ACCELERATE YOUR SPARK WITH INTEL OPTANE DC PERSISTENT MEMORY

INTEL SSP

Notices and Disclaimers

All products, computer systems, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

The cost reduction scenarios described are intended to enable you to get a better understanding of how the purchase of a given Intel based product, combined with a number of situation-specific variables, might affect future costs and savings. Circumstances will vary and there may be unaccounted-for costs related to the use and deployment of a given product. Nothing in this document should be interpreted as either a promise of or contract for a given level of costs or cost reduction.

The benchmark results reported above may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations.

Results have been estimated based on tests conducted on pre-production systems, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance results are based on testing as of 03-14-2019 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to www.intel.com/benchmarks.

Intel processors of the same SKU may vary in frequency or power as a result of natural variability in the production process.

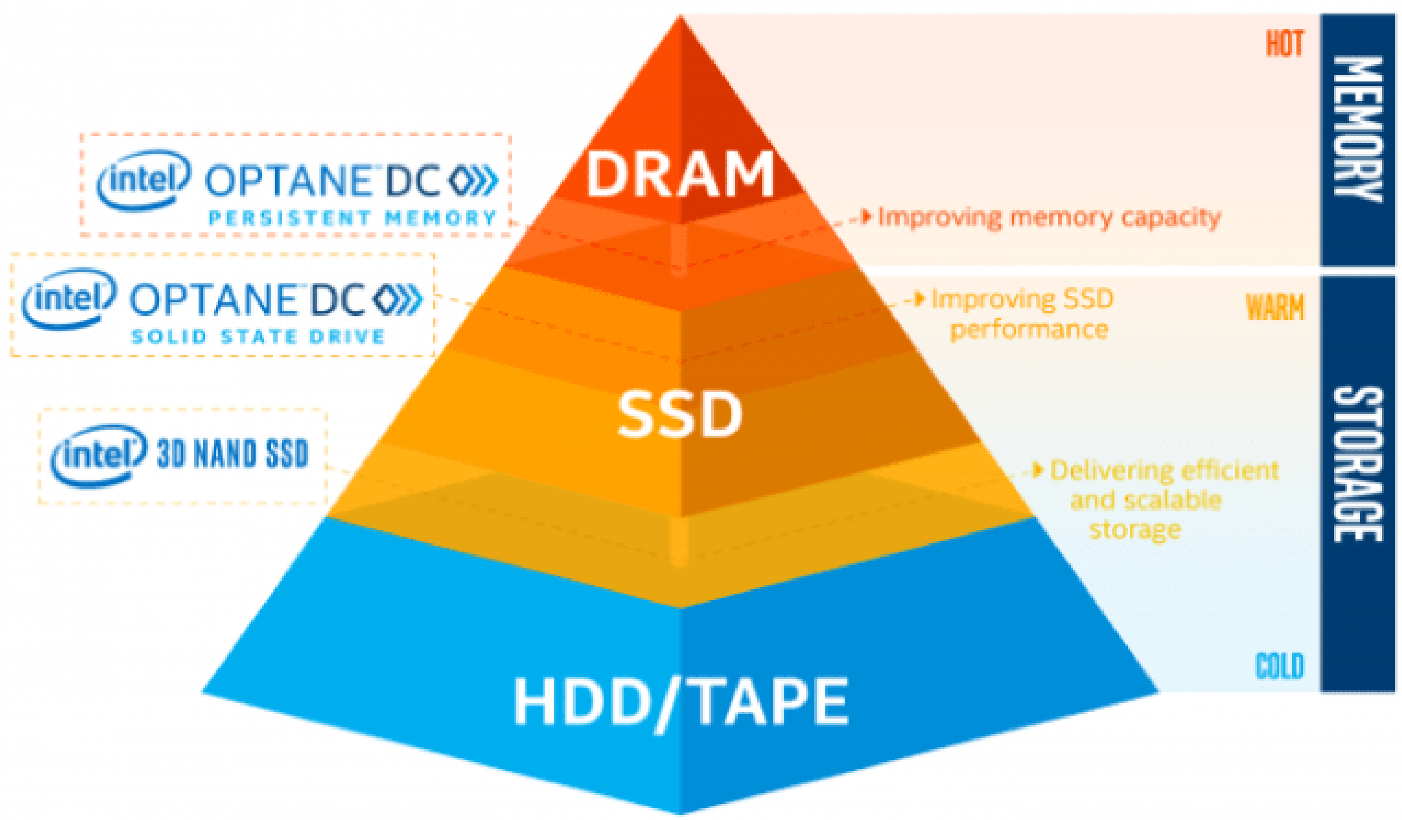
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.
Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Agenda

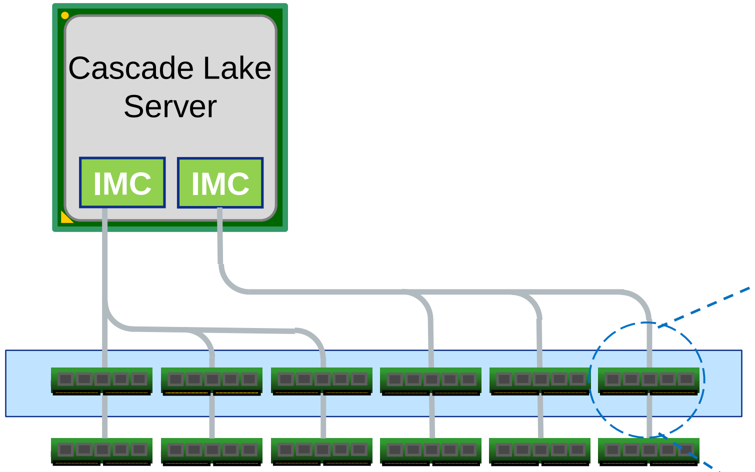
- DCPMM Introduction
- DCPMM on Spark
  - Spark SQL
  - Machine Learning - Kmeans

Re-architecting the Memory/Storage Hierarchy

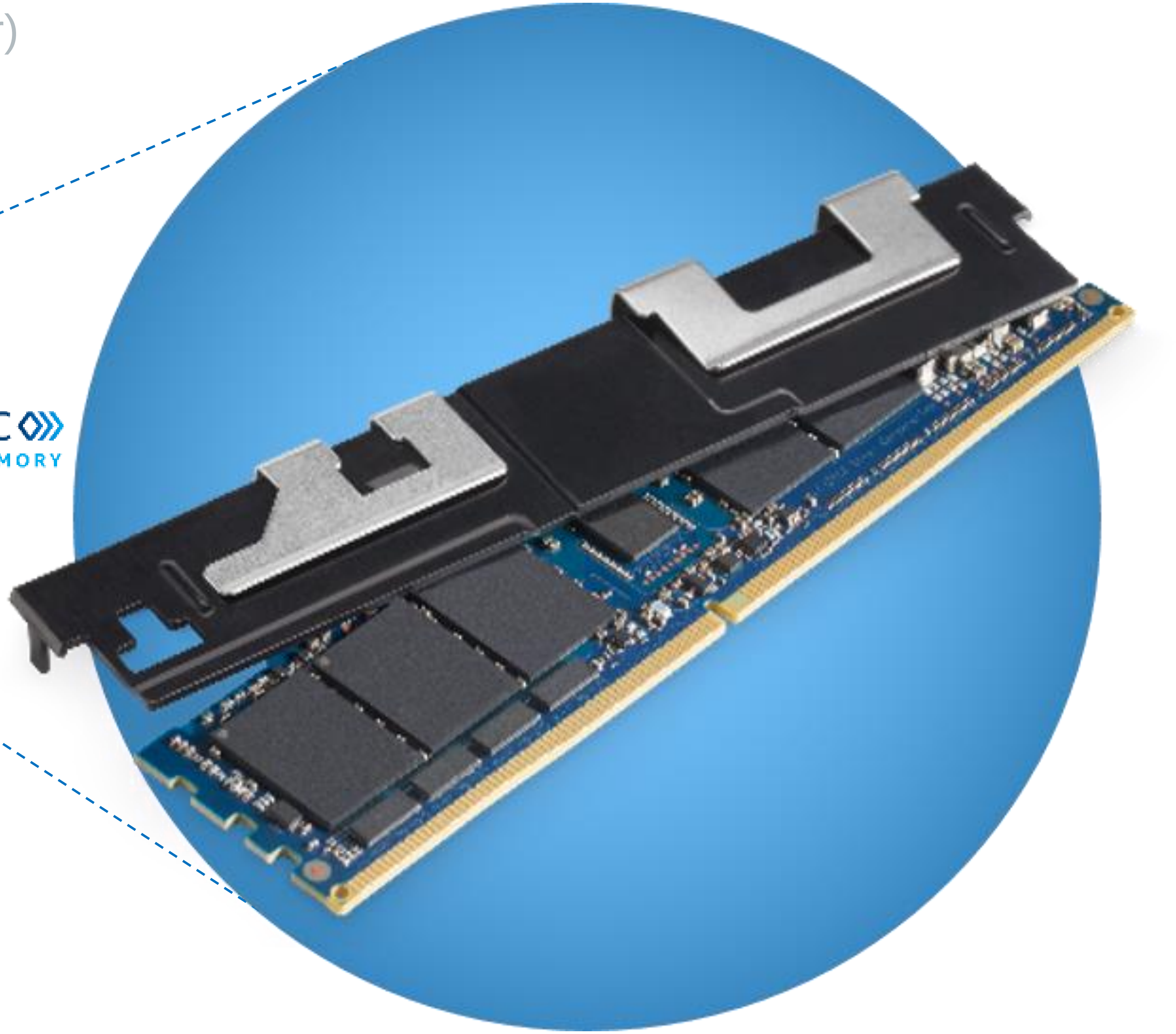


INTEL® OPTANE™ DC PERSISTENT MEMORY - PRODUCT OVERVIEW

(Optane™ based Memory Module for the Data Center)



* DIMM population shown as an example only.



DIMM Capacity

- 128, 256, 512GB

Speed

- 2666 MT/sec

Capacity per CPU

- 3TB (not including DRAM)

