Flash Data Fabric:
A Substrate for Flash Optimizing Applications

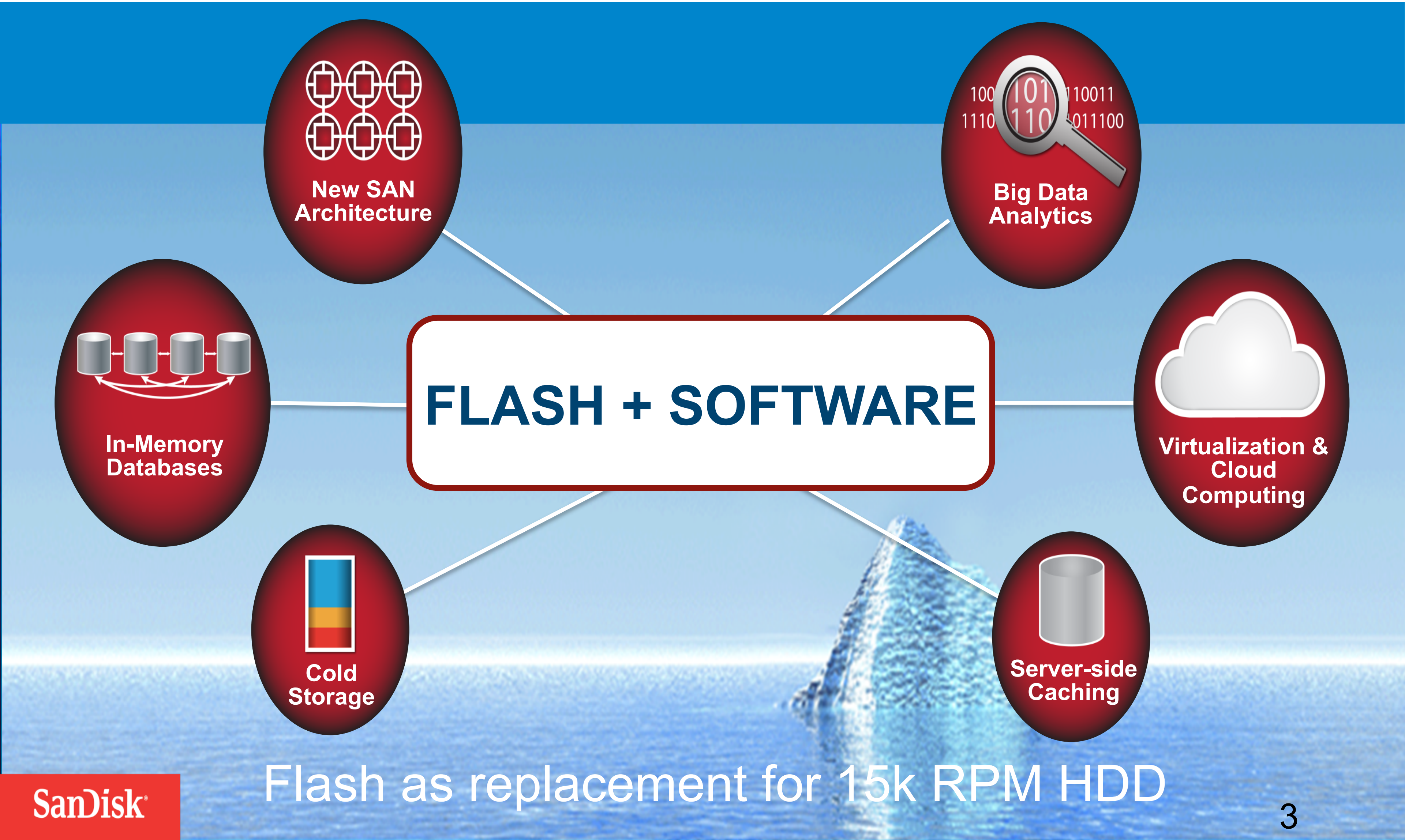
Brian O’Krafka, Fellow

August 15, 2013

Overview

- Flash Optimization: Why and How
- Some Examples:
  - In-Memory Data Grids
  - In-Memory Databases
  - NoSQL Databases
- Conclusion

