FAWN: A Fast Array of Wimpy Nodes

David G. Andersen, Jason Franklin, Michael Kaminsky*,
Amar Phanishayee, Lawrence Tan, Vijay Vasudevan
Carnegie Mellon University, *Intel Labs SOSP'09

Outline

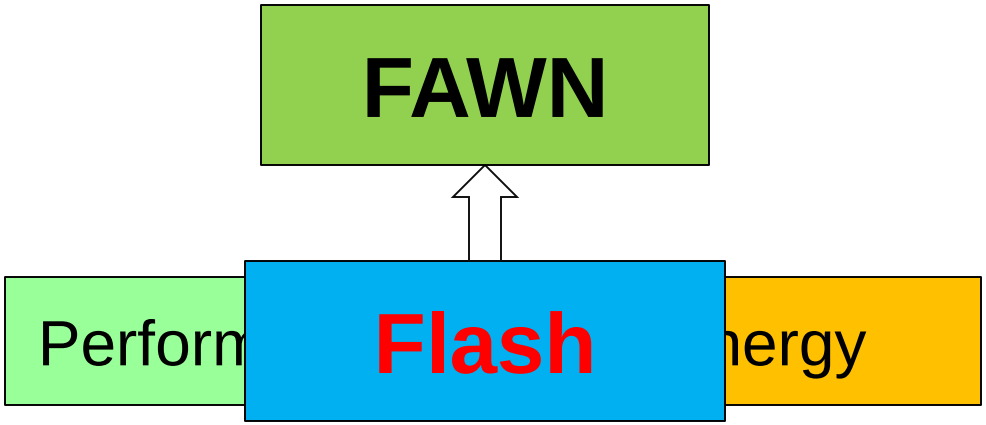
- Introduction
- Problems
- Designs
  - **FAWN-KV**
  - **FAWN-DS**
- Evaluation
- Related Work
- Conclusions
- Acknowledgments

Introduction

- Large-scale data-intensive applications are growing in both size and importance.
- Common characteristics:
  - **I/O intensive, requiring random access over large datasets;**
  - **Massively parallel with thousands of concurrent, mostly-independent operations;**
  - **High load requires large clusters to support;**
  - **The size of objects stored is typically small.**

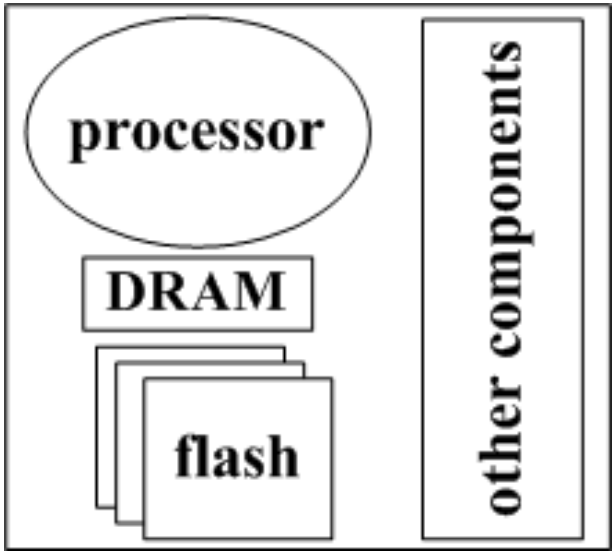
Problems

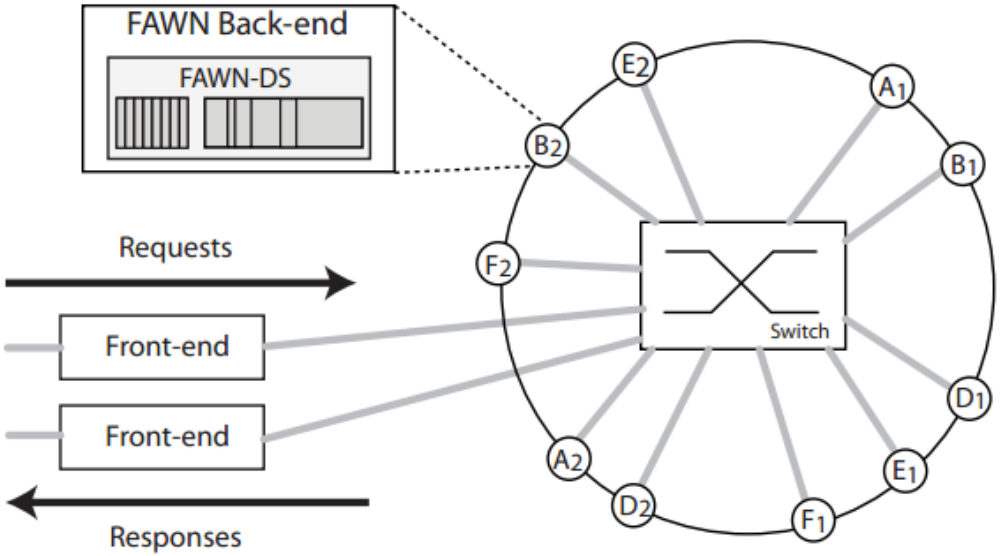
- Small-object random-access workloads are ill-served by conventional disk-based clusters.
- DRAM-based clusters are expensive and consume a surprising amount of power.



What is FAWN?

- FAWN:
  - **Hardware: a specified wimpy node, embedded CPU as the processor and limited DRAM and flash as the storage medium.**
  - **Software: FAWN-KV System, a system that can manage thousands of FAWN nodes efficiently.**





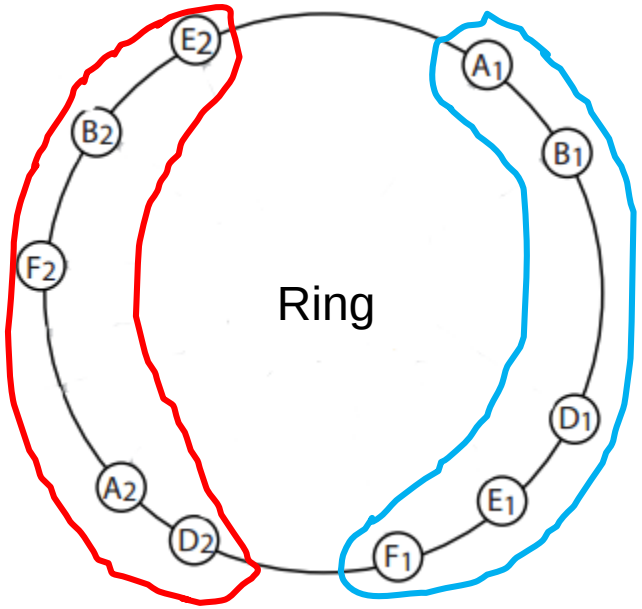
Why FAWN?

