FAWN

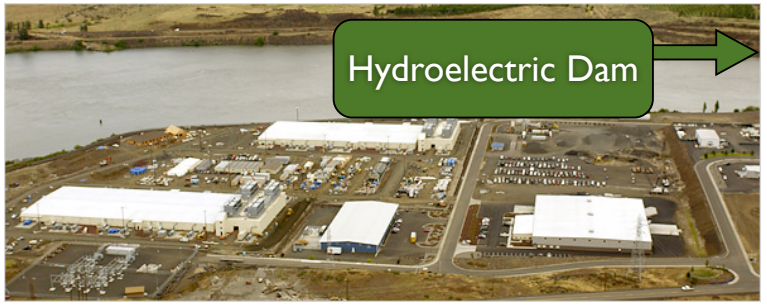
A Fast Array of Wimpy Nodes

David Andersen, Jason Franklin, Michael Kaminsky*,
Amar Phanishayee, Lawrence Tan, **Vijay Vasudevan**

Carnegie Mellon University *Intel Labs Pittsburgh

Energy in computing

- Power is a significant burden on computing
  - 3-year TCO soon to be dominated by power



Can we reduce energy use by a factor of ten?

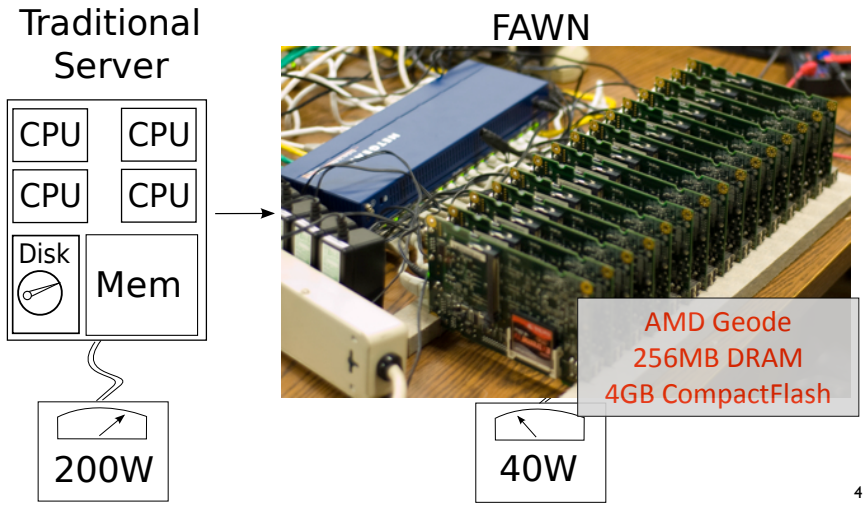
Still serve the same workloads

Avoid increasing capital cost

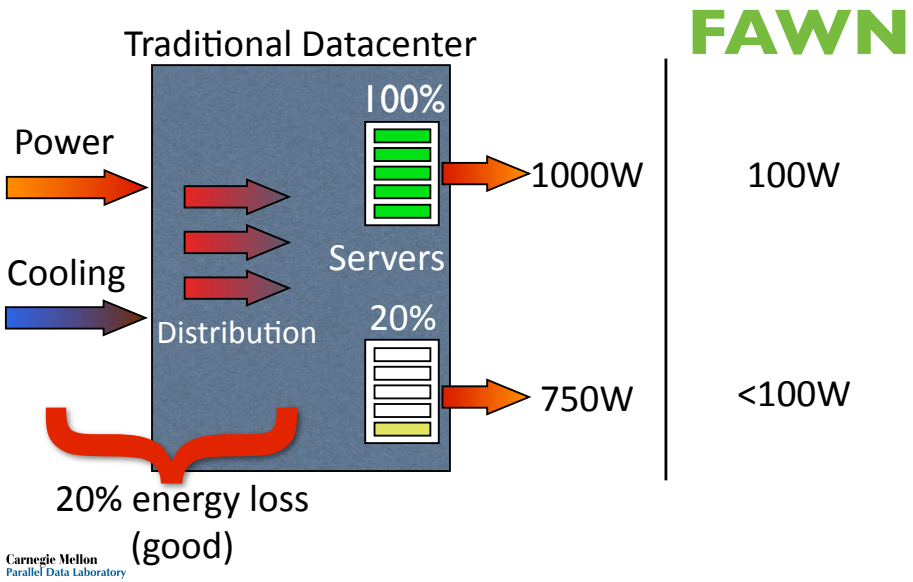
FAWN

Fast Array of Wimpy Nodes

Improve computational efficiency of data-intensive computing using an array of well-balanced low-power systems.



