13th ANNUAL WORKSHOP 2017

USER VERBS FOR SCALED PERFORMANCE WITH SHARED MEMORY

Santosh Shilimkar, Avneesh Pant, Sumanta Chatterjee & Amarnath Jolad

March 27, 2017

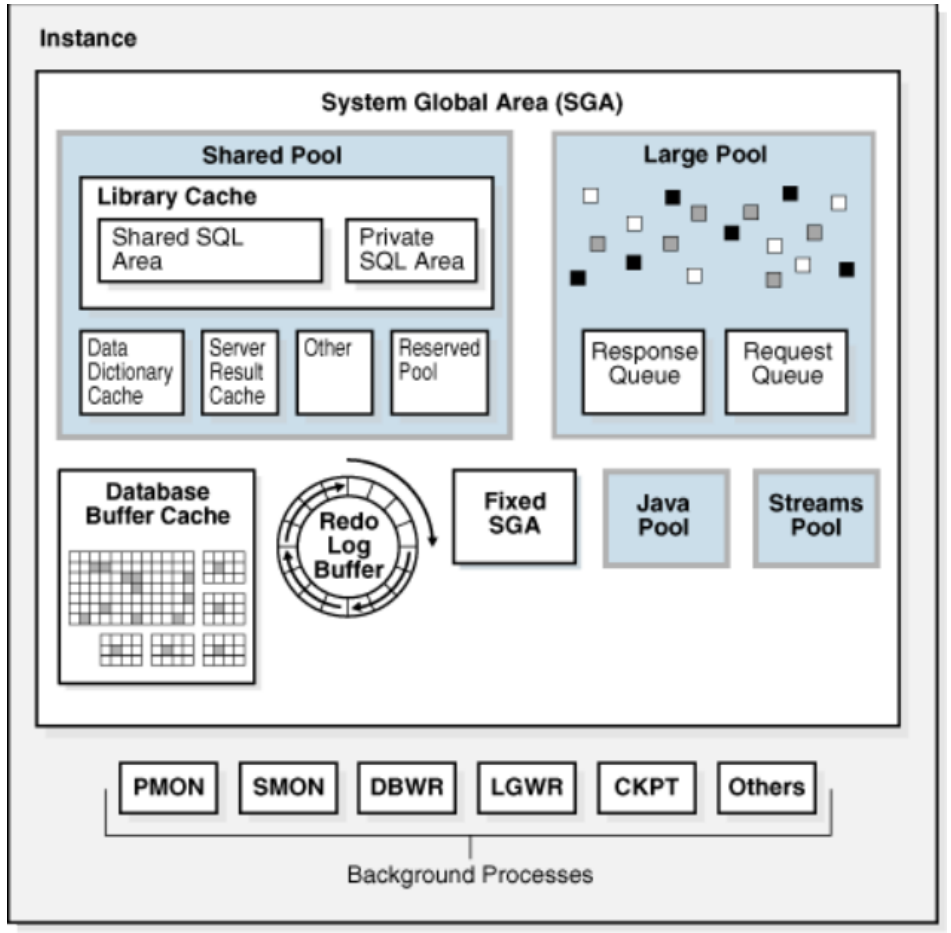
AGENDA

- Overview of Oracle Process and Shared Memory model.
- Shortcoming of existing upstream verbs for large shared memory.
- Shared PD API semantics and example usage.
- Preliminary results with using Shared PD in Oracle.
- Opportunity with using Large shared MR and potential verb API for it.
- Possible inclusion of these verb extension to standards.