Software Unlocks Flash Potential in the Enterprise



Flash Optimization: Why and How

Flash-Optimized Applications and FDF

- Flash-optimized applications:
  - Exploit the high capacity, low latency, persistence and high throughput of flash memory
  - Have extensive parallelism to enable many concurrent flash accesses for high throughput
  - Use DRAM as a cache for hot data
  - Get in-DRAM performance at in-flash capacity and cost, enabling server consolidation
- SanDisk Flash Data Fabric (FDF) is a substrate for flash-optimized applications
  - Caching, key-value stores, databases, message queues, custom apps
  - Leverages flash for high performance, high availability
  - Enables low TCO through high server consolidation
  - Executes on bare metal or virtualized
- Many applications realize limited benefits from flash without system level optimization
  - FDF incorporates the flash optimizations required to fully exploit flash
  - Applications can be fully flash-optimized using FDF
- FDF incorporates:
  - Intelligent granular DRAM caching
  - Heavily optimized access paths for high performance
  - Optimized threading to maximize concurrency and minimize response time
  - Configurable flash management algorithms to optimize different workloads

In-Memory Data Grids

Example 1: Memcached

- Memcached is an open-source distributed key-value memory caching system
- Originally developed by Danga Interactive for LiveJournal
- Commonly used to reduce load on databases. Applications typically:
  - look for data in memcached
  - if not found, access the database and insert into memcached
  - memcached uses LRU replacement to make room for new objects
- Memcached service offered by leading cloud service providers
- Provides a basic “CRUD” key-value interface (Create, Replace, Update, Delete)
- FDF-Memcached based on Memcached version 1.4.15
- Compare against Couchbase, an open source version of memcached supporting persistence on flash

Memcached/FDF Performance

memslap 90% Read, 10% Write

Configuration	TPS	FDF Cache Miss Rate	CPU Utilization	Flash Utilization
FDF-Memcached	285K	10%	8/24	90%
Couchbase 2.0.1	35K	N/A	N/A	10%

- Intel Westmere server with 2 x 2.9GHz sockets, 24 cores, 96G DRAM
- SSD: 8 x 200G SSD with software RAID 0
- Memslap Benchmark set-up
  - Remote client with 10G network connection
  - 1K fixed object, uniform distribution with configurable read/write mix (eg: 90% read, 10 % update)
- 20GB FDF DRAM cache

In Memory Databases

Example 2: Redis

- Redis (REmote DIctionary Server) is an open-source, in-memory key-value store with some persistence capabilities

- Supports more complex data types:
  - strings, hashes, lists, sets, sorted sets

- Additional features beyond memcached:
  - asynchronous replication to 1 or more slaves
  - snapshot facility using fork() + copy-on-write
  - append-only logging with configurable fsync() policy
  - pub/sub capability

