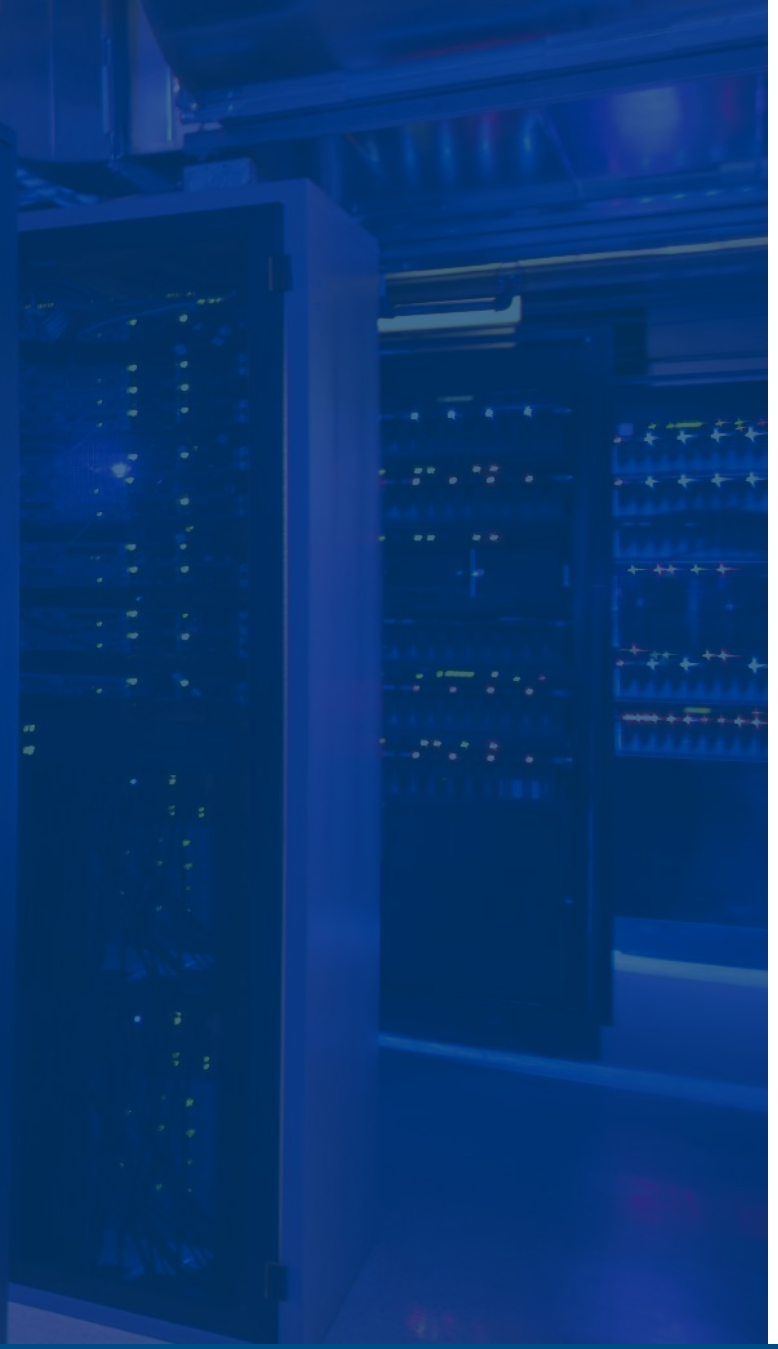
MemKeyDB: Adopting Redis to Persistent Memory

Jakub Schmiegel
Software Architect
Intel



Agenda

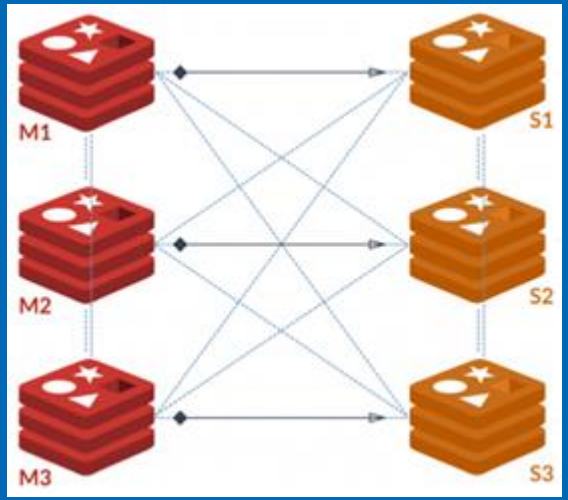
1. **Redis introduction**
   What is Redis? How to make it persistent?
2. **Redis with PMem**
   Challenges of modification
3. **Memkind**
   Memory allocator
4. **MemKeyDB**
   Features and internals

What is Redis?

