Adrenaline: Pinpointing and Reining in Tail Queries with Quick Voltage Boosting

Chang-Hong Hsu, Yunqi Zhang, Michael A. Laurenzano, David Meisner, Thomas Wenisch, Jason Mars, Lingjia Tang, Ronald G. Dreslinski
When datacenters meet users
When datacenters meet users
When datacenters meet users

User-facing services are everywhere!
User-facing services need to be FAST!
User-facing services need to be FAST!

- When users send queries,
User-facing services need to be FAST!

- When users send queries, they expect fast responses
User-facing services need to be FAST!

• When users send queries, they expect fast responses

Latency is the key to the high quality of user-facing services
How fast is “FAST”?
How fast is “FAST”?

For user-facing services, “fast” means low tail latency.
“Tail latency?”

• Tail latency is the latency of the **slowest queries** of the entire distribution

• Tail latency represents the “**worst-case**” **quality** of a service (worst-case QoS)

• Tail latency directly relates to **user experience**
“Fast” means low tail latency

• Imagine you are sending out search queries…
“Fast” means low tail latency

• Imagine you are sending out search queries…
  • “0.2-second response” vs. “0.1-second response”
“Fast” means low tail latency

• Imagine you are sending out search queries…
  • “0.2-second response” vs. “0.1-second response”

• How does a “3-second response” sound?
“Fast” means low tail latency

• Imagine you are sending out search queries…
  • “0.2-second response” vs. “0.1-second response”

Improving the already “fast enough” MEAN latency does not necessarily give better user experience

• How does a “3-second response” sound?
“Fast” means low tail latency

• Imagine you are sending out search queries…
  • “0.2-second response” vs. “0.1-second response”

Improving the already “fast enough” MEAN latency does not necessarily give better user experience

• How does a “3-second response” sound?

Reduce the long TAIL latency makes a lot of sense!
“Fast” means low tail latency

• Imagine you are sending out search queries…
  • “0.2-second response” vs. “0.1-second response”

Improving the already “fast enough” MEAN latency does not necessarily give better user experience

• How does a “3-second response” sound?

Reduce the long TAIL latency makes a lot of sense!
“Fast” means low tail latency

• Imagine you are sending out search queries…
  • “0.2-second response” vs. “0.1-second response”

Improving the already “fast enough” MEAN latency does not necessarily give better user experience

• How does a “3-second response” sound?

Reduce the long TAIL latency makes a lot of sense!
OK, I’ll **boost the voltage/frequency of my cores** when the load is high, making the service run fast.
OK, I’ll **boost the voltage/frequency of my cores** when the load is high, making the service run fast.

Wait a sec…I do want my service fast, but I remember $P_{\text{dyn}} \propto f^3$.

That means **A LOT of extra energy**! No!!
OK, I’ll **boost the voltage/frequency of my cores** when the load is high, making the service run fast.

Wait a sec…I do want my service fast, but I remember $P_{dyn} \propto f^3$. That means **A LOT of extra energy**! No!!

Can I have a **fast service** with only **little energy overhead**?
Yes! We can achieve that goal if we can
1. pinpoint those queries in the tail
2. boost these tail queries specifically
Yes! We can achieve that goal if we can
1. pinpoint those queries in the tail
2. boost these tail queries specifically

Adrenaline introduces query-level boosting
✓ Identify and take advantages of the differences among queries
✓ Switching core’s voltage/frequency quickly to boost for the slow-running queries
Queries are highly variable

• Even for the same web service, queries
  - come from different users
  - have different contents
  - require different actions
Queries are highly variable

• Even for the same web service, queries
  - come from different users
  - have different contents
  - require different actions

We observe that…
Queries are highly variable

- Even for the same web service, queries
  - come from different users
  - have different contents
  - require different actions

We observe that...

→ they need different amount of time to process
Queries are highly variable

- Even for the same web service, queries come from different users
- have different contents
- require different actions

We observe that...
- they need different amount of time to process
- they react differently to core’s V/f scaling
Variability matters!
Variability matters!

Variability plays a key role in helping identify the critical and beneficial queries to boost
Variability matters!

Variability plays a key role in helping identify the critical and beneficial queries to boost

**Insight:**
For a user-facing service, if we know
1. “what query to target”
2. “how much I can boost”
we can boost cores’ V/f accordingly and intelligently
“What query to target?”

Characteristic 1: high contribution to the tail

Query distribution

Query Latency

- QueryType1
- QueryType2
- QueryType3
“What query to target?”

Characteristic 1: **high contribution to the tail**

![Query distribution graph]

- QueryType1
- QueryType2
- QueryType3

Candidate
High tail-contribution example in Memcached

What is in this graph
- Latency distributions (CDFs) of different types of requests
- Running with a fixed core frequency
High tail-contribution example in Memcached

What is in this graph
- Latency distributions (CDFs) of different types of requests
- Running with a fixed core frequency

Observation
- SETs have 2-3x longer tail latency
How much can we boost?

Characteristic 2: **High boost-ability**

![Graph showing query latency vs. core frequency for QueryType1 and QueryType2.](image-url)
How much can we boost?

