

RETAIL: OPTING FOR LEARNING SIMPLICITY TO ENABLE QoS-AWARE POWER MANAGEMENT IN THE CLOUD

Shuang Chen,* Angela Jin,** Christina Delimitrou, José F. Martínez

Cornell University

*Currently with Shuhai Lab at Huawei Cloud

** Currently with UC Berkeley

汇报人：林捷





- 定义在尾延迟上的 QoS(Quality of Service)约束 (e.g., 99th percentile)



- QoS defined in tail latency (e.g., 99th percentile)



Happy == (Latency ≤ 1s)

	0.1s	Happy
	0.1s	Happy
	0.1s	Happy
	1.9s	Angry
	1.9s	Angry
	1.9s	Angry

Average: 1s

99th percentile: 1.9s

- QoS defined in tail latency (e.g., 99th percentile)



Happy == (Latency ≤ 1s)

	0.1s	Happy
	0.1s	Happy
	0.1s	Happy
	1.9s	Angry
	1.9s	Angry
	1.9s	Angry

Average: 1s
99th percentile: 1.9s


	1s	Happy
	1s	Happy
	1s	Happy
	1s	Happy
	1s	Happy
	1s	Happy

Average: 1s
99th percentile: 1s

- QoS defined in tail latency (e.g., 99th percentile)



Happy == (Latency ≤ 1s)

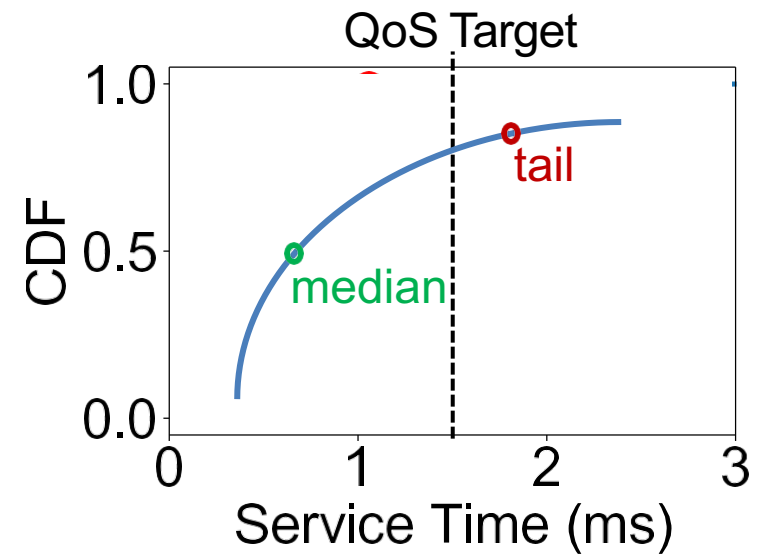
	0.1s	Happy
	0.1s	Happy
	0.1s	Happy
	1.9s	Angry
	1.9s	Angry
	1.9s	Angry