- Increasing CPU-I/O Gap
  - **Using wimpy processors selected to reduce I/O-included idle cycles.**
- CPU power consumption grows super-linearly with speed
- Dynamic power scaling on traditional systems is surprisingly inefficient

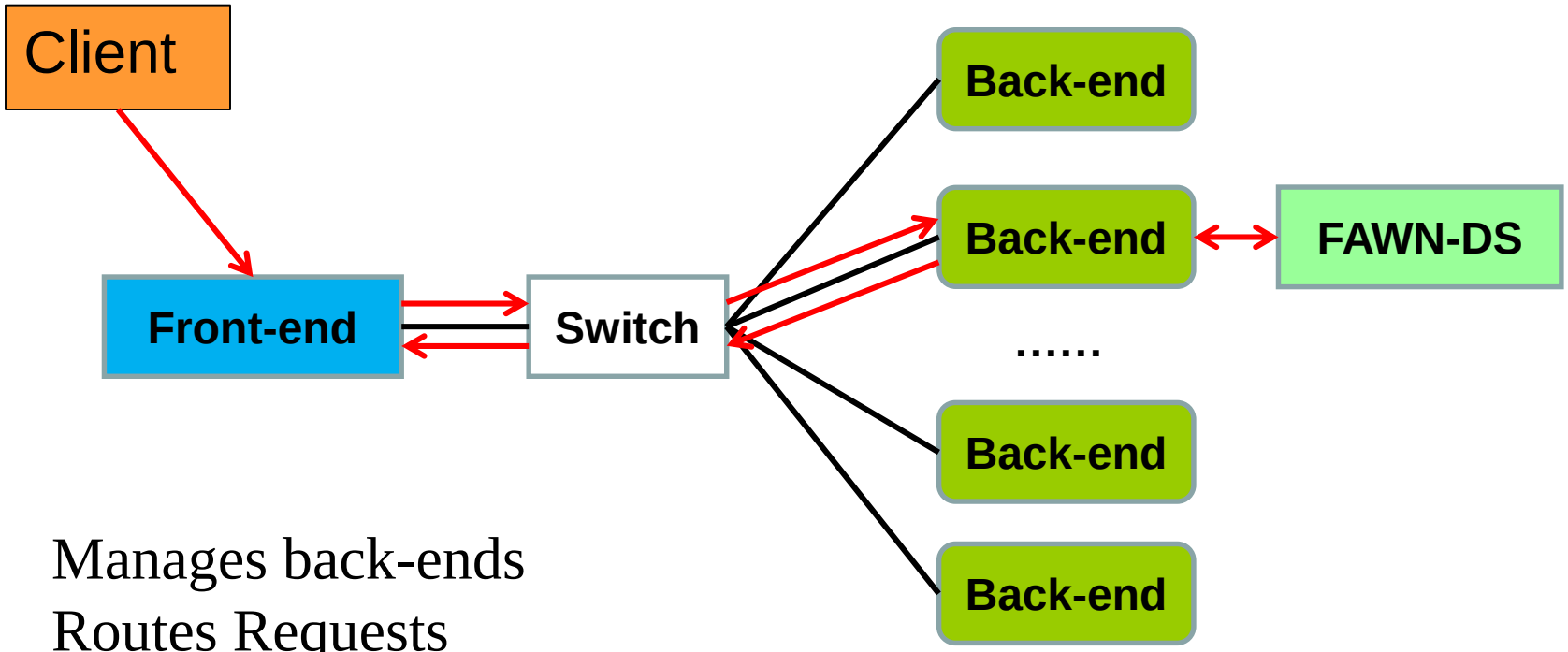
FAWN-KV Architecture-I

- Back-end: responsible for serving particular key.
- Front-end:
  - **Maintain membership list.**
  - **Forward requests to back-end node.**