- Key-value store, cache
  - Values: 8 types + modules
  - 100+ commands to manipulate on data
  - Single threaded, multi-IO
  - In-memory DB
- Horizontal scaling – challenge
  - Memory pressure

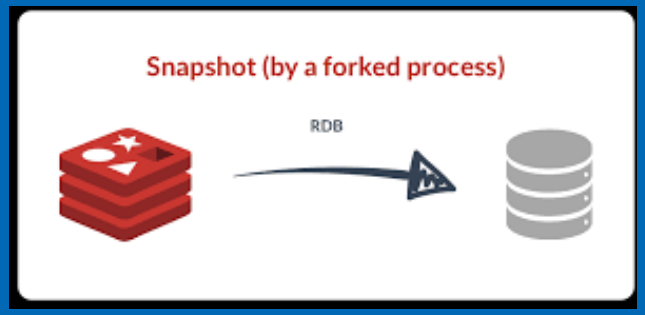
DB-ENGINES

Rank			DBMS	Database Model	Score		
May 2021	Apr 2021	May 2020			May 2021	Apr 2021	May 2020
1.	1.	1.	Redis	Key-value, Multi-model	162.17	+6.28	+18.69



Persistency of data in Redis

- RDB snapshots
  - Perfect for backup
  - Possible data loss
- AOF logging
  - More durable
  - Slower
- Replication



Redis with Persistent Memory

- Store data on Persistent Memory, benefit from persistency
  - Redis as cache
- How to make new solution easy to use
  - Different releases are used
  - Various configurations
  - Don’t create new product
- Experiments:
  - Transactional allocation on FS DAX, Action API, Memkind on FS DAX

Memkind – volatile allocator

- Heterogenous heap manager:
  - DRAM, PMEM, HBW

```
void * ptr_default = memkind_malloc(MEMKIND_DEFAULT, size);
```

```
struct memkind *pmem_kind = NULL;
memkind_create_pmem("/mnt/pmem", PMEM_MAX_SIZE, &pmem_kind);
void* ptr_pmem = memkind_malloc(pmem_kind, size);
```

  - Common “free”

```
memkind_free(NULL, ptr_default);
memkind_free(NULL, ptr_pmem);
```

Redis + Memkind

Requirements

- Native persistency: RDB, AOF
- Use DRAM and PMEM, define Ratio
- Support for all Redis commands, structures and API
- Small modification
- Acceptable performance

