PERC: Persistent, Efficient, Recoverable, Consistent
Research Proficience Evaluation Proposal

Tony Mason[*]

Department of Computer Science, University of British Columbia

Abstract

This is a research proposal focused on advancing my research area and satisfy the **Research Proficiency Evaluation**, which is a qualifying requirement for the PhD program in Computer Science at the University of British Columbia.

PERC is a study of the interactions of persistence and durability with respect to the application of byte-addressable non-volatile memory (**NVM**) when added to high performance key-value stores (**KVS**). The issues to be explored include specifying a clear behavior model for NVM, with an emphasis on the class of failures that are anticipated and must be handled for *crash consistency*, evaluating the changes that must be made to accomodate various durability guarantees (write, explicit sync, epoch) and the impact of those changes upon performance of existing KVS. The expectation is that the work will provide greater insight into the tradeoffs in the various approaches.

1 Committee

The following individuals have graciously agreed to serve as my committee for the purposes of the Research Proficiency Evaluation:

Norm Hutchinson
Alexandra Fedorova
Andrew Warfield

[*]fsgeek@cs.ubc.ca

2 Introduction

Byte-addressable Non-volatile memory (**NVM**) that is directly connected to the memory bus, such as **NVDIMM-P** has been anticipated for many years but is just now emerging as actual, usable hardware. Prior NVM solutions have consisted of either *Block addressable non-volatile Memory* — typically flash, or battery backed up DRAM with block addressable non-volatile memory for persistent storage. These prior NVM solutions do not behave like directly accessed NVDIMM.

There is a body of existing work that suggests how NVM can be used to improve performance of performance-sensitive usages, including work done via a simulator. However, this work suffers from weaknesses including: a lack of actual hardware and a naïve model for failure. Understanding NVM and how to correctly and efficiently use it will be important as we begin integrating it into critical systems infrastructure.

In this work, I propose analyzing the behavior of real systems that include NVM in order to understand the operational characteristics of that memory, identify a workable model for failure analysis that is consistent with the available information, identify potential ways in which NVM can be utilized to provide persistence, efficiency, recoverability, and consistency, and evaluate the tradeoffs inherent in integrating NVM into performance critical software.

3 Background

The motivation for this proposal is based upon my general research area. The RPE supplement describes my original broad research direction. The exploration of that research direction has led to this current proposal. Since that time I have been exploring more specific areas, with an eye towards refining my understanding of the research area. The initial work was an exploration in the general issues around extending file system functionality; this resulted in the Finesse project, which evaluated using FUSE as an extensibility tool, by combining support for the existing VFS interface with a non-system call approach that utilizes message passing to access new functional interfaces. That work was successful in demonstrating the ability to extend the interface, but was not persuasive. Initial external feedback sees merit in the idea, but noted it is not sufficiently mature.

There are a number of challenges to the Finesse approach:

- FUSE has no "pass-through" mechanism (e.g., **ioctl**).
- Message passing on Linux is restrictive. Existing mechanisms are viable, but severely limited (e.g., POSIX message queues) and require that we do considerable work to add functionality via this mechanism.
- Despite the vast number of FUSE file systems in existence, most are limited and not easily extensible.
- FUSE is itself not particularly well engineered for adding a dynamic extensibility model.

There are certainly a number of research opportunities within the FUSE area. However, this does not further my origianl research goal. While I am not adverse to a change in direction, doing so requires a compelling reason to do so. Thus far, I have not found a persuasive argument to do so; I prefer to think this is due to discipline rather than stubbornness.

Directly related to the original research area, I have been mulling over various models of data representation and engaging in "thought experiments" in which I considered what a file system that supports semantics might do. In early April I spoke with another PhD student in the deparment, Francesco Vitale, who is doing research in HCI. His research area is personal digital data; I tend to think of this as data that is created based upon an individual's own actions. Thus, it would be distinguished from automatically created log data, for example. He provided me with a number of HCI references that strongly support my concern about hierarchical file systems not meshing well — in essence, users's have been trained to implement their own organizational models for their data within the interface provided *despite* the clear case that it is not a good fit. The HCI community is constrained by what they can actually do within the existing systems model. It is useful to note that this type of data relationship modeling is not a single system problem; if anything, the problem becomes vastly more interesting when we look at it as one that must cross system boundaries. User data is not generally limited to a single system. I do not know what the model looks like right now, but a general model that enables creating general, cross-realm relationships is critical for such an approach to be broadly useful.

