ReFS v2

Cloning, projecting, and moving data

J.R. Tipton
jrtipton@microsoft.com

What are we talking about?

- Two technical things we should talk about
  - Block cloning in ReFS
  - ReFS data movement & transformation
- What I would love to talk about
  - Super fast storage (non-volatile memory) & file systems
  - What is hard about adding value in the file system
    - Technically
    - Socially/organizationally
- Things we actually have to talk about
  - Context

Agenda

- ReFS v1 primer
- ReFS v2 at a glance
- Motivations for v2
- Cloning
- Translation
- Transformation

ReFS v1 primer

- Windows allocate-on-write file system
  - A *lot* of Windows compatibility
- Merkel trees verify metadata integrity
  - Data integrity verification optional
- Online data correction from alternate copies
- Online chkdsk (AKA salvage AKA fsck)
  - Gets corruptions out of the namespace quickly

ReFS v2 intro

- Available in Windows Server Technical Preview 4
- Efficient, reliable storage for VMs: fast provisioning, fast diff merging, & tiering
- Efficient erasure encoding / parity in mainline storage
- Write tiering in the data path
  - Automatically redirect data to fastest tier
  - Data spills efficiently to slower tiers
- Read caching
- Block cloning
  - End-to-end optimizations for virtualization & more
- File system-y optimizations
  - Redo log (for durable AKA O_SYNC/O_DSYNC/FUA/write-through)
  - B+ tree layout optimizations
  - Substantially more parallel
  - “Sparse VDL” – efficient uninitialized data tracking
  - Efficient handling of 4KB IO

Why v2: motivations

- Cheaper storage, but not slow storage
- VM density
  - Incentive to share precious resources
    - Don’t do more IO than necessary
    - Don’t take more capacity than necessary
- VM provisioning
- More hardware flavors; less homogeneity in a commodity box
  - SLC, MLC, TLC flash
  - Shingled magnetic recording (SMR disks)
  - Scary fast storage

Why v2: random write performance

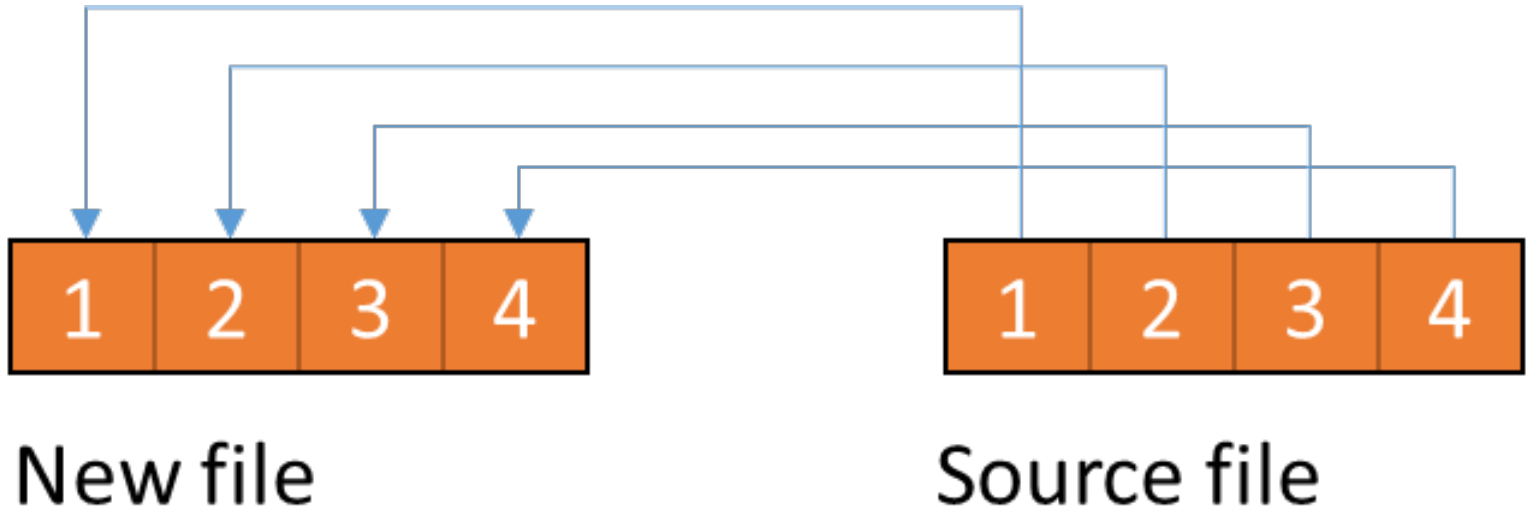
- Lots of magic in marketplace
- Magic not holding up well in harsh environments
  - Exceptionally fast hardware
  - Small random writes
  - Writes that must be durable on ack (FUA AKA sync AKA write-through)

Block cloning in ReFS

- Clone any block of one file into any other block of another file

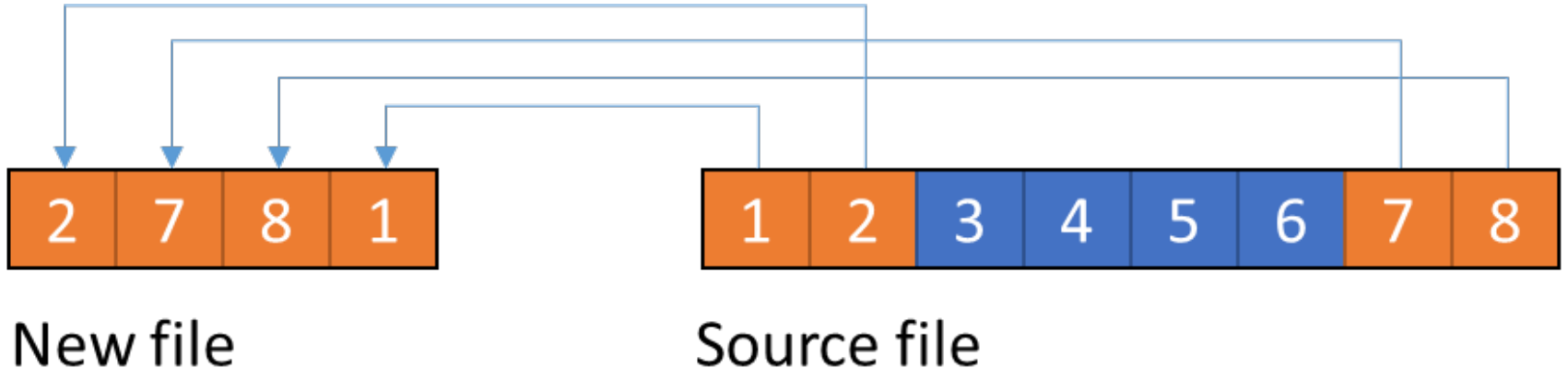
Block cloning in ReFS

- Clone any block of one file into any other block of another file
  - Full file clone



Block cloning in ReFS

- Clone any block of one file into any other block of another file
  - Reordering some or all data within a file



Block cloning in ReFS

- Clone any block of one file into any other block of another file
  - Can be used to ‘project’ data from one area into another w/o copy

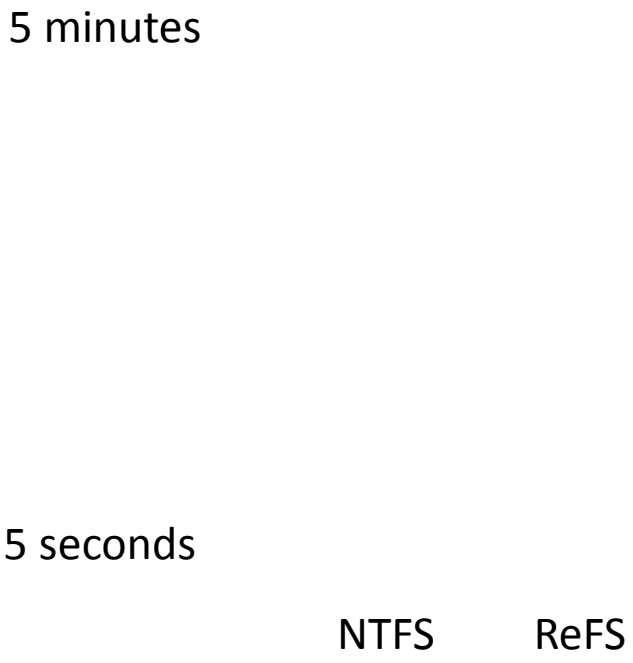
Block cloning in ReFS, cont.

- Copy-on-write only used when needed
  - ReFS knows when only one copy remaining
  - ReFS knows if it is okay to write in place
- Cloning/projecting is metadata-only operation
  - Fine-grained synchronization
  - Time is linear with number of extents involved

Block cloning in practice

- Deleting a Hyper-V VM checkpoint
  - Differencing disks track differences from parent
  - At some point, parent data will be discarded
    - Easiest to write old & new data to new VHD
    - Block cloning “moves” the data quickly
    - Cleans up VHD metadata (data logically appears in preferred order)
  - Extremely fast on ReFS v2

Block cloning in practice: delete checkpoint



- Deleting VM checkpoint requires reordering of data into different VHDs
- On traditional file systems this means moving a lot of blocks
- ReFS can make this trivially fast

Block cloning in practice: VM provisioning

1 minute

100 VHDs

< 1 sec

1 VHD

NTFS ReFS

NTFS ReFS

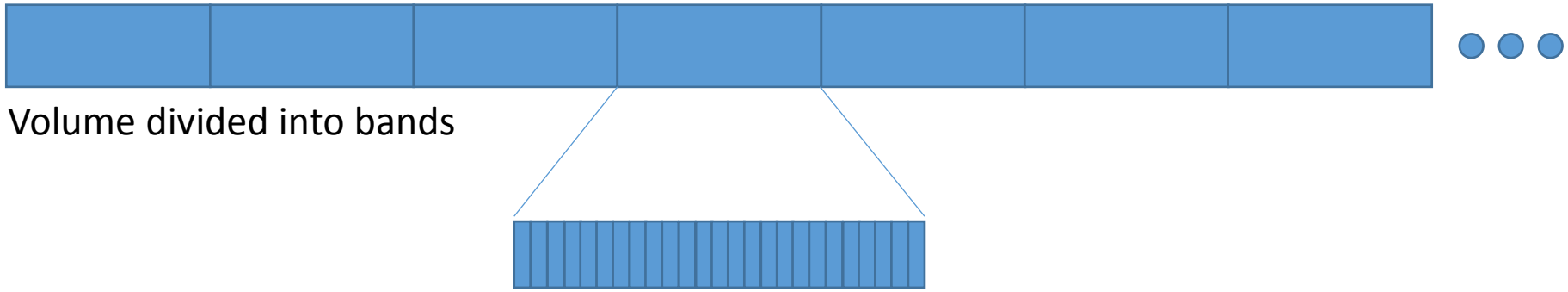
Block cloning observations

- App-directed
- Avoids data copies
- Meta-point: not just data copying
  - Some subsystem metadata challenges can be offloaded
  - Hyper-V not only system utilizing this; avoiding data copies not only motivation
- No free lunch
  - Multiple valid copies requires copy-on-write for modification
  - Some metadata overhead to track state
  - Slam dunk in most cases, but not all
  - Spreading cost around

ReFS cluster bands

- Volume internally divvied up into ‘bands’
  - Bands are large relative to IO (e.g. 64MB, 256MB)
- Bands contain regular FS clusters (4KB, 64KB)
- ReFS tracks bands & clusters independently
  - Per-band metadata
- Bands can come and go
- Mostly invisible outside of file system
  - Aside from performance counters, etc.

ReFS cluster bands illustrated (poorly)



key	*value*
File offset	• Band ID • Offset within band • Checksum

File extent tables

ReFS cluster bands: translation

- Less fancy explanation: “ReFS can move bands around”
- Moving = reading, writing, updating band pointer
- Efficient write caching & parity/erasure encoding
  - Redirect writes to bands in fast tier
  - Per-band metadata tracks heat, etc.
  - ReFS moves bands between storage tiers
    - Movement is sequential
  - Made more efficient by just-in-time allocation

ReFS cluster bands: translation, cont.

- Multi-tiered environments is focus in this release
  - Log structured?
    - Logs generally don’t write in-place, incurring GC all the time
  - Copy-on-write?
    - Only when necessary
- From 100% 3x mirroring to 95% parity
  - Last 5% can be had, too
- Where possible, act like a ‘dumb’ file system for efficiency
  - Only get smart when necessary
  - Being smart can be painful (at least that’s what smart people tell me)

ReFS cluster bands: translation, cont.

- Small writes accumulate in storage tier where writes are cheap
  - For example
    - Mirror
    - Flash
    - Log-structured arena
- Bands are shuffled to storage tier where random writes are expensive
  - Most efficient is always to write sequentially
  - Band transfers are fully sequential

ReFS cluster bands: transformation

- Less fancy explanation: “ReFS can do stuff to the data in a band”
- Can happen in background
  - Data path can just allocate-on-write & is otherwise unaffected
- Examples
  - Band compaction
    - Put cold bands together, squeeze out free space
    - Does not affect read data path
  - Compression
    - Good candidate for background activity
    - Affects read data path (decompress)

ReFS v2: miscellany

- These new systems are not easily compared to previous systems
- Can be challenging for software organizations to evaluate
  - Why is ZFS so much slower than UFS in my favorite scenario?
  - Why is ReFS not doing what NTFS does in my other favorite scenario?
  - Answers not obvious in vacuum – it's about the real world situation
- File systems taking on more responsibility than before
  - Automatically imbue many stacks with new traits
  - FS has a lot of "knowledge"
  - Trendy
  - ..but also complex, forced to make tradeoffs in general systems, and trendy.

ReFS v2

- Data cloning
- Data movement
- Data transformation
- Smart when smart makes sense
  - Switches to ‘dumb’ when dumb is better
- Takes advantage of hardware combinations
- And lots of other stuff, too