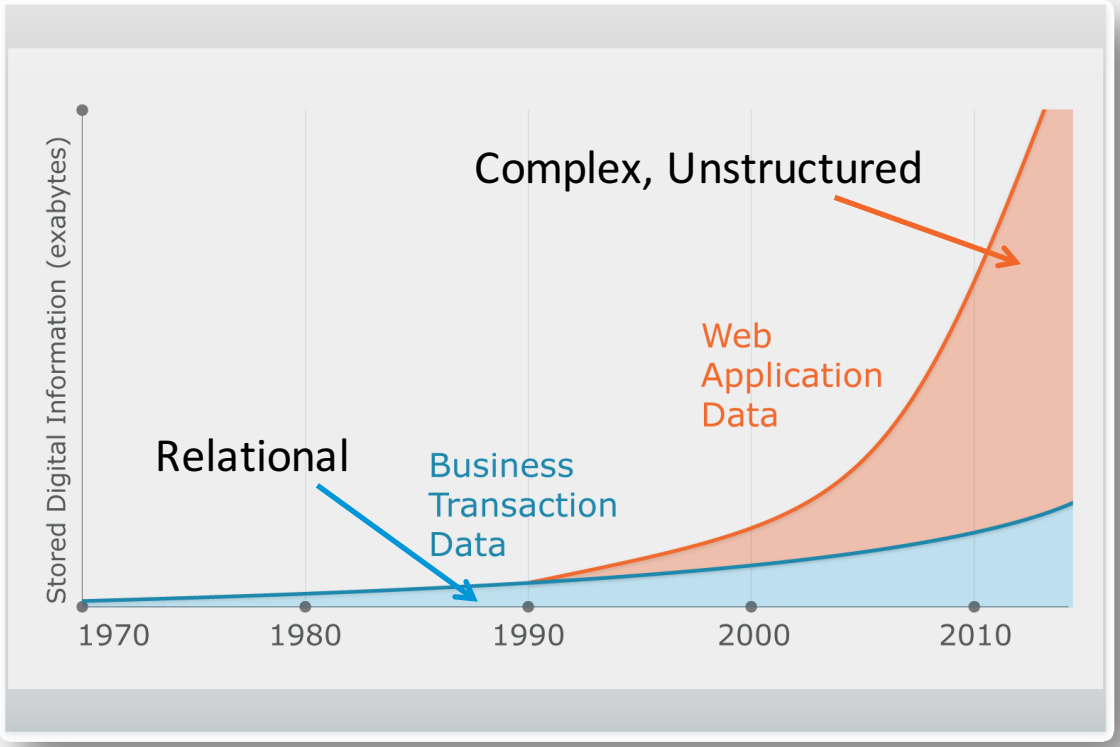
TimeTrader: Exploiting Latency Tail to Save Datacenter Energy for Online Search

Balajee Vamanan, Hamza Bin Sohail, Jahangir Hasan, and T. N. Vijaykumar

Searching large data is becoming prevalent

- Data is growing exponentially



- **Online Search (OLS)**: **Interactively** query and access data
  - Key component of many web applications, **not only** *Web Search* (*e*.g., Facebook, Twitter, advertisements)

Energy management for OLS is hard

Traditional energy management does not work

1. Interactive, strict service-level agreements (SLAs)
   - **Cannot batch**
2. Short response times and inter-arrival times (e.g., 200 ms and ≈ 1 ms for Web Search)
   - **Low-power modes (e.g., p-states) not applicable**
3. Shard data over 1000s of servers → each query searches *all* servers
   - **Cannot consolidate workload to fewer servers**

Carefully slow down (not turn off) servers without violating SLAs

Previous work

- ***Pegasus*** *[ISCA'14]:* Achieves load-proportional energy
  - Uses (global) **datacenter-wide** response times
    - **Shifts** latency distribution at *low* loads

✓ Saves energy at low loads

✘ Does not save much at high loads

  - Saves 0% at peak load
  - Datacenters operate at moderate-peak during the day (diurnal pattern) → savings desired at high loads