Thus, my present thinking about how to approach the broader problem: I need to have a more flexible file system that permits me to work with the HCI community to model different ways of creating data attributes. The approach I am currently considering is to model data as a *graph*. The model then is one in which we have files (vertices) and relationships (edges). This permits a mapping of the traditional hierarchical file system onto such a graph. There is a rather small body of work for graph file systems in existence; most of the work is graph databases.

A common approach to implementing graph databases is to utilize a key-value store (**KVS**). This observation piqued my curiosity about KVS. There has been tremendous interest in KVS recently, with a half-dozen major systems papers on this topic over the past twelve months. Thus, there is a broad base of interesting information on constructing a KVS.

One area of KVS that has not been as well explored is their intersection with byte-addressable non-volatile memory (**NVM**). When I first started working with Andy, the area of interest was in using NVM as a mechanism for acellerating file systems operations by optimizing the write path (validate, log, return). It was this work that ultimately led me to the Finesse work; essentially all of my prior file systems work has been *in-kernel* on both UNIX and Windows (but not Linux).

The original design of that project ("Intentions") involved a key-value store as well, with the intention that it would be able to utilize NVM as it became available. Hence, returning to this area for consideration of this project was logical in terms of the flow of areas that I have been researching for over a year now.

Thus, the "bigger picture" for this project is that by constructing a key-value store, I have a natural basis upon which to explore the Finesse-style extensions. For example, adding a traditional key-value interface would be one logical extension. Another might be to explore a model in which the file system is itself implemented in a distributed fashion — an area that I discussed with Andy at one point — so that different nodes could each take a delegated region of the file system to manage. This would look very much like a distributed file system on a single system.

Another area to explore is an optimized IPC mechanism. I have taken inspiration from the recent work on delegation at SOSP 2017. [**?**] Combining this with a shared memory IPC mechanism could definitely provide a useful improvement to the project as well as be useful for some of the other work envisioned.

4 Proposal

For this project, I choose to focus on one aspect of the larger research area. One motivation for this is to ensure that it constitutes a research area that I can undertake for a moderate period of time (four months) that will yield quantifiable research progress. I note that beyond satisfying the UBC RPE requirement, I hope to obtain work that can be incorporated into a useful research contribution in the form of a research paper for submission. My initial target would be the EuroSys conference deadline in late October, though NSDI (late September) is another possibility. A paper submission is not a requirement for the RPE; the paper may include additional work beyond that done for the RPE.

This is a research proposal and as such is subject to revision during the lifetime of the research. It has been written based upon a broad survey of the literature — a literature base that is under considerable "churn" at the present time. For example, in earlier iterations of this work I was more interested in key-value stores. In the period October 2018 to June 2018 there are multiple substantial contributions in this area, including Anna (§**??**), KV-Direct [**?**], Faster [**?**], and Nibble [**?**].

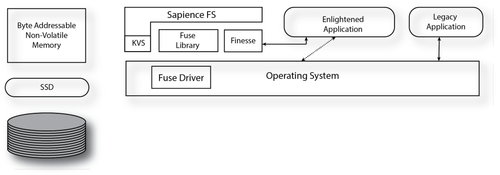
Similarly, the imminent availability of NVM hardware has created a bit of a frenzy of papers evaluating how it can be integrated into existing systems. This includes clfB-tree [**?**], Efficient Logging in NVM [**?**], UDORN [**?**], Continuous Checkpointing [**?**], NVHT [**?**], WHISPER [**?**], Persistent Lock-Free Queue for NVM [**?**], and Persistent Memory Transactions [**?**].

In doing this evaluation, I noted that much of the work assumes that cache line writes are **guaranteed** in the presence of a power failure. WHISPER, for example, references an article on `lwn.net` that purportedly guarantees that this is the case. When I followed up on that citation it referenced an Intel source that "recommended" platform vendors correct for this. A slide deck presented several years ago by an Intel Fellow in fact says that it is up to the platform vendor to provide this functionality. I did find a SuperMicro patent filing that includes additional **Memory Type Range Registers** that they proposed for this very purpose.

Even if they guarantee no cache-line tearing, there are no guarantees that any data structures more than a single cache line long would be properly preserved. The Oracle work [**?**] explicitly chooses to use checksums, while one of the other works [TBD] suggests using an alternating bit for detecting tears.

Thus, I would argue that the existing failure models are overly optimistic. A model in which such

Figure 1: Sapience File Systems Project



failures can be detected and handled would be useful. Analysis of that could form an element of a useful contribution in this area. Even if the hardware can handle this for a single cache line, the need to protect multiple cache line operations or dependencies seems prudent.

