===
SoCC 2017 Review #148A

Paper #148: Distributed Shared Persistent Memory

Overall merit: 4. Accept
Reviewer expertise: 3. Knowledgeable

===== Paper summary =====

The authors present a system that provides the abstraction of distributed shared memory. They implement their solution in-kernel and have an extensive evaluation. Several of the ideas of the paper are being actively discussed in the community and the evaluation lacks comparisons with existing systems, however this is a good contribution to the body of work.

===== Comments for author =====

The paper is well written. The ideas are good, however for this paper to be a Strong Accept it would have been good to quantitatively compare with existing state-of-art systems, like FaRM.

At this point we have several different solutions to the same basic problem (as seen from the 90+ references, and I still don't feel like we are making major progress. Part of the problem is lack of real hardware, forcing the authors to use normal RAM and guess how the performance would look with NVMs.

The evaluation has many strong parts and uses real traces and real systems (graph engine and a NoSql database). Just to contrast with an SOSP paper: what makes the difference for SOSP would be a comparison with a competing system like RAMCloud or FARM. For SOCC that is not necessary, but it would have been a plus.

Minor: I find the title too broad. Several systems address the same problem. The title should reflect something unique about your work.

===
SoCC 2017 Review #148B

Paper #148: Distributed Shared Persistent Memory

Overall merit: 4. Accept
Reviewer expertise: 3. Knowledgeable

===== Paper summary =====

This is paper present Hotpot, a system offering a distributed shared memory interface on top of persistent non-volatile memories. Applications can access data in Hotpot using standard load/store instructions, while in the back HotPot uses paging hooks to synchronize data across the nodes and further makes sure local cached copies are always consistent. Explicit operations to commit/persist data are also provided, as well as fault tolerance implemented via replication. The system is evaluated using two applications, MongoDB data and PowerGraph.

The system as described in the paper looks complete and takes care of every aspect one can think of. One argument that can be made is that Hotpot looks like an incremental extension from their earlier work Mojim, which is primary/backup replication system for non-volatile memories.

The other potential problem is that Hotpot (like Mojim) is designed for non-volatile memory that can be attached directly to the memory bus and accessed via CPU load/store instructions. Given that such hardware doesn't exist in production they used DRAM to evaluate their system. However, DRAM obviously has lower latencies, maybe a factor 10 or more lower than future NVMs. That may have implications on the performance and maybe even on the system design. Those implications are not evaluated and captured in this paper. For instance, the paper uses an RDMA based network stack to access remote data. With slower NVMs directly attached to the memory bus, RDMA might be challenged as it has to buffer incoming data somewhere if it cannot be placed fast enough.

All in all, however, I still think this the paper is interesting and above the bar to be accepted.

===
SoCC 2017 Review #148C

Paper #148: Distributed Shared Persistent Memory

Overall merit: 3. Weak accept
Reviewer expertise: 3. Knowledgeable

===== Paper summary =====

The paper presents a distributed shared persistent memory system. Using this system applications can be ported to set data on this persistent storage, avoid data marshaling and still get persistence guarantees. The paper presents benchmarks that shows performance improvements in some cases even relatively to volatile local storage.

===== Comments for author =====

The paper deals with an interesting issue and most of the implementation seems reasonable. Yet, there are some design choices that do not seem too reasonable. The lack of proper baseline in the evaluation, i.e., configuration which performs _better_ than Hotpot seems worrying.

The first technical issue is regarding "shadow-paging", which the system claims to use. According to the paper, hotpot allocates a new page on write, writes the modified data and only then changes the pointer (PTE?). This design does not sound reasonable, and I even doubt whether the implemented system does so. It would require an instruction emulator to emulate the instruction that was trying to perform the write, which is hard and expensive. Instead, it makes more sense to just do copy-on-write and restart the write operation after the pointer was changed.

Second, the paper claims to ensure that the persistent virtual address range is alway available by changing brk to exclude the persistent memory. This is not enough. The loader may still load the text or the stack of the application to the memory to the persistent memory address (especially when ASLR is on).

Third, it seems that MRMW does not provide cache-coherency in the common meaning: it seems that two threads can write to the same page at the same time, and that the changes that one of the threads made can be lost (Section 4.3.2: "The new physical page... will not affect the DN's dirty data"). In this case, it is not clear how existing applications can be adapted.

Fourth, the reasoning behind not dealing with concurrent failure of the CN and ON is not exactly convincing. Even if the time it takes for a commit to take is short, the question what the probability for a failure remains open. If commits are very frequent, or if commits involve multiple ONs, the risk of two nodes failing is not negligible. Without quantifying it, this safety relaxation may not be acceptable.

Finally, there are many small details and claims which do not fully make sense. The benefit from using a "cached" copy as a replica is not clear, especially as it can affect endurance (if the local copy is on PM instead of DRAM). The interface, which uses "MAP_HOTPOT" sounds somewhat inappropriate. Hotpot is claimed to exploit PMs byte addressability, but I do not see how. Avoiding eviction of dirty pages can cause high latency when eventually the pages are evicted and as a result high tail latencies; that is one of the reason for OSes to evict them gradually. Hotpot is claimed to run threads across machines, yet there is a need to clarify exactly what it means; the threads are probably very limited in what they can do, for example - what happens if one of the threads tries to kill another thread (on a remote machine)?

The benchmarks are not too convincing despite Hotpot good performance. It is somewhat worrying that in all the benchmarks the alternatives, even those which do not guarantee persistence, perform worse than HotPot. It usually indicates that the wrong baseline was chosen. A proper baseline could have been using heap memory instead of tmpfs. Even better, evaluations using PMEM/DAX could have been beneficial. The performance numbers of MongoDB seem rather poor, apparently since MongoDB 3 was not used.

Nits:
- NoSql -> NoSQL
- "we propose to use use" - remove one "use"
- "applications that use memory-mapped I/Os" - should be "memory-mapped files"
- "load/store access access" - remove one "access"
- "V2.7.0" -> "v2.7.0"
- "we separate the MRMW commit process into three phases, similar to traditional two-phase commit protocols" - ??!!