Learning to Scale Out
By Scaling Down

The FAWN Project


Carnegie Mellon University  *Intel Labs Pittsburgh
** Princeton University  *** Georgia Tech
Infrastructure: PUE
2005: 2–3
2012: ~1.1

Leave it to industry
Infrastructure: PUE
2005: 2–3
2012: ~1.1
*Leave it to industry*

Proportionality

1000W

750W

200W
Infrastructure: PUE
2005: 2–3
2012: ~1.1
Leave it to industry

Proportionality

Servers
100%

1000W

FAWNs
100%

Efficiency

300W

750W

200W
Infrastructure: PUE
2005: 2–3
2012: ~1.1
Leave it to industry

Proportionality
1000W
750W
200W

Servers
100%
20%

FAWNs
100%

Efficiency
300W
Gigahertz is not free

Speed and power calculated from specification sheets
Power includes “system overhead” (e.g., Ethernet)
The Memory Wall

- Disk Seek
- DRAM Access
- CPU Cycle

Year:
- 1980
- 1985
- 1990
- 1995
- 2000
- 2005
The Memory Wall

- Disk Seek
- DRAM Access
- CPU Cycle

Bridge gap: Caching, speculation, etc.

Year:
- 1980
- 1985
- 1990
- 1995
- 2000
- 2005
The Memory Wall

Drum Seek

Bridge gap: Caching, speculation, etc.

Disk Seek

DRAM Access

CPU Cycle

Year


Principles
Key-Value Systems
FAWN-KV Design
Evaluation

Transistors

Have the soul of a capacitor
The Memory Wall

Transistors

Have the soul of a capacitor

Charge Here

Moves charge carriers here

Which lets current flow

The bridge gap: Caching, speculation, etc.

Disk Seek

DRAM Access

CPU Cycle

Year


Principles

Key-Value Systems

FAWN-KV Design

Evaluation
Gigahertz hurts

Remember:
Memory capacity costs you
“Wimpy” Nodes

1.6 GHz Dual-core Atom
32-160 GB Flash SSD
Only 1 GB DRAM!
“Each decimal order of magnitude increase in parallelism requires a major redesign and rewrite of parallel code” - Kathy Yelick
The FAWN Quad of Pain

- Load Balancing
- Parallelization
- Hardware Specificity
- Memory Capacity
It’s not just masochism

Moore

Dennard

(Figures from Danowitz, Kelley, Mao, Stevenson, and Horowitz: CPU DB)

All systems will face this challenge over time
FAWN:
It started with a key-value store
Key-value storage systems

• Critical infrastructure service
• Performance-conscious
• Random-access, read-mostly, hard to cache
Small record, random access

<table>
<thead>
<tr>
<th>99 friends</th>
<th>See All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carsten Varming</td>
<td></td>
</tr>
<tr>
<td>Timor Tsentsiper</td>
<td></td>
</tr>
<tr>
<td>Arvind Chari</td>
<td></td>
</tr>
<tr>
<td>Corey Ilycan</td>
<td></td>
</tr>
<tr>
<td>John Bethencourt</td>
<td></td>
</tr>
<tr>
<td>Ram Ravichandran</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sep 21</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dan Wendlandt wrote</strong> at 6:47pm</td>
</tr>
<tr>
<td>have a good one man. hope the facebook TG was fun, the email was hilarious</td>
</tr>
<tr>
<td>Wall-to-Wall – Write on Dan's Wall</td>
</tr>
</tbody>
</table>

| **Patrick Gage Kelley wrote** at 2:42pm |
| Oh! birthday! |
| Wall-to-Wall – Write on Patrick's Wall |

| **Jagan Seshadri wrote** at 1:50pm |
| Happy birthday Vij! 24 and there's so much more... |
| Wall-to-Wall – Write on Jagan's Wall |

<table>
<thead>
<tr>
<th>Sep 19</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vish Subramanian wrote</strong> at 3:48am</td>
</tr>
<tr>
<td>happy birthday dude, its been awhile!</td>
</tr>
<tr>
<td>Wall-to-Wall – Write on Vish's Wall</td>
</tr>
</tbody>
</table>

| **Bobby Gregg wrote** at 2:22pm |
| hi vijay! i'm super early but i'm bad about checking facebook regularly nowadays so i wanted to say happy birthday. let's catch up about our respective grad school woes. |
| Wall-to-Wall – Write on Bobby's Wall |
Small record, random access

Select name, photo from users where uid=513542;
Small record, random access

Select name, photo from users where uid=818503;
Small record, random access

Select name, photo from users where uid=468883;
Small record, random access

Select name, photo from users where uid=124111;

Select wallpost from posts where pid=13821828188;
Small record, random access

Select wallpost from posts where pid=89888333522;
Select name, photo from users where uid=474488;
Select name, photo from users where uid=124566;
Select name, photo from users where uid=124111;
Select name, photo from users where uid=12223;
Select wallpost from posts where pid=13821828188;
Select wallpost from posts where pid=12314144887;
Select wallpost from posts where pid=738838402;
Select wallpost from posts where pid=097788;
Select wallpost from posts where pid=357845;
FAWN-DS and -KV: Key-value Storage System