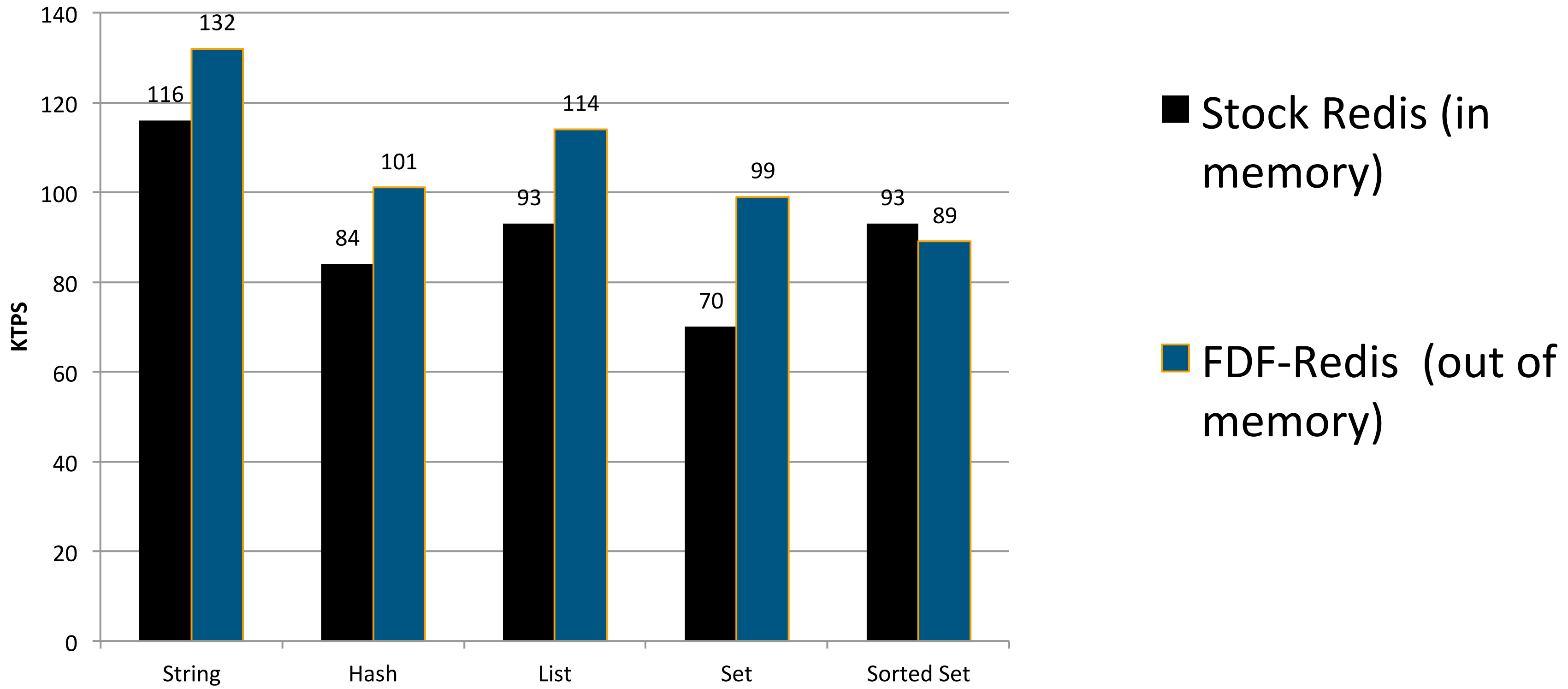
- Single-threaded

- FDF-Redis prototype based on Redis 2.7.4

Redis Benchmark Environment

- "Bare Metal":
  - Intel Westmere server with 2 x 2.9GHz sockets, 24 cores, 96G DRAM
  - SSD: 8 x 200G SSD with software RAID 0
- AWS:
  - 16 Core CPU, 64G Memory AWS CentOS, SSD enabled instance
- YCSB Benchmark set-up:
  - Bare Metal: Remote client with 10G network connection
  - AWS: client on same instance as server (to avoid network bottleneck)
  - 1K fixed object, uniform distribution with configurable read/write mix (eg: 95% read, 5 % update)
- For Redis:
  - FDF-Redis: 32 threads, 32G Redis cache and 4G FDF cache
  - Hash, list, set and sorted set use 10 x 100 byte fields as object

FDF-Redis Performance ("Bare Metal")

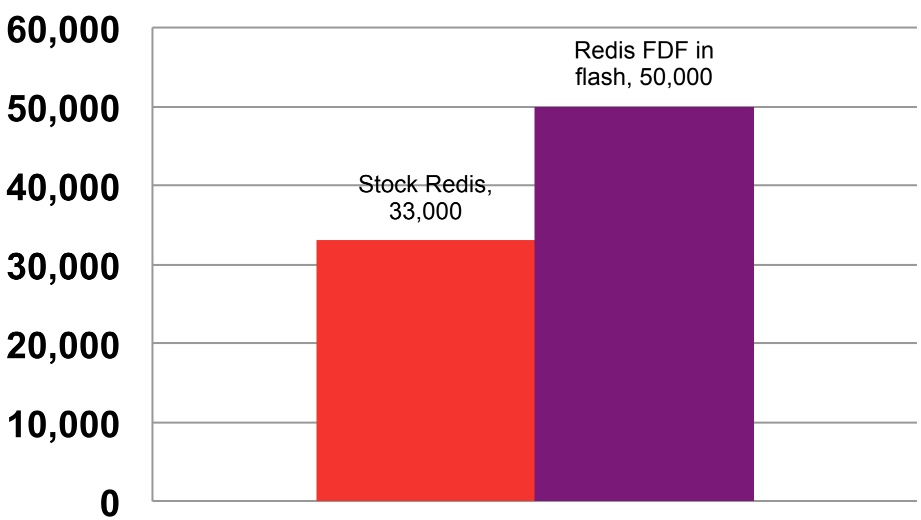


FDF-Redis Throughput with Data Set in Flash matches Stock -Redis throughput with data set in DRAM

