Projects Since HPTS 2015

Charlie Johnson
(and interns and others)

HPTS ‘15

I’m going down!

(How to recover from failure in milliseconds)

Charlie Johnson

Availability depends upon Time to Recover

- Mean Time Between Failures (MTBF) is important.
- Mean Time to Recover (MTR) is much more important.
- Availability = A = MTBF / (MTBF + MTR)
- lim(Availability) ≈ 1
  MTR -> 0
- As the Mean Time to Recover decreases, Availability approaches 100%
- ∴ Probability of Failure = F = 1-A ≈ 0
- ∴ Reliability = 1/F = 1/(1-A) ≈ inf
- HPE Nonstop has implemented this and now does node takeovers in milliseconds: called “CPU Broadcast”: by hacking the NMI driver to send out a death multicast message. They now have the fastest takeover in the world.

TxHPC at NVMW 2017

PERSISTENT REGIONS THAT SURVIVE NVM MEDIA FAILURES

Onkar Patil, Mesut Kuscu, Tuan Tran, Charles Johnson, Joseph Tucek, Harumi Kuno

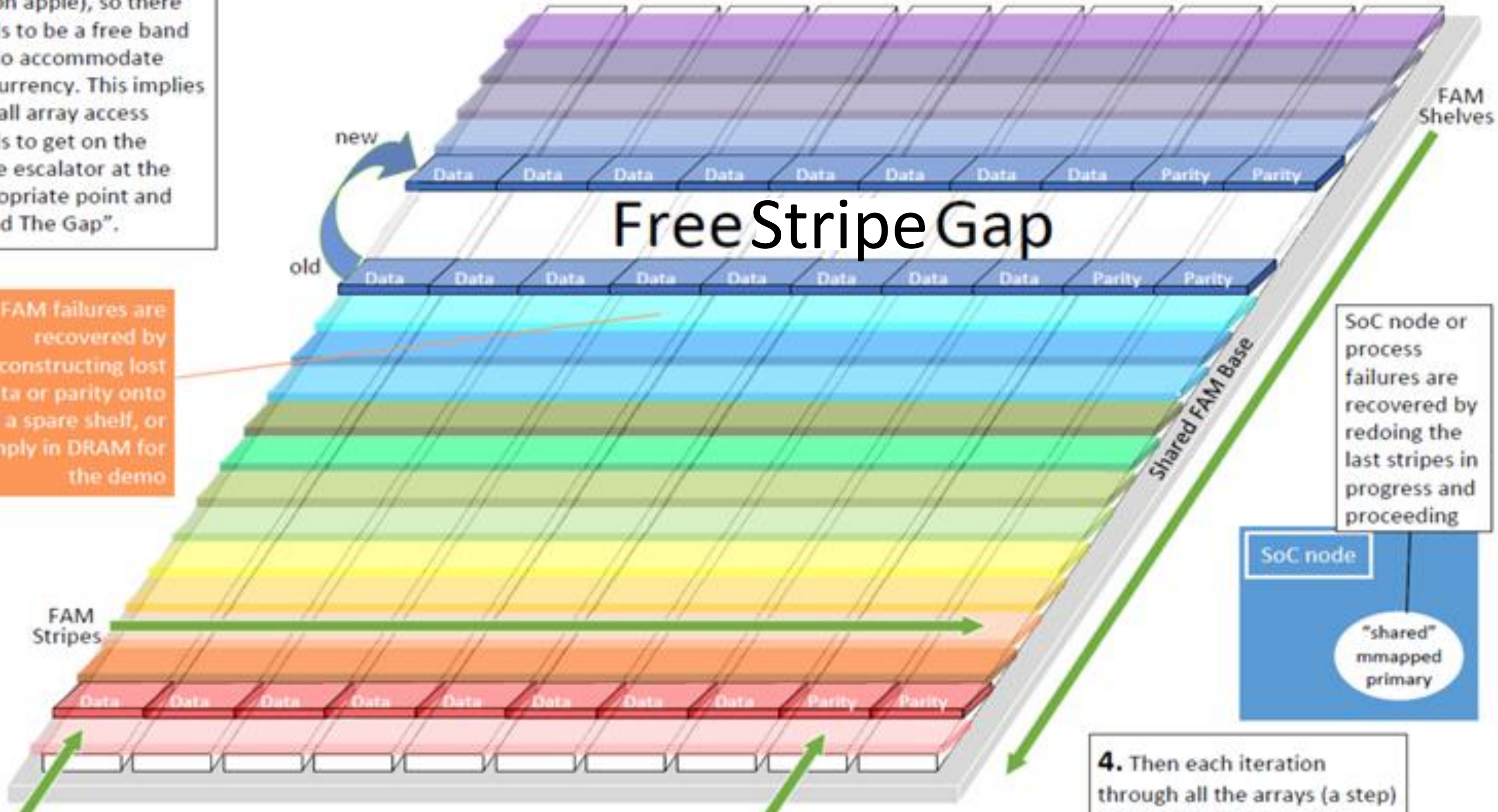
Hewlett Packard Labs, Palo Alto, CA

NVMW 2017

What are the details?

