

HPSS RAIT Architecture

07/01/2009

Contributors: James Hughes, Dave Fisher, Kent Dehart, Benny Wilbanks and Jason Alt

Overview.....	2
Motivation.....	2
Reliability of RAIT.....	3
Existing HPSS Features Applicable to RAIT.....	4
HPSS RAIT Architecture.....	5
Failure Recovery Scenarios	7
Writing RAIT Volumes	7
Reading RAIT Volumes	7
Normal	8
Recovery	8
Offline Repack and Data Recovery	9
Proof of Concept.....	9

Overview

Tape is still relevant to supercomputer data centers because the ratio of disk to RAM is approaching on the order of 25 to 1^[1]. This means 25 files the size of memory and the computer is no longer useful for programs that store a lot of information.

Tape is a natural because it uses no energy when in long term storage, it is more robust to move than disk and storage can have wider environmental tolerances.

Tape devices are getting fast for sequential operations. LTO6 will operate at 324 MB/s assuming a 1.2:1 compression ratio^[2].

Tape has a significant drawback in that tape capacity is not growing as fast as the RAM size of the future supercomputers. This fact leads to Striping.

Motivation

The motivation is that history has shown that there are many potential interruptions to jobs that are caused by tape handling and reading. These errors are not supposed to happen but they do. These include library errors like the arm failing with a tape in the gripper, the tape falling out of the gripper, incorrectly inserting the tape into the throat of

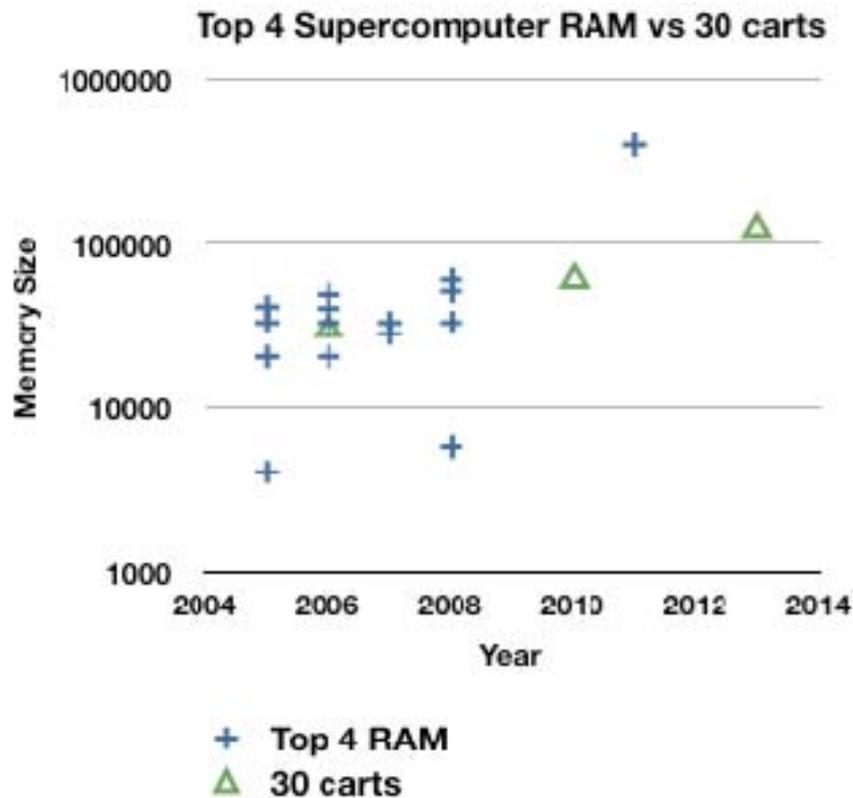


Figure 1

the drive incorrectly. Failures of the drive can be the inability to load the tape, metadata errors on the tape, tensioning errors, breaking tape, running off the end of the reel, microcode errors, etc. The media can have unrecoverable data errors on it caused by many things including tape damage.

While each vender says that these items are supposed not to happen, Murphy Law prevails.

