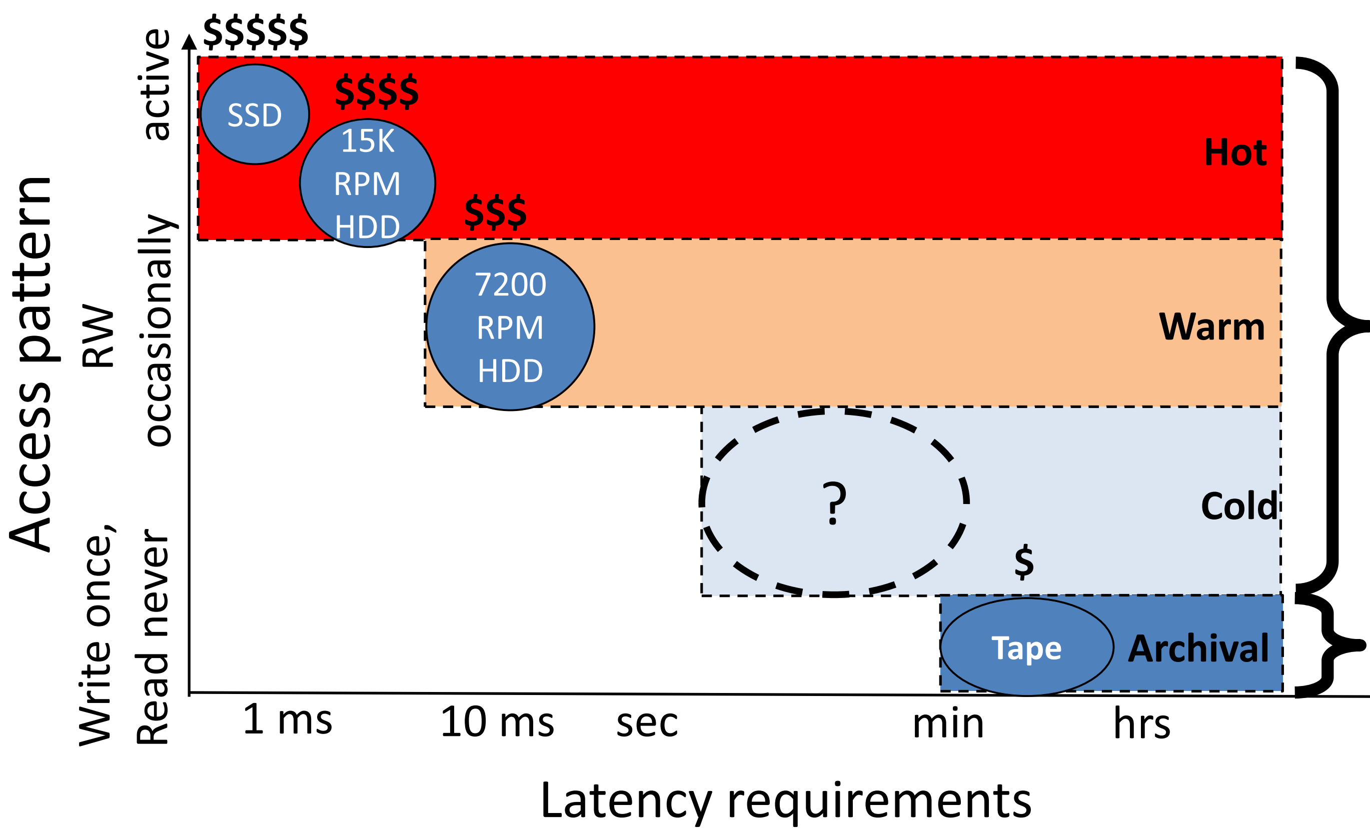
Pelican: A building block for exascale cold data storage

Shobana Balakrishnan, Richard Black, Austin Donnelly, Paul England, Adam Glass, Dave Harper, *Sergey Legtchenko*, Aaron Ogus, Eric Peterson, Antony Rowstron

Microsoft Research

Background: cold data in the cloud

- Cold data: **“written once – read rarely”** access pattern
- Large fraction of stored data

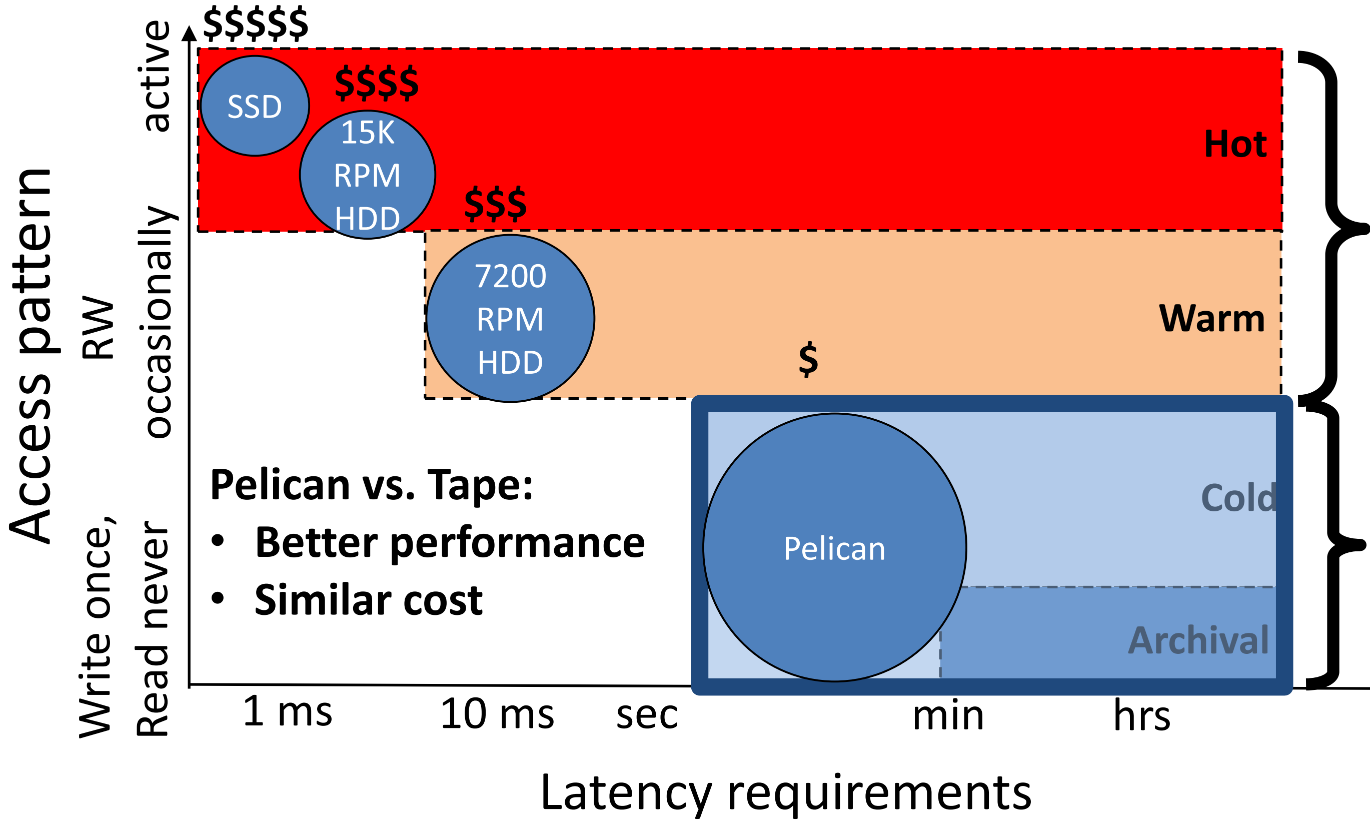


Hot tier
- Provisioned for peak
- High throughput
- Low latency
- High cost

Archive tier
- Low cost
- High latency (hours)

