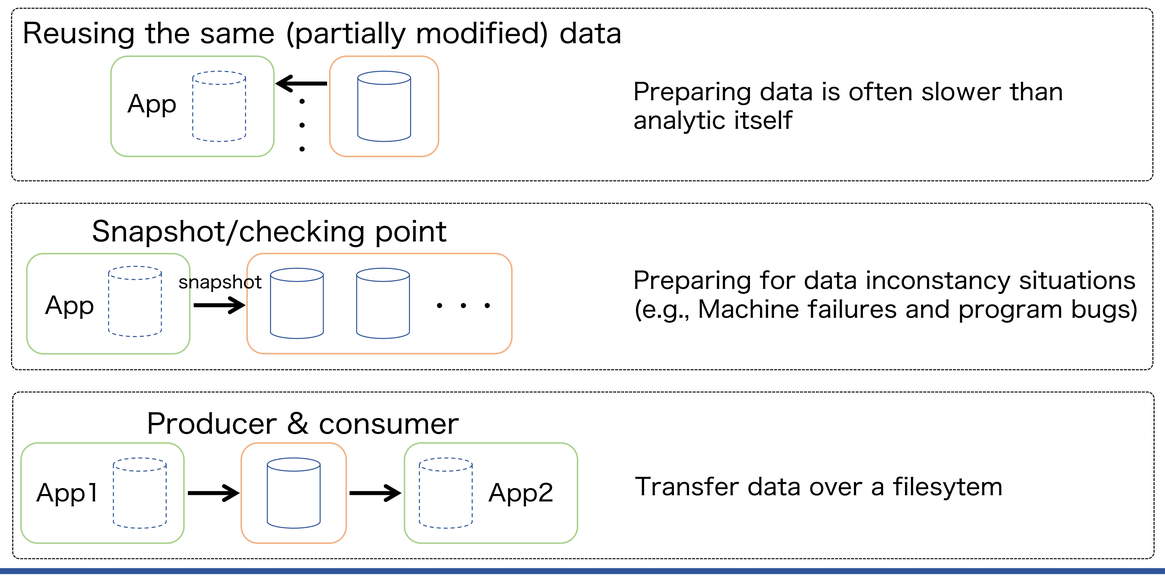
Simplified Interface to Complex Memory (SICM)

Metall: A Memory Allocator for Persistent Memory

Keita Iwabuchi, Roger Pearce, and Maya Gokhale (CASC, LLNL)

1) Storing Data Beyond Single Execution Time To Accelerate Exascale Data Analytics



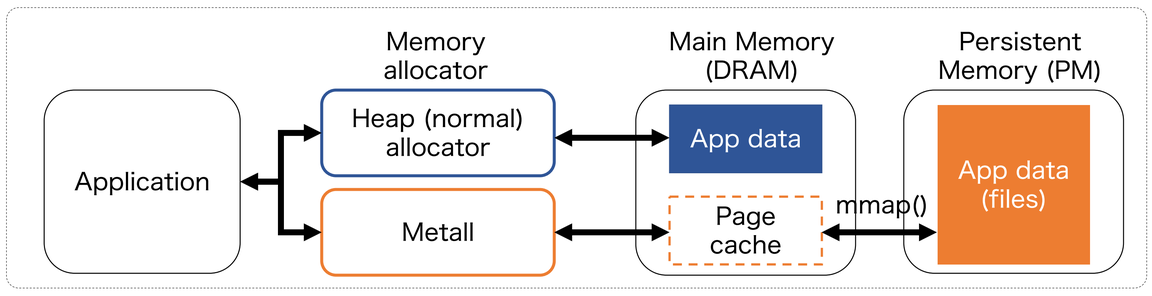
Challenges

- Storing/reattaching complex data structures from/to persistent memory
  - Not only 1-D array, e.g., std::map<int, std::vector<int>>
  - Can change capacity dynamically even after reattaching the data
- Various types of persistent memory technologies in exascale machines
  - NVDIMM (Intel Optane DC PM), local and global filesystems
- Efficient snapshot
  - Just copying whole 'exascale' data is expensive