- DDR4 electrical & physical
- Close to DRAM latency
- Cache line size access

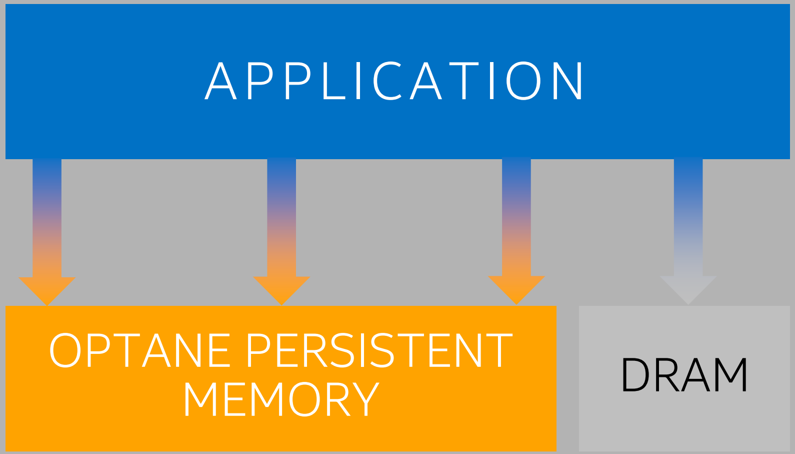
PERSISTENT MEMORY OPERATING MODES

Memory Mode & AppDirect

INTEL® OPTANE™ DC PERSISTENT MEMORY SUPPORT FOR BREADTH OF APPLICATIONS

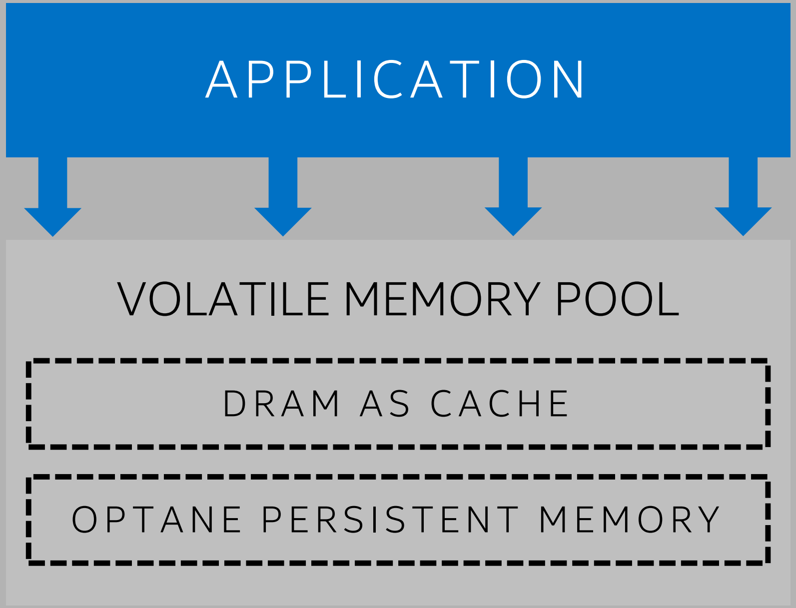
APP DIRECT MODE

PERSISTENT PERFORMANCE
& MAXIMUM CAPACITY



MEMORY MODE

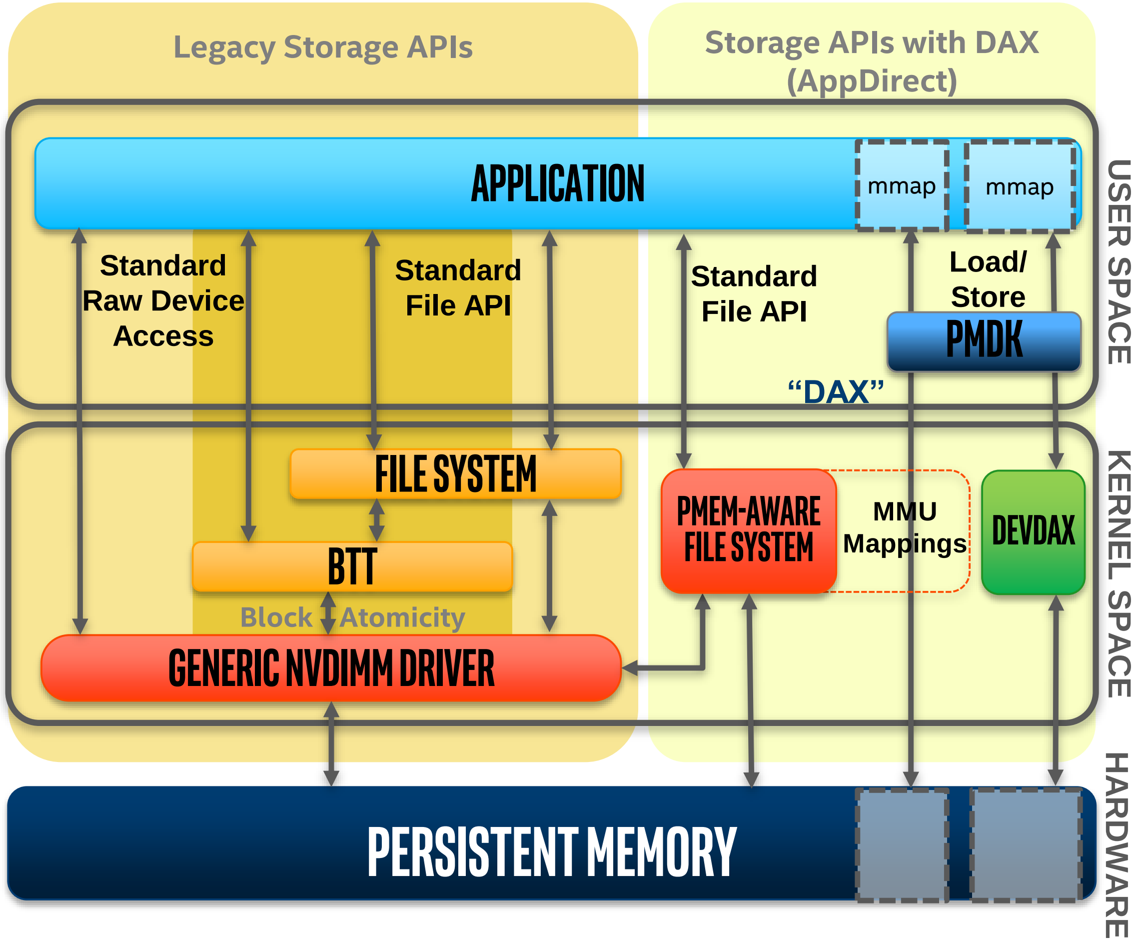
AFFORDABLE MEMORY CAPACITY
FOR MANY APPLICATIONS



APP DIRECT MODE OPTIONS

- No Code Changes Required
- Operates in Blocks like SSD/HDD
  - Traditional read/write
  - Works with Existing File Systems
  - Atomicity at block level
  - Block size configurable
    - 4K, 512B*
- NVDIMM Driver required
  - Support starting Kernel 4.2
- Configured as Boot Device
- Higher Endurance than Enterprise SSDs
- High Performance Block Storage
  - Low Latency, higher BW, High IOPs

*Requires Linux



- Code changes may be required*
- Bypasses file system page cache
- Requires DAX enabled file system
  - XFS, EXT4, NTFS
- No Kernel Code or interrupts
- No interrupts
- Fastest IO path possible

* Code changes required for load/store direct access if the application does not already support this.

Some Challenges in Spark: Memory???

```
18/05/17 14:09:55 INFO storage.ShuffleBlockFetcherIterator: Getting 5262 non-empty blocks out of 6390 blocks
18/05/17 14:09:55 INFO storage.ShuffleBlockFetcherIterator: Started 29 remote fetches in 14 ms
18/05/17 14:09:55 INFO aggregate.ObjectAggregationIterator: Aggregation hash map reaches threshold capacity (128 entries), spilling and fa
18/05/17 14:09:57 INFO python.PythonRunner: Times: total = 735, boot = -1711, init = 1732, finish = 714
18/05/17 14:09:58 INFO python.PythonRunner: Times: total = 1074, boot = -1412, init = 1417, finish = 1069
18/05/17 14:09:58 INFO executor.Executor: Finished task 164.0 in stage 1.0 (TID 6554). 2359 bytes result sent to driver
18/05/17 14:09:58 INFO executor.CoarseGrainedExecutorBackend: Got assigned task 6583
18/05/17 14:09:58 INFO executor.Executor: Running task 193.0 in stage 1.0 (TID 6583)
18/05/17 14:09:58 INFO storage.ShuffleBlockFetcherIterator: Getting 5249 non-empty blocks out of 6390 blocks
18/05/17 14:09:58 INFO storage.ShuffleBlockFetcherIterator: Started 4 remote fetches in 7 ms
18/05/17 14:09:58 INFO aggregate.ObjectAggregationIterator: Aggregation hash map reaches threshold capacity (128 entries), spilling and fa
18/05/17 14:10:27 ERROR executor.CoarseGrainedExecutorBackend: RECEIVED SIGNAL TERM
18/05/17 14:10:27 ERROR util.Utils: Uncaught exception in thread stdout writer for python
java.lang.OutOfMemoryError: Java heap space
        at scala.collection.mutable.ResizableArray$class.ensureSize(ResizableArray.scala:103)
        at scala.collection.mutable.ArrayBuffer.ensureSize(ArrayBuffer.scala:48)
        at scala.collection.mutable.ArrayBuffer.$plus$plus$eq(ArrayBuffer.scala:99)
```

Aggregated Metrics by Executor

Executor ID ▴	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Input Size / Records
1	CANNOT FIND ADDRESS	3 s	2	2	0	52.9 MB / 0
2	CANNOT FIND ADDRESS	3 s	2	2	0	52.4 MB / 0
3	CANNOT FIND ADDRESS	6 s	2	2	0	54.1 MB / 71198

Tasks (7)

Index ▴	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	53	0	FAILED	PROCESS_LOCAL	2 / l4331pp.sss. .com	2016/12/22 09:09:58	3 s	2 s	52.4 MB / 0	java.lang.OutOfMemoryError: Java heap space +details java.lang.OutOfMemoryError: Java heap space at org.apache.spark.ml.tree.impl.DTStatsAggregator.<init>(DTStatsAggregator.scala:77) at org.apache.spark.ml.tree.impl.RandomForest$$anonfun$9$$anonfun$10.apply(RandomForest.scala:528) at org.apache.spark.ml.tree.impl.RandomForest$$anonfun$9$$anonfun$10.apply(RandomForest.scala:524) at scala.Array$.tabulate(Array.scala:331) at org.apache.spark.ml.tree.impl.RandomForest$$anonfun$9.apply(RandomForest.scala:524) at org.apache.spark.ml.tree.impl.RandomForest$$anonfun$9.apply(RandomForest.scala:521) at org.apache.spark.rdd.RDD$$anonfun$mapPartitions$1$$anonfun$apply$23.apply(RDD.scala:785) at org.apache.spark.rdd.RDD$$anonfun$mapPartitions$1$$anonfun$apply$23.apply(RDD.scala:785) at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:38) at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:319) at org.apache.spark.rdd.RDD.iterator(RDD.scala:283) at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:79) at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:47) at org.apache.spark.scheduler.Task.run(Task.scala:86) at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:274)
0	55	1	FAILED	PROCESS_LOCAL	2 / l4331pp.sss .com	2016/12/22 09:10:01	0 ms		0.0 B / 0	ExecutorLostFailure (executor 2 exited caused by one of the running tasks) Reason: Container marked as failed: container_e63_1481607361601_8315_02_000003 on host: l4331pp.sss .com. Exit status: 143. Diagnostics: Container killed on request. Exit code is 143 +details ExecutorLostFailure (executor 2 exited caused by one of the running tasks) Reason: Container marked as failed: container_e63_1481607361601_8315_02_000003 on host: l4331pp.sss Exit status: 143. Diagnostics: Container killed on request. Exit code is 143 Container exited with a non-zero exit code 143 Killed by external signal
0	58	2	RUNNING	NODE_LOCAL	4 / l4331pp.sss .com	2016/12/22 09:10:04	10 min		0.0 B / 0	
1	54	0	FAILED	PROCESS_LOCAL	1 / l4332pp.sss. .com	2016/12/22 09:09:58	3 s	2 s	52.9 MB / 0	java.lang.OutOfMemoryError: Java heap space +details
1	56	1	FAILED	PROCESS_LOCAL	1 / l4332pp.sss. .com	2016/12/22 09:10:01	0 ms		0.0 B / 0	ExecutorLostFailure (executor 1 exited caused by one of the running tasks) Reason: Container marked as failed: container_e63_1481607361601_8315_02_000002 on host: l4332pp.sss .com. Exit status: 143. Diagnostics: Container killed on request. Exit code is 143 +details
1	57	2	FAILED	RACK_LOCAL	3 / l4328pp.sss. .com	2016/12/22 09:10:03	5 s	2 s	54.1 MB / 71198	java.lang.OutOfMemoryError: Java heap space +details