Front-end:Back-end = 1:n



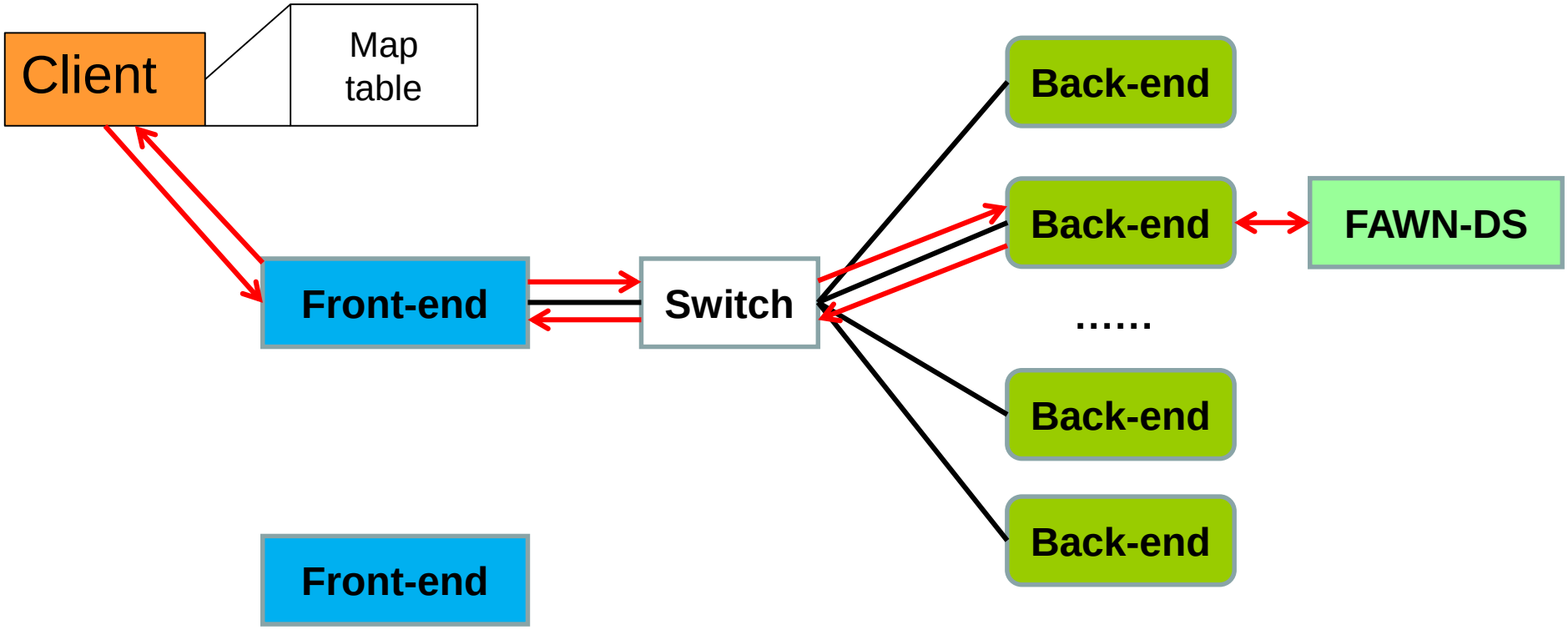
FAWN-KV Architecture-II



Manages back-ends
Routes Requests

If the front-end which the client contacted with was not the back-end belonged to, How to deal this scene?

FAWN-KV Architecture-III

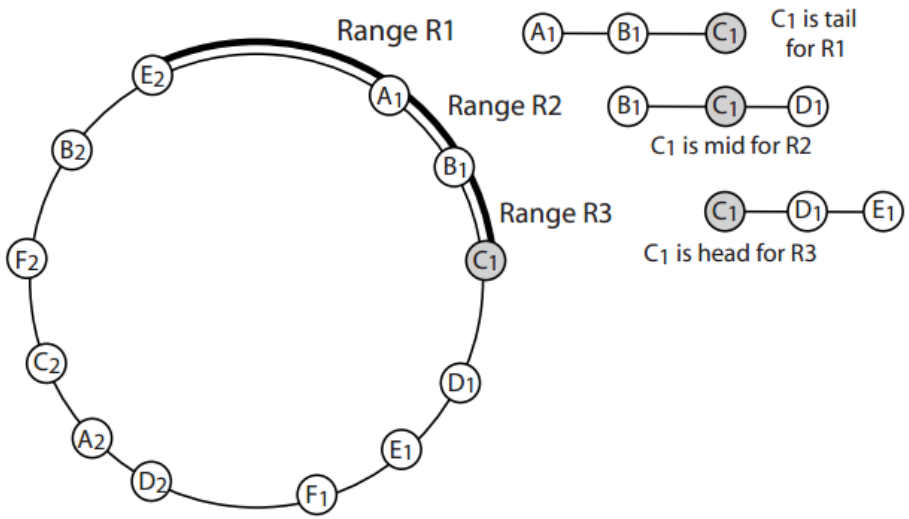


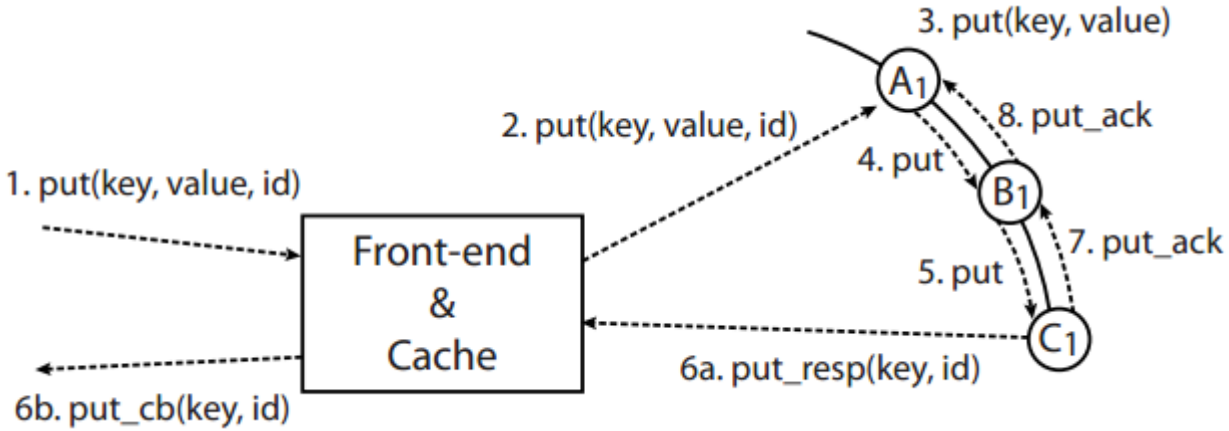
1、client aware of the front-end mapping

2、front-end cache values.

