{[i]}$;
  $E_{total} += E_{batch}$;
  for $j = 1$ To $j \leq b^{[i]}$ **do**
    $q.pop()$;
  end for
  $i++$;
end while

提 纲

1 研究背景

2 研究动机

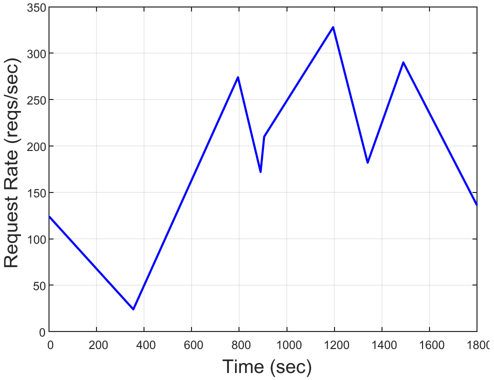
3 调度算法

4 实验结果

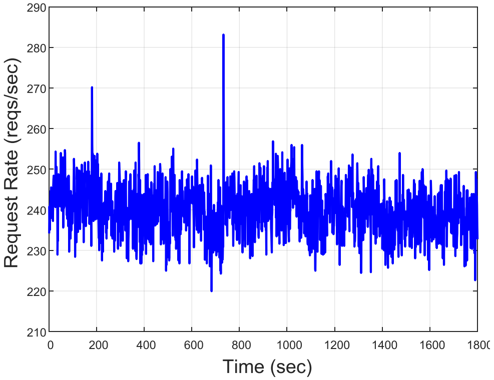
5 总结

实验结果

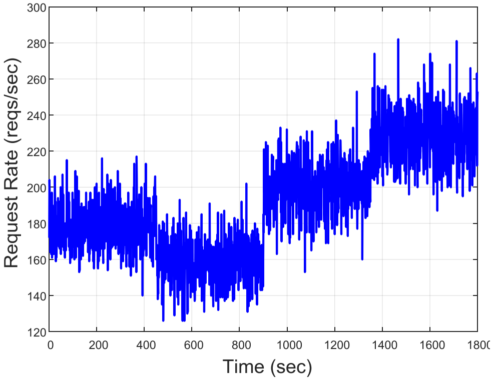
- 实验环境
  - **GPU:** M40, V100
  - **DNN Model:** ResNet-50, Inception-v3
  - **Platform:** TensorRT
  - **SLO setting**: 200~500ms (200为默认值)
  - **Workload:** SogouQ, WiKi, Dynamic



SogouQ



WiKi



Dynamic

实验结果

● 满足SLO同时降低能耗

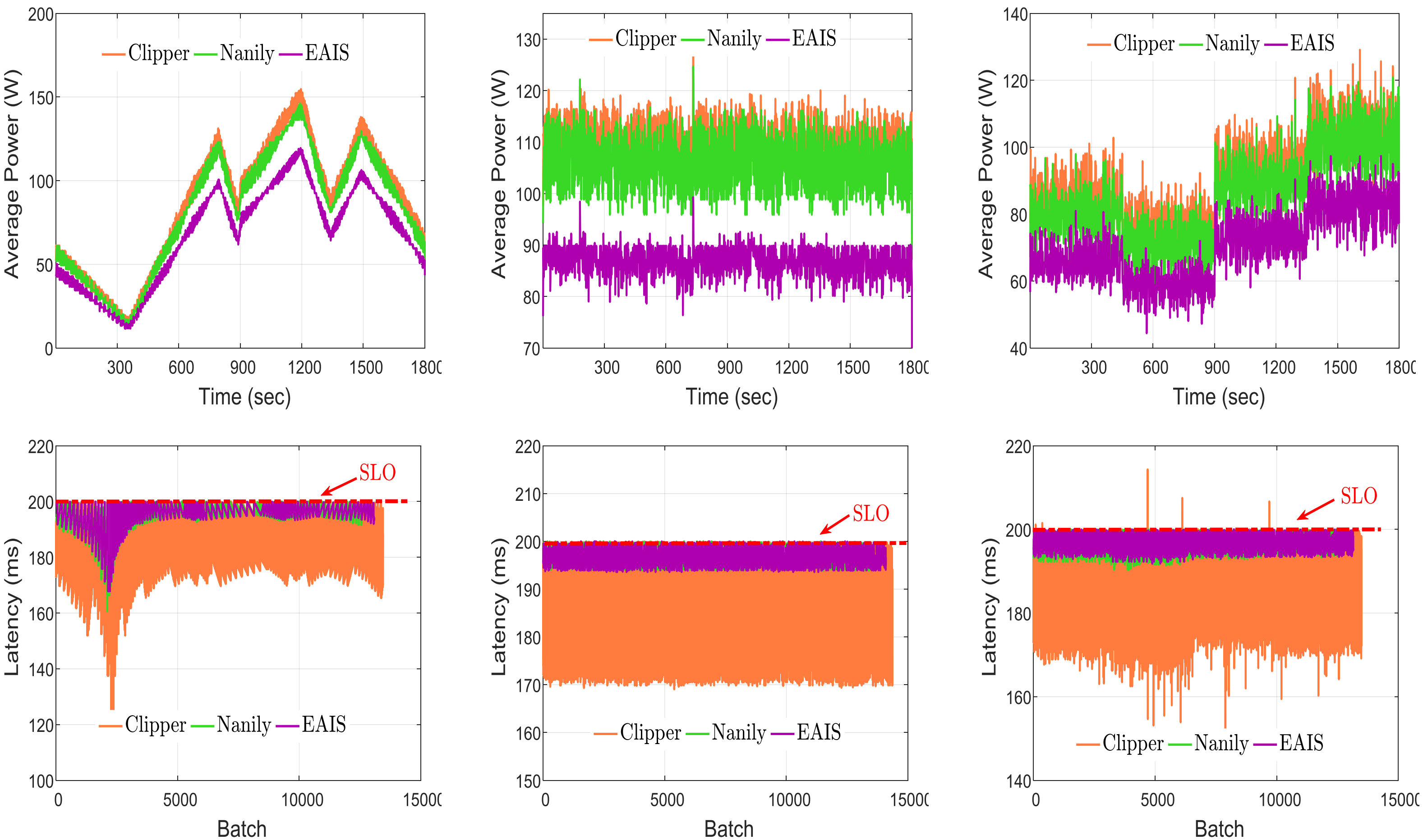
Table The total energy consumption (x10^4J) under different workloads.

