SanDisk®

Flash-Extending In-Memory Computing

Brian O'Krafka, Engineering Fellow

June 30, 2015

Forward-Looking Statements

During our meeting today we will make forward-looking statements.

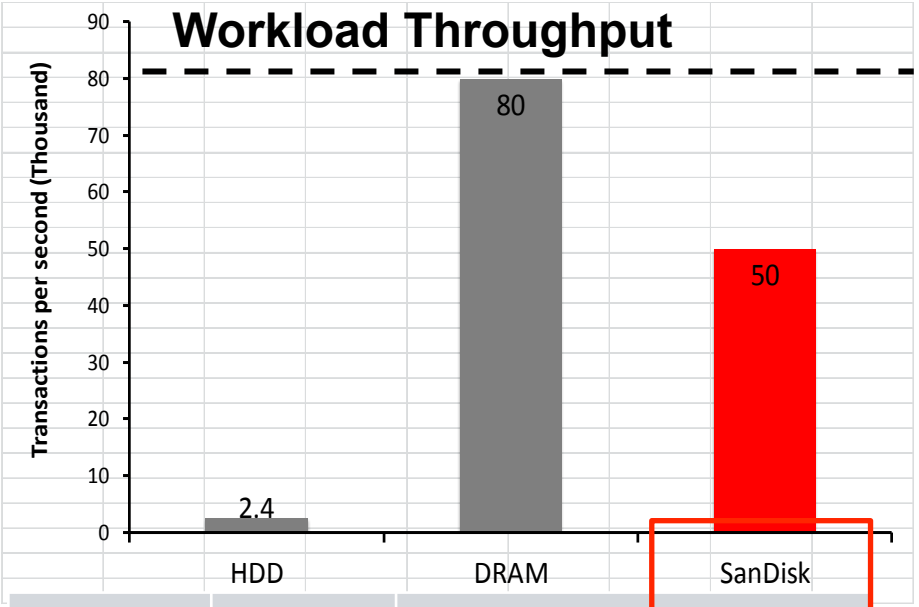
Any statement that refers to expectations, projections or other characterizations of future events or circumstances is a forward-looking statement, including those relating to products and their capabilities, performance and compatibility, cost savings and other benefits to customers.

Actual results may differ materially from those expressed in these forward-looking statements due to a number of risks and uncertainties, including the factors detailed under the caption “Risk Factors” and elsewhere in the documents we file from time to time with the SEC, including our annual and quarterly reports. We undertake no obligation to update these forward-looking statements, which speak only as of the date hereof.

Overview

- Flash-extending in-memory computing applications
- Using a general purpose key-value library for flash-extension/flash-optimization
- Examples:
  - Memcached
  - Redis
  - GigaSpaces
  - MongoDB
  - Couchbase
  - Cassandra
- TCO
- Conclusion

Flash-Extending In-Memory Compute: Reduce TCO



	HDD	DRAM	SanDisk
Capacity	✓	✗	✓
Performance	✗	✓	✓

Typical performance results*

Servers needed for 3TB dataset

	HDD	DRAM	SanDisk
No. of servers	34	6	2
Power (kW)	12.7	2.8	0.8
$ per transaction	$8.44	$2.49	$1.02

* Based on internal SanDisk assumptions of representative performance, not actual performance

Flash-Extending In-Memory Apps

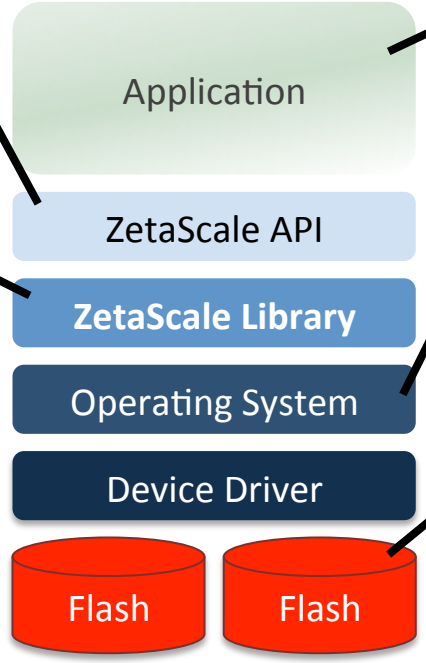
- **Flash-extending in-memory applications**
  - Exploit flash latency and IOPS
  - Requires extensive parallelism
  - Cache hot data in DRAM
  - Get "good-enough" performance at in-flash capacity and cost, enabling server consolidation
- **Key-value abstraction is a good semantic fit for extending many in-memory apps**
  - A good key-value storage engine can simplify flash-extension
  - Many applications manage data internally as objects
  - Need more than basic CRUD functionality: crash-safeness, transactions, snapshots, range queries
  - *Typical applications*: caching, databases, message queues, data grids
- ***Flash extending applications***: use key-value library to stage data between DRAM and flash
- ***Flash optimizing applications***: replace application storage engine with a more optimal key-value library
- **A good key-value library can dramatically reduce the work required to flash extend or flash optimize applications**

ZetaScale™ Software Crash-Safe Object Store

Features and Options

- Key/value paradigm
- C++ and Java interface
- Compiled into Application

- Containers
  Multiple namespaces offer file system type structure like folders and directories
  Hash table indexes provide fast range queries
- Transactions
  Guarantee that multiple data objects can be written **atomically**
- Snapshots
  Offer easy method to copy memory data to **persistent** storage
- Caching Layer
  Assures that frequently used data is readily accessed
- Dynamically loadable
- User callable



- Any user application, typically:
  - NoSQL database
  - In-Memory Compute application

- Operating Systems supported:
  - Linux Centos 6.5
  - Linux RHEL 6.5

- Flash vendor independent
  Works with flash from any brand and / or vendor
- Device interface independent
  Supports any flash device interface, including SAS, SATA, PCIe or NVMe