FAWN-KV Architecture-IV

- Replication and Consistency
  - **Chain replication: strong consistency.**



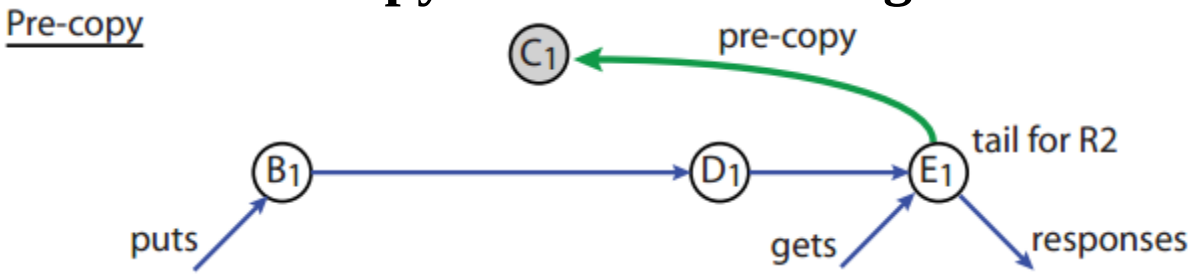


FAWN-KV Architecture-V

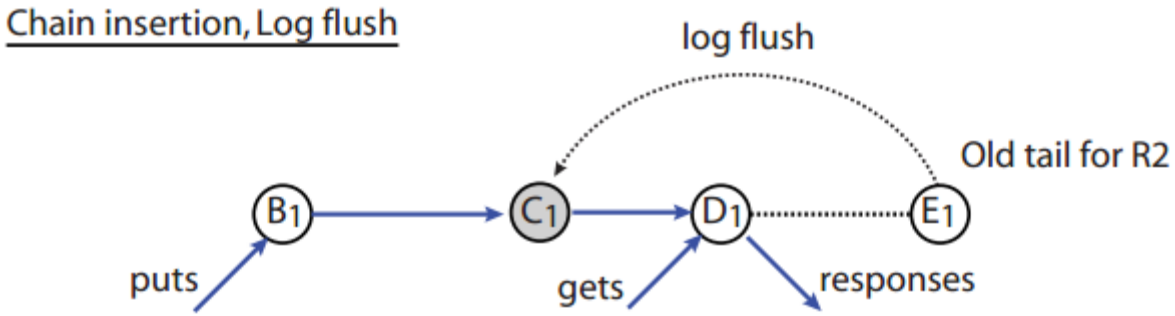
- Joins and Leaves
  - **Joins:**
    - Key range split;
    - Data transmission, new vnode should get a copy of the key range;
    - Update the front-end to valid the new vnode for requests;
    - Free the space of the vnode witch down from the chain.

FAWN-KV Architecture-VI

- Phase 1: Datastore pre-copy
  - **E1 sends C1 a copy of the datstore log file.**

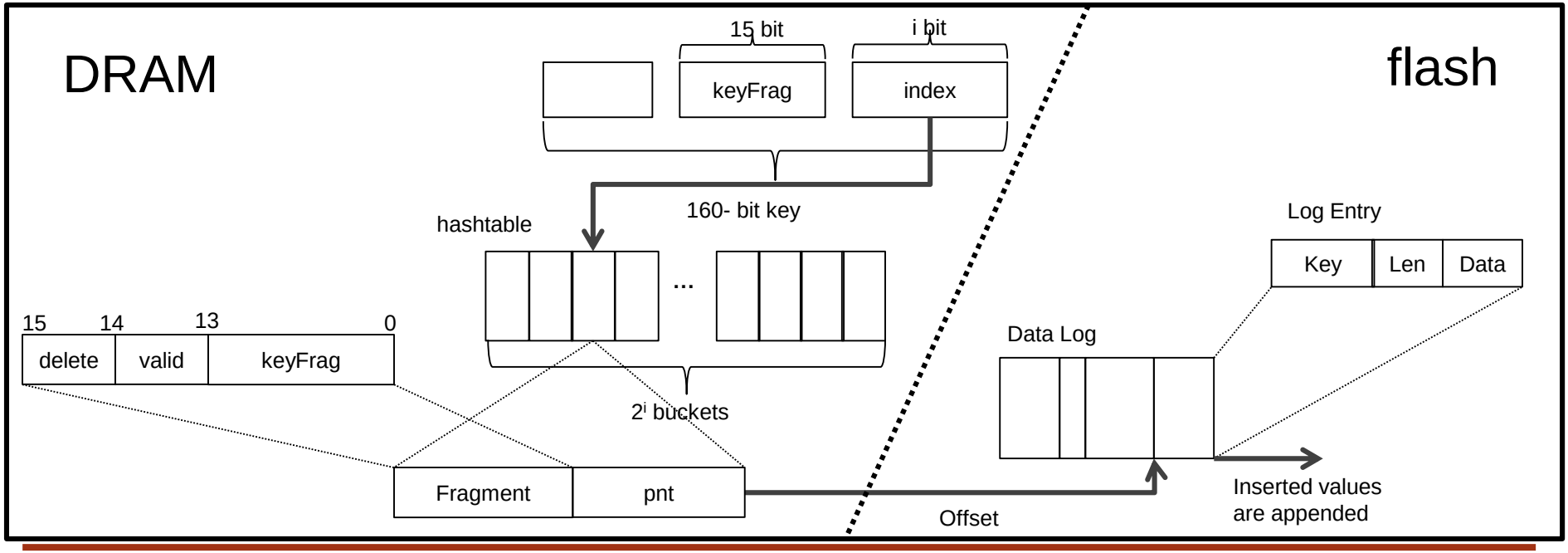


- Phase 2: Chain insertion, log flush and play-forward
  - **Update each node's neighbor state to add C1 to the chain;**
  - **Ensure any in-flight updates sent after the phase 1 completed are flushed to C1.**



FAWN-DS-I

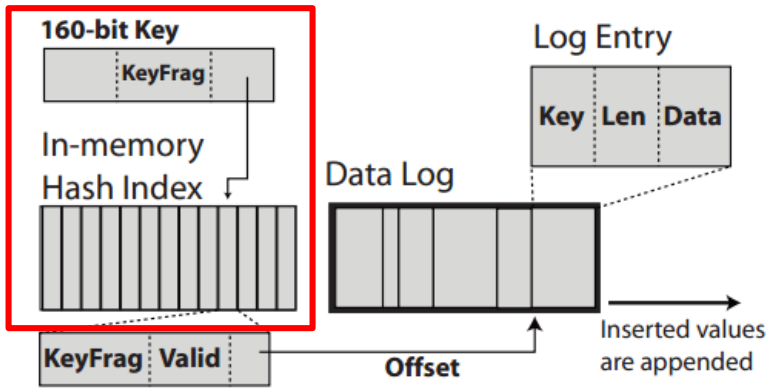
- FAWN-DS
  - **Log-structured key-value store;**
  - **Using a in-DRAM hash table to map keys to an offset in the append-only Data Log on flash.**



