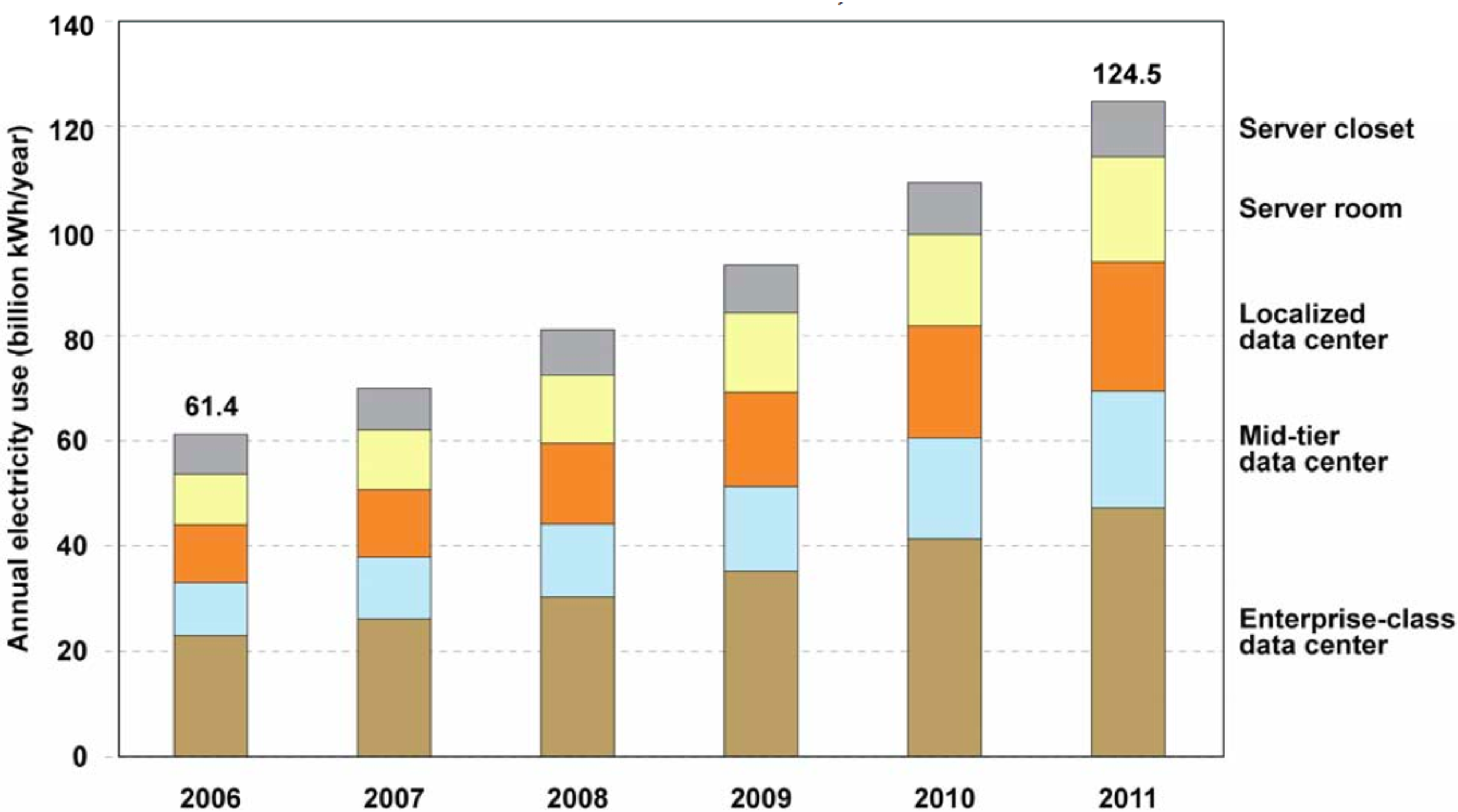
FAWN
A Fast Array of Wimpy Nodes

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

22nd ACM Symposium on Operating Systems Principles – October 2009

Projected Electricity Usage



Source: Report to Congress on Server and Data Center Energy Efficiency – August 2007