ZetaScale Flash-Optimized Applications

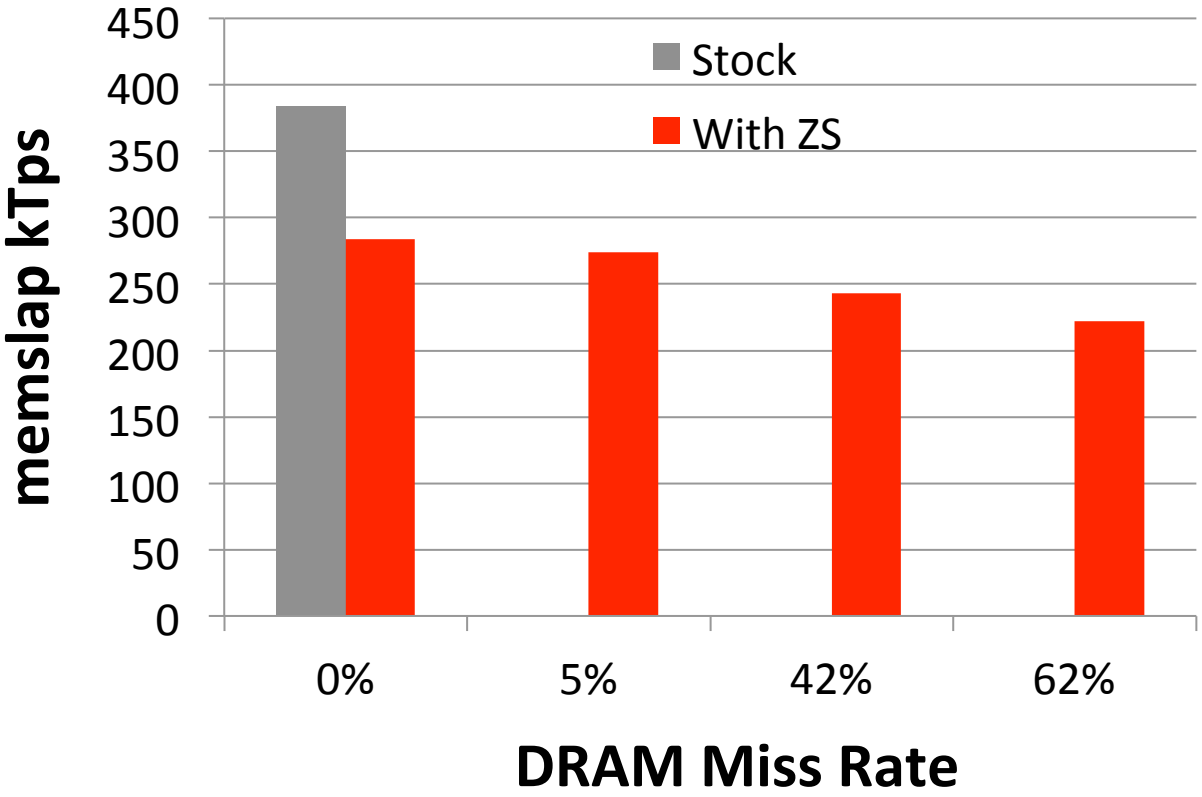
memcached

- Memcached is an open source, in-memory distributed key-value cache/store
  - CRUD API (create, replace, update, delete)
  - ASCII and Binary protocols
  - High performance
  - Written in C, clients available for most popular languages

- Test Hardware:
  - Dell R720 server: Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz. 2 physical CPUs - 8 cores/16 threads on each, visible as 32 CPUs, 128 GB DRAM, 10G ethernet
  - SSD: 400G * 8 x Lightning® SSD with md RAID 0
  - Remote client with 10G ethernet

- Test Software:
  - Memcached v1.4.15, CentOS release 6.5
  - ZetaScale™ software flash size: 500G
  - Memslap benchmark: 64 threads, 512 concurrency, 250 byte key, 1024 byte value, set/get = 1:9, 3600s run time

Memcached with ZetaScale Performance

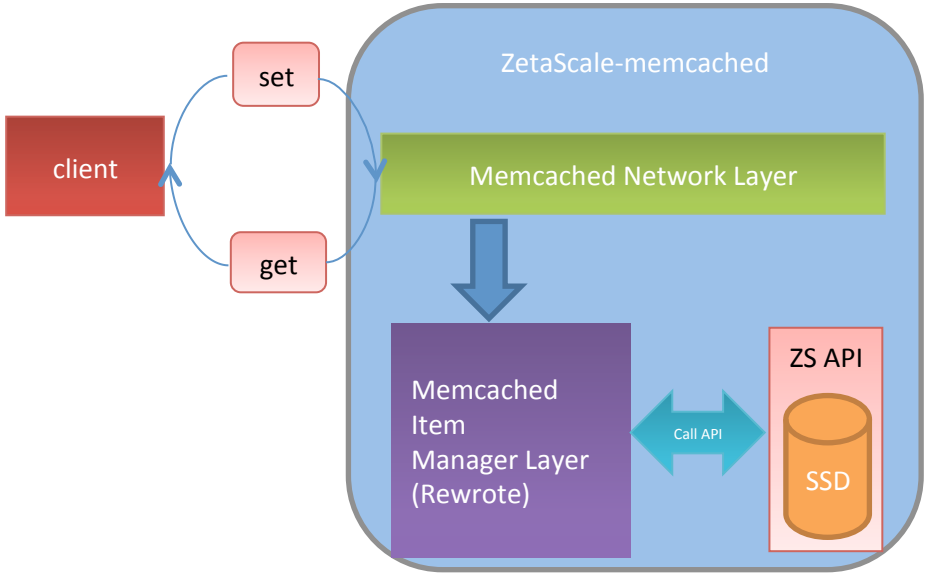


Memcached throughput with data set in Flash is similar to Stock-Memcached throughput with data set in DRAM

ZS = ZetaScale Software

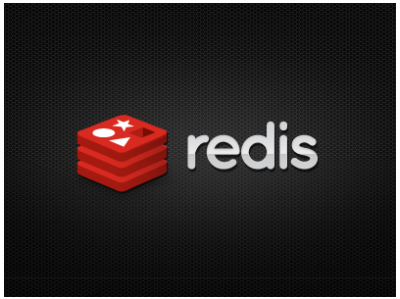
Source: Based on internal testing by SanDisk; Jan 2015

