Federated Array of Bricks

Y Saito et al
HP Labs

CS 6464

Presented by

Avinash Kulkarni

Agenda

- Motivation
- Current Approaches
- FAB
  - Design
  - Protocols, Implementation, Optimizations
  - Evaluation
- SSDs in enterprise storage

Motivation

- Enterprise storage devices offer high reliability and availability
- Rely on custom hardware solutions to gain better performance
- Redundant hardware configuration to eliminate any single point of failure, but still very centralized
- Workload characteristics vary widely based on client application
- Advantages – good performance, reliability, failover capabilities
- Disadvantages – expensive, high cost of development, lack of scalability
- Can we build a distributed, reliable but cheap storage solution that is also scalable and strongly consistent?
- Could we do this without altering clients in any way?

Have we come across some solutions in this class?

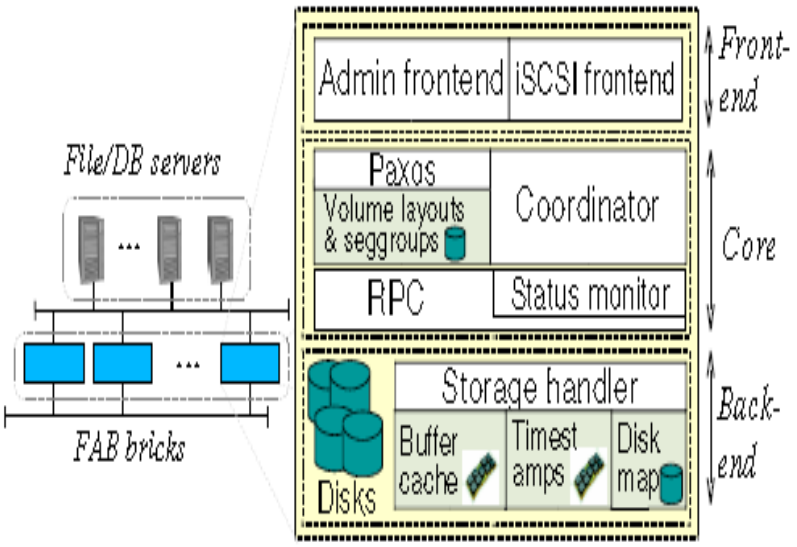
- GFS – Google File System
  - Cheap, Scalable, Reliable
  - Optimized for workloads that are mostly composed of reads, and appends
  - Requires clients to be aware of file system semantics
  - Some aspects are centralized
- Amazon - Dynamo
  - Cheap, Scalable, Reliable
  - Optimized for key-value storage
  - Eventual consistency
  - Pushes conflict resolution on to clients
  - Anti entropy, sloppy quorum, hinted handoff needed to manage ‘churn’
- So, neither fits all our needs

Enter FAB

- Aims to provide a scalable, reliable distributed storage solution that is also strongly consistent and cheap
- Requires no modification to clients
- Innovations
  - Voting based replication
  - Dynamic quorum reconfiguration
- Strongly consistent
  - Safety over liveness
- Not optimized for any particular workload pattern
- Assumptions
  - Churn is low?
  - Failstop
  - Non-byzantine

FAB Structure

- Organized as a decentralized collection of bricks
- Each brick is composed of a server attached to some amount of storage
- All servers are connected with standard Ethernet interface
- Each server runs the same software stack composed
  - iSCSI front end,
  - Core layer for metadata processing,
  - backend layer for disk access
- Each server can take on 2 roles
  - Coordinator
  - Replica
- Observation
  - Buffer cache management is done by backend layer on each individual server. No global cache…