FAWN-DS-II

- Back-end Interface:
  - **Get(key, key_len, &data);**
  - **Delete(key, key_len);**
  - **Insert(key, key_len, data, length).**
- Key step of the above:
  - **Find the correct bucket of the key in the Hash index.**

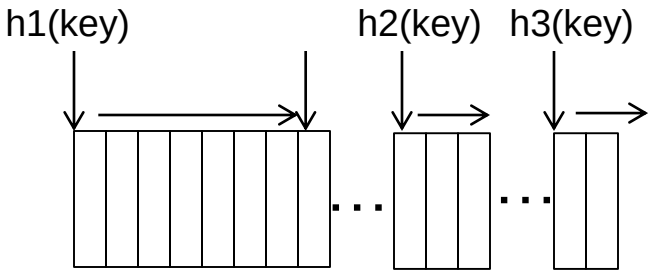
How to map the key to hash index? 2^{160} to 2^{i}?



FAWN-DS-III

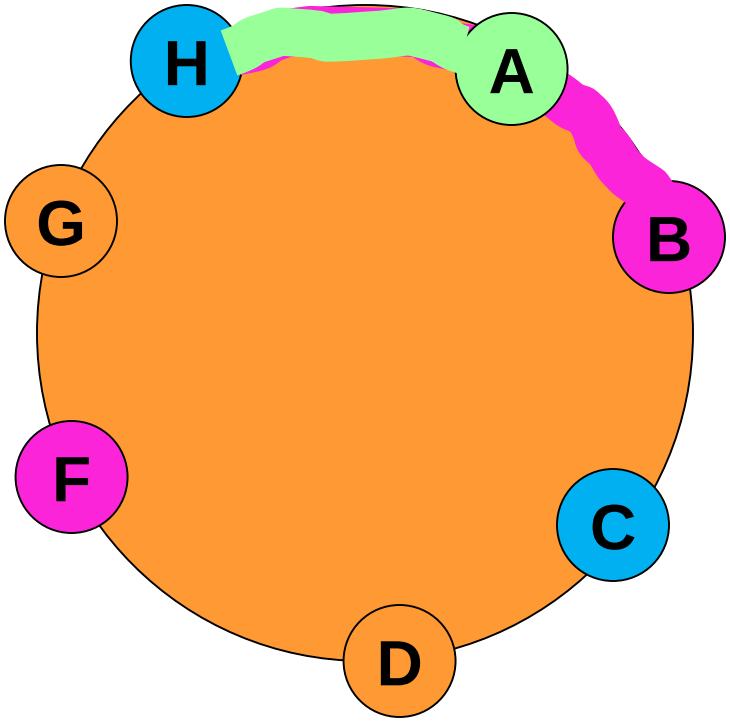
- Conflict chain: depth = 8.
- Different hash functions: three funcs.

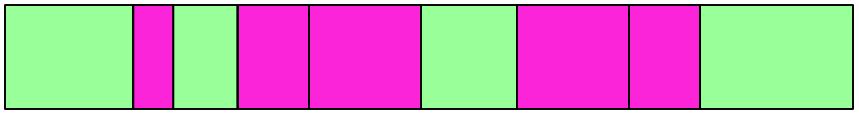
```
/* KEY = 0x93df7317294b99e3e049, 16 index bits */
INDEX = KEY & 0xffff; /* = 0xe049; */
KEYFRAG = (KEY >> 16) & 0x7fff; /* = 0x19e3; */
for i = 0 to NUM_HASHES do
    bucket = hash[i](INDEX);
    if bucket.valid && bucket.keyfrag==KEYFRAG &&
       readKey(bucket.offset)==KEY then
        return bucket;
    end if
    {Check next chain element...}
end for
return NOT_FOUND;
```



FAWN-DS-IV

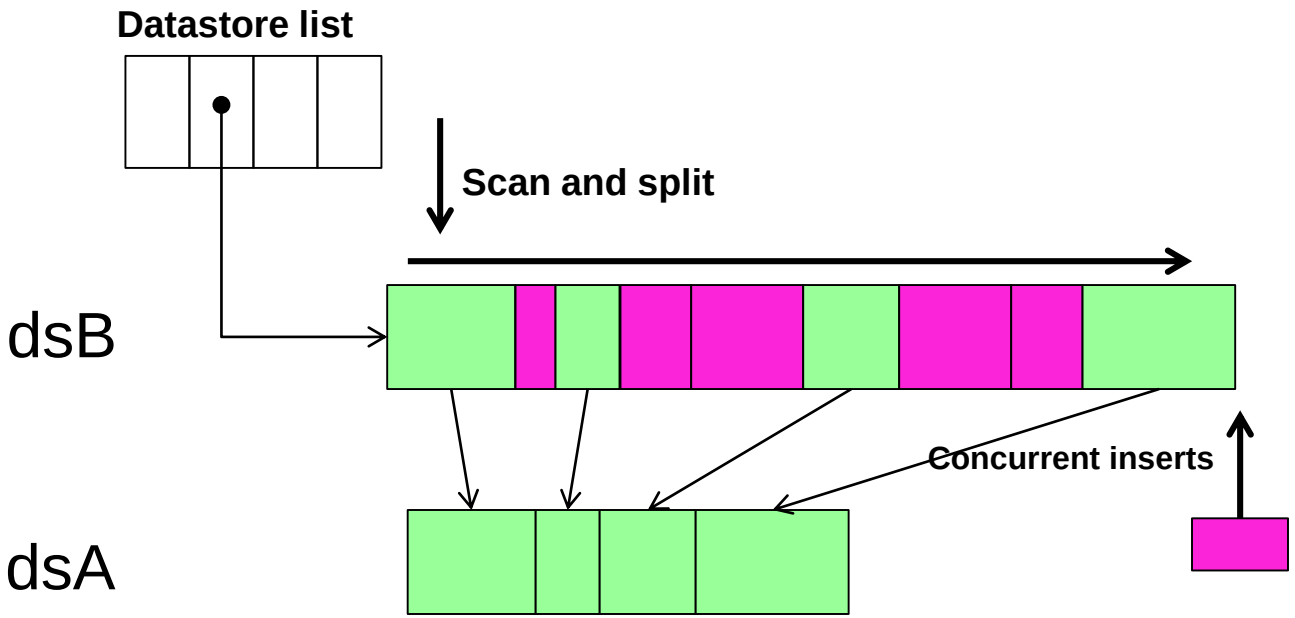
- Maintenance: Split, Merge, Compact
  - **Split: triggered by a node addition.**