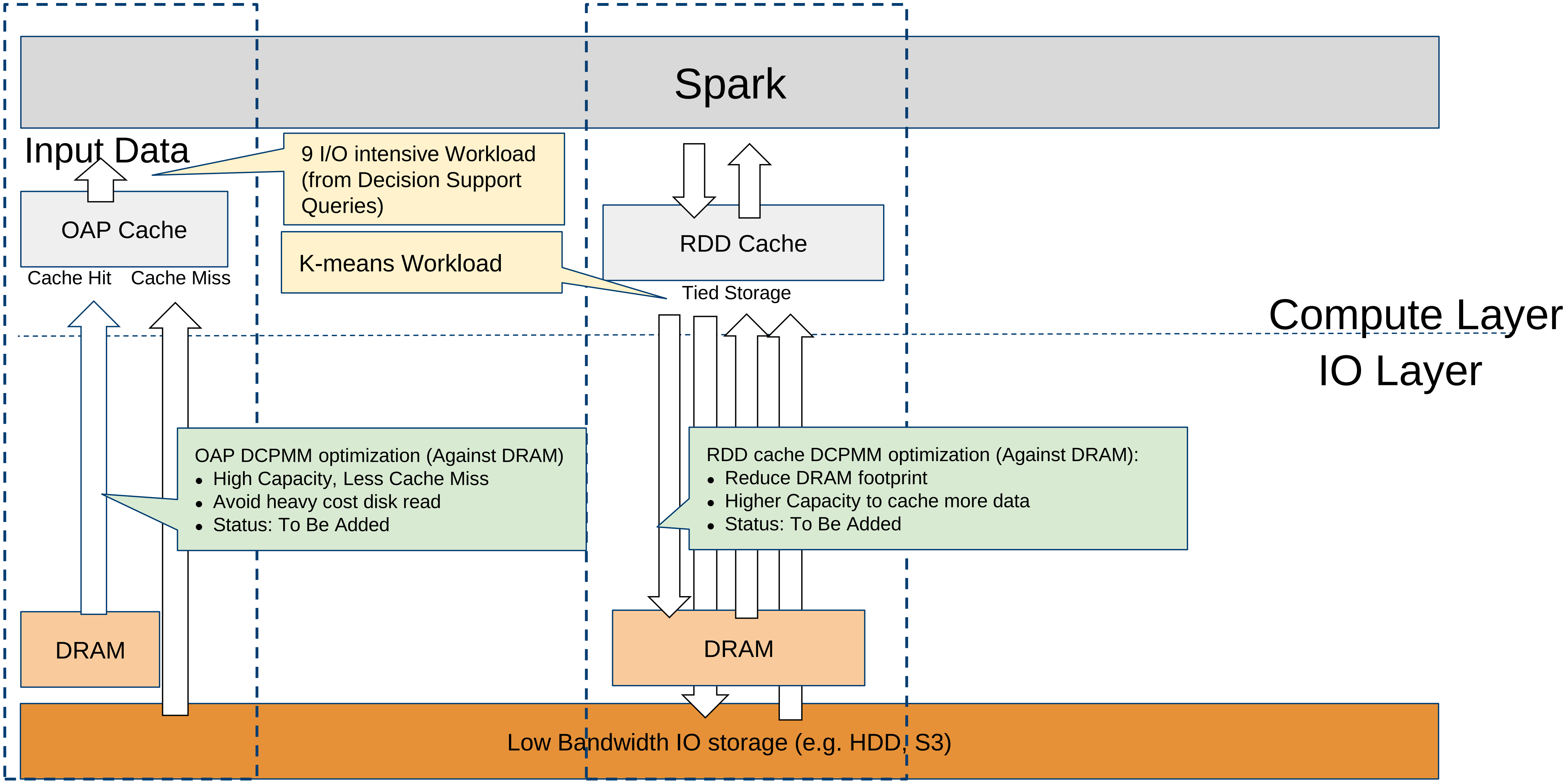
Out of memory

Tasks

Index ▲	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Write Time	Shuffle Write Size / Records	Shuffle Spill (Memory)	Shuffle Spill (Disk)
0	20	0	SUCCESS	NODE_LOCAL	0 / 192-168-1-135.tpgi.com.au	2016/02/16 11:22:57	25 s	5 s	1477.6 KB / 19089	0.2 s	43.1 MB / 1221696	302.8 MB	23.6 MB
1	21	0	SUCCESS	NODE_LOCAL	0 / 192-168-1-135.tpgi.com.au	2016/02/16 11:23:22	23 s	4 s	1478.9 KB / 19090	0.1 s	43.1 MB / 1221760	367.4 MB	28.0 MB
2	22	0	SUCCESS	NODE_LOCAL	0 / 192-168-1-135.tpgi.com.au	2016/02/16 11:23:45	21 s	4 s	1478.4 KB / 19089	0.1 s	43.1 MB / 1221696	367.4 MB	28.5 MB
3	23	0	SUCCESS	NODE_LOCAL	0 / 192-168-1-135.tpgi.com.au	2016/02/16 11:24:06	22 s	5 s	1478.4 KB / 19090	0.1 s	43.1 MB / 1221760	367.4 MB	27.2 MB
4	24	0	SUCCESS	NODE_LOCAL	0 / 192-168-1-135.tpgi.com.au	2016/02/16 11:24:28	20 s	3 s	1478.4 KB / 19091	0.1 s	43.1 MB / 1221824	393.9 MB	25.4 MB
5	25	0	SUCCESS	NODE_LOCAL	0 / 192-168-1-135.tpgi.com.au	2016/02/16 11:24:48	21 s	4 s	1477.5 KB / 19091	0.1 s	43.1 MB / 1221824	367.4 MB	26.9 MB

Large Spill

SPARK DCPMM OPTIMIZATION OVERVIEW



Use Case 1: Spark SQL

OAP I/O Cache

OAP (Optimized Analytics Package)

Goal

- IO cache is critical for I/O intensive workload especially on low bandwidth environment (e.g. Cloud, On-Prem HDD based system)
- Make full use of the advantage from DCPMM to speed up Spark SQL
  - High capacity and high throughput
  - No latency and reduced DRAM footprint
  - Better performance per TCO

Feature

- Fine grain cache (e.g. column chunk for Parquet) columnar based cache
- Cache aware scheduler (V2 API via preferred location)
- Self managed DCPMM pool (no extra GC overhead)
- Easy to use (easily turn on/off), transparent to user (no changes for their queries)

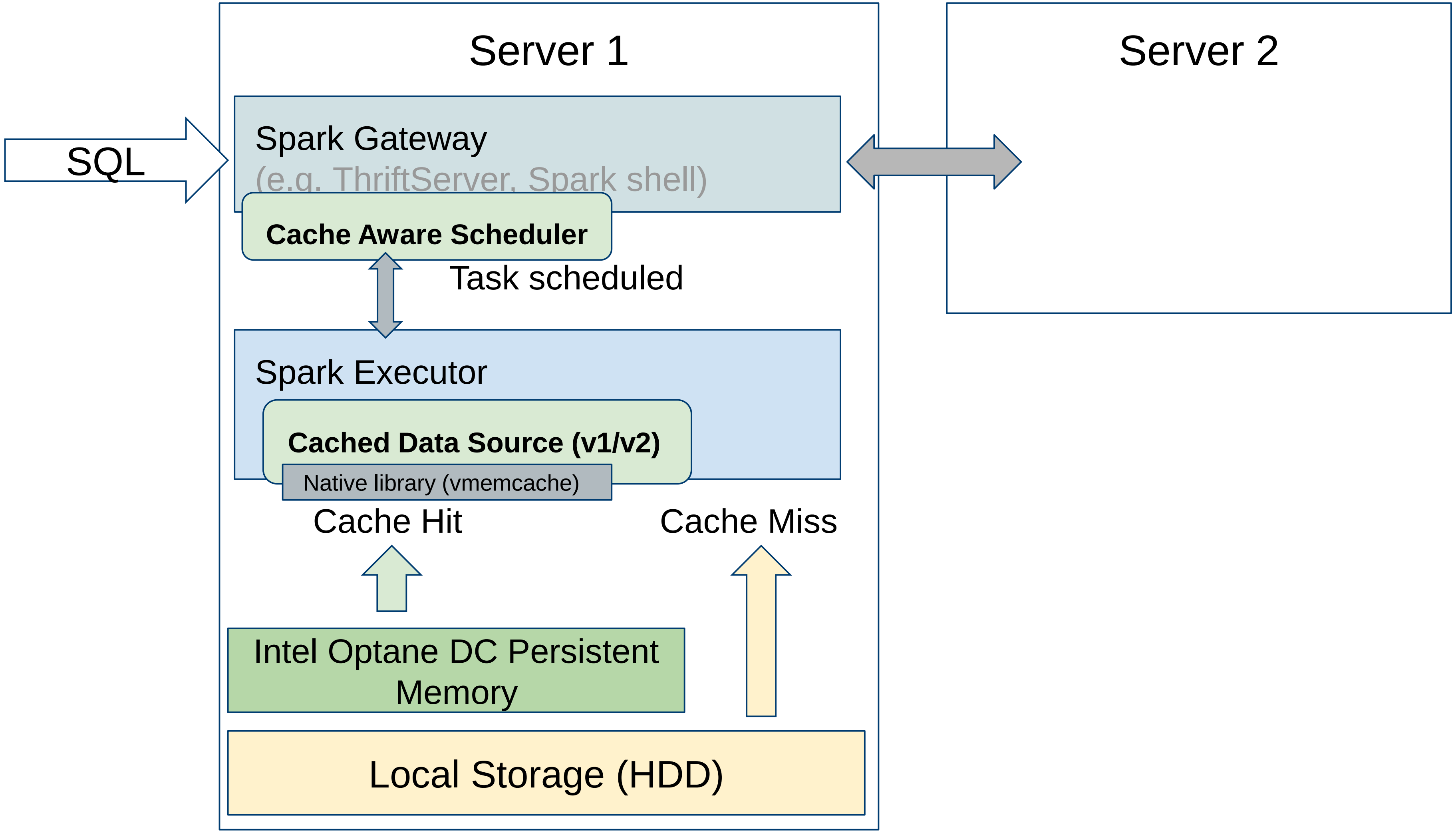
https://github.com/Intel-bigdata/OAP

SPARK DCPMM FULL SOFTWARE STACK

SQL workload	Unmodified SQL
Spark SQL	Unchanged Spark
OAP	Provide scheduler and fine gain cache based on Data Source API
VMEMCACHE	Abstract away hardware details and cache implementation
PMEM-AWARE File SYSTEM	Expose persistent memory as memory-mapped files (DAX)
Intel® Optane™ DC Persistent Memory Module	

Native library to access DCPMM

Deployment Overview



Cache Design - Problem statement

Local LRU cache

Support for large capacities available with persistent memory (many terabytes per server)

