NFD - Task #1706

Reduce implicit digest computation in ContentStore

06/26/2014 11:39 PM - Junxiao Shi

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Junxiao Shi	**% Done:**	100%
Category:	Tables	**Estimated time:**	6.00 hours
Target version:	v0.3		

Description

Every time a Data packet is inserted to the ContentStore, its full Name (including the implicit digest component) is requested, which involves implicit digest computation; benchmark shows that **implicit digest computation accounts for 48% time in a find-insert scenario**.

The purpose of requesting a full Name upon insertion is to determine the relative order between two Data packets that have the same Name (without implicit digest), and to determine whether a cached Data can satisfy an Interest that may contain an implicit digest as its last Name component or exclude a range of implicit digests.
However, most Data packets have unique Names so their ordering can be determined without knowing the full Name, and most Interests do not appear to have an implicit digest as its last Name component and do not appear to exclude a range of implicit digests.
Therefore, we can **optimize for the expected case**, to **eliminate implicit digest computation unless when necessary**.

This Task is to implement the idea in tables-concept-cs_20140117.pptx, so that implicit digest computation is performed only when necessary.

Related issues:

Related to ndn-cxx - Task #1707: Reduce implicit digest computation in Intere...	**Closed**
Blocked by ndn-cxx - Task #1640: ImplicitSha256DigestComponent	**Closed**
Blocked by NFD - Task #2254: Rewrite ContentStore based on ContentStore stub	**Closed**

History

#1 - 06/26/2014 11:45 PM - Alex Afanasyev

Can you link/reference the benchmark?

#2 - 06/26/2014 11:46 PM - Junxiao Shi

The benchmark is done with a modified version of tests/other/cs-smoketest.cpp.

- add data->getFullName() in the dataWorkload preparation loop, so that implicit digest is computed before CS insertion; or comment out this line, and CS insertion algorithm would have to compute implicit digest
- replace the loop of varying nInsertions with a single execution of 70000 insertions; this is needed because dataWorkload array would contain implicit digests after the first execution

The results are:

- CS insertion computes implicit digest: 62.0152 seconds
- pre-computed implicit digest - CS insertion does not compute implicit digest: 32.3143 seconds
- time saving: 48%

#3 - 06/26/2014 11:53 PM - Junxiao Shi

- Related to Task #1707: Reduce implicit digest computation in Interest::matchesData added

#4 - 08/27/2014 09:55 AM - Junxiao Shi

- Assignee set to Yi Huang

- Target version set to v0.3

#5 - 09/01/2014 09:47 PM - Yi Huang

- Status changed from New to In Progress

#6 - 09/01/2014 09:50 PM - Junxiao Shi

@Yi, don't hurry with the implementation of this Task. ContentStore needs a major refactoring, and this is only part of it. Do read the slides.

#7 - 09/12/2014 12:59 PM - Junxiao Shi

@Yi did a benchmark on a Macbook Pro:

- CS insertion computes implicit digest: 0.443175 seconds
- pre-computed implicit digest - CS insertion does not compute implicit digest: 0.165358 seconds
- time saving: 63%

Implicit digest computation consumes a greater percentage of time in CS insertion on Macbook.

Another observation is: the same 70000 insertions is 100+ times faster than my benchmark on a Ubuntu 12.04 VM.

It's surprising to see such a significant difference.
This difference is likely to be more than hardware.

I don't remember what compiler options I was using when doing the benchmark.
Yi should repeat the benchmark on Ubuntu using the same compiler options that were used on Macbook.

#8 - 09/24/2014 03:29 PM - Junxiao Shi

Since tables-concept-cs_20140117.pptx is written, Data::getFullName function is introduced in ndn-cxx.
This function computes the implicit digest upon first invocation, and caches the result internally.
It would be a duplicate to store a copy of full Name outside of the Data object.

A solution to this problem is: create a Compare of Data where Data::getFullName invocation is minimized, in order to establish an ordered sequence.
CS algorithms never expressly test whether implicit digest has been computed, but only obtain full Name when it's required for a comparison.

#9 - 09/28/2014 04:54 PM - Yi Huang

Here is my design of implementing CS:

What does not change:

- All public functions have the same name so there is no need to rewrite code outside the CS.
- I did not think of a better way to keep the cleaning index. Therefore, I am keeping the use of boost::multi_index_container.
- There is not many changes to current CS Entry. I just removed things that manipulate skiplist iterators.

Using SkipList implementation from repo-ng:

- The SkipList can be created with a type and a comparator of that type. Inserting and Finding elements takes a object of that type. Here's my design of using the SkipList:
  - To reduce unnecessary digest computation, the comparator function should first compare "getName()". If "getName()" appear to be the same on both side, compare "getFullName()".
  - Since inserting and finding in SkipList requires a object of the type of element in the list, we need to construct a new CS entry once Cs::insert is called (same for Cs::find). This is because the SkipList cannot search object by index. I don't know whether this is tolerable overhead.