Distributed Key Value Store

Dynamo — amazon.com

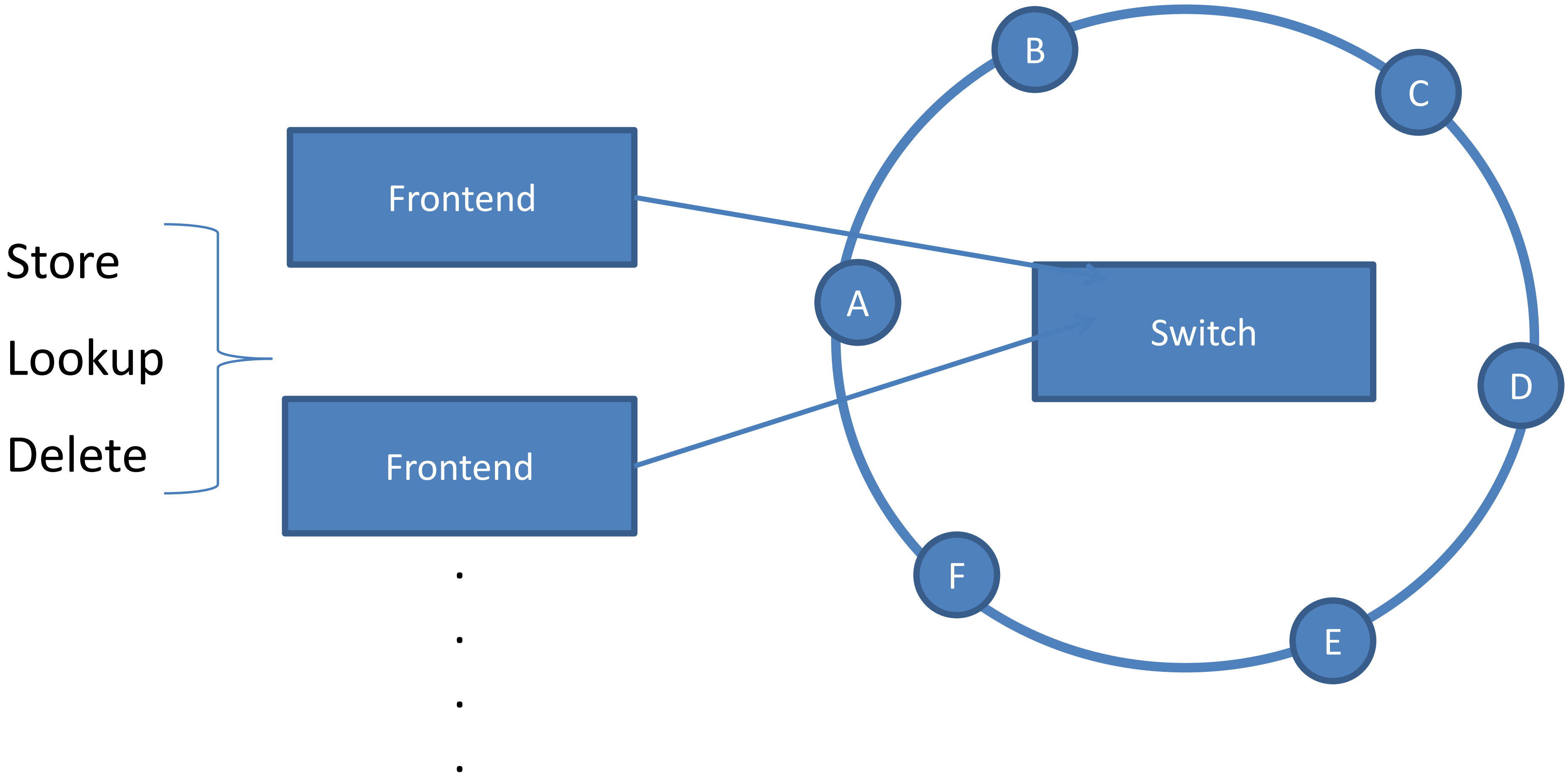
Cassandra — facebook

Voldemort — LinkedIn

200 Watts

200 Watts → 10 Watts

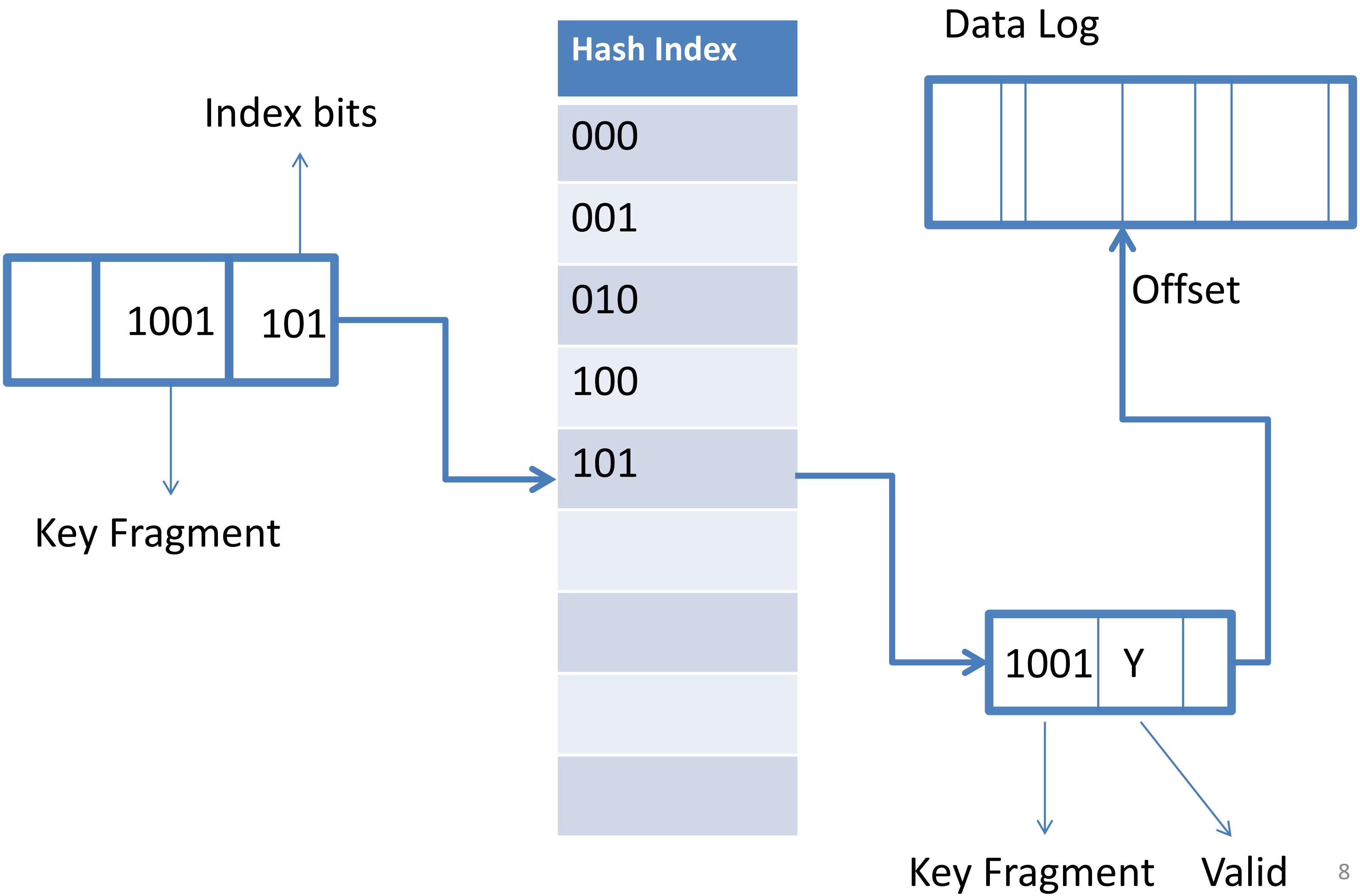
FAWN-KV store



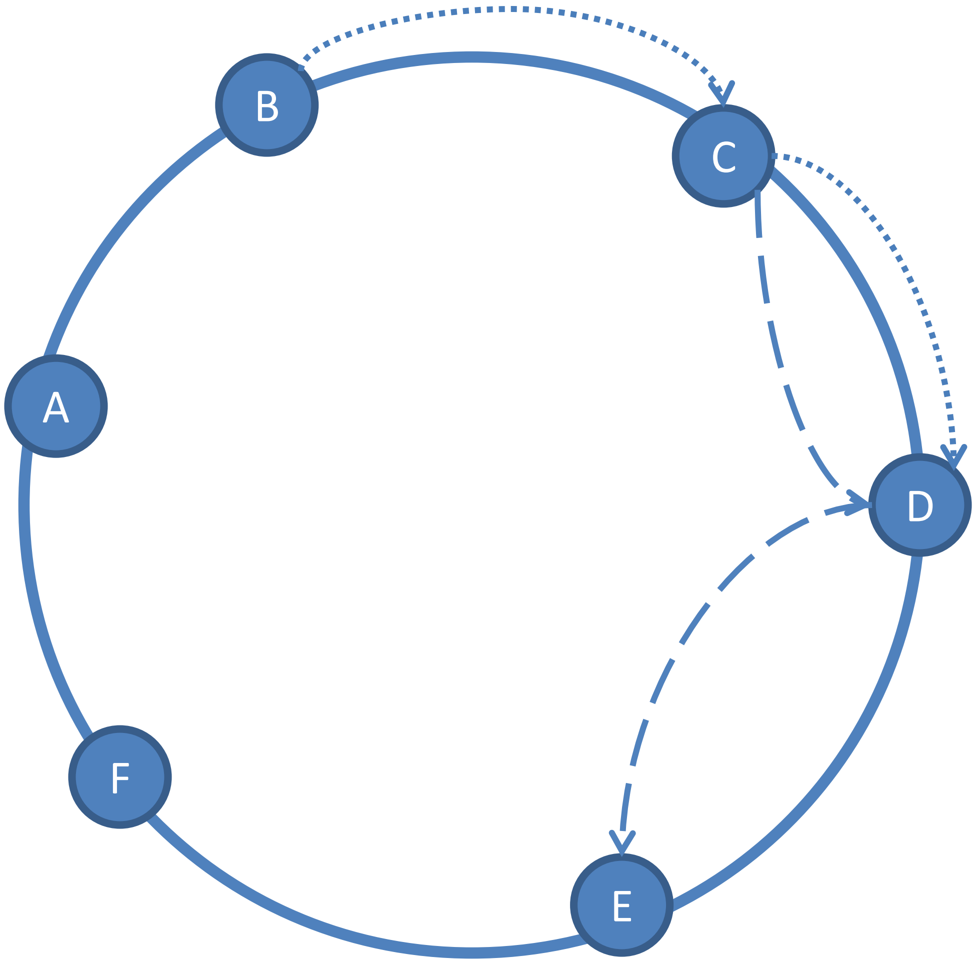
Flash

- Fast random access
  - Optimized for random reads
- Slow random writes
  - Sequential Writes using append only log

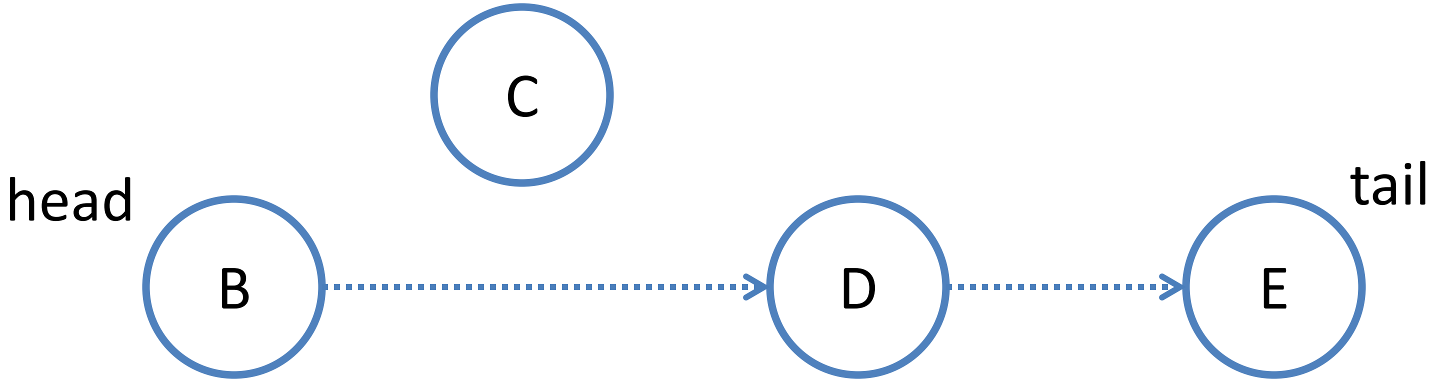
FAWN-DS


Hash Index
000
001
010
100
101
Data Log
Index bits
1001
101
Key Fragment
Offset
1001
Y
Key Fragment
Valid