Copy-on-Write support

- FS DAX
- KMEM DAX

Automatic NUMA recognition

- Closest NUMA node

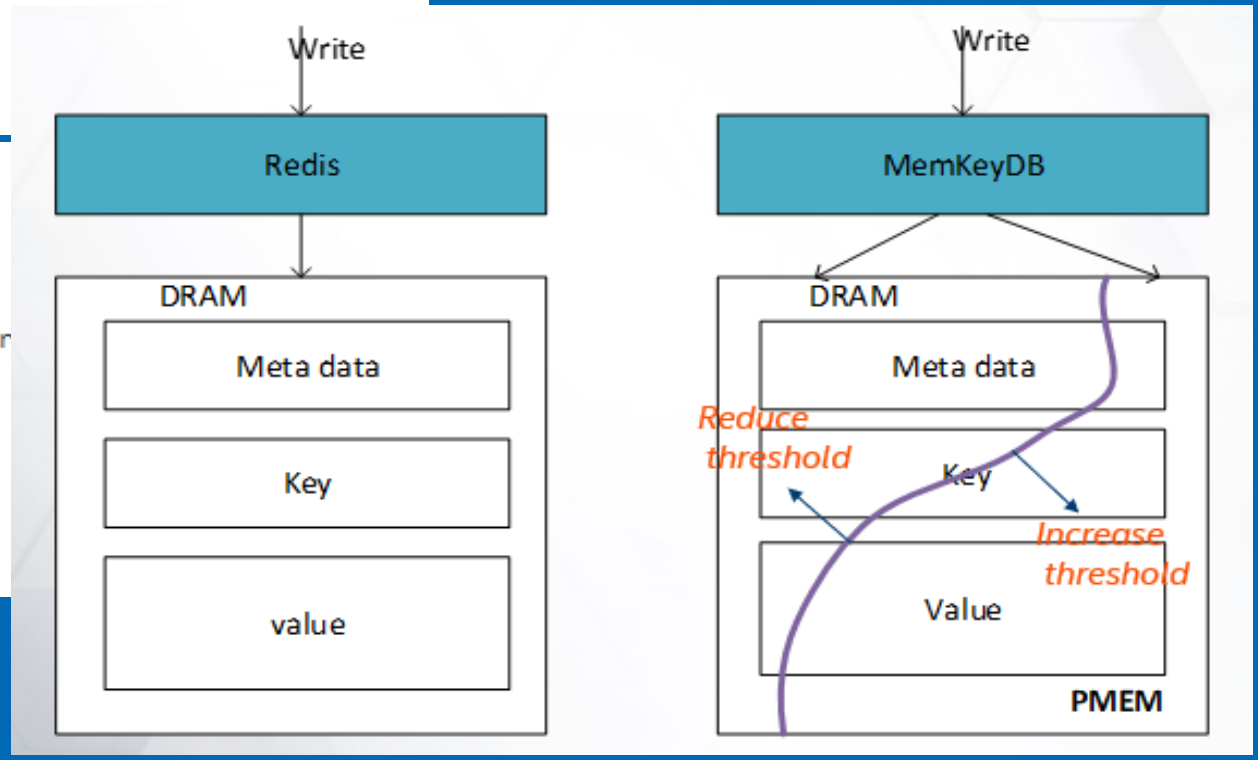
```
$ numactl -H

  available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1
node 0 size: 80249 MB
node 0 free: 68309 MB
node 1 cpus: 28 29 30 31 32 33 34 35 36 37 38 39 40 41
node 1 size: 80608 MB
node 1 free: 71958 MB
node 2 cpus:
node 2 size: 1026048 MB
node 2 free: 1026048 MB
node 3 cpus:
node 3 size: 512000 MB
node 3 free: 512000 MB
node distances:
node   0   1   2   3
  0:  10  21  17  28
  1:  21  10  28  17
  2:  17  28  10  28
  3:  28  17  28  10
```