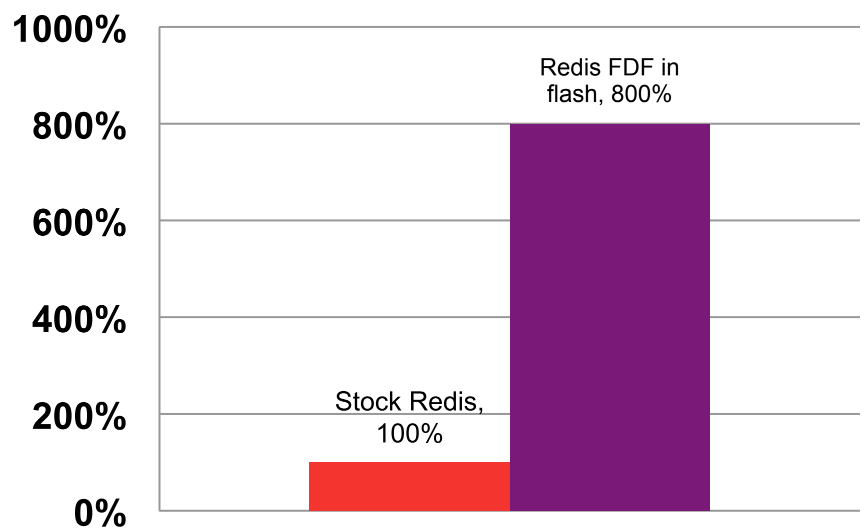
SanDisk

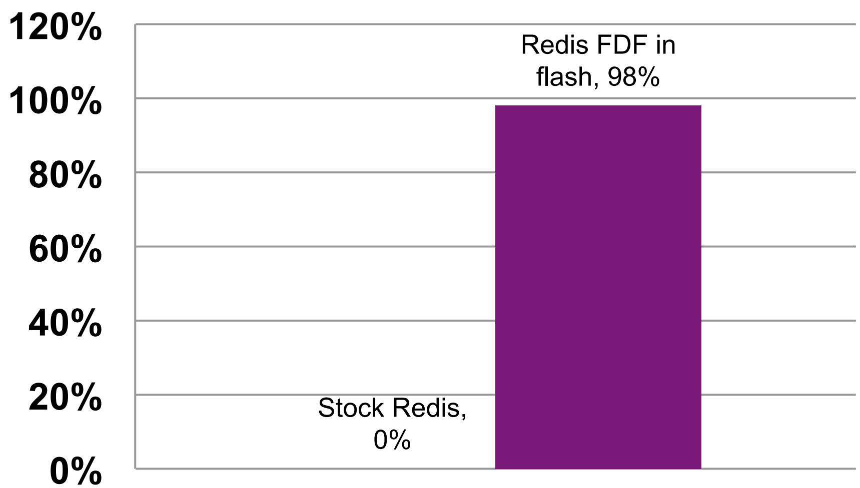
Redis Results on AWS (Using "Hash" Data Structure with Stock Redis data in DRAM FDF-Redis data in Flash)