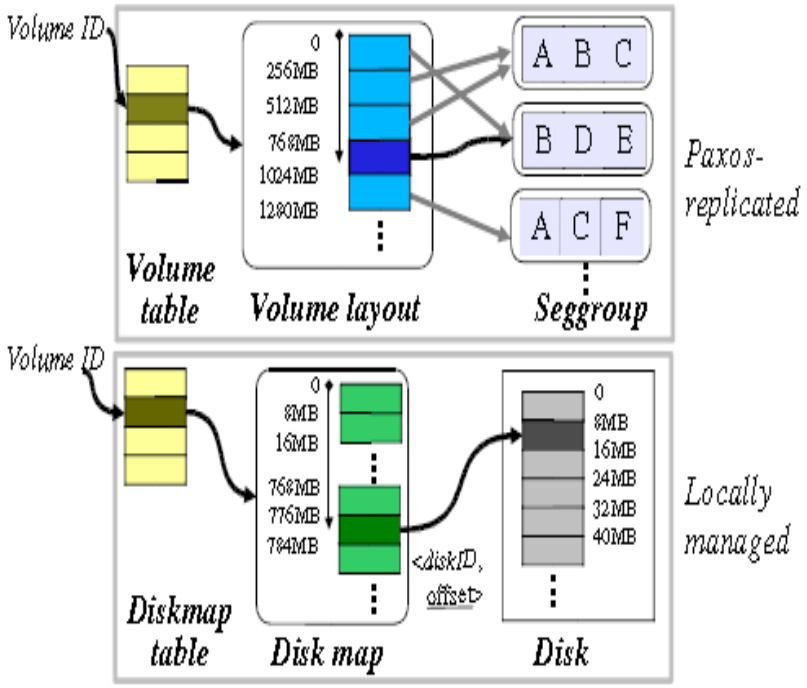


FAB operation

- Client sends a request <vol id, offset, length> to coordinator
- Coordinator locates the subset of bricks that store data for the request
- Coordinator runs replication or erasure code protocol against set of bricks, passing the <vol id, offset, length> to them
- Bricks user the tuple provided to locate the data on physical disks they contain

Key Data Structures

- Volume layout
  - Maps a logical offset to a seggroup at segment granularity(256MB)
- Seggroup
  - Describes layout of segment, including bricks that store the data
- Disk Map
  - Maps logical offset in a request tuple to physical offset on disk at page granularity(8MB)
- Timestamp table
  - Information about recently modified blocks of data



Data layout & Load balancing

- All segments in a seggroup store data at same replication/erasure code level
- Logical segments are mapped semi-randomly, preferring seggroups that are lightly utilized
- Logical block to physical offset mapping on bricks is done randomly
- How many seggroups per brick?
  - Higher the number, the more evenly load can be spread in the system on event of a brick failure
  - But as we increase the ratio, we reduce the reliability of the system because it increases the chances of data loss from combination failures
  - FAB uses 4 as an optimal number
- Load balancing
  - Status messages flow between replicas, exchanging information about load
  - Gossip based failure detection mechanism
  - Not static like Dynamo which relies on consistent hashing for distributing load
  - No concept of virtual nodes to accommodate heterogeneity of storage server capabilities.

Design decisions

- 2 approaches to redundancy
  - Replication
  - Erasure codes
- Replication
  - Each segment of data is replicated across some number of replicas
  - Unlike Dynamo, no version numbers are assigned to copies of data as vector timestamps
  - Instead, each replica maintains 2 timestamps for each 512 byte block of data on the replica
    - ordT: Timestamp of the newest ongoing write operation
    - valT: Timestamp of the last completed write that has been written out to disks

Design Decisions - Replication

- Write proceeds in 2 phases
  - Order phase
    - Generate a new timestamp for this write, and send it to the bricks. Wait till majority accept
  - Write phase
    - Replicas update data from client and set valT to be equal to ordT generated above
- Read is usually 1 phase
  - Except when an incomplete write is detected.
    - Read must now discover the value with the newest timestamp from majority, then writes that value with timestamp greater than any previous writes to a majority
- Unlike Dynamo, Read always returns the value of the last recent write that was committed by a majority.

Design Decisions – Replication Protocol

```
// I/O coordinator code.
proc write(val)
   ts ← NewTimestamp()
   send [Order, {}, ts] to bricks in the seggroup
   if a majority reply "yes"
      send [Write, val, ts] to bricks in the seggroup
      if a majority reply "yes" return OK
   return ABORTED
proc read()
   send [Read] to bricks in the seggroup
   if a majority reply "yes" and all timestamps are equal
      return the val in a reply.
   ts ← NewTimestamp()  // Slow "recover" path starts
   send [Order, "all", ts] to bricks in the seggroup
   if a majority reply "yes"
      val ← the value with highest valTs from replies
      send [Write, val, ts] to bricks in the seggroup
      if a majority reply "yes" return val
   return ABORTED
```