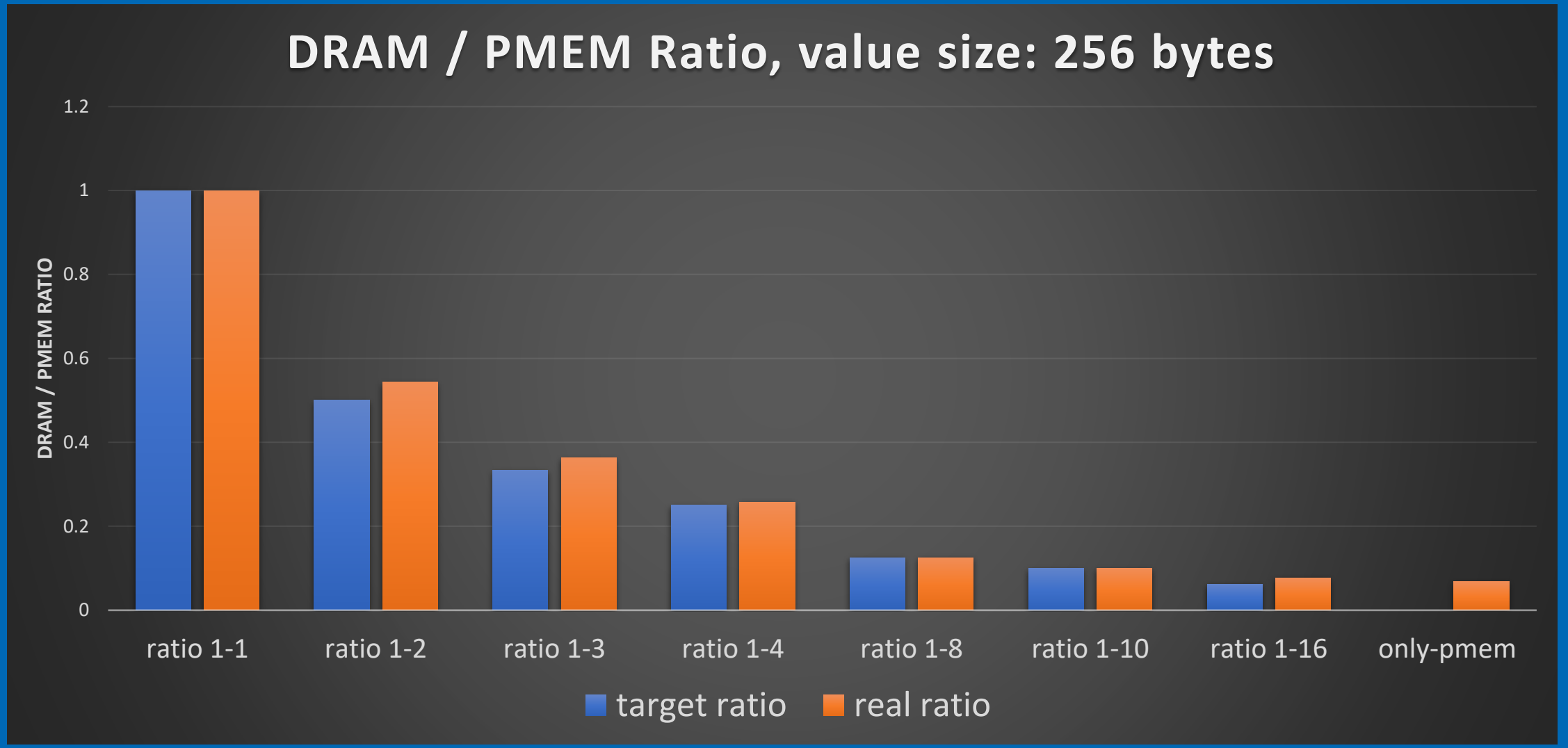
DRAM / PMEM memory ratio

```
# Redis supports four different Memory Allocation Policies:
#
# only-dram: use only DRAM - do not use Persistent Memory
# only-pmem: use only Persistent Memory - do not use DRAM
# threshold: use both Persistent Memory and DRAM - use threshold described by static-threshold
# ratio: use both Persistent Memory and DRAM - use ratio described by dram-pmem-ratio
#
# By default Redis use only-dram configuration.
memory-alloc-policy only-dram
```

```
# The syntax of dram-pmem-ratio directive is the following:
#
# dram-pmem-ratio <dram_value> <pmem_value>
#
# Expected proportion of memory placement between DRAM and Persistent Memor
# Real DRAM:PMEM ratio depends on workload and its variability.
# dram_value and pmem_value are values from range <1,INT_MAX>
# In the example below the behavior will be to:
# Place 25% of all memory in DRAM and 75% in Persistent Memory
dram-pmem-ratio 1 3
```

```
# Keep hashtable structure always on DRAM
hashtable-on-dram yes
```




DRAM / PMEM Ratio, value size: 256 bytes
DRAM / PMEM RATIO
1.2
1
0.8
0.6
0.4
0.2
0
ratio 1-1
ratio 1-2
ratio 1-3
ratio 1-4
ratio 1-8
ratio 1-10
ratio 1-16
only-pmem
target ratio
real ratio

What we changed in Redis

- DRAM and PMEM allocations

```
264
265 void *zmalloc(size_t size) {
266     return (size < pmem_threshold) ? zmalloc_dram(size) : zmalloc_pmem(size);
267 }
268
```

```
162
163 void *zmalloc_dram(size_t size) {
164     void *ptr = malloc(size+PREFIX_SIZE);
```