One large data center with most, if not all, off the tape vendors states that they have experience that 1 out of every 100 tape handling events end up having an issue that interrupts the job execution. This ratio is not moving fast. Paying more for tape drives may reduce these errors, but RAIT hides them all.

This paper assumes an error rate of 1% of all tape handling operations.

Combining the RAM size of a machine requiring on the order of 200 tapes for a single dump of the memory, and a 1% chance of a problem on each tape, the probability of an error somewhere in that volume set is 75%. On write, the job can restart the operation. But in read, the data is lost. This leads to the requirement for mirroring in addition to striping.

In addition to failed operations, tape drive error recovery requires mechanical motion that takes perceptible time. This is time that may actually be taken away from the supercomputer's productivity. Tape has (and will continue to) take seconds to back-hitch, minutes to reposition if the location is lost, and hours to rebuild media Table of Contents. These numbers are for all tape vendors. The situation may be that a drive goes into error recovery, and thus be late compared to the other drives. A potential improvement in the RAIT system can be to let late drive come back into the mix later, if at all

Reliability of RAIT

Reliability with RAIT has been published before^[3]. While this paper has a tape reliability in terms of 10^{-3} probability of an error with a tape operation, a more informal method polled a large tape shop, and their expectation is to have an error every 100 cartridges, or 10^{-2} error rate. These are quite different and can reflect reality when other factors like libraries and Fibre Channel errors are factored in. A rigorous study of tape system reliability, similar to the one accomplished recently for disk^[4], would be a valuable

¹ <http://www.ncsa.uiuc.edu/BlueWaters/>, accessed April 8, 2009

² http://searchstorage.techtarget.co.uk/news/column/0,294698,sid181_gci1295968,00.html#, accessed April 8, 2009

³ J. Hughes, C. Milligan, J. Debiez, "High Performance RAIT" , [Tenth NASA Goddard Conference on Mass Storage Systems and Technologies and Nineteenth IEEE Symposium on Mass Storage Systems](#), Adelphi, Maryland, USA, April 2002, [pdf](#)

⁴ E. Pinheiro, W.D. Weber and L.A. Barroso, "Failure Trends in a Large Disk Drive Population", Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST'07), February 2007, http://labs.google.com/papers/disk_failures.pdf downloaded April 8, 2008

result. Just as the real world disk failure rates showed that there is a disconnect between vendor claims and customer experience, the same could be true of tape.

If one assumes a 400 cartridge Striped group, which is not mirrored, these two error rates give the probability of a 40% a tape operation failing, to a probability that there will be 4 failures. Both of these are unacceptable failure rates.

If one even assumes a 10 cartridge striped group, the failure probabilities are 1% or 10%. These numbers are also unacceptable. Currently, the only recourse is to mirror the data.

In HPSS, the way that volume groups are handled, the reliability of RAIT with one parity can be estimated as the square of the error rate (i.e. the probability of 2 unrelated failures). Even one parity will take the failure rate down to 10^{-4} or 10^{-6} which is one in 10,000 or one in a million full reads of the dataset. This also assumes that whenever a failure occurs, the failure is repaired.

With 3 parities the numbers are 10^{-6} and 10^{-9} .

Existing HPSS Features Applicable to RAIT

The HPSS RAIT architecture is a software implementation running on configurations very similar to what existing customers have already deployed. Being 100% implemented in software, performance improvements are enabled by Moore's Law, and can be implemented on commodity hardware. HPSS already has enterprise scale flexibility and these features dovetail nicely with the existing HPSS Striping design and business model^[5].

The HPSS system currently implements a multiple stripe solution (up to 16 tapes wide) that includes mounting a volume group from a pool of tape devices, concurrent transfer from application to tape devices, write recovery by mounting new volume group, read recovery using multiple copies, and the ability to repack a volume group to reclaim empty space on a volume. The HPSS RAIT architecture extends this multiple stripe solution to include parity generation, parity storage, and data recovery.

Several HPSS reliability features that enable striped operation will be beneficial to RAIT operation. Currently, if there is a write failure, the movers could continue to have the same behavior that they have now, that is, declare end of virtual volume, mount a new set and ask the application to start that segment again. This allows a guarantee that a RAIT group is not degraded when written.