Pegasus saves energy at low loads but limited savings at high loads

TimeTrader: contributions

Each query leads to 1000s of sub-queries

Each query's budget set to accommodate 99th %-ile sub-query

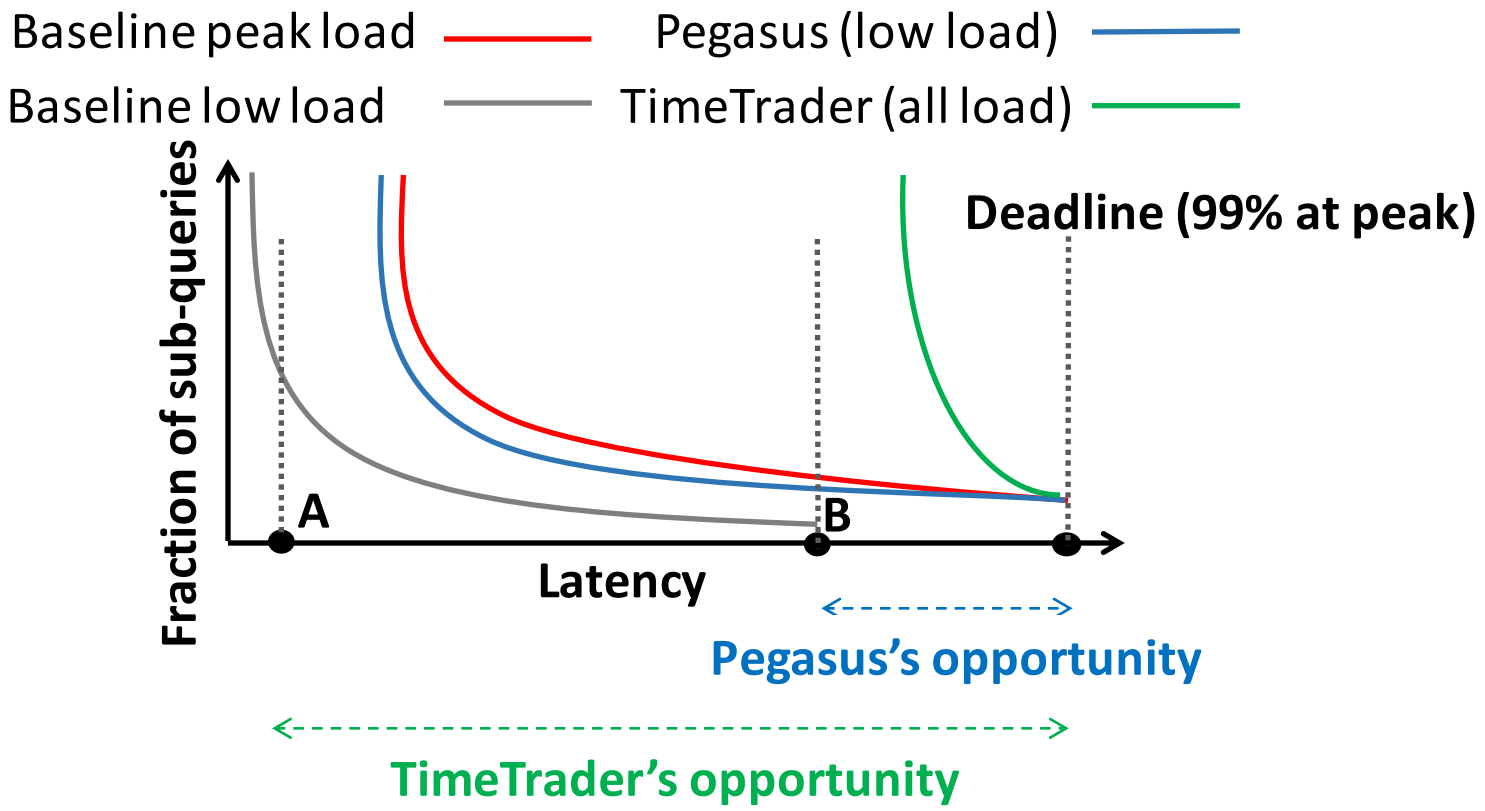
- In both network and compute parts

- Key observation: 80% sub-queries complete in 20% of budget

1. TimeTrader exploits *both* network and compute slack to save energy

TimeTrader: contributions (cont.)