Lightweight, efficient and embeddable

In-memory

Scalable

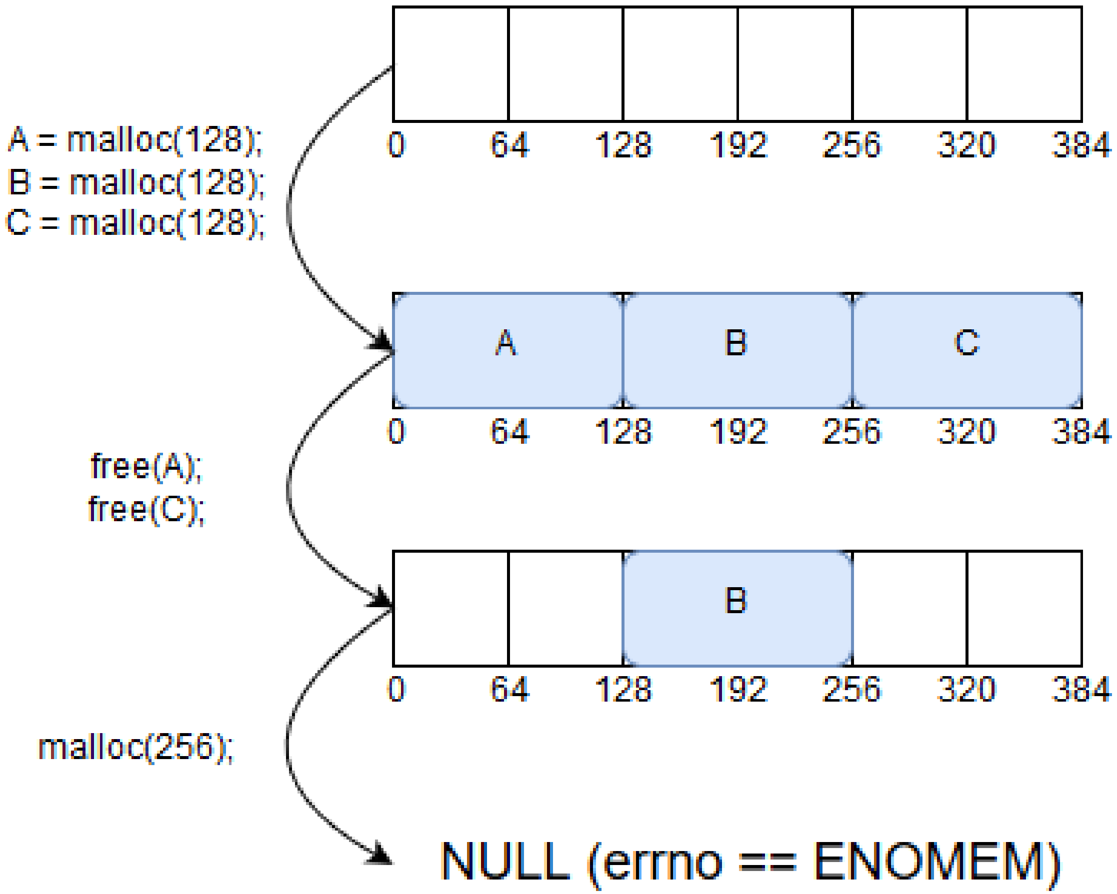
Cache Design - Fragmentation

Manual dynamic memory management a'la dlmalloc/jemalloc/tcmalloc/palloc causes fragmentation

Applications with substantial expected runtime durations need a way to combat this problem

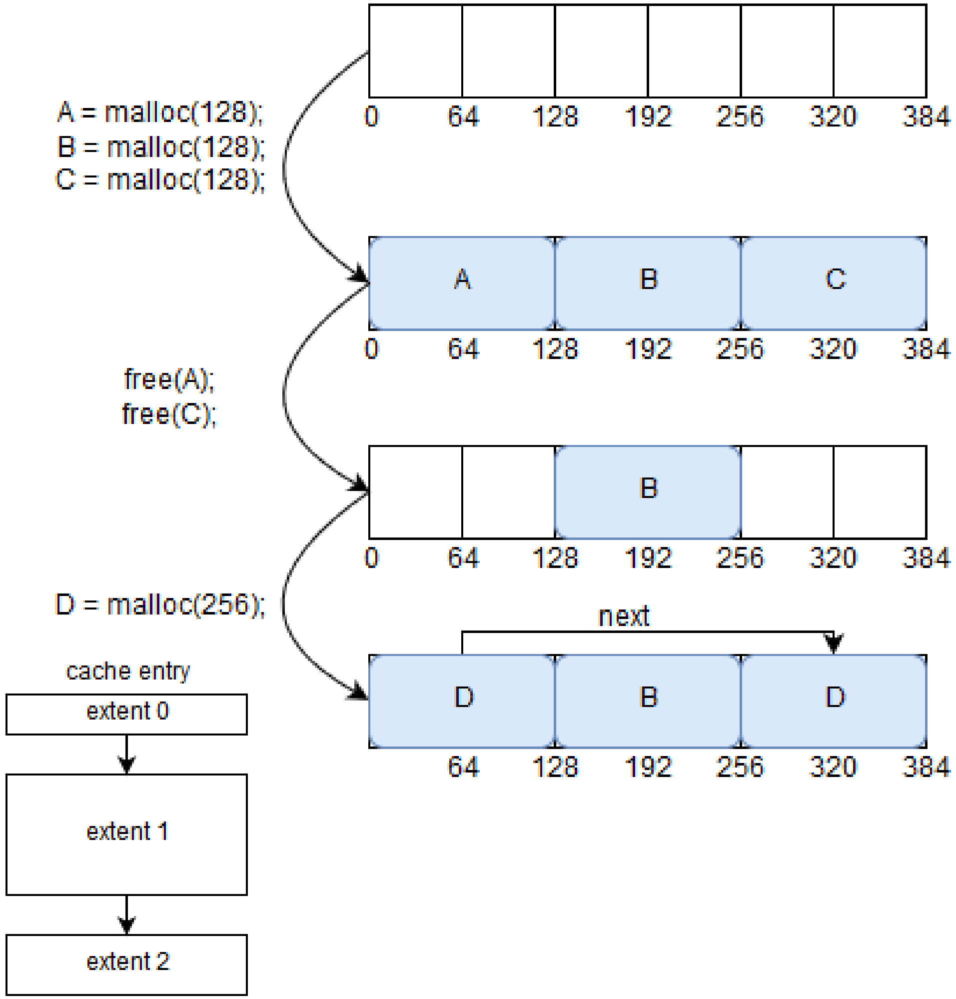
- Compacting GC (Java, .NET)
- Defragmentation (Redis, Apache Ignite)
- Slab allocation (memcached)

Especially so if there's substantial expected variety in allocated sizes



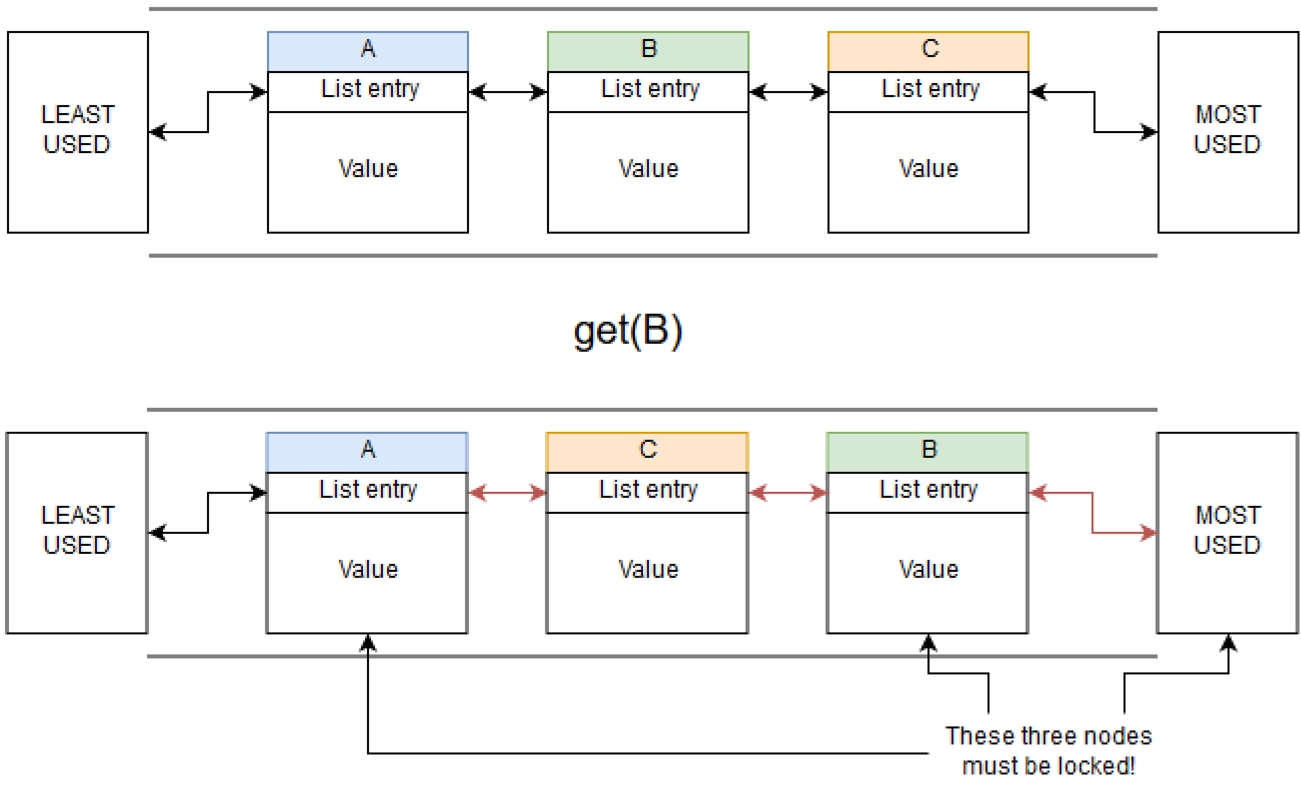
Cache Design - Extent allocation

- If fragmentation is unavoidable, and defragmentation/compacting is CPU and memory bandwidth intensive, let’s embrace it!
- Usually only done in relatively large blocks in file-systems.
- But on PMEM, we are no longer restricted by large transfer units (sectors, pages etc)



Cache Design - Scalable replacement policy

- Performance of libvmemcache was bottlenecked by naïve implementation of LRU based on a doubly-linked list.
- With 100st of threads, most of the time of any request was spent
  waiting on a list lock…
- Locking per-node doesn't solve the problem…