TPS



CPU Utilization

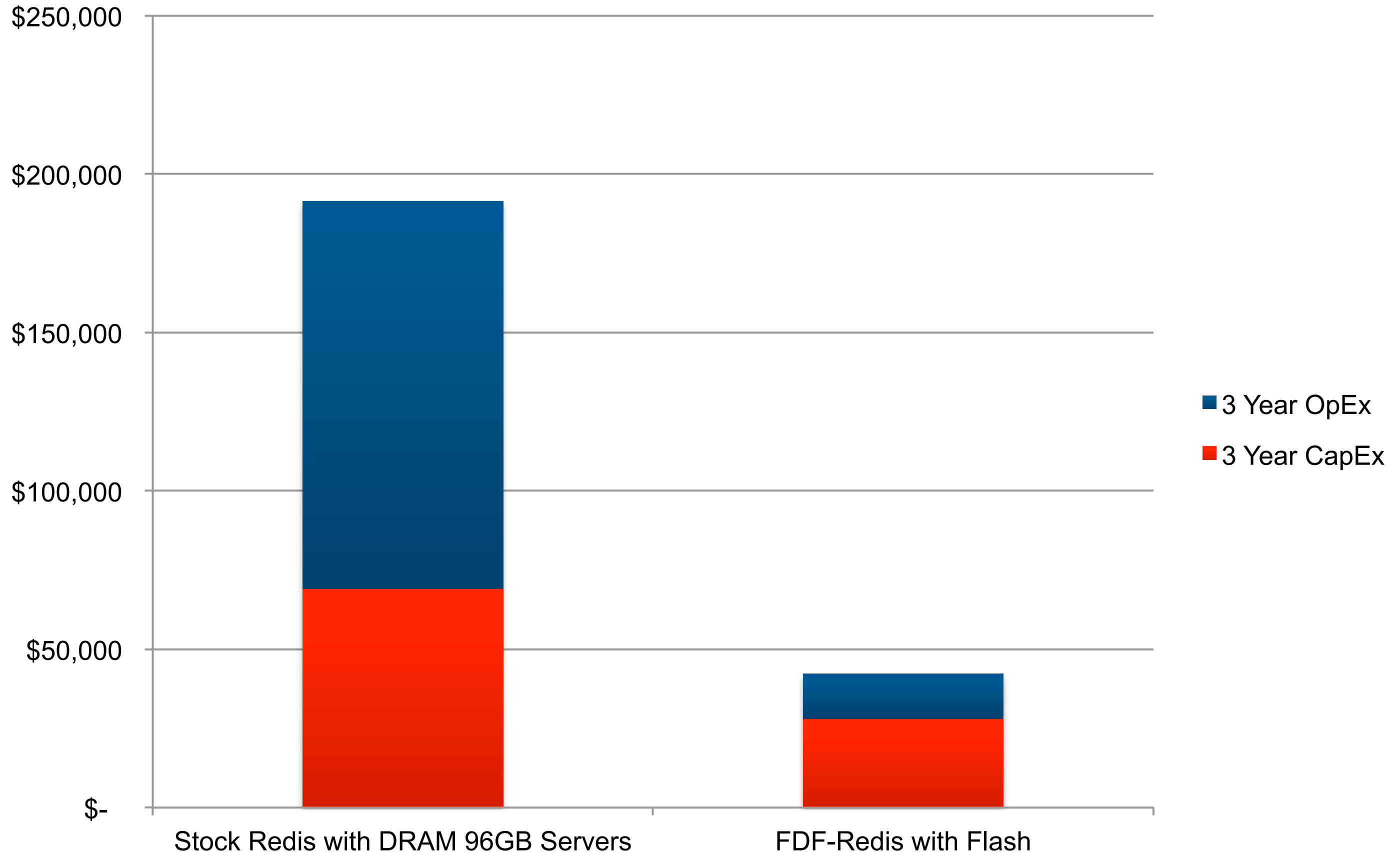




Flash Utilization

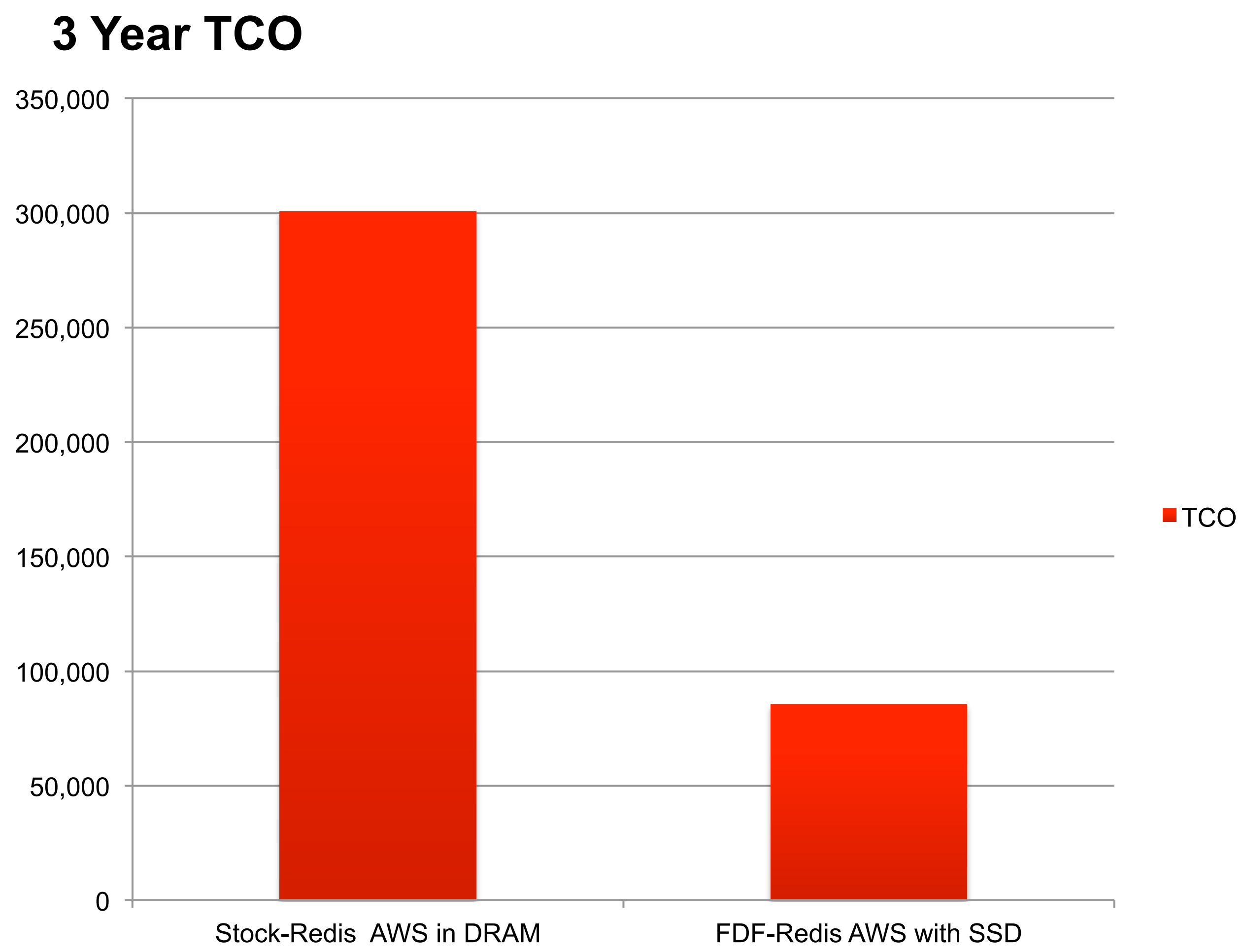
TCO : Stock Redis vs FDF-Redis (bare metal)

requirement : 80k TPS and 1 TByte data set



TCO : Stock Redis vs FDF-Redis (AWS)

requirement : 80k TPS and 1 TByte data set



NoSQL Databases

Example 3: Cassandra

- Cassandra is an open source distributed key-value store
- Key features:
  - support for large scale synchronous and asynchronous replication, including across data centers
  - automatic fault-tolerance and scaling
  - tunable consistency (from “writes never fail” to “block for all replicas to be readable”)
  - efficient support for large rows (1000’s of columns)
  - CQL (SQL-like) query language
  - supports multiple indices
- Optimized for high write workloads
- FDF-Cassandra prototype based on Cassandra 2.1.4