The striped mirror read reliability features also help RAIT. Currently, if a drive fails during a striped read operation on a tape group that has a mirror, HPSS dismounts the current set and repositions to the same segment and can continue the operation. This similar functionality can be used to reconfigure the RAIT movers to recreate a RAIT volume group that encounters excessive read errors beyond the parity width.

⁵ <http://www.hpss-collaboration.org/hpss/about/HPSSOverview2007.pdf>, accessed April 8, 2009

HPSS RAIT Architecture

It is a natural progression for HPSS to base its RAIT implementation on the current HPSS striped virtual volume implementation. The interface for an administrator to configure a RAIT volume will be enhanced to specify a number of data blocks (say d) and a number of redundancy blocks (say p). An erasure-resilient coding scheme will be used to build the redundancy block, it would allow recovery with the loss of up to p physical tape volumes and produce a $d + p$ striped volume.

A popular and widely documented erasure-resilient coding scheme is maximal distance separable (MDS) codes. One specific way to customize a Reed-Solomon code (using a Cauchy matrix) to implement a MDS code is outlined in^[6]. Additional work has been done at analyzing performance of various Reed-Solomon based implementations^{[7][8]}. There is at least one patent^[9] that identifies how using just XOR operations an MDS code can be implemented. There appears to be several good candidate coding schemes to use to build HPSS RAIT redundancy blocks.

Because of compression issues (redundancy data probably being much less compressible than user data), it would make much more sense to block-interleave the redundancy blocks across all the physical tapes in the RAIT virtual volume. This technique is employed by RAID 6 disk arrays to provide redundancy in the presence of multiple disk failures. Figure 1 show this for a 6+2 striped virtual volume.

T1	T2	T3	T4	T5	T6	T7	T8
D1	D2	D3	D4	D5	D6	C1	C2
C1	C2	D1	D2	D3	D4	D5	D6
D1	D2	C1	C2	D3	D4	D5	D6
D1	D2	D3	D4	C1	C2	D5	D6
D1	D2	D3	D4	D5	D6	C1	C2
C1	C2	D1	D2	D3	D4	D5	D6

T = Tape
D = Data Block
C = Redundancy Block

Figure 1

⁶ <http://wwwcs.uni-paderborn.de/cs/ag-bloemer/forschung/publikationen/erasure.pdf>

⁷ <http://www.cs.utk.edu/~plank/plank/papers/FAST-2009.pdf>

⁸ <http://www.usenix.org/events/fast/tech/slides/plank.pdf>

⁹ S. Winograd, B.M. Trager, "Method for constructing erasure correcting codes whose implementation requires only XORs", US Patent number 7350126, assigned to IBM.

The keystone of the RAIT solution is the “RAIT Engine” layer in the data stream between the clients/movers and tape movers. This new component would handle a data stream as if it were a HPSS mover. Its responsibility would be to gather stripes of data from the clients/movers. Each engine would calculate redundancy blocks and build an image of the actual tape stripe. The tape movers would then pull the completed data and redundancy blocks from the RAIT Engine.

A new “RAIT Controller” component in the Core Server would coordinate the control of the RAIT engines and the associated tape movers. The controller would start some number of RAIT engines and $d+p$ movers. The number of RAIT engines per transfer will dictate the level of parallelism of the transfer. This type of architecture is depicted in Figure 2. This figure demonstrates a write from c client/mover machines, using 4 RAIT Engines, to an array of 6 tape drives in a 4+2 configuration.

This technique of ‘pre-constructing’ the tape stripe in the RAIT Engine was found to result in the least network overhead^[10] ($h(p/hd) + (c-1)/c + hd(1/hd) = p/d + (c-1)/c + 1$).