When insert to CS:

1. A new CS entry with new data will be created.
2. Call SkipList::find to see whether there is an entry with the same name (including digest if needed) already there in SkipList.
3. If yes:
   1. check unsolicited. If old entry is not unsolicited and new entry is, discard the new entry and return.
   2. Update stale time then return.
4. If no:
   1. Evict one entry if Cs is full.
   2. Insert the new entry to CleanupIndex
   3. Insert the new entry into SkipList then return.

When lookup (This is basically the steps described in Junxiao's slides. Moving forward in SkipList is done by using iterator provided by SkipList in 2nd step):

1. A new CS entry with new data will be created.
2. Use "lower_bound()" to locate starting point. And set nameLength to the number of components in the Interest, set lastMatch to nil.
3. If last component in Interest Name may be an implicit digest, compute the digest of current CS entry.
4. If Interest Name is not a prefix of current CS entry's Name plus implicit digest if computed, goto step 9.
5. if current CS entry violates MinSuffixComponents, MaxSuffixComponents, PublisherPublicKeyLocator, Exclude, MustBeFresh selectors, go to step 8.
6. If ChildSelector prefers leftmost child, return current CS entry.
7. If ChildSelector prefers rightmost child, and ((lastMatch is nil) or (current CS entry and lastMatch have different nameLength-th component)), set lastMatch to current CS entry.
8. Move to next CS entry in the ordered sequence, and goto step 3.
9. Return lastMatch.

When evict entry:

1. Unsolicited entries are evicted first.
2. Stale entries are evicted next.
3. If there is no unsolicited or stale entry, other entries are evicted by the order they are created.
4. This can be maintained by boost::multi_index_container.

For periodically cleanup:

- We decided not to do this during our last meeting with Beichuan.

Additional questions:

- Since now we are using SkipList from repo-ng, I think we need to make NFD depends on it. I am not very familiar with waf. How do I do that?
- Like I mentioned above. Is creating new CS entry just for finding existing entry in SkipList a tolerable overhead?

#10 - 09/28/2014 05:20 PM - Junxiao Shi

- File repo-index_20140617.pptx added

An alternative to SkipList is to add an ordered index to the multi-index container, where the order is determined by full Name.
Please explain why SkipList is chosen over an ordered index.

If SkipList is proven to be beneficial, constructing an Entry during lookups is acceptable because time complexity of doing so is constant.
Such Entry should be placed on the stack rather than the heap.

Please justify the choice of eviction single entry during insertion, over periodical mass evictions.

repo-index_20140617.pptx describes two Interest matching algorithms when ChildSelector=rightmost.
The algorithm in note-9 is the "alternate" described in these slides, and it is inefficient in certain cases.
I suggest using the first algorithm.

NFD should never depend on repo-ng.
I suggest moving SkipList to ndn-cxx, if SkipList is proven to be beneficial for ContentStore.

#11 - 09/29/2014 01:02 PM - Yi Huang

I am actually not sure whether our skiplist is going to out perform multi-index container. I did a quick look up on multi-index container's performance. It says the implementation is either matching or better than performance of STL containers.

http://www.boost.org/doc/libs/1_56_0/libs/multi_index/doc/performance.html

#12 - 09/29/2014 01:25 PM - Junxiao Shi

Neither STL nor Boost has a SkipList.
The SkipList is implemented by repo-ng author, and is not STL container.

Please do some analysis or benchmark to compare which one of the following is faster:

- SkipList for name ordering, Multi-Index Container for cleanup
- Multi-Index Container for both name ordering and cleanup

#13 - 10/08/2014 02:11 PM - Alex Afanasyev

- Related to Feature #1207: CS policy interface added

#14 - 10/17/2014 08:34 AM - Junxiao Shi

- Status changed from In Progress to New

- Assignee deleted (Yi Huang)

Yi is assigned to other tasks, so this Task is back to New status. The whole ContentStore redesign work is blocked by *ImplicitSha256DigestComponent* spec approval and library support.

#15 - 11/05/2014 09:26 AM - Junxiao Shi

- Blocked by Task #1640: ImplicitSha256DigestComponent added

#16 - 11/17/2014 09:25 PM - Junxiao Shi

A possible starting point of this Task is http://gerrit.named-data.net/447 patchset 4.

Although the commit is called "stub", it's a much cleaner ContentStore.