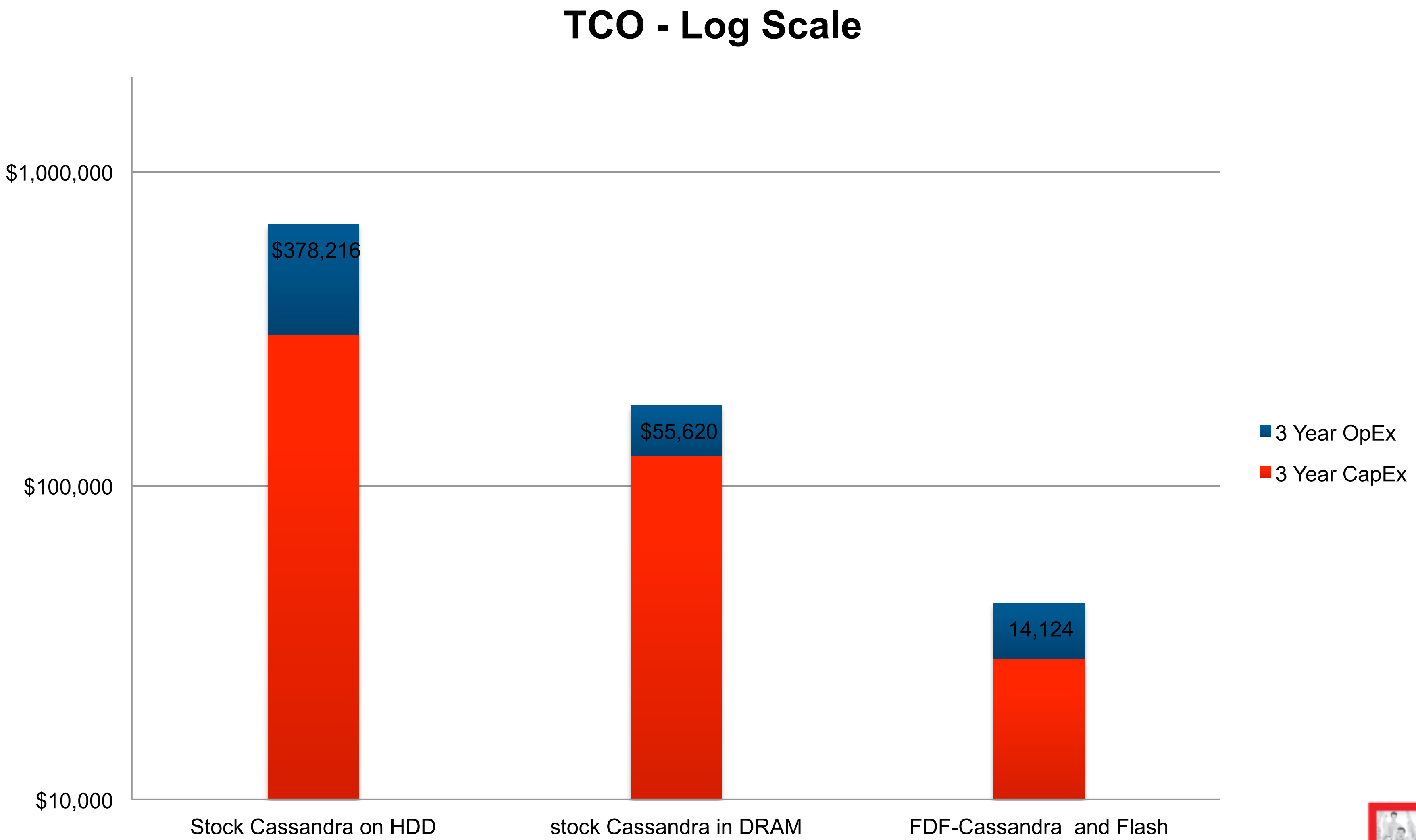
Cassandra Performance

95/5 workload	Stock Cassandra	FDF Cassandra
Hard Drives	1.2k tps 100% HDD utilization 1 of 16 cores utilization	N/A
64GB Data (fits in memory)	40K tps 12 of 24 cores utilization	124K tps 18 of 24 cores utilization
256GB Data (data set in flash)	25K tps 90% flash utilization 18 of 24 cores utilization	95K tps 90% flash utilization 19 of 24 cores utilization

- Intel Westmere server with 2 x 2.9GHz sockets, 24 cores, 96G DRAM
- SSD: 8 x 200G SSD with software RAID 0
- YCSB Benchmark set-up
  - Remote client with 10G network connection
  - 1K fixed object, uniform distribution with configurable read/write mix (eg: 95% read, 5 % update)
- 48GB FDF DRAM cache

TCO : Cassandra

requirement : 80k TPS and 1 TByte data set



Conclusion

Conclusion

- Many applications realize limited benefits from flash without optimization

- Flash optimization of applications can yield near in-DRAM performance with the datasets spilling into flash

- Critical flash optimizations include:
  - Intelligent granular DRAM caching
  - Heavily optimized access paths for high performance
  - Optimized threading to maximize concurrency and minimize response time
  - Granular locking for high concurrency

- Flash optimizations have been encapsulated in the SanDisk Flash Data Fabric (FDF): a substrate for flash-optimized applications
  - Typical applications: caching, key-value stores, databases, message queues, custom apps
  - Leverages flash for high performance, high availability
  - Enables low TCO through high server consolidation
  - Proof points: memcached, redis, cassandra

Complete Top to Bottom Capabilities

Performance	Scalability	System Utilization	Endurance	Cost	Life Cycle

Full Stack Enables Segment Optimized Solutions

NAND TECH | NAND DIE | WAFER | SCALE | MFG | CONTROLLER | SSD | SOFTWARE

SanDisk

SanDisk®

Thank you!