Average: 1s
99th percentile: 1.9s

	1s	Happy
	1s	Happy
	1s	Happy
	1s	Happy
	1s	Happy
	1s	Happy

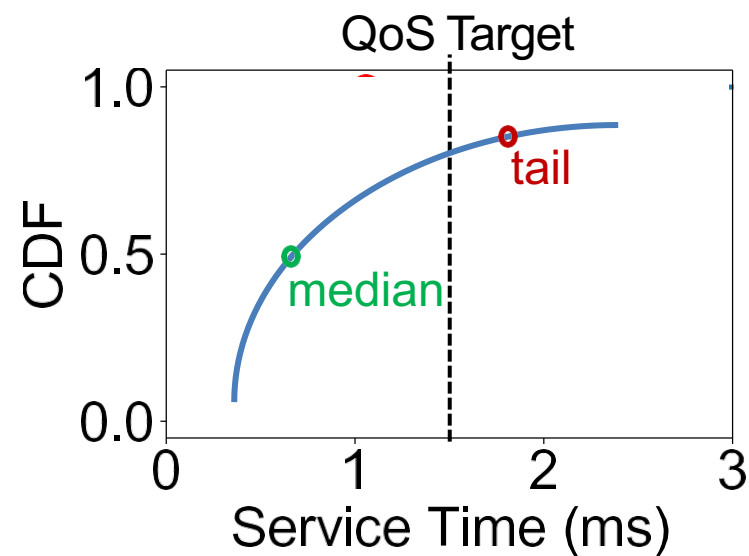
Average: 1s
99th percentile: 1s





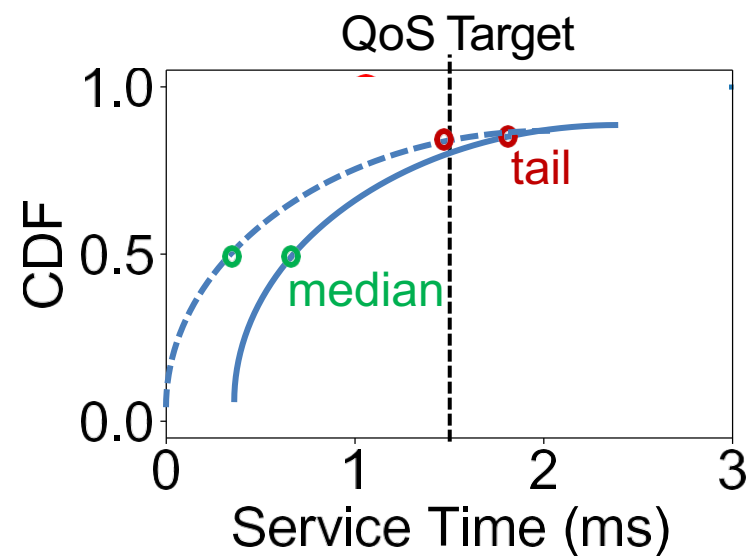
■ 应用程序级资源管理 (Application-level resource management)

- 传统的资源管理器将每个应用程序作为一个整体进行管理



■ 应用程序级资源管理 (Application-level resource management)

- 传统的资源管理器将每个应用程序作为一个整体进行管理

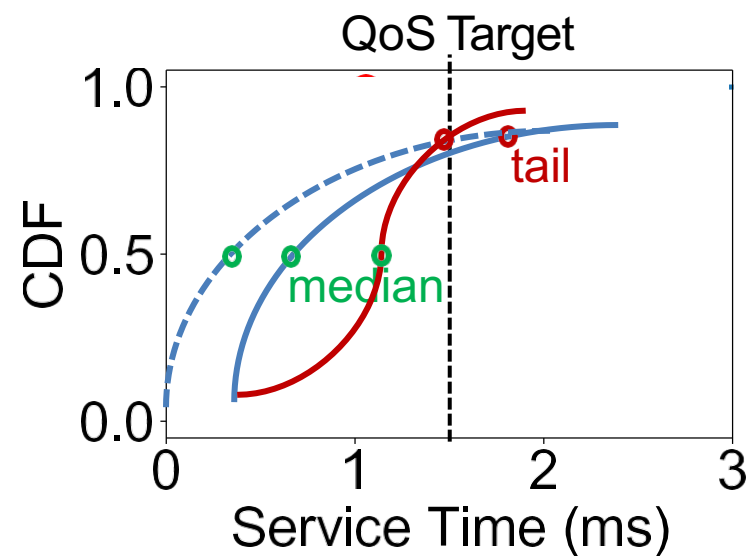


■ 应用程序级资源管理 (Application-level resource management)

- 传统的资源管理器将每个应用程序作为一个整体进行管理

■ 请求级资源管理 (Request-level resource management)

- 使每个请求都满足QoS
 - » 为运行长请求的核心分配高频率
 - » 为运行短请求的核心分配低频率
- 更高的资源/电源效率



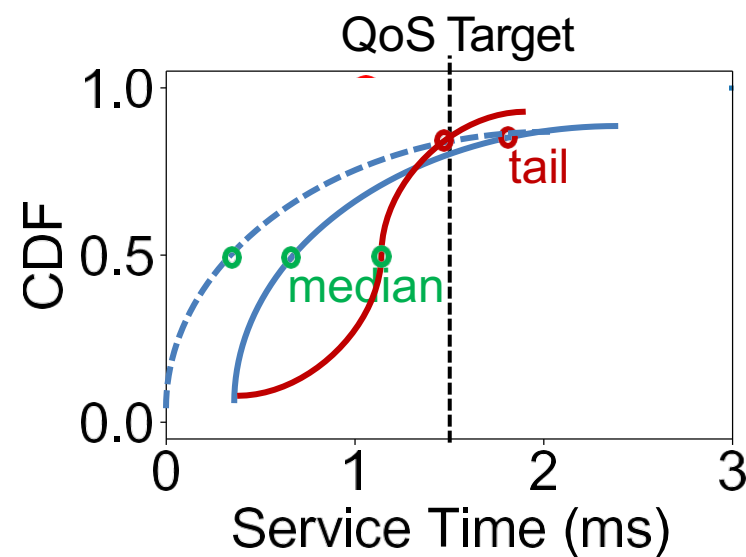
■ 应用程序级资源管理 (Application-level resource management)

- 传统的资源管理器将每个应用程序作为一个整体进行管理

■ 请求级资源管理 (Request-level resource management)