ZetaScale-Memcached Integration



- Replace memcached get/put routines with calls to ZetaScale get/put
- Use existing memcached multithreading to get sufficient parallelism to drive flash IOPS
- ZetaScale automatically caches hot objects in its own DRAM cache, so bypass stock memcached DRAM cache code

redis

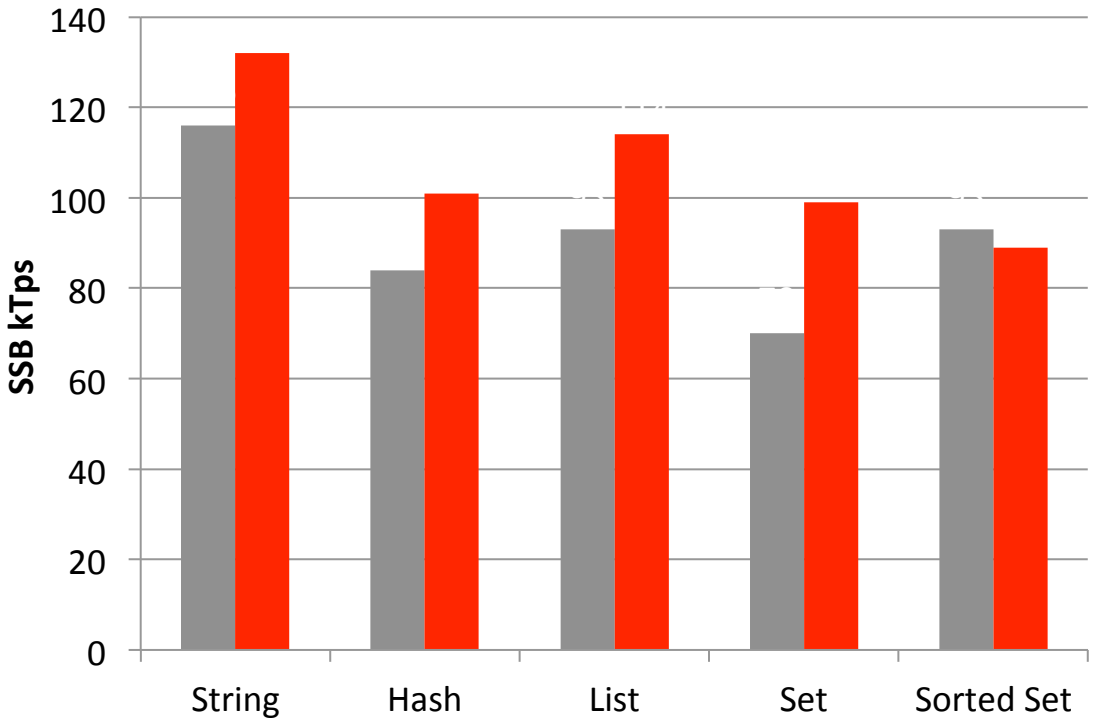


- Redis (REmote DIctionary Server) is an open-source, in-memory key-value
  - Supports more complex data types such as strings, hashes, lists, sets, sorted sets
  - asynchronous replication to 1 or more slaves
  - snapshot facility using fork() + copy-on-write
  - append-only logging with configurable fsync () policy
  - pub/sub capability

- Test Hardware:
  - HP Server: 2 x 6-core 2.90 GHz Intel Westmere; DRAM: 96G; Flash: 8 x 200G Lightning SSDs
  - Remote client with 10G network connection

- Test Software:
  - Redis 2.7.4
  - YCSB: uniform workload with 95% read and 5% update
  - Strings were 1K bytes; Hash, Lists, Sets and Sorted Sets were 10 x 100 bytes
  - Dataset used was 16 million objects for stock Redis and 64 million objects for Redis with ZetaScale

Redis with ZetaScale Performance



- Stock Redis (in memory)
- ZS+Redis (from flash, 4x larger dataset)

ZetaScale Software-Redis throughput with data set in Flash is similar to Stock-Redis throughput with data set in DRAM

ZS = ZetaScale Software

Source: Based on internal testing by SanDisk; Nov 2013

What Was Required to Exploit Flash?

- Replace Redis DRAM-to-storage staging code with calls to FDF get/put
- Convert Redis from single thread to multi-thread to drive flash IOPS

GigaSpaces

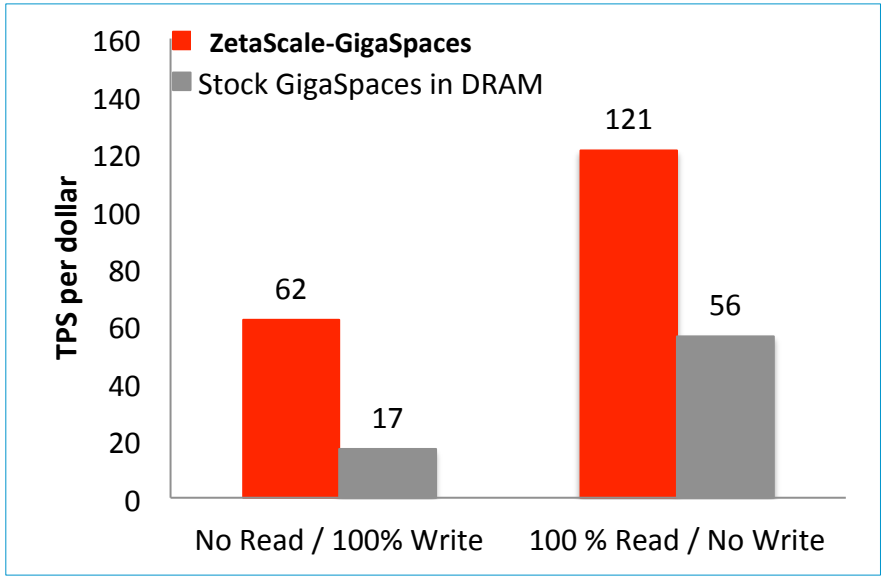
- GigaSpaces XAP is in-memory compute application platform design for real-time big data analytics applications
  - Leverages distributed real-time computation libraries such as Storm and Apache Samza to process unbounded streams of data
  - ZetaScale can manage large amount of data across a grid of high capacity servers
  - Can model the data using Objects/SQL, Documents or relational
  - Supports a variety of programing interfaces: Java, .Net , C++ , Scala

