Australian
National
University

FlexHeap: Dynamic I/O-Aware Heap Resizing for Managed Applications

Iacovos G. Kolokasis
kolokasis@ics.forth.gr

Shoaib Akram
shoaib.akram@anu.edu.au

Foivos Zakkak
fzakkak@redhat.com

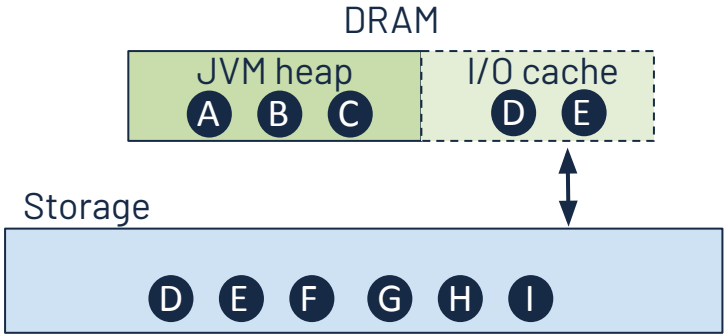
Polyvios Pratikakis
polyvios@ics.forth.gr

Angelos Bilas
bilas@ics.forth.gr

JVM heap and OS I/O cache compete for DRAM

- Modern JVM systems process large datasets from storage
  - E.g., Elasticsearch, Cassandra, Spark

- They require both large heaps and large OS I/O cache



JVM heap and OS I/O cache compete for DRAM

- Modern JVM systems process large datasets from storage
  - E.g., Elasticsearch, Cassandra, Spark

- They require both large heaps and large OS I/O cache
  - Larger JVM heap reduces **GC frequency**

Elasticsearch - Wikipedia workload

	Larger JVM heap	**Larger I/O cache**
# GC	3,616	6,199

JVM heap and OS I/O cache compete for DRAM

- Modern JVM systems process large datasets from storage
  - E.g., Elasticsearch, Cassandra, Spark

- They require both large heaps and large OS I/O cache
  - Larger JVM heap reduces **GC frequency**
  - Larger I/O cache reduces **I/O stalls**

Elasticsearch - Wikipedia workload

	Larger JVM heap	**Larger I/O cache**
# GC	3,616	6,199
I/O traffic (GB)	4,046	1,677

JVM heap and OS I/O cache compete for DRAM

- Modern JVM systems process large datasets from storage
  - E.g., Elasticsearch, Cassandra, Spark
- They require both large heaps and large OS I/O cache
  - Larger JVM heap reduces **GC frequency**
  - Larger I/O cache reduces **I/O stalls**
- **Partitioning available DRAM** affects **performance** significantly

Elasticsearch - Wikipedia workload

	Larger JVM heap	Larger I/O cache
# GC	3,616	6,199
I/O traffic (GB)	4,046	1,677
Ingest (DPS)	5,034	4,680
Search (QPS)	8.02	16.05

JVM heap and OS I/O cache compete for DRAM

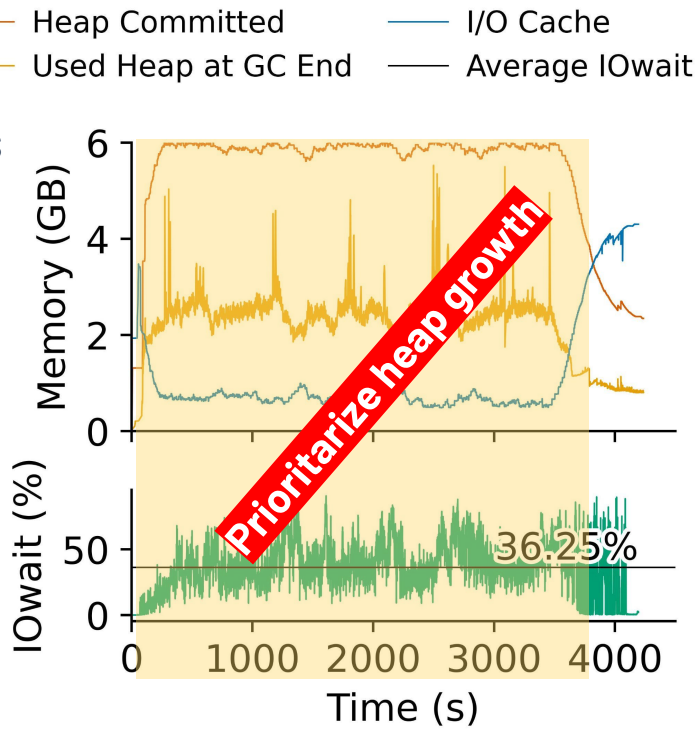
- Modern JVM systems process large datasets from storage
  - E.g., Elasticsearch, Cassandra, Spark
- They require both large heaps and large OS I/O cache
  - Larger JVM heap reduces **GC frequency**
  - Larger I/O cache reduces **I/O stalls**
- **Partitioning available DRAM** affects **performance** significantly
- **Challenge**
  - How much DRAM to provide for each use?
  - How to adjust dynamically during different application phases?