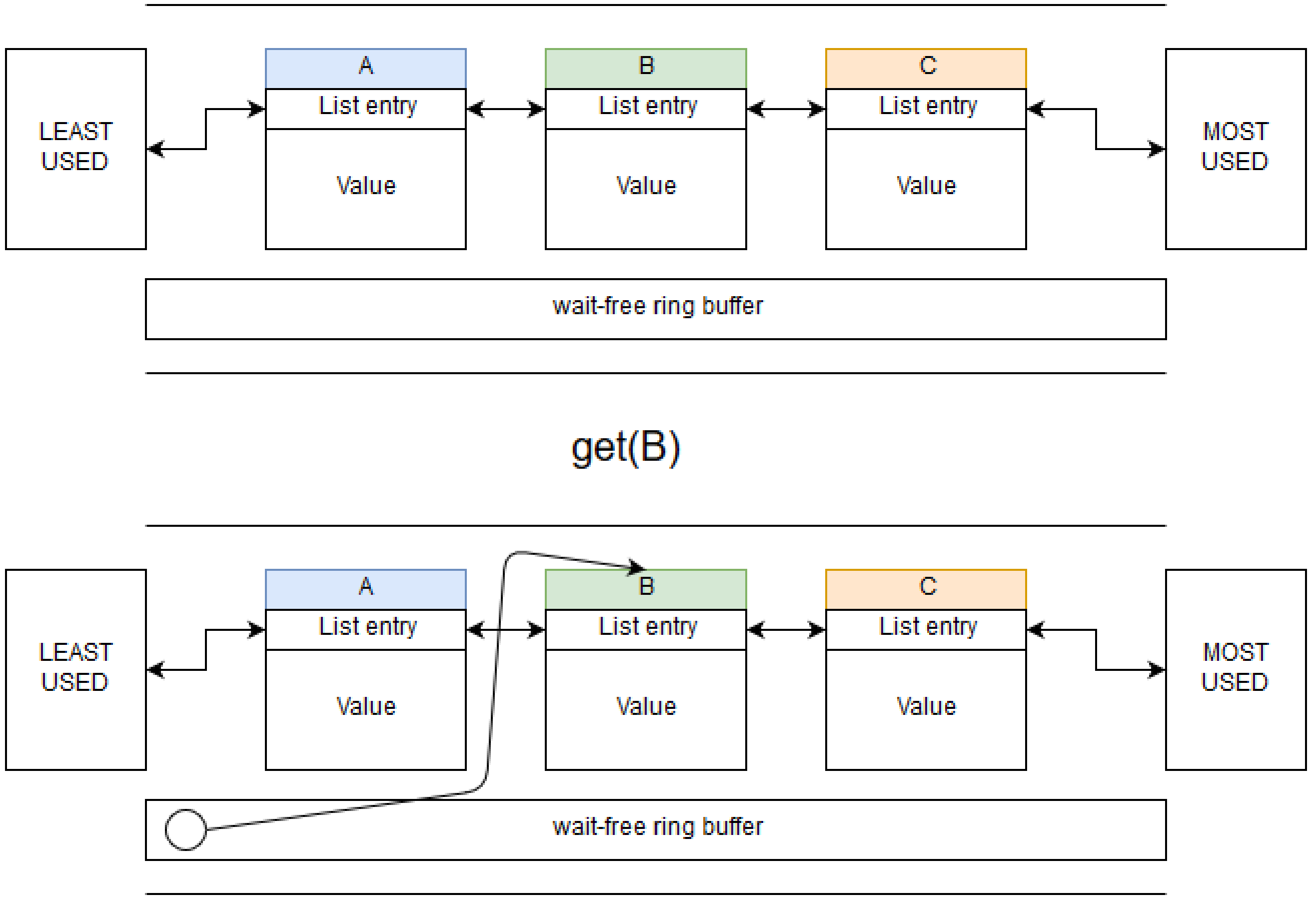


Cache Design - Buffered LRU

Our solution was quite simple.

We've added a wait-free ringbuffer which buffers the list-move operations

This way, the list only needs to get locked during eviction or when the ringbuffer is full.



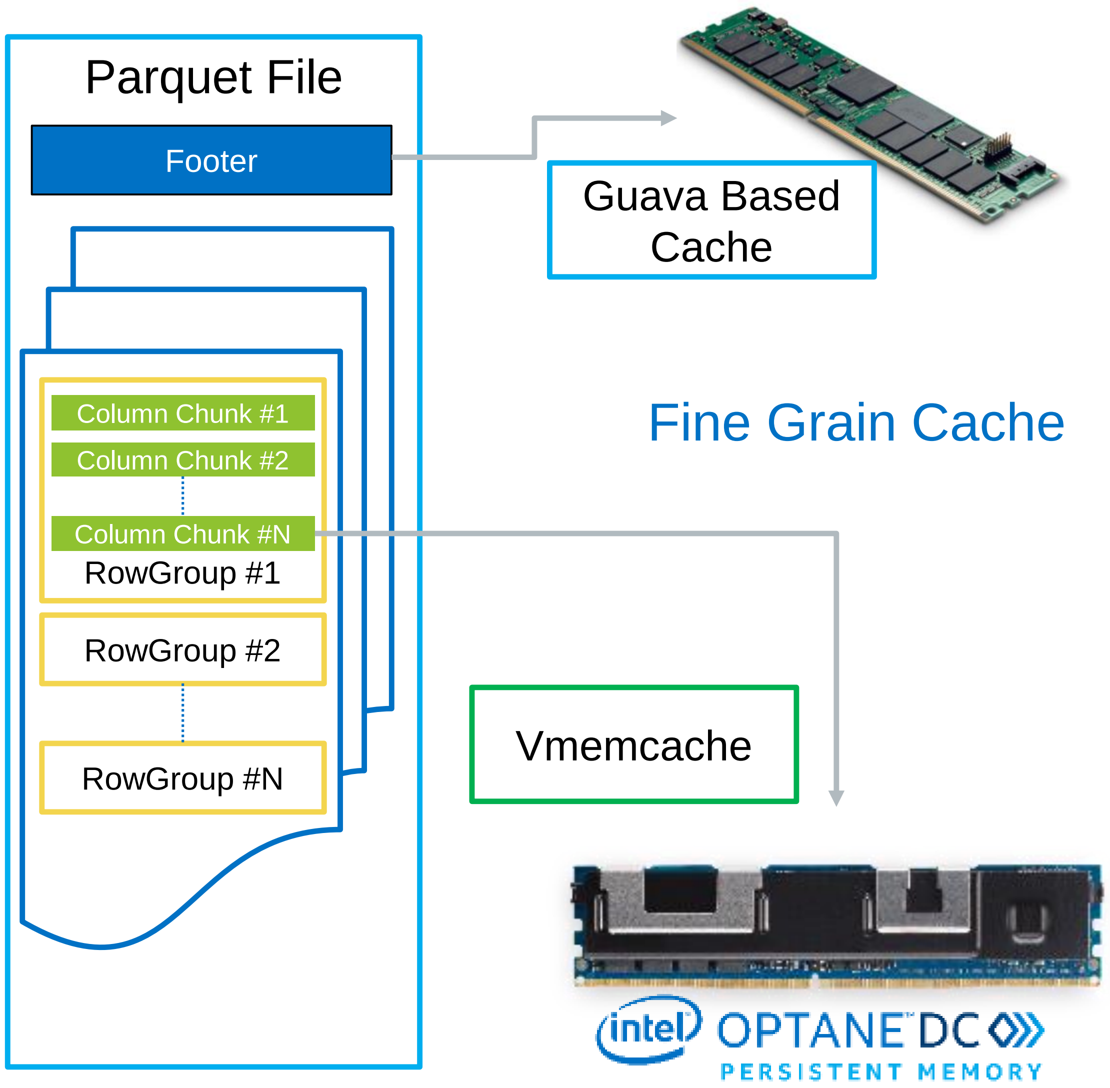
Cache Design - Lightweight, embeddable, in-memory caching

```
VMEMcache *cache = vmemcache_new("/tmp", VMEMCACHE_MIN_POOL,
VMEMCACHE_MIN_EXTENT, VMEMCACHE_REPLACEMENT_LRU);

const char *key = "foo";
vmemcache_put(cache, key, strlen(key), "bar", sizeof("bar"));

char buf[128];
ssize_t len = vmemcache_get(cache, key, strlen(key),
        buf, sizeof(buf), 0, NULL);

vmemcache_delete(cache);
```

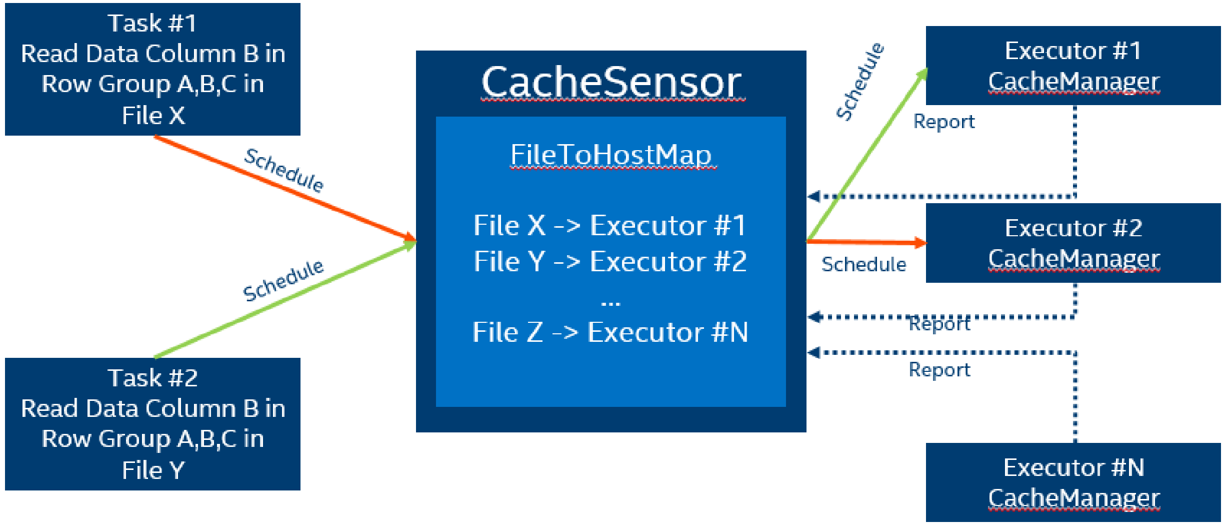
libvmemcache has normal get/put APIs, optional replacement policy, and configurable extent size
Works with terabyte-sized in-memory workloads without a sweat, with very high space utilization.
Also works on regular DRAM.

https://github.com/pmem/vmemcache

Cache Design – Status and Fine Grain



Fine Grain Cache

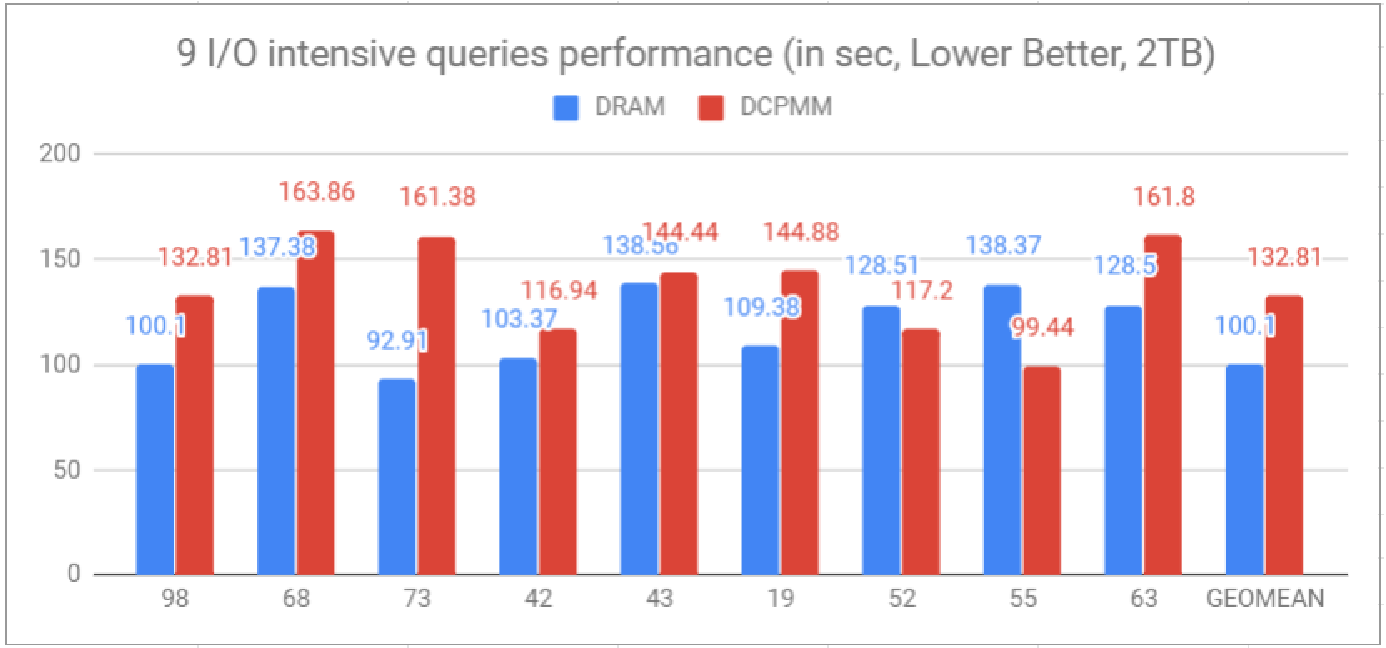


Cache Status & Report

Experiments and Configurations

		DCPMM	DRAM
Hardware	DRAM	192GB (12x 16GB DDR4)	768GB (24x 32GB DDR4)
	Intel Optane DC Persistent Memory	1TB (QS: 8 x 128GB)	N/A
	DCPMM Mode	App Direct (vmemcache)	N/A
	SSD	N/A	N/A
	CPU	2 * Cascadelake 8280M (Thread(s) per core: 2, Core(s) per socket: 28, Socket(s): 2 CPU max MHz: 4000.0000 CPU min MHz: 1000.0000 L1d cache: 32K, L1i cache: 32K, L2 cache: 1024K, L3 cache: 39424)	
	OS	4.20.4-200.fc29.x86_64 (BKC: WW06'19, BIOS: SE5C620.86B.0D.01.0134.100420181737)	
Software	OAP	1TB DCPMM based OAP cache	610GB DRAM based OAP cache
	Hadoop	8 * HDD disk (ST1000NX0313, 1-replica uncompressed & plain encoded data on Hadoop)	
	Spark	1 * Driver (5GB) + 2 * Executor (62 cores, 74GB), spark.sql.oap.rowgroup.size=1MB	
	JDK	Oracle JDK 1.8.0_161	
Workload	Data Scale	2TB, 3TB, 4TB	
	Decision Making Queries	9 I/O intensive queries	
	Multi-Tenants	9 threads (Fair scheduled)	