- Test Hardware:
  - 2 sockets 2.8GHz CPU with total 12 cores, 148G DRAM
  - Fusion ioMemory™ ioDrive® Duo PCIe card with md RAID 0

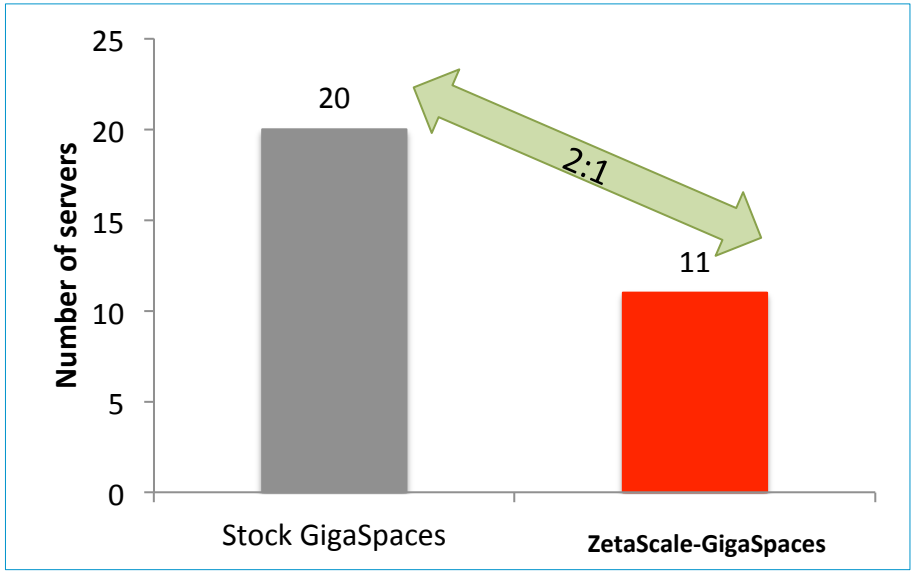
- Test Software:
  - Gigaspaces-10.0.0-XAP Premium-m2
  - CentOS 5.8
  - GS-provided YCSB client
  - 1KB object size and uniform distribution

GigaSpaces/ZetaScale XAP MemoryXtend

Provides 2x – 3.6x Better TPS/$



While Reducing Servers by 50%



- 1KB object size and uniform distribution
- 2 sockets 2.8GHz CPU with total 24 cores, CentOS 5.8, Fusion ioMemory ioDrive Duo PCIe card, md RAID 0
- YCSB measurements performed by SanDisk; cost calculations by GigaSpaces

Source: Based on internal testing by SanDisk

What Was Required to Exploit Flash?

- Stage objects in and out of DRAM using ZetaScale via existing “Off-Heap” interface:
  - GS put calls ZSWriteObject() API
  - GS replace calls ZSWriteObject() API
  - GS get calls ZSReadObject() API
  - GS remove calls ZSDeleteObject() API

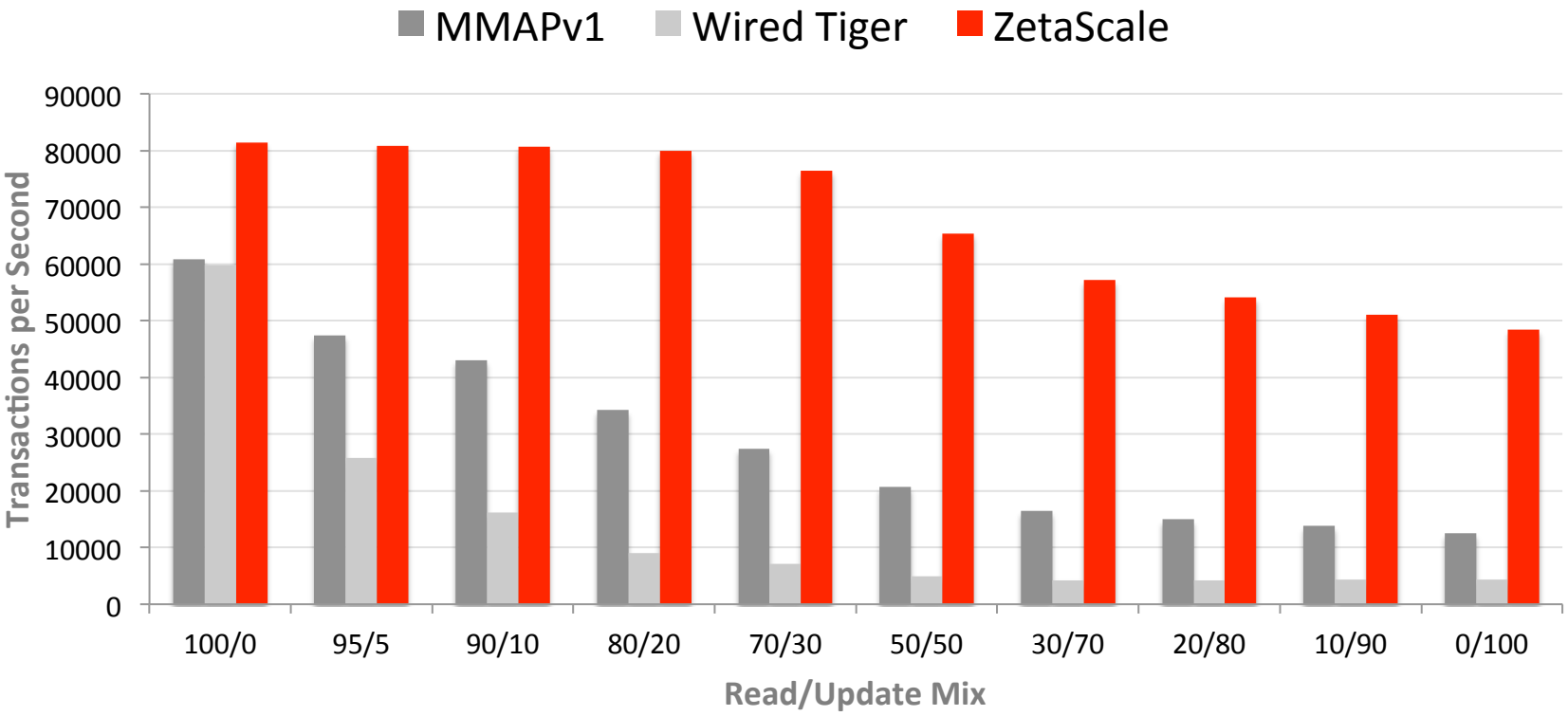
MongoDB

- MongoDB (from “humongous”) is an open source NoSQL document store
  - JSON-style documents
  - Built-in sharding across multiple nodes
  - Automatic resharding when adding or deleting nodes
  - Rich, document-based queries
  - Supports multiple indices