2. Uses (local) **per-sub-query** latencies → **reshapes** sub-query response time distribution at *all* loads



A: median baseline at low load

B: 99% baseline at low load

TimeTrader: contributions (cont.)

3. Leverages network signals to determine slack
   - Does **not need fine-grained clock** synchronization
4. Employs Earliest Deadline First (EDF) scheduling
   - Slowing down a sub-query affects other queued sub-queries
     - Critical sub-queries affected the most
   - Allows critical sub-queries to bypass others

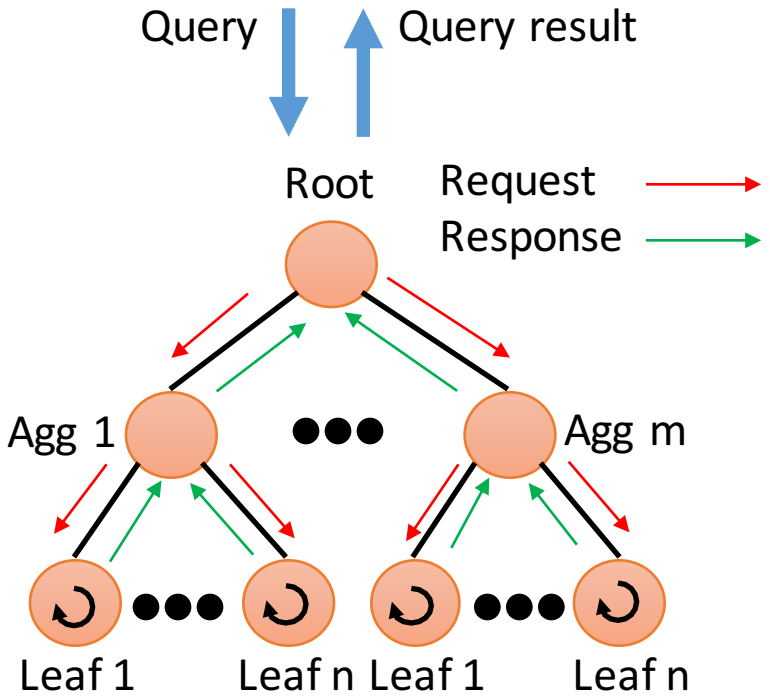
TimeTrader is the first to explore cross-layer optimization between network and architecture layers; saves 15% - 40% energy at peak - 30% loads

Outline

- Introduction
- **Background**
  - **OLS architecture**
  - **Tail latencies and SLA budgets**
- TimeTrader's design
- Methodology
- Results

Background: OLS architecture

- Partition-aggregate (tree)
- Request-compute-reply
  - Request: root → leaf
  - Compute: leaf node
  - Reply: leaf → root

