Rapid resynchronization for replicated storage

Activity-logging for DRBD

Philipp Reisner

27th January 2004

Abstract

The use of DRBD in real world applications has shown that resynchronization takes too much time. This paper discusses an algorithm to speed up resynchronization of primary nodes unexpectedly leaving the cluster (crashing).

1 Introduction

DRBD consists of a Linux kernel module and some user level tools. Its purpose is to do online storage replication. As such it can be used to build *shared nothing* high availability (HA) clusters out of Linux nodes. See [PFI98] for a discussion of HA clusters and an in-depth comparison of shared nothing and shared storage HA clusters. See also linux-ha `http://linux-ha.org`, which covers the topic of HA on Linux.

To get an in-depth introduction to DRBD, have a look at [REI00]. There you will find the discussion of asynchronous mirroring vs synchronous mirroring, the problem of write ordering and how to find the node with the up-to-date data after a complete cluster restart.

DRBD is not the only effort in this area. It is also possible to combine Linux's software raid driver called *multiple devices driver for Linux* (MD) and the *network block device* (NBD) driver to achieve similar functionality, described in [CLE03].

The main focus of this paper is to discuss the new developments in DRBD, which make DRBD suitable for storage sizes in the range of terra bytes. I will also compare it to the developments of the MD and NBD combination.

2 Overview

With current DRBD clusters, a full resynchronization is performed each time the primary node leaves the cluster (i.e. crashes). However it turned out that quite frequently improper handling by the operator is the cause for a full resynchronization.

During a full resynchronization process, the data on the standby node is not in consistent state, and therefore the cluster is still in degraded mode, thus a failure of the primary node would lead to complete outage of the service.

In order to avoid the need for a full resynchronization process upon the rejoining of a former cluster member, two problems need to be solved. (Consider node A as the surviving node, node B crashed as primary and rejoining the cluster.)

1. We need to know which blocks were modified in the meantime by node A.

2. We need to know which blocks were modified by node B before it crashed.

The first issue was already addressed in previous DRBD releases by the quick-sync bitmap. A solution to the second problem is the activity-logging scheme presented in this paper.

2.1 Activity-logging

The intention of the activity-log is to mark a limited area of the storage as active. In case of a node crash, this active area is a superset of the blocks mentioned in item 2 of the previous section. The required properties are: (1) A representation of the activity log must be stored in non-volatile memory; (2) All updates to the activity log must be transactional; (3) We want to minimize the write operations to the activity log, to maximize overall performance.

For *activity-logging* we see the storage in 4 MB sized *extents*. Each of these extents can be either active or inactive. If a block (usually 4 KB) is to be written to the storage, DRBD first has to ensure that the corresponding extent is active.

2.1.1 Sizing considerations

An extent is identified by a 32 bit number, therefore DRBD will be able to deal with storage sizes up to 16384 TB $(4\text{MB} \cdot 2^{32} = 2^{22} \cdot 2^{32} = 2^{54}$ Byte$)$. While the IO Subsystem of Linux-2.5.x and higher uses 64 bit sector numbers and 512 byte sectors, it can deal with storage sizes of up to 8192 EB $(2^{64} \cdot 2^{9} = 2^{73}$ Byte$)$.

3 Expected Speedup

As an example we consider a storage with a size of 1 TB, and a system that is capable of resynchronizing 8 MB per second. With conventional DRBD we would expect a resynchronization time of $2^{40} \div (8 \cdot 2^{20}) = 2^{17}$ seconds $\simeq 36$ hours.

If we assume that with *activity-logging* we have an active set size of 1 GB (= 256 extents) we have a resynchronization time of $2^{30} \div (8 \cdot 2^{20}) = 2^{7}$ seconds $\simeq 2$ minutes.

4 Implementation

The user will have to configure the maximum number of active extents. By this number he has to balance the trade-off between resynchronization time and the number of updates to the activity-log. A too small number will result in frequent updates to the log, therefore degrading write performance, a too high number will result in a long resynchronization time.

4.1 In memory representation

The active extents will be represented by a hash-table in core-memory. There is a hash-anchor table and a table of elements. The components of the elements are:

```
struct hlist_node collision;  // 8 byte
struct list_head list;        // 8 byte
unsigned int refcnt;          // 4 byte
unsigned int lc_number;       // 4 byte, a total of 24 byte (28)
```

The *lc_number* identifies the extent, *collision* links the hash-table-collision-chains, the double linked list *list* is used to implement the replacement policy. All extents not in the hash-table are in inactive state. The references counter is increased when an IO operation is started, it is decreased when an IO operation is completed.

Since the maximum number of active extents is known, all the elements of the hash-table are preallocated. The implementation uses the hlist implementation introduced with Linux-2.6.

At an active set size of 1 GB, this gives a memory usage of 7 KB (actually 28 byte for each entry, since the hash anchor table itself has 4 bytes per entry).

4.1.1 Replacement policy

In case the free list is empty, an active extent is set back to inactive and therefore it is evicted from the hash table. The extent, the blocks of which have not been written for the longest time, is chosen for eviction (=LRU policy).

If a write operation is carried out on an extent's block, the extent gets moved to the head of the access list. At any time the least recently used extent can be found at the end of the list. An extent that becomes active (is added to the hash table) gets inserted at the head of the access list.

In case the references counter is not zero, the eviction from the hash table has to be delayed until the references counter reaches zero.

In most textbooks on operating systems (like [STA95]) the LRU algorithm is considered as the best replacement policy for these kinds of cache replacement problems.