ORACLE PROCESS - OVERVIEW

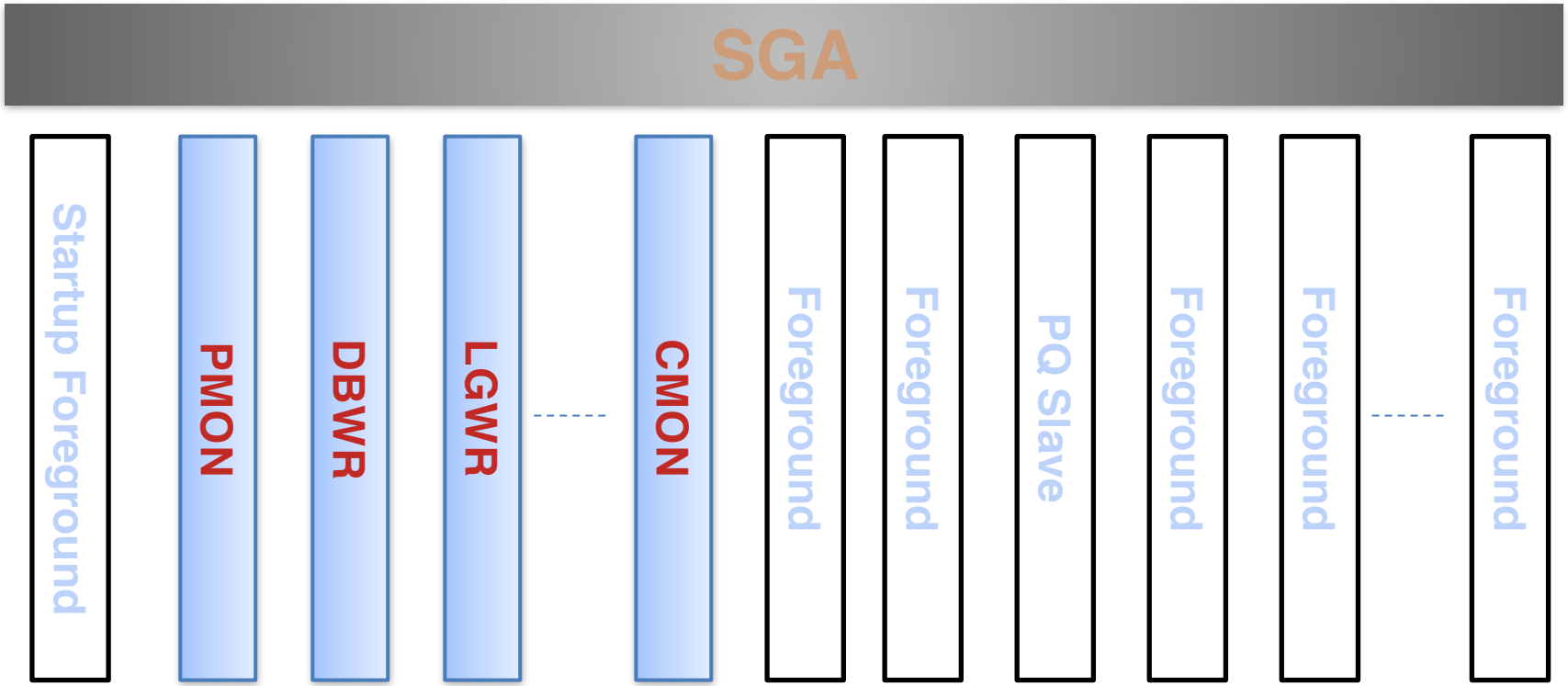
- **An “Oracle Process” is an execution entity**
- **Run time instantiation of DB server code that executes user requests and background tasks**
- **An “Oracle Process” is not always an “OS process”**
  - a “OS Process” on most UNIX platforms by default.
  - a “OS Thread”, default on Windows and available on most UNIX platforms.
  - a “User thread” or “Fiber” available on Windows.
- **Foreground/Shadow process : DB Server process created to service a client**
- **Background process : processes created to perform DB specific tasks**