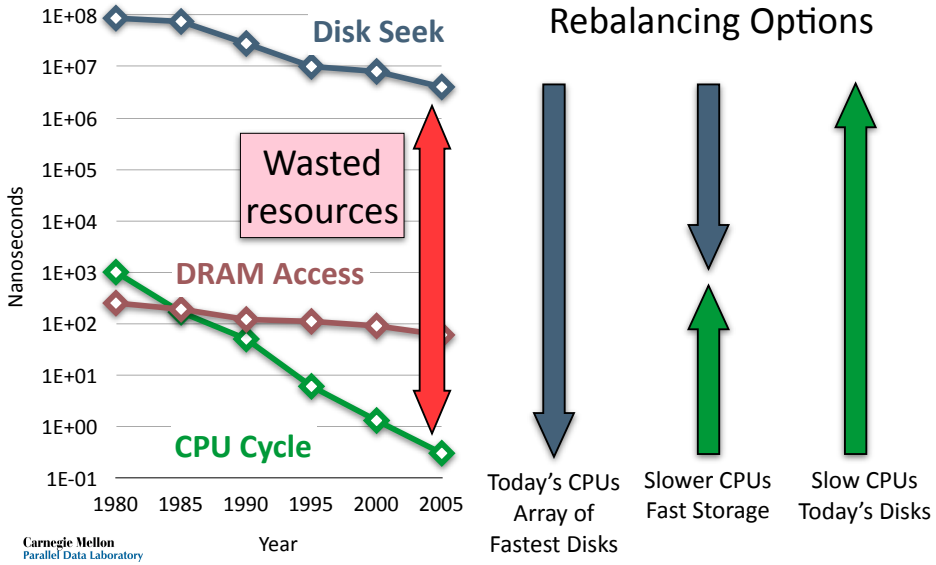
Goal: reduce peak power



Overview

- Background
- **FAWN Principles**
- FAWN-KV Design
- Evaluation
- More Workloads
- Conclusion