Elasticsearch - Wikipedia workload

	Larger JVM heap	Larger I/O cache
# GC	3,616	6,199
I/O traffic (GB)	4,046	1,677
Ingest (DPS)	5,034	4,680
Search (QPS)	8.02	16.05

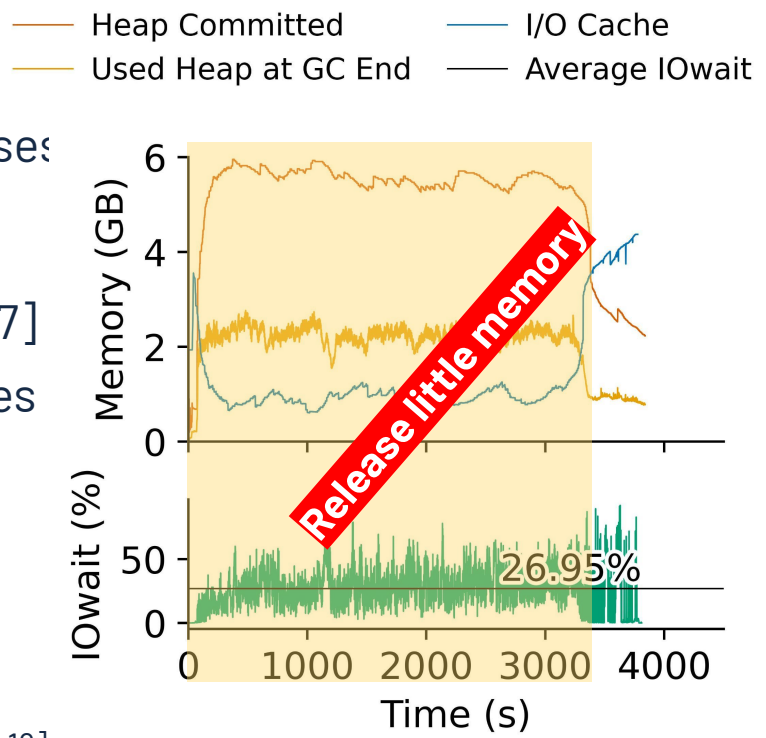
Existing heap resizing mechanisms is oblivious to I/O

- Either **reduce GC cost** to improve application performance [G1 Ergonomics, ZCPU-MPLR 23]
  - Priority is heap growth in mixed GC and I/O phases



Existing heap resizing mechanisms is oblivious to I/O

- Either **reduce GC cost** to improve application performance [G1 Ergonomics, ZCPU-MPLR 23]
  - Priority is heap growth in mixed GC and I/O phases
- Or **reduce server memory usage** to allow more applications to run [VG1-ISMM 18, ElasticMem-ATC17]
  - Release little memory in mixed GC and I/O phases
- No existing mechanism considers I/O cost



1. Heap size adjustment with CPU control [MPLR 23]
2. Dynamic vertical memory scalability for OpenJDK cloud applications [ISMM 18]
3. Elastic memory management for cloud data analytics [ATC 17]

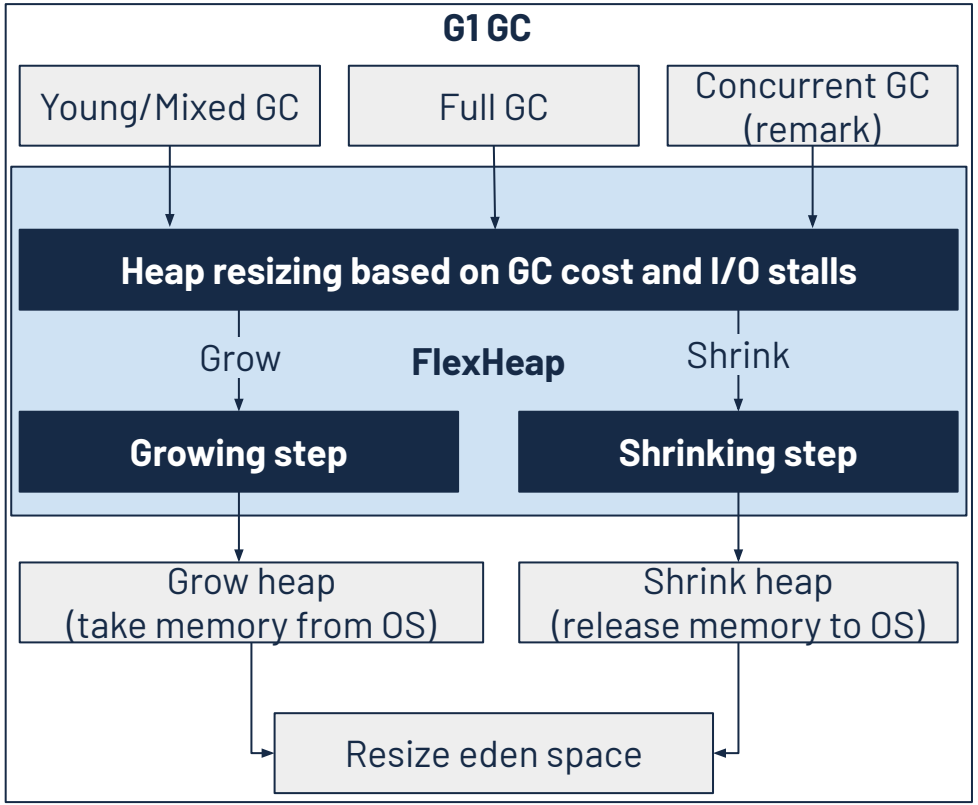
Today: Limit the heap size for more I/O cache

- When I/O cost is high, main approach is to statically allocate I/O cache
  - Cap the heap to a maximum size → Rest of memory used by I/O cache
  - Elasticsearch and Cassandra recommend a **50/50 static heap–I/O cache split**

- However, static DRAM partitioning
  - Is not straightforward to find a proper split
  - Cannot adapt to dynamic application behavior

FlexHeap

- Consider both GC and I/O cost
- **Dynamically** resize the heap
  - Rest of memory used by I/O cache
- Designed for **G1**
  - Default OpenJDK collector
  - State-of-the-art garbage collector
- FlexHeap adds **two main decisions** to G1
  - Grow or shrink the heap?
  - By what amount?