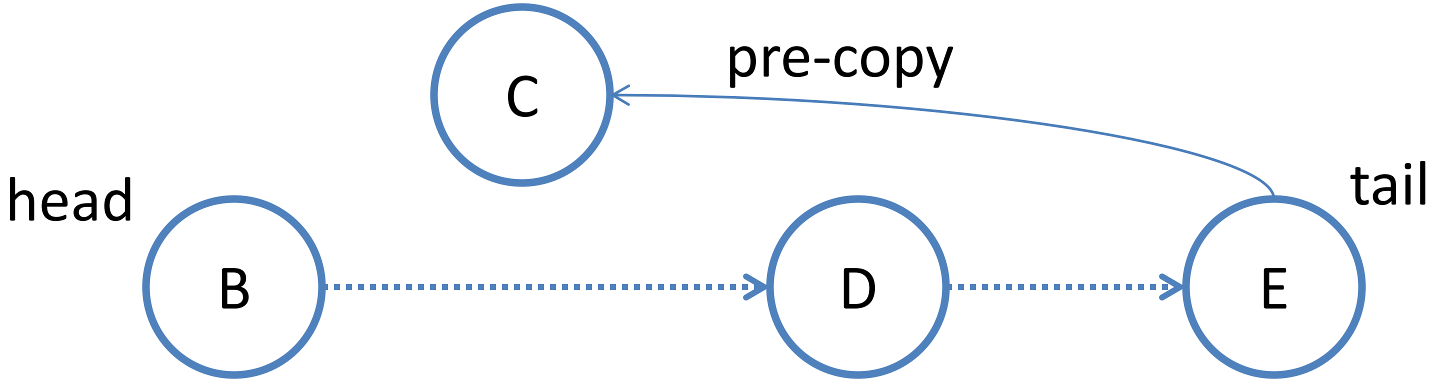
FAWN – Replication R = 3



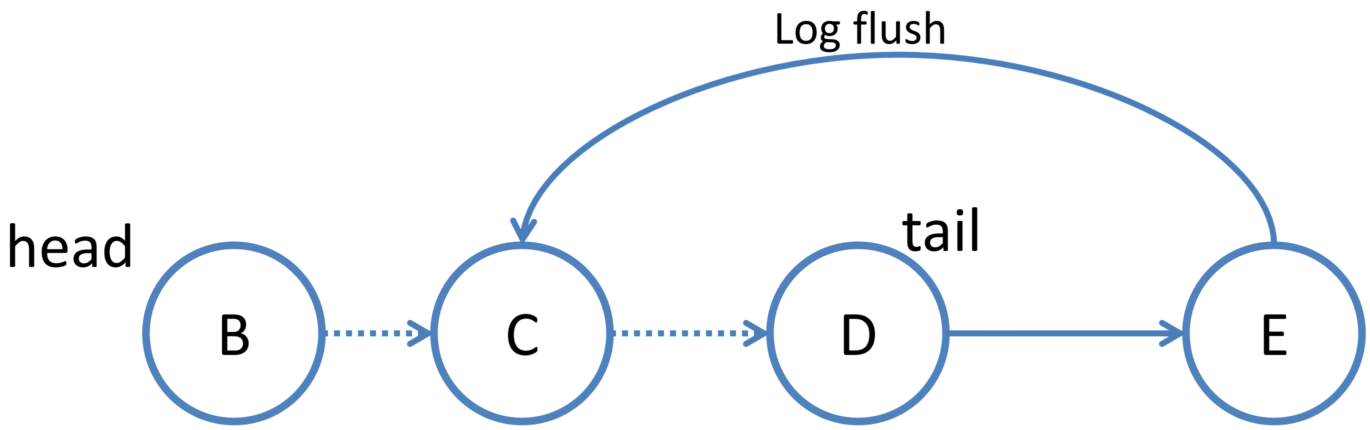
FAWN – Join Protocol



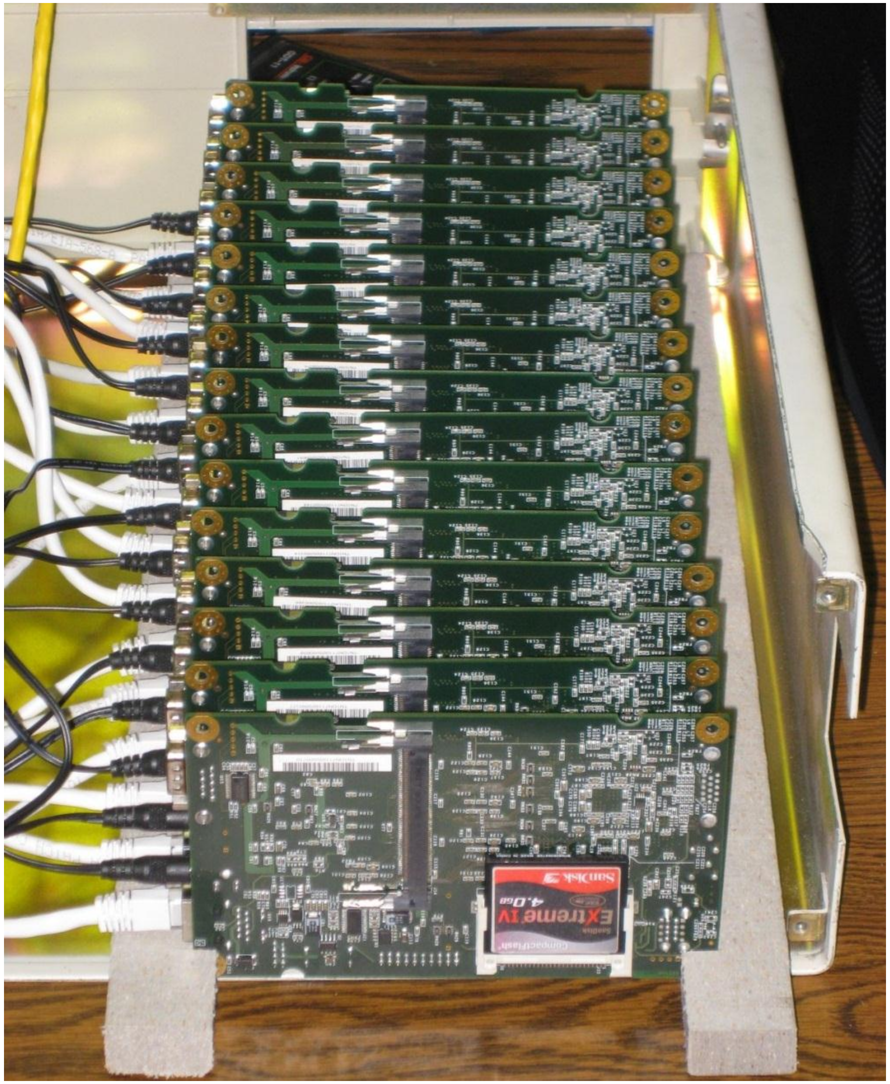
FAWN – Join Protocol



FAWN – Join Protocol

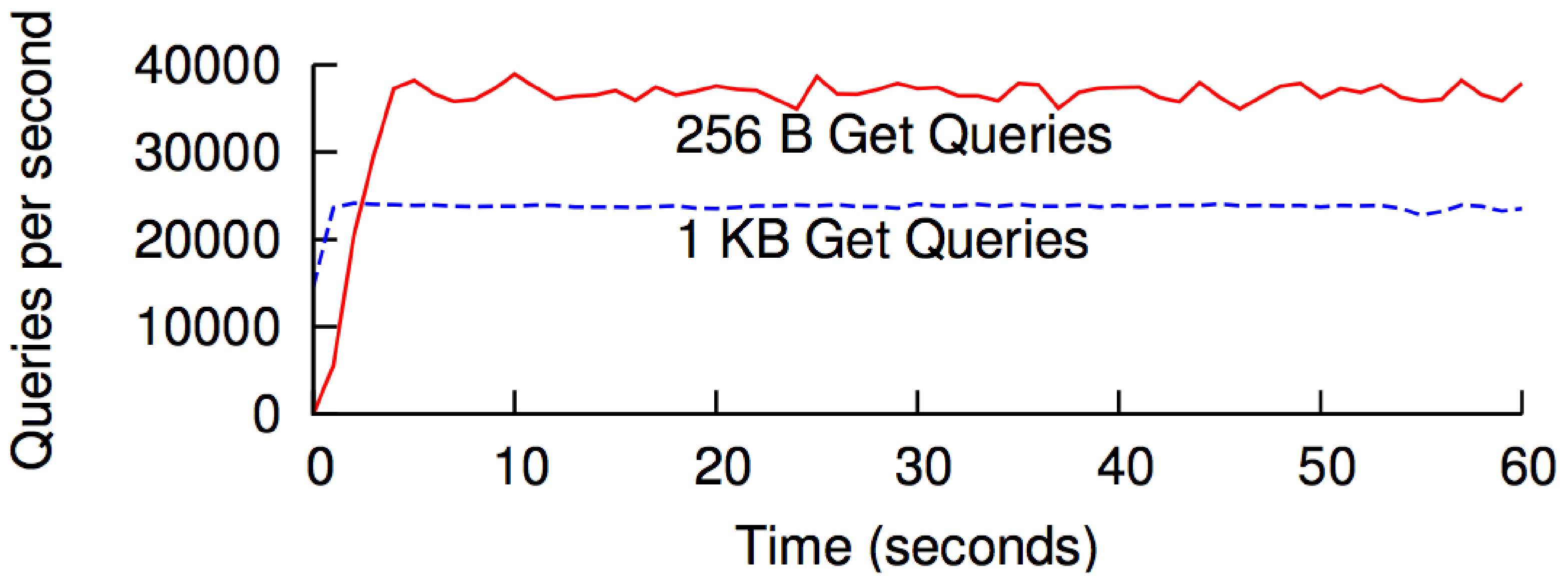


FAWN cluster



Source:
http://www.cs.cmu.edu/~fawnproj

Throughput



1100 - 1700 QPS per node

21 Node FAWN cluster - 20 GB data

No Frontend cache

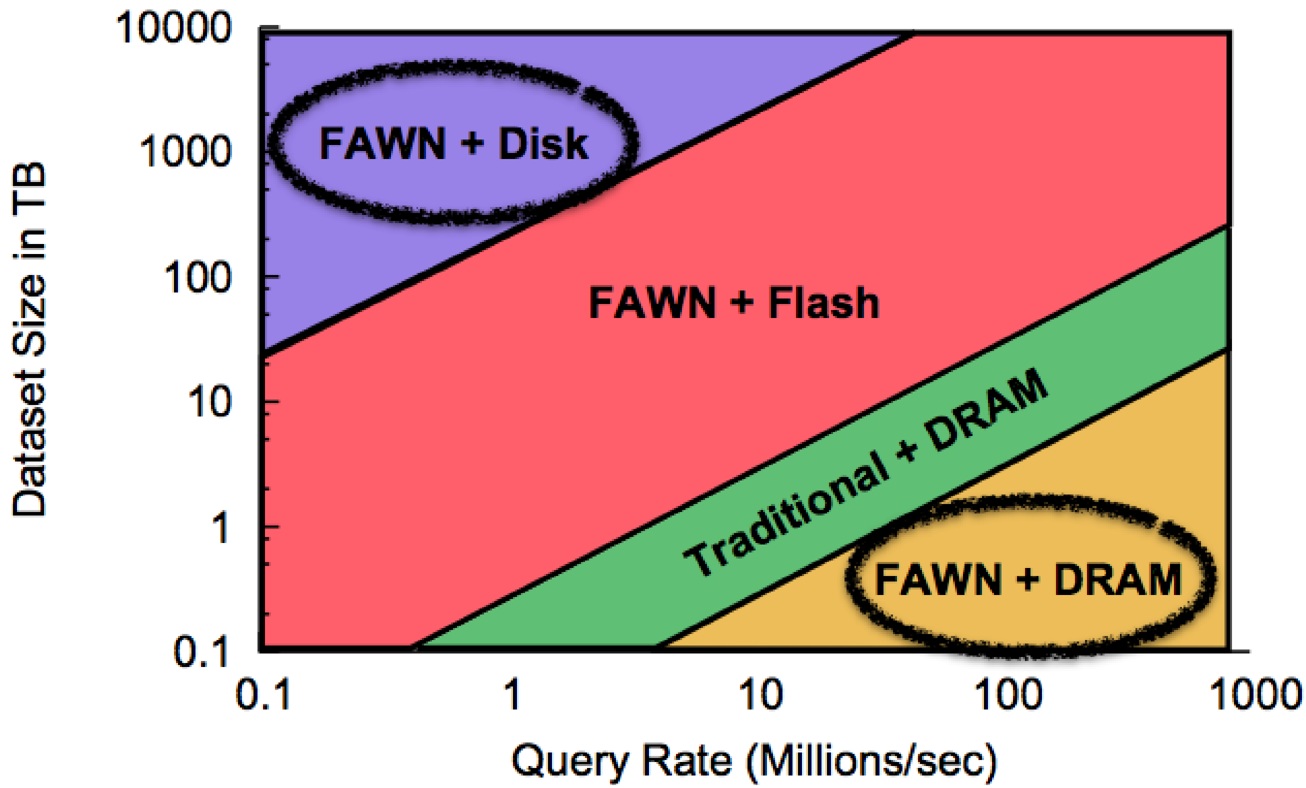
Source: http://www.sigops.org/sosp/sosp09/papers/andersen-sosp09.pdf

Performance and Power

System/Storage	QPS	Watts	Queries/Joule
Alix3cs/Sandisk(CF)	1298	3.75	346
Desktop / Mobi(SSD)	4289	83	51.7
Desktop / HD	171	87	1.96

256 Byte lookups

FAWN vs Traditional servers



Total Cost = Capital Cost + 3 Year Power Cost (0.10 /kWh)

Source: http://www.sigops.org/sosp/sosp09/slides/andersen-slides-sosp09.pdf

Discussion

- Rethink Hadoop/Dryad for FAWN
  – Read as Key Value Pairs in place of bulk reads
- Low Power Processor vs SSD savings
  – CPU Intensive workloads
- From RAID to FAWN
  – I/O bound drives, Memory wall
  – Flash Arrays, Limited power
- Log Based store
  – More efficient for frequent reads

Questions

SSD vs HDD

	Sandisk 5000	HDD – WD2500 250GB 7400RPM
Access Time	0.1 ms	13.4 ms
Sequential read rate	28.5 MB/s	
Sequential write rate	24 MB/s	
Random read IOPS	1424 QPS	
Random write IOPS	125 QPS	

HAYSTACK

Peter Vajgel, Doug Beaver and Jason Sobel

Presented by: Rini Kaushik

Facebook Photo Storage Needs

- 15 Billion Photos
  - Each has 4 images → 60 Billion Photos → 1.5PB (10^{15}) Bytes
- Growth rate
  - 220 million new photos/week → 25TB storage
- Bandwidth requirements
  - 550,000 photos/second
  - Assuming avg photo size = 1MB → 550GB/sec bandwidth
- 300 million users currently
  - 1.3 billion people with quality internet
  - 4x growth possible
- Two workloads
  - Profile pictures – heavy access, smaller size
  - Photos – intermittent access, more in the beginning, periodically afterwards