The existing mechanisms for consistency are simplistic and tend to utilize fencing fairly aggressively. The prior literature also seems to converge around *Epoch Consistency*, which if properly applied should require only a single store fence per epoch.

One aspect of analysis that seems to be missing from the prior work is an actual evaluation of the costs of various operations. I have had the "Jeff Dean" numbers in my head for quite some time, but decided that nine year old performance numbers were probably not entirely accurate. Tracking down this data confirmed this suspicion (and took some digging to find). Current high-speed DDR-4 memory has a read latency of approximately 15 nanoseconds, and 25 nanoseconds for write operations (source is a publication by Crucial discussing clock cycles versus actual latency. [**?**]).

More broadly, understanding the costs of contention, such as on a spin lock, between threads of the same core as well as cores of the same CPU or cores on separate CPUs may be helpful in reasoning about synchronization trade-offs.

This dovetails into this next point: another interesting line of inquiry is to evaluate the possibility of finding a *hybrid* model for data management. For example, prior work looks at delegation. [**?**, **?**] One objection to high performance shared nothing models is they do not perform well when the workload becomes mixed. Rather than abandon shared nothing, it might make more sense to have an adaptive model for switching from delegation (shared nothing) to synchronized (synchronization). Existing work takes an "either/or" position on these, when it seems clear they actually have strengths in different scenarios. Thus, a system in which we switch from one to the other might make better sense from a scaling perspective. This is a fairly open-ended research area and may be beyond the scope of a short research project area. I keep it in mind because it may provide useful performance insights.

4.1 Goals

The goals for this proposal are to evaluate the performance trade-offs between performance and persistence, as realized by augmenting existing KVS. To do this effectively, the project must:

- **Describe a Failure Model** — in order to reason about crash consistency, it is important to have a clearly defined failure model. In turn, this allows me to specify how crash consistency is guaranteed, relative to the failure model. At a minimum, evaluating a cache-tearing model, versus atomic cache write-back model, will be evaluated.

- **Evaluate durability** — there is an inherent trade-off between durability and performance. At a minimum, my work must consider write-durability. In addition, I propose to also investigate other models, including fast-write (no durability guarantee), fsync (explicit durability), and epoch durability. For write-durability, aggressive flush versus block until completion will be evaluated.

- **Performance Evaluation** — using at least two existing KVS, evaluate the impact of adding NVM durability to those systems. Evaluation shall be both in terms of software impact ("how extensive are the changes?") as well as benchmark impact.

The choice of KVS to use for this work will be part of the work itself. At the present time, I would suggest considering choosing from MassTree, Anna, Faster, and Nibble. It may make the most sense to pick at least two that have been previously evaluated using the same benchmarks. Choosing implementations that have used different benchmarks may make it more difficult to compare to the prior published work.

The specific failure model needs to identify the particular problems that need to be considered. This includes: using volatile addresses in non-volatile memory, validating that for each intermediate state there is a clear mechanism for restoring consistency across a crash, ensuring we can handle out-of-order write-back, and torn writes. For example, something in the CPU cache may be evicted in an order that differs from the order in which it was written into the cache, which can change the temporal ordering. A torn write occurs when only part of a cache line is written, such as in a power failure scenario. Given that much of the prior work assumes no cache line tearing, it would be useful to measure the costs associated with that (e.g., checksumming, which currently has a cost of just over one CPU clock cycle per byte if using the Intel optimized CRC-32 algorithm.)

Durability relates to the guarantees provided to the caller. In a model where I am modifying only existing KVS it makes sense to conform to the guarantees of the existing systems; if I can identify specific alternatives that are useful, they could be evaluated but likely not in the scope of this project.

The key to this work is performance. Thus, I need to be convincing in the micro-implementations that I make (ergo, demonstrating that they have excellent performance) as well as how they are implemented within the context of the existing KVS. While it would be ideal if my final results were a library, that may compromise performance, which suggests that I must determine if the approach needs to be optimized to the existing KVS.

The lack of availability to **NVM** is a potential challenge to this work. While it would be ideal if I can gain access to it, that is not guaranteed. Thus, part of this work will be to either use an existing simulation, or provide a simple simulation to mimic the behavior of NVM on an existing system. This is a significant open area that presents a challenge for this project.

I propose considering the following consistency mechanisms as part of this work:

- **Versioning** — in this mechanism multiple *versions* of a data structure are maintained. The benefit of this approach is that it is easy to reason about. The downside is that it amplifies storage utilization.