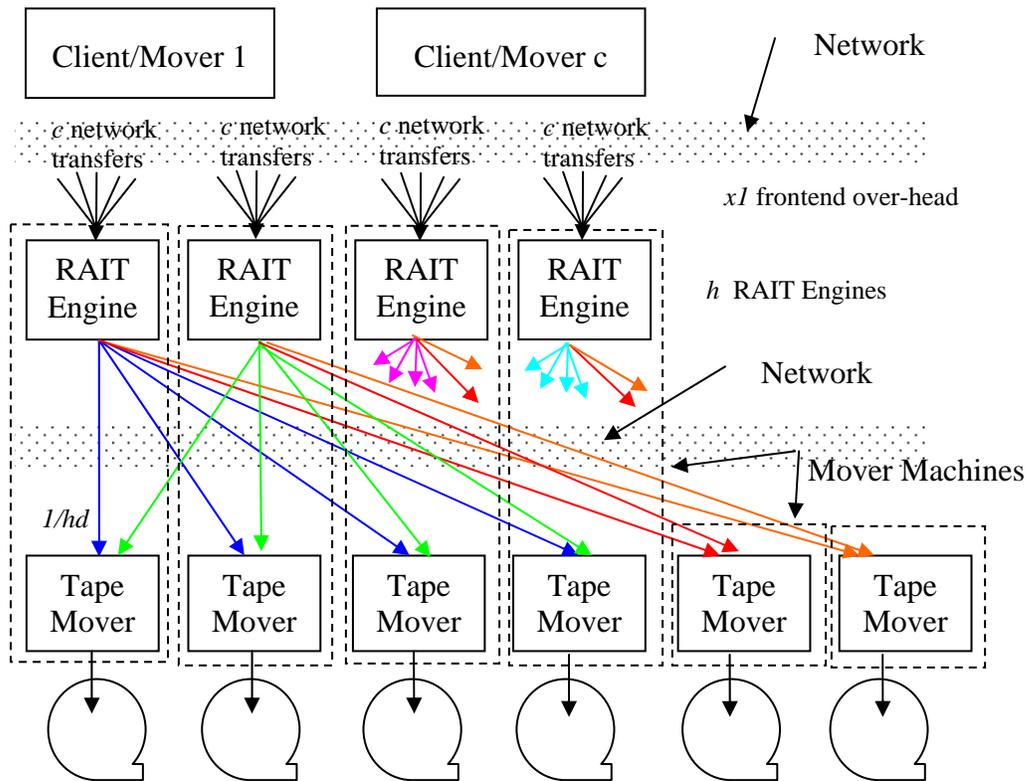


Figure 2

¹⁰ HPSS RAIT Network Overhead Analysis, July 01, 2009.

The RAIT Controller would process the client IOD (or issue a REPLYWHENREAYD IOD to the source movers) and use the network address to build a control commands for the RAIT Engines. Then the RAIT Engines could be started and sent this network address information via a control message. Once successfully connected to the clients/movers, the RAIT Engines would create listen sockets of there own and pass this information back in the reply to the RAIT Controller. The RAIT Controller would then have enough information to build the IODs to issue to the tape movers. This interchange would need to be reversed to support tape aggregation (i.e. passive tape movers).

It is also believed that with minor additions to the current HPSS metadata, we could implement the RAIT function without having to write any additional metadata to the physical tapes. The RAIT Engine could easily differentiate redundancy from data blocks, by knowing the rotation schema, stripe width and starting redundancy block position index. Stating redundancy block position would be determined using a modulo function of the tape section number.

Failure Recovery Scenarios

The failure modes are somewhat different for reads and writes, and in both cases are accomplished without job failure.

Writing RAIT Volumes

The writing of RAIT volumes will require that data flow through the RAIT engines, where it is interleaved with the generated redundancy, and then moved to the tape movers. The IODs to be sent to the tape movers will be constructed so that each tape mover requests data to be written next from the correct RAIT engine. HPSS currently has the concept of a striped IOD descriptor that describes the magnitude of the stride the Mover must make in the storage segment address space to accomplish the recording of a single element within the striped volume. For the RAIT Tape Mover IOD, the sink descriptor will consist of a striped device addresses and the source descriptor will consist of a striped network address. The network stripe will consist of network addresses for the participating RAIT engines, and will indicate to the tape movers which RAIT engine should be contacted for a particular tape stripe.