Background: cold data in the cloud

- Cold data: **“written once – read rarely”** access pattern
- Large fraction of stored data



Hot tier
- Provisioned for peak
- High throughput
- Low latency
- High cost

Cold tier:
- High density
- Low hardware cost
- Low operating cost
- Latency lower than tape

Right-provisioning

- Provision resources **just** for the cold data workload:
  - **Disks:**
    - Archival and SMR instead of commodity
  - **Power**
  - **Cooling**
  - **Bandwidth**

  } Enough for bandwidth required by workload instead of for all disks spinning

  - **Servers:**
    - Enough for data management instead of 1 server/ 40 disks

- Benefits of removing unnecessary resources:
  - High density of storage
  - Low hardware cost
  - Low operating cost (capped performance)

Pelican: rack-scale appliance for cold data

- **Converged design:**
  - Power, cooling, mechanical, storage & software co-designed
- **Right-provisioned for cold data workload:**
  - Resources for **just** workload requirements
- **At most 8% disks spun up**
- **2 servers**
- **No Top of Rack switch**
  - 4x 10Gbps uplinks from the servers
- **1,152 disks in 52U: 22 disks/U**

Other disk-based storage:
➡ Up to 15/U

- **5+ PB of raw storage**

Pelican rack prototype

Pelican: rack-scale appliance for cold data

- **Converged design:**
  - Power, cooling, mechanical, storage & software co-designed
- **Right-provisioned for cold data workload:**
  - Resources for **just** workload requirements
- **At most 8% disks spun up**
- **2 servers**
- **N**
- **1**
- **5**

Pelican rack prototype

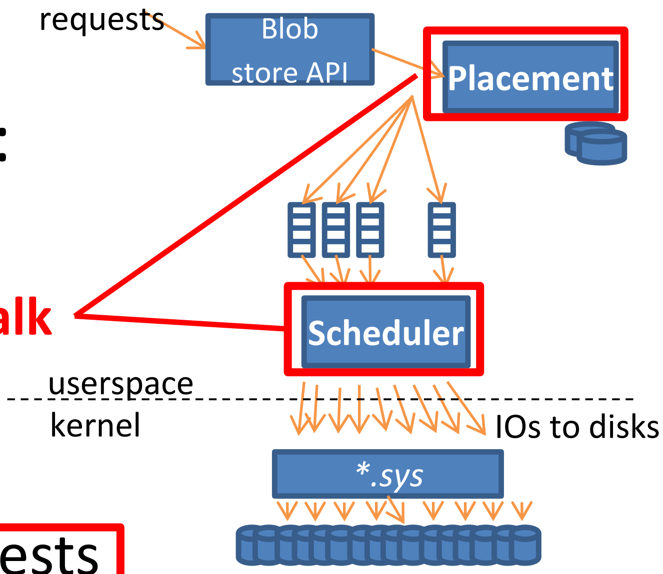
- **Total cost of ownership comparable to tape**
- **Lower latency than tape**

➢ Challenging resource limitations managed in software

Pelican storage stack: handling right-provisioning

- Co-designed with hardware
- Constraints over sets of active disks:
  - Hard: *power, cooling, failure domains*
  - Soft: *bandwidth, vibration*
- Software challenges:
  - **Data placement:** concurrency of requests
  - **IO scheduling:** minimize spin ups, fairness
  - **Recovery:** minimize window of vulnerability

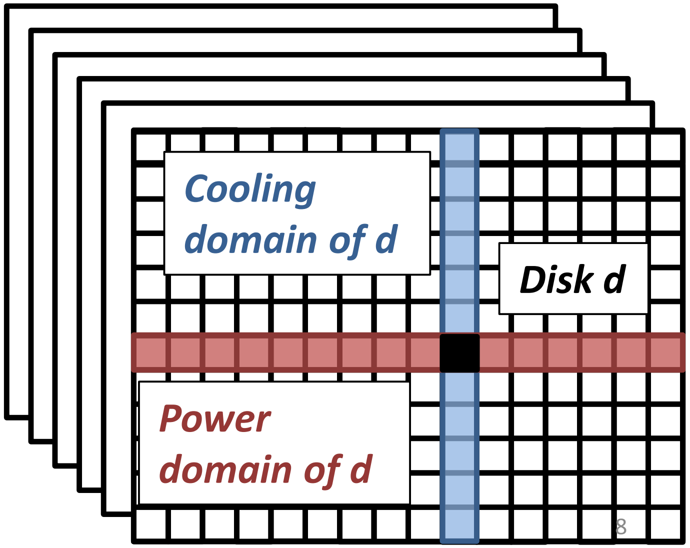
In this talk



Impact of right provisioning on resources

- **Systems provisioned for peak performance:**
  - Any disk can be active at any time
- **Right-provisioned system:**
  - Disk part of a *domain* for each resource
  - Domain supplies limited resources
  - Disk active *if* enough resources in all its domains
- **Pelican domains:**
  - power, cooling, vibration, bandwidth
- **Resource limitations:**
  - 2 active out of 16 per power domain
  - 1 active out of 12 per cooling domain
  - 1 active out of 2 per vibration domain

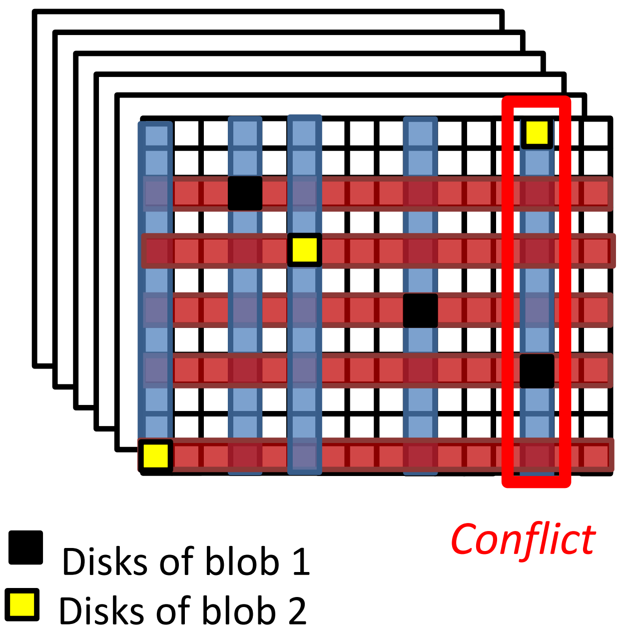
Rack: 3D array of disks



Data placement: maximizing request concurrency

- Blob erasure-encoded on a ***set*** of concurrently active disks
- **In fully provisioned systems:**
  - Any two sets can be active
  - No impact of placement on concurrency
- **In right-provisioned systems:**
  - Sets can conflict in resource requirements
  - Conflicting cannot be concurrently active
  - **Challenge: form sets that minimize $P_{Conflict}$**

First approach: random placement

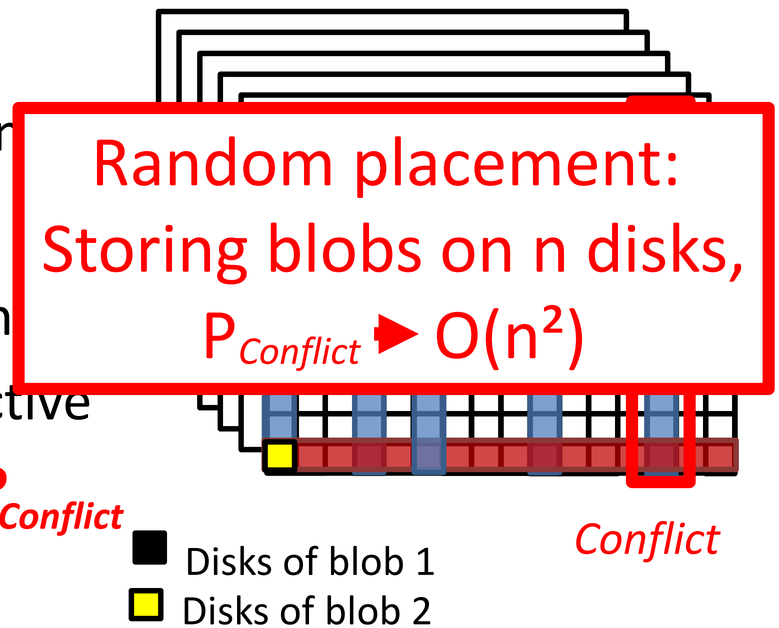


Rack: 3D array of disks

Data placement: maximizing request concurrency

- Blob erasure-encoded on a ***set*** of concurrently active disks
- **In fully provisioned systems:**
  - Any two sets can be active
  - No impact of placement on concurren
- **In right-provisioned systems:**
  - Sets can conflict in resource requirem
  - Conflicting cannot be concurrently active
  - **Challenge: form sets that minimize $P_{Conflict}$**

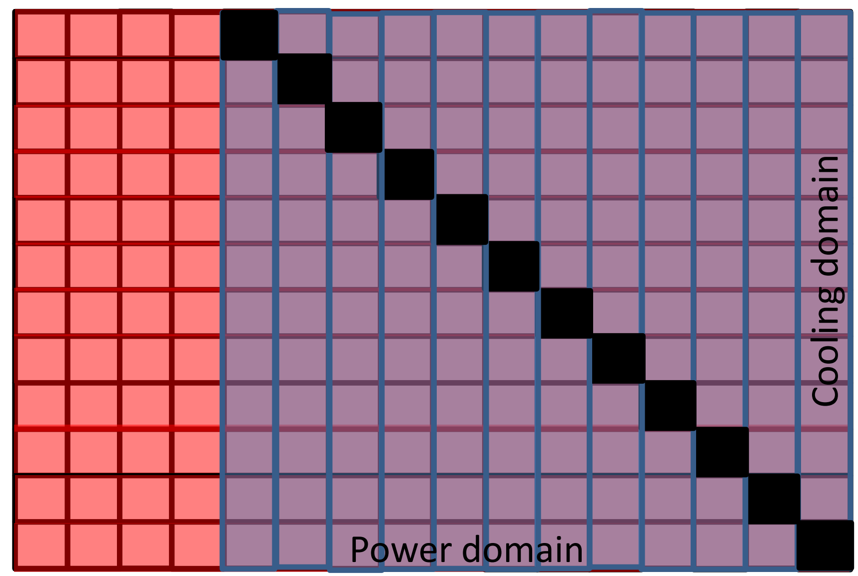
First approach: random placement



Rack: 3D array of disks

Pelican data placement

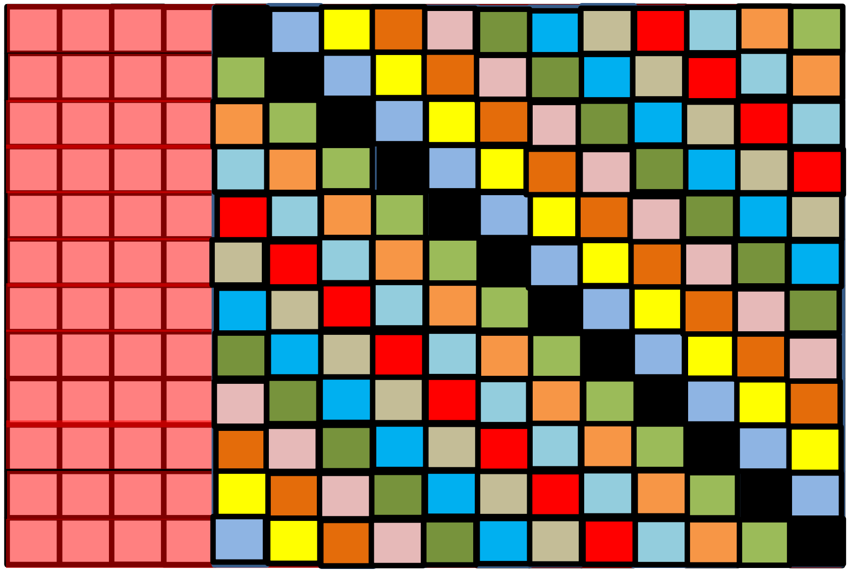
- **Intuition**: concentrate all conflicts over a few sets of disks
- Statically partition disks in **groups** in which disks can be concurrently active
- Property:
  - Either fully conflicting
  - Or fully independent
- Blob is stored in one group
  - $P_{Conflict} \rightarrow O(n)$
- Groups encapsulate constraints:
  - Unit of IO scheduling
  - No constraint management at runtime



Schematic side-view of the rack

Pelican data placement

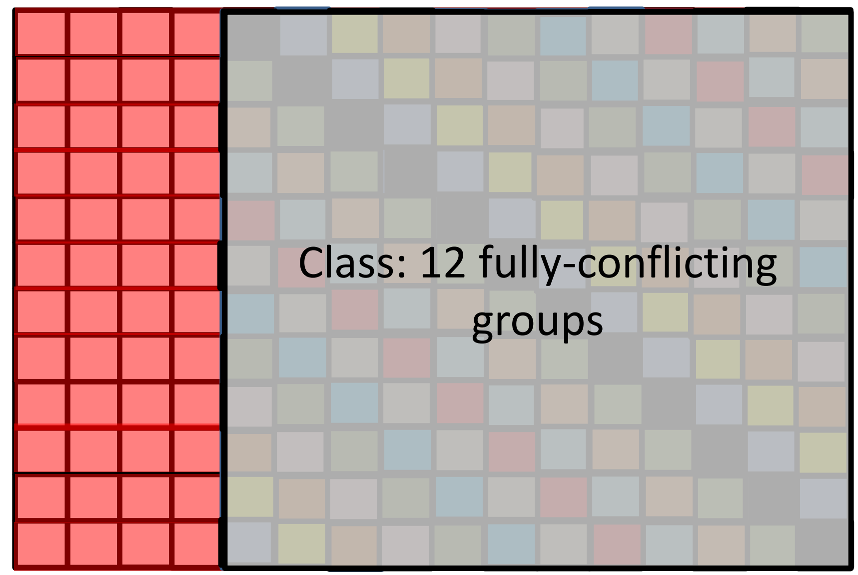
- **Intuition**: concentrate all conflicts over a few sets of disks
- Statically partition disks in **groups** in which disks can be concurrently active
- Property:
  - Either fully conflicting
  - Or fully independent
- Blob is stored in one group
  - $P_{Conflict} \rightarrow O(n)$
- Groups encapsulate constraints:
  - Unit of IO scheduling
  - No constraint management at runtime



Schematic side-view of the rack

Pelican data placement

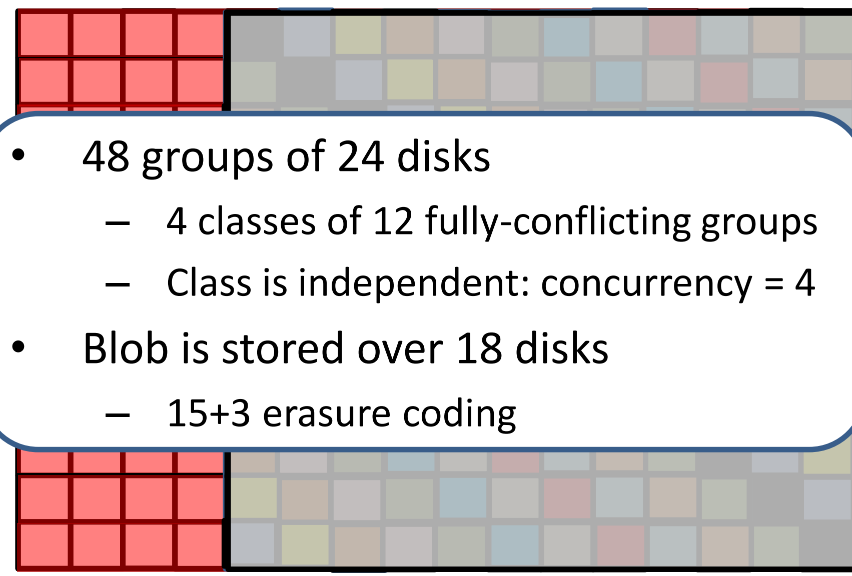
- **Intuition**: concentrate all conflicts over a few sets of disks
- Statically partition disks in **groups** in which disks can be concurrently active
- Property:
  - Either fully conflicting
  - Or fully independent
- Blob is stored in one group
  - $P_{Conflict} \rightarrow O(n)$
- Groups encapsulate constraints:
  - Unit of IO scheduling
  - No constraint management at runtime



Schematic side-view of the rack

Pelican data placement

- **Intuition**: concentrate all conflicts over a few sets of disks
- Statically partition disks in **groups** in which disks can be concurrently active
- Property:
  - Either fully conflicting
  - Or fully independent
- Blob is stored in one group
  - $P_{Conflict} \rightarrow O(n)$
- Groups encapsulate constraints:
  - Unit of IO scheduling
  - No constraint management at runtime



- 48 groups of 24 disks
  - 4 classes of 12 fully-conflicting groups
  - Class is independent: concurrency = 4
- Blob is stored over 18 disks
  - 15+3 erasure coding

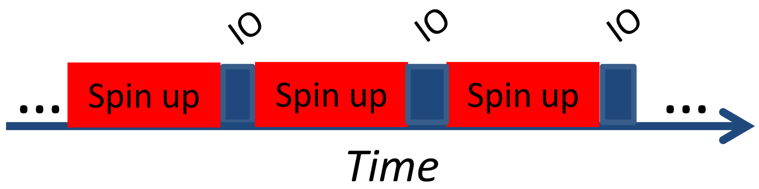
Schematic side-view of the rack

IO Scheduling: “spin up is the new seek”

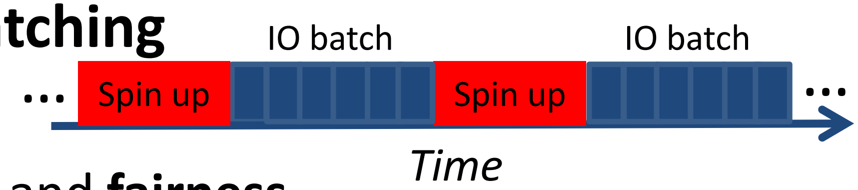
Four independent schedulers

Each scheduler: 12 groups, only one can be active

- **Naïve scheduler: FIFO**
  – Avg. group activation time: 14.2 sec
  – High probability of spinup after each request
  – Time is spent doing spinups!



- **Pelican scheduler: Request batching**
  – Limit on maximum re-ordering
  – Trade-off between **throughput** and **fairness**
  – Weighted fair-share between client and rebuild traffic



Outline: challenges of right-provisioning

1. **Challenge:** conflicts in domains reduce concurrency
   Solution: *constraint-aware data placement*
2. **Challenge:** "spinup is the new seek"
   Solution: *IO scheduler that amortizes spinup latency*

Last part of the talk:

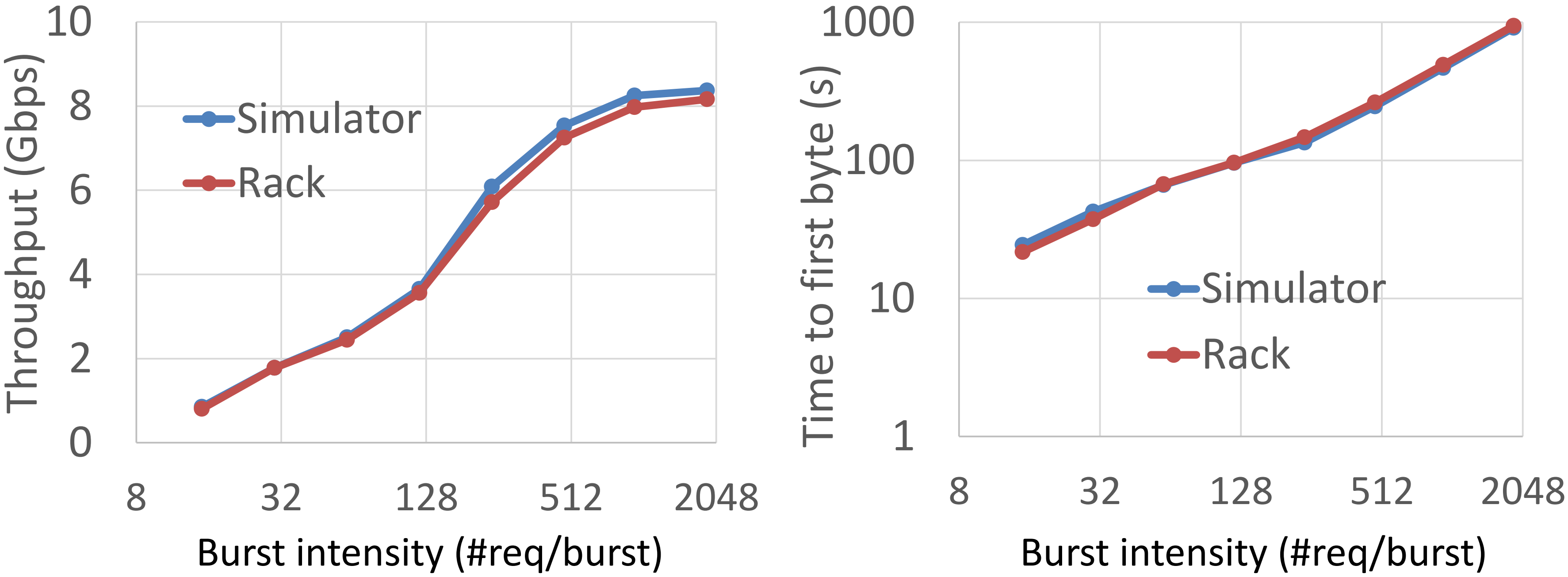
Performance impact of right-provisioning

Evaluating impact of right-provisioning

- **Pelican vs. rack with all disks active (called FP)**
- **Cross-validated discrete-event simulator**
- **Metrics (more in the paper):**
  - Rack throughput
  - Latency (time to first byte)
  - Power consumption
- **Open loop workload:**
  - Poisson arrival process
  - Read requests on 1GB blobs
  - Varying workload rate up to 8 requests/s

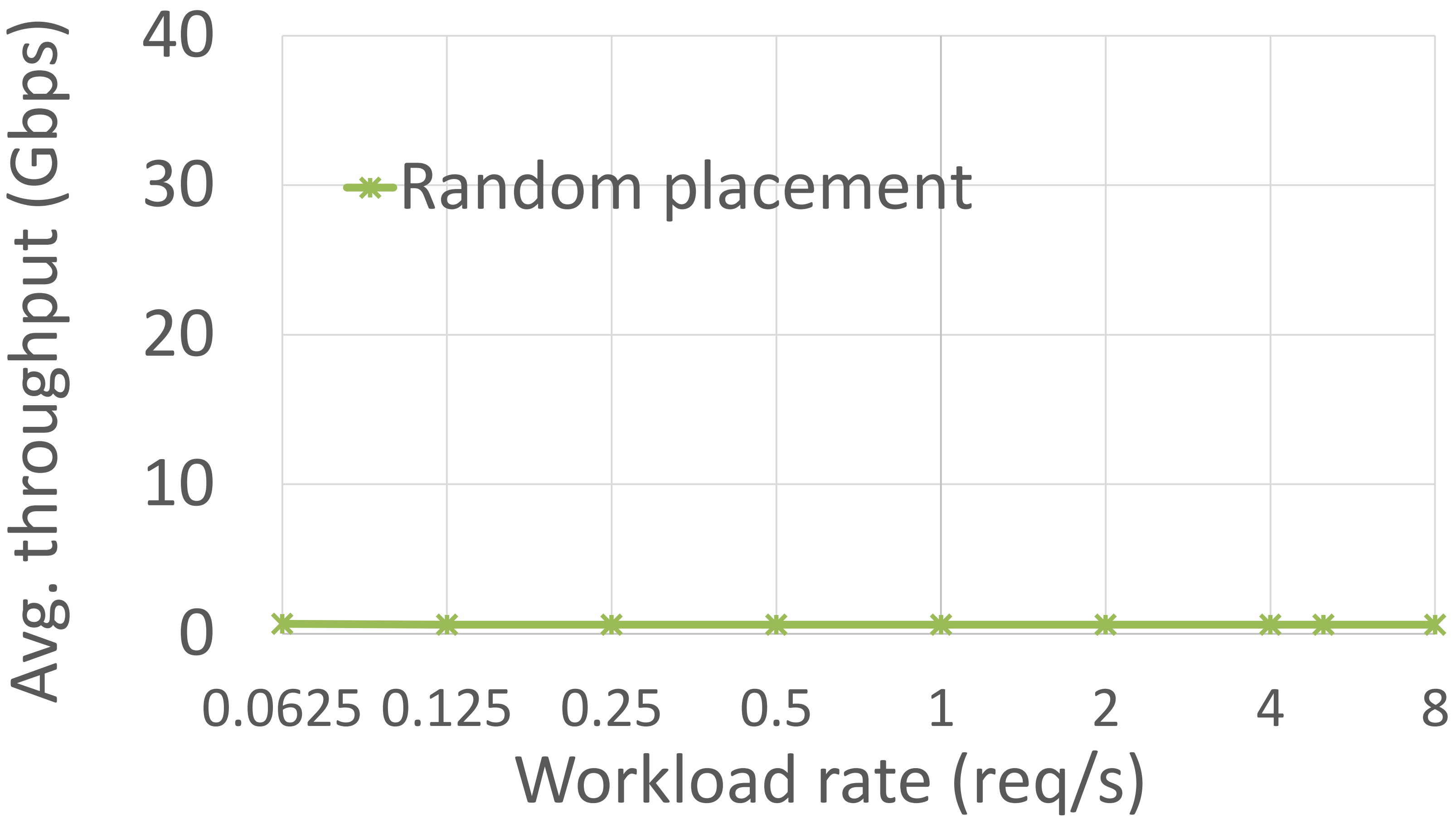
First step: simulator cross-validation

- Burst workload, varying burst intensity

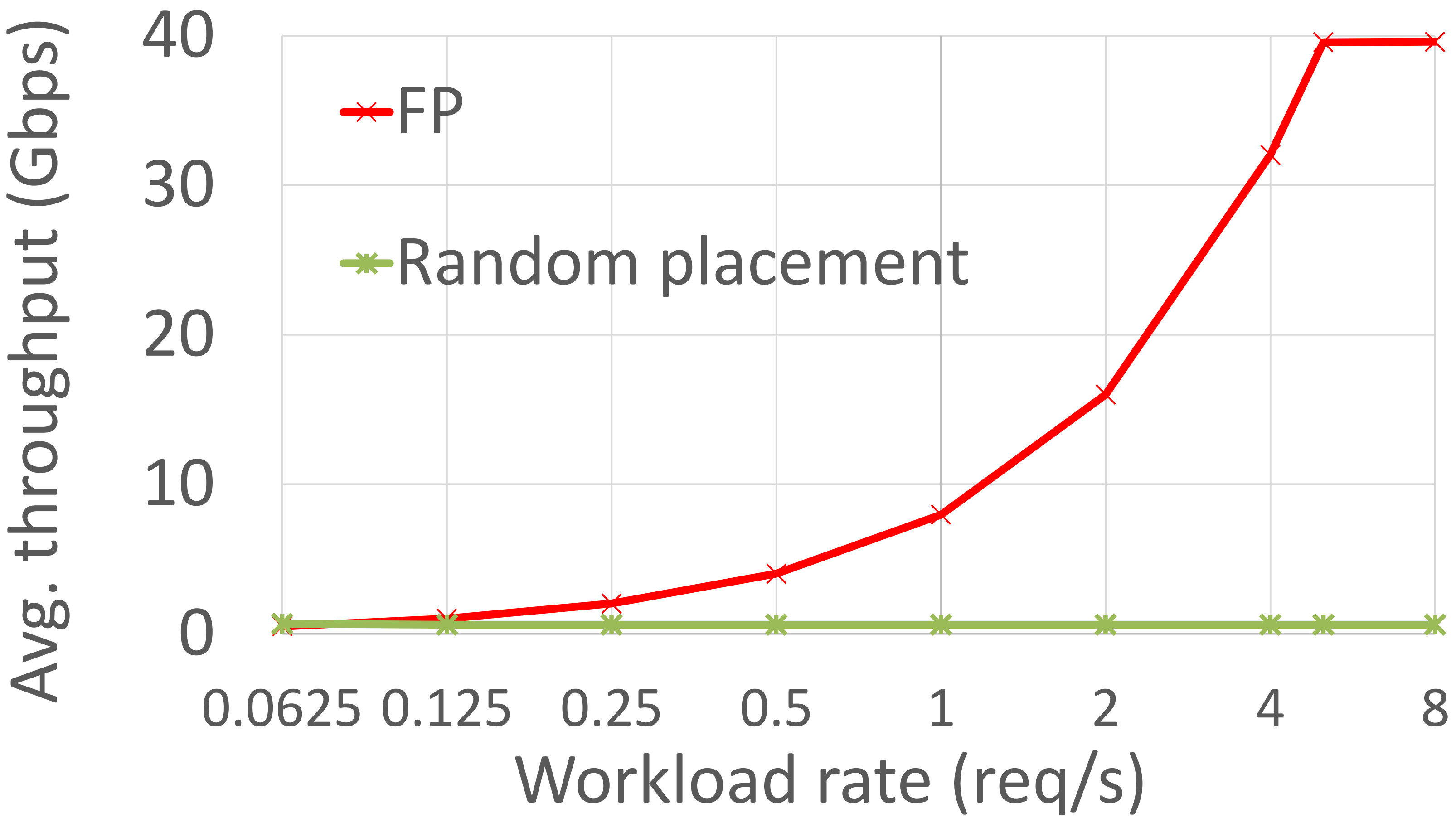


Simulator accurately predicts real system behaviour for all metrics.
See paper for more results.

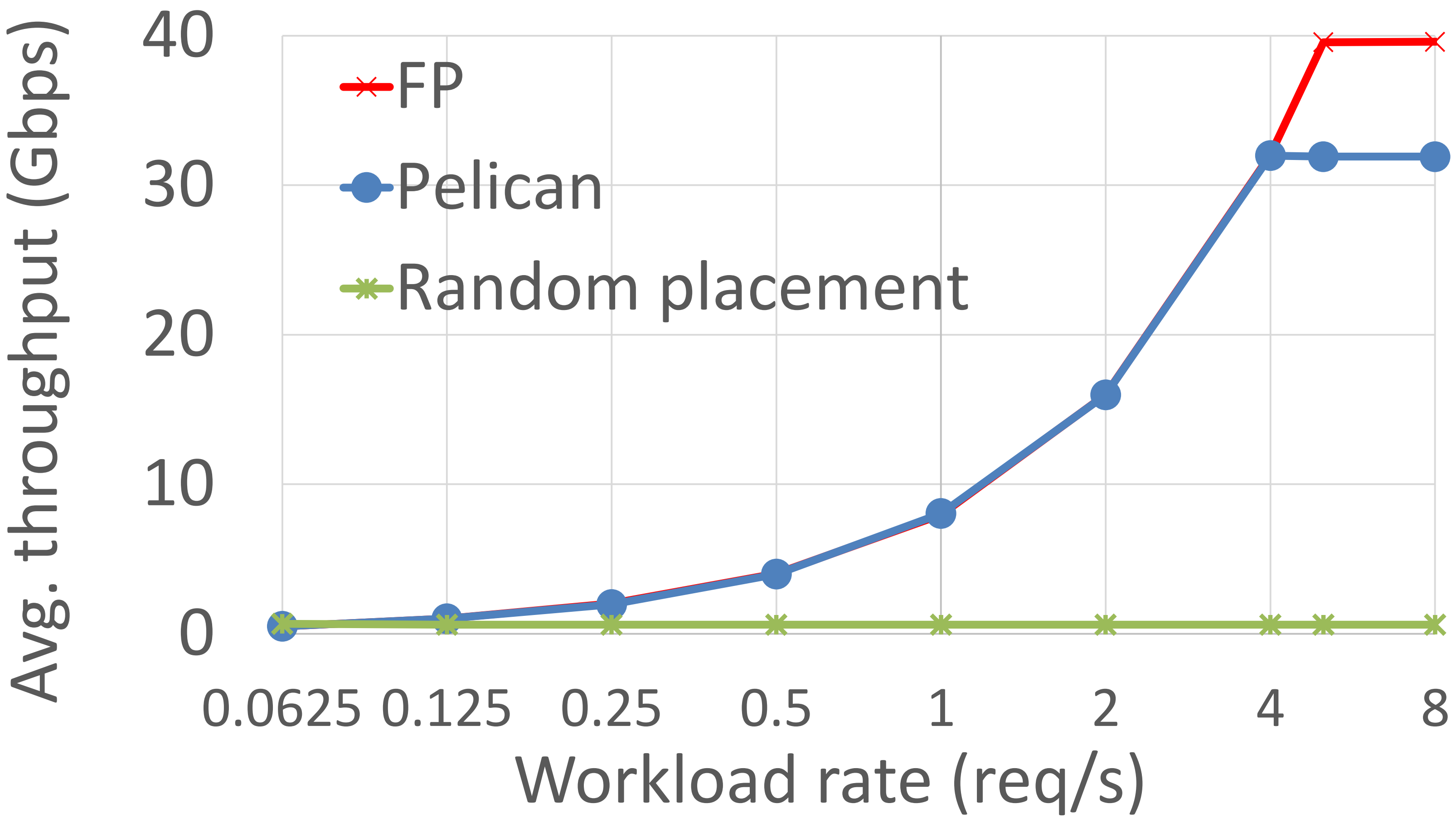
Rack throughput



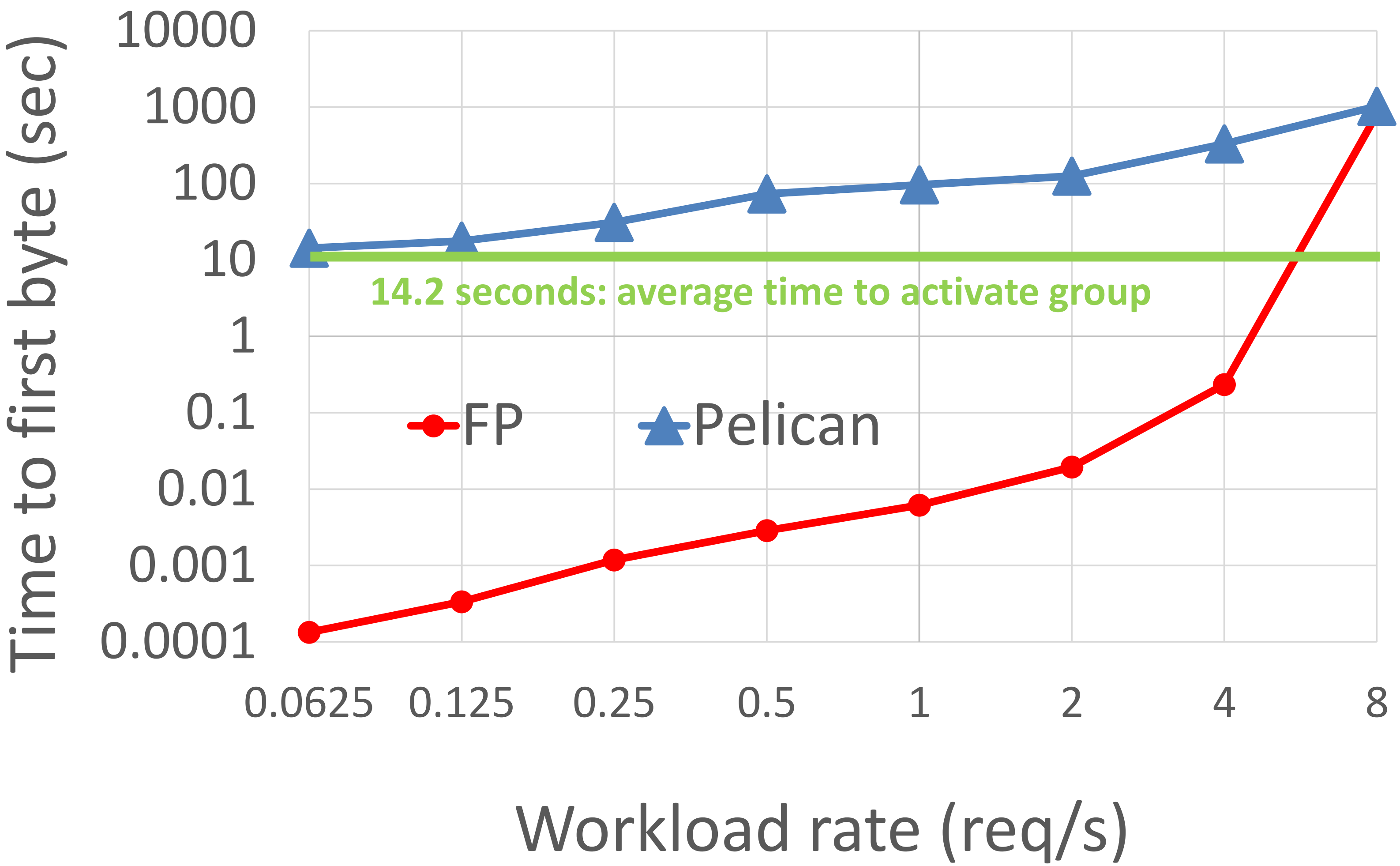
Rack throughput



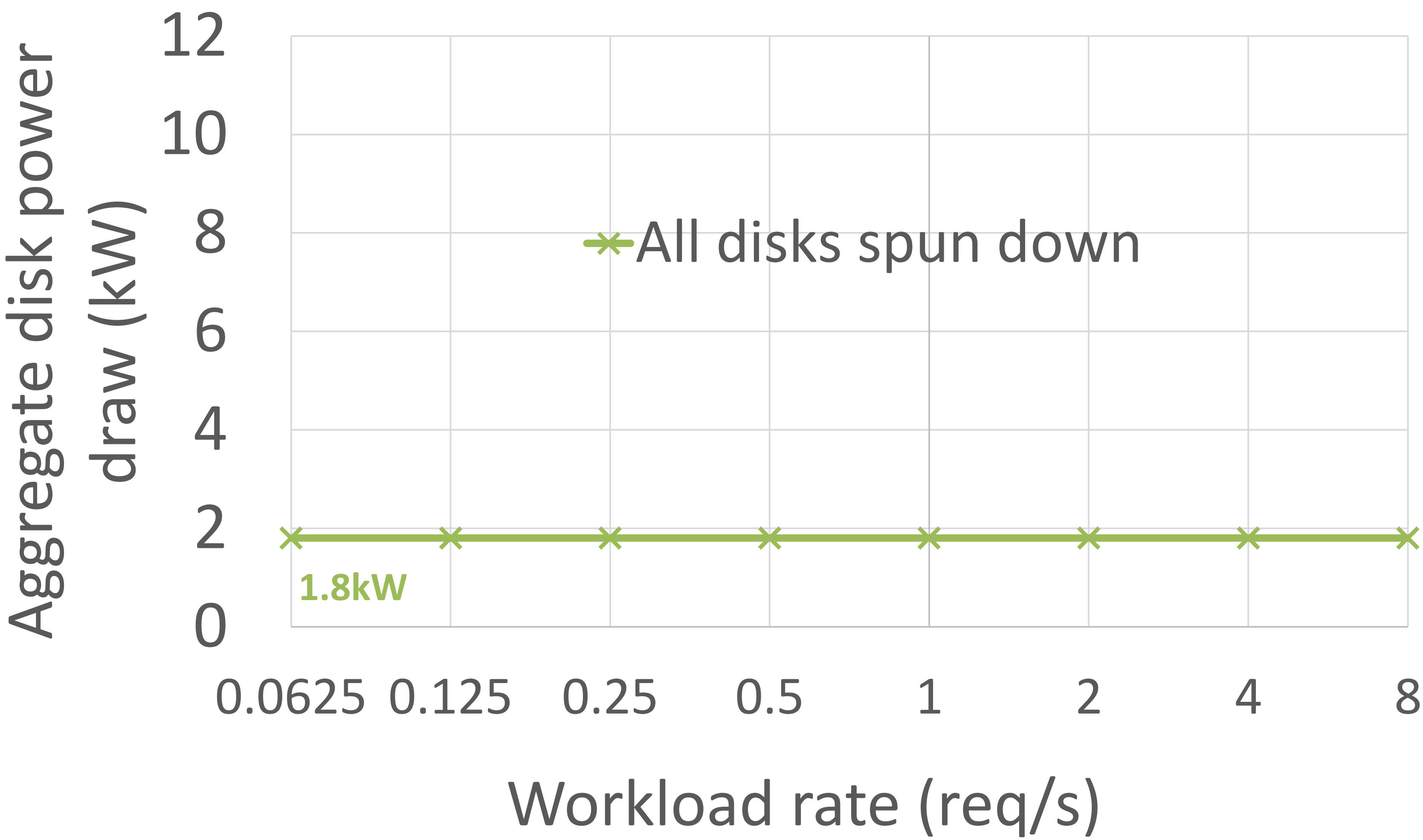
Rack throughput



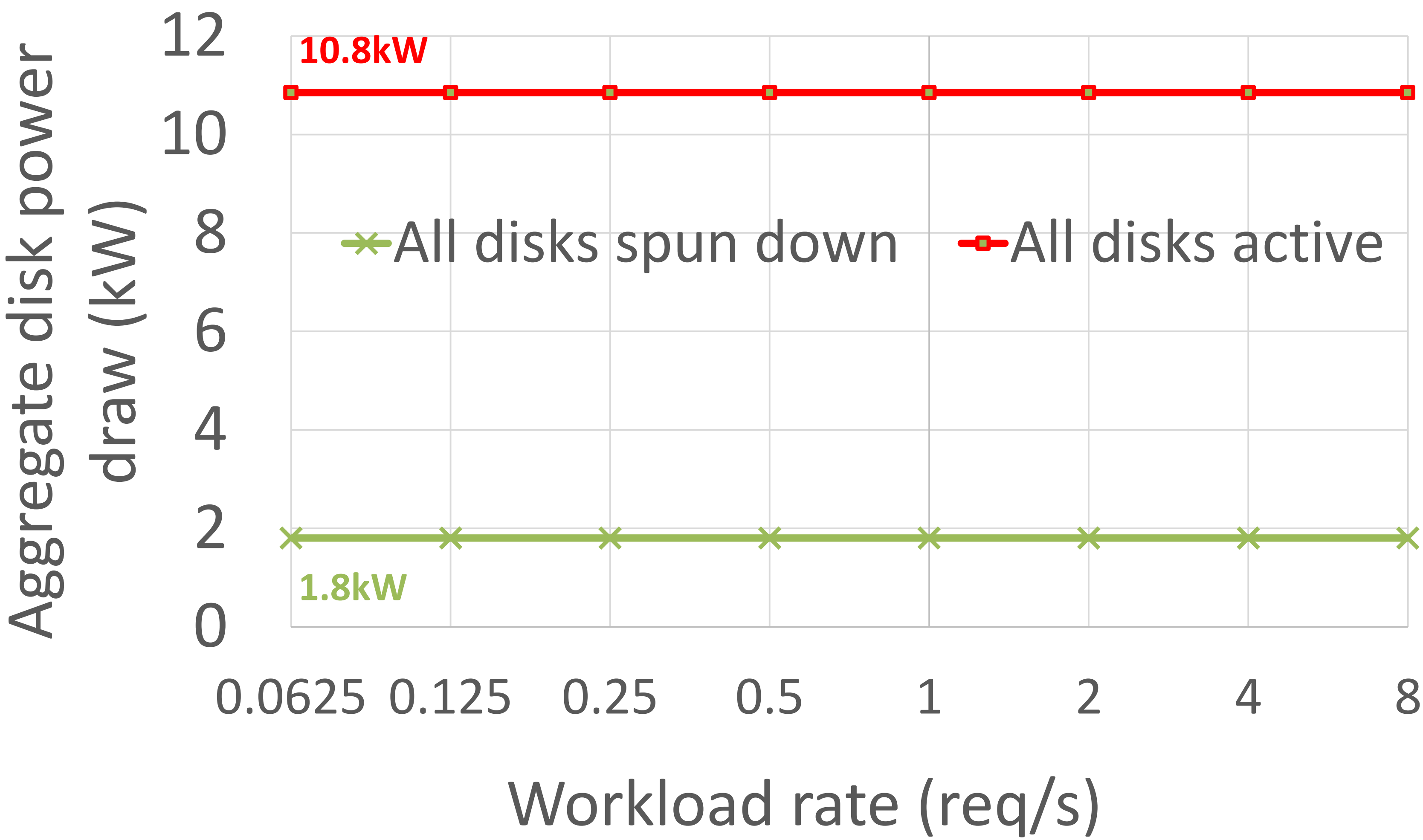
Time to first byte



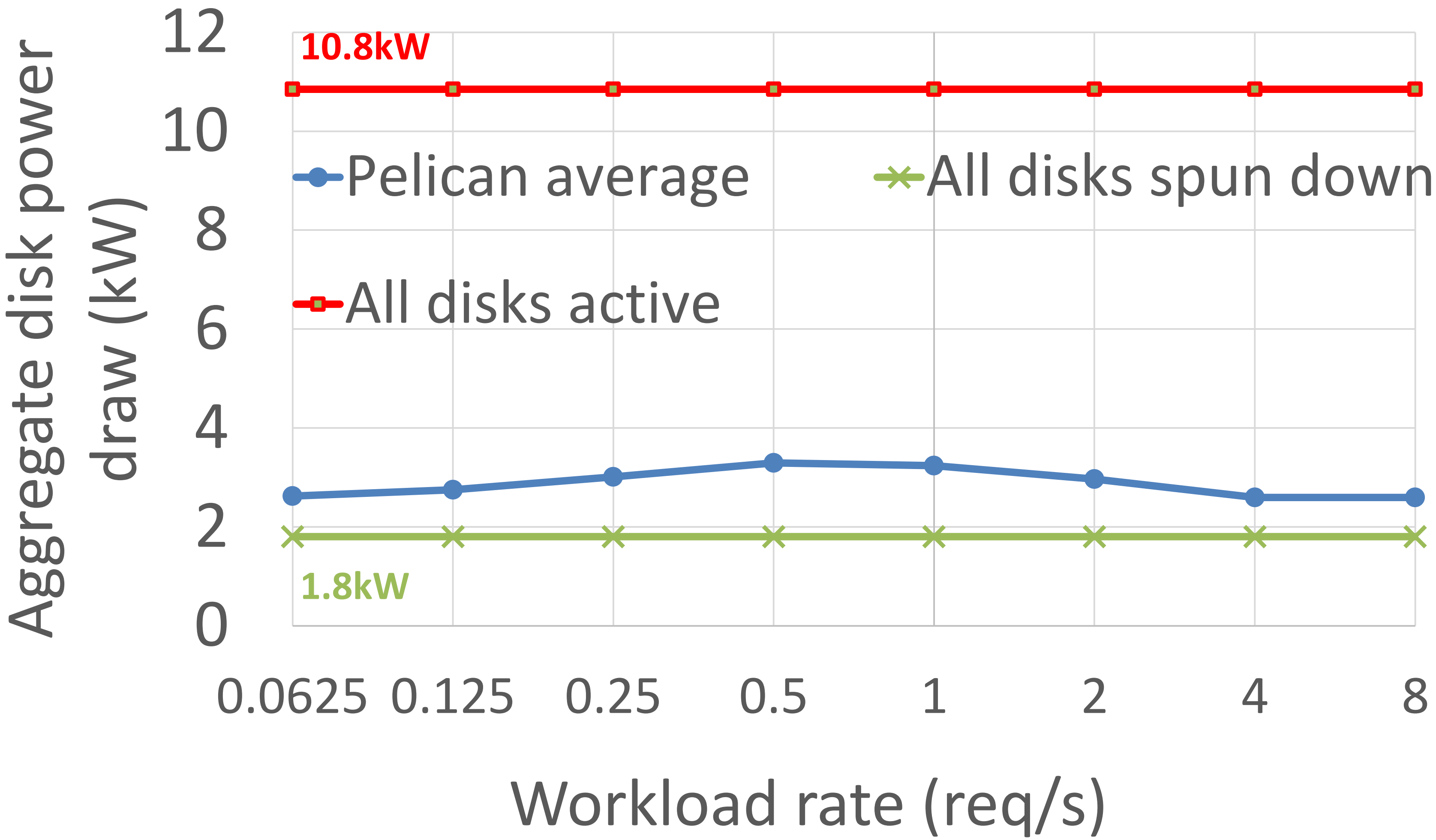