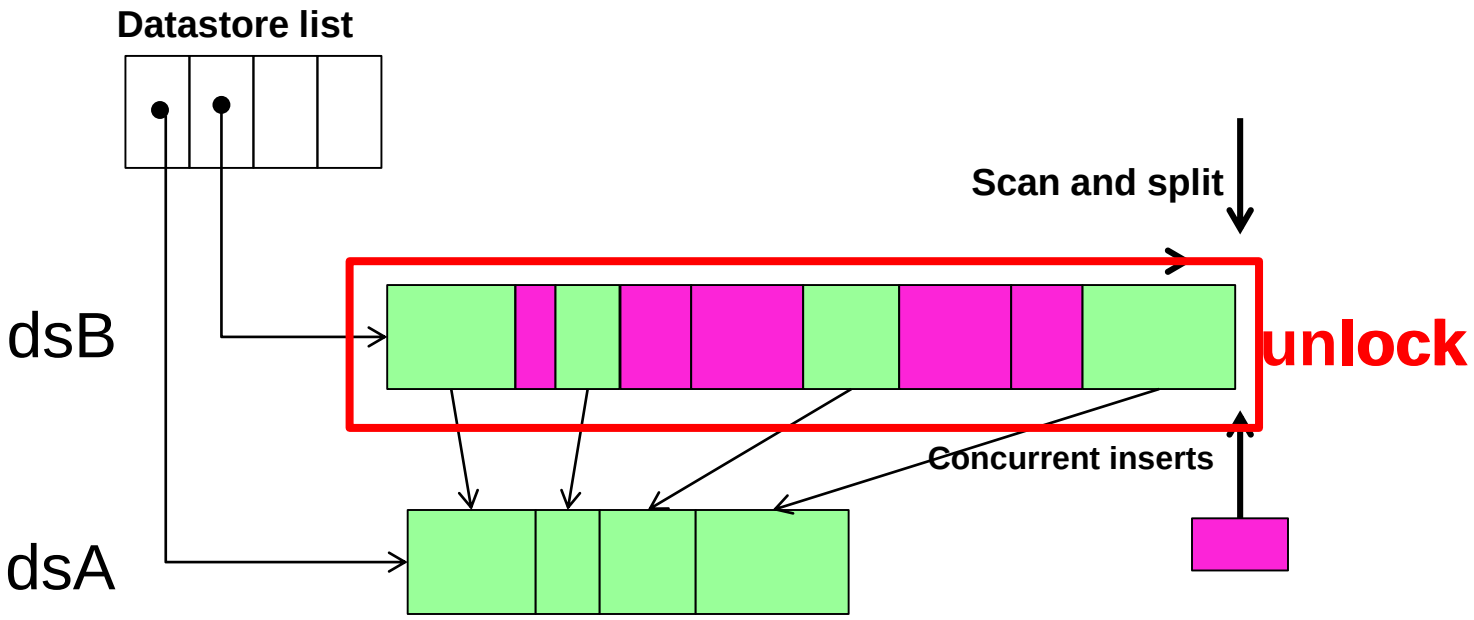
Nodes Stream Data Range-I

- Create new Datastore A(dsA);
- Scan Datastore B(dsB) and transfer the data in rang A to dsA.



Nodes Stream Data Range-II

- Create new Datastore A(dsA);
- Scan Datastore B(dsB) and transfer the data in rang A to dsA.



Evaluation

- Evaluation Items:
  - **K/V lookup efficiency comparison;**
  - **Impact of Ring Membership Changes;**
  - **TCO analysis for random read.**
- Evaluation Hardware:
  - **AMD Geode LX processor, 500MHz;**
  - **256 MB DDR SDRAM, 400MHz;**
  - **100Mbit/s Ethernet;**
  - **4GB Sandisk Extreme IV CF.**

K/V Lookup Efficient Comparison-I

System / Storage	QPS	Watts	Queries / Joule
Embedded Systems			
Alix3c2 / Sandisk(CF)	1298	3.75	346
Soekris / Sandisk(CF)	334	3.75	89
Traditional Systems			
Desktop / Mobi(SSD)	4289	83	51.7
MacbookPro / HD	66	29	2.3
Desktop / HD	171	87	1.96