- 使每个请求都满足QoS
 - » 为运行长请求的核心分配高频率
 - » 为运行短请求的核心分配低频率
- 更高的资源/电源效率

■ 如何去区分请求是长还是短呢？



- **Adrenaline [MICRO'15]:** feature-driven
 - » E.g., if request type is SET, increase frequency
 - ☹ 为特定应用程序的手动选择特征
 - ☹ 无法细区分同一类别中的请求
- **Gemini [MICRO'20]:** feature-driven, neural-network-based
 - » Predicted latency > QoS, increase frequency
 - ☹ 为网站搜索手动选择特征

- **Adrenaline [MICRO'15]:** feature-driven
 - » E.g., if request type is SET, increase frequency
 - ☹ 为特定应用程序的手动选择特征
 - ☹ 无法细区分同一类别中的请求
- **Gemini [MICRO'20]:** feature-driven, neural-network-based
 - » Predicted latency > QoS, increase frequency
 - ☹ 为网站搜索手动选择特征

可否在**LC**应用上实现**通用**的请求延迟预测？

Application	Masstree	ImgDNN	Sphinx	Xapian	Moses	Shore	Silo
Domain	Key-value store	Image recognition	Speech recognition	Web search	Real-time translation	Database (disk/SSD)	Database (in-memory)
Dataset	One million <key,value> pairs	MNIST [21]	CMU AN4 [11]	English Wikipedia	Spanish articles [6]	TPC-C [16], 1 warehouse	
QoS Target	1ms	5ms	4s	8ms	120ms	5ms	1ms
Median:Tail Ratio	0.84	0.81	0.36	0.27	0.26	0.25	0.19
Request	90% <GET, key> 10% <PUT, key, value>	An image with a handwritten digit	Path to an audio file	A single-word term	A Spanish phrase to be translated into English	47% PAYMENT 45% NEW_ORDER 4% ORDER_STATUS 4% STOCK_LEVEL	
Classification	Little or no variation	Little or no variation	Predicted by request features	Predicted by application features	Predicted by request features	Predicted by request and application features	
Feature(s)	N.A.	N.A.	Audio file size	Document count	Word count	Request type, Item count, Rollback	

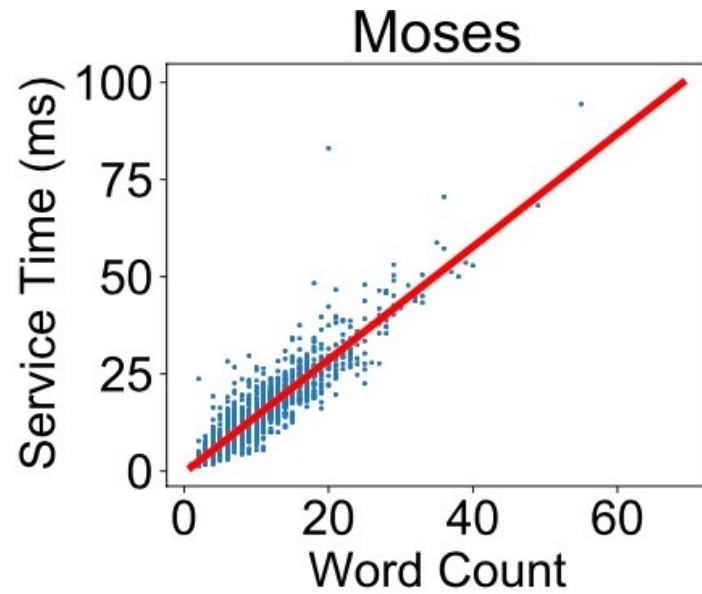
- 调查是否可以预测7种不同LC应用的延迟

Application	Masstree	ImgDNN	Sphinx	Xapian	Moses	Shore	Silo
Domain	Key-value store	Image recognition	Speech recognition	Web search	Real-time translation	Database (disk/SSD)	Database (in-memory)
Dataset	One million <key,value> pairs	MNIST [21]	CMU AN4 [11]	English Wikipedia	Spanish articles [6]	TPC-C [16], 1 warehouse	
QoS Target	1ms	5ms	4s	8ms	120ms	5ms	1ms
Median:Tail Ratio	0.84	0.81	0.36	0.27	0.26	0.25	0.19
Request	90% <GET, key> 10% <PUT, key, value>	An image with a handwritten digit	Path to an audio file	A single-word term	A Spanish phrase to be translated into English	47% PAYMENT 45% NEW_ORDER 4% ORDER_STATUS 4% STOCK_LEVEL	
Classification	Little or no variation	Little or no variation	Predicted by request features	Predicted by application features	Predicted by request features	Predicted by request and application features	
Feature(s)	N.A.	N.A.	Audio file size	Document count	Word count	Request type, Item count, Rollback	