Outline

- Motivation

- FlexHeap design
  - Deciding to grow or shrink the heap
  - Estimating the impact of GC and I/O stalls
  - Deciding the resizing step

- Evaluation

- Conclusions

Deciding to grow or shrink the heap

- What is the impact of GC and I/O on application performance?
- FlexHeap tracks **lost potential for compute** by mutators
  - Due to GC or I/O, in **CPU cycles**
  - **Lost potential (cycles) = GC cost + I/O stalls**

- Divide execution in intervals, based on GC phases (young GC etc.)
  - Track **history** of GC and I/O

Deciding to grow or shrink the heap

- What is the impact of GC and I/O on application performance?
- FlexHeap tracks **lost potential for compute** by mutators
  - Due to GC or I/O, in **CPU cycles**
  - **Lost potential (cycles) = GC cost + I/O stalls**

State
Wait Grow

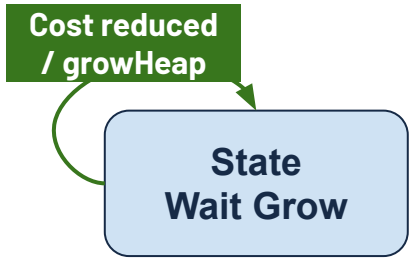
- Divide execution in intervals, based on GC phases (young GC etc.)
  - Track **history** of GC and I/O

- At the end of each interval, **decide how to resize the heap**
  - **Wait for resize** operation to take effect

State
Wait Shrink

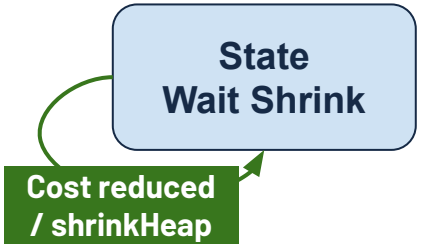
Deciding to grow or shrink the heap

- What is the impact of GC and I/O on application performance?
- FlexHeap tracks **lost potential for compute** by mutators
  - Due to GC or I/O, in **CPU cycles**
  - **Lost potential (cycles) = GC cost + I/O stalls**



- Divide execution in intervals, based on GC phases (young GC etc.)
  - Track **history** of GC and I/O

- At the end of each interval, **decide how to resize the heap**
  - **Wait for resize** operation to take effect
- **Reduce total:** If total decreases → **repeat the same action**

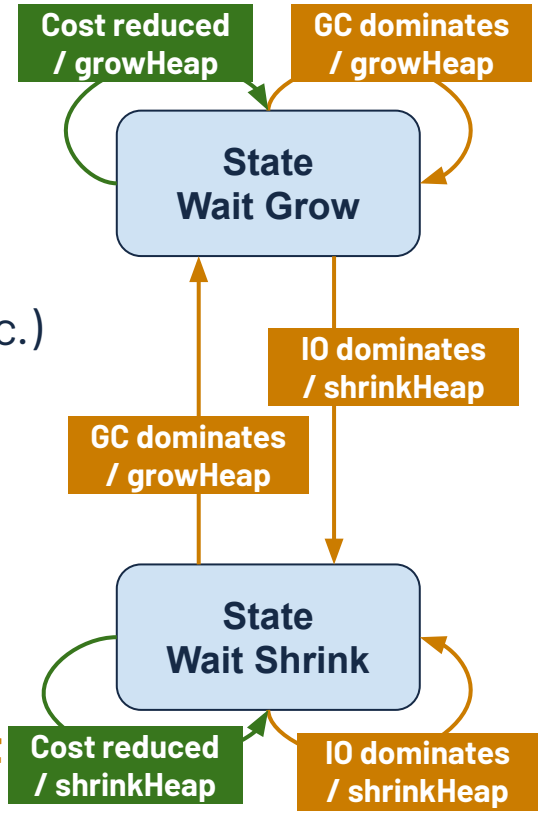


Deciding to grow or shrink the heap

- What is the impact of GC and I/O on application performance?
- FlexHeap tracks **lost potential for compute** by mutators
  - Due to GC or I/O, in **CPU cycles**
  - **Lost potential (cycles) = GC cost + I/O stalls**

- Divide execution in intervals, based on GC phases (young GC etc.)
  - Track **history** of GC and I/O

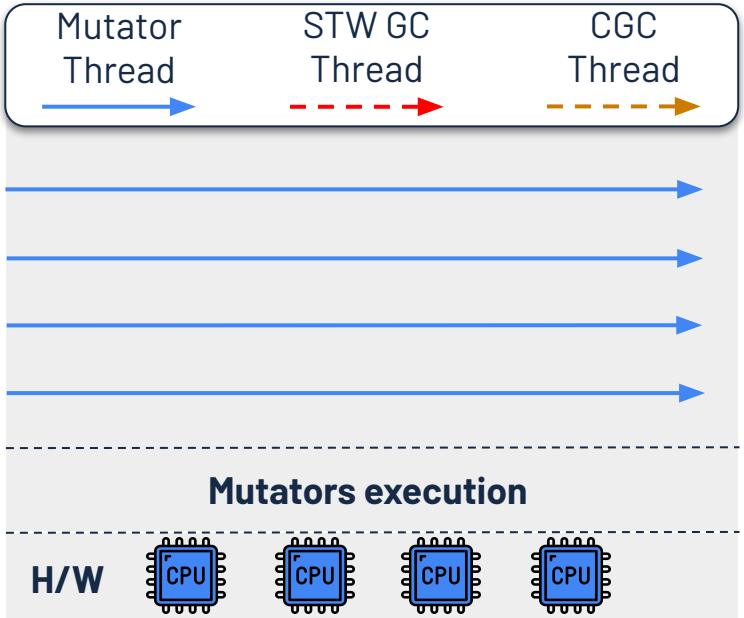
- At the end of each interval, **decide how to resize the heap**
  - **Wait for resize** operation to take effect
- **Reduce total:** If total decreases → **repeat the same action**
- **Reduce dominant:** If total increases → **identify dominant cost**
  - Dominant = based on difference, not absolute value