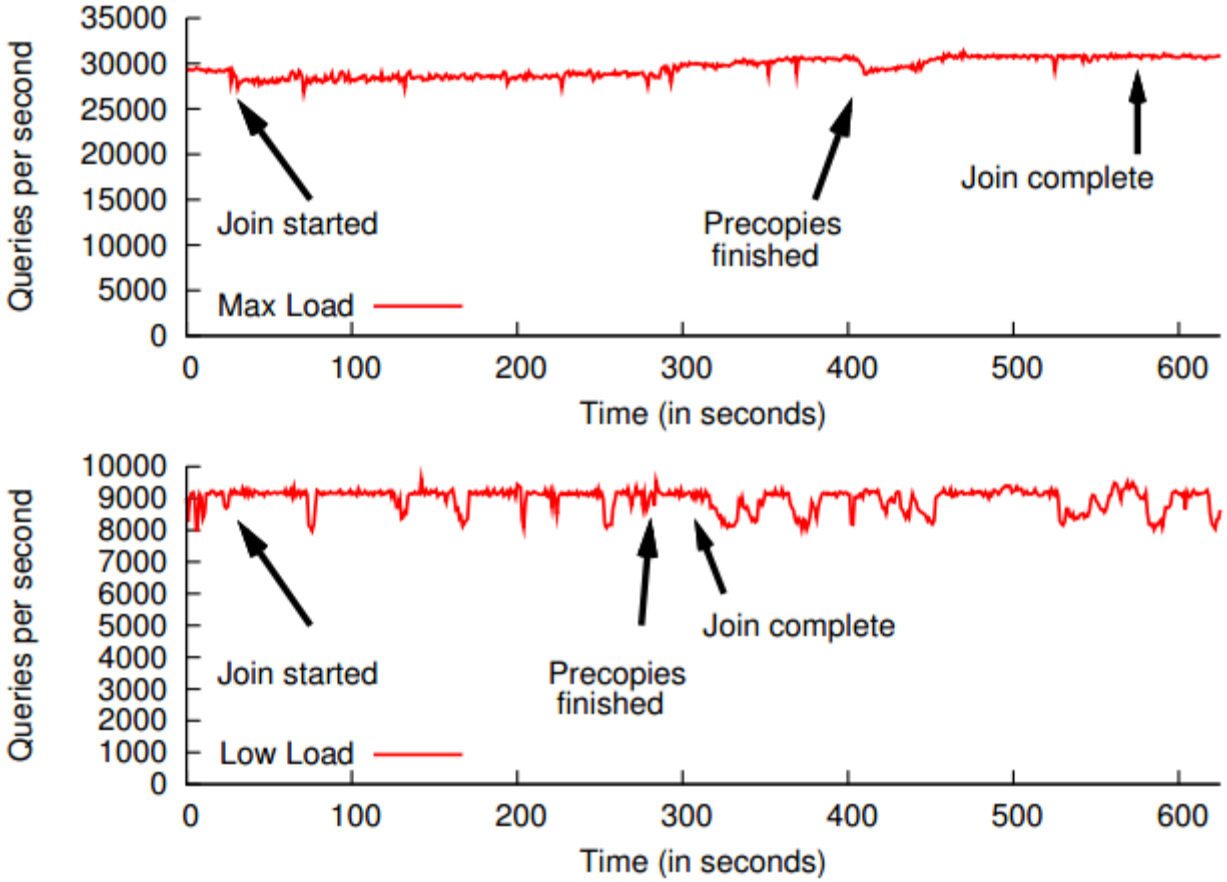
- FAWN-based system over 6x more efficient than the other traditional systems

K/V Lookup Efficient Comparison-II

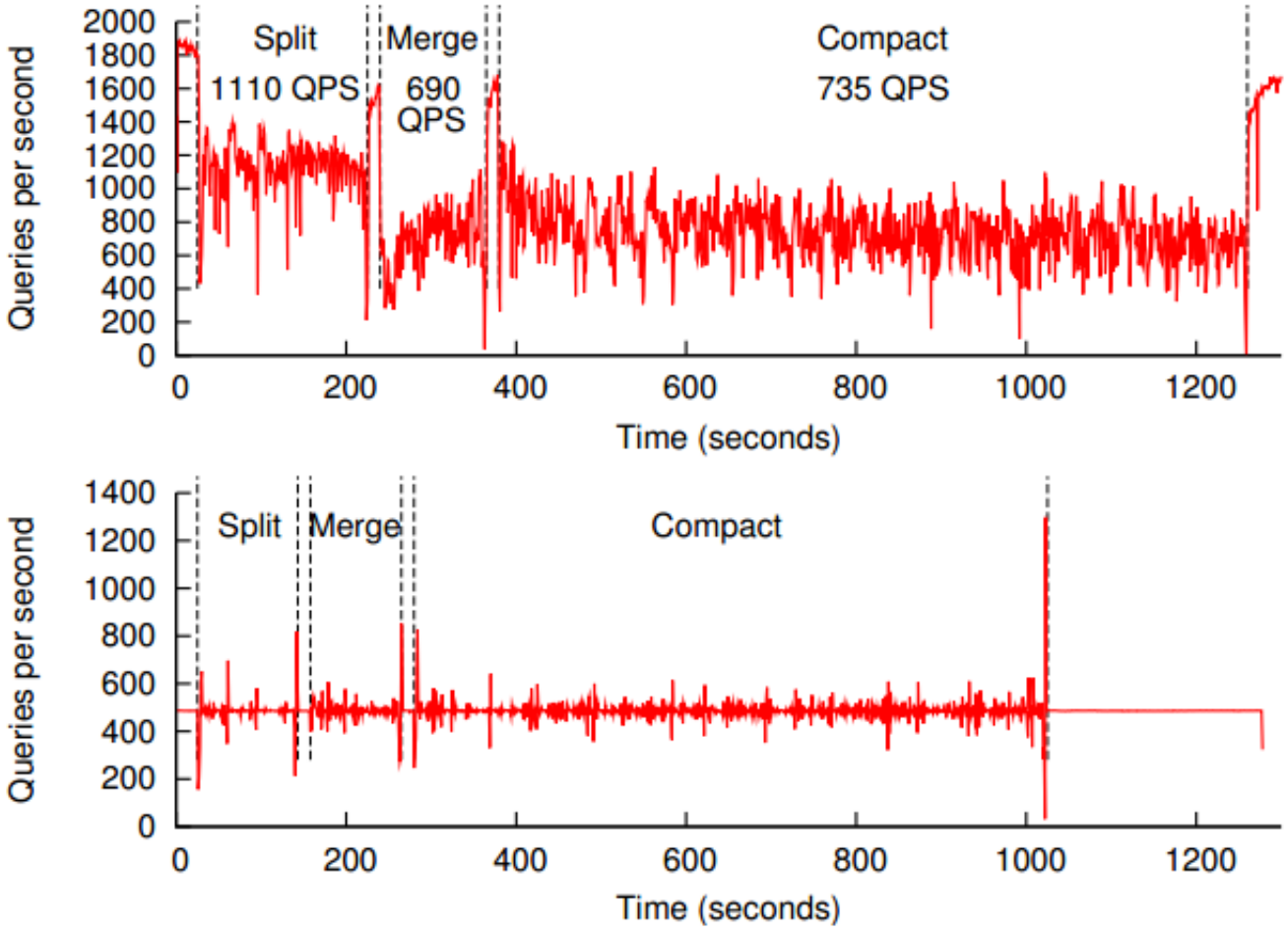
DS Size	*1 KB Rand Read* (in queries/sec)	*256 B Rand Read* (in queries/sec)
10 KB	72352	85012
125 MB	51968	65412
250 MB	6824	5902
500 MB	2016	2449
1 GB	1595	1964
2 GB	1446	1613
3.5 GB	1150	1298

Table 2: Local random read performance of FAWN-DS.