GPUs	Networks	SogouQ			WiKi			Dynamic		
		Clipper	Nanily	**EAIS**	Clipper	Nanily	**EAIS**	Clipper	Nanily	**EAIS**
M40	ResNet-50	15.647	14.846	**12.201**	20.077	19.070	**15.669**	16.183	15.366	**12.629**
	Inception-v3	20.247	19.205	**15.204**	26.004	24.697	**19.553**	20.955	19.880	**15.739**
V100	ResNet-50	7.567	7.155	**5.447**	9.576	9.071	**6.907**	7.814	7.388	**5.625**
	Inception-v3	9.768	9.22	**7.202**	12.285	11.627	**9.086**	10.075	9.511	**7.432**

- EAIS可以降低总能耗17.82%~28.02%；
- EAIS
  - 低负载时，使用较低的核心频率降低能耗；
  - 高负载时，使用较高的核心频率减小GPU的计算时间；
- Clipper和Nanily使用最高核心频率；

实验结果

- ResNet-50, M40



Power

Latency

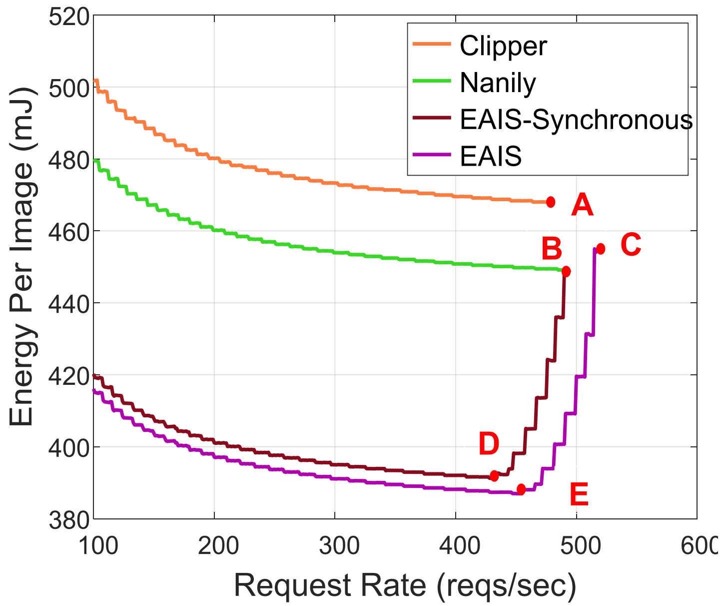
SogouQ

WiKi

Dynamic

实验结果

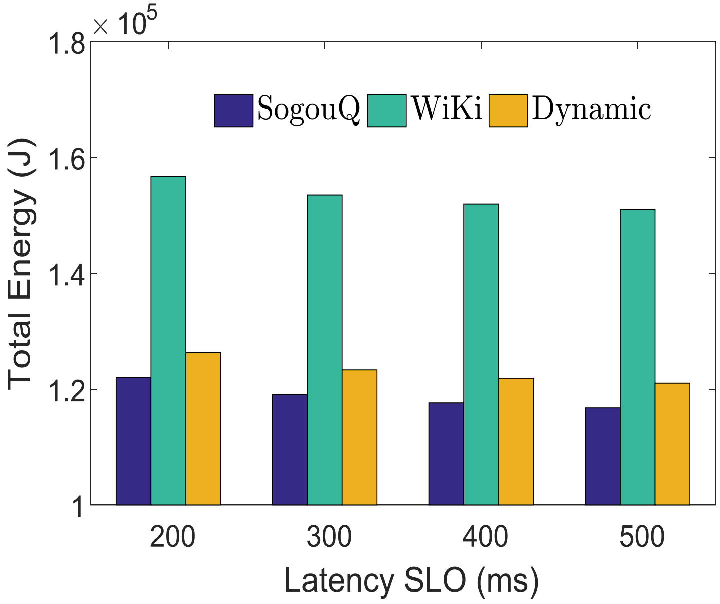
- 深入EAIS
  - ➢ 稳定请求速率&能耗



- 当请求速率较小时，四种调度算法的能耗趋势相似。EAIS节省能耗17.29% (Clipper)，13.68% (Nanily)，2% (EAIS- Synchronous)。
- 当请求速率较大时，与Clipper和Nanily相比，EAIS将吞吐量分别提高了8.33%和5.05%。
- Nanily与EAIS-Synchronous的最大吞吐量是一致的 (点B)。

实验结果

● SLO对能耗的影响



- 延迟为200ms的能耗最高，500ms的能耗最低。
- 随着延迟SLO的增大，能耗先降低然后趋于平缓。

提 纲

1 研究背景

2 研究动机

3 调度算法

4 实验结果

5 总结

总结

- EAIS可以将能耗降低多达28.02%，同时满足延迟SLO；
- EAIS在不同的延迟SLO约束下也具有良好的通用性。

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

谢谢！