- Test Hardware:
  - 2 x 8-core 2.6 GHz Intel Xeon; 64G DRAM; 8 x 200G Lightning SSDs
  - client co-resident on server

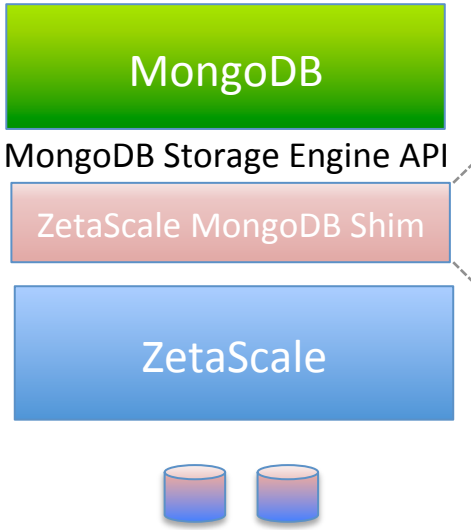
- Test Software:
  - CentOS 6.6; MongoDB 3.0.1
  - YCSB: point read, update and insert; 1K objects; 15 minute runs
  - For Read/Update: 128G dataset contained 128 million 1K objects

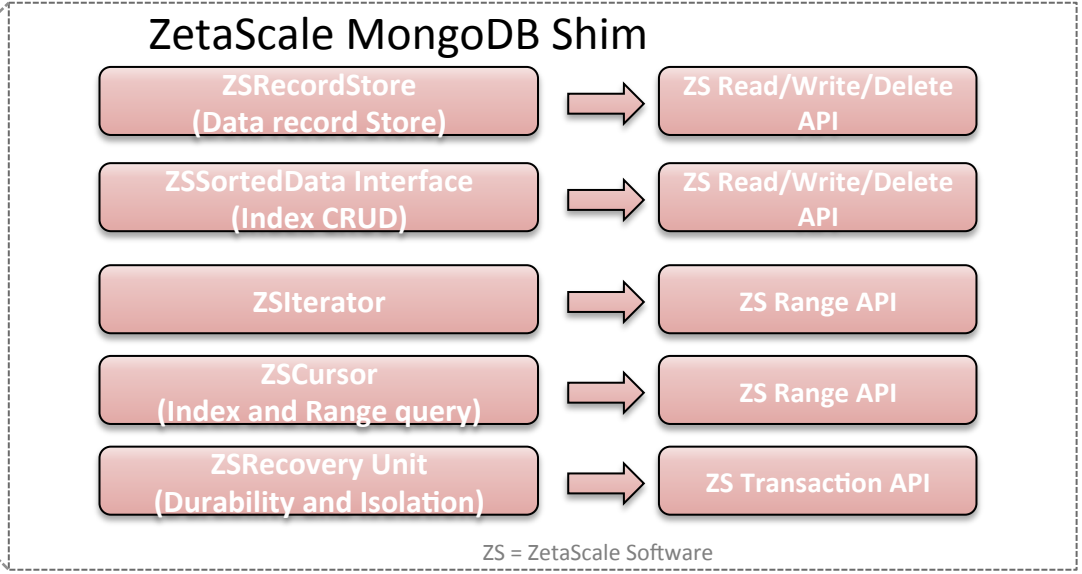
MongoDB with ZetaScale: Read/Update 128G



Source: Based on internal testing by SanDisk; Apr/May 2015

MongoDB ZetaScale Integration





- MongoDB collection and indexes map to one or more ZetaScale Btree containers.
- Record location is identified by unique auto generated ID
- Secondary indexes record location as value