DATABASE INSTANCE & MEMORY MANAGEMENT



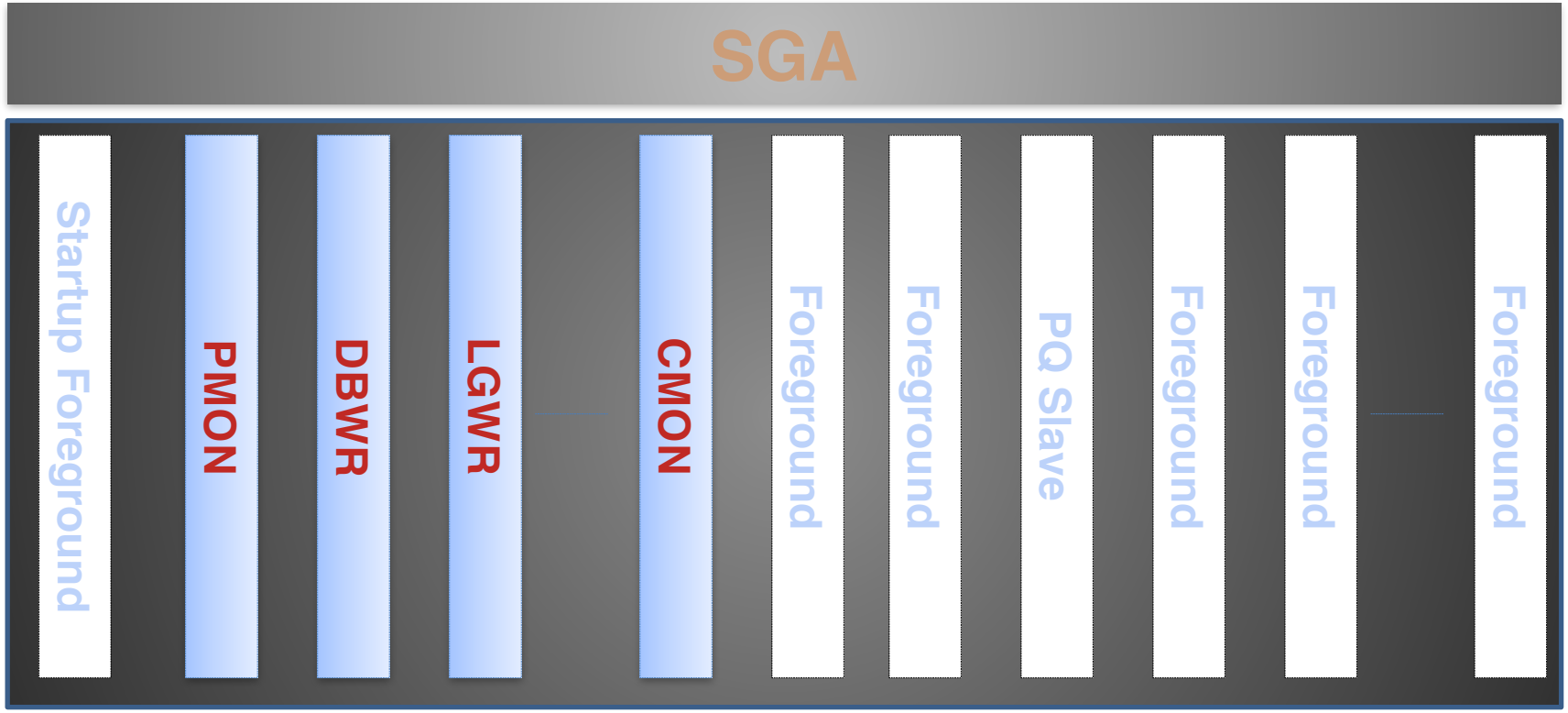
- Oracle instance is a collection of processes and shared memory
- All processes map the shared memory symmetrically
- ~90% of memory is used for caching DB blocks. Remaining used for various types of memory pools
- Memory can be dynamically moved between pools and buffer cache under pressure.
- Want to perform RDMA transfers from any process to any shared memory address.
- Large number of processes per node (20K+)