Estimate lost potential due to GC

- GC can be stop-the-world (STW) or concurrent to mutator threads (CGC)
  - GC cost = STW GC cost + CGC cost

- FlexHeap model converts both into **lost compute cycles** for mutators

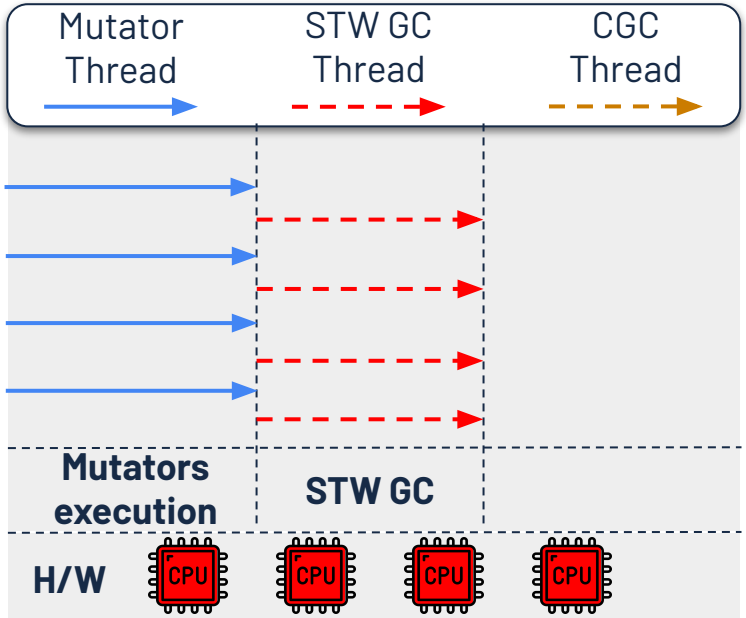


Estimate lost potential due to GC

- GC can be stop-the-world (STW) or concurrent to mutator threads (CGC)
  - GC cost = STW GC cost + CGC cost

- FlexHeap model converts both into **lost compute cycles** for mutators

- **Stop-the-world GC cost**
  - Lost potential for mutator work during pause
  - Pause time × **min(active mutators, CPU cores)**



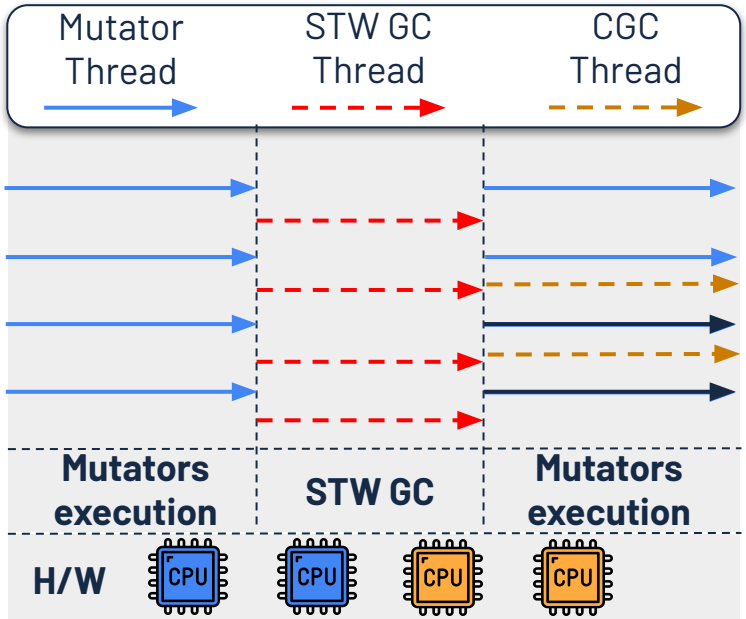
Estimate lost potential due to GC

- GC can be stop-the-world (STW) or concurrent to mutator threads (CGC)
  - GC cost = STW GC cost + CGC cost

- FlexHeap model converts both into **lost compute cycles** for mutators

- **Stop-the-world GC cost**
  - Lost potential for mutator work during pause
  - Pause time × **min(active mutators, CPU cores)**

- **Concurrent GC cost**
  - Lost potential if CGC runs **and** mutators wait
  - Count: if CGC runs while mutators are waiting



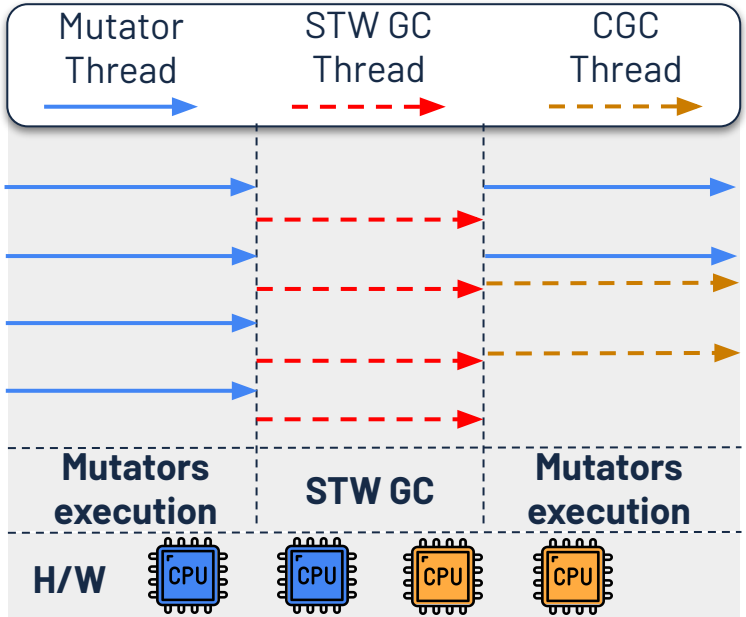
Estimate lost potential due to GC

- GC can be stop-the-world (STW) or concurrent to mutator threads (CGC)
  - GC cost = STW GC cost + CGC cost