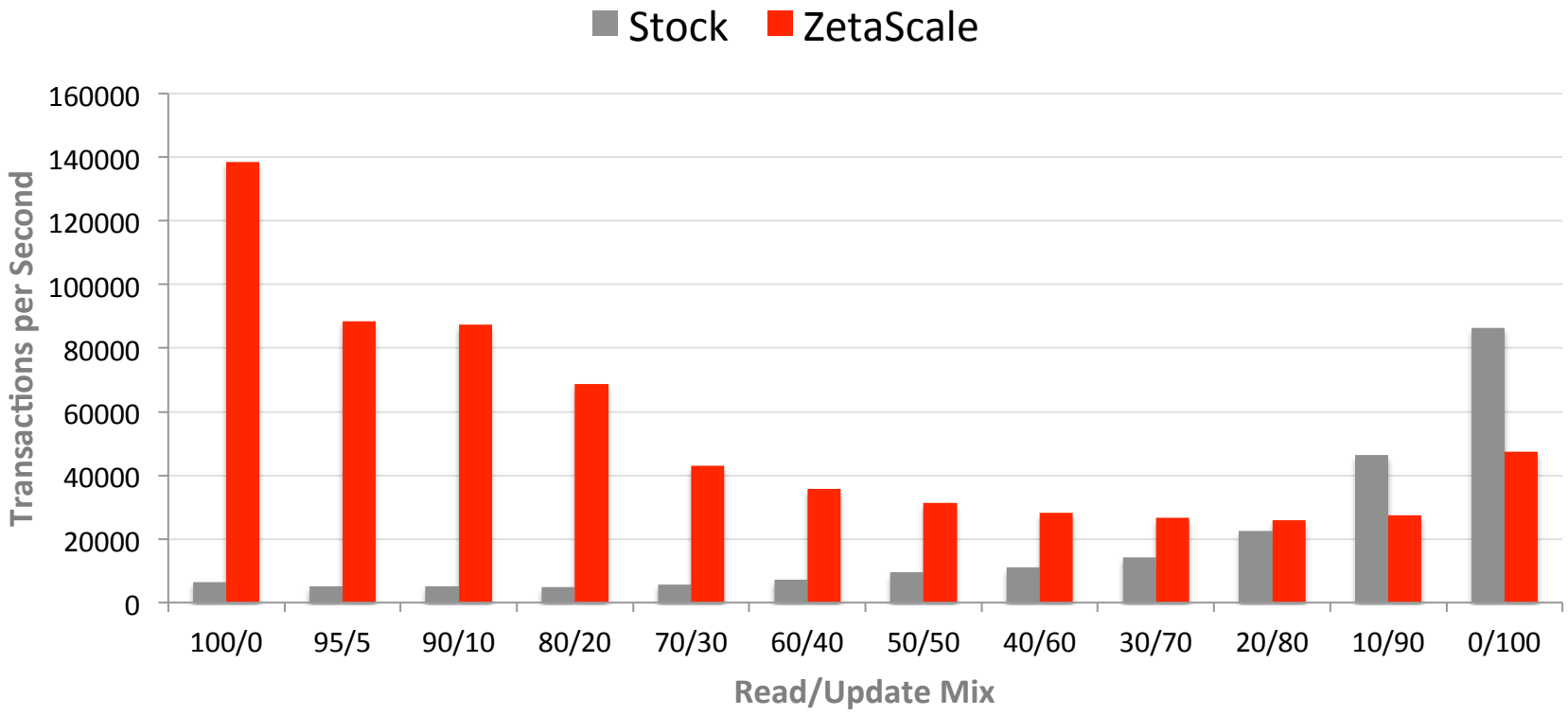
Couchbase

- Couchbase Server is an open-source NoSQL distributed database with a flexible data model
  - Integrated object caching via memcached
  - On-demand elastic scalability
  - Supports binary and JSON data types
  - Supports indexes on JSON fields
  - Inter and intra data center replication

- Test Hardware:
  - 2 x 8-core 2.60 GHz Intel Xeon E5-2670; 64G DRAM; 8 x 200G Lightning SSDs
  - remote client: 8 core 2.53 GHz Intel Xeon E5540; 64G DRAM, 10G ethernet, Oracle Linux 6.3

- Test Software:
  - CentOS 6.5; Couchbase 3.03
  - Stock Couchbase: 48GB DRAM; Threads: 24 frontend (FE), 4 backend read (BR), 4 backend write (BW)
  - ZetaScale Couchbase: Couchbase: 8GB DRAM, ZetaScale: 40GB DRAM; Threads: 64 FE, 4 BR, 32 BW
  - YCSB; 24M 1K objects for in-memory test; 128M 1K objects for Flash test; 128 threads

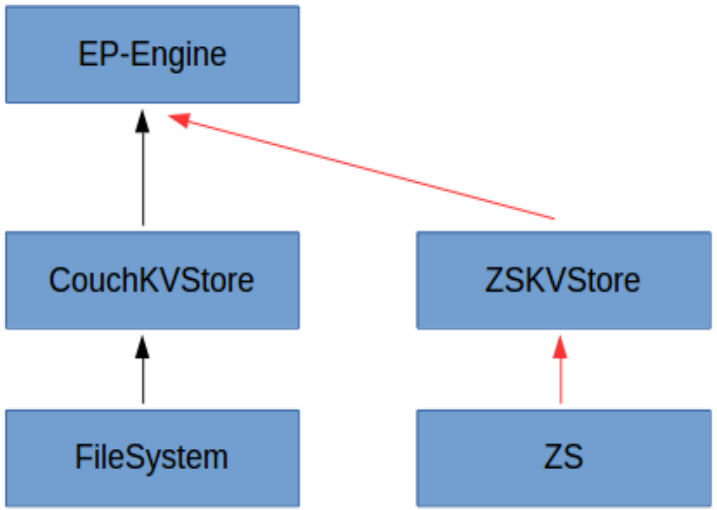
Couchbase with ZetaScale

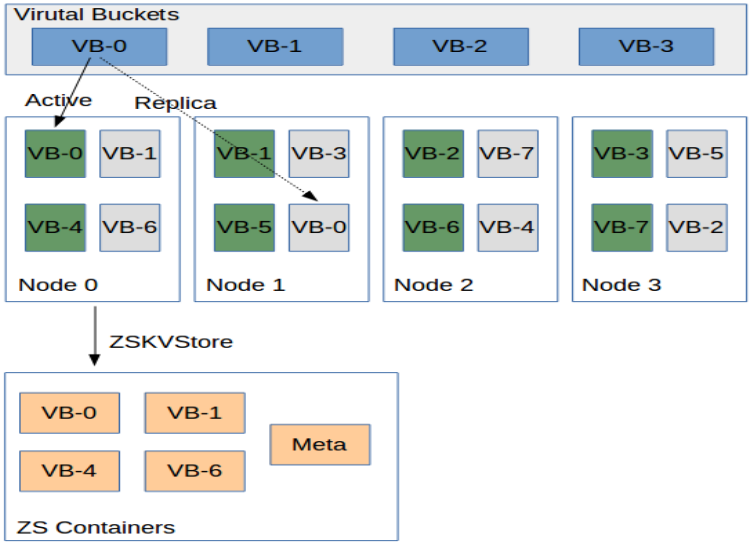


Source: Based on internal testing by SanDisk; Apr 2015

ZetaScale Couchbase Integration Highlights

- Replace CouchKVstore to ZetaScale KV storage engine
- Couchbase VBs are mapping to ZetaScale Containers





