SSDSupport

Current Situation

ATS currently supports SSDs, however it does not take the best advantage of their special characteristics. In particular, SSDs could be used as an addition level in the cache (between RAM and traditional platter-based hard drives (HDDs)), and SSDs could benefit from reducing writes. Currently we write all misses to disk but we could write only (for example) on seeing the second miss to a particular URL.

Use Cases

- large working set, few HDDs, low RAM reverse proxy
- large working set, many HDDs, probably forward proxy, but perhaps shard CDN
- no moving parts forward or reverse proxy, 0 HDDs

NOTE: situations with a small working set and adequate RAM fall will not benefit performance-wise from an SSD, however because of the relatively limited number of write cycles supported by SSDs currently, specialized SSD support might improve longevity and reliability.

Features

- support for re-writing warm content to an SSD if read from spinning media
  - this is uses the SSD as an additional level in the hierarchy between RAM and HDD
- support for only writing the second miss in an SSD-only environment

The Interim Caching Layer

From V6.0.0, the interim caching feature is removed, please consider other options.

When we thinking about the storage of ATS, the original design is to support many disks, with the same size (best work with block devices without raid), and build up the volumes (partitions), then assign each of the volumes to some domain (hostname), we find out that if we don't make big change in the storage design, we don't have a easy way to archive the multi-tiered caching storage, then we come to the following INTERIM solution:

- we think that if your deployed mixed storage, the fast device is less. IE, 1/10 of the slow devices, in size or in counts.
- we assume that in most case, 8 fast disks is fairly many.
- we assume your slow disk should not go beyond 32TB per disk.
- the slow devices holding all the data, should not loss them when fast disk fails.
- compare to the storage percent, ie 10%, the fast devices may loss data during the server restart.
- the fast disks should balance to all the slow disks, in size and even in IOPS, the volumes is building upon the slow disks, with slices from every disks, we should spread the load on each fast disk too.

we make a design with balance:

- a interim cache of the cache, which is lived on the fast disks (SSD in most case), will loss the data if you restart the server process. make the data on interim caching device a real interim.
- we support 8 fast disks at most
- we make a block level interim cache, which does not contains any index info

what we have done:

- we steal 4bit from the Dir struct, to identify each of the fast disks
- we cut down the up limit of the disk in size to 32TB
- we split each fast disks into volumes configured in the volume.config, and bind them.
- we store the interim data in the dir on the slow storages
- write data from the origin server on the slow disks
- we setup a interal LRU list, limited in size (1M bucket), and bump the busy blocks into fast device
- we do hot data evocation on the interim caching devices, to make it more efficient, we do compare the blocks with the LRU list
- the interim cache device (fast disks), works with RRD writing too.

coins and pins:

- coins:
  - we make a solid solution without big change in the storage architecture
  - the LUR help us get the hot blocks, it is efficient
  - the block layer interim caching can help on small objects and big ones too
- pins:
  - loss data if the server process crash Fixed in TS-2275
  - limited in block device only for used as interim caching device
the interim caching device space is not a add-on, but a copy of the hot data on the slow devices
- we set lower the max disk size of the storage from 0.5PB to 32TB.
- the interim caching function is not enabled by default configuration

**codes & structs:**

the change of Dir:

```c
@@ -155,15 +157,42 @@ struct FreeDir
    unsigned int reserved:8;
    unsigned int prev:16; // (2)
    unsigned int next:16; // (3)
+    #if TS_USE_INTERIM_CACHE == 1
+      unsigned int offset_high:12; // 8GB * 4K = 32TB
+      unsigned int index:3; // interim index
+      unsigned int ininterim:1; // in interim or not
+    #else
+      uint16_t offset_high; // 0: empty
+    #endif

    FreeDir() { dir_clear(this); }
```

we split the stat of read_success into disk, interim and ram:

```c
@@ -2633,6 +2888,11 @@ register_cache_stats(RecRawStatBlock *rsb, const char *prefix)
+      REG_INT("interim.read.success", cache_interim_read_success_stat);
+      REG_INT("disk.read.success", cache_disk_read_success_stat);
+      REG_INT("ram.read.success", cache_ram_read_success_stat);
    REG_INT("write.active", cache_write_active_stat);
    REG_INT("write.success", cache_write_success_stat);
    REG_INT("write.failure", cache_write_failure_stat);
```