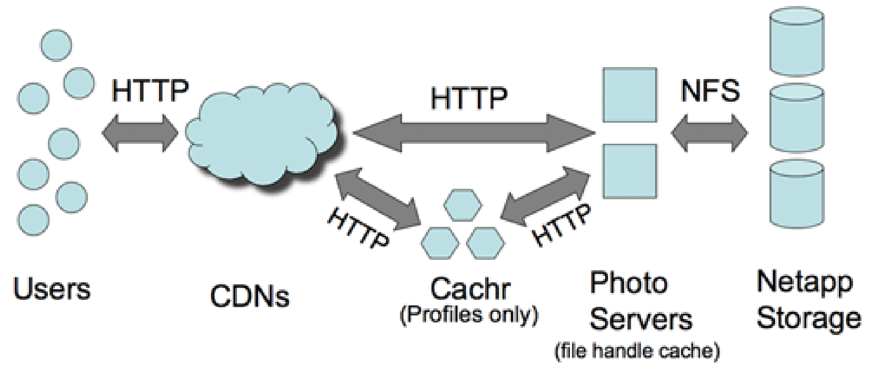
Motivation for Haystack

- Build a specialized store for photo workload
  - Highly Scalable to meet growing storage needs
  - High disk bandwidth
    - Reduce metadata disk IO
  - Reduce Content Distribution Network (CDN) reliance
  - Build from commodity servers as opposed to expensive Netapp filers ($2million each)
  - Simple key-value lookup of photos, no need for Posix

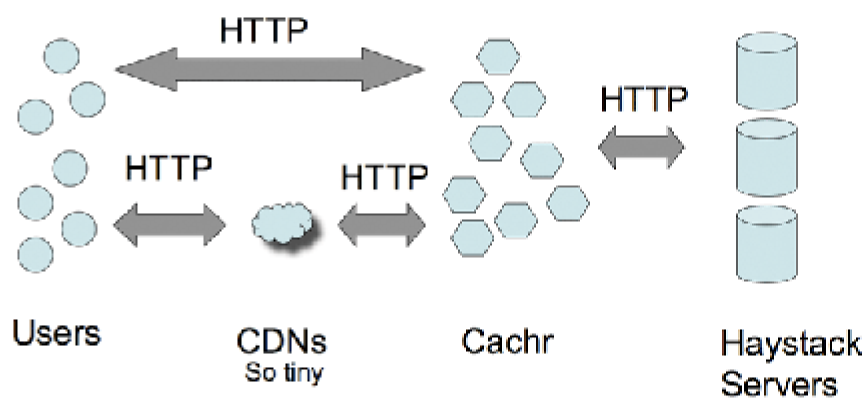
Enter Haystack

- Generic Object store
- Several photos (needles) combined into a large 10 GB append able file (Haystack)
- Index file per Haystack for determining needle offsets

Then and Now



NAS

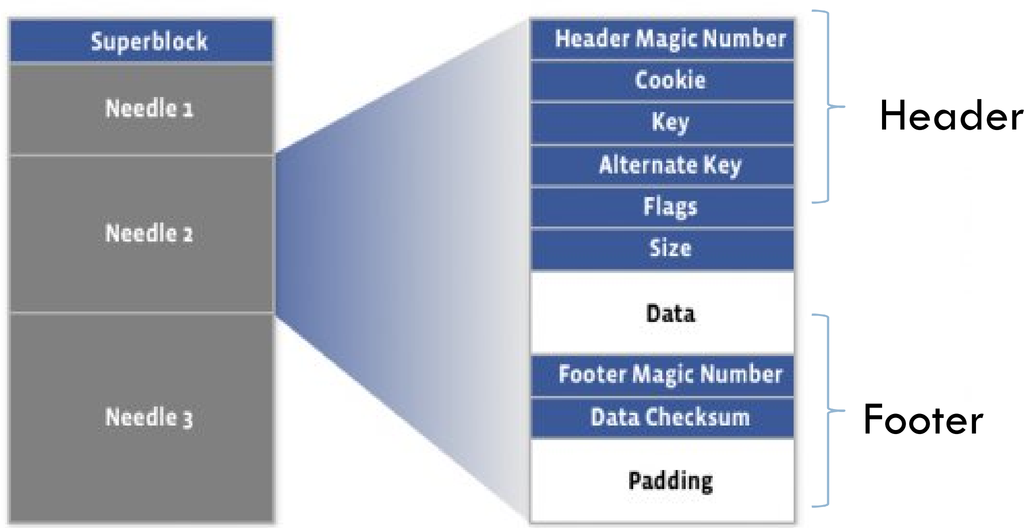


Haystack

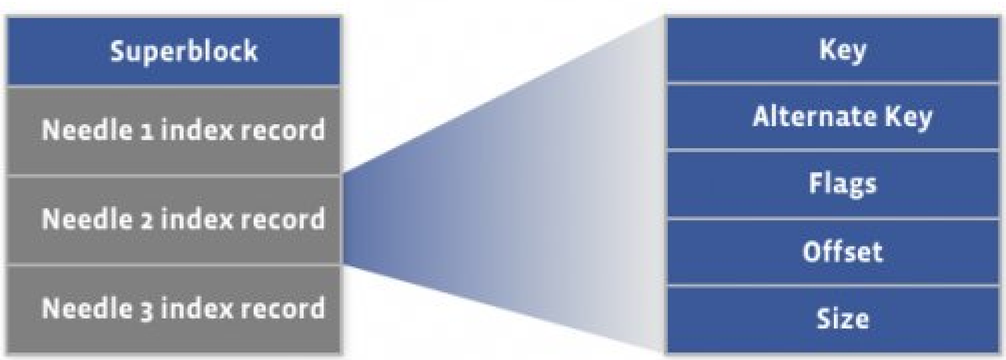
Storage Challenges before Haystack

- Photos stored in traditional Netapp's NFS Filers (Network Attached Storage (NAS))
- Metadata Too Huge to be Cached
  - Posix compliance resulted in more metadata/file
  - Each image a file → 60 Billion Files → 15TB metadata (256B inode)
- 10 disk IO (3 with lookup cache) per file for metadata
  - Drastically reduces disk throughput
- No direct path from client → storage → limited bandwidth
- Result
  - Relied heavily on CDNs to cache data to meet goals
    - 99.98% hit rate profile
    - 92% photos
  - NAS more as a backup
    - Inefficient and Expensive

Haystack object



Haystack



Index File

Advantages

- Reduced disk IO → higher disk throughput
  - 1 MB of in-memory secondary metadata for every 1GB on-disk
    - 10TB per node → 10GB metadata → easily cacheable
- Simpler metadata → easier lookups
  - Not posix compliant
  - No directory structures/file names → 64 Bit ID instead of file names
- Single photo serving and storage layer
  - Direct IO path between client and storage
  - → higher bandwidth
- Less metadata for XFS

Haystack Infrastructure

- Photo Store Server
  - Accepts HTTP requests → Haystack store operations
  - Maintains in-memory Haystack Index
- Haystack Object Store
- Filesystem
  - Extent based XFS
- Storage
  - 2 x quad-core CPUs
  - 16GB – 32GB memory
  - hardware raid controller with 256MB – 512MB of NVRAM cache
  - 12+ 1TB SATA drives

Operations

- Upload
  - Photo assigned 64 bit ID
  - Scaled into 4 image sizes
  - profileID, photo key → pvolID (volumeID) mapping stored in MySQL DB
    - pvolID used to identify the volume container of a haystack
- Read
  - profileID , photo key, size and cookie
  - Output – needle data
- Write/Modify
  - Selects a haystack to store the photo
  - Updates in-memory index
  - Modify results in new version with higher offset
- Delete

Existing limitations/Discussion

- Adhoc data allocation of photos→ haystacks
  - If photos (in the same album) are placed at different times by the same user, it would be good if they are placed sequentially or close by for better data locality.
- No support for delete/overwrites
  - May lead to a lot of unnecessary versions and data → hence, reduced storage efficiency
- Compaction operation seems to be pretty expensive as it involves creating a new copy of haystack. LFS has a much more sophisticated cleaning mechanism
  - What happens if request come at the same time?
- If a file is updated, is it guaranteed to be placed on the same Haystack ID or a separate one? If old Haystack is already full, how will version check work? How will the older versions get identified and deleted?
- Assumes just one disk read per photo
  - what if XFS doesn't have the information in the cache, then it will have an extra lookup for the file
  - Once, Haystack's size becomes bigger than the largest extent size supported by XFS, extra lookups may be necessary if a needle is split across extents
- It would be good to have an abstraction at the album level as well to reduce the lookup overhead

Existing Limitations

- Haystack is tailored for small files that don't change very often, instead of for a small number of large files that are changing all the time.
- Privacy concerns about photo accesses—are cookies sufficient?
- The volume id is hardcoded in the photo which may be a problem if the haystacks need to be moved to a different volume for capacity balancing. Some indirection would have been good
- How is consistency maintained between the CDN and the Haystack?