ORACLE PROCESS: CLASSIC EXECUTION MODEL

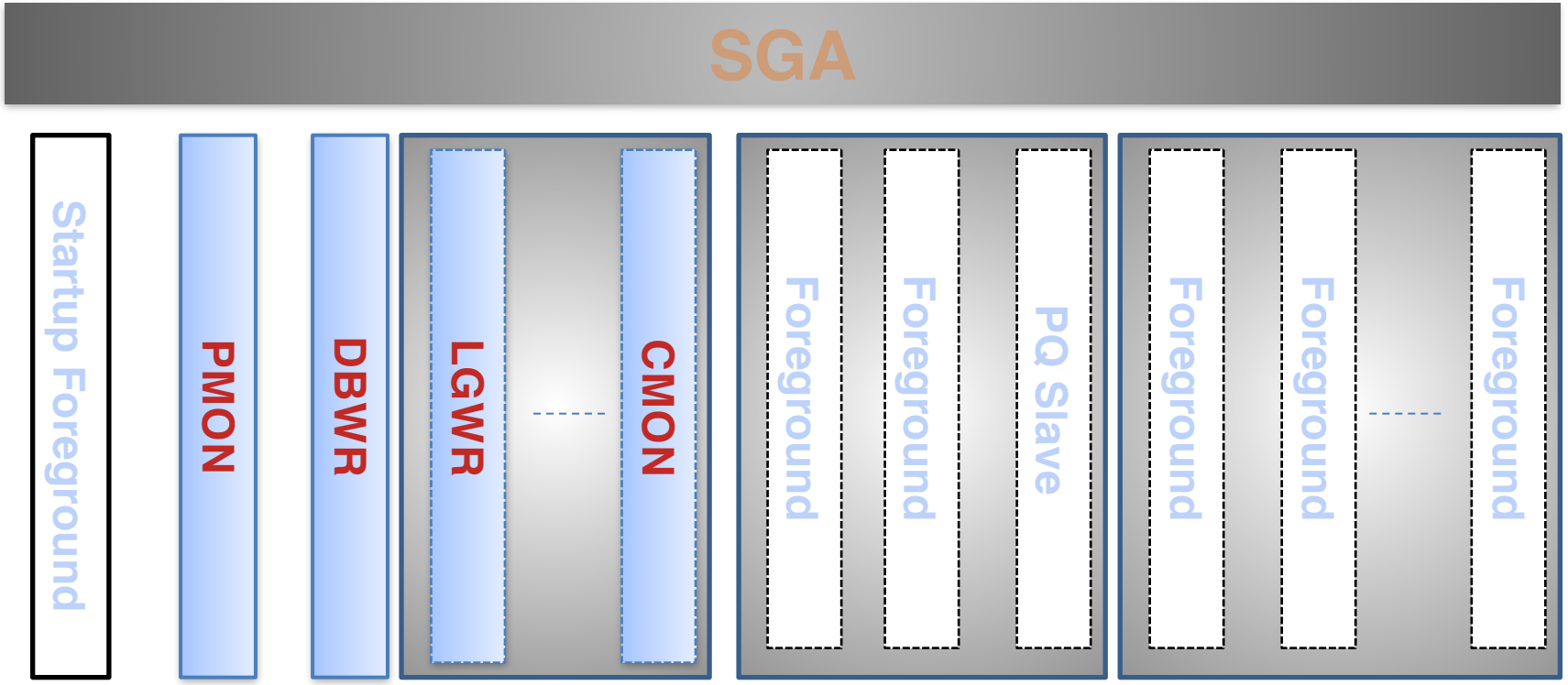


Solid lines represent OS process boundaries

ORACLE PROCESS: SINGLE PROCESS MULTI-THREADED MODE



ORACLE PROCESS: THREADED EXECUTION MODEL



SHARED PD VERB ORACLE USAGE

- **Attempting to register entire memory region in each process is not scalable.**
- **Hence Oracle uses shared PD support.**
  - Single process allocates a shared PD and registers all memory with it.
  - Other processes use the same shared PD with their IB context.
  - Allows sharing of memory registrations across all processes.
  - Each process still creates separate QP/SQ/RQ/CQ etc.
  - Re-use of memory mapping greatly reduces MPT and MTT entries for an instance leading to performance improvements.
  - PD is valid as long as there is one user process attached to it

SHARED PD VERB USAGE MECHANICS

PROCESS-A

1. Allocate a PD using *ibv_alloc_pd()*
2. Mark the allocated PD as shareable using *ibv_alloc_shpd()* which return a ‘shpd’ handle.
3. Store the shared PD handle in shared memory to make it accessible to other processes.
4. Associate the allocated PD for further resource allocation.