Goal: improve Queries/Joule

500MHz CPU
256MB DRAM
4GB CompactFlash
FAWN-DS and -KV: Key-value Storage System

Goal: improve Queries/Joule

Unique Challenges:

- Wimpy CPUs, limited DRAM
- Flash poor at small random writes
- Sustain performance during membership changes

Principles | Key-Value Systems | FAWN-KV Design | Evaluation
Avoiding random writes
Avoiding random writes

In DRAM
Hashtable

In Flash
Data region
Avoiding random writes

In DRAM
Hashtable

In Flash
Data region

Put \( K, V \)
Avoiding random writes

In DRAM
Hashtable

In Flash
Data region

Put \( K_i, V \)
Avoiding random writes

In DRAM

Hashtable

In Flash

Data region

K₁, V

Put
Avoiding random writes

In DRAM
Hashtable

In Flash
Data region

Put
Avoiding random writes

In DRAM
Hashtable

In Flash
Data region

Put

All writes to Flash are sequential
Research Example

• Developed DRAM-efficient system to find location on flash
  • (“Partial-key hashing”) 2008-9
• We’ve continued this since then:
  • Partial-key cuckoo hashing 2011
  • Optimistic concurrent cuckoo hashing 2012
Evaluation Takeaways

- 2008: FAWN-based system 6x more efficient than traditional systems
- Partial-key hashing enabled memory-efficient DRAM index for flash-resident data
- Can create high-performance, predictable storage service for small key-value pairs
And then we moved to Atom + SSD

<table>
<thead>
<tr>
<th></th>
<th>6x</th>
<th>8x</th>
<th>30-60x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geode</td>
<td>500Mhz</td>
<td>256MB</td>
<td>4GB CF Card</td>
</tr>
<tr>
<td>Atom</td>
<td>1.6 Ghz</td>
<td>2GB</td>
<td>~2k IOPS</td>
</tr>
<tr>
<td>1 single-core</td>
<td></td>
<td></td>
<td>~60k IOPS</td>
</tr>
<tr>
<td>120GB SSD</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
backend store
hyper-optimized
for low DRAM
and large flash
Systems begat algorithms:

“Practical Batch-Updatable External Hashing with Sorting”

H. Lim et al., ALENEX 2012

(Recently heard that Bing uses several state-of-the-art, memory-efficient indexes)
And now... Load imbalance

• Distributed key-value system
And now... Load imbalance

• Distributed key-value system
And now... Load imbalance

- Distributed key-value system
And now... Load imbalance

- Distributed key-value system
And now... Load imbalance

• Distributed key-value system

1. get(key)
And now... Load imbalance

• Distributed key-value system

1. get(key)
2. BackendID=hash(key)
And now... Load imbalance

- Distributed key-value system

1. get(key)
2. BackendID=hash(key)
3. val=lookup(key)
And now... Load imbalance

- Distributed key-value system

1. `get(key)`
2. `BackendID=hash(key)`
3. `val=lookup(key)`
4. `return val`
And now... Load imbalance

• Distributed key-value system

1. `get(key)`
2. `BackendID=hash(key)`
3. `val=lookup(key)`
4. `return val`
And now... Load imbalance

• Distributed key-value system

1. get(key)
2. BackendID=hash(key)
3. val=lookup(key)
4. return val

SLA: 850,000 queries/sec

Backend1

Backend2

10,000 queries/sec

Atom CPU

SSD

Back.. 85

Back.. 88
Measured tput on FAWN testbed

Overall throughput (KQPS)

- uniform
- Zipf (1.01)
- adversarial

n: number of nodes
Queries

FrontEnd

Backend1

Backend2

... 

Backend8

Overall throughput (KQPS)

uniform
Zipf (1.01)
adversarial

n: number of nodes
How many items to cache?

Overall throughput (KQPS)

- uniform
- Zipf (1.01)
- adversarial

Queries

FrontEnd

cache

Backend1

Backend2

... 

Backend8

Backend8
small/fast cache is enough!

We prove that, for \( n \) nodes
- Only need to cache \( O(n \log n) \) most popular entries
- With 100 backend nodes, need only about 4,000 items in the cache. Tiny!
Worst case? Now best case
Thus...
"Brawny" server

O(N log N)  
[“small cache” socc 2011]

Multi-reader parallel cuckoo hashing  
[“MemC3” - NSDI 2013]

Insanely Fast Cache

“Wimpy” servers

"Wimpy" servers  
[FAWN, SOSP 2009]

SILT
SILT
SILT  
[SILT, SOSP 2011]

Entropy-coded tries  
[SOSP + ALENEX]

Partial-key cuckoo hashing

Cuckoo filter
highly parallel, lower-GHz, (memory-constrained?):

Architectures, algorithms, and programming