Questions

- Does every node = 1 haystack or multiple haystacks?
- Why is the haystack expected to be just 10G? What is the rationale?
- How is the haystack to node mapping done?
- Why aren't access permissions important. How else do they enforce security especially if the clients are reading the photos directly?
- What happens if an overwrite comes and haystack is already full, the new version may land in lower offset

Posix compliance resulted in more metadata/file

- File length
- Device ID
- Storage block pointers
- File owner
- Group owner
- Access rights on each assignment: read, write execute
- Change time
- Modification time
- Last access time
- Reference counts

NAS/Clustered NAS Limitations

- Limited in capacity, bandwidth and scalability
- Single Filer (NFS head) clients → NFS filer → storage
  - No direct path from client → storage → limits bandwidth
- Clustered Filers
  - Multiple filer heads, still no direct IO path
- NFS protocol has inherent limitations
  - RPC
  - Memory copying
  - Too many name lookups
  - Small block transfer size

A Case for redundant Arrays of Inexpensive disks (RAID)

Feb 2010

D. Patterson, G. Gibson & R. Katz
Presented by: Rini Kaushik

Disk/CPU Trends

Capacity		Transfer Rate		Rotation + Seek Time		CPU	
↑	60% / Yr	↑	40% / Yr	↓	8% / Yr	↑	2x in 1.5 yrs until 2002
2x in 1.5 yrs		2x in 2 yrs		1/2 in 10 yrs		Now 20% / yr	

- Access time = Seek Time + Rotational Latency + Size/BW

Positioning time (Seek Time + Rotational Latency)

Transfer time (Size/BW)

Limited by:
•Mechanical Delays
•Settle time
•Capacity/Cost/Power/Performance tradeoffs
•4 + 2 = 6ms (17W, 300GB)
•8.5 + 4.2 = 12.7ms (11W, 1TB)

Areal density 130Gbits/sq inch
Sustained internal transfer rate – 125MB/sec

Disk Wall

Type	Cache	Main Memory	Disk Storage
Access time (ns)	0.5-25	50-250	10,000,000
Bandwidth (MB/sec)	5000-20,000	2500-10,000	50-120

- 2GHz CPU – 0.25ns
- Well tuned and highly concurrent OLTP application blocks for IO 10% of the time
- Amdahl's law
  - CPU 10X faster, still speedup 5X
  - CPU 100X faster, still speedup 10X – huge potential wastage
- Discussion – When and how can we amortize the disk wall?

Exponential growth in the IO needs

- **High IO per sec rate**
  - Low-end Exchange server's IO per sec needs:
    - Average user, .75 IOPS × 2,000 mailboxes=1,500 IO per sec

115 IO/s

- **High data rate (bytes transferred/sec)**
  - HD Media, content distribution and editing – 15GB/sec
  - Petascale – 100GB/s

125 MB/s

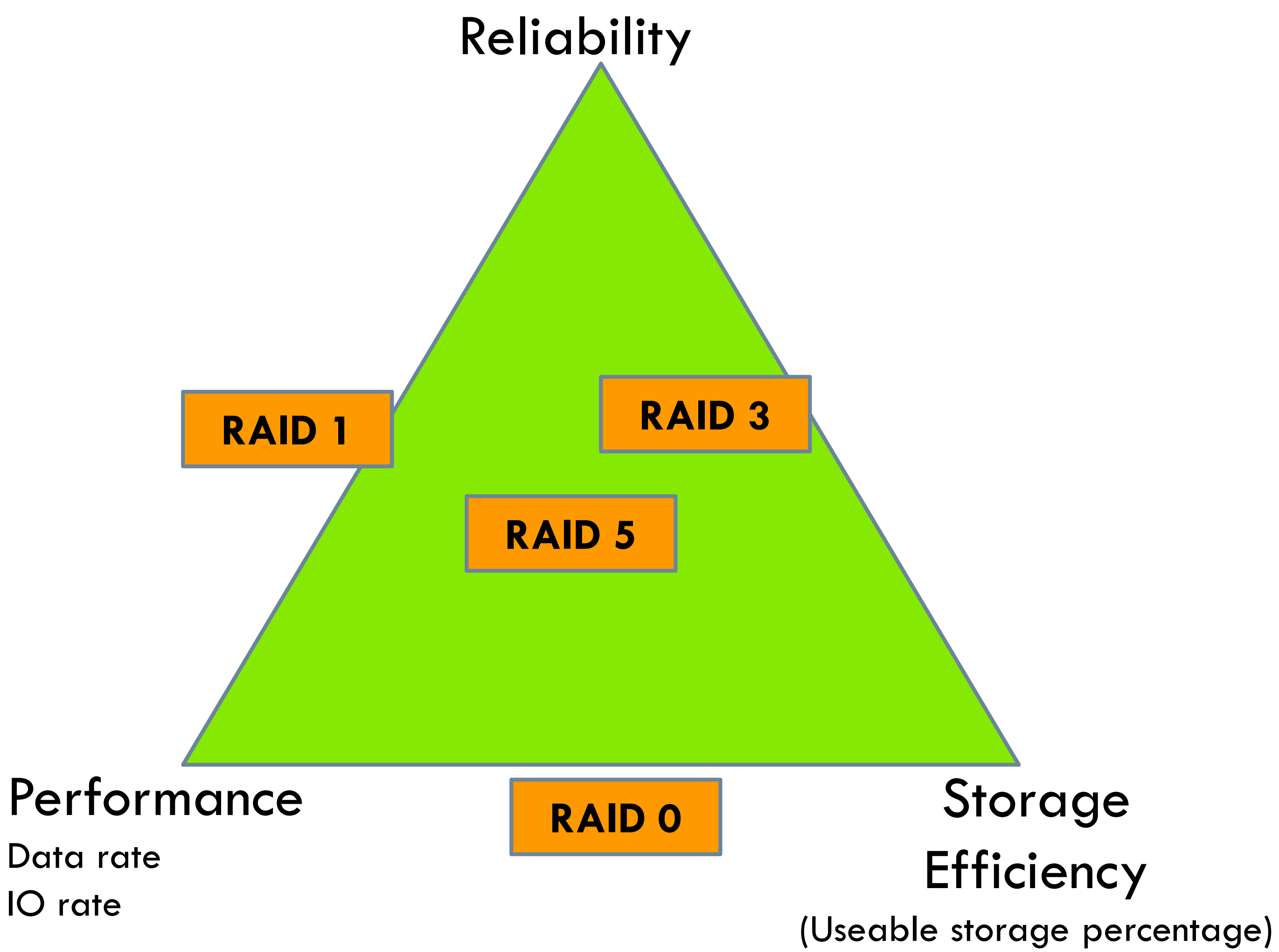
- **Capacity needs – Petabytes**
  - Cancer research, a single drop of blood generates more than 60 Gigabytes
  - Digital content, including media and entertainment, imaging