- **Logging** — in this mechanism we utilize a journaling technique (redo, undo, or both) to permit recoverable atomic updates to operations. The upside to this is that it is broadly general. The downside is that it is more complex and may increase the serialization (fencing) requirements.

- **Shadow Copying** — in this mechanism we use a read-copy-update mechanism for propagating changes through related data structures in such a way that the final update also makes the change visible.

- **HTM** — in this mechanism, we use hardware transactional memory for performing operations. The upside to this is that HTM exploits the existing hardware cache mechanisms, which happens to be the area we are concerned about. The downside to this is that HTM has substantial limitations to our ability to utilize it as it may not be present on many platforms and even when it is avaialble there are significant restrictions on its usefulness.

Realistically, a viable solution is likely to employ one or more of these techniques, as appropriate.

This work will explore some or all of the following:

- What to measure
- The tradeoffs between the consistency mechanisms
- Which consisency mechanisms are appropriate for specific situations
- Are there any prospective hybridizations of consistency mechanisms that exploit characteristics of memory?
- Impact of persistent memory versus dynamic memory
- Effect of read versus write asymmetry
- Strong versus weak consistency guarantees (focused on API models)
- Cost/benefit of various approaches applied to specific use cases
- Provide a description of actions for each consistency mechanism (”what happens when”)
- How to expose NVM to a container

Note that the initial analysis will be done with conventional dynamic memory and then consider how replacement with NVM changes the analysis. What functionality becomes redundant? What functionality is *missing* or needs to be added in the new model. How does it impact the tradeoffs and how well does it address the needs for our systems? How do we add any missing functionality?

My expectation is that to use NVM efficiently will involve investigating specific techniques that permit minimizing fence operations, such as *batching*. For example, if the interface guarantees that an update is persistent when the **put** operation is complete, I may be able to defer the put operation. Thus, for example, a rather straight-forward mechanism would be to fence if there is no more pending work to perform or if there is pending work to perform and some time (the *epoch* period) has not yet elapsed. While this will increase the latency for a single operation, it should yield a system that provides better overall throughput.

Another approach that may make sense (and could ultimately lead in a direction of buiding a new optimized KVS) is to provide a richer interface: one in which the operations may be asynchronous. This permits higher levels to issue a series of calls and then either force a fence or ask for a return once a specific epoch has been achieved.

4.2 Deliverables

The deliverables for this project are:

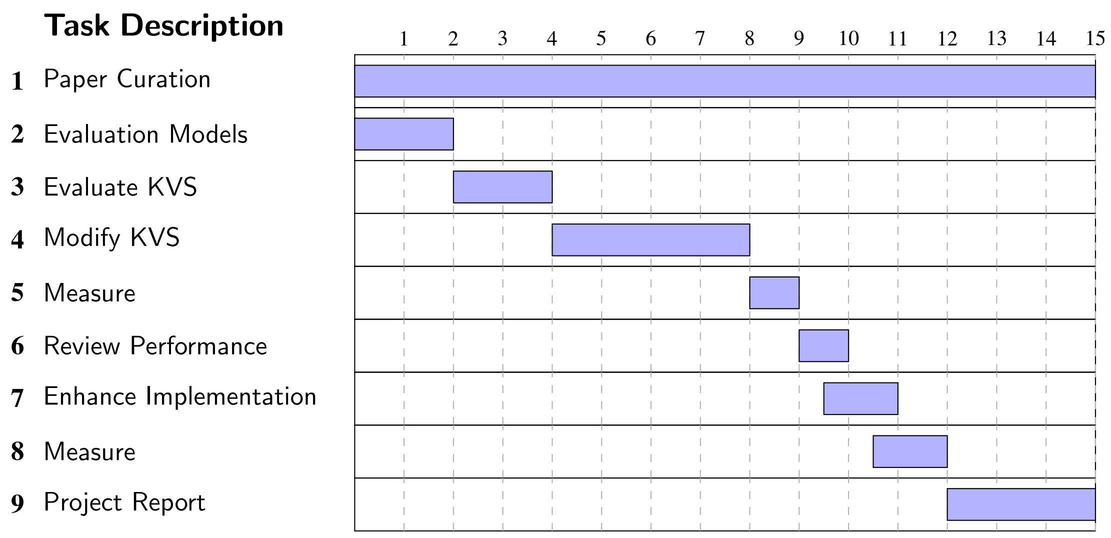
- **Reference Curations** — thus far I have done short (approximately 1000 works) curations for a number of the background papers in this work. I intend on continuing to do this work. I do not expect to curate all those papers, but there are a number of additional papers that are relevant and useful in understanding this work. Thus, I will continue to do those write-ups as part of this project work.
- **Project Paper** — the final outcome of this project should be a paper that summarizes the work that I did for this project, my findings, and my conclusions. It should be of sufficient detail that a reader can understand what I did and reproduce my results.
- **Presentation** — a presentation of my work and findings, suitable for presentation to a general computer science audience to explain the project, my findings, and my conclusions.
- **Project Repository** — a complete repository that includes any code I used for the project, data that I collected and produced during the project, the paper, and the presentation for the project.