PROCESS-B

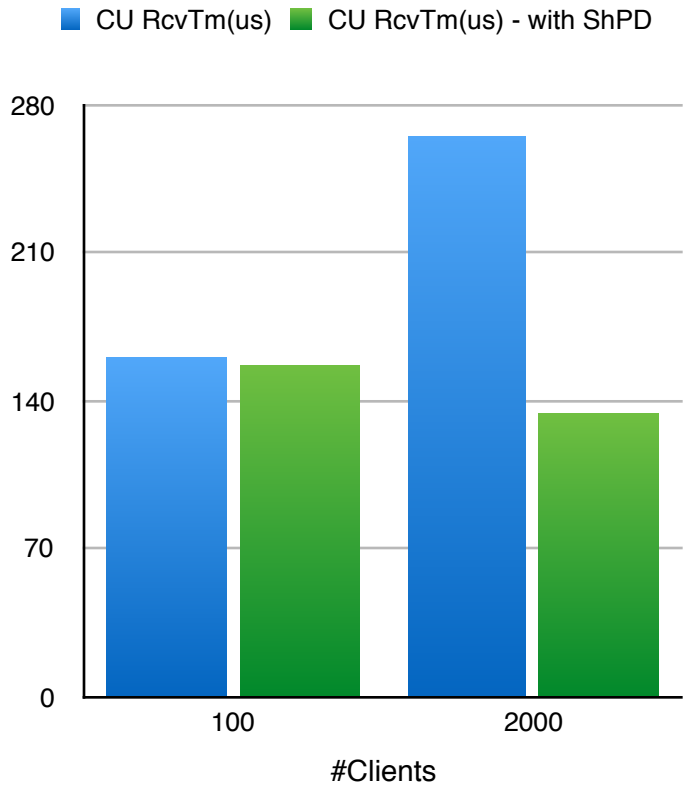
1. Use the shared PD handle and retrieve the PD using *ibv_share_pd()*
2. Use the retrieved PD for its own IB resource management.
3. Associate the retrieved PD for its own resource allocation.

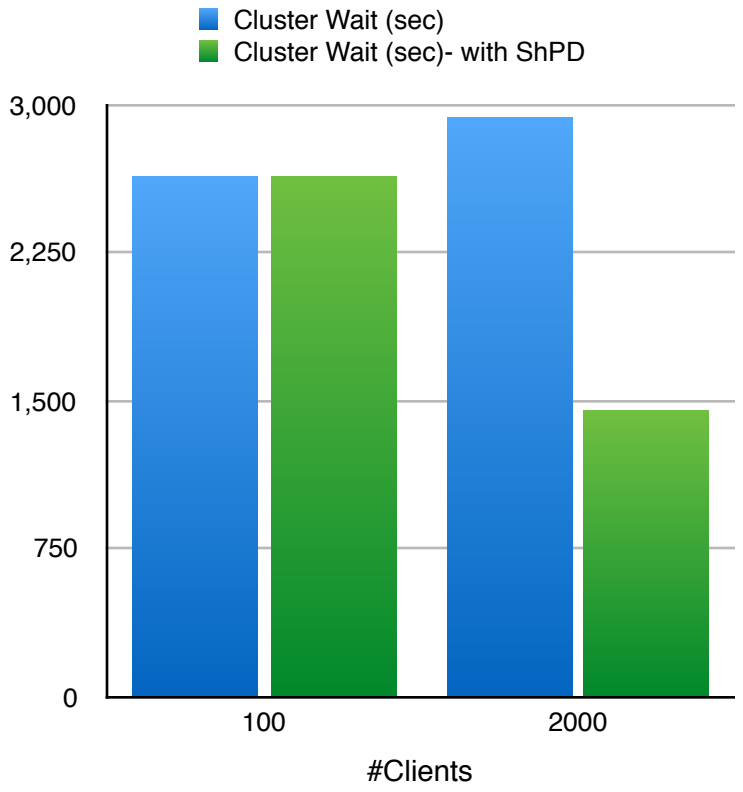
SHARED PD PERFORMANCE DATA : SETUP DETAILS