- FlexHeap model converts both into **lost compute cycles** for mutators

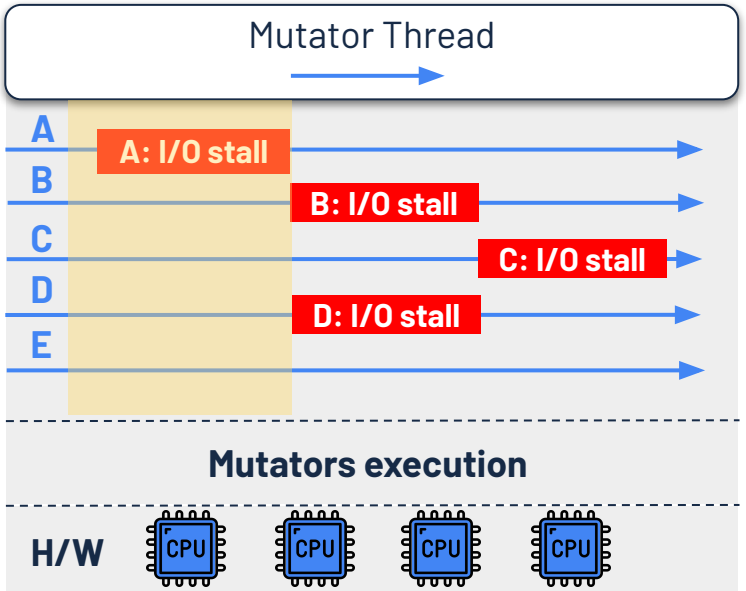
- **Stop-the-world GC cost**
  - Lost potential for mutator work during pause
  - Pause time × **min(active mutators, CPU cores)**

- **Concurrent GC cost**
  - Lost potential if CGC runs **and** mutators wait
  - Count: if CGC runs while mutators are waiting
  - Ignore: if enough idle cores exist for both



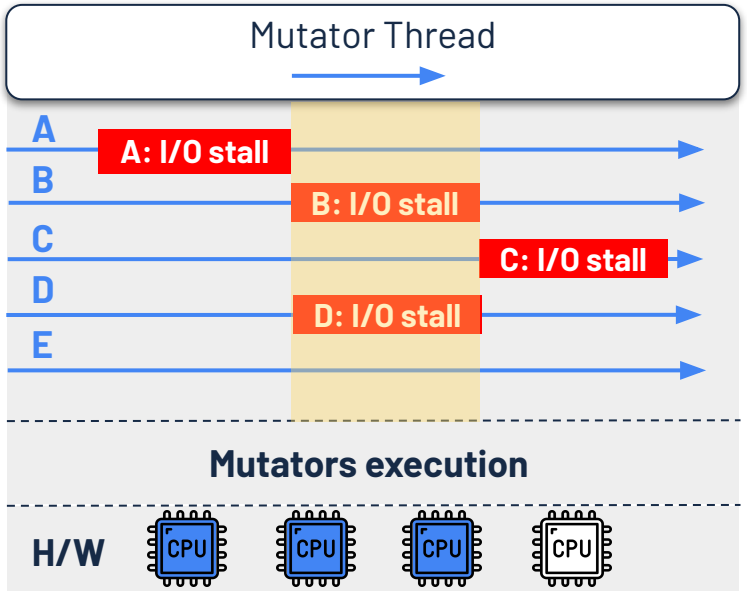
Estimate lost potential due to I/O

- I/O is **different** from GC cost
  - GC consumes CPU cores that mutators could use
  - I/O blocks mutators waiting for storage even when CPUs are busy ⟶ affects latency



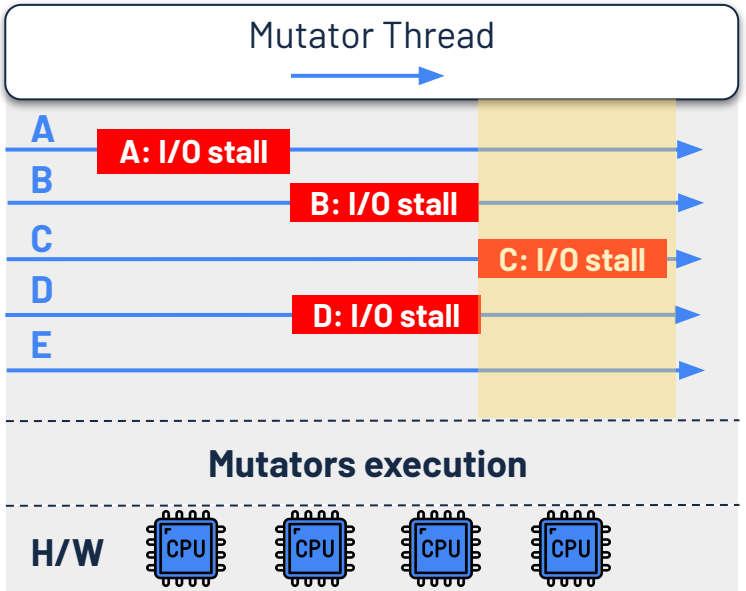
Estimate lost potential due to I/O

- I/O is **different** from GC cost
  - GC consumes CPU cores that mutators could use
  - I/O blocks mutators waiting for storage even when CPUs are busy ⟶ affects latency



Estimate lost potential due to I/O

- I/O is **different** from GC cost
  - GC consumes CPU cores that mutators could use
  - I/O blocks mutators waiting for storage even when CPUs are busy ⟶ affects latency