Test Scenario 1: Both Fit In DRAM And DCPMM - 2TB

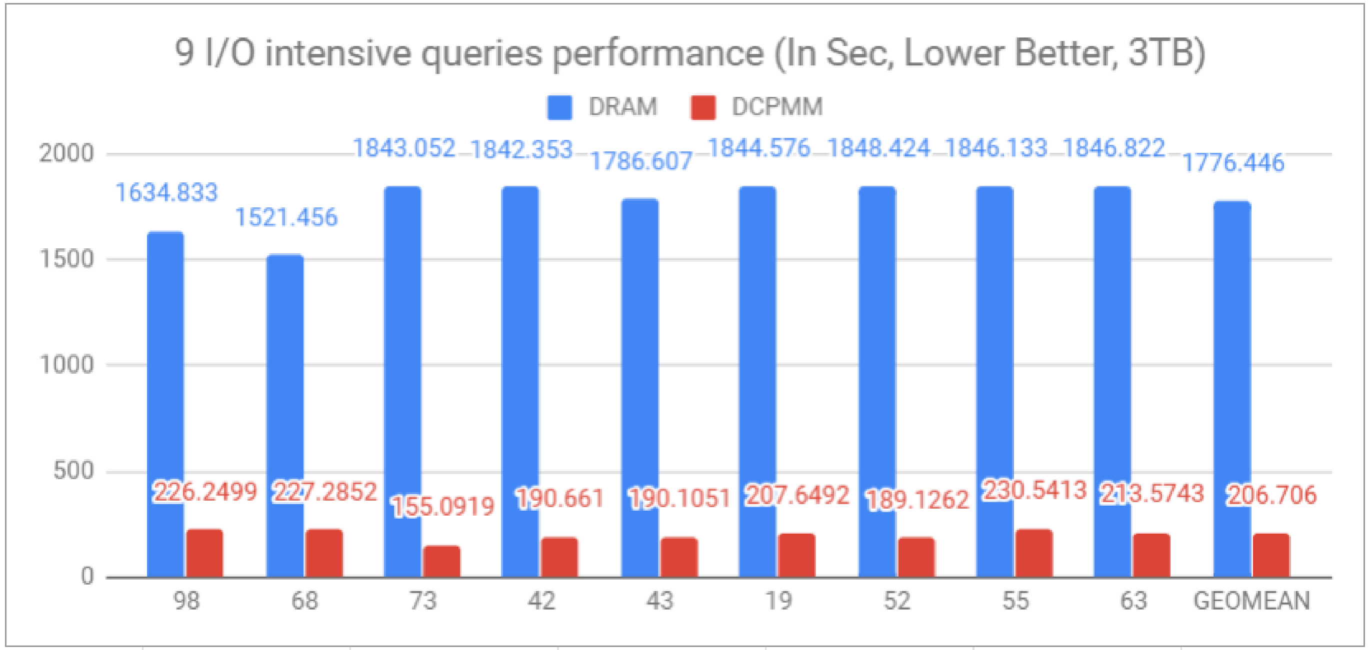


On Premise Performance

With 2TB Data Scale, both DCPMM and DRAM based OAP cache can hold the full dataset (613GB). DCPMM is *24.6%* (=1- 100.1/132.81) performance lower than DRAM as measured on 9 I/O intensive decision making queries.

*The performance gain can be much lower if IO is improved (e.g. compression & encoding enabled) or some hacking codes fixed (e.g. NUMA scheduler)

Test Scenario 2: Fit In DCPMM Not For DRAM - 3TB

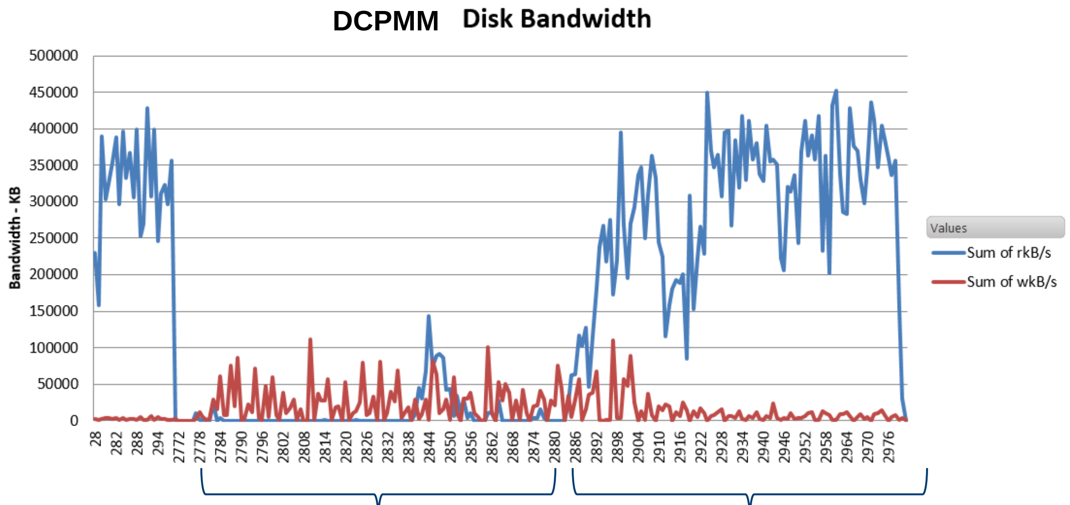


On Premise Performance

With 3TB Data Scale, only DCPMM based OAP cache can hold the full dataset (920GB). DCPMM shows *8X** performance gain over DRAM as measured on 9 I/O intensive decision making queries.

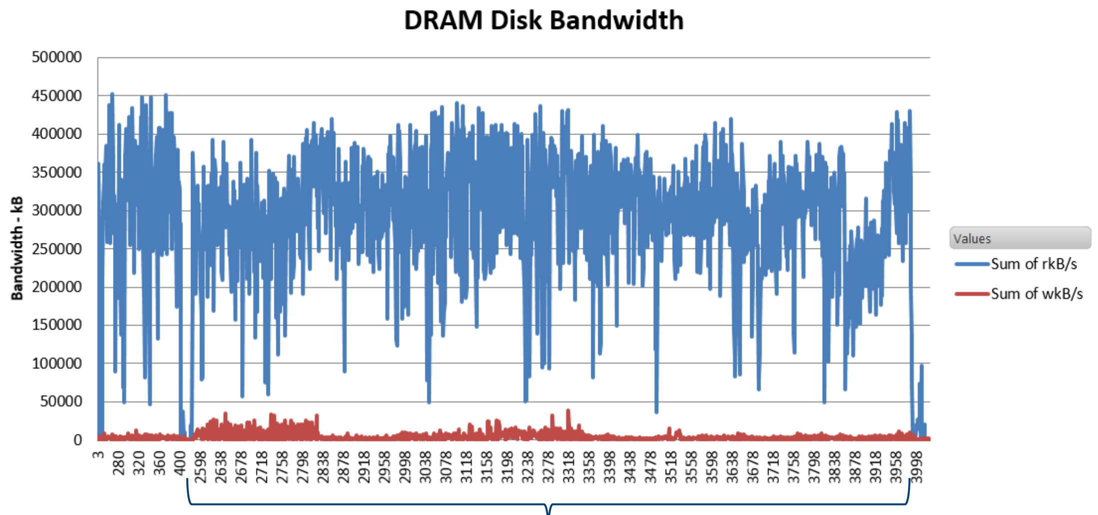
*The performance gain can be much lower if IO is improved (e.g. compression & encoding enabled) or some hacking codes fixed (e.g. NUMA scheduler)

Performance Analysis - System Metrics



OAP Cache works avoiding disk read (blue) and only disk write for shuffle (red)

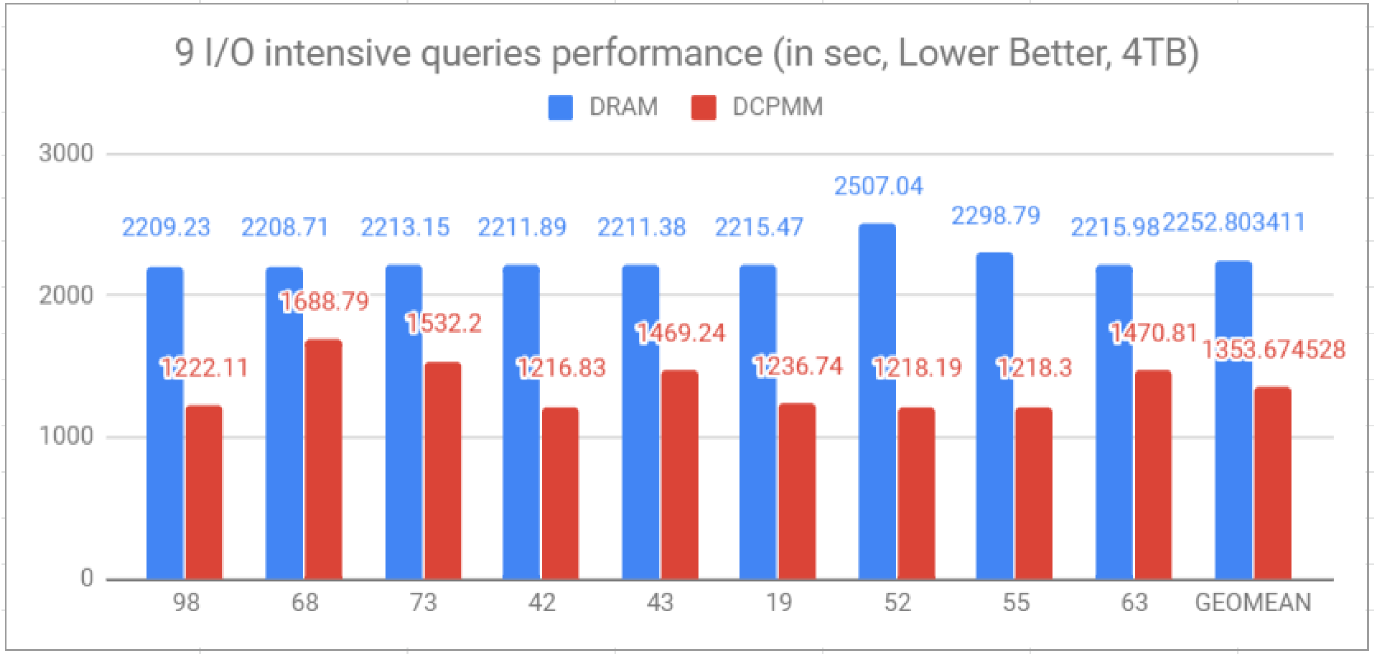
Disk read (blue) comes from shuffle data



Disk read (blue) happens from time to time

- Input data is all cached in DCPMM while partially for DRAM
- DCPMM reaches up to 18GB/s bandwidth while DRAM case it's bounded by disk IO which is only about 250MB/s ~ 450MB/s
- OAP cache doesn't apply for shuffle data (intermediate data) that extra IO pressure put onto DRAM case
- In DCPMM case Spark reads from disk about 27.5GB while DRAM one reads about 395.6GB

Test Scenario 3: None Of DRAM &DCPMM Fit - 4TB



On Premise Performance

With 4TB Data Scale, none of DCPMM and DRAM based OAP cache can hold the full dataset (1226.7GB). DCPMM shows ***1.66X**** (=2252.80/1353.67) performance gain over DRAM as measured on 9 I/O intensive decision making queries.