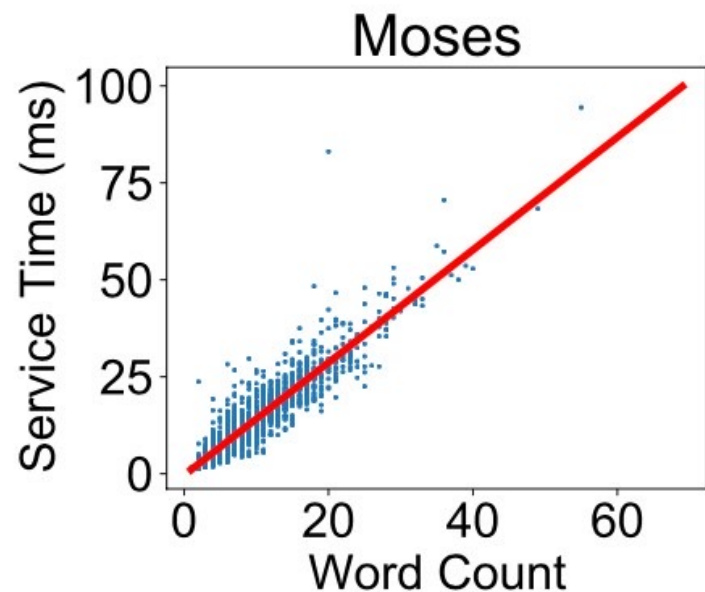
■ 调查是否可以预测7种不同LC应用的延迟

- 请求延迟（Request latency）= 服务时间 + 排队延时
- 查找与服务时间相关的特征

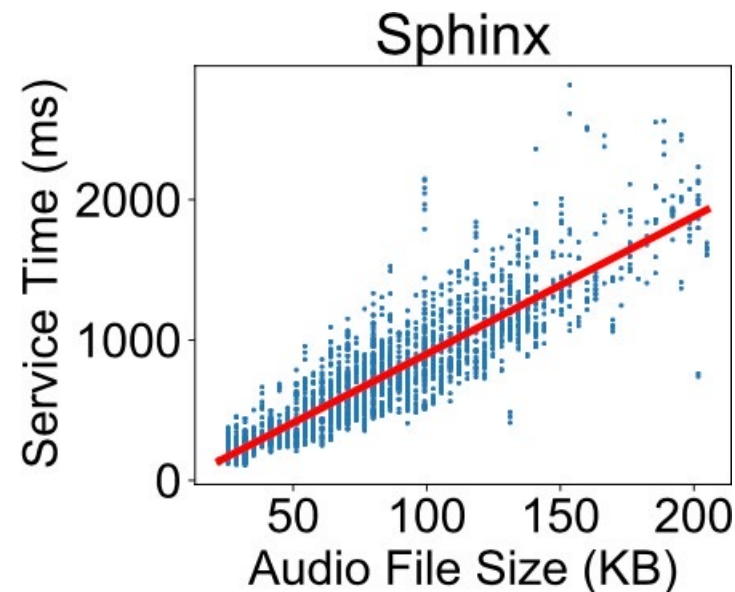
- **请求特征（Request features）**
 - 请求大小, 请求类型, 等.
 - 在请求 *到达* 时获得
- **应用特征（Application features）**
 - 中间变量
 - 在请求处理 *期间* 获得



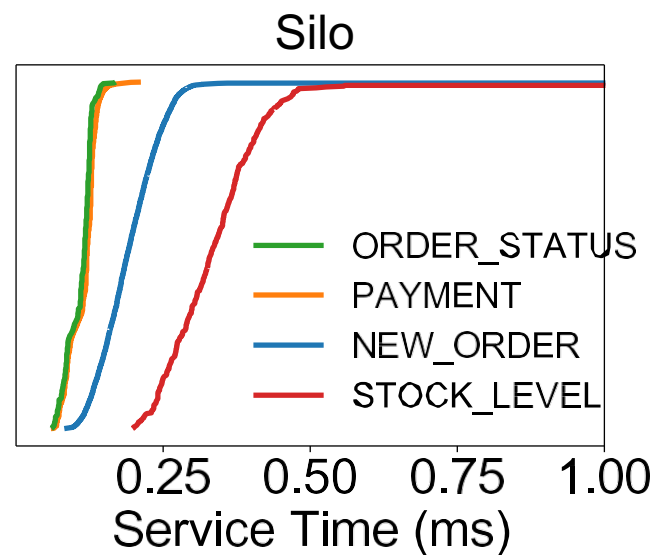
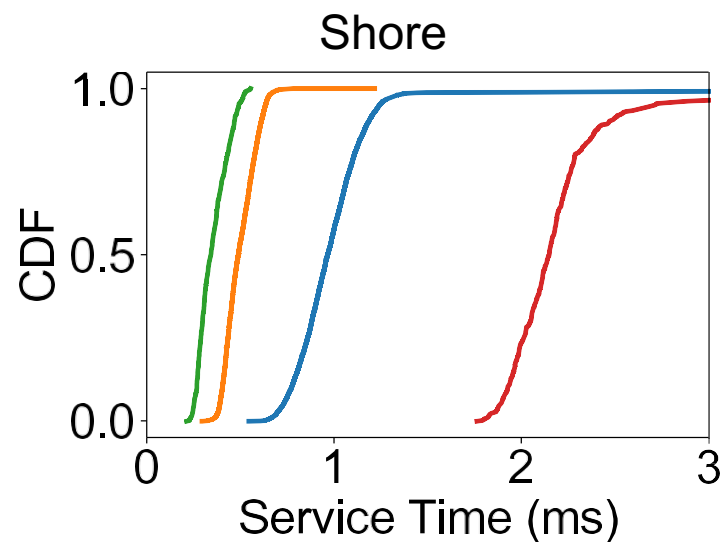
- Real-time translation
- Input request:
a Spanish phrase