Take Six: Software RAID 6 [10 shelves on separate Zbridges on one ZSwitch: 8 + 2 parity] at ~20% cost, using a "free band gap" (concurrency sized) and stripe iteration through the arrays, as if the data were moving up an escalator through the FAM (with much less overhead and work than Take Five.)

1. Stripes must be atomically written, but NOT update-in-place (the poison apple), so there needs to be a free band gap to accommodate concurrency. This implies that all array access needs to get on the stripe escalator at the appropriate point and "Mind The Gap".



FAM failures are recovered by reconstructing lost data or parity onto a spare shelf, or simply in DRAM for the demo

SoC node or process failures are recovered by redoing the last stripes in progress and proceeding

2. Using the page size of 4096 for the FAM Data Block, you get 512 64-bit floating or integer array elements, and then 4096 of said array elements per stripe for thread parallel data and parity computation in cores (one FPU and one Integer Unit per core)

3. That leaves two to three hyperthreads (out of 4) per core for executing non-temporal (hopefully) load and store instructions

4. Then each iteration through all the arrays (a step) is completed by iterating through the stripes.

Diagram by Charles Johnson

TxHPC presentation and code

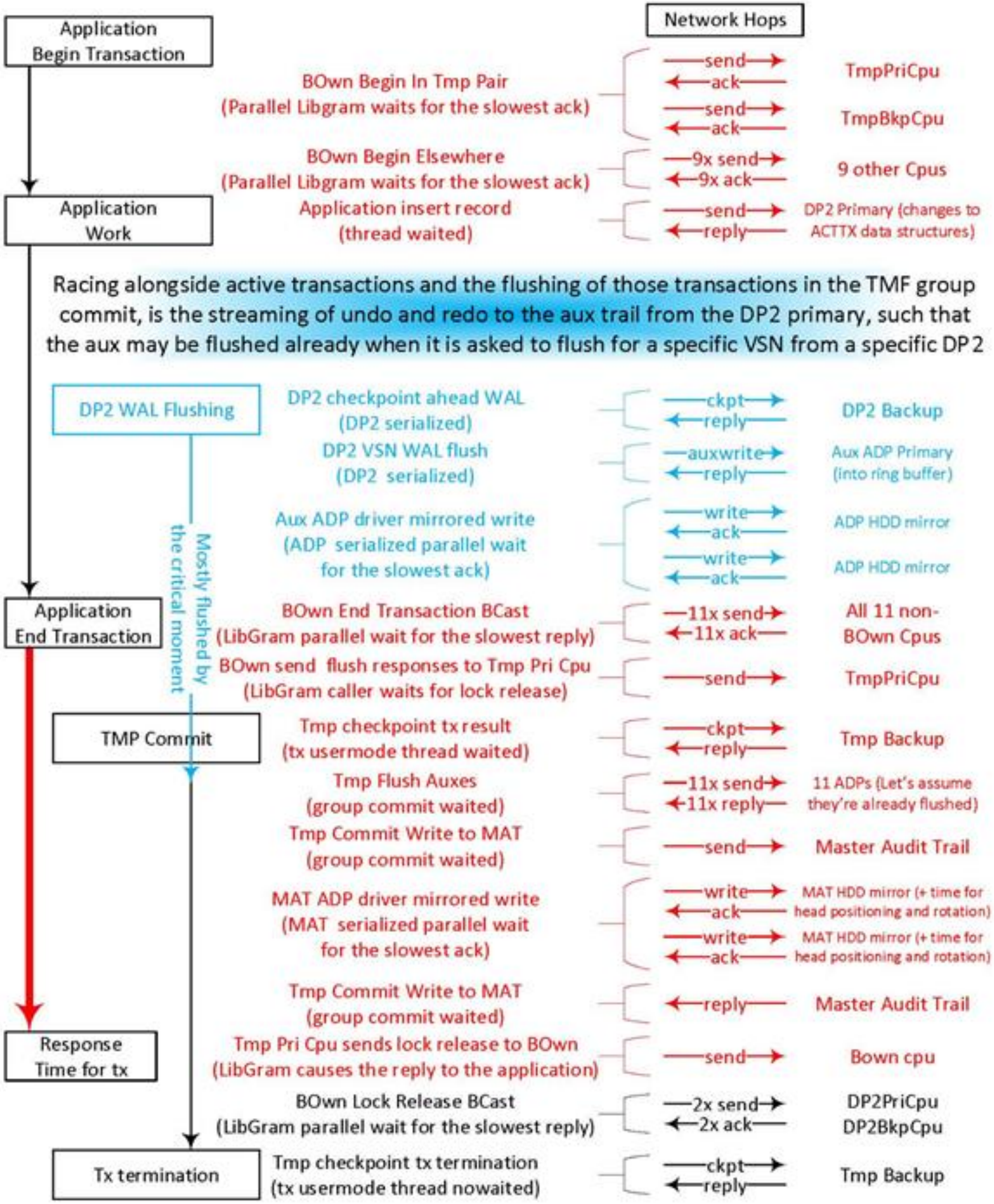
- Two-page abstract: http://nvmw.ucsd.edu/2017/assets/abstracts/20
- Slides: http://nvmw.ucsd.edu/2017/assets/slides/20
- Open source github: https://github.com/HewlettPackard/Redhead
- TxHPC source (uses Jerasure 2.0 + GKComplete): https://github.com/HewlettPackard/Redhead/tree/master/include/StencilForTxHPC/TxHPC4TM