- **Database Cache Fusion Usecase**
  - **Node1(Exadata X5-2)**: 30 LMS Servers with 100 GB SGA: LMS is also called GCS(Global Cache Service)
  - **Node2(Exadata X5-2)**: 100/2000 Clients. Minimal local cache of 1 GB to maximize reading the Data blocks from remote LMS servers
- **Infiniband Fabric with Mellanox Connext-X3 HCAs**
  - Mellanox HCAs Memory protection & Address translation is managed using MPT and MTT tables.
  - HCA MTT & MPT caches can be considered as TLB caches.

SHARED PD PERF DATA: RX TIME, CLUSTER WAIT

Clients	CU RcvTm(us)	CU RcvTm(us) - with ShPD	Cluster Wait (sec)	Cluster Wait (sec)- with ShPD
100	159.97	156.01	2,639	2,634
2000	265.01	134.28	2,933	1,445





Remarks:

- CU RcvTime is the round trip time to fetch a block from remote instance (includes CPU processing)
- Cluster Wait Time is cumulative time waiting for network IO
- Receive time and Cluster Wait time is half with shared PD vs no shared PD
  - Benefit pronounced with larger number of clients

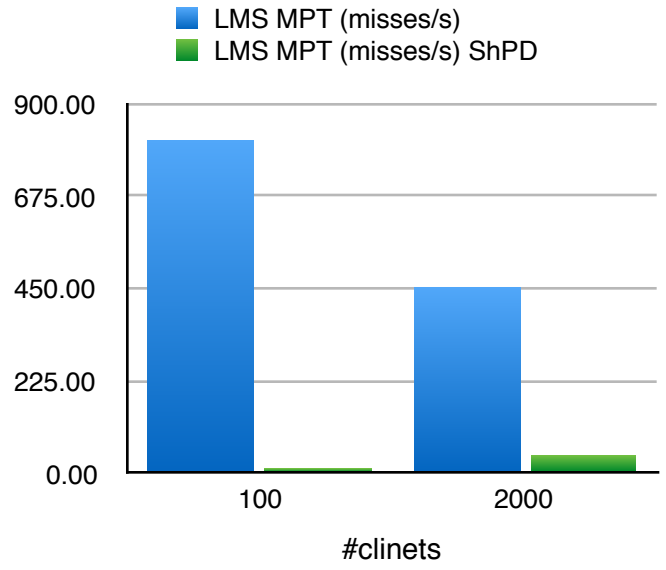
SHARED PD PERF DATA: MPT/MTT

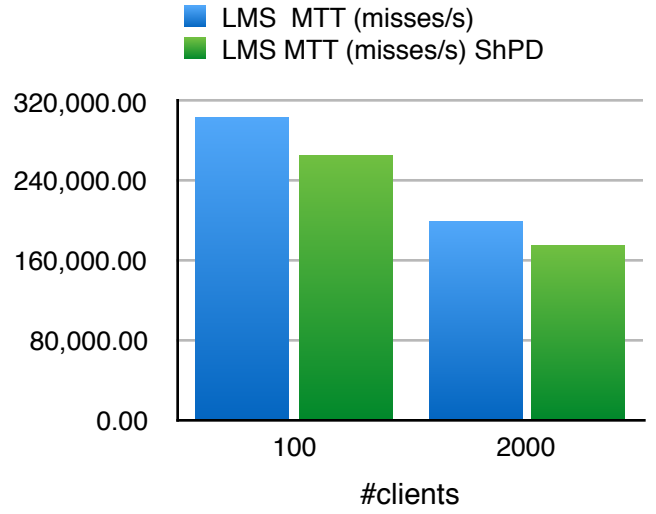
Clients	LMS MPT (misses/s)	LMS MPT (misses/s) ShPD	LMS MTT (misses/s)	LMS MTT (misses/s) ShPD
100	813.40	15.60	302,594.60	267,421.40
2000	454.20	45.20	198,924.60	176,391.20