2 TB

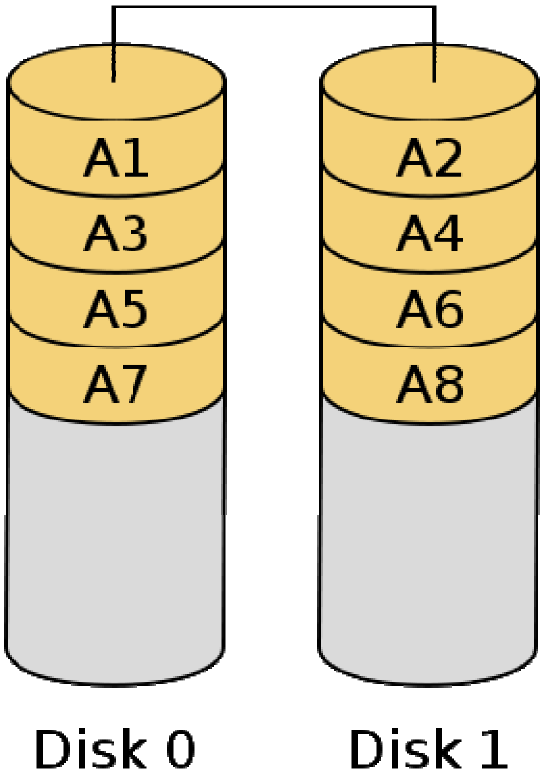
Redundant Array of Independent Disks

- Higher performance -- Striping
  - Higher Data rate (MB/s)
    - Multiple disks cooperate in transferring one large block
  - Higher I/O per second
    - Multiple independent disks service multiple independent requests
- Better Reliability
  - Via redundancy
  - Fault tolerance of 1-2 disks
  - Availability during recovery
- At **lower cost and power** than Single Large Expensive Disk (SLED)

RAID Levels


Reliability
RAID 1
RAID 3
RAID 5
RAID 0
Performance
Data rate
IO rate
Storage
Efficiency
(Useable storage percentage)

RAID 0



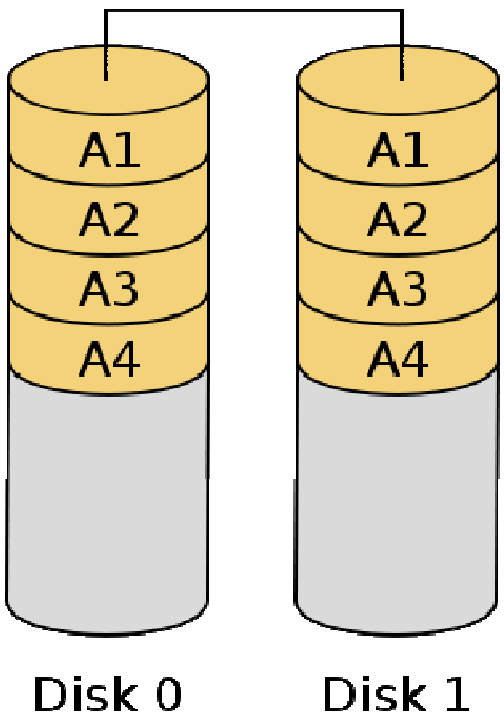
- ✓ High performance
  - ✓High read/write data rate
  - ✓High read/write IO rate
- ✓ High storage efficiency
- X Zero fault tolerance

No redundancy

	Small Read	Small Write	Large Read	Large Write	Storage Efficiency
RAID 0	N	N	N	N	1

N = # of Disks in the stripe

RAID 1

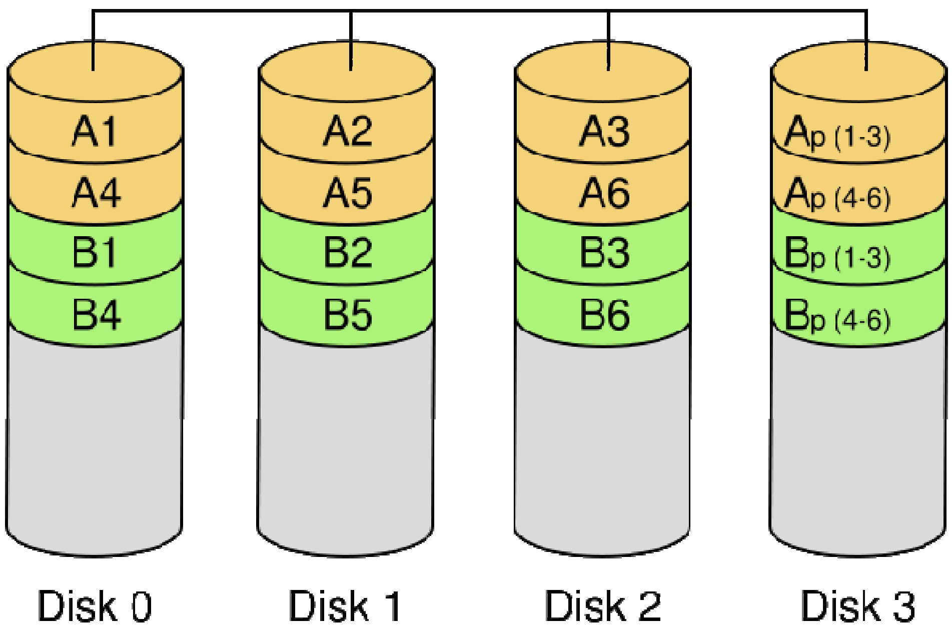


Synchronized
Slowdown = 1
Unsynchronized
Slowdown <= 2

- ✓ High performance
  - ✓ High read data rate
  - ✓ High read IO rate
  - ✓ OK write IO/data rate
    1 write → 2 writes
- ✓ Best fault tolerance
- ✓ Lowest recovery time
- X Low storage efficiency

	Small Read	Small Write	Large Read	Large Write	Storage Efficiency
RAID 0	N	N	N	N	1
RAID 1	N	N/2	N	N/2	0.5

RAID 3

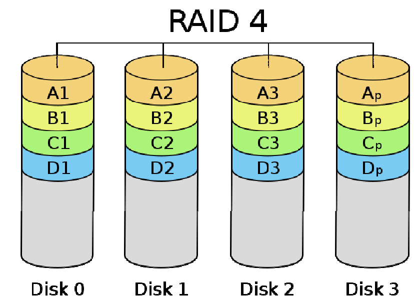


✓High Sequential read/write data rate
✓Good storage efficiency
✓Fault tolerance for one disk failure
X Very poor Random read/write IO rate
1 small read/write spans all disks and reduces concurrency

	Small Read	Small Write	Large Read	Large Write	Storage Efficiency
RAID 0	N	N	N	N	1
RAID 3	1	0.5	N-1	N-1	(N-1)/N

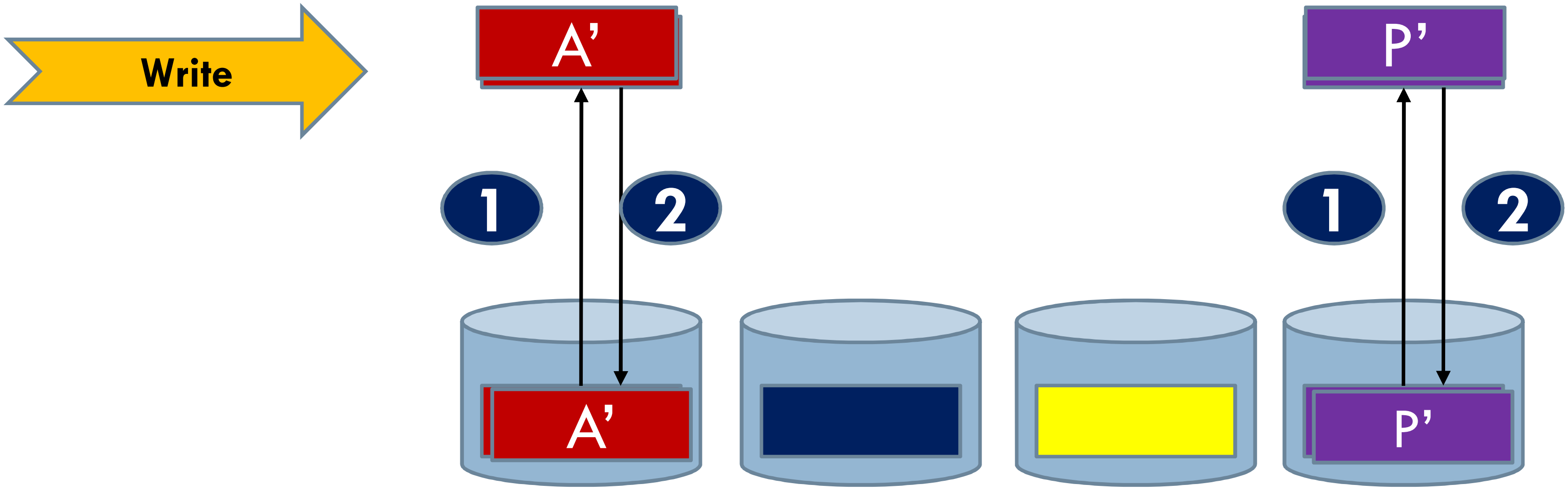
Byte Interleaved
Single Parity disk

RAID 4/5



- Interleaving Granularity – Block level
- Pros
  - High small read performance
  - Large reads/writes that span the entire stripe are very efficient
- Cons
  - Dismal Low Small write performance
  - Single parity disk needs to be updated for all writes and serves as bottleneck
- Discussion – Additive or subtractive parity?
- Discussion – What can we do to remove the single parity bottleneck?