Towards balanced systems



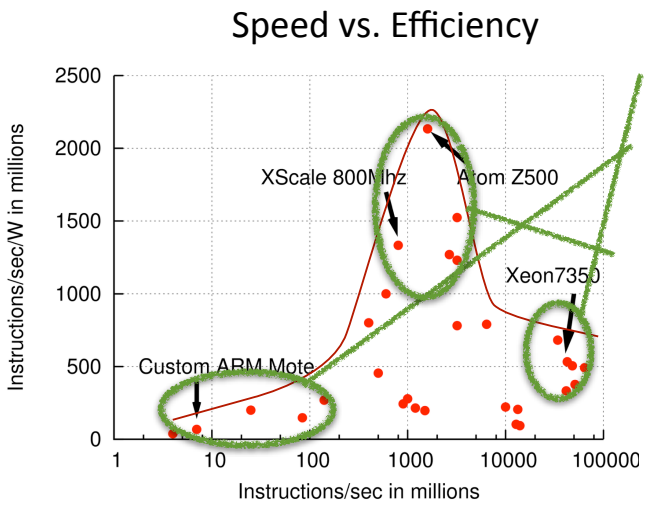
Rebalancing Options

Today's CPUs
Array of
Fastest Disks

Slower CPUs
Fast Storage

Slow CPUs
Today's Disks

Targeting the sweet-spot in efficiency



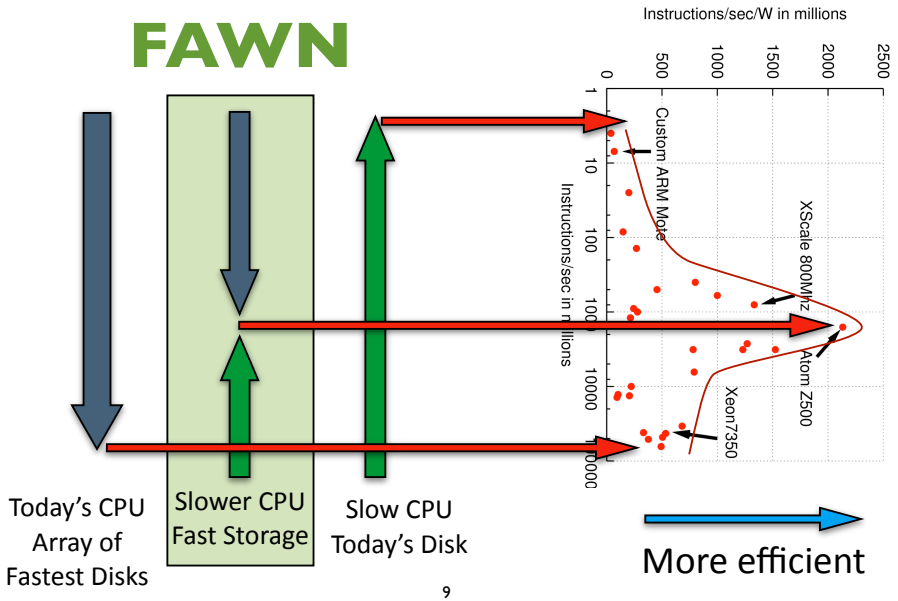
Fastest processors exhibit superlinear power usage

Fixed power costs can dominate efficiency for slow processors

FAWN targets sweet spot in system efficiency when including fixed costs

(Includes 0.1W power overhead)

Targeting the sweet-spot in efficiency

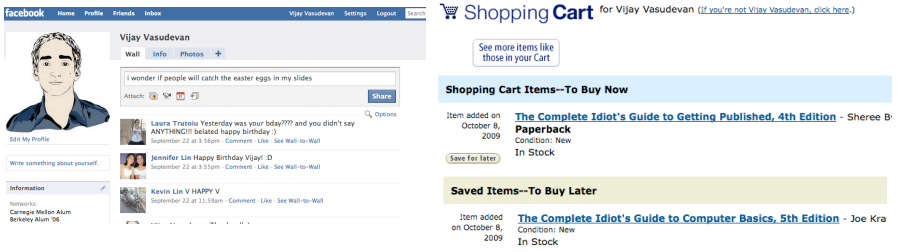


Overview

- Background
- FAWN Principles
- **FAWN-KV Design**
  - Architecture
  - Constraints
- Evaluation
- More Workloads
- Conclusion

Data-intensive key value

- Critical infrastructure service
- Service level agreements for performance/latency
- Random-access, read-mostly, hard to cache



FAWN-KV: Key Value System

- Energy-efficient cluster key-value store
  - Goal: improve **Queries/Joule**
- Prototype: Alix3c2 nodes with flash storage
  - 500MHz CPU, 256MB DRAM, 4GB CompactFlash

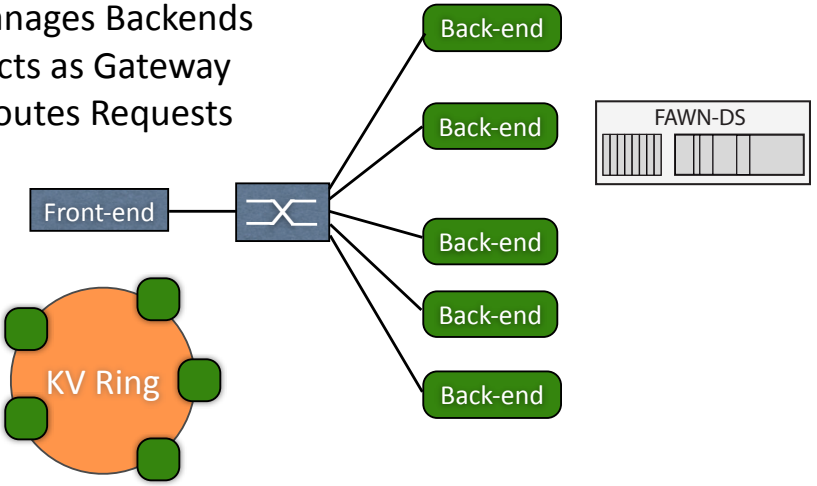
FAWN-KV: Key Value System

Unique Challenges:

- **Efficient and fast failover**
- **Wimpy CPUs, limited DRAM**
- **Flash poor at small random writes**
- 500MHz CPU, 256MB DRAM, 4GB CompactFlash

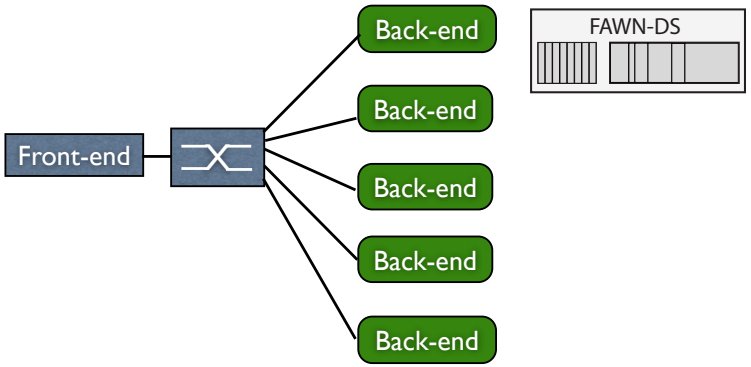
FAWN-KV architecture

Manages Backends
Acts as Gateway
Routes Requests



Consistent hashing

FAWN-KV architecture

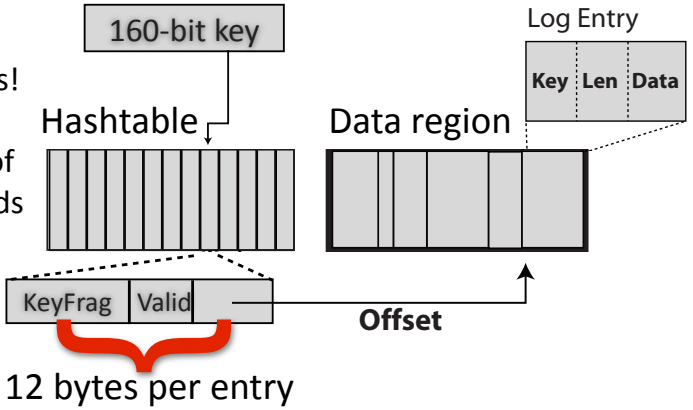


FAWN-DS		FAWN-KV	
Limited Resources	☐	Efficient Failover	☐
Avoid random writes	☐	Avoid random writes	☐

From key to value

KeyFrag != Key
Potential collisions!

Low probability of
multiple Flash reads



12 bytes per entry

FAWN-DS		FAWN-KV	
Limited Resources	☑	Efficient Failover	☐
Avoid random writes	☐	Avoid random writes	☐

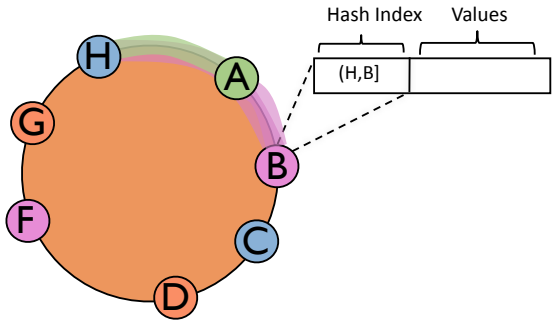
Log-structured datastore

- Log-structuring avoids small random writes

Get → Random Read
Put → Append
Delete → Append

	FAWN-DS	FAWN-KV
Limited Resources	☑	
Efficient Failover		☐
Avoid random writes	☑	☐

On a node addition



Node additions, failures require transfer of key-ranges

Nodes stream data range

Data in original range
Data in new range

Stream from B to A

Concurrent Inserts, Minimizes locking

Compact Datastore

- **Background operations sequential**
- **Continue to meet SLA**

	FAWN-DS	FAWN-KV
Limited Resources	☑	
Efficient Failover		☑
Avoid random writes	☑	☑

FAWN-KV take-aways

- Log-structured datastore
  - Avoids random writes at all levels
  - Minimizes locking during failover
- Careful resource use but high performing
- Replication and strong consistency
  - Variant of chain replication