Nonstop SQL Subtransactions

- Nonstop clustered group 3-phase commit is the slowest in the business, response time in 10s of ms. at best for standard configurations.
- They used to have 90% of the trading business, only a couple of exchanges left: this is because all flash trading completes for the front-running and arbitrage of a single trade in 15 µs.
- 1n 1999 came up with a solution, presented to HPTS, but the problem wasn't pressing, then.
- Now it's an issue, so I was called in to fix the Nonstop commit code that I designed and wrote (with much help from Pat, Shel, Jimbo, Matt M., J. Carley, J. Klein, etc.)
- With SQL Subtransactions, we could get 4 orders of magnititude, with H/W work maybe another 2-3 orders in both throughput and response time (Big Ω.)

Single Record Insert on a 12 cpu Nonstop with 11 ADPs using TMF transactions

- It's completely scaled out and bullet proof.
- That translates to slow.
- There are lot of waited steps in RED.
- The part in BLUE could be very fast if we could just execute that part.
- We need a new transaction type that fits into the old transaction recovery system: SQL subtransactions.
- They need a new delivery system: a special message as a top-level transaction.
- They need to execute completely within a single disk process instance, call it a DPX.
- They need collocation to a single processor cache hierarchy to reduce response time.
- They need all resources to be confined to a single disk process.
- They need buffering/multiplexing to increase efficiency and throughput.
- They need to be ACID and the same level of high availability as TMF transactions.
- They need to be as programmable as TMF transactions, modulo the issues of closure on collocated resources.



Application Begin Transaction

Network Hops

BOwn Begin In Tmp Pair (Parallel Libgram waits for the slowest ack) — send→ ←ack TmpPriCpu; send→ ←ack TmpBkpCpu

BOwn Begin Elsewhere (Parallel Libgram waits for the slowest ack) — 9x send→ ←9x ack 9 other Cpus

Application Work

Application insert record (thread waited) — send→ ←reply DP2 Primary (changes to ACTTX data structures)

Racing alongside active transactions and the flushing of those transactions in the TMF group commit, is the streaming of undo and redo to the aux trail from the DP2 primary, such that the aux may be flushed already when it is asked to flush for a specific VSN from a specific DP2

DP2 WAL Flushing

Mostly flushed by the critical moment

DP2 checkpoint ahead WAL (DP2 serialized) — ckpt→ ←reply DP2 Backup

DP2 VSN WAL flush (DP2 serialized) — auxwrite→ ←reply Aux ADP Primary (into ring buffer)

Aux ADP driver mirrored write (ADP serialized parallel wait for the slowest ack) — write→ ←ack ADP HDD mirror; write→ ←ack ADP HDD mirror

Application End Transaction

BOwn End Transaction BCast (LibGram parallel wait for the slowest reply) — 11x send→ ←11x ack All 11 non-BOwn Cpus

BOwn send flush responses to Tmp Pri Cpu (LibGram caller waits for lock release) — send→ TmpPriCpu

TMP Commit

Tmp checkpoint tx result (tx usermode thread waited) — ckpt→ ←reply Tmp Backup

Tmp Flush Auxes (group commit waited) — 11x send→ ←11x reply 11 ADPs (Let's assume they're already flushed)

Tmp Commit Write to MAT (group commit waited) — send→ Master Audit Trail

MAT ADP driver mirrored write (MAT serialized parallel wait for the slowest ack) — write→ ←ack MAT HDD mirror (+ time for head positioning and rotation); write→ ←ack MAT HDD mirror (+ time for head positioning and rotation)

Response Time for tx

Tmp Commit Write to MAT (group commit waited) — ←reply Master Audit Trail

Tmp Pri Cpu sends lock release to BOwn (LibGram causes the reply to the application) — send→ Bown cpu

BOwn Lock Release BCast (LibGram parallel wait for the slowest reply) — 2x send→ ←2x ack DP2PriCpu DP2BkpCpu

Tx termination

Tmp checkpoint tx termination (tx usermode thread nowaited) — ckpt→ ←reply Tmp Backup

- SQL Subtransactions has reached detailed design, 3rd revision of the spec awaiting a spot in the very busy Nonstop software development schedule (currently supporting the new Virtualized Nonstop VM on x86_64 for Gen9 hardware.)
- On to the next project that advances the state of the art of resilience.