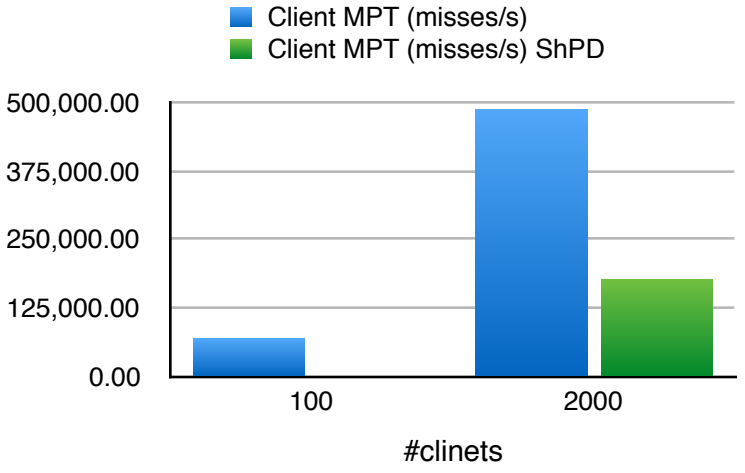
Clients	Client MPT (misses/s)	Client MPT (misses/s) ShPD	Client MTT (misses/s)	Client MTT (misses/s) ShPD
100	71,049.60	1,235.00	420,346.40	11,803.80
2000	486,088.20	173,697.00	494,301.80	385,486.80

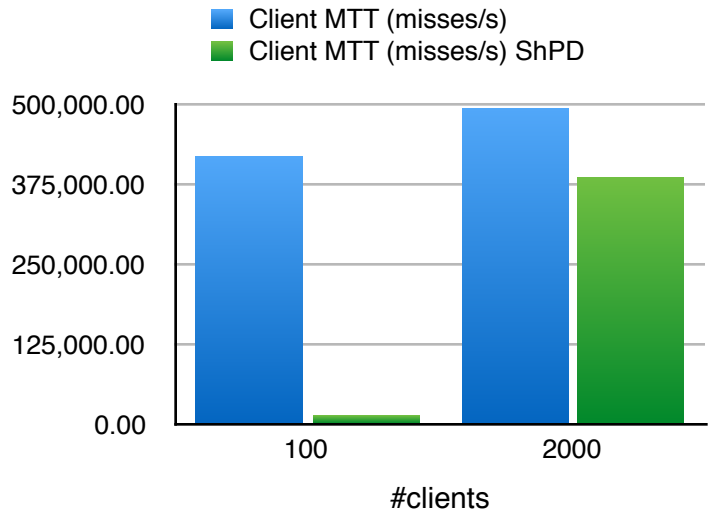
Remarks:

- MPT misses on the client reduced by 2.5X to 3X with shared PD
- 20% reduction in MTT misses









SHARED LARGE MR VERB USE CASE

- **Leveraging MTT cache with minimal entries by use of large/contiguous page(s) MR is well known optimization. We would like to exploit it further by using such large MRs across processes with a shared PD.**
- **Mellanox team floated the idea with couple of experimental verbs (ibv_exp_reg_mr/ ibv_exp_reg_shared_mr) but looks like they fail off the radar.**
- **Shared Large MR Verb:**
  1. Allocate and register a piece of contiguous memory using shared PD on process A
  2. Map/attach to the memory allocated in (1) in another process B that is also sharing the same PD
  3. Do direct IO operations to this memory in B using the shared memory registration i.e. don't want to duplicate memory registration/keys/mtt in each process sharing this region.
- **The shared contiguous memory needs to be mapped with the same address across all processes i.e. symmetric mapping of segments across processes.**