Overview

- Background
- FAWN principles
- FAWN-KV Design
- **Evaluation**
- More Workloads
- Conclusion

Evaluation roadmap

- Key-value lookup efficiency comparison
- Impact of background operations
  - On performance and latency
- TCO analysis for random read workloads

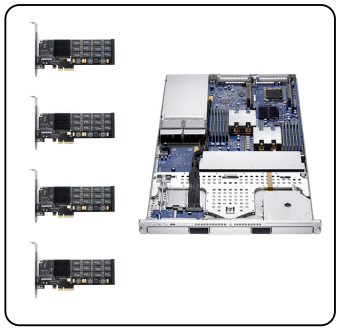
FAWN-DS lookups

System	QPS	Watts	QPS / Watt
Alix3c2/Sandisk(CF)	1298	3.75	346
Desktop/Mobi (SSD)	4289	83	51.7
MacbookPro / HD	66	29	2.3
Desktop / HD	171	87	1.96

- Our FAWN-based system over 6x more efficient than 2008-era traditional systems

Modern system speculation

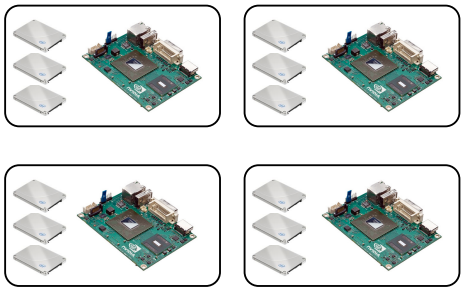
Traditional



420,000 IOPS
250W

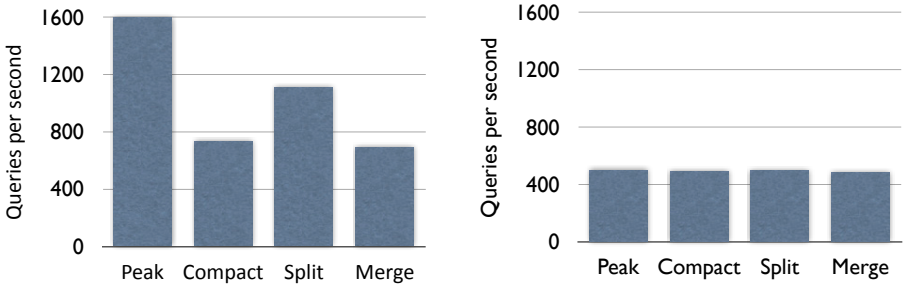
vs.

FAWN



420,000 IOPS
100W

Impact of background ops

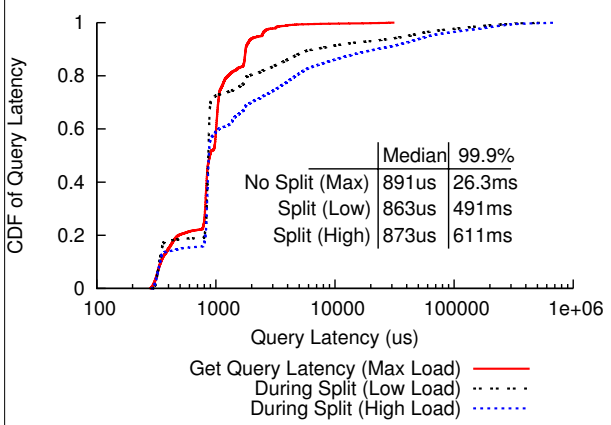


Peak query load

30% of peak query load

Background operations have:
- **Moderate** impact at peak load
- **Negligible** impact at 30% load

Impact on query latency



Low median latency (includes network)

High 99.9% latency during maintenance (but still okay)

When to use FAWN?