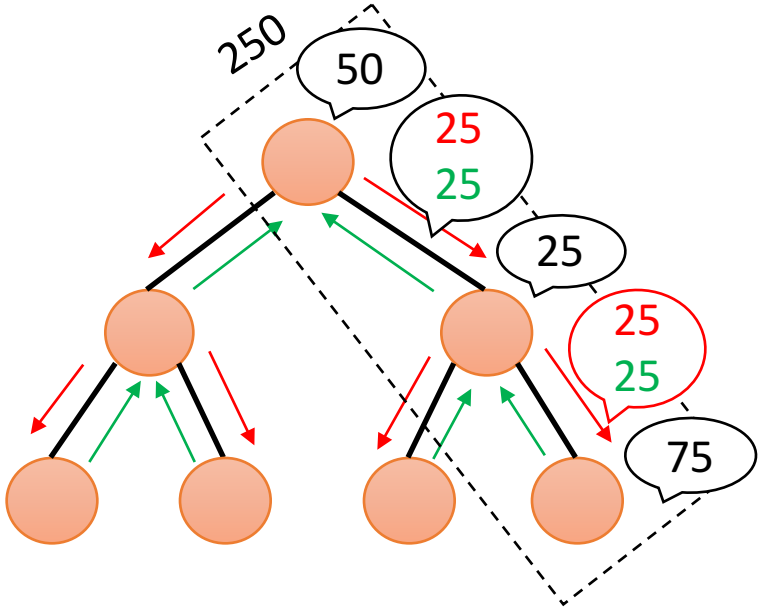


- Root waits until deadline to generate overall response
  - Some replies may take longer in network or compute
    - Collisions in network → TCP retransmit
    - Imperfect sharding → disproportionately large compute
  - Long tails → tail latency problem

OLS tail latencies and SLA budgets

- Missed replies due to tail latency affect quality
  - e.g., SLA of 1% missed deadlines
  - Time budgets based on 99th %-ile latency of sub-queries

- To optimize network and compute separately
  - Split budget into network and compute
  - e.g., total = 250 ms
    → flow deadlines 25 ms



80% of sub-queries complete within 20% of budget → slack

Outline

- Introduction
- Background
- **TimeTrader's design**
  - **Key ideas**
  - **Determining slack**
  - **Slowing down based on slack**
  - **EDF**
- Methodology
- Results

TimeTrader: design

- TimeTrader exploits per-sub-query slack
  - Sources of slack
    - Network
      - ✓ Request
      - ✗ Reply (after compute, unpredictable)
    - Compute
      - ✗ Actual compute (query dependent)
      - ✓ Queuing (due to local load variations)

1. Determine (a) request slack and (b) compute slack
2. Slow down based on slack and load
3. Shield critical sub-queries from slowed down sub-queries
   - EDF implementation

- Intel's Running Avg. Power Limit (RAPL) to set power states

1(a): Determine request slack

- Requests traverse network from root to leaf
- Timestamp at parent and leaf?
  - ✘ Clock skew ≈ slack
  - ✓ Estimate based on already-available network signals
    - e.g., Explicit Congestion Notification (ECN) marks, TCP timeouts

If a leaf doesn't see either ECN or Timeouts

Request Slack = Network Budget – Median Latency

Else

Request Slack = 0

1(b): Determine compute-queuing slack

- Exploit local load variations in each leaf server
- Slack calculation
  - Details in the paper

$$Total\ Slack = Request\ Slack + Compute\text{-}Queuing\ Slack$$

2: Slow down based on slack

- Given the slack, calculate slow down factor
  - As a fraction of compute budget

- But not all of the slack can be used
  - Slack calculation assumes no queuing
  - Need to scale slack based on load

$$Slowdown\ factor = Total\ Slack * Scale / Compute\ Budget$$

- *Scale* depends on load
  - Feedback controller (details in the paper)

3: Shield critical sub-queries

- Earliest Deadline First (EDF)
  - Implementation details in the paper

Outline

- Introduction
- Background
- TimeTrader's design
- **Methodology**
- **Results**