4.1.2 Shrinking the active set

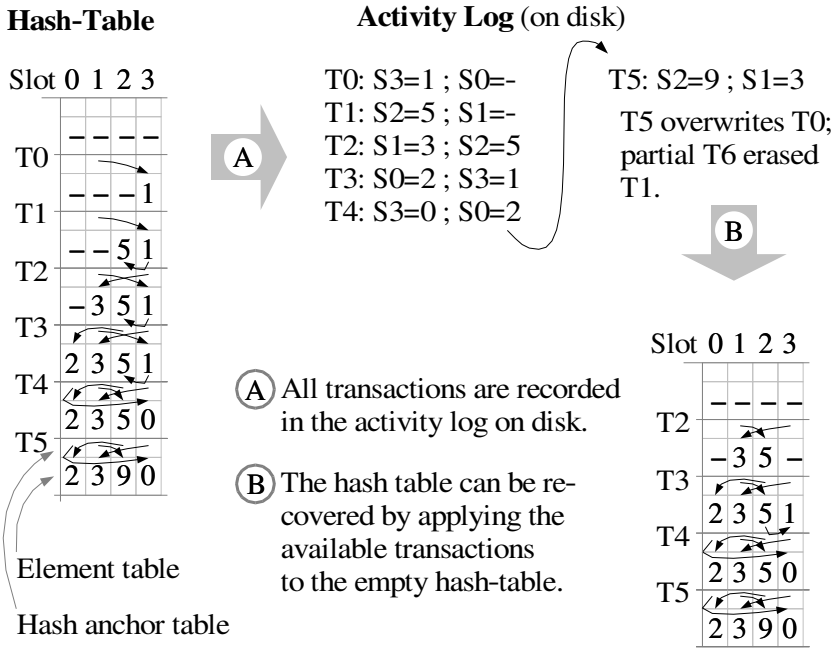
Since a smaller active set implies a shorter resynchronization time, the least recently used extent is dropped from the active set at a tunable time interval (some seconds to minutes). The active set is shrunken to half of the configured maximum.

4.2 On disk data structure

One interesting property of the presented algorithm is that the extent number of only one slot changes upon insertion of a new extent. This is also true if an older extent is evicted from the table in the same transaction.

The activity log is represented by individual transactions (512 Byte) on disk. In each of these transactions a transaction number, up to 62 extent numbers with slot numbers and a trivial checksum are stored. The first of the 62 tuples is reserved for the slot that is changed with this transaction, the remaining 61 are filled in a cyclic scheme from the table.

In the following example we see a hash table with 4 slots and assume that beside the tuple for the transaction one tuple is available for storing the hash-table in a cyclic scheme. The extents number 1,5,3,2,0 and 9 are added to the data structure. The machine crashes and needs to recover from the on disk log.



Storing the whole table in cyclic order is essential if one extent is not evicted from the hash table for a full cycle of transactions. E.g. consider a pattern of extents 1,5,3,2,1,0,9; at this point you would not find the extent number 1 in your activity log if you did not have the additional cyclic scheme.

4.3 Quick-sync bitmap

In previous DRBD releases the quick-sync bitmap was kept in memory. Writing each update to the quick-sync bitmap to disk would have caused too much overhead. With the introduction of the activity-log, it becomes feasible to maintain a copy of the quick-sync bitmap on disk. A 128 byte chunk of the bitmap is written to disk if the corresponding extent gets set to inactive.

In case the degraded node is restarted, it can recover its quick-sync bitmap from the on disk representation. The area covered by the activity-log is completely marked as out-of-sync.

5 Comparison to MD+NBD

In the MD+NBD approach an *intent log* [CLE03] is used. The intent log is a bitmap on disk and a table of counters in memory. Currently each bit represents 1024 bytes of the storage medium. Before any write request may start, the on bit disk must be set. The counter is used to track the number of open pending write requests. The *mdflushd* kernel thread tries to set all bits back to zero every 5 seconds.

Compared to DRBD's activity log, the intent log provides a rather weak hot-spot detection. I think that a decent detection of the current working set (or hot-spot) is an important prerequisite for an efficient storage replication solution, since file systems, Linux's disk schedulers and database systems are trying to maximize performance by optimizing for adjacent disk accesses.

6 Summary

Activity-logging is only performed on the primary node of a DRBD cluster. If this node leaves the cluster (i.e. crashes) and joins the cluster later, it recovers its activity log and can therefore dramatically reduce the number of blocks it has to get from its peer.

I want to stress that the presented activity-log differs in purpose and content from the bitmap used for quick-resynchronization. In the bitmap a degraded cluster (the only remaining node) records the modifications it performs to its storage in order to sync-up a former member node.

References

[STA95] Stallings, William. *Operating Systems – 2nd edition*. Prentice-Hall International, London, 1995

[PFI98] Gregory F. Pfister. *In search of Clusters 2nd edition.* Prentice-Hall PTR, Upper Saddle River 1998.

[REI00] Philipp Reisner. *DRBD – Distributed Replicated Block Device*. 9th International Linux System Technology Conference, September 4-6, 2002 in Cologne, Germany. http://www.drbd.org/fileadmin/drbd/publications/drbd_lk9.pdf

[CLE03] Paul Clements and James E.J. Bottomley. *High Availability Data Replication.* Proceedings of the Linux Symposium, July 23th-26th, 2003 in Ottawa, Canada. http://archive.linuxsymposium.org/ols2003/Proceedings/All-Reprints/Reprint-Clements-OLS2003.pdf