**how to enable interim caching:**

configure to enable the interim caching function:
add `--enable-interim-cache` option.

we introduced two record config:

- `proxy.config.cache.interim.storage`:
  - disk device(s) for the interim cache, we only support block devices in full path, multiple disks may separate by SPACE. example: LOCAL proxy.
  - `config.cache.interim.storage STRING /dev/sdb /dev/sdc1`
- `proxy.config.cache.interim.migrate_threshold`:
  - controls how many times one content should be migrate to the interim cache from the storage, which will control the writing on the interim disk.
  - default to `2`, which means it should be seen in the LRU list two times before we thinking of migrate it to the interim caching device.

**some of the result:**

we have a system with 160G SSD + 3 * 500G SAS, 16G RAM, 4 cores, here is the result of tsar and iostat -x:
<table>
<thead>
<tr>
<th>Time</th>
<th>qps</th>
<th>cons</th>
<th>Bps</th>
<th>rt</th>
<th>rpc</th>
<th>hit</th>
<th>ramhit</th>
<th>band</th>
<th>ssdhit</th>
</tr>
</thead>
<tbody>
<tr>
<td>24/06/13-10:30</td>
<td>901.83</td>
<td>18.89</td>
<td>22.6M</td>
<td>17.36</td>
<td>47.74</td>
<td>87.30</td>
<td>68.08</td>
<td>88.90</td>
<td>22.49</td>
</tr>
<tr>
<td>24/06/13-10:35</td>
<td>934.12</td>
<td>18.88</td>
<td>22.0M</td>
<td>14.34</td>
<td>49.47</td>
<td>87.60</td>
<td>68.53</td>
<td>90.70</td>
<td>22.21</td>
</tr>
<tr>
<td>24/06/13-10:40</td>
<td>938.14</td>
<td>18.92</td>
<td>21.7M</td>
<td>15.36</td>
<td>49.58</td>
<td>87.70</td>
<td>68.02</td>
<td>89.50</td>
<td>22.45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>avg-cpu:</th>
<th>%user</th>
<th>%nice</th>
<th>%system</th>
<th>%iowait</th>
<th>%steal</th>
<th>%idle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5.47</td>
<td>0.00</td>
<td>15.62</td>
<td>25.09</td>
<td>0.00</td>
<td>53.82</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device:</th>
<th>rqqm/s</th>
<th>wrqm/s</th>
<th>r/s</th>
<th>w/s</th>
<th>rsec/s</th>
<th>wsec/s</th>
<th>avgrq-sz</th>
<th>avgqsz</th>
<th>svctm</th>
<th>%util</th>
</tr>
</thead>
<tbody>
<tr>
<td>sda</td>
<td>0.00</td>
<td>7.33</td>
<td>25.67</td>
<td>3.33</td>
<td>1600.00</td>
<td>1438.00</td>
<td>104.76</td>
<td>0.45</td>
<td>15.46</td>
<td>12.17</td>
</tr>
<tr>
<td>sdb</td>
<td>0.00</td>
<td>0.00</td>
<td>28.67</td>
<td>11.33</td>
<td>1461.00</td>
<td>8723.00</td>
<td>254.60</td>
<td>0.74</td>
<td>18.47</td>
<td>11.21</td>
</tr>
<tr>
<td>sdc</td>
<td>0.00</td>
<td>0.00</td>
<td>25.67</td>
<td>2.00</td>
<td>2178.00</td>
<td>1373.33</td>
<td>128.36</td>
<td>0.40</td>
<td>14.05</td>
<td>11.04</td>
</tr>
<tr>
<td>sdd</td>
<td>0.00</td>
<td>0.00</td>
<td>196.00</td>
<td>4.00</td>
<td>14790.00</td>
<td>2823.00</td>
<td>88.06</td>
<td>0.13</td>
<td>0.66</td>
<td>0.41</td>
</tr>
</tbody>
</table>

...