SHARED PD FURTHER USE CASE

- **Process private heaps created on symmetrically mapped region**
  - Automatically reclaimed on process exit
  - Re-use HCA mappings for region across processes whose heaps resides within region
  - Dynamically grow and register shared memory regions on demand
- **Allow building proxy communication models more efficiently**
  - Pool of background threads to progress communication on behalf of compute processes allowing for better compute/communicate overlap
  - Allow direct DMA access to private memory of processes (SHMEM symmetric heap allocated on the node).
- **Integrating with DPDK/SPDK infrastructure to share memory mapping and registration across large number of initiator processes**

CONCLUSION

- Shared Memory use is common across applications and these verbs improve the scaling for such use cases.
- The optimizations we are using/exploring are very generic, hardware agnostic and would benefit RDMA technologies. We have been shipping it with IB HCAs and also testing ROCE HCAs.
- We believe they are potentially good candidates as standard verb API extensions.

OPENFABRICS
ALLIANCE

13th ANNUAL WORKSHOP 2017

THANK YOU

Santosh Shilimkar, Avneesh Pant, Sumanta Chatterjee & Amarnath Jolad

ORACLE

BACKUP

SHARE PD API MAN PAGE(S)

IBV_ALLOC_SHPD(3) Libibverbs Programmer's Manual IBV_ALLOC_SHPD(3)

NAME

ibv_alloc_shpd - allocate unique id for sharing a protection domain (PD).

SYNOPSIS

```
#include <infiniband/verbs.h>

struct ibv_shpd *ibv_alloc_shpd(struct ibv_pd *pd, uint64_t
                        share_key, struct ibv_shpd *shpd);
```

DESCRIPTION

ibv_alloc_shpd() allocates a unique identifier required for sharing the PD pd with another process. share_key is a 64 bit key which needs to be provided with ibv_share_pd() call by another process to share the same PD in that other process. The argument shpd specifies a pointer to a user allocated area where libibverbs can write the unique identifier for the pd.

RETURN VALUE

ibv_alloc_shpd() returns shpd - a pointer to the area where the unique identifier is written - or NULL if the request fails.

NOTES

ibv_alloc_shpd() can be called on a particular PD only once.

IBV_SHARE_PD(3) Libibverbs Programmer's Manual IBV_SHARE_PD(3)

NAME

ibv_share_pd - share a protection domain (PD).

SYNOPSIS

```
#include <infiniband/verbs.h>

struct ibv_pd *ibv_share_pd(struct ibv_context *context,
                struct ibv_shpd *shpd, uint64_t share_key);
```

DESCRIPTION

ibv_share_pd() shares the protection domain specified by a unique identifier shpd for the RDMA device context context. share_key is the 64 bit key used to generate the unique identifier shpd. ibv_pds created using ibv_share_pd() can be deallocated using ibv_dealloc_pd(). Libibverbs keeps track of each instance of the shared PD and removes the PD from RDMA device when the last instance of the shared PD is deallocated.

RETURN VALUE

ibv_share_pd() returns a pointer to the shared pd or NULL if the request fails.

NOTES

Even though the same PD is shared by multiple contexts of an RDMA device or processes, the life span of each resource created in an 'ibv_pd' linked to a context or process is limited by the life span of that instance of 'ibv_pd'. e.g. The life span of an MR mr1 created under ibv_pd pd1 (which is an instance of shared PD shPD1) will end whenever pd1 is deallocated, even though underlying shPD1 may continue to live on. Sharing PD is not supported among 'ibv_context' created for different RDMA devices.