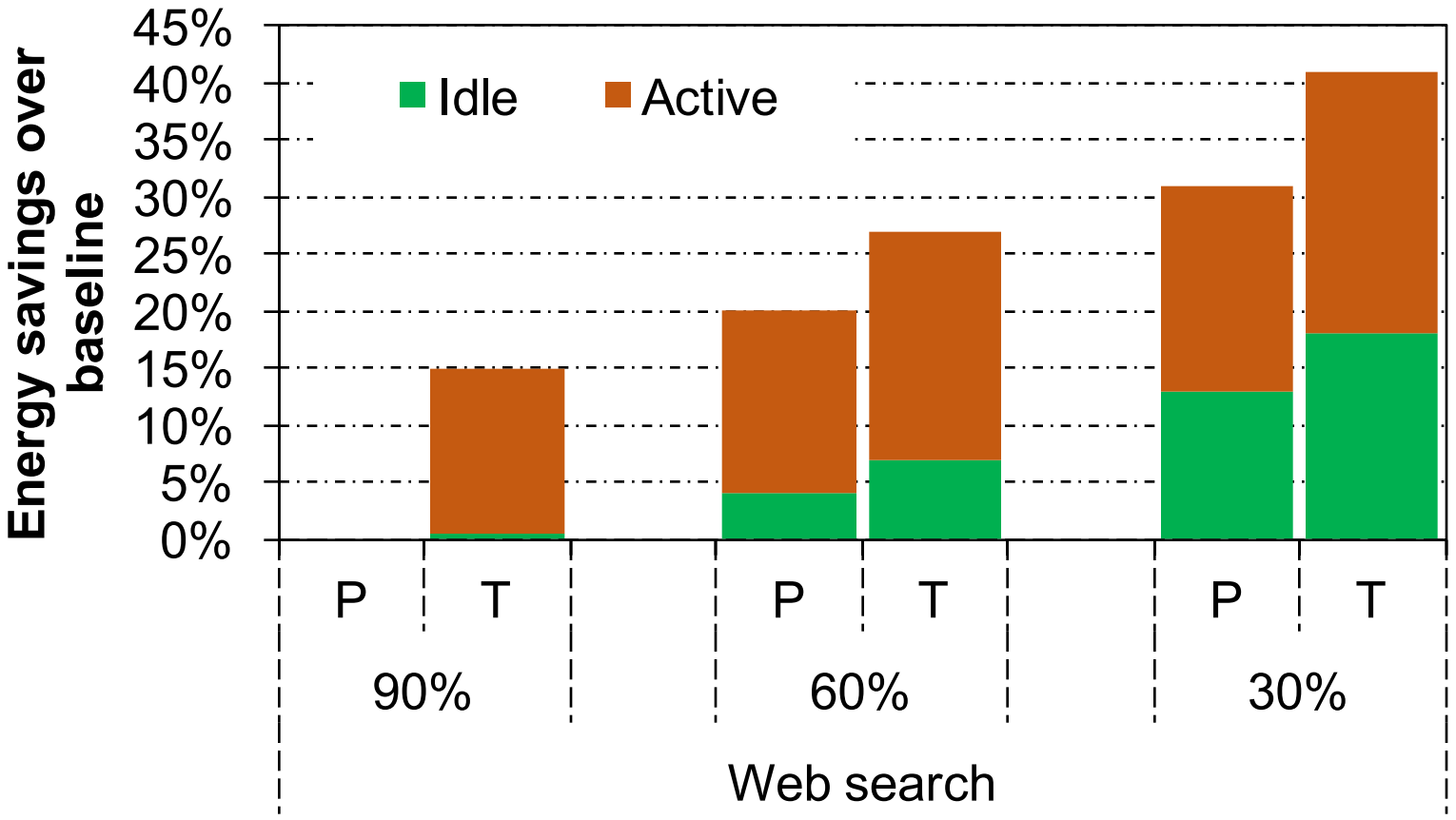
Methodology

- Compute
  - Real measurements for service time, power
- Network
  - Real measurements at a small scale
  - Tails effects only in large clusters → simulate with ns-3