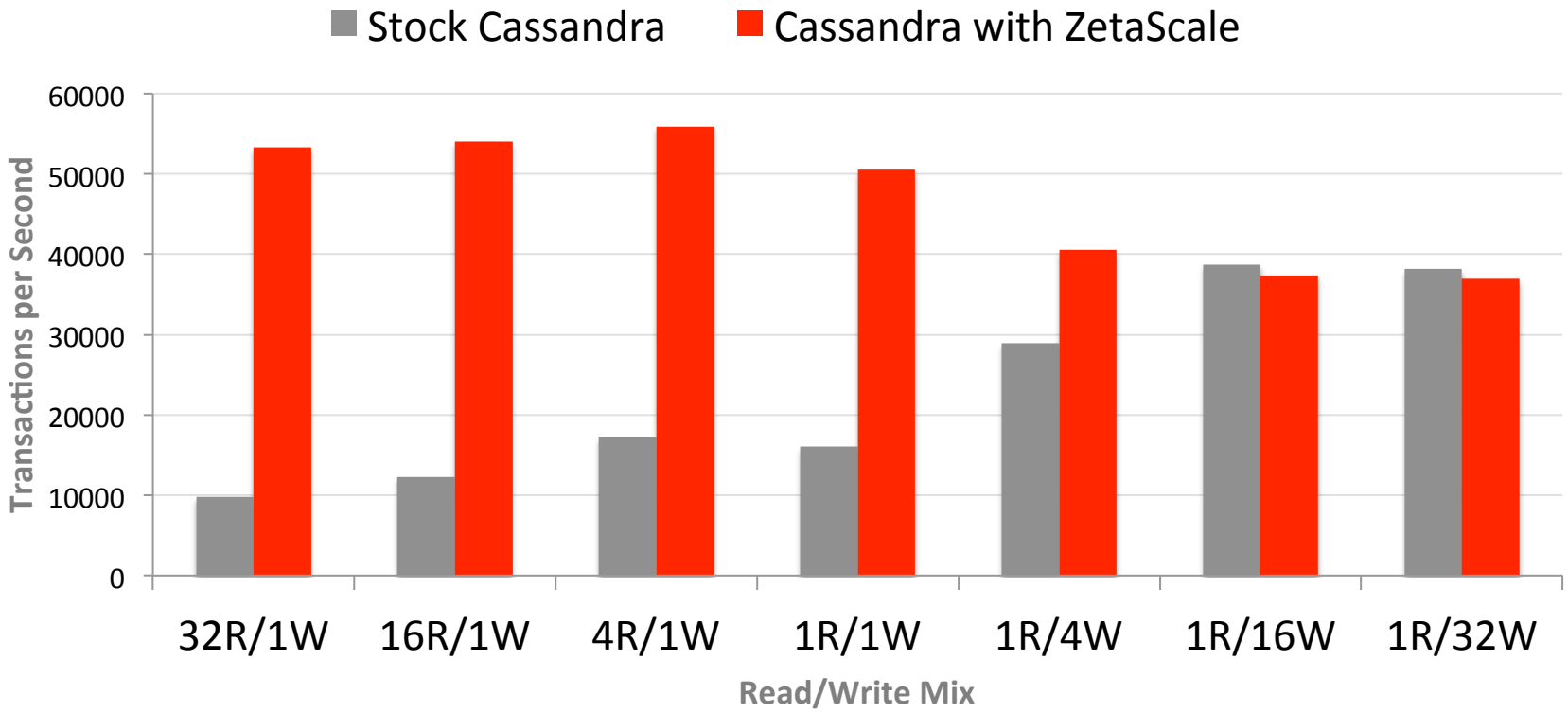
Cassandra

- Cassandra is an open source distributed key-value store
  - large scale synchronous/asynchronous replication
  - automatic fault-tolerance and scaling
  - tunable consistency efficient support for large rows (1000's of columns)
  - CQL (SQL-like) query language
  - supports multiple indices
  - Optimized for high write workloads

- Test Hardware:
  - Dell R720: 2 x 8-core Intel 2.60GHz CPU; DRAM: 128G; Flash: 8 x Lightning SSDs; Controller: LSI 9207 HBA
  - remote client with 2 x 8-core Intel Xeon CPU, 10G ethernet

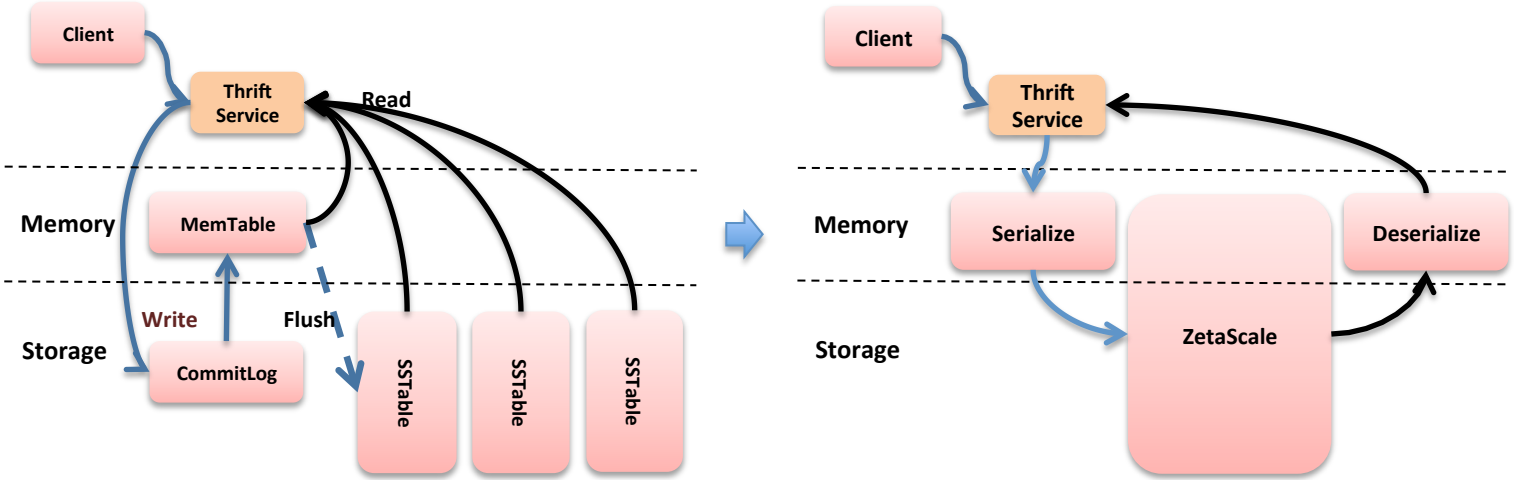
- Test Software:
  - Cassandra v2.0.3: stock and with ZetaScale
  - Datastax modified Cassandra-stress tool
  - 60M rows, 5 columns per row, 100 byte object size; 128 threads