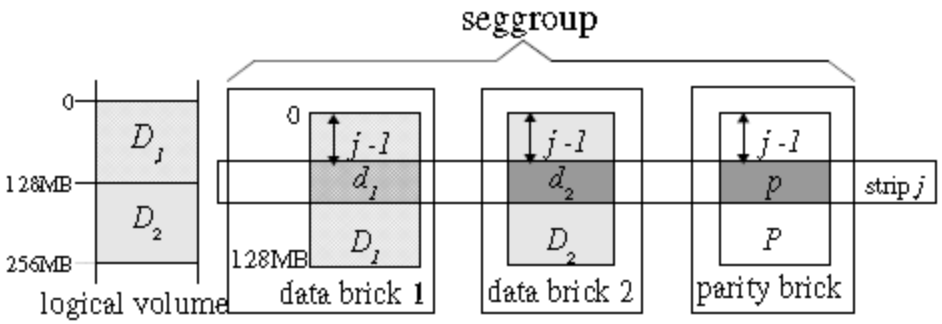

```
// Storage handler code. Variable val stores the block contents.
when Receive [Read]
   status ← (valTs ≥ ordTs)
   reply [status, valTs, val]
when Receive [Order, targets, ts]
   status ← (ts > max(valTs, ordTs))
   if status ordTs ← ts
   if targets = "all" or this block ∈ targets reply [valTs, val, status]
   else reply [valTs, status]
when Receive [Write, newVal, ts]
   status ← (ts > valTs and ts ≥ ordTs)
   if status val ← newVal; valTs ← ts
   reply [status]
```

Design Decisions – Erasure Coding

- FAB supports (m, n) Reed-Solomon erasure coding.
  - Generate (n-m) parity blocks
  - Reconstruct original data blocks by reading any m of n blocks
- Coordinator must now collect m + ceil((n – m)/2) responses for any operation, so that intersection of any 2 quorums will contain at least m bricks
- Write requests are now written to a log during phase 2 instead of being flushed to the disk right away
- On detecting incomplete write, read will scan log to find newest strip value that can be reconstructed
- Log does not have to grow infinitely. Coordinator can direct bricks to compress logs once response is returned to client. A asynchronous ‘Commit’ operation is sent to bricks to direct them to flush log writes to disks

Design Decisions – Erasure Code

- Some modifications to replication layer needed to implement EC on FAB
  - Addition of a log layer on replicas to eliminate strip corruption on incomplete writes
  - Addition of a asynchronous commit operation at the end of write to compress and garbage collect logs on replicas