Workload: Web Search (Search) from CloudSuite 2.0

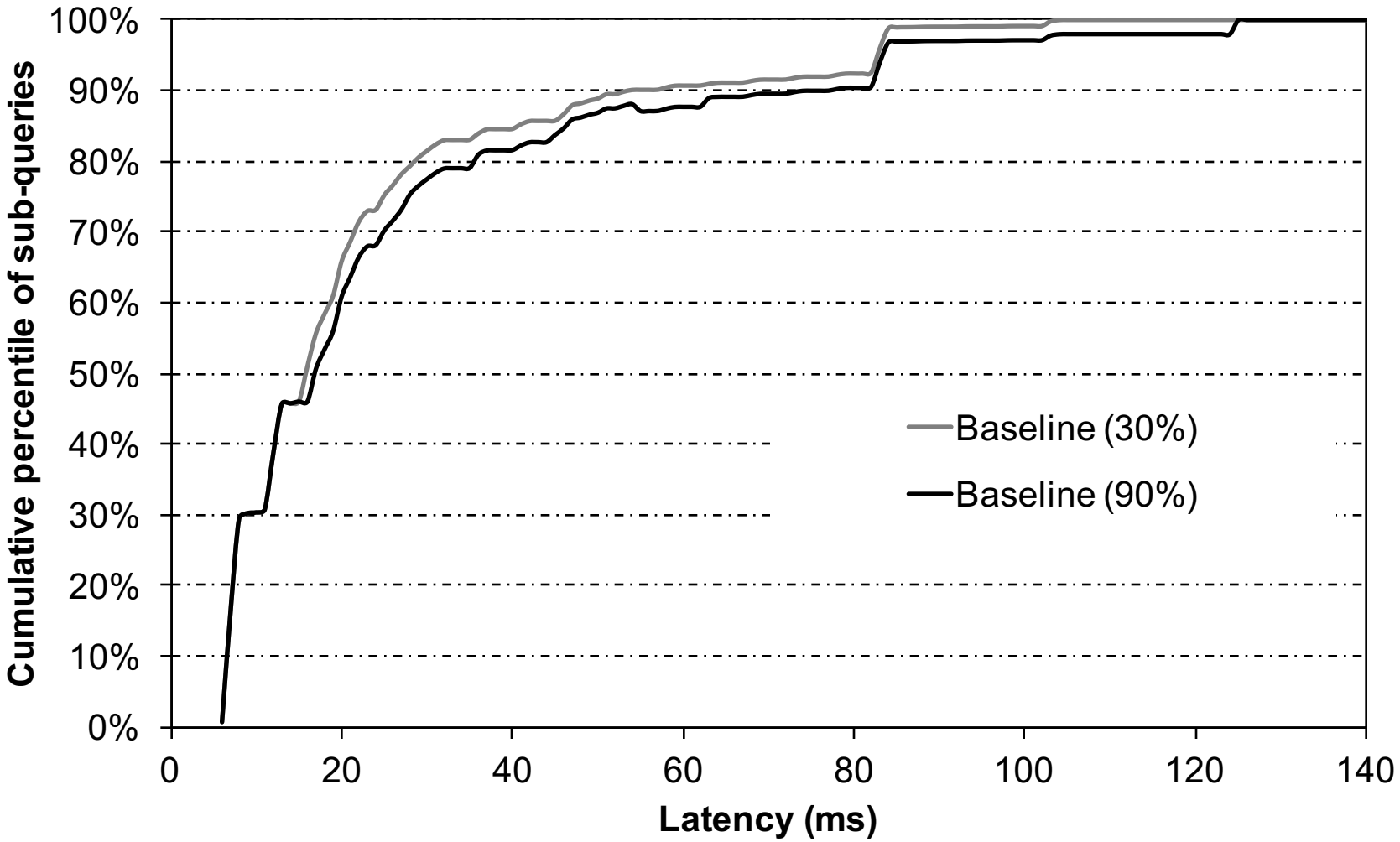
- Search index from Wikipedia
- 3000 queries/s at peak load → 90% load
- Deadline budget: Overall 125 ms
  - Network (request/reply): 25 ms
  - Compute: 75 ms
- SLA of 1% missed deadlines