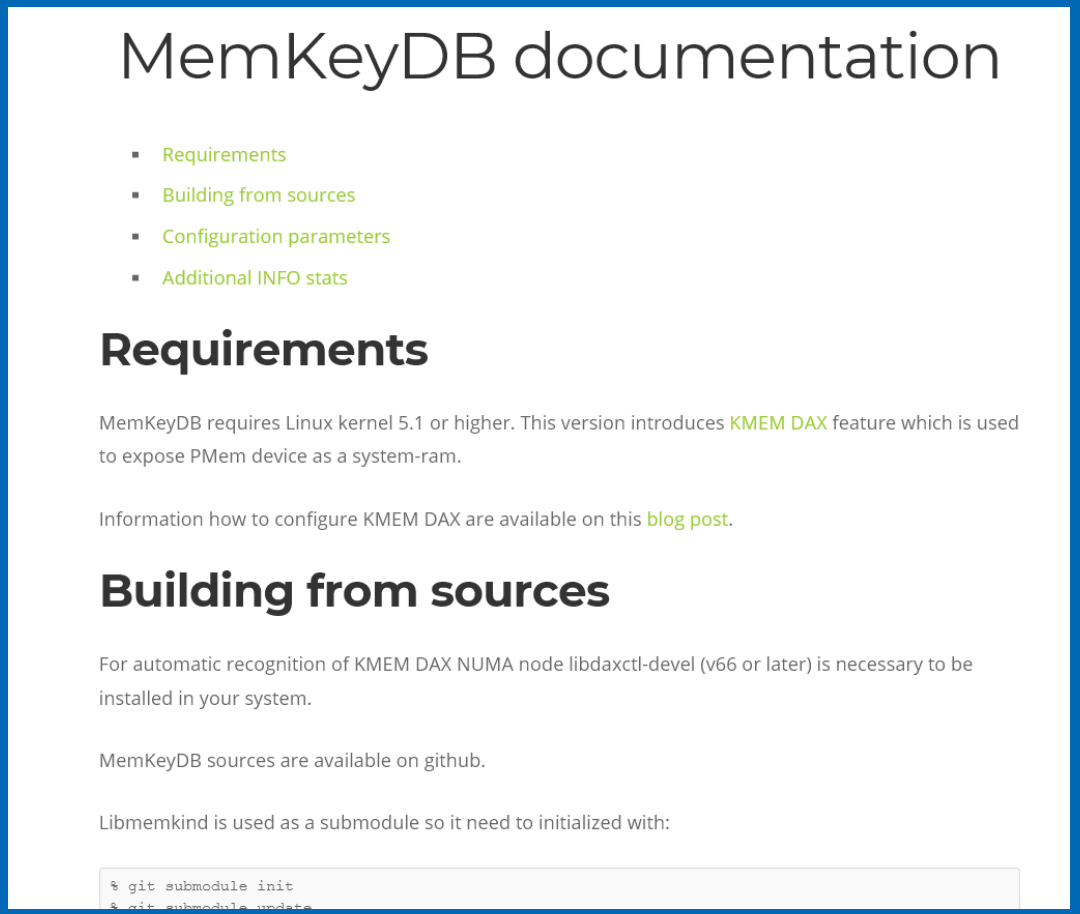
```
204 static void *zmalloc_pmem(size_t size) {
205     void *ptr = memkind_malloc(MEMKIND_DAX_KMEM, size+PREFIX_SIZE);
```

- Optimizations

```
@@ -108,8 +108,8 @@ client *createClient(connection *conn) {
        c->name = NULL;
        c->bufpos = 0;
        c->qb_pos = 0;
-       c->querybuf = sdsempty();
-       c->pending_querybuf = sdsempty();
+       c->querybuf = sdsdramempty();
+       c->pending_querybuf = sdsdramempty();
```

What we delivered

- https://github.com/memKeyDB/memKeyDB
- Ported to more than 10 Redis versions
- Webpage: memkeydb.io
- Benchmarks



MemKeyDB documentation

- Requirements
- Building from sources
- Configuration parameters
- Additional INFO stats

Requirements

MemKeyDB requires Linux kernel 5.1 or higher. This version introduces KMEM DAX feature which is used to expose PMem device as a system-ram.

Information how to configure KMEM DAX are available on this blog post.

