Fast Clone Deletion

Sara Hartse | Delphix
Outline

- Clone deletion now
- Fast deletion algorithm
- Algorithm scalability
- Performance gains
Clone Deletion

- Clones are mutable copies of existing datasets
- Copy on write means that creating a clone is as simple as pointing to the root of a given snapshot
- Throughout the course of the clone’s lifetime it diverges from the original
- Deleting a clone requires determining which blocks are still shared with the snapshot and which blocks are unique to the clone
  - Iterate over on-disk tree, ignore sections based on birth time
Best Case

Contiguous Writes
Worst Case

Sparse writes
Time to Delete Clone with 500MB of New Data

- **random**
- **contiguous**

Time to delete the clone (seconds)

Size of original dataset (G)

<table>
<thead>
<tr>
<th>Size of original dataset (G)</th>
<th>random</th>
<th>contiguous</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.9</td>
<td>1.14</td>
</tr>
<tr>
<td>2</td>
<td>4.74</td>
<td>1.05</td>
</tr>
<tr>
<td>5</td>
<td>4.64</td>
<td>1.67</td>
</tr>
<tr>
<td>10</td>
<td>5.42</td>
<td>3.43</td>
</tr>
<tr>
<td>20</td>
<td>13.98</td>
<td>7.5</td>
</tr>
<tr>
<td>30</td>
<td>16.52</td>
<td>13.8</td>
</tr>
</tbody>
</table>
Fast Delete

● Keep track of clone specific writes and deletes as they occur
● Store them in a **livelist**
● To delete the clone, just have to process each element in the livelist
● Work is proportional to the number of writes to the clone
Livelist algorithm

- Enqueue blockpointers \textit{allocated} and \textit{freed} on the clone as the writes occur.

- When it’s time to delete the clone, determine the not yet freed blocks and free them:
  - Step backwards through the livelist: insert frees into an AVL tree, check for membership of allocs in the AVL tree.
Start at the end of the Livelist
Insert block 4 in AVL tree
Check for block 5 in AVL tree. Free it.
Check for block 4 in AVL tree. Ignore it.
Insert block 3 into AVL tree
Insert block 1 into AVL tree
Check for block 3 in AVL tree. Ignore it.
Check for block 2 in AVL tree. Free it.
Check for block 1 in AVL tree. Ignore it.
Pros

- Deletion work is now proportional to the number of writes to the clone
- Low insertion cost - we know exactly where to put the block pointers

Cons

- Livelist can grow arbitrarily large and we’ll have to load the whole thing into memory to delete the clone
- Tricky to destroy incrementally
Sublists

- Break livelist into smaller sublists
- Decide which sublist to insert into based on birth time
- How big should they be?
- Natural way to implement incremental destroy
Asynchronous Destroy

- Want to limit the amount of work we do per sync
  - Only destroy one sublist each transaction group
- Loading a sublist into memory could be very expensive
- Some delete work must be synchronous and some can be in the background

```bash
> zfs destroy clone

Store livelist id in pool
Signal thread
>

Load livelist into memory
Determine blkptrs to delete
Call synctask

Free blkptrs
Update livelist info in pool
Pros

● Limited how much memory is loaded in at once
● Can delete quickly and incrementally

Cons

● Number of sublists can grow arbitrarily large
  ○ The more sublists we have, the more costly insertion is
  ○ Disk space
Condensing sublists

- After a block is freed, the livelist contains irrelevant information
- We can condense the list to store only what we need
Merging sublists

- Now we can merge smaller sublists and reduce their overall number
In Summary

- Made the work of deleting a clone proportional to the number of writes to that clone using a **livelist**
- Limited memory loaded at once using **sublists**
  - Makes it easier to delete **incrementally** and **asynchronously**
- Slowed the growth of the number of sublists by periodically **condensing** the sublists
Least Improvement: contiguous writes

Time taken to destroy: existing method v/s livelist

- Continuously writing 50MB of data: old_way 0.028 minutes vs livelist 0.023 minutes
- Continuously writing 500MB of data: old_way 0.075 minutes vs livelist 0.069 minutes
Most Improvement: sparse writes

Time taken to destroy a clone: existing method v/s liveliest

- **existing_method**
- **livelist**

<table>
<thead>
<tr>
<th>Data Written in MB</th>
<th>Existing Method</th>
<th>Livelist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone has 50GB of data, write 50MB of sparse data</td>
<td>4.6 minutes</td>
<td>0.033 minutes</td>
</tr>
<tr>
<td>Clone has 500GB of data, write 500MB of sparse data</td>
<td>46.06 minutes</td>
<td>0.33 minutes</td>
</tr>
</tbody>
</table>
Conclusion

- Livelist method of clone deletion gives dramatic performance improvements in the worst case scenarios
  - Gains in the best case as well
- Tweaks were needed to make the algorithm scalable for production use
  - Balancing space and efficiency
- Coming soon!
Thank you!

Sowrabha Gopal

Matt Ahrens

Serapheim Dimitropoulos
Questions?