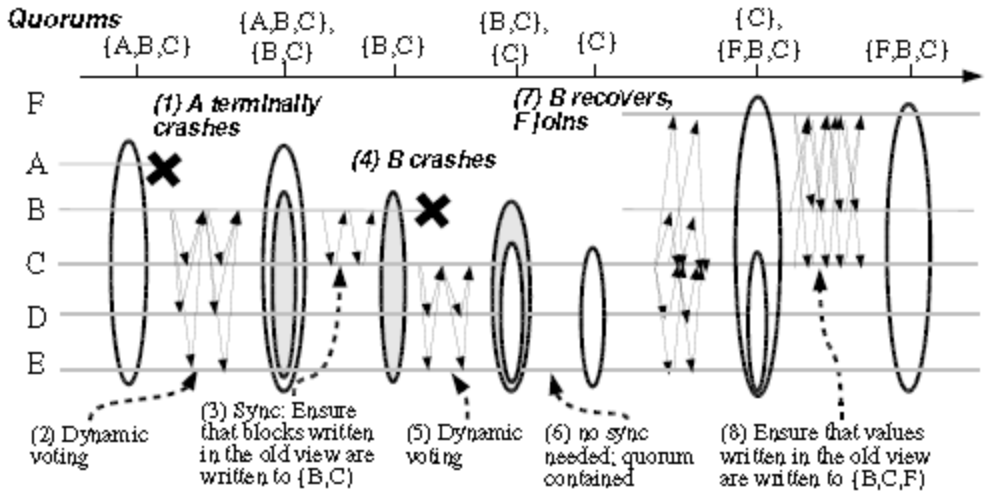


Design Decision - Optimizations

- Timestamps
  - One 24 byte timestamp for every 512byte block is very expensive in terms of storage
  - Recall that timestamps are needed only to distinguish between concurrent operations
  - So, coordinator can ask bricks to garbage collect the timestamps once all replicas have committed the operation to disks
  - Most requests span multiple blocks, so they will have the same timestamps. An ordered tree could be used to store a mapping between timestamps and the data blocks they pertain to
- Read operations
  - Reads of data from a quorum can be expensive in terms of bandwidth. So read requires only timestamps from replicas, while data could be efficiently read off one replica
- Coordinator failures are not handled by the system as a simplification. Clients can detect these, and issue the same request to a new coordinator

Design Decisions - Reconfiguration

- View agreement through dynamic voting in a three-round membership protocol
- Need for a synchronization step after a new view is agreed upon



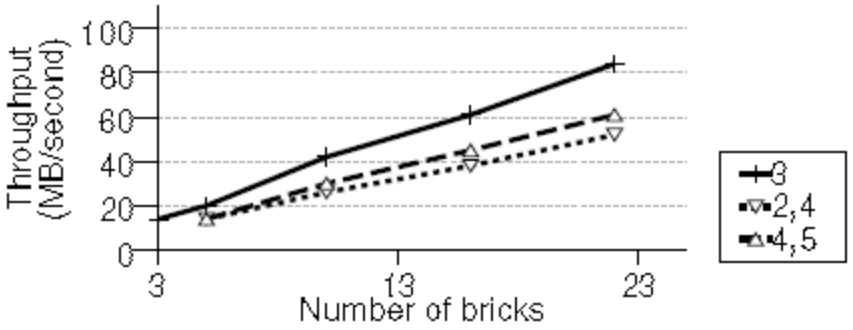
Evaluation

- Application performance
- 3 different workloads – tar(bulk reads), untar(bulk writes), compile(reads and writes)

	Untar	Tar	Compile
Local disk	21.76	14.80	318.9
Local RAID 1	22.32	14.64	319.2
iSCSI+raw disk	24.21	24.32	323.9
FAB (3way repl.)	21.57	24.61	316.0
FAB (2,4 erasure code)	38.22	27.81	322.0
FAB (4,5 erasure code)	33.33	26.22	319.5
FAB (3way repl., no cache)	28.34	26.13	327.0

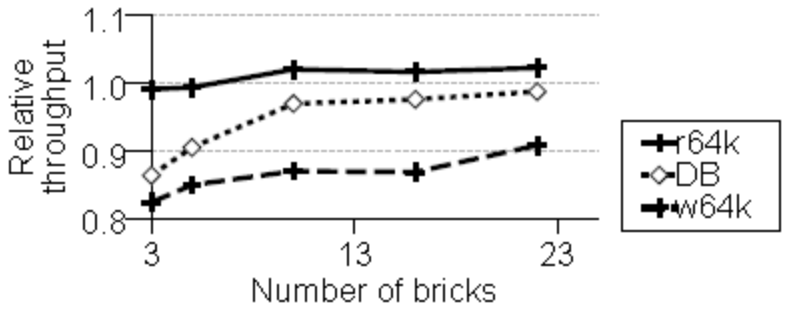
Evaluation

- Scalability – aggregate throughput and random read/write throughput



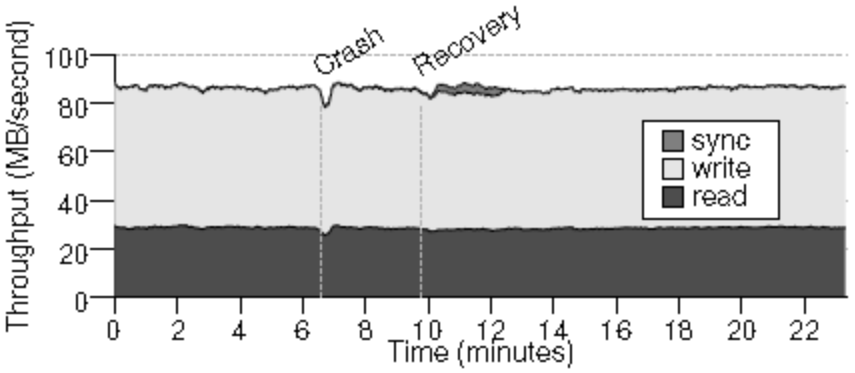
Evaluation

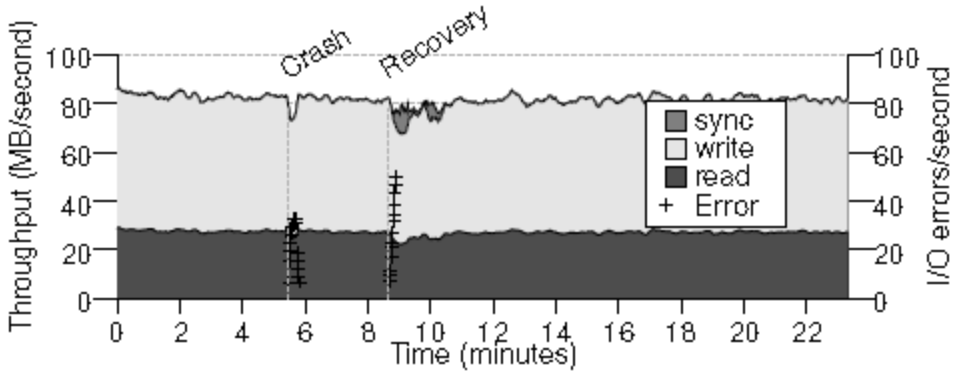
- Scalability – comparison of throughput with master-slave FAB



Evaluation

- Handling changes – comparison with master-slave FAB





SSDs in enterprise storage – An analysis of tradeoffs

Dushyanth Narayanan et al

Are SSDs useful in current enterprise environment?

- Use of SSDs in enterprise storage systems requires optimizing for performance, capacity, power and reliability
- Currently, SSDs offer very high advantage in terms of random access read operations, and lower power consumption
- But they are very expensive
  - Orders of magnitude costlier than current disks(IDE/SCSI)
  - Prices have been declining, and predicted to fall further
  - But are there situations in current enterprise environment where SSDs could be used now?

Analysis of tradeoffs

- Here's a listing of device characteristics

Device	Price (US$)	Capacity (GB)	Power (W)	Sequential xfer (MB/s)		Random access (IOPS)		Wear rate (GB/day)
				Read	Write	Read	Write	
Memoright MR 25.2	739 509 389	32 16 8	1.0 1.0 1.0	121	126	6450	351	500
Seagate Cheetah 10K	339 123	300 146	10.1 7.8	85	84	277	256	n/a
Seagate Cheetah 15K	172 349	146 300	12.5 12.5	88	85	384	269	n/a
Seagate Momentus 7200*	150 53	200 160	0.8 0.8	64	54	102	118	n/a

Analysis of tradeoffs

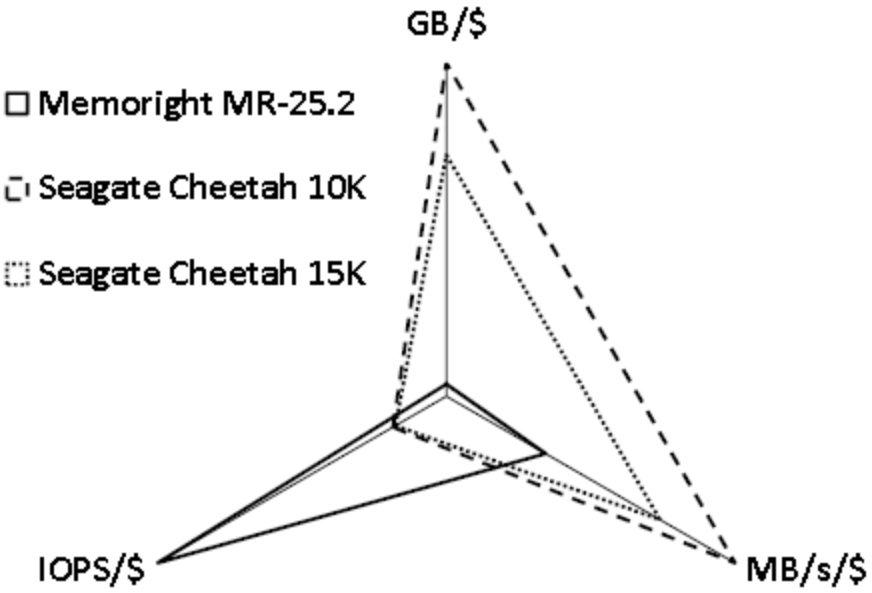
- Authors traced typical workloads in enterprise environment and enumerated device characteristics

Metric	Unit
Capacity	GB
Random-access reads	IOPS
Random-access writes	IOPS
Random-access I/Os	IOPS
Sequential reads	MB/s
Sequential writes	MB/s
Availability	Number of nines
Reliability	Number of nines

Metric	Unit
Capacity	GB
Performance	IOPS (read, write)
	MB/s (read, write)
Reliability	MTBF
Wear rate(SSDs)	GB/day
Power consumption	W (idle and active)
Purchase cost	$

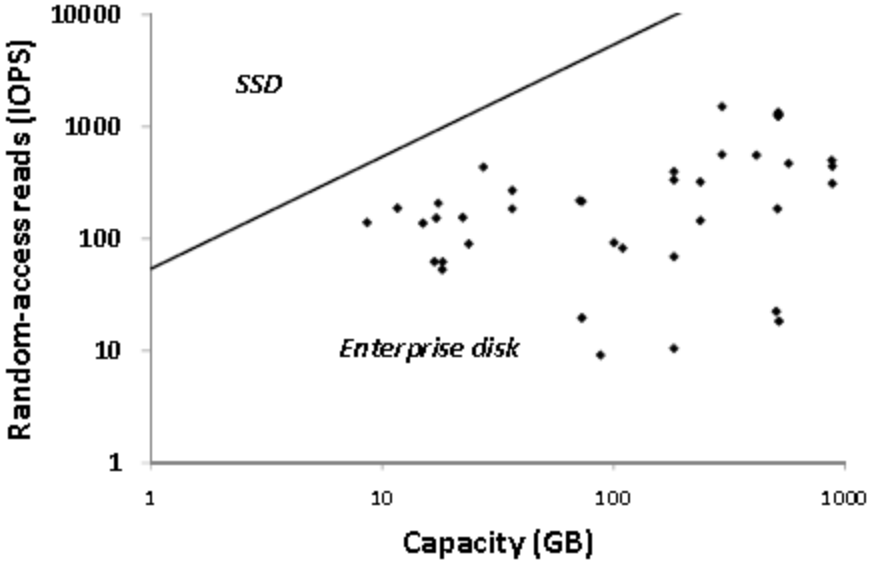
Analysis of tradeoffs

- How do SSDs rack up against normal drives?



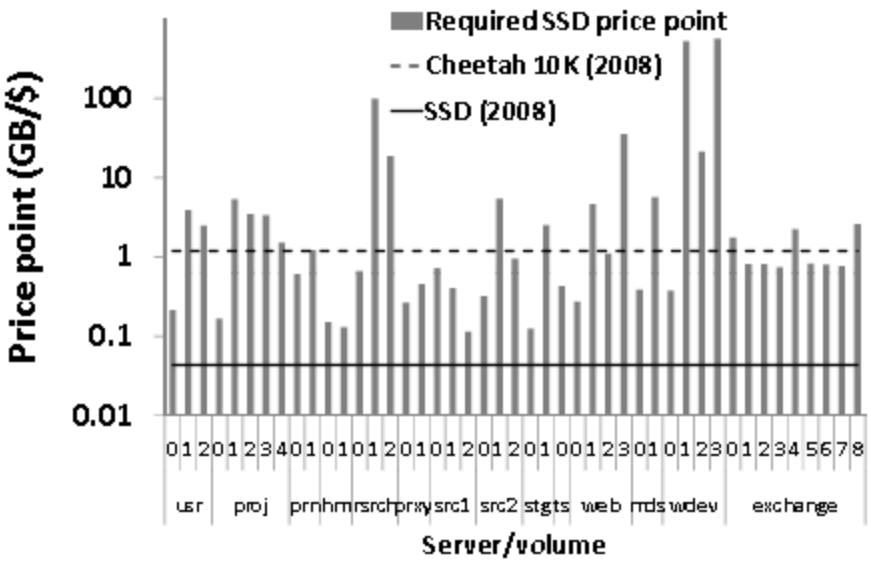
Analysis of tradeoffs

- Can SSDs completely replace normal drives for the workloads traced based on their capacity/random-reads requirement? Not really.



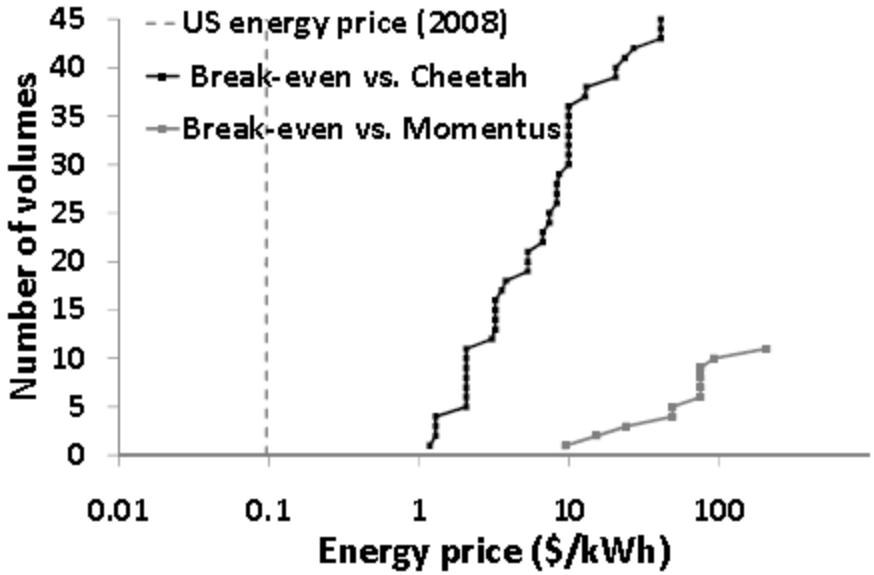
Analysis of tradeoffs

- How do SSDs measure up to normal drives on price points? Not very well at present.



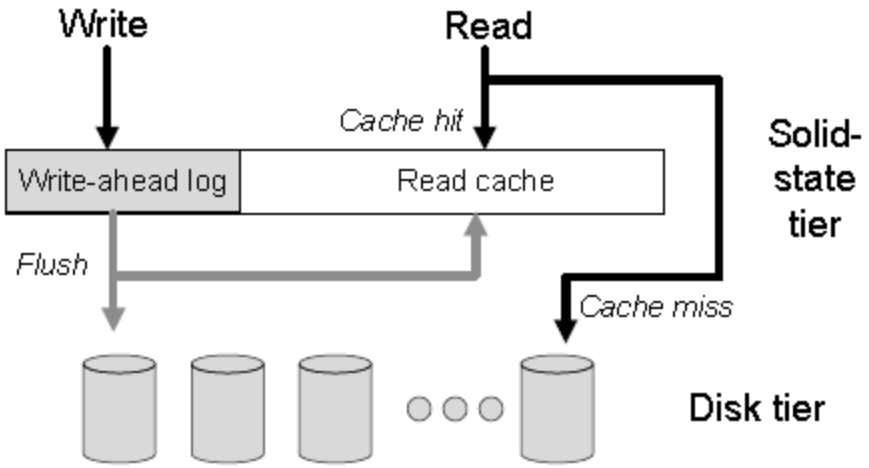
Analysis of tradeoffs

- How about power? Can we not save a lot of money based on the small power footprint of SSDs? Turns out we don't really save much at current energy prices



Analysis of tradeoffs

- So, is there a place for SSDs in enterprise storage?
- Maybe – recall the tiered storage model we use with DRAM and normal drives
- DRAM serves as a buffer for underlying slow, normal drives
- Insight – SSDs could do serve the same purpose! They are fast, but cheaper than DRAM. Also, they are persistent, compared to DRAM.
- We need to account for the poor random write performance before we do this. Use newer flash file systems that convert random writes to sequential log entries



Some discussion points

- Each brick in FAB maintains its own cache – how does that affect performance?
- How can FAB accommodate heterogeneous storage servers?
- Any others?

Analysis of tradeoffs