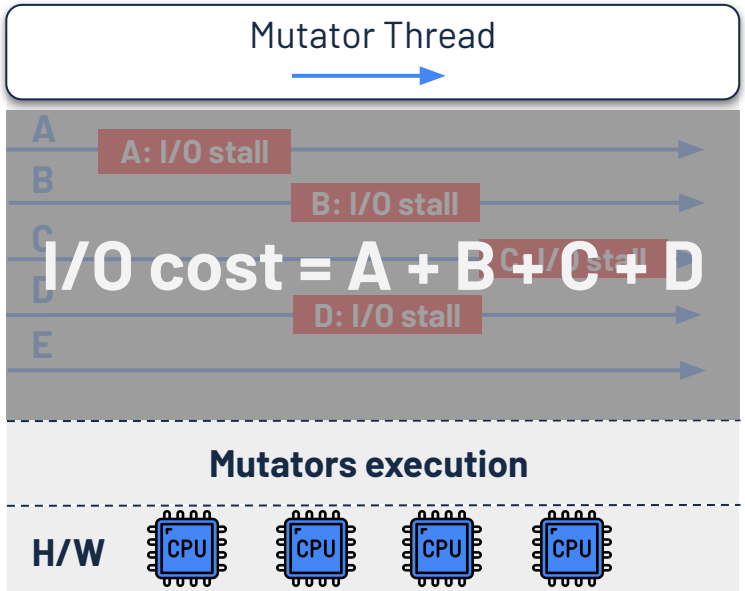


Estimate lost potential due to I/O

- I/O is **different** from GC cost
  - GC consumes CPU cores that mutators could use
  - I/O blocks mutators waiting for storage even when CPUs are busy

- I/O cost = sum of stalled mutators CPU time

- No kernel interface exposes per mutator I/O stall
  - iowait is **process-level**, not thread level
  - Kernel does not distinguish I/O stall time from scheduler delay
- FlexHeap approximates per-mutator I/O stall time
  - Uses eBPF on `sched_switch`
  - Tracks mutator time blocked in **D-state**

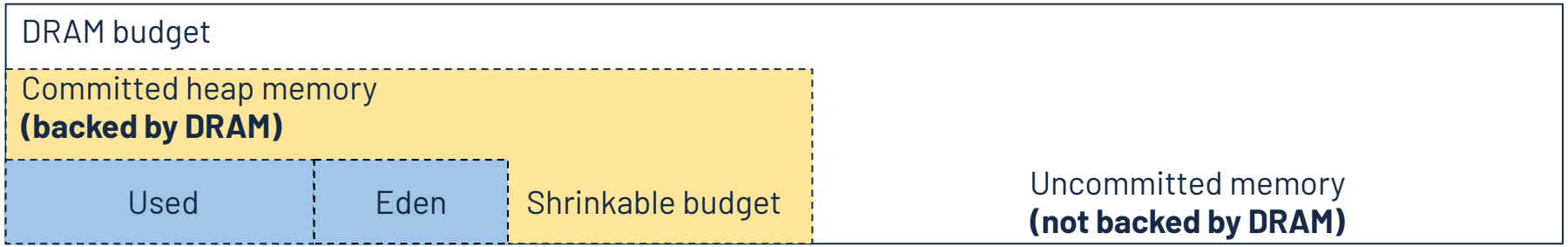


Estimate the resizing step

- Shrinking the heap without hurting G1 pause time goal
- G1 estimates the eden target size required to satisfy the paue-time goal
- FlexHeap ensures that the shrink step will leave enough space for the required eden
  - Shrinkable budget = free heap size - eden target size

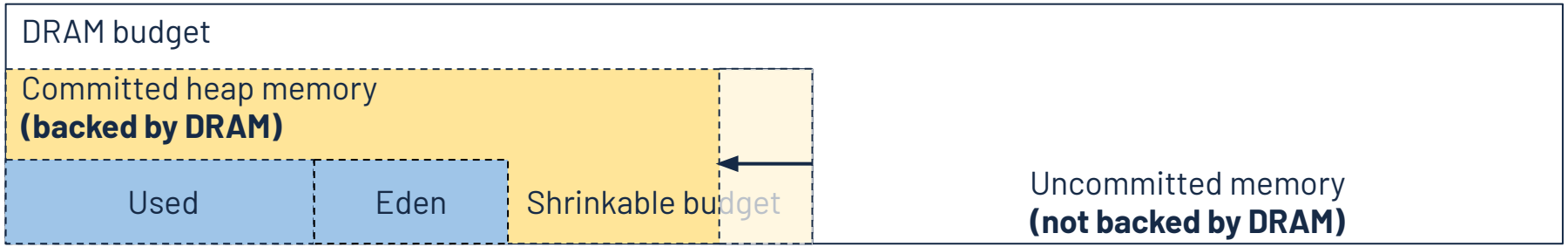
Estimate the resizing step

- Shrinking the heap without hurting G1 pause time goal
- G1 estimates the eden target size required to satisfy the paue-time goal
- FlexHeap ensures that the shrink step will leave enough space for the required eden
  - Shrinkable budget = free heap size - eden target size



Estimate the resizing step

- Shrinking the heap without hurting G1 pause time goal
- G1 estimates the eden target size required to satisfy the paue-time goal
- FlexHeap ensures that the shrink step will leave enough space for the required eden
  - Shrinkable budget = free heap size - eden target size



- FlexHeap uses a sigmoid function to **choose how much of that budget to release**
  - **Proportional step to estimated cost change**, except for extremes

Outline

- Motivation
- FlexHeap design
  - Deciding to grow or shrink the heap
  - Estimating the impact of GC and I/O stalls
  - Deciding the resizing step
- Evaluation
- Conclusions