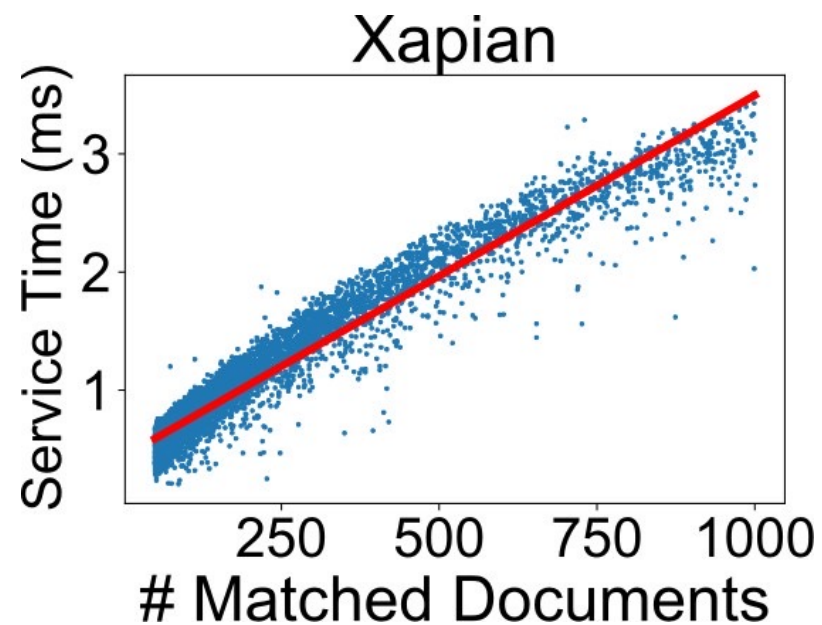


- Real-time translation
- Input request:
a Spanish phrase



- Speech recognition
- Input request:
a path to an audio file





- Web search
- Input: a search term

- 所有应用程序都具有与请求服务时间密切相关的直观特征（ *intuitive* features ）



- 所有应用程序都具有与请求服务时间密切相关的直观特征（ *intuitive* features ）
- 相关关系非常简单



- 所有应用程序都具有与请求服务时间密切相关的直观特征（ *intuitive features* ）
- 相关关系非常简单
- 将应用程序分为四类
 - 几乎没有变化: ImgDNN, Masstree
 - 根据请求特征预测: Moses, Sphinx
 - 应用特征预测: Xapian
 - 组合: Shore, Silo

- 所有应用程序都具有与请求服务时间密切相关的直观特征（*intuitive features*）
- 相关关系非常简单
- 将应用程序分为四类
 - 几乎没有变化：ImgDNN, Masstree
 - 根据请求特征预测：Moses, Sphinx
 - 应用特征预测：Xapian
 - 组合：Shore, Silo

我们可以为 *通用* LC应用程序构建一个 *简单有效* 的延迟预测模型！

- **ReTail: 请求级延迟预测以减少尾延迟**
- **具有请求级延迟预测的LC应用程序的QoS感知功率管理系统**
- **ReTail 特征选择**
 - 选择与请求服务时间最相关的特征
 - 适用于任何常规LC应用
- **ReTail 延迟预测**
 - 线性回归
- **ReTail QoS感知的功率管理**
 - 决定每个请求的最佳频率

- **Input: 一个日志，包含**
 - 用户提供的N个样本集
 - 每个请求示例的特征菜单
 - » 请求特征，如请求类型、请求大小等
 - » 应用程序中的潜在中间变量
 - 利用源代码中的跟踪和日志记录语句



- **Input:** 一个日志，包含
 - 用户提供的N个样本集
 - 每个请求示例的特征菜单
 - » 请求特征，如请求类型、请求大小等
 - » 应用程序中的潜在中间变量
 - 利用源代码中的跟踪和日志记录语句
- **Output:** 与请求服务时间最相关的最佳特征

- **Input:** 一个日志，包含
 - 用户提供的N个样本集
 - 每个请求示例的特征菜单
 - » 请求特征，如请求类型、请求大小等
 - » 应用程序中的潜在中间变量
 - 利用源代码中的跟踪和日志记录语句
- **Output:** 与请求服务时间最相关的最佳特征
- **选择程序:**
 - 按 *相关度* 的降序排序所有特征
 - » 数值特征：皮尔逊相关系数
 - » 分类特征：相关比的平方
 - 选择第一个特征
 - 一次再选择一个特征，直到相关度不再提高

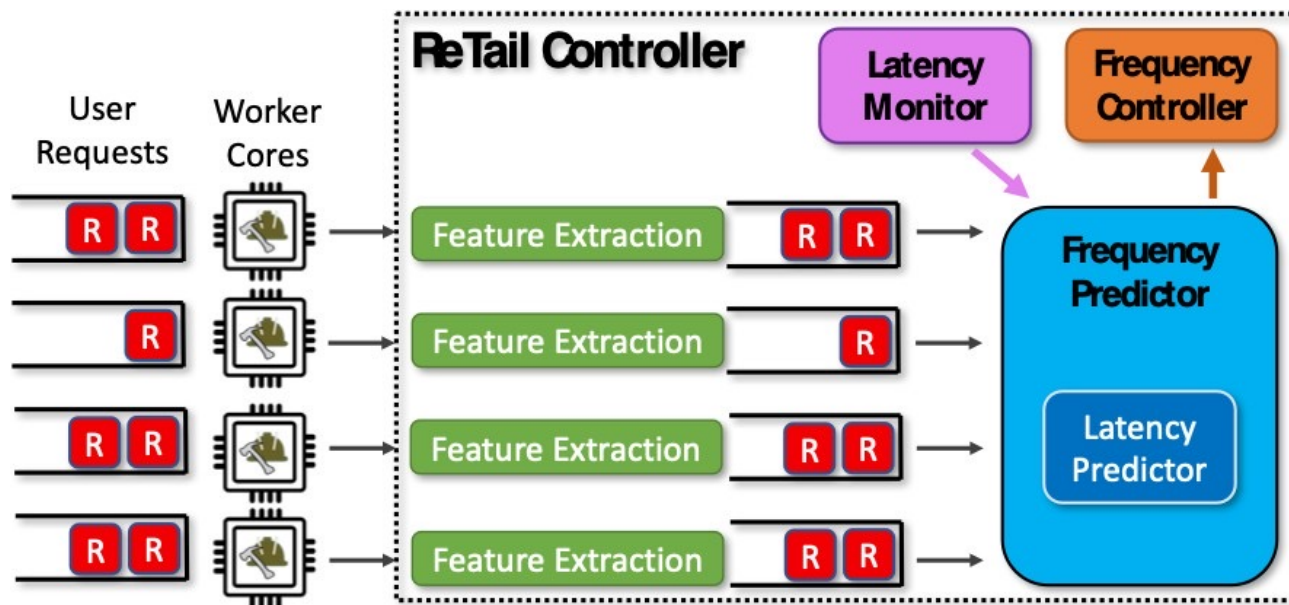
		Model Info				Overhead		Accuracy		
		#Layer	#Neuron/layer	#Epoch	Batch size	Training	Inference	R²	RMSE	RMSE/QoS
Xapian	Linear Regression	N.A.				0.003s	5 μ s	0.959	0.334ms	4.18%
	NN-Gemini	5	128	15	32	9.7s	363 μ s	0.973	0.270ms	3.38%
	NN-Tuned	1	16	5	32	0.98s	107 μ s	0.974	0.264ms	3.30%
Moses	Linear Regression	N.A.				0.003s	5 μ s	0.854	3.622ms	3.02%
	NN-Gemini	5	128	500	32	85.1s	514 μ s	0.833	3.867ms	3.22%
	NN-Tuned	1	4	400	1024	0.74s	258 μ s	0.854	3.617ms	3.01%
Sphinx	Linear Regression	N.A.				0.003s	5 μ s	0.746	217.929ms	5.45%
	NN-Gemini	5	128	1000	32	36.15s	344 μ s	0.747	217.396ms	5.43%
	NN-Tuned	3	128	700	32	15.39s	300 μ s	0.747	217.474ms	5.43%

■ 分类和线性回归

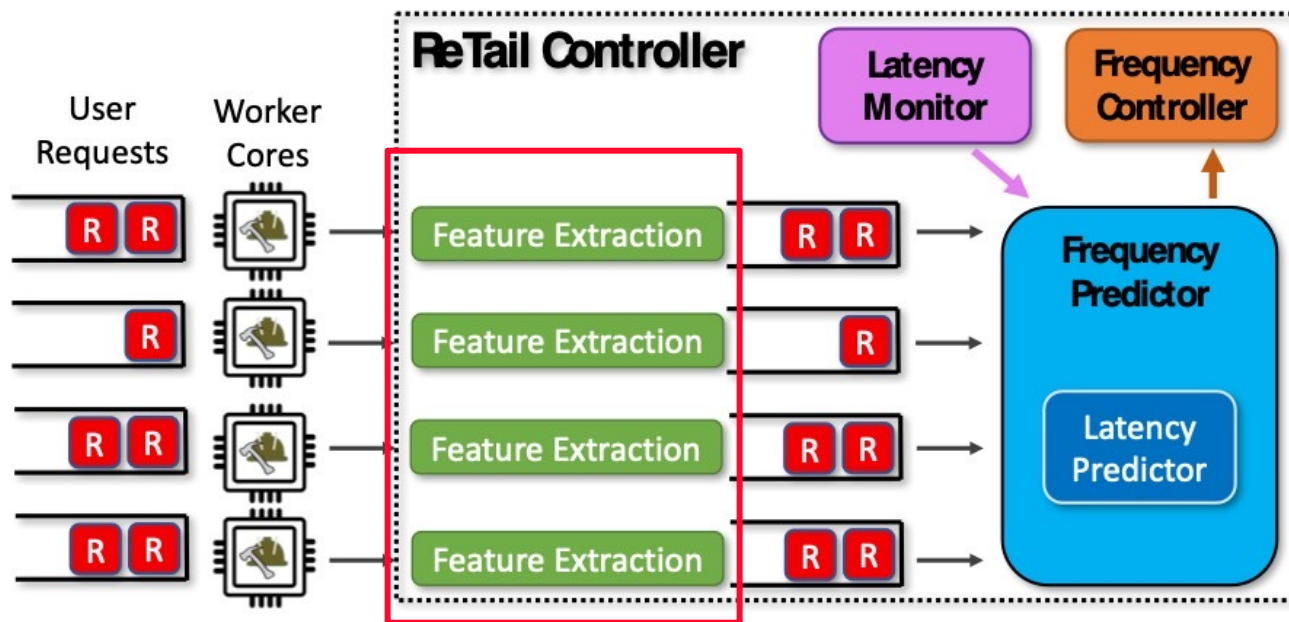
- 大多数关系是分类的或线性的
- 与神经网络的比较
 - » 较小的训练和推理开销
 - » 与神经网络几乎相同的精度
- 可解释的



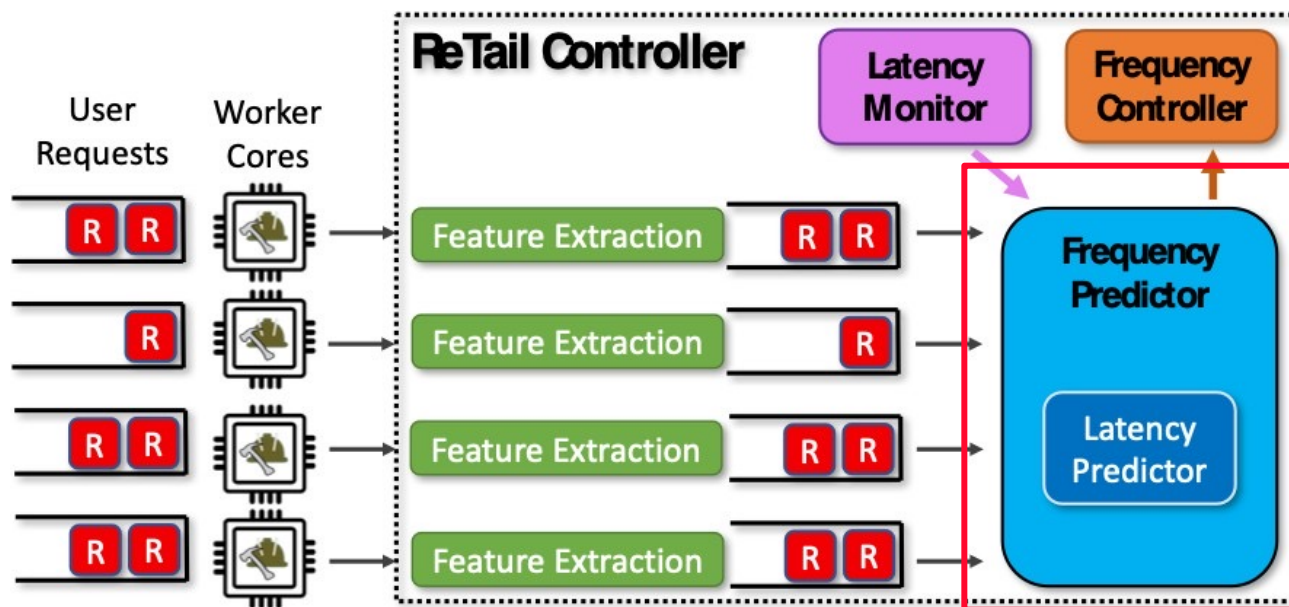
- 找到满足QoS的最小频率



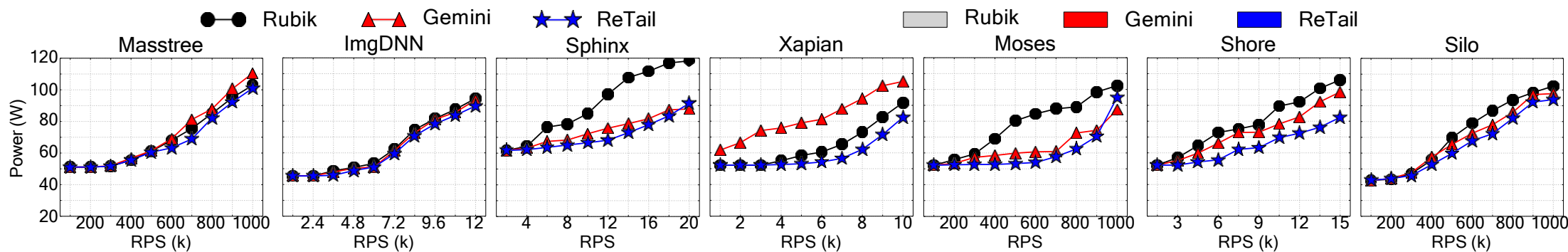
- 找到满足QoS的最小频率



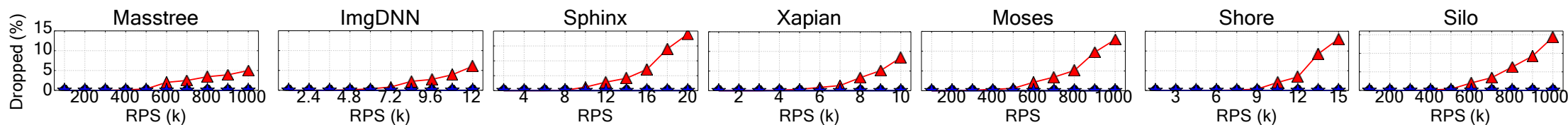
- 找到满足QoS的最小频率



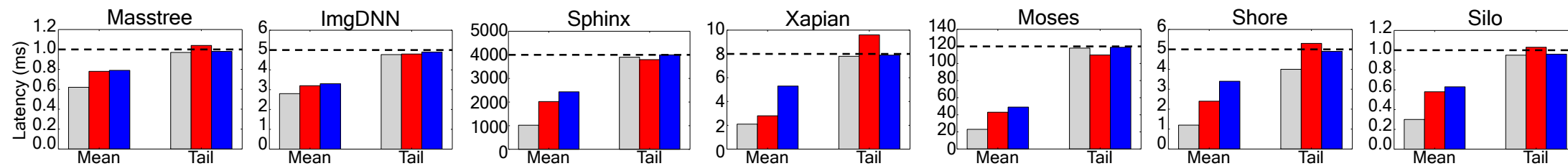
- **Server: Intel Xeon Gold 6152 CPU @ 2.1GHz**
 - Power manager: one reserved core in socket 0
 - LC app: socket 0
 - Clients: socket 1
- **Power measurement: CPU Energy Meter**
 - Measures energy consumption of socket 0
 - Divides the execution time of the LC app
- **ACPI-Freq: 1~2.1GHz in 0.1GHz steps**
- **Baselines:**
 - Rubik [MICRO'15]: statistical model
 - Gemini [MICRO'20]: NN-based, only considers request features



(a) Power consumption under each power manager at various input loads.



(b) Percentage of dropped requests under each power manager at various input load.



(c) Mean and tail latency under each power manager at max load. The horizontal dotted lines are the QoS targets.

- 与Rubik和Gemini相比，分别节能12%和9%
- 不丢弃任何请求

	Masstree	ImgDNN	Sphinx	Xapian	Moses	Shore	Silo
Rubik	0.05	0.9	2500	2.8	47.1	3.9	0.5
Gemini	0.03	0.8	217	3.6	3.6	2.2	0.2
ReTail	0.04	0.8	217	0.3	3.6	0.3	0.1

- **ReTail 最低的 Root-Mean-Square-Error (RMSE)**
- **ReTail 优于 Gemini 更复杂的 NN 模型，因为**
 - NN 的高推断开销延迟频率调整
 - Gemini 只考虑请求特性，而 ReTail 也考虑应用特性

- 利用请求级延迟预测提高电源效率
- ReTail 特征选择
- ReTail 延迟预测: 一个简单的学习模型就够了!!
- ReTail 功率管理
- 与没有QoS违规的最先进的功率管理器相比, 节电高达36% (平均9%)

RETAIL: OPTING FOR LEARNING SIMPLICITY TO ENABLE QoS-AWARE POWER MANAGEMENT IN THE CLOUD

Thanks!