The expectation is that I will officially present this in Fall 2018 at a data and time to be determined.

4.3 Schedule

The schedule for this project is shown in Figure 2

Figure 2: Project Schedule by Week



In addition, the presentation of this work will be scheduled (to be determined) for Fall 2018.

5 Supplement

The supplemental material that was originally included with this document has been moved to a separate document entitled *PERC Supplementation Material*.

References

[1] Calciu, I., Dice, D., Harris, T., Herlihy, M., Kogan, A., Marathe, V., and Moir, M. Message passing or shared memory: Evaluating the delegation abstraction for multicores. In *International Conference on Principles of Distributed Systems* (2013), Springer, pp. 83–97.

[2] Chandramouli, B., Prasad, G., Kossmann, D., Levandoski, J., Hunter, J., and Barnett, M. Faster: A concurrent key-value store with in-place updates. In *Proceedings of the 2018 ACM International Conference on Management of Data* (2018), ACM.

[3] Chen, X., Sha, E. H.-M., Abdullah, A., Zhuge, Q., Wu, L., Yang, C., and Jiang, W. Udorn: A design framework of persistent in-memory key-value database for nvm. In *Non-Volatile Memory Systems and Applications Symposium (NVMSA), 2017 IEEE 6th* (2017), IEEE, pp. 1–6.

[4] Cohen, N., Friedman, M., and Larus, J. R. Efficient logging in non-volatile memory by exploiting coherency protocols. *Proc. ACM Program. Lang. 1*, OOPSLA (Oct. 2017), 67:1–67:24.

[5] Crucial. Speed vs latency: Why cas latency isn't an accurate measure of memory performance, 2017. http://www.crucial.com/usa/en/memory-performance-speed-latency (accessed April 16, 2018).

[6] Friedman, M., Herlihy, M., Marathe, V., and Petrank, E. A persistent lock-free queue for non-volatile memory. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, NY, USA, 2018), PPoPP '18, ACM, pp. 28–40.

[7] GILES, E., DOSHI, K., AND VARMAN, P. Continuous checkpointing of htm transactions in nvm. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on Memory Management* (2017), ACM, pp. 70–81.

[8] KIM, W.-H., SEO, J., KIM, J., AND NAM, B. clfb-tree: Cacheline friendly persistent b-tree for nvram. *ACM Transactions on Storage (TOS) 14*, 1 (2018), 5.

[9] LI, B., RUAN, Z., XIAO, W., LU, Y., XIONG, Y., PUTNAM, A., CHEN, E., AND ZHANG, L. Kv-direct: High-performance in-memory key-value store with programmable nic. In *Proceedings of the 26th Symposium on Operating Systems Principles* (New York, NY, USA, 2017), SOSP '17, ACM, pp. 137–152.

[10] MARATHE, V., MISHRA, A., TRIVEDI, A., HUANG, Y., ZAGHLOUL, F., KASHYAP, S., SELTZER, M., HARRIS, T., BYAN, S., BRIDGE, B., ET AL. Persistent memory transactions. *arXiv preprint arXiv:1804.00701* (2018).

[11] MERRITT, A., GAVRILOVSKA, A., CHEN, Y., AND MILOJICIC, D. Concurrent log-structured memory for many-core key-value stores. *Proceedings of the VLDB Endowment 11*, 4 (2017), 458–471.

[12] NALLI, S., HARIA, S., HILL, M. D., SWIFT, M. M., VOLOS, H., AND KEETON, K. An analysis of persistent memory use with whisper. *SIGARCH Comput. Archit. News 45*, 1 (Apr. 2017), 135–148.

[13] ROGHANCHI, S., ERIKSSON, J., AND BASU, N. ffwd: delegation is (much) faster than you think. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), ACM, pp. 342–358.

[14] ZHOU, J., SHEN, Y., LI, S., AND HUANG, L. Nvht: an efficient key-value storage library for non-volatile memory. In *Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies* (2016), ACM, pp. 227–236.