Methodology

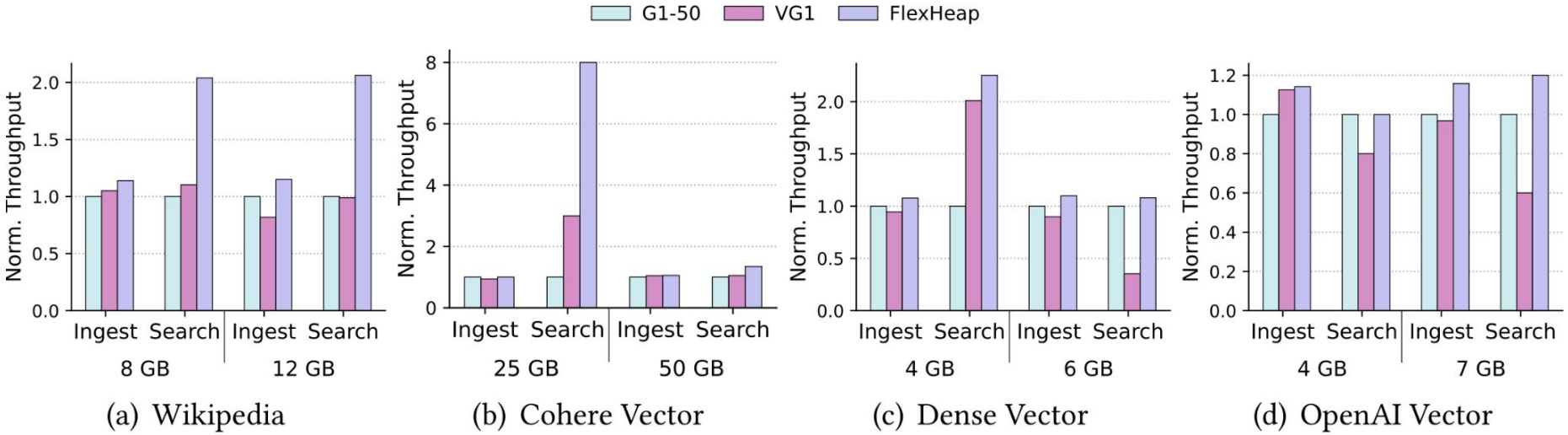
- We implement FlexHeap in OpenJDK 21

- We identify workloads that exhibit continuously changing heap + I/O behavior
  - Spark running Spark-SQL (TPC-DS workloads)
  - Elasticsearch running ESRally

- We use two main baselines
  - G1: Resize the heap to reduce GC cost (latest resizing mechanism from OpenJDK 26)
  - VG1: Resize the heap to reduce server memory use

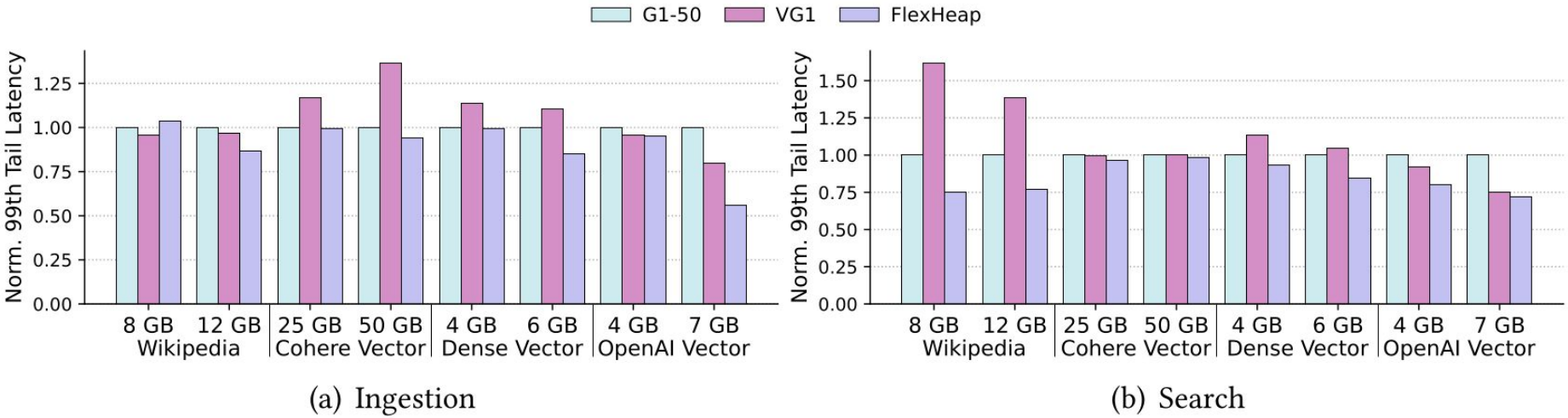
- Server with 2 TB NVMe SSD, 256 GB DRAM, and 32 cores
  - Limit DRAM capacity with cgroup

FlexHeap improves Elasticsearch throughput



- FlexHeap improves ingestion throughput
  - between 5-20% compared to G1-50 and by up to 30% compared to VG1
- FlexHeap improves search throughput
  - by up to 7x compacted to G1-50 and by up to 5x compared to VG1