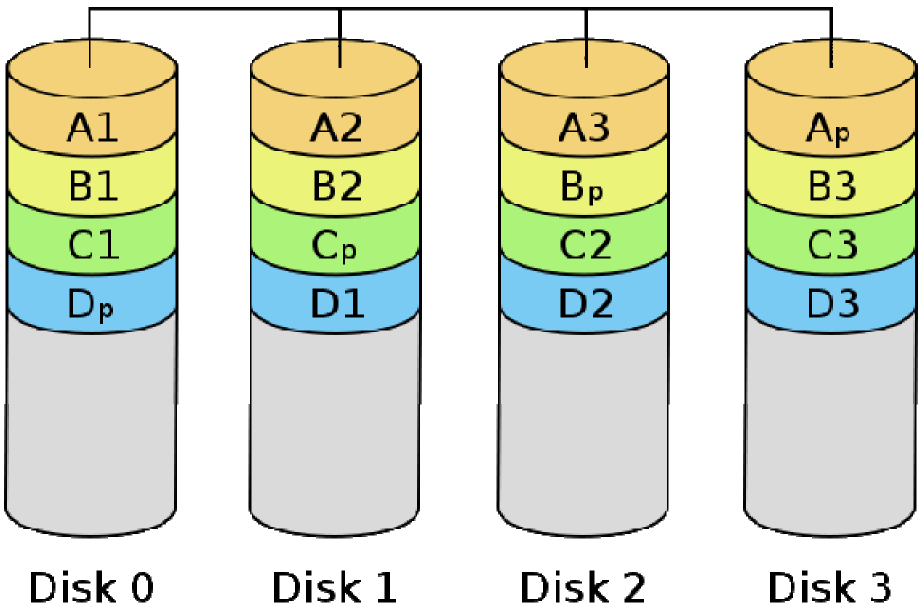
RAID 4/5 Small Writes



Subtractive Parity Computation
P' ← A ⊕ **A'** ⊕ **P**

	Small Read	Small Write	Large Read	Large Write	Storage Efficiency
RAID 0	N	N	N	N	1
RAID 5	N	0.25N	N, N-1	N-1	(N-1)/N

RAID 5



- ✓High Sequential read/write data rate, read random IO rate
- ✓Good storage efficiency
- ✓Fault tolerance for one disk failure
- ✓ No parity bottleneck
- X Random write performance very poor

Discussion – How can we improve the small write performance?

Interleaving Granularity
Block level
Distributed Parity

Performance Comparison

	Small Read	Small Write	Large Read	Large Write	Storage Efficiency	Fault Tolerance	Usage
RAID 0	1	1	1	1	1	0	Scientific computing
RAID 1	1	0.50	1	0.50	0.5	1	OLTP E-Commerce
RAID 5	1	0.25	1	(N-1)/N	(N-1)/N	1	Webserver Multimedia OLAP DSS

Discussion – Which workload will be most suitable for each of these levels and why?

Throughput relative to RAID 0 for performance/cost
N = # of disks in a Group

Reliability

Single Disk	MTTF	200,000 hrs = 23 yrs
N disks	MTTF/N	83 days
Single parity RAID	(MTTF * MTTF) / N * (G - 1) * MTTR	3000 yrs
Double parity RAID	(MTTF * MTTF * MTTF) / N * (G - 2) * (G - 1) * MTTR * MTTR	38 million yrs

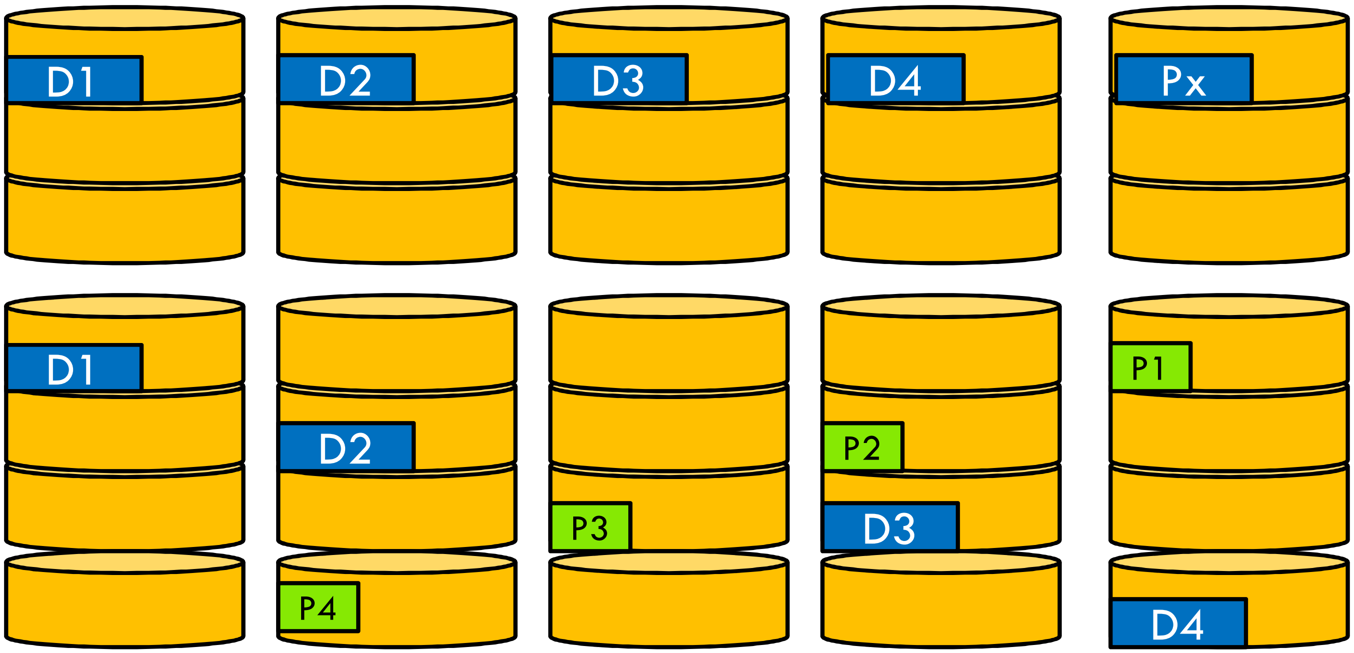
G = Group size = 16
N = Number of disks = 100
MTTR = 1 hr
Assumptions:
 Independent failures
Only disks considered
MTTR – Mean time to repair
MTTF – Mean time to failure

Discussion – Is independent failure assumption valid?

Discussion on Assumptions of Paper's performance analysis

- Assumes a perfect workload
  - Single Full-stripe Large reads/writes only
    - No performance penalty for parity update
- Assumes a perfect layout of files on the disk
  - Sequentially accessed files allocated sequentially on the disks in full-stripes
  - Randomly accessed files perfectly load balanced across disks
    - No Hotspots

Impact of Partial vs. Full Stripe Write



High Impact on performance (with parity)

- **File layout can drastically lower RAID's performance**
- **Reality**
  - File System Fragmentation
  - File boundaries may be unaligned with stripe boundaries

Discussion on Limitations

- Scalability
  - E.g. Single RAID controller bottleneck for throughput (e.g. 6GB/sec LSI Engenio 7900)
  - RAID with striping will need to be rebuild upon adding more disks to the stripe
- Limited fault tolerance
  - Fault tolerance at entire disk level failure
  - No support for data corruption