Idle and Active Energy Savings

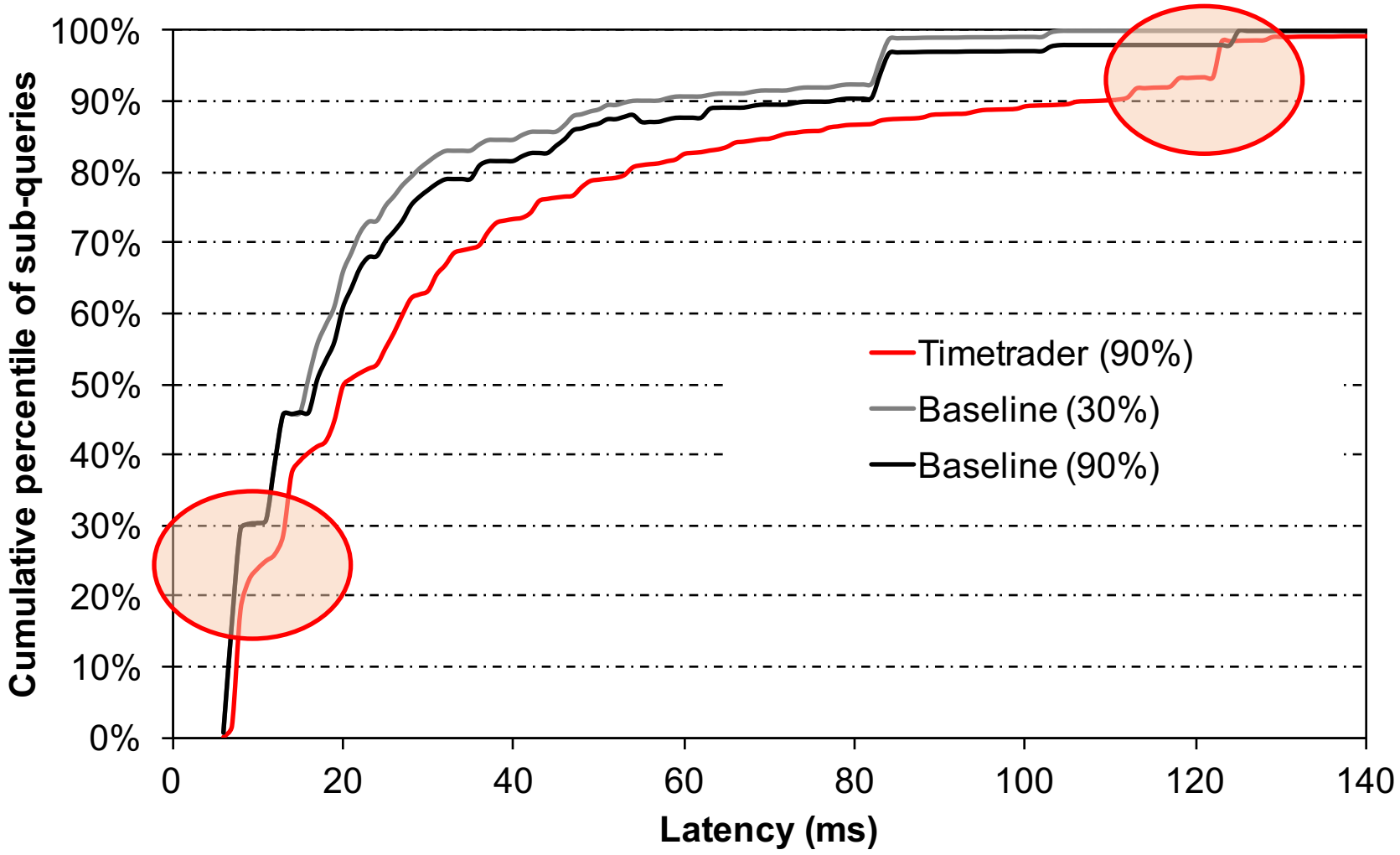


TimeTrader saves 15% - 40% energy over baseline (17% over Pegasus at 30% load)

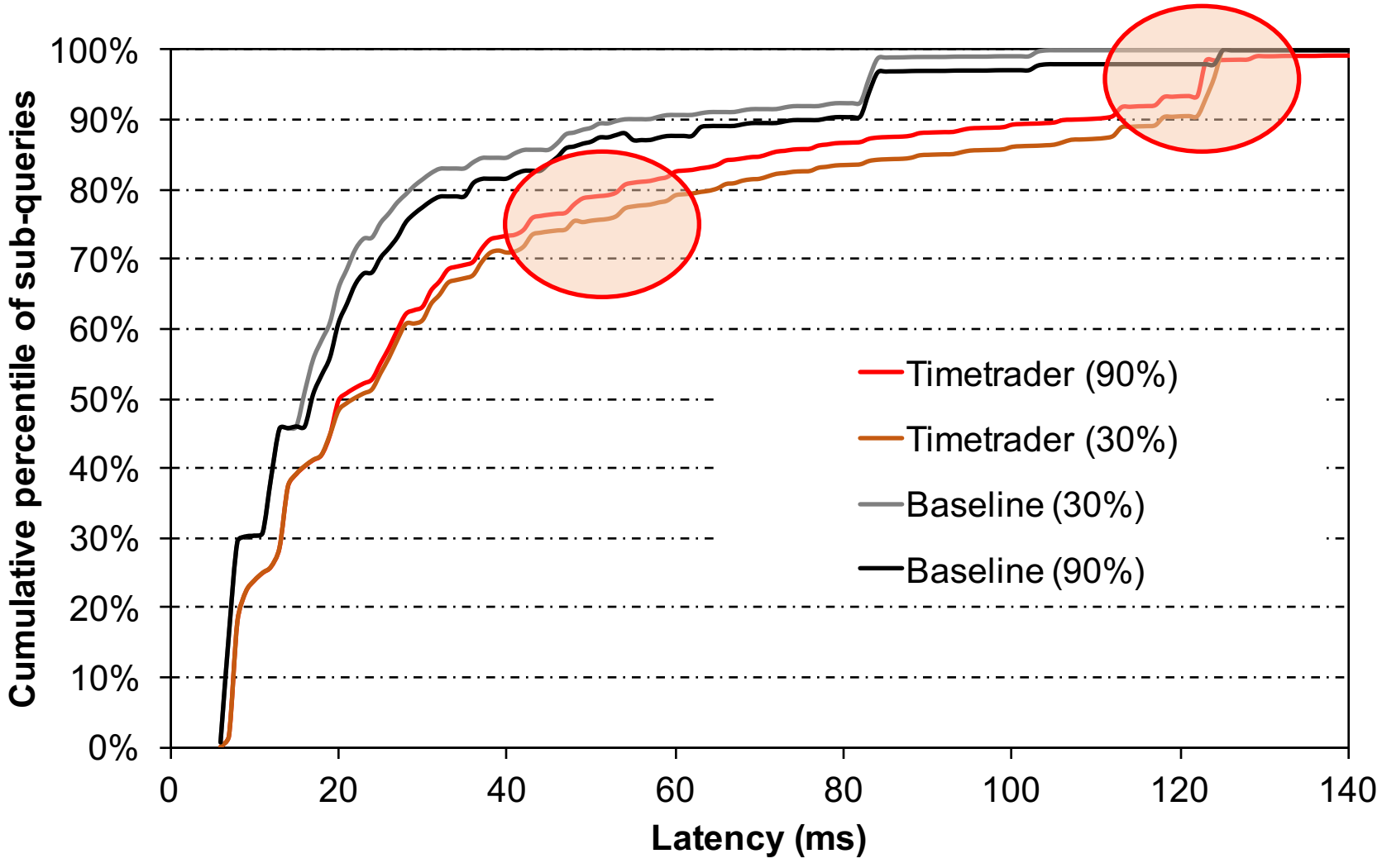
Response time distribution



Response time distribution



Response time distribution



TimeTrader reshapes distribution at all loads;
Slows ~80% of requests at all loads

Conclusion

TimeTrader…

- Exploits sub-query slack
- Reshapes response time distribution at *all* loads
  - Saves 15% energy at peak, 40% energy at 30% load
- Leverages network signals to estimate slack
- Employs EDF to decouple critical sub-queries from sub-critical sub-queries

TimeTrader converts the performance disadvantage of latency tail into an energy advantage!