The control messages to the RAIT engines will need to relate the number of data and redundancy elements, the number of RAIT engines, as well as, its RAIT engine index. Each Mover will return, in the IOR, the amount of data written for the IOD and the final position on tape. The final position could be recorded as the next offset to write for the corresponding tape. The RAIT Controller would be responsible for determining how much of the data written was redundancy and adjusting the IOR to the higher layers to reflect the amount of end user data actually written.

The failures are handled with the existing striped group error recovery that marks this group “end of volume” and then mounts the next group and tells the host to start the segment again.

Reading RAIT Volumes

The reading of RAIT volumes will come in these two flavors:

Normal

Normal or non-recovery reads should take place when the source RAIT volume has all of its volumes marked as accessible, and there are no data errors detected. In this case, the IODs are built to ask the tape movers to read all the data off tape. It is much faster to read all the blocks and just toss the unneeded redundancy blocks than to position around them. There are two ways to approach this. The easiest of these is to just have the RAIT engines read the entire tape stripe and not use the redundancy blocks, unless needed. The other solution would entail modifications to the Mover to never send the redundancy data blocks to the RAIT engines. Figure 3 shows the data flow for a normal read.

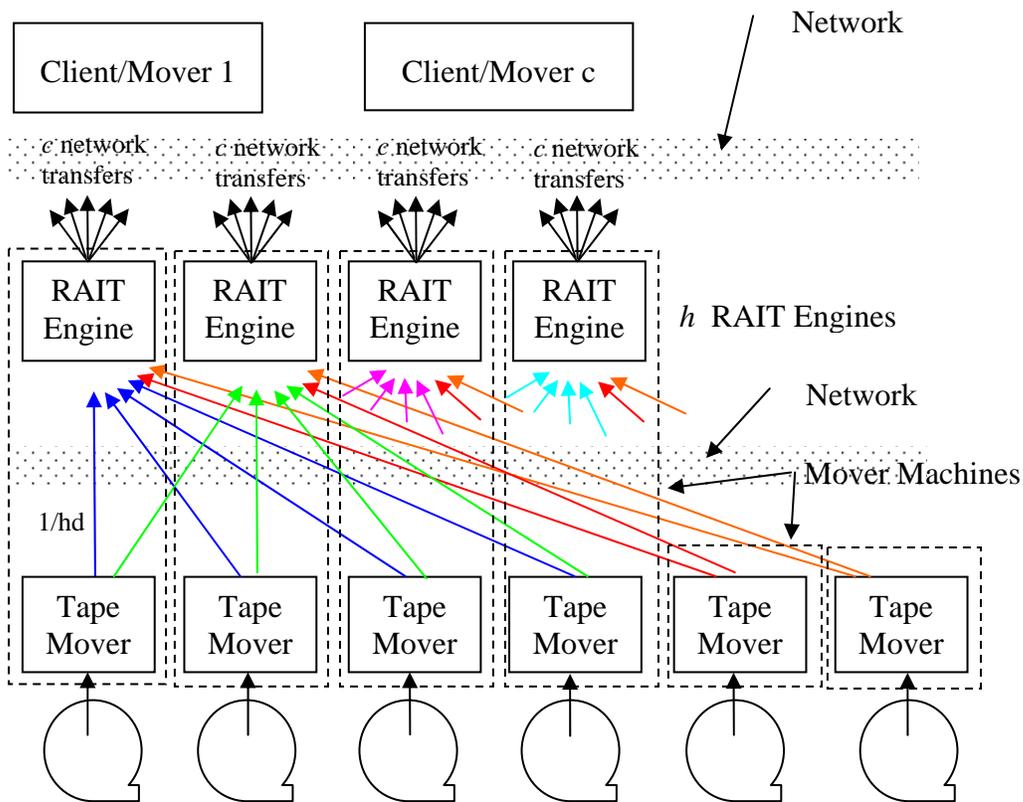


Figure 3

Recovery

A recovery read should take place when the source RAIT volume has any of its volumes marked as inaccessible or if there is a read error. In this case, the IODs and other control messages (those for the RAIT engines) are built to indicate the missing physical tape(s). The PVL will also need the ability to mount part of a RAIT volume, not requiring the mount of the inaccessible tape volumes. For recovery to progress, d blocks (either data or redundancy) for every stripe must be directed to the RAIT engine that will be re-

generating any missing data block(s). The case where two tapes can't be mounted is demonstrated in Figure 4, below.