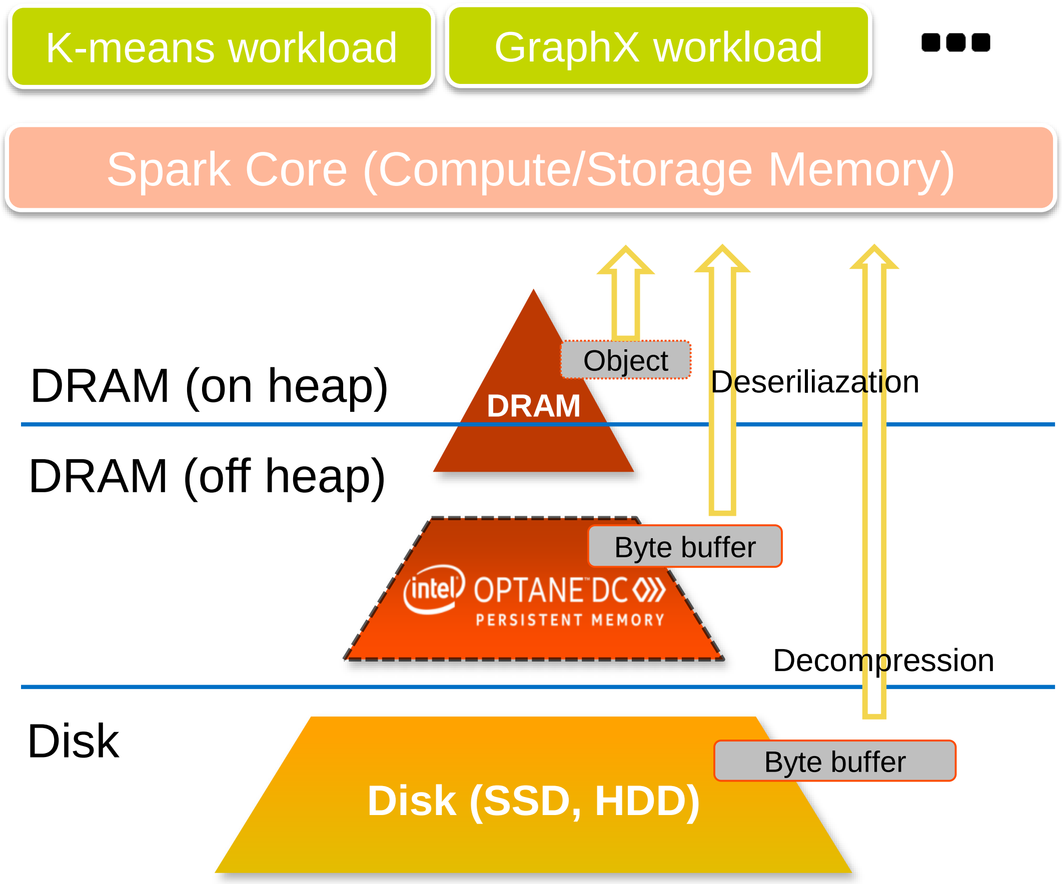
*The performance gain can be much lower if IO is improved (e.g. compression & encoding enabled) or some hacking codes fixed (e.g. NUMA scheduler)

Use Case 2: Machine Learning

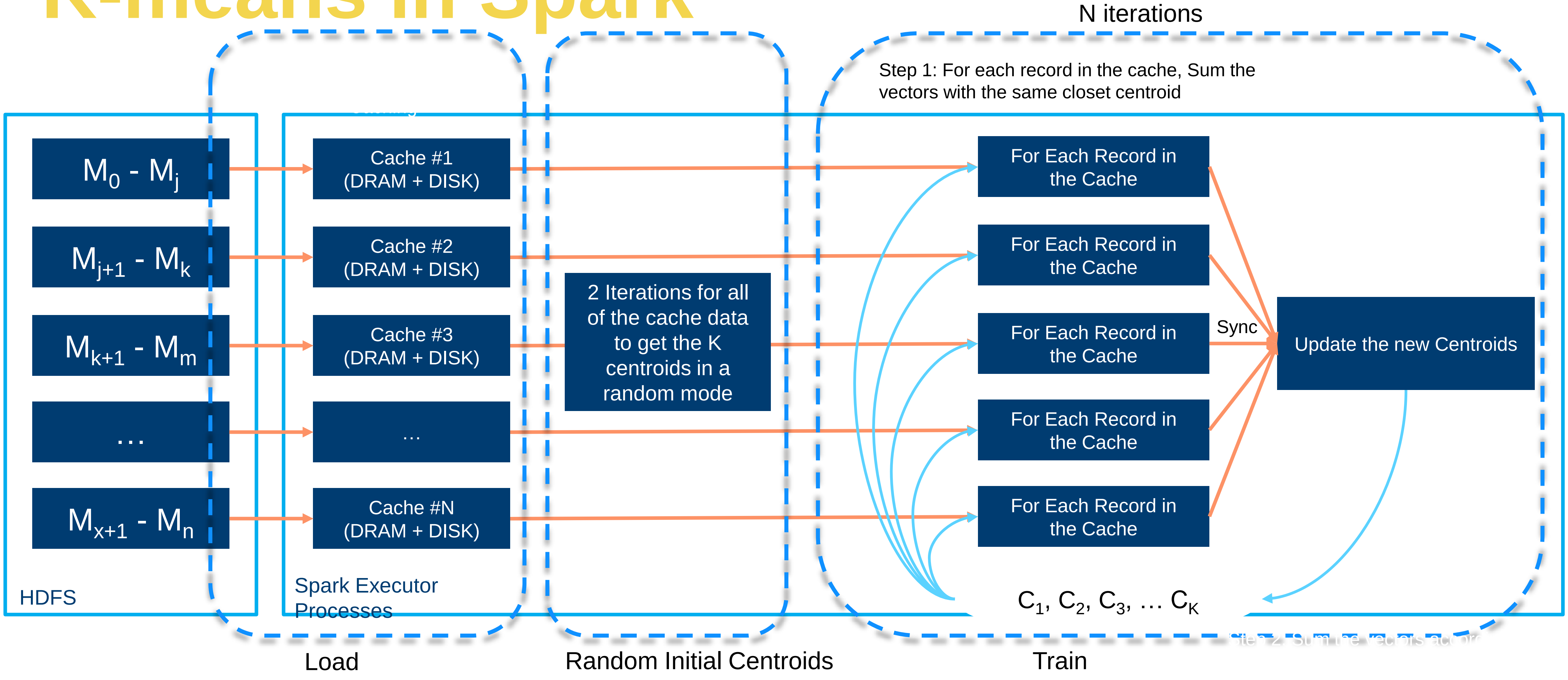
Spark K-means

DCPMM Storage Level

- Spark Storage Level
  - A few storage levels serving for different purposes including memory and disk
  - Off-heap memory is supported to avoid GC overhead in Java
  - Large capacity storage level (disk) is widely used for iterative computation workload (e.g. K-means, GraphX workloads) by caching hot data in storage level
- DCPMM Storage Level
  - Extend memory layer
  - Using Pmem library to access DCPMM avoiding the overhead of decompression from disk
  - Large capacity and high I/O performance of DCPMM shows better performance than tied solution (original DRAM + Disk solution)

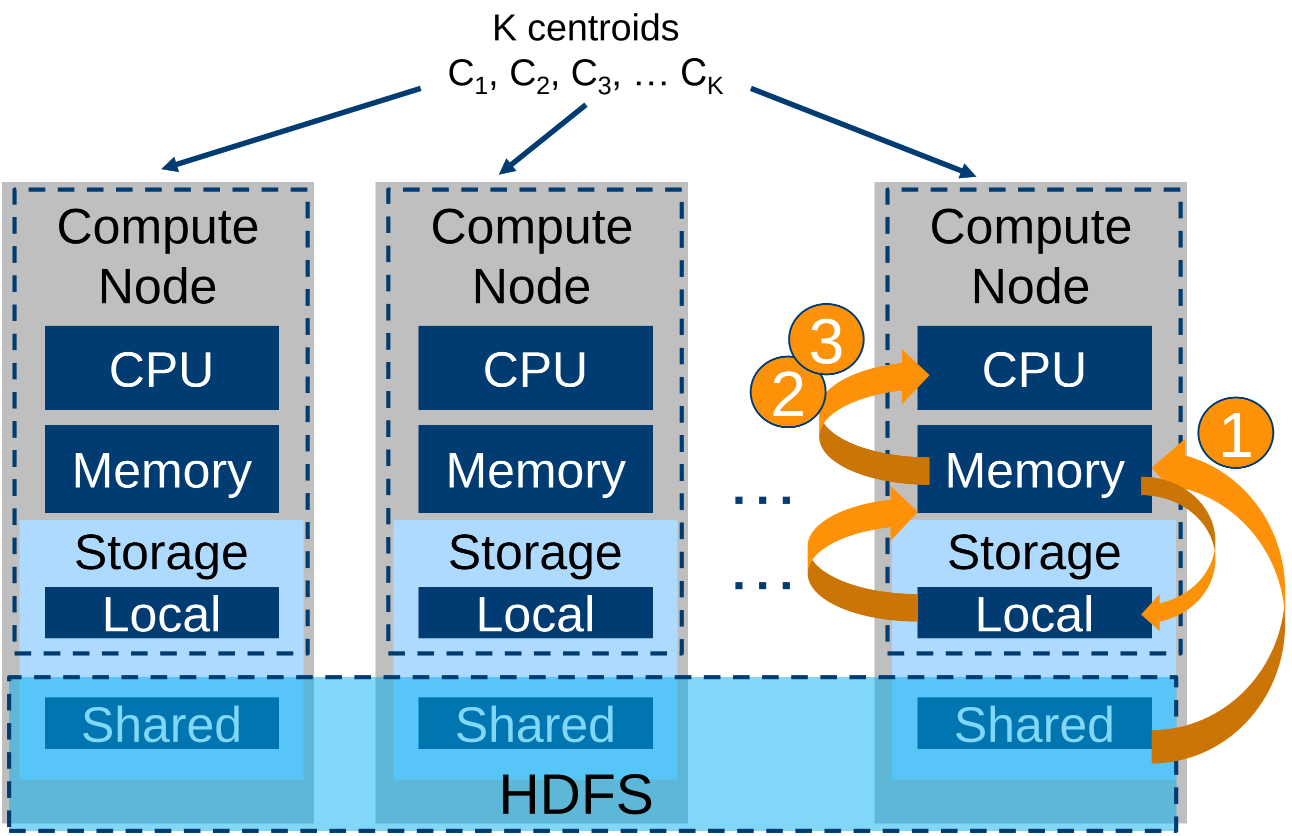


K-means in Spark



Load the data from HDFS to DRAM (and DCPMM / SSD if DRAM cannot hold all of the data)(**Load**), after that, the data will not be changed, and will be iterated repeatedly in **Initialization** and **Train** stages.