Cassandra with ZetaScale



Source: Based on internal testing by SanDisk; Mar 2014

ZetaScale Cassandra Integration Highlights



- Replace Cassandra LSMTree (memtable & sstables) with ZetaScale storage engine
- Route object get/put calls to ZetaScale get/put
- Disable stock Cassandra journal: ZetaScale maintains its own journal
- ZetaScale indexing is used for row and column range queries
- ZetaScale transactions are used to enforce atomicity of row updates and secondary index modifications
- ZetaScale snapshots are used for full and incremental backups
- Compaction is eliminated!

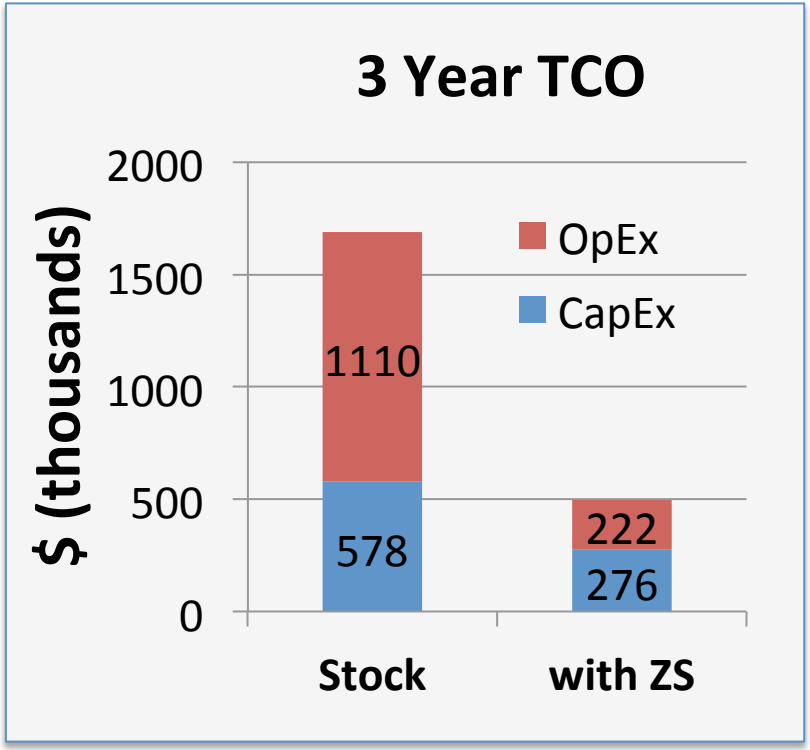
Flash Extension and TCO

Parameters		
Starting nr of nodes	100	Nodes
Storage per node	2.0	TB
Consolidation Rate	5	

	Stock SSD	ZetaScale SSD	Diff
TPS Consolidation	5	1	
Server Cost			
Base Unit + I/O	$ 1,700.00	$ 1,700.00	
CPUs	$ 500.00	$ 500.00	
DRAM	$ 1,206.00	$ 1,206.00	
SSD	$ 2,000.00	$ 10,000.00	same amount of total storage
Build Cost	$ 5,406.00	$ 13,406.00	
Markup	1.00	1.00	
ASP	$ 5,406.00	$ 13,406.00	
Capacity			
DRAM size (GB)	96	96	
Storage capacity (RAID)	2000	10000	
Power per Node			
Base Unit + I/O	100	100	
CPUs	120	120	
DRAM	48	96	
SSD/HDD	15	25	
Total watts per Server	283	341	
Total Deployment	**SSD**	**ZetaScale**	
Nodes	100	20	
Server CapEx	$ 540,600	$ 268,120	
Number of racks	20	4	
Rack Install	$ 20,000	$ 4,000	
Rack per month	$ 17,000	$ 3,400	
Power install	$ 7,000	$ 1,400	
Power per month	$ 6,900	$ 1,380	
Pipe install	$ 10,000	$ 2,000	
Pipe per month	$ 7,000	$ 1,400	
Monthly OpEx	$ 30,900	$ 6,180	
			Saving
CapEx	$ 577,600	$ 275,520	52%
3Yr OpEx	$ 1,112,400	$ 222,480	80%
TCO	$ 1,690,000	$ 498,000	71%

CapEx Assumptions		
Platform	Build Cost	Power
E5-2650L	$ 1,000	60
720 Chassis + 10GBE	$ 1,700	100
E5-2650L	$ 1,000	60
Memory DIMM	Cost	Power
8GB	$ 110	2
16GB	$ 201	4
32GB	$ 839	8
SSD	Cost per GB	Power
Consumer 1GB	$ 0.50	5
SAS / SATA 1GB	$ 1.00	5
PCIe 1GB	$ 1.50	5

OpEx Assumptions			
Hosted Data Center	Stock SSD	ZetaScale SSD	
Rack Loading	5	5	Servers/rack
Rack Install	$1,000	$1,000	Per rack
Rack per month	$850	$850	Per rack
Power install	$350	$350	Per rack
Power per month	$345	$345	Per rack
Pipe install	$500	$500	Per rack
Pipe per month	$350	$350	Per rack



Conclusion

- In-memory compute applications can use flash to extend capacity and still maintain good performance, leading to reduced TCO
- Key-value abstraction is a good semantic fit for extending many in-memory apps
- Need sufficient functionality: crash-safeness, transactions, snapshots, range queries
- Proof points using the ZetaScale key-value library: Memcached, Redis, GigaSpaces, MongoDB, Couchbase, Cassandra
- Proof points show that although performance drops using flash extension, it is still good
- For capacity limited applications, flash extension can reduce overall TCO significantly

SanDisk®

Thank You!

brian.okrafka@sandisk.com

@BigDataFlash
#bigdataflash
ITblog.sandisk.com
http://bigdataflash.sandisk.com