Goal

To develop an allocator for file-backed PM and design a new data management scheme for exascale applications

2) Metall: A Memory Allocator for Persistent Memory



- Enables applications to allocate heap-based objects into PM, just like main-memory
- Metall leverages a memory-mapped file mechanism (mmap)
  - Applications can access mapped area as if it were regular memory
  - mmap can map files bigger than the DRAM capacity
- Incorporates the state-of-the-art allocation algorithms
  - Some key ideas from SuperMalloc[Kuszmaul'15]

API

- C++ API developed by Boost.Interprocess to enable applications to allocate custom C++ data structures in PM with the C++ standard style
- Provides space efficient and fast snapshot capabilities

`void* allocate(size_t n)`	Allocates *n* bytes
`T* construct<T, Args>(char* name) (Args... args)`	Constructs an object of *T* stores its address with *name*
`T* find<T>(char* name)`	Find an already allocated object by *name*
`bool snapshot(char* dir_path)`	Make a snapshot of the current status to *dir_path*

Examples

Allocate and reattach int object

create.cpp

```
void main () {
 metall::manager metall_mgr(metall::create_only, "/ssd/test");
 int* n = metall_mgr.construct<int>("val0")();
 *n = 10;
}
```

open.cpp

```
void main () {
 metall::manager metall_mgr(metall::open_only, "/ssd/test");
 int* n = metall_mgr.find<int>("val0").first;
 std::cout << *n << std::endl;
}
```

Terminal

```
$ ./create
$ ./open
10
```

Allocate and reattach STL vector container object

```
using vec_t = vector<int, metall::allocator<int>>;
```

create.cpp

```
void main () {
 metall::manager metall_mgr(metall::create_only, "/ssd/test");
 vec_t* pvec = metall_mgr.construct<vec_t>("vec")
                         (metall_mgr.get_allocator());
 pvec->push_back(10);
}
```

open.cpp

```
void main () {
 metall::manager metall_mgr(metall::open_only, "/ssd/test");
 auto pvec = metall_mgr.find<vec_t>("vec").first;
 std::cout << pvec->at(0) << std::endl;
}
```

Terminal

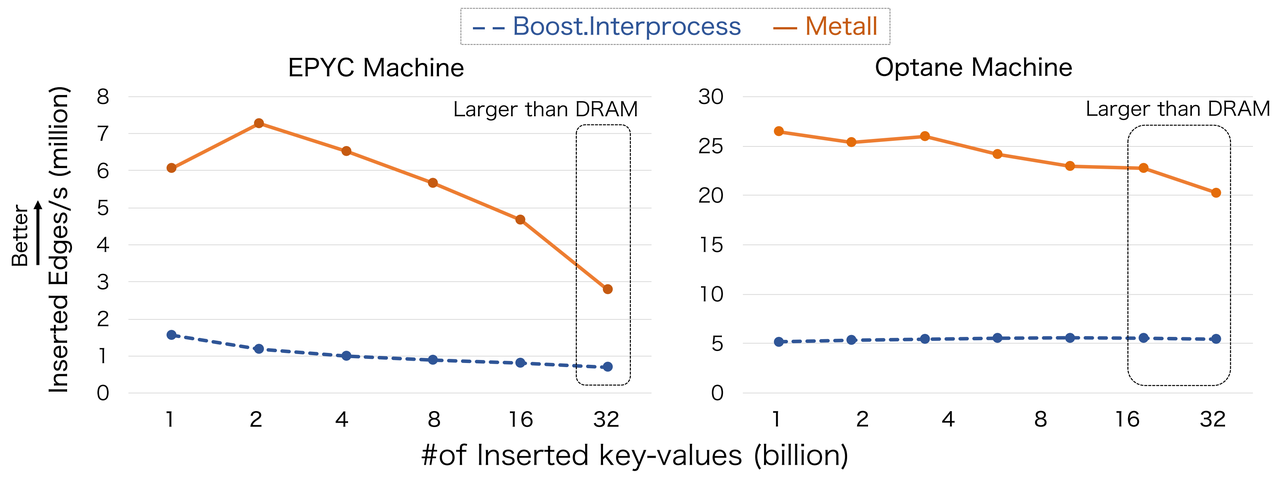
```
$ ./create
$ ./open
10
```

3) Evaluation

EPYC	
Storage	NVMe SSD with XFS
DRAM	256 GB
CPU	AMD EPYC CPU x 2 (96 threads)