TCO = Capital Cost + Power Cost ($0.10/kWh)

Traditional (200W)
Five 2 TB disks
160GB PCI-e Flash SSD
64GB FBDIMM per node

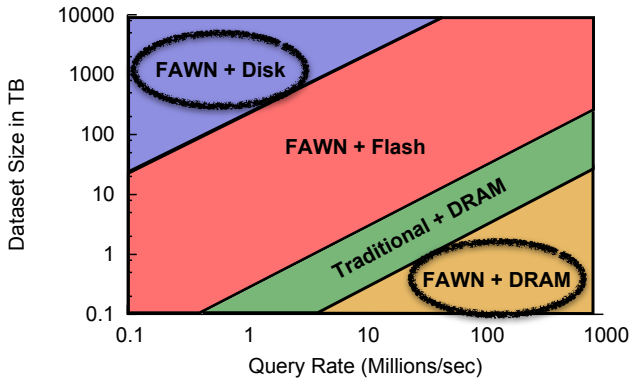
~$2000-8000 per node

FAWN (10W each)
2 TB disk
64GB SATA Flash SSD
2GB DRAM per node

~$250-500 per node

Architecture with lowest TCO

for random access workloads



Ratio of query rate to dataset size determines storage technology

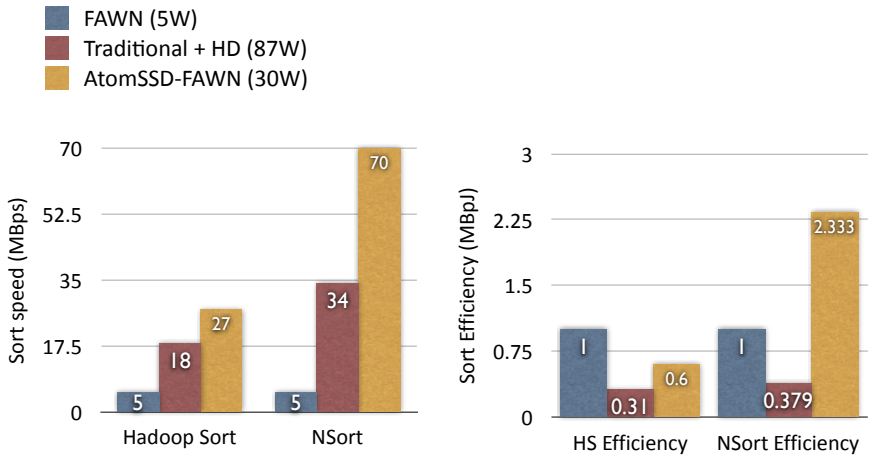
Graph ignores management, cooling, networking...

FAWN-based systems can provide lower cost per {GB, QueryRate}

Overview

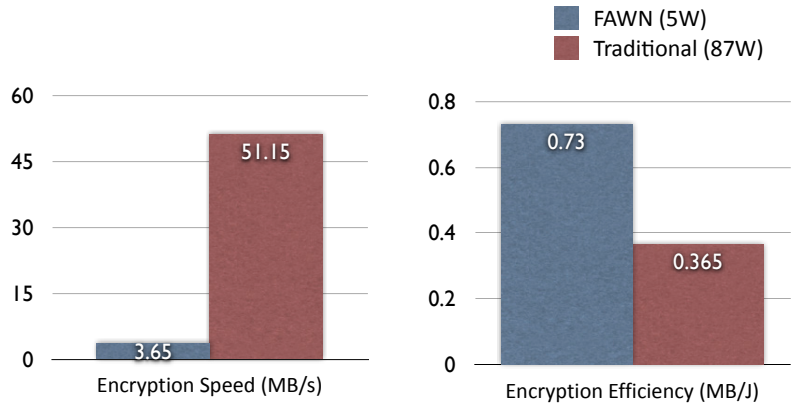
- Background
- FAWN principles
- FAWN-KV Design
- Evaluation
- **More Workloads**
- Conclusion

Sorting small records



CPU-bound encryption

FAWN is 2x more efficient for CPU-bound operations!



Performance Efficiency

Future directions

- We need better metrics!
  - Energy-efficiency vs. {Latency, Cost}
  - Manageability, reliability, programmability ...
- Real workloads on FAWN
  - MapReduce, Web Servers, OLTP?
- FAWN-optimized data processing filters
  - Can we make Sawzall/LINQ/Pig efficient on FAWN?

Conclusion

- FAWN architecture reduces energy consumption of cluster computing
- FAWN-KV addresses challenges of wimpy nodes for key value storage
- Initial results generalize to other workloads
- *"Each decimal order of magnitude increase in parallelism requires a major redesign and rewrite of parallel code" - Kathy Yelick*

What about the network?

adds 1 low power FAWN switch per "pod"