Power consumption



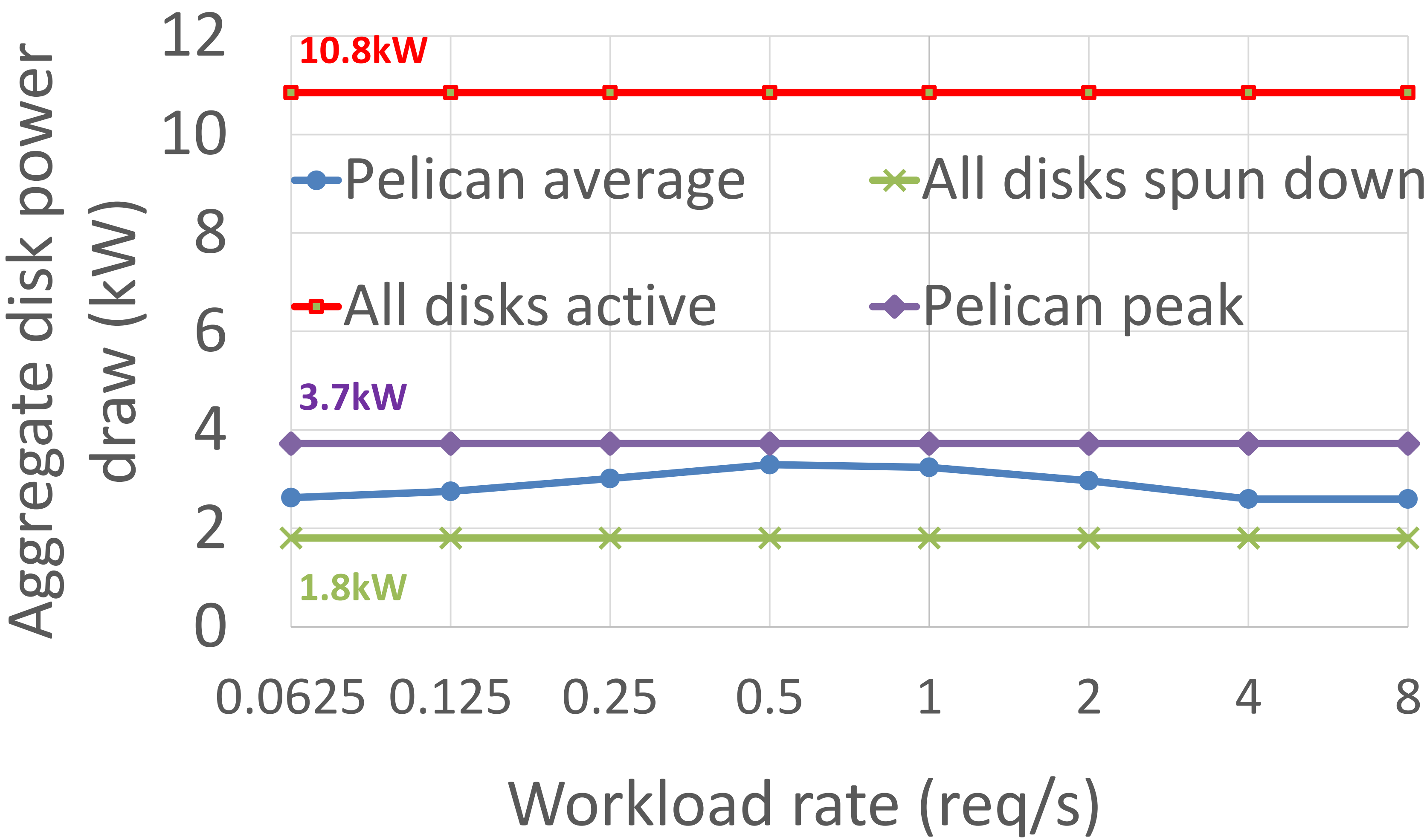
Power consumption



Power consumption



Power consumption: **3x lower peak**



Conclusion

- Rack-scale hardware/software co-design
  - Storage right-provisioned for cold data workload
  - Efficient constraint-aware software storage stack
- Prototype rack storing 5+ PB of raw data in 52U
- Challenging design process:
  - Many constraints to handle manually
  - Sensitive to hardware changes
- Follow up work:
  - "**Flamingo: Synthesizing cold storage stacks for Pelican-like systems**"
  - See our poster in tonight's session