



Thinking Persistently Designing applications for persistent memory

Adrian Jackson

EPCC

a.jackson@epcc.ed.ac.uk

@adrianjhpc 

Persistence window



- Key concept is persistent window
 - When is data volatile? when is data persistent
- What is required to recover from application/hardware failure?
- What is required for your application?
- `pmem_persist()` or `pmem_flush()` `pmem_drain()` required to ensure data is on disk
 - None of these are atomic!
 - Failure mid drain could result in undefined data state



Volatile B-APM usage

- For correctness nothing special is required here, you can simply use the B-APM as volatile memory
- If you are using pmdk pmem then you will want to clean up files after your application finishes
- Otherwise you will take up space on the device(s) even after reboot
- Performance considerations are important
 - Asymmetry of read and write may have a big impact
 - Caching may help for write, but it depends on your access pattern (write after eviction requires read first)



Persistent B-APM usage

- Strategy needed to recover data on failure
- Transactional approach
 - Use higher level pmem library functions
- Application logic
 - Using low level pmem functions
- Main focus is hardware failure
 - i.e. reboot but memory still intact
- Data resiliency another issue
 - What if an NVDIMM fails
 - Using low level pmem functionality there is no automatic redundancy
 - No RAIDing

Consider what your application currently does



- Can your application cope with failure during I/O at the moment?
 - Failure during POSIX I/O can easily lead to files in mixed states
 - Failure even after I/O routines finish may lead to mixed states (O/S caching)
- However, persistent memory usage is likely to change your application design
 - So consider this from the beginning

How to ensure consistency?



- Simplest option is to double up on key data:
 - Work on current data in one set of arrays/variables
 - Have previous iteration/timestep/update of data in another set of arrays/variables
 - Once current data is persisted set a persistent variable to indicate which is correct
 - Then switch to the other set of arrays
- This involves potential data copies and doubling memory requirements, so may not be desirable...



How to ensure consistency?

- Read from DRAM/pmem, write to DRAM
 - Persistent to pmem
 - Block dataset and keep working set in DRAM
 - Persist as required
 - Only persisting a subset of data at any one time, so only require duplication of this subset
- Finer granularity
- More sensible performance choices

How to ensure consistency?

- Guard persist calls with variable change, i.e.:

```
checkpoint_flag = -1
pmem_persist(checkpoint_flag, 1)
pmem_persist(checkpoint, 100000);
checkpoint_flag = 1
pmem_persist(checkpoint_flag, 1)
```

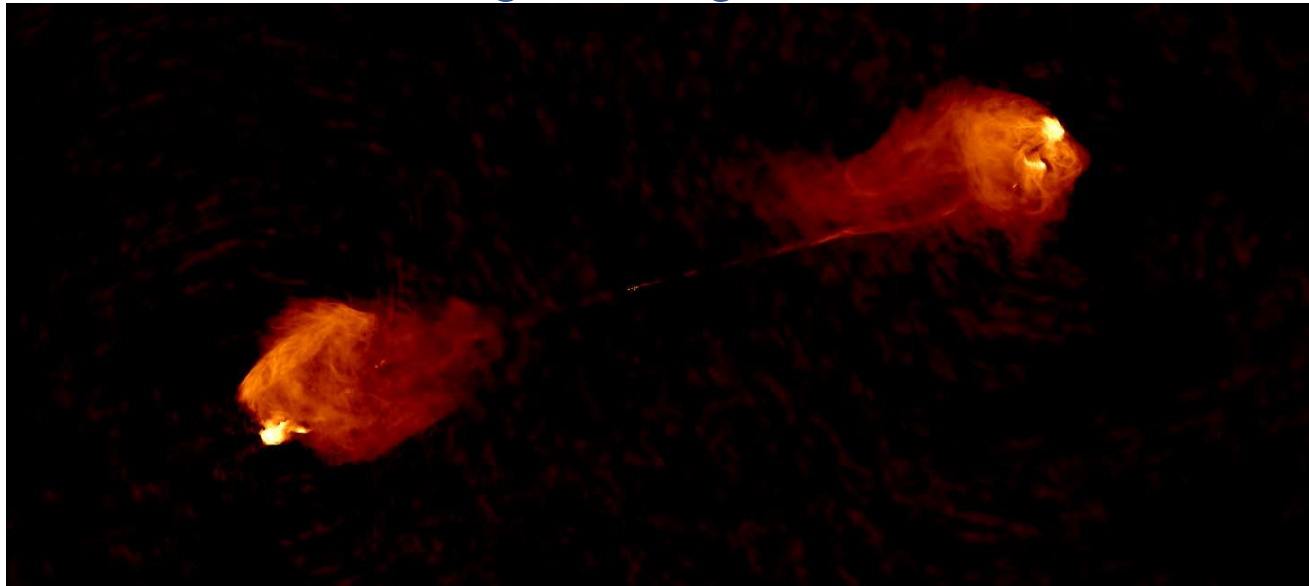
...

```
Load checkpoint flag
if(checkpoint_flag == 1){
    read_checkpoint()
}
```


How to ensure consistency?



- Use B-APM for read only datasets
 - Might sound silly, but for a range of applications may be sensible
 - Bio-informatics, ML, image processing
 - Large datasets that don't get changed



Persistent safety



- As with filesystems, some level of redundancy is required to ensure data is safe
- This could be copies of data
 - Both within a single run, and across runs
- Cleverer safety options are possible
 - Erasure coding
 - Append write
 - Mirroring
- Consider using a higher level product
 - i.e. DAOS



Visibility and Persistency

- Coherency != Persistency
 - There are differences between threading and memory coherency (visibility) and persistence coherency
- Persistency coherency not enforced in hardware
 - Changes to the same non-volatile memory location from different threads
- Avoid multi-threaded *persisting* unless you really know what you're doing

Summary



- Working out what to put in NVRAM, when to persist, and when you can be sure something is safe is key
 - This is the main challenge for using B-APM
- Using as a filesystem removes this issue
 - Because POSIX
- For true B-A work filesystems will reduce achievable performance
 - Better off buying SSDs
 - But potentially a sensible stop-gap approach (think OpenMP rather than MPI)