Impact of Ring Membership Changes-I



Impact of Ring Membership Changes-II



TCO Analysis for Random Read-I

- TCO = Capital Cost + Power Cost ($0.1/kWh)

System	Cost	W	QPS	Queries / Joule	GB / Watt	TCO / GB	TCO / QPS
Traditionals:							
5-2TB HD	$2K	250	1500	6	40	0.26	1.77
160GB PCIe SSD	$8K	220	200K	909	0.72	53	0.04
64GB DRAM	$3K	280	1M	3.5K	0.23	59	0.004
FAWNs:							
2TB Disk	$350	20	250	12.5	100	0.20	1.61
32GB SSD	$500	15	35K	2.3K	2.1	16.9	0.015
2GB DRAM	$250	15	100K	6.6K	0.13	134	0.003

TCO Analysis for Random Read-II

- How many nodes are required for a cluster?

$$N = max\left(\frac{DS}{\frac{gb}{node}}, \frac{QR}{\frac{qr}{node}}\right)$$

TCO Analysis for Random Read-III

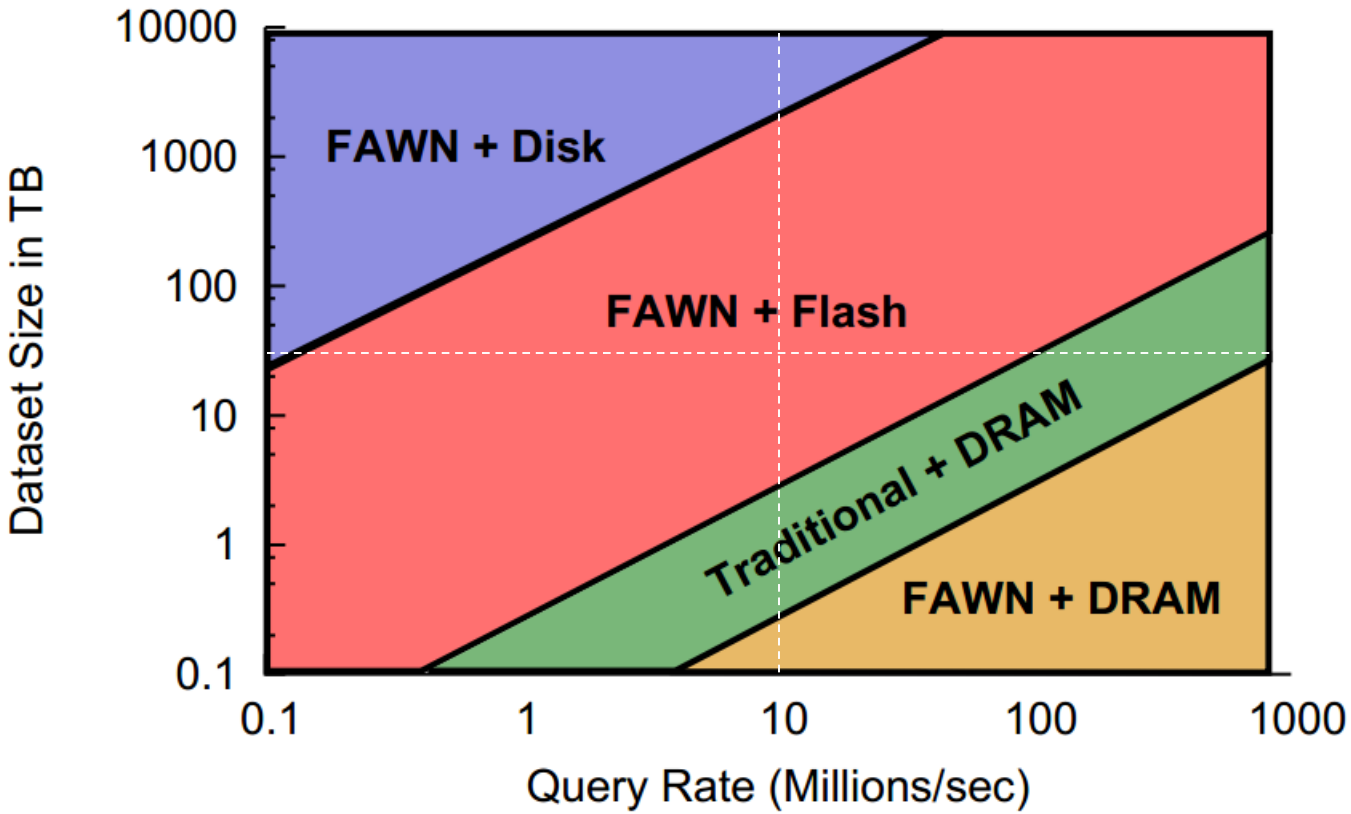


Figure 16: Solution space for lowest 3-year TCO as a function of dataset size and query rate.

Related Work

- Hardware architecture:
  - **Pairing an array of flash chips and DRAM with low-power CPUs for low-power data intensive computing.**
- File systems for Flash:
  - **Several file systems, such as JFFS2, are specialized for use on flash.**
- High-throughput Storage and Analysis:
  - **Some systems like Hadoop, provide bulk throughput for massive datasets with low selectivity.**

Conclusions

- FAWN architecture reduce energy consumption of cluster computing.
- FAWN-KV address the challenges of wimpy nodes for a key-value store:
  - **Log-structured , memory efficient datastore;**
  - **Efficient replication;**
  - **Meets the energy efficiency and performance goals.**

Acknowledgment

- Article Understanding：
  - **Prof. Xiong**
  - **Fengfeng Pan**
  - **Zigang Zhang**
- PPT Production：
  - **Fengfeng Pan**
  - **Biao Ma**

Thank You!