Building from sources

For automatic recognition of KMEM DAX NUMA node libdaxctl-devel (v66 or later) is necessary to be installed in your system.

MemKeyDB sources are available on github.

Libmemkind is used as a submodule so it need to initialized with:

```
% git submodule init
% git submodule update
```

MemKeyDB in Public Clouds

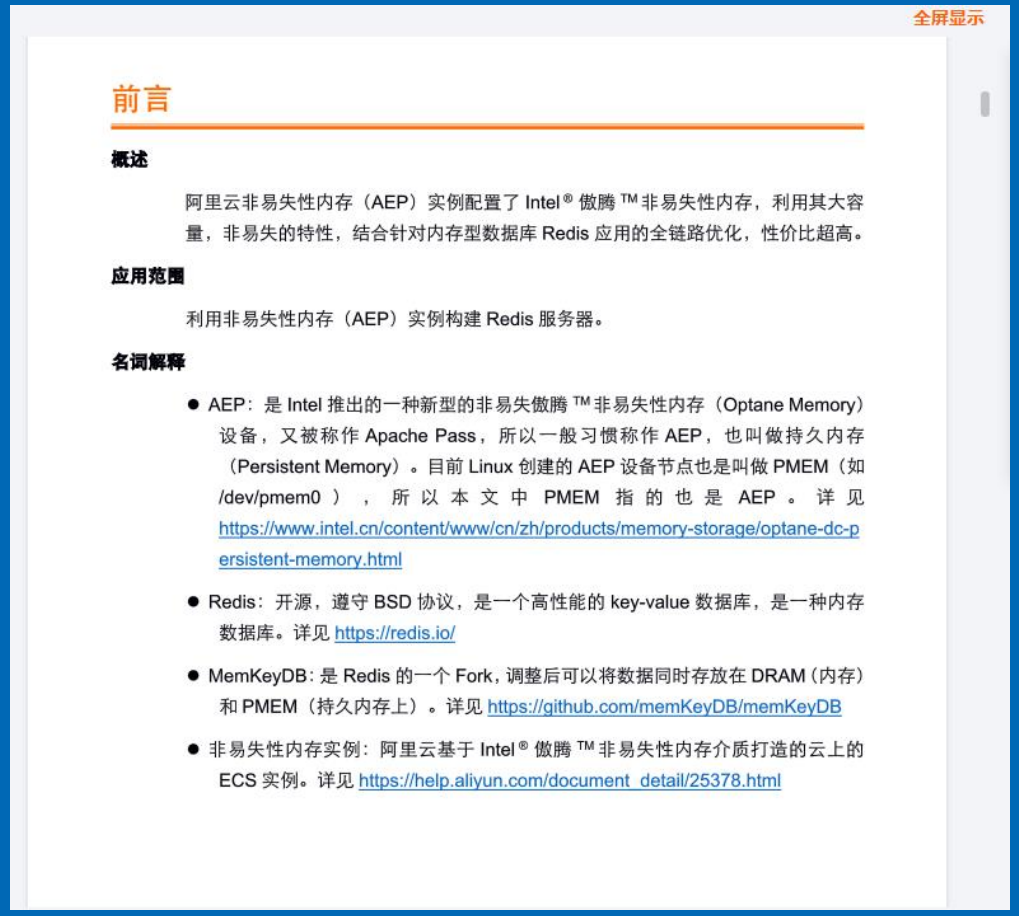
- https://www.alibabacloud.com/help/zh/doc-detail/188250.htm
- https://partners-intl.aliyun.com/help/doc-detail/188250.htm

Deploy Redis applications on re6p instances

Last Updated: Apr 25, 2021

You can run Redis applications on persistent memory optimized instances to reduce memory costs per GiB. To ensure performance, you must modify your Redis applications. To reduce your modification costs, Alibaba Cloud provides re6p instance types for Redis applications. You can deploy Redis applications on re6p instances by running several commands. In this topic, Alibaba Cloud Linux 2 and CentOS are used to demonstrate how to deploy Redis applications on re6p instances.

- Whitepaper:
  https://bp.aliyun.com/detail/170
  60+ page instruction and description of this solution (Chinese only)




intel®
SPDK, PMDK, Intel® Performance Analyzers
Virtual Forum