#17 - 12/14/2014 08:51 PM - Junxiao Shi

- Related to deleted (Feature #1207: CS policy interface)

#18 - 12/14/2014 08:51 PM - Junxiao Shi

- Blocked by Task #2254: Rewrite ContentStore based on ContentStore stub added

#19 - 12/15/2014 01:00 PM - Junxiao Shi

- Assignee set to Shashwat Vidhu Sher

#20 - 01/27/2015 12:05 AM - Alex Afanasyev

Does the merged CS implementation implements the delayed digest calculation or not yet?

#21 - 01/27/2015 12:28 PM - Shashwat Vidhu Sher

- Assignee deleted (Shashwat Vidhu Sher)

Dropping the task due to parallel committments

#22 - 01/28/2015 06:13 PM - Junxiao Shi

- Status changed from New to In Progress

- Assignee set to Junxiao Shi

Before I begin implementation, here's a baseline benchmark:

test case	platform	regular	without digest computation	saving
find(miss)-insert	Ubuntu 14.10	7412611	4023103	46%
find(miss)-insert	OSX 10.9	3037630	1814464	40%
insert-find(hit)	Ubuntu 14.10	6799756	3731573	45%
insert-find(hit)	OSX 10.9	3028404	1825252	40%

Time reported is the total duration for the timed portion, in microseconds.

"without digest computation" comes from a modified benchmark where Data::getFullName is invoked on every Data packet before timed portion.

"saving" is calculated as (1 - "without digest computation" / regular).

The projected saving is an upper bound of the actual saving when this Task completes, because:

- Digest computation is necessary when relative order between a Data packet and another Data packet with same Name must be determined, and in certain query conditions.
- Additional complexity caused by this Task would increase constant factor.

#23 - 01/28/2015 07:07 PM - Junxiao Shi

- % Done changed from 0 to 40

http://gerrit.named-data.net/1688

The initial implementation is uploaded as commit 56591cd366368630b84bb06fbd3778ba71b12012.

Benchmark result is unsatisfactory:

test case	platform	current	projected	56591cd3
find(miss)-insert	Ubuntu 14.10	7412611	4023103	7130155
find(miss)-insert	OSX 10.9	3037630	1814464	2904101
insert-find(hit)	Ubuntu 14.10	6799756	3731573	7106507
insert-find(hit)	OSX 10.9	3028404	1825252	2915203

There might be hidden getFullName call somewhere.
If I pre-compute the digest, benchmark results are: 3776805, 1754864, 3874197, 1758086.

#24 - 01/28/2015 07:42 PM - Junxiao Shi

- Status changed from In Progress to Code review

- % Done changed from 40 to 100

I added a counter of expected getFullName calls into commit 56591cd366368630b84bb06fbd3778ba71b12012, and it's surprising to see that getFullName is called 600000 times in either test case.

This is caused by the benchmark design:
The benchmark generates 50000 unique Names, and each Name is reused four times to make 200000 Data packets, which are inserted into a ContentStore of 50000 capacity.
The 50001st Data being inserted has the same Name as the 1st Data. This requires a comparison between them in order to determine the relative order, which would require getFullName on both Data packets.

The solution is to change the benchmark to generate 100000 unique Names.
The 100001st Data being inserted will have the same Name as the 1st Data, but the 1st Data is already evicted by that time.

The new benchmark is in commit 9b842ad6913a2066aa74fa58eedb151916a9d33e.

New benchmark results (including baseline) are: (note that each test case now has 400000 iterations)

test case	platform	current	projected	9b842ad6
find(miss)-insert	Ubuntu 14.10	18374417	9441264	9591541
find(miss)-insert	OSX 10.9	7625666	4877360	4722637
insert-find(hit)	Ubuntu 14.10	17559825	9859690	9975561
insert-find(hit)	OSX 10.9	7574497	4962484	4799287

I believe this result is good enough.

#25 - 02/01/2015 07:12 PM - Junxiao Shi

> I can think of one potential issue with multiple data packets with the same name (different digests). What is the intended behavior for CS? Keep just one copy?

At API level, ContentStore is not obligated to store or keep any packet.

In current design, the ContentStore will store all Data packets with same Name but different digests.
This behavior is tested in DigestOrder and DigestExclude test cases.

The computation cost is bounded:
When second Data packet with same Name arrives, implicit digests are computed for both Data packets.
When third or more Data packet with same Name arrives, implicit digest is computed for the incoming Data packets, while cached Data packets already have their implicit digests computed.

At most two implicit digest computations can occur per insertion.

#26 - 02/02/2015 08:32 PM - Junxiao Shi

- Status changed from Code review to Closed

Files

tables-concept-cs_20140117.pptx	62.9 KB	06/27/2014	Junxiao Shi
repo-index_20140617.pptx	45.5 KB	09/29/2014	Junxiao Shi