Map File System Offsets to Volume LBA's

1. Overview / Purpose

This task will involve implementing logic in dt to map file system offsets to volume (LUN) Logical Block Address (LBA). This is necessary to reduce the amount of time required for product engineers when debugging data corruption. dt reports file relative offsets and LBAs, but *not* the **real** LBA desired by developers, thus this enhancement. 😊

Note: For those new to storage, please know LBA is the Logical Block Address. This LBA is normally 512 bytes but can be larger values such as 2k or 4k, etc.

Remember: dt supports a **dsize=value** option, so when set to 4k, the LBA's reported below will be 4k, not 512 bytes.

1.2. I/O Tracing

There are several ways to acquire LBA's accessed, but many require setup to enable tracing, and this tracing can impact performance and the ability to reproduce to original corruption.

Today, dt emits a re-read command line that can be used to access the file blocks **after** tracing is enabled. But that said, many don't know this and it's an extra step to locate volume LBA's.

While switching to direct disk testing resolves this issue, many folks test through a file system, since this mandate has been established in the past. Future automation changes will allow us to do concurrent file system and direct disk I/O.

2. File Systems Support

Each host file system type generally has different metadata, so converting file offsets to LBAs often requires intimidate knowledge of the file system involved.

In the case of Windows NTFS, someone tracked down the source code to map NTFS file offsets to volume LBA. This code was integrated into dt.

The Windows device control is IOCTL_VOLUME_LOGICAL_TO_PHYSICAL

For Linux, the this mapping will be done by using the File Extent Map (FIEMAP) file control to find the mapping of file offset to volume LBA. The *hdparm* tool supports this today.

For other OS's and/or other file systems, alternative methods will need to be implemented. But today, most testing is with Linux ext4 and Windows NTFS, so these were prioritized first.

3. What's Supported?

For Linux, the file offset to physical LBA support has been tested with ext4 and XFS to date.

For Windows, support has been added for NTFS file systems with drive letters (but *not* mount points).

4. Implementation

When a data corruption occurs, the file offset to LBA mapping occurs automatically. When the initial error is detected, the file extent map for the error record is displayed. Areas that previously reported file relative LBA's have also been updated to report physical LBA's.

In addition, two new dt commands have been added to report the file map information:

1. **showfslba** - Show file system offset to physical LBA.
2. **showfsmap** - Show file system map extent information.

Command Formats:

File System Map Command Format:
showfslba [bs=value] [count=value] [limit=value] [offset=value]

Show FS offset(s) mapped to physical LBA(s)
The default is to show LBA for specified offset.

showfsmap [bs=value] [count=value] [limit=value] [offset=value]

Show the file system map extent information.
The default is to show the full extent map.

With these two dt commands, there's a consistent interface for both operating systems!

Please know that these commands can be used on any data file, not just those created by dt. 😊

Previously, folks on Linux used *hdparm* and on Windows the *file2lba.exe* tool.

4.1. Windows Example

This is what you'll see on a Windows system **after** a data corruption:

```
dt.exe (j:1 t:1):          Starting Physical LBA: 12846944 (0xc40760)
dt.exe (j:1 t:1):            Ending File Offset: 3145728 (0x300000)
dt.exe (j:1 t:1):           Ending Physical LBA: 277576287 (0x108b7a5f)
dt.exe (j:1 t:1):             Error File Offset: 0 (0x0)
dt.exe (j:1 t:1):           Error Offset Modulos: %8 = 0, %512 = 0, %4096 = 0
dt.exe (j:1 t:1):    Starting Physical Error LBA: 12846944 (0xc40760)
dt.exe (j:1 t:1):            4096 byte Error LBA: 1605868 (0x1880ec)
dt.exe (j:1 t:1):        Corruption Buffer Index: 0 (byte index into read buffer)
dt.exe (j:1 t:1):         Corruption Block Index: 0 (byte index in miscompare block)
dt.exe (j:1 t:1):
dt.exe (j:1 t:1): File: iot.data, LBA Size: 512 bytes
dt.exe (j:1 t:1): Physical Disk: \\.\PhysicalDrive0, Cluster Size: 4096 on \\.\C: [NTFS]
dt.exe (j:1 t:1):
dt.exe (j:1 t:1):    File Offset    Start LBA      End LBA     Blocks      VCN        LCN
dt.exe (j:1 t:1):              0     12846944     12847072        128        0    1317100
dt.exe (j:1 t:1):          65536    573043488    573045472       1984       16   71341668
dt.exe (j:1 t:1):        1081344    364490568    364490640         72      264   45272553
dt.exe (j:1 t:1):        1118208    446773544    446775520       1976      273   55557925
dt.exe (j:1 t:1):        2129920    446770464    446770528         64      520   55557540
dt.exe (j:1 t:1):        2162688    277574368    277576288       1920      528   34408028
```

Note: The Linux display is the same, except for the Virtual Cluster Number (VCN) and Logical Cluster Number (LCN).

Wherever physical LBA's are reflected, you'll see the word "Physical" added. If the file offset cannot be mapped, this is reflected by "**<not mapped>**", which indicates the LBA was **not** written, thus a sparse file.

4.2. Linux IOT Example

```
dt (j:1 t:1):                       Record #: 1
dt (j:1 t:1):         Starting Record Offset: 1048576
dt (j:1 t:1):                Transfer Count: 1048576 (0x100000)
dt (j:1 t:1):           Ending Record Offset: 2097152
dt (j:1 t:1):          Starting Physical LBA: 6387784 (0x617848)
dt (j:1 t:1):            Ending Physical LBA: 6389831 (0x618047)
dt (j:1 t:1):            Read Buffer Address: 0x7f5c4462f000
dt (j:1 t:1):           Pattern Base Address: 0x7f5c44731000
dt (j:1 t:1):                           Note: Incorrect data is marked with asterisk '*'
dt (j:1 t:1):
dt (j:1 t:1):                   Record Block: 0 (bad data)
dt (j:1 t:1):            Record Block Offset: 1048576 (LBA 6387784)
dt (j:1 t:1):          Expected Block Number: 2048 (0x00000800)
dt (j:1 t:1):          Received Block Number: 969120825 (0x39c39c39)
dt (j:1 t:1):          Received Block Offset: 496189862400 (Data Range: 1048576 - 2097152)
dt (j:1 t:1):        Seed Detected at Offset: 4 (0x4) (word 1, zero based)
dt (j:1 t:1):       Data Written During Pass: 0
dt (j:1 t:1):         Calculated Block Number: 969120825 (0x39c39c39)
dt (j:1 t:1):      Expected Data is for Seed: 0x01010101 (pass 1)
dt (j:1 t:1):     Received Data is from Seed: 0x00000000 (wrong data)
```

Wherever dt previously reported the range of blocks, as above, it now reports the starting and ending Physical LBA.

4.3. File System Map Example

```
PS C:\cygwin64\home\millerob\dt.v22\windows> x64/Release/dt.exe if=iot.data showfsmap
dt.exe (j:0 t:0): File: iot.data, LBA Size: 512 bytes
dt.exe (j:0 t:0): Physical Disk: \\.\PhysicalDrive0, Cluster Size: 4096 on \\.\C: [NTFS]
dt.exe (j:0 t:0):
dt.exe (j:0 t:0):    File Offset    Start LBA      End LBA     Blocks      VCN        LCN
dt.exe (j:0 t:0):              0     12846944     12847072        128        0    1317100
dt.exe (j:0 t:0):          65536    573043488    573045472       1984       16   71341668
dt.exe (j:0 t:0):        1081344    364490568    364490640         72      264   45272553
dt.exe (j:0 t:0):        1118208    446773544    446775520       1976      273   55557925
dt.exe (j:0 t:0):        2129920    446770464    446770528         64      520   55557540
dt.exe (j:0 t:0):        2162688    277574368    277576384       2016      528   34408028
dt.exe (j:0 t:0):        3194880    316560544    316560632         88      780   39281300
dt.exe (j:0 t:0):        3239936    606635616    606637632       2016      791   75540684
dt.exe (j:0 t:0):        4272128    534732248    534734272       2024     1043   66552763
dt.exe (j:0 t:0):        5308416     19829936     19831856       1920     1296    2189974
PS C:\cygwin64\home\millerob\dt.v22\windows>
```

4.4. File System LBA Example

```
PS C:\cygwin64\home\millerob\dt.v22\windows> x64/Release/dt.exe if=iot.data showfslba offset=1m count=10
dt.exe (j:0 t:0):    File Offset   Physical LBA
dt.exe (j:0 t:0):        1048576      573045408
dt.exe (j:0 t:0):        1049088      573045409
dt.exe (j:0 t:0):        1049600      573045410
dt.exe (j:0 t:0):        1050112      573045411
dt.exe (j:0 t:0):        1050624      573045412
dt.exe (j:0 t:0):        1051136      573045413
dt.exe (j:0 t:0):        1051648      573045414
dt.exe (j:0 t:0):        1052160      573045415
dt.exe (j:0 t:0):        1052672      573045416
dt.exe (j:0 t:0):        1053184      573045417
PS C:\cygwin64\home\millerob\dt.v22\windows>
```

When file offsets reside in the same extent or cluster, you will see ascending LBA numbers. The NTFS cluster size is configurable and as is the Linux file system block size.

5. Reference Trace

Today, dt provides file system offset mapping to physical LBA's for Linux and Windows. For Linux, this mapping has been verified for ext4 and XFS file systems and is accomplished via a file system IOCTL someone thankfully added on Linux. For other OS's and other file systems, dt does **not** provide file system offset to physical LBA mapping, but since most folk's test with Linux and Windows, no effort has been made to add *metadata prowlers* for other OS's. Also know, this mapping does *not* work with logical volumes (LVM's). For LVM's there may be system utilities to provide this mapping, but reference traces can be used instead.

I've been asked several times, "*How do I acquire this physical LBA mapping on other OS's?*" For example, on Solaris, where I may be using UFS or ZFS file systems, how can I find the physical LBA's of the corrupted record dt reported?

This is the reason why dt reports the re-read command line, as shown below, and is what I'm referring to as a *reference trace!* By enabling host tracing (*tcpdump* or *Wireshark*), or analyzer trace, or array tracing, all you need to do is execute the re-read command line and observe the physical LBA's in the trace.

Also know, while the re-read command line is for the entire record, you can also modify this to re-read just the corrupted blocks or individual blocks, using previous reported corruption information. And in case it's not obvious, please know this *must* be done with the original file system, **not** after copying the file elsewhere! 🙂

Example dt re-read command line:

dt if=/<file path>/<file name>.data-j1t4 bs=810496 count=1 offset=22848000 prefix="<file path>@<host name>" pattern=iot iotseed=0xb5b5b5aa flags=direct enable=btags disable=retryDC,savecorrupted,trigdefaults

This is reported after performing re-reads after a data corruption.