FlexHeap improves Elasticsearch latency

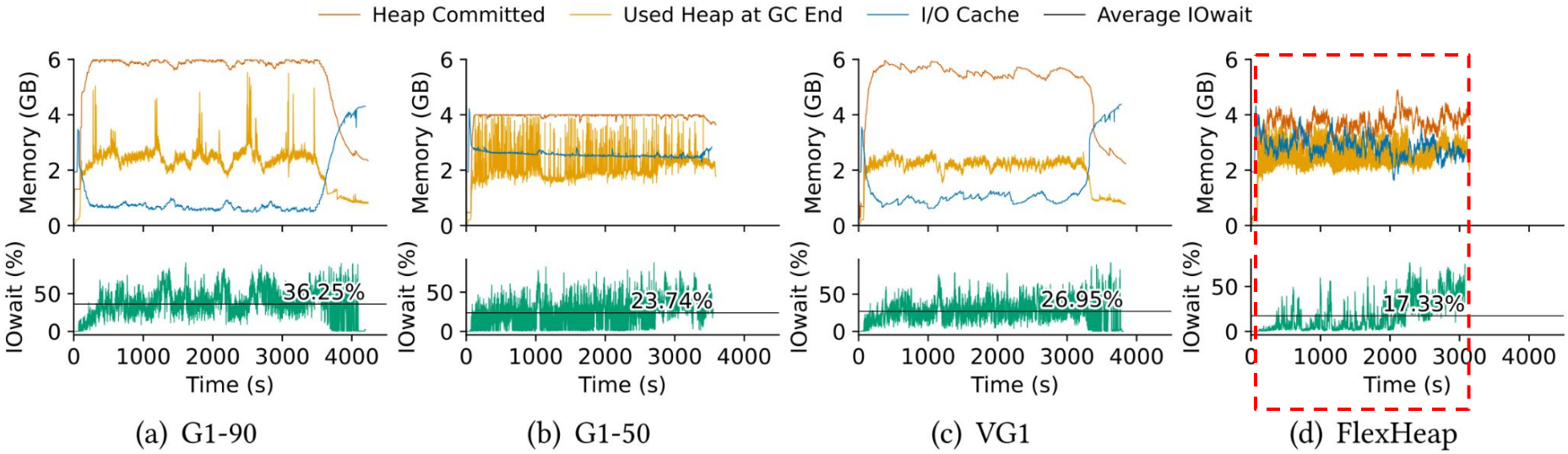


(a) Ingestion

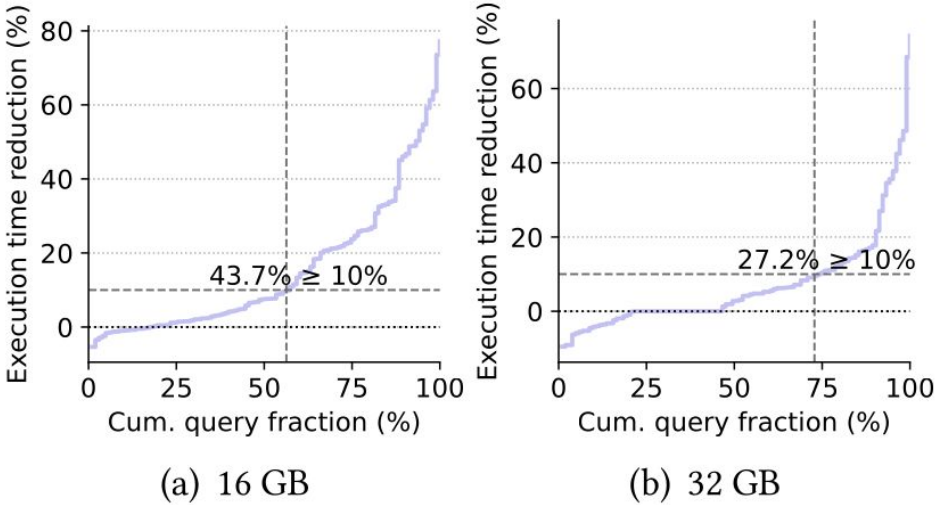
(b) Search

- For ingestion tasks, FlexHeap improves p99
  - by up to 10% compared to G1-50 and by up to 30% compared to VG1
- For search tasks, FlexHeap improves p99
  - by up to 25% compared to G1-50 and by up to 2x compared to VG1

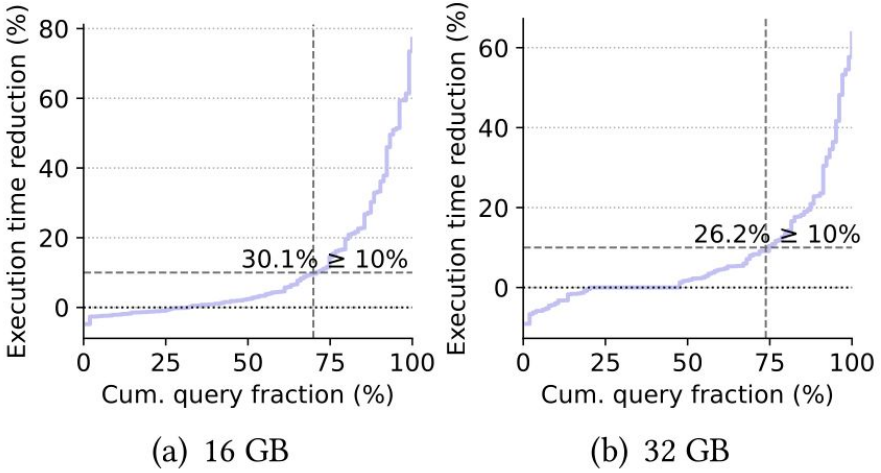
FlexHeap adapts to application behavior



FlexHeap improves Spark-SQL performance



FlexHeap vs G1



FlexHeap vs VG1

- 43.7% and 27.2% of queries have execution time reduction >= 10% compared to G1
- 30% and 26% of queries have execution time reduction >= 10% compared to VG1

Key Takeaway

- JVM-based search and analytics systems need memory for both heap and I/O purposes
- Partitioning memory to reduce both GC and I/O cost is a challenging problem
  - Existing heap resizing mechanisms do not consider I/O cost
- FlexHeap partitions DRAM by estimating the lost potential from GC and I/O
  - Uses GC cost and I/O stalls to estimate lost potential
- Takes two decisions dynamically at GC-phase boundaries (e.g. young GC)
  1. Grow or shrink the heap?
  2. By how much?
- FlexHeap improves throughput by up to 33% and 50% compared to G1 and VG1

FlexHeap: Dynamic I/O-Aware Heap Resizing for Managed Applications

www.github.com/CARV-ICS-FORTH/flexheap

@jackkolokasis | www.jackkolokasis.com

We thankfully acknowledge the support of the European Commision projects
AERO (GA No 101092850) and EUPEX (GA No 101033975)