Optane	
Storage	Intel Optane DC Persistent Memory x 2 with Memory Mode
DRAM	192 GB
CPU	Intel Skylake x 2 (96 threads)

Simple KV Store Construction (evaluate memory allocation speed)

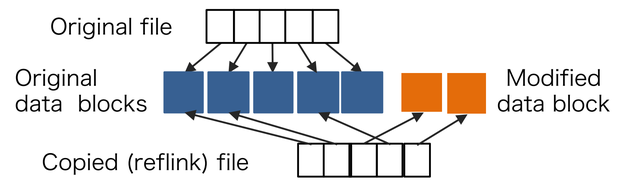


- Metall works well on both PM configurations
- Recent PM technologies are really fast!

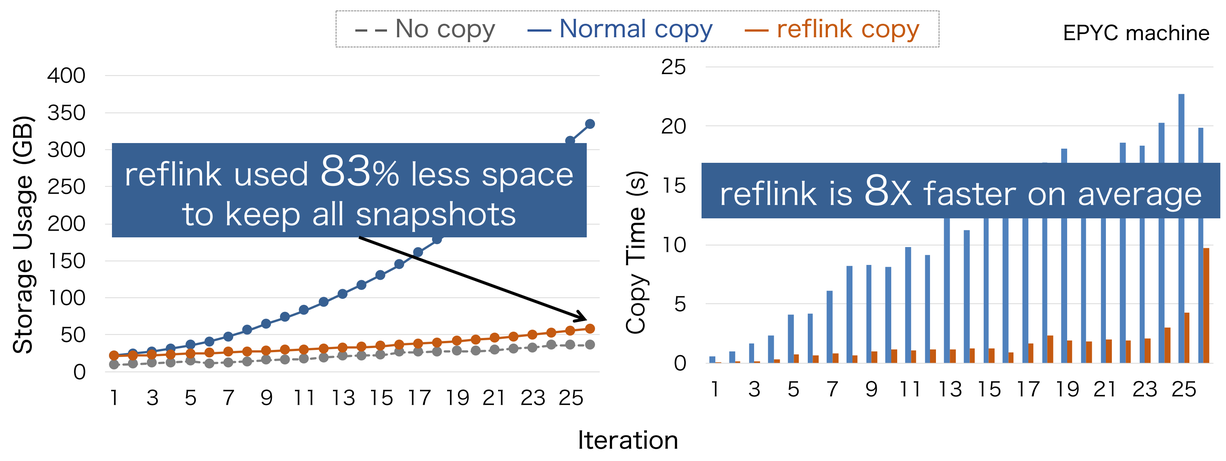
Memory Efficient Snaphost

– Snapshot (copy) backing-files to another location using **reflink**

- reflink (copy-on-write file copy)
  - –Implemented in XFS, ZFS, Btrfs, and Apple File System (APFS)



- Workload (incremental key-value store construction)
  - Take a snapshot after inserting each chunk (64M edges)
  - Insert edges in the original KV store



Useful to implement a simple and light data consistency support

i.e., take a snapshot when opening the data in case of data inconsistency situations (e.g., process failure and coding bug)

2020 Exascale Computing Project Annual Meeting / Feb 3–7, 2020 / Houston, TX

Prepared by LLNL under Contract DE-AC52-07NA27344 (LLNL-POST-802162). Experiments were performed at the Livermore Computing facility.

Metall is available at github.com/LLNL/metall