Characteristic 2: **High boost-ability**

![Graph showing Core Frequency vs. Query Latency with lines for QueryType1 and QueryType2. The graph illustrates the boost-ability with a candidate point highlighted.](image)

How much can we boost?
High boost-ability example in Memcached

What is in this graph
- Latency distributions of different types of requests
- Running with different core frequencies

![Graph showing latency distributions for GET, SET, and DEL requests with 1.2GHz and 2.4GHz core frequencies.](image)
High boost-ability example in Memcached

What is in this graph
- Latency distributions of different types of requests
- Running with different core frequencies

Observation
- Boosting the core from the lowest to the highest frequency improves SET’s tail latency from 13µs to 7µs
High boost-ability example in Memcached

**What is in this graph**
- Latency distributions of different types of requests
- Running with different core frequencies

**Observation**
- Boosting the core from the lowest to the highest frequency improves SET’s tail latency from **13µs to 7µs**

**SET requests are good candidates for boosting**
Query-level boosting with Adrenaline

✓ Pinpoint queries that are highly likely to contribute to the tail

✓ Quickly boost the core via ultra-fast switching circuitry
Design of Adrenaline

Adrenaline runtime system

Queries → Query info → Query identifier → Decision engine

Short Stop Circuitry
[Pinckney et al. VLSIC’13]

VR → Short Stop
Design of Adrenaline

Adrenaline runtime system

Query identifier
Decision engine

Short Stop Circuitry
[Pinckney et al. VLSIC’13]

Queries
Query info

Fine-grain switching
Design of Adrenaline

Adrenaline runtime system

Queries → Query info

Query identifier

Decision engine

Short Stop Circuitry

[Pinckney et al. VLSIC’13]

Switching latency: ~10’s ns

Fine-grain switching

Adrenaline: Pinpointing and Reining in Tail Queries with Quick Voltage Boosting
Design of Adrenaline

Adrenaline runtime system

Queries

Load monitor

Load info

Query info

Query identifier

Decision engine

Short Stop Circuitry

[Pinckney et al. VLSIC’13]

VR

VR

Short Stop

Switching latency: ~10’s ns

Fine-grain switching

Adrenaline: Pinpointing and Reining in Tail Queries with Quick Voltage Boosting
Boosting based on the query-level indicators

Rapidly identifying query types
- Candidate vs. non-candidate type
- Needs to be simple to achieve low overhead
Boosting based on the query-level indicators

**Query identifier**

**Rapidly identifying query types**
- **Candidate** vs. **non-candidate** type
- Needs to be simple to achieve low overhead

**Decision engine**

**Candidate** type:
- Boost this query as soon as possible

**Non-candidate** type:
- Boost when query runtime exceeds half of QoS target
Adrenaline: a closer look
Adrenaline: a closer look
Adrenaline: a closer look

Requests

GET GET GET SET DEL GET GET SET

PDF

Mean

P99

Latency

Adrenaline: Pinpointing and Reining in Tail Queries with Quick Voltage Boosting
Coarse-grain solution shifts entire distribution

![Diagram showing PDF distribution with mean and P99 markers.]
Coarse-grain solution shifts entire distribution

Coarse-grain solutions boost every query in that time interval
Coarse-grain solution shifts entire distribution

Coarse-grain solutions boost every query in that time interval
Adrenaline targets the tail
Adrenaline targets the tail

Adrenaline focuses on boosting candidate requests

PDF

Mean

P99

Latency

Adrenaline: Pinpointing and Reining in Tail Queries with Quick Voltage Boosting
Adrenaline targets the tail

Adrenaline focuses on boosting candidate requests
Measured latency distributions

• Compared to coarse-grain DVFS, Adrenaline…
Measured latency distributions

- Compared to coarse-grain DVFS, Adrenaline...
  - reduces the tail latency significantly, achieving superior QoS
Compared to coarse-grain DVFS, Adrenaline:

- reduces the tail latency significantly, achieving superior QoS
- trades the mean latency for energy saving
Evaluation methodology

• Measure request latency distributions on a real system
  • Intel Ivy Bridge + 136 GB RAM
  • Analyze Adrenaline & coarse-grain DVFS in BigHouse
    [Meisner et al. ISPASS’12]
Evaluation methodology

• Measure request latency distributions on a real system
  • Intel Ivy Bridge + 136 GB RAM
  • Analyze Adrenaline & coarse-grain DVFS in BigHouse
    [Meisner et al. ISPASS’12]

• Two major benchmarks from CloudSuite [Ferdman et al. ASPLOS’12]
  • Memcached and Web Search
Evaluation methodology

• Measure request latency distributions on a real system
  • Intel Ivy Bridge + 136 GB RAM
  • Analyze Adrenaline & coarse-grain DVFS in BigHouse
    [Meisner et al. ISPASS’12]

• Two major benchmarks from CloudSuite [Ferdman et al. ASPLOS’12]
  • Memcached and Web Search

• We generate various workload compositions using Facebook’s published result [Atikoglu et al. SIGMETRIC’12]
Rein in the tail with Adrenaline

Optimizing for tail latency:
• Same energy budget
• Memcached:
  • Max: up to 2.5x
  • Avg: up to 1.6x
• Websearch
  • Max: up to 3x
  • Avg: up to 1.7x
Rein in the tail with Adrenaline

Optimizing for tail latency:
- Same energy budget
- Memcached:
  - Max: up to 2.5x
  - Avg: up to 1.6x
- Websearch
  - Max: up to 3x
  - Avg: up to 1.7x
When we want low energy

Optimizing for energy
- Same tail latency target
- Memcached:
  - Max: up to 1.8x
  - Avg: up to 1.2x
- Websearch
  - Max: up to 2x
  - Avg: up to 1.5x
Conclusion
Conclusion

• User-facing services requires responsiveness and energy efficiency
Conclusion

• User-facing services requires responsiveness and energy efficiency

• Pinpointing and boosting for tail queries gives the best of both worlds
Conclusion

• User-facing services requires responsiveness and energy efficiency

• Pinpointing and boosting for tail queries gives the best of both worlds

• Adrenaline outperforms coarse-grain DVFS
  ✓ Up to 3x tail latency improvement
  ✓ Up to 2x energy saving improvement