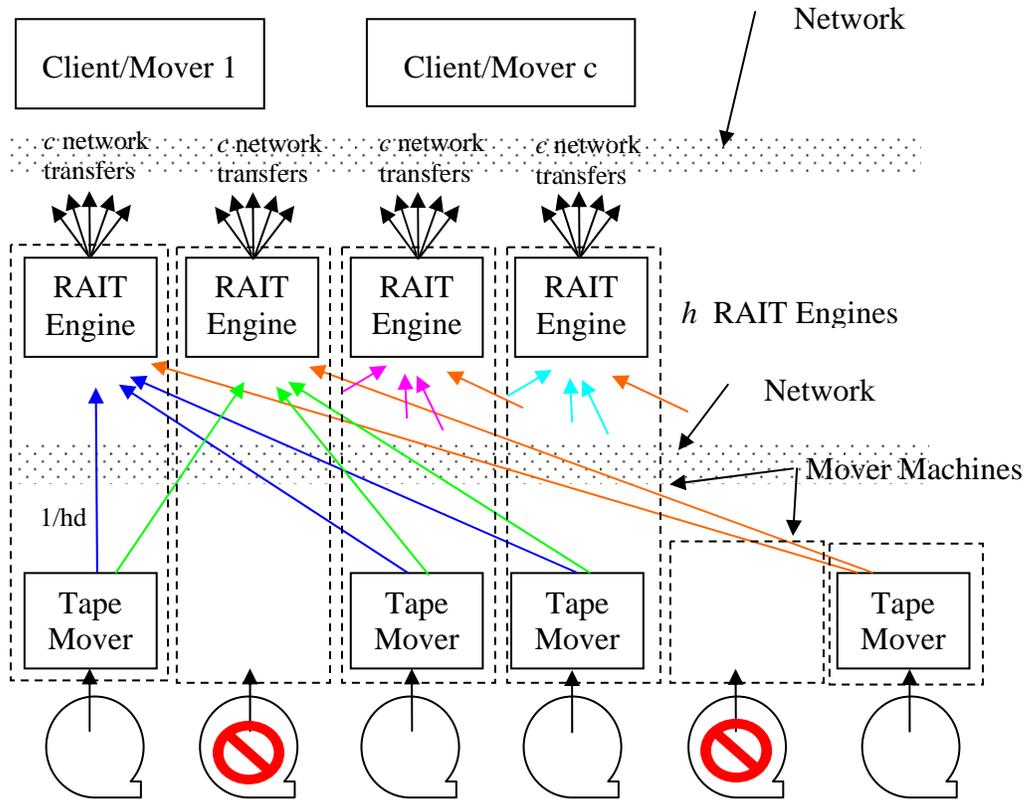


Figure 4

Offline Repack and Data Recovery

The existing capability to repack a volume will be enhanced to support RAIT volumes. This includes mounting fewer than the total number of tapes for the volume and reconstruction of missing data / redundancy blocks. The repack utility program will continue to just request relocation of storage segments. The core server functions will handle the mount errors and regeneration of missing data or redundancy segments.

Proof of Concept

To study the feasibility of the architecture and verify performance, a prototype will be developed. This effort will be concentrated on building a rudimentary RAIT engine, test driver and test client application. The driver will be a simple application that drives the Movers and RAIT engines outside of HPSS. Hard coded values will be used to describe

the configuration of the storage devices and characteristics of the data to be stored. The exercise will allow performance of the design to be verified using the HPSS Mover, outside of the full HPSS system. In addition, building the test driver application should give insight into some of the functionality the RAIT Controller will need to provide. The majority of the RAIT Engine prototype code can be salvaged to produce a production ready application.