K-means Basic Data Flow



1 **Load**

- Load data from HDFS to memory.
- Spill over to local storage

2 **Initialization**

- Compute using initial centroid based on data in memory or local storage

3 **Train**

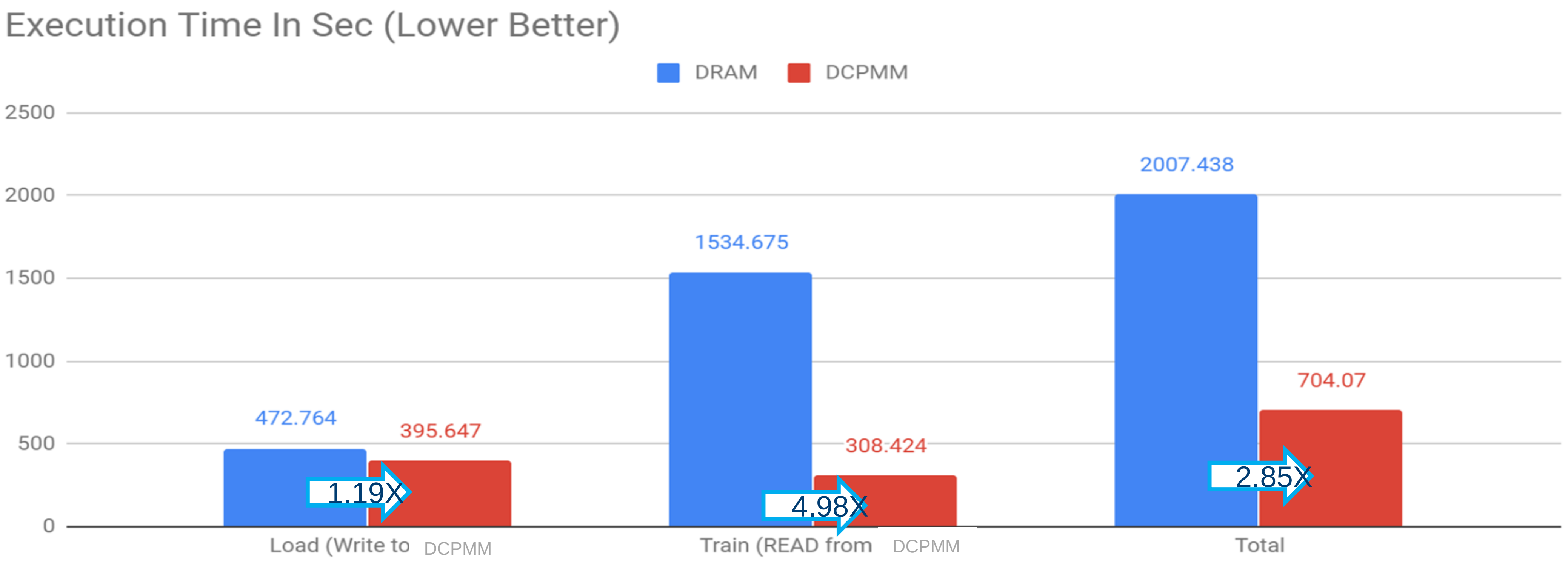
- Compute iterations based on local data

Experiments and Configurations

[1] Modified to support DCPMM

		#1 AD	#2 DRAM
Hardware	DRAM	192GB (12x 16GB DDR4 2666)	768GB (24× 32GB DDR4 2666)
	Intel Optane DC Persistent Memory	1TB (8x 128GB QS)	N/A
	DCPMM Mode	AppDirect	N/A
	CPU	Intel(R) Xeon(R) Platinum 8280L CPU @ 2.70GHz	
	Disk Drive	8x Intel DC S4510 SSD (1.8T) + 2 * P4500 (1.8T)	
Software	CPU / Memory Allocation	Driver (5GB + 10 cores) + 2 * Executor (80GB + 32 Cores)	
	Software Stack	Spark (2.2.2-snapshot: bb3e6ed9216f98f3a3b96c8c52f20042d65e2181) + Hadoop (2.7.5)	
	OS & Kernel & BIOS	Linux-4.18.8-100.fc27.x86_64-x86_64-with-fedora-27-Twenty_Seven BIOS: SE5C620.86B.0D.01.0299.122420180146	
	Mitigation variants (1,2,3,3a,4, L1TF)	1,2,3,3a,4, L1TF	
	Cache Spill Disk	N/A	8 * SSD
	NUMA Bind	2 NUMA Nodes bind 2 Spark Executors respectively	
	Data Cache Size (OffHeap)	DCPMM (no spill)	680GB DRAM (partially spill)
Workload	Record Count / Data Size / Memory Footprint	Row size 6.1 Billion Records / data size 1.1TB / cache data 954GB	
	Cache Spill Size	0	235GB
	Kmeans (Iteration times)	10	
	Kmeans (K value)	5	

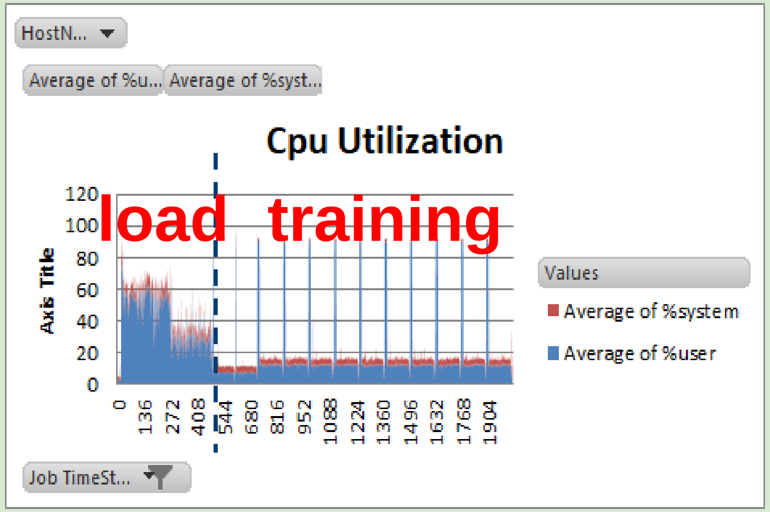
Performance Comparison(DCPMM AD V.S. DRAM)

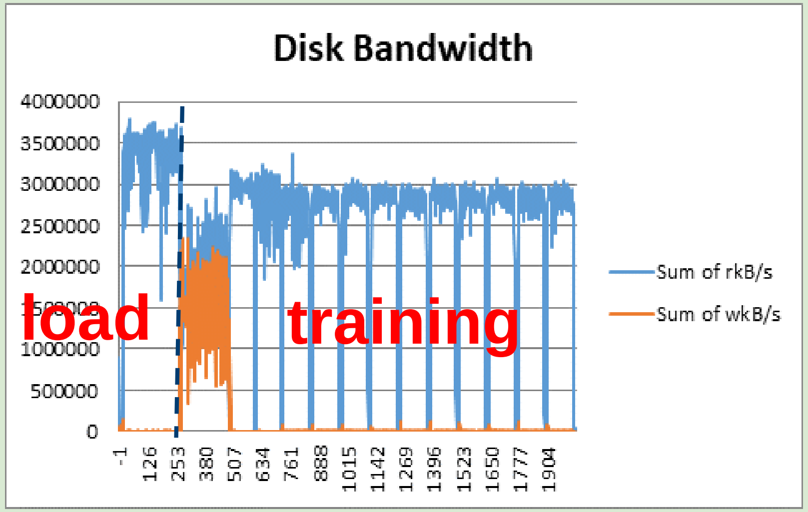


- DCPMM provides larger “memory” than DRAM and avoid the data spill in this case, hence got up to 4.98X performance in Training(READ), and 2.85X end to end performance;
- The DCPMM AppDirect mode is about ~19% faster than DRAM in Load(WRITE cache into DCPMM / DRAM);
- Training(READ) execution gap of DRAM case caused by storage media, as well as the data decompression and de-serialization from the local spill file;

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks. Configurations: See page11.Test by Intel on 14/03/2019.

System Analysis

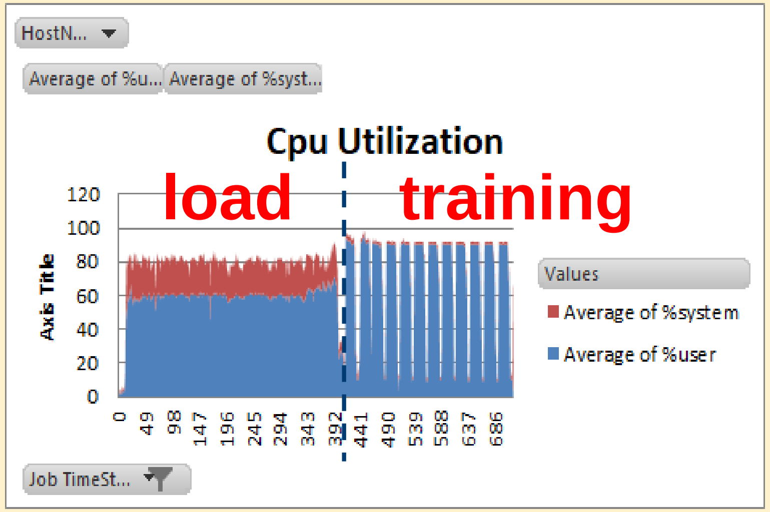


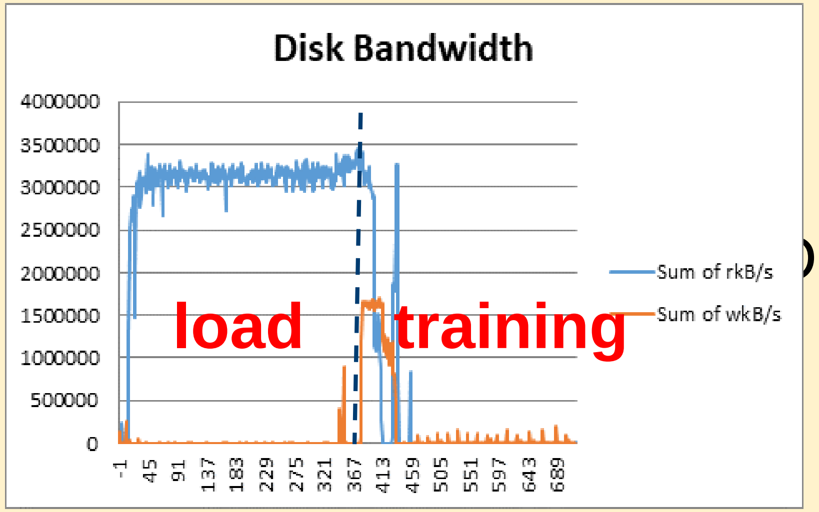


DRAM

DRAM

- SSD read bandwidth is 2.86GB/s
- It's I/O bound leading to low CPU utilization





DCPMM

DCPMM

- Much high CPU utilization than DRAM only system

Summary

- Intel Optane DC Persistent Memory brings high capacity, low latency and high throughput
- Two operation modes for DCPMM:
    - App-Direct
    - Memory Mode
- Good for Spark in:
    - memory bound workload
    - I/O intensive workload

Q&A