# Experiment Changes

1. Review A
**Q:** How about comparing with FaRM?
**A:** We will compare with FaRM in our final version. But since FaRM is not open-source, we will only be able to use the raw numbers from FaRM. To have a fair comparison, we will setup a testing environment that's as close as possible to original FaRM settings and re-evaluate some microbenchmarks.

2. Review C
**Q:** It is worrying that Hotpot performs better than any other systems listed in paper.
**A:** We are implementing a multiple-reader single-writer (MRSW) DSM system in kernel on top of the same networking stack as used in Hotpot. This DSM system uses release consistency, which is close to the coherence consistency of Hotpot. We will also port our graph engine to it. We are close to finishing the implementation of this DSM system. We believe that comparing with a DSM system on top of the same networking stack is the best way to demonstrate Hotpot's performance pros and cons. We also plan to open source this DSM system, together with Hotpot.

# Writing Changes

1. Reviewer C
**Q:** What does MRMW cache-coherency mean: can two threads write to the same page at the same time? If not, how to adapt existing applications?
**A:** Multiple threads within one process in Hotpot work in the same way as normal multi-threading and thus support existing multi-threaded programs. NVM is attached to main-memory bus, so it will use the same hardware CPU cache-coherence mechanisms as DRAM. Notice that the cache we refer to is not CPU cache, but a local cached copy of remote data. We will make this clearer in our final version.

2. Reviewer C
**Q:** Hotpot does not take endurance into account.
**A:** There are work that look into improving NVM endurance and there are also papers saying that many types of NVM are much more endure than flash and all. So currently, we are not addressing endurance and will consider it in the future. We will make this clearer in our final version.

3. Reviewer C
**Q:** Shadow-paging is not used properly in paper.
**A:** In implementation, we are doing COW, the write is resumed after the new page is created. This means we are not able to first writing out-of-place. We will change this in our final version.

4. Reviewer C
**Q:** Can persistent virtual address range still be available in next run? Especially with ASLR.
**A:** Hotpot saved persistent virtual address range is guaranteed to be totally within brk heap vma, which is always above .text segment. If this range conflicts with stack vma, kernel will have a fault and exit. So both loader and kernel runtime will ensure that text, stack, persistent range will not conflict. Furthermore, if ASLR is on, during the next run, our modified loader will make sure that the new randomized brk will not exceed the saved persistent virtual address range. We will make this clearer in our final version.

5. Reviewer C
**Q:** What does MAP_HOTPOT mean?
**A:** MAP_HOTPOT is a special flag to tell Hotpot that we are creating a new persistent virtual memory range across nodes. All online nodes will communicate and have a consensus, after which this agreed persistent virtual memory range will be saved, and used by our modified loader in next run. We will make this clearer in our final version.

6. Reviewer C
**Q:** How Hotpot distribute threads across machines?
**A:** Currently, Hotpot uses a shell script in CD to start applications in different nodes. The shell script will pass arguments to different nodes, and application itself can start different threads based on passed arguments. The only requirement of this model is that the binary of the application needs to be present in all nodes. Rather than sending code to different nodes, our current model is easier to deploy and test. We will make this clearer in our final version.

## Changes we'll do if we have time

1. Reviewer C
**Q:** Evaluations using PMEM/DAX could have been beneficial.
**A:** We find that pmem library has several APIs have similar semantics with Hotpot. Specifically, `pmem_map()`, `pmem_persist()` and `pmemlog_appendv()` can be used to port MongoDB. We will try to finish the porting and compare it with Hotpot MongoDB if we have enough time.

## Changes that we won't be able to do

1. Reviewer C
**Q:** Should handle concurrent CN and ON no-pm failure.
**A:** The key to do recovery of Hotpot xact is to find a complete redo-log or an undo log. The CN has a complete redo-log, the logs from all involved ONs combined together is a complete log, which is the same as CN's. So, it means, 1) if only CN fails, we could reconstruct a complete redo-log from all ONs, 2) if only some ON fail, we could reconstruct a complete redo-log from

CN. However, if CN fails and at least 1 ON also fails, then we are not able to reconstruct a complete redo-log.  I'm afraid that we won't have enough time to change our transaction implementation (to make an extra redo-log copy to another node, or persist to CN local disk), and have a complete re-evaluation.

2. Reviewer B
**Q:** What if future NVM is 10x slower than DRAM? There will be implications on system design.
**A:** Hotpot is built based on the assumption that future NVM will have similar read latency as DRAM, and has a 2x write latency compared to DRAM. We agree with reviewer B that if NVM is 10x slower than DRAM, then it may have lots implications on Hotpot design. But I'm afraid we don't have enough time to address this issue.

3. Reviewer C
**Q:** Why MongoDB v2 is used instead of MongoDB v3?
**A:** We chose to use MongoDB v2 because we already have an in-house modified version from our previous work Mojim. MongoDB MMAPv1 storage engine is a natural fit for Hotpot. The new WiredTiger engine will create a separate file for each new table, which does not suit Hotpot's goal. MongoDB v3 also supports MMAPv1 engine, but I'm afraid we won't have enough time to port MongoDB v3.

4. Reviewer A
**Q:** How about comparing with RAMCloud?
**A:** It would be interesting and valuable to compare RAMCloud and Hotpot. Unfortunately, the limited set of applications that RAMCloud currently support are not a good comparison for Hotpot and we are afraid that we won't have time to port applications to RAMCloud before the camera-ready deadline.

## Questions we already answered in the paper

1. Reviewer C
**Q:** The benefit from using a "cached" copy as a replica is not clear.
**A:** "cached" copy in PM means a persistent cached copy, which can be remained in PM as an extra replica for original data (ON). In this way, replica can be created implicitly during runtime, instead of bulk creation at xact-commit time, to meet the user-required rep-degree.

2. Reviewer C
**Q:** Avoiding eviction of dirty pages can cause high latency when eventually the pages are evicted and as a result high tail latencies.
**A:** Hotpot targeted application has well-defined sync-point, e.g., do some change, and do commit. This implies that applications themselves will commit dirty pages. If Hotpot chose to evict dirty pages, there is a high chance that this page will be accessed or committed by applications in the near future, causing thrashing and unnecessary performance overhead.

3. Reviewer C

**Q:** It is worrying that Hotpot performs better than other alternatives, even tmpfs which does not persistence guarantees. Why not use heap instead of tmpfs as baseline?

**A:** Tmpfs is slower than Hotpot even though tmpfs does not make any data persistent, because tmpfs uses the slower MongoDB replication mechanism on IPoIB. MongoDB has persistence requirement. Tmpfs is the fastest that one can get in making data “persistent”.

4. Reviewer C

**Q:** How is PM byte addressability exploited in Hotpot?

**A:** After mmap(), applications can use direct load/store to access global shared persistent memory within mmap() returned virtual address range. It is showed in Figure 5: sample code using Hotpot.