Edit Code

Jump to bottom

WIP: Store preferable infohashes for queries #7786

Draft **qstokkink** wants to merge 1 commit into `Tribler:main` from `qstokkink:add_user_activity`

Conversation 65 Commits 1 Checks 16 Files changed 15

qstokkink commented 2 weeks ago • edited Member

Related to #7632

This PR adds integrations with local and remote search results and stores the preferable torrent (according to what the user clicked and downloaded).

Currently, this integration is only local and the propagation of preferable torrents depends on the emergent effect of them being checked more often in the torrent checker, causing them to be propagated in the content discovery community more. In a future PR, we can implement more direct interaction with other peers to share recommendations.

The uncovered lines of this PR are in a `TYPE_CHECKING` block and, therefore, impossible to cover.

qstokkink changed the title ~~**WIP: Store preferrable infohashes for queries**~~ **WIP: Store preferable infohashes for queries** 2 weeks ago

qstokkink force-pushed the `add_user_activity` branch 6 times, most recently from `f794284` to `78b304f` 2 weeks ago Compare

qstokkink marked this pull request as ready for review 2 weeks ago

qstokkink requested a review from **Tribler/reviewers** as a code owner 2 weeks ago

qstokkink requested review from **drew2a** (assigned from **Tribler/reviewers**) and removed request for **Tribler/reviewers** 2 weeks ago

qstokkink changed the title ~~**WIP: Store preferable infohashes for queries**~~ **READY: Store preferable infohashes for queries** 2 weeks ago

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

Comment on lines +31 to +38

```
31 +    with db_session():
32 +        queries = layer.Query.select()[:]
33 +
34 +        assert len(queries) == 1
35 +        assert queries[0].query == "test query"
36 +        assert len(queries[0].infohashes) == 1
37 +        assert list(queries[0].infohashes)[0].infohash == b"\x00" * 20
38 +        assert float_equals(list(queries[0].infohashes)[0].preference, 1.0)
```

drew2a 2 weeks ago Member

later. It also confines the `with` block to only include lines that actually need the `db_session`. The suggested version avoids multiple, yet identical, conversions to a list and retrieving the first element of the list.

Suggested change

```
31  -    with db_session():
32  -        queries = layer.Query.select()[:]
33  -
34  -        assert len(queries) == 1
35  -        assert queries[0].query == "test query"
36  -        assert len(queries[0].infohashes) == 1
37  -        assert list(queries[0].infohashes)[0].infohash == b"\x00" * 20
38  -        assert float_equals(list(queries[0].infohashes)[0].preference, 1.0)
31  +    with db_session():
32  +        test_query = layer.Query.get()
33  +        infohashes = list(test_query.infohashes)
34  +
35  +    assert test_query.query == "test query"
36  +    assert len(infohashes) == 1
37  +
38  +    infohash = infohashes.pop()
39  +    assert infohash.infohash == b"\x00" * 20
40  +    assert float_equals(infohash.preference, 1.0)
```

Commit suggestion | Add suggestion to batch

Reply...

Resolve conversation

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

Comment on lines +48 to +55

```
48  +        queries = layer.Query.select()[:]
49  +        winner, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x00" * 20)[:]
50  +        loser, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x01" * 20)[:]
51  +
52  +        assert len(queries) == 1
53  +        assert queries[0].query == "test query"
54  +        assert float_equals(winner.preference, 1.0)
55  +        assert float_equals(loser.preference, 0.0)
```

drew2a 2 weeks ago • edited — Member

NIT: the version avoids unnecessary `select` querying and list copying:

Suggested change

```
48  -        queries = layer.Query.select()[:]
49  -        winner, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x00" * 20)[:]
50  -        loser, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x01" * 20)[:]
51  -
52  -        assert len(queries) == 1
53  -        assert queries[0].query == "test query"
54  -        assert float_equals(winner.preference, 1.0)
```

```
+        winner = layer.InfohashPreference.get(lambda x: x.infohash == b"\x00" * 20)
+        loser = layer.InfohashPreference.get(lambda x: x.infohash == b"\x01" * 20)
+
+    assert float_equals(winner.preference, 1.0)
+    assert float_equals(loser.preference, 0.0)
+
+    assert test_query.query == "test query"
```

Commit suggestion | Add suggestion to batch

Reply...

Resolve conversation

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

```
+    """
+    Test that queries with a loser can be stored and retrieved.
+    """
+    layer.store("test query", InfoHash(b"\x00" * 20), {InfoHash(b"\x01" * 20)})
```

drew2a 2 weeks ago — Member

From the text of the test, it is not clear why one infohash is called "winner" and another is called "loser". I'm not questioning the naming here (which I will do in the class `UserActivityLayer`).

Here, I suggest helping the reader by showing that the "loser" is just an infohash that passes as the third function parameter, and in the `with` statement, you retrieve this infohash.

Suggested change

```
-    layer.store("test query", InfoHash(b"\x00" * 20), {InfoHash(b"\x01" * 20)})
+    layer.store("test query", infohash=InfoHash(b"\x00" * 20), losing_infohashes={InfoHash(b"\x01" * 20)})
```

Commit suggestion | Add suggestion to batch

Reply...

Resolve conversation

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

Comment on lines +67 to +78

```
+        queries = layer.Query.select()[:]
+        winner, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x00" * 20)[:]
+        loser_1, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x01" * 20)[:]
+        loser_2, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x02" * 20)[:]
+        loser_3, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x03" * 20)[:]
+
+        assert len(queries) == 1
+        assert queries[0].query == "test query"
```

```
78 +         assert float_equals(loser_3.preference, 0.0)
```

drew2a 2 weeks ago • edited

Member

NIT: More compact yet correct version of the same logic. It uses `select` where it is intended to query a list and `get` when it is supposed to be just a single entity.

Suggested change

```
67 -         queries = layer.Query.select()[:]
68 -         winner, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x00" * 20)[:]
69 -         loser_1, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x01" * 20)[:]
70 -         loser_2, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x02" * 20)[:]
71 -         loser_3, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x03" * 20)[:]
72 -
73 -         assert len(queries) == 1
74 -         assert queries[0].query == "test query"
75 -         assert float_equals(winner.preference, 1.0)
76 -         assert float_equals(loser_1.preference, 0.0)
77 -         assert float_equals(loser_2.preference, 0.0)
78 -         assert float_equals(loser_3.preference, 0.0)
67 +         test_query = layer.Query.get()
68 +         winner = layer.InfohashPreference.get(lambda x: x.infohash == b"\x00" * 20)
69 +         losers = list(layer.InfohashPreference.select(lambda x: x.infohash != b"\x00" * 20))
70 +
71 +     assert test_query.query == "test query"
72 +     assert float_equals(winner.preference, 1.0)
73 +     assert len(losers) == 3
74 +     assert all(float_equals(ls.preference, 0.0) for ls in losers)
```

Commit suggestion | Add suggestion to batch

Reply...

Resolve conversation

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

Comment on lines +62 to +64

```
62 +     layer.store("test query", InfoHash(b"\x00" * 20), {InfoHash(b"\x01" * 20),
63 +                                                        InfoHash(b"\x02" * 20),
64 +                                                        InfoHash(b"\x03" * 20)})
```

drew2a 2 weeks ago • edited

Member

NIT: It is easier to read the test function if you specify the argument names:

Suggested change

```
62 -     layer.store("test query", InfoHash(b"\x00" * 20), {InfoHash(b"\x01" * 20),
63 -                                                        InfoHash(b"\x02" * 20),
64 -                                                        InfoHash(b"\x03" * 20)})
62 +     layer.store("test query", infohash=InfoHash(b"\x00" * 20), losing_infohashes={InfoHash(b"\x01" * 20),
63 +                                                                                  InfoHash(b"\x02" * 20),
64 +                                                                                  InfoHash(b"\x03" * 20)})
```

Reply...

Resolve conversation

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

Comment on lines +93 to +104

```
93  +         queries = layer.Query.select()[:]
94  +         entry_1, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x00" * 20)[:]
95  +         entry_2, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x01" * 20)[:]
96  +         entry_3, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x02" * 20)[:]
97  +         entry_4, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x03" * 20)[:]
98  +
99  +         assert len(queries) == 1
100 +         assert queries[0].query == "test query"
101 +         assert float_equals(entry_1.preference, 0.2)
102 +         assert float_equals(entry_2.preference, 0.8)
103 +         assert float_equals(entry_3.preference, 0.0)
104 +         assert float_equals(entry_4.preference, 0.0)
```

drew2a 2 weeks ago — Member

NIT: the version avoids unnecessary select querying and list copying:

Suggested change

```
93  -         queries = layer.Query.select()[:]
94  -         entry_1, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x00" * 20)[:]
95  -         entry_2, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x01" * 20)[:]
96  -         entry_3, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x02" * 20)[:]
97  -         entry_4, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x03" * 20)[:]
98  -
99  -         assert len(queries) == 1
100 -         assert queries[0].query == "test query"
101 -         assert float_equals(entry_1.preference, 0.2)
102 -         assert float_equals(entry_2.preference, 0.8)
103 -         assert float_equals(entry_3.preference, 0.0)
104 -         assert float_equals(entry_4.preference, 0.0)
93  +         test_query = layer.Query.get()
94  +         entry_1 = layer.InfohashPreference.get(lambda x: x.infohash == b"\x00" * 20)
95  +         entry_2 = layer.InfohashPreference.get(lambda x: x.infohash == b"\x01" * 20)
96  +         entry_3 = layer.InfohashPreference.get(lambda x: x.infohash == b"\x02" * 20)
97  +         entry_4 = layer.InfohashPreference.get(lambda x: x.infohash == b"\x03" * 20)
98  +
99  +     assert test_query.query == "test query"
100 +     assert float_equals(entry_1.preference, 0.2)
101 +     assert float_equals(entry_2.preference, 0.8)
102 +     assert float_equals(entry_3.preference, 0.0)
103 +     assert float_equals(entry_4.preference, 0.0)
```

Commit suggestion | Add suggestion to batch

Reply...

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

```
106 +
107 + def test_store_delete_old(layer: UserActivityLayer) -> None:
108 +     """
109 +     Test result decay after updating.
```

drew2a 2 weeks ago — Member

The description of the test likely suffers from missing specifications about the difference between it and the previous test.

Reply...

Resolve conversation

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

Comment on lines +119 to +132

```
119 +         queries = layer.Query.select()[:]
120 +         entry_1, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x00" * 20)[:]
121 +         entry_2, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x01" * 20)[:]
122 +         entry_3, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x02" * 20)[:]
123 +         should_be_dropped = layer.InfohashPreference.select(lambda x: x.infohash == b"\x03" * 20)[:]
124 +         entry_4, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x04" * 20)[:]
125 +
126 +         assert len(queries) == 1
127 +         assert queries[0].query == "test query"
128 +         assert float_equals(entry_1.preference, 0.2)
129 +         assert float_equals(entry_2.preference, 0.0)
130 +         assert float_equals(entry_3.preference, 0.0)
131 +         assert should_be_dropped == []
132 +         assert float_equals(entry_4.preference, 0.8)
```

drew2a 2 weeks ago — Member

NIT: the version avoids unnecessary select querying and list copying:

Suggested change

```
119 -         queries = layer.Query.select()[:]
120 -         entry_1, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x00" * 20)[:]
121 -         entry_2, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x01" * 20)[:]
122 -         entry_3, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x02" * 20)[:]
123 -         should_be_dropped = layer.InfohashPreference.select(lambda x: x.infohash == b"\x03" * 20)[:]
124 -         entry_4, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x04" * 20)[:]
125 -
126 -         assert len(queries) == 1
127 -         assert queries[0].query == "test query"
128 -         assert float_equals(entry_1.preference, 0.2)
129 -         assert float_equals(entry_2.preference, 0.0)
130 -         assert float_equals(entry_3.preference, 0.0)
131 -         assert should_be_dropped == []
132 -         assert float_equals(entry_4.preference, 0.8)
```

```
+        entry_2 = layer.InfohashPreference.get(lambda x: x.infohash == b"\x01" * 20)
+        entry_3 = layer.InfohashPreference.get(lambda x: x.infohash == b"\x02" * 20)
+        entry_4 = layer.InfohashPreference.get(lambda x: x.infohash == b"\x04" * 20)
+        should_be_dropped = layer.InfohashPreference.get(lambda x: x.infohash == b"\x03" * 20)
+
+    assert test_query.query == "test query"
+    assert float_equals(entry_1.preference, 0.2)
+    assert float_equals(entry_2.preference, 0.0)
+    assert float_equals(entry_3.preference, 0.0)
+    assert float_equals(entry_4.preference, 0.8)
+    assert not should_be_dropped
```

Commit suggestion | Add suggestion to batch

Reply...

Resolve conversation

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

Comment on lines +148 to +161

```
+        queries = layer.Query.select()[:]
+        entry_1, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x00" * 20)[:]
+        entry_2, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x01" * 20)[:]
+        entry_3, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x02" * 20)[:]
+        entry_4, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x03" * 20)[:]
+        entry_5, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x04" * 20)[:]
+
+        assert len(queries) == 1
+        assert queries[0].query == "test query"
+        assert float_equals(entry_1.preference, 0.2)
+        assert float_equals(entry_2.preference, 0.0)
+        assert float_equals(entry_3.preference, 0.0)
+        assert float_equals(entry_4.preference, 0.0)
+        assert float_equals(entry_5.preference, 0.8)
```

drew2a 2 weeks ago — Member

NIT: the version avoids unnecessary select querying and list copying:

Suggested change

```
-        queries = layer.Query.select()[:]
-        entry_1, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x00" * 20)[:]
-        entry_2, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x01" * 20)[:]
-        entry_3, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x02" * 20)[:]
-        entry_4, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x03" * 20)[:]
-        entry_5, = layer.InfohashPreference.select(lambda x: x.infohash == b"\x04" * 20)[:]
-
-        assert len(queries) == 1
-        assert queries[0].query == "test query"
-        assert float_equals(entry_1.preference, 0.2)
-        assert float_equals(entry_2.preference, 0.0)
-        assert float_equals(entry_3.preference, 0.0)
-        assert float_equals(entry_4.preference, 0.0)
-        assert float_equals(entry_5.preference, 0.8)
```

```
150 +         entry_2 = layer.InfohashPreference.get(lambda x: x.infohash == b"\x01" * 20)
151 +         entry_3 = layer.InfohashPreference.get(lambda x: x.infohash == b"\x02" * 20)
152 +         entry_4 = layer.InfohashPreference.get(lambda x: x.infohash == b"\x03" * 20)
153 +         entry_5 = layer.InfohashPreference.get(lambda x: x.infohash == b"\x04" * 20)
154 +
155 +     assert test_query.query == "test query"
156 +     assert float_equals(entry_1.preference, 0.2)
157 +     assert float_equals(entry_2.preference, 0.0)
158 +     assert float_equals(entry_3.preference, 0.0)
159 +     assert float_equals(entry_4.preference, 0.0)
160 +     assert float_equals(entry_5.preference, 0.8)
```

Commit suggestion | Add suggestion to batch

Reply...

Resolve conversation

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

```
167 +     """
168 +     layer.store("test query", InfoHash(b"\x00" * 20), {InfoHash(b"\x01" * 20)})
169 +
170 +     assert layer.get_preferable(b"\x00" * 20) == b"\x00" * 20
```

drew2a 2 weeks ago — Member

There is a type mismatch: "Expected type 'InfoHash', got 'bytes' instead"

Reply...

Resolve conversation

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

```
176 +     """
177 +     layer.store("test query", InfoHash(b"\x00" * 20), {InfoHash(b"\x01" * 20)})
178 +
179 +     assert layer.get_preferable(b"\x01" * 20) == b"\x00" * 20
```

drew2a 2 weeks ago — Member

There is a type mismatch: "Expected type 'InfoHash', got 'bytes' instead"

Reply...

Resolve conversation

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

```
185 +     """
186 +     layer.store("test query", InfoHash(b"\x00" * 20), {InfoHash(b"\x01" * 20)})
187 +
188 +     assert layer.get_preferable(b"\x02" * 20) == b"\x02" * 20
```

drew2a 2 weeks ago — Member

There is a type mismatch: "Expected type 'InfoHash', got 'bytes' instead"

Reply...

Resolve conversation

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

```
198 +     random_selection = layer.get_random(limit=1)
199 +
200 +     assert len(random_selection) == 1
201 +     assert list(random_selection)[0] == b"\x01" * 20
```

drew2a 2 weeks ago — Member

This is an unnecessary conversion to a list and retrieval of the first item:

Suggested change

```
201 -     assert list(random_selection)[0] == b"\x01" * 20
201 +     assert random_selection.pop() == b"\x01" * 20
```

Commit suggestion | Add suggestion to batch

Reply...

Resolve conversation

drew2a reviewed 2 weeks ago

View reviewed changes

src/tribler/core/components/database/db/layers/tests/test_user_activity_layer.py

```
20 +
21 + def float_equals(a: float, b: float) -> bool:
22 +     return round(a, 5) == round(b, 5)
23 +
```

drew2a 2 weeks ago — Member

In this file, you are using the same construction over and over again:

```
def test_example(layer: UserActivityLayer) -> None:
    layer.store("test query", InfoHash(b"\x00" * 20), set())

    with db_session():
        queries = layer.Query.select()[:]
```

This is basically a single test case that repeated in `test_store_no_losers`, `test_store_with_loser`, `test_store_weighted_decay`, `test_store_delete_old`, `test_store_delete_old_over_e`.

I would suggest that you extract it to a separate test. Then there will be no need to repeat this in other tests, and they will more accurately describe the specific test case they are testing, without any excess.

Like:

```
def test_store_query(layer: UserActivityLayer) -> None:
    layer.store("test query", InfoHash(b''), set())

    with db_session():
        test_query = layer.Query.get()

    assert test_query.query == "test query"
```

Reply...

Resolve conversation

drew2a requested changes last week

View reviewed changes

drew2a left a comment — Member

The PR appears to be a nice feature that looks promising for Tribler.

It is an interesting concept that appears quite similar to ClickLog. If that's the case, then it would be beneficial to add links to the ClickLog documentation as a reference in the newly added classes and components.

I've suggested a few code improvements and raised some points for discussion.

Also, I'm adding **@kozlovsky** as a reviewer since there is a new database structure implementation involved.

src/tribler/core/components/database/db/layers/user_activity_layer.py

Comment on lines +13 to +23

```
13 + if typing.TYPE_CHECKING:
14 +     @dataclass
15 +     class InfohashPreference:
16 +         infohash: bytes
17 +         preference: float
18 +         parent_query: Query
19 +
20 +     @dataclass
21 +     class Query:
22 +         query: str
23 +         infohashes: typing.Set[InfohashPreference]
```

drew2a 2 weeks ago — Member

These structures are used solely in the `_select_superior` method, and there is no direct transformation into this datatype in the calling code, as they merely replicate the existing structures described in `UserActivityLayer`. Adopting this approach of duplicating definitions necessitates updating the structures twice (once for the original and again for the duplicate), which increases the risk of errors during future updates. The developer responsible for this task should:

1. Be aware that there are two definitions that require changes.
2. Make changes twice, which is more error-prone than making a change once.

My suggestion is to avoid duplication by refactoring the existing code. There are several methods to achieve the same class behavior but without duplication.

Reply...

Resolve conversation

src/tribler/core/components/database/db/layers/user_activity_layer.py

```
52 +         self.Query = Query
53 +         self.InfohashPreference = InfohashPreference
54 +
55 +     def store(self, query: str, infohash: InfoHash, losing_infohashes: typing.Set[InfoHash]) -> None:
```

drew2a 2 weeks ago — Member

I have a point for discussion regarding this function interface. It is the naming.

You're using a "win-lose" representation which I find misleading, as it suggests a game-like process of identifying winners and losers. However, according to your function description, it's not about winning and losing, but rather about determining which infohashes were used (downloaded) and which weren't used (not downloaded).

I suggest reconsidering the naming to choose a more appropriate representation.

Reply...

Resolve conversation

src/tribler/core/components/user_activity/user_activity_component.py

```
43 +         database_component = await self.require_component(DatabaseComponent)  # local_query_results notification
44 +         torrent_checker_component = await self.require_component(TorrentCheckerComponent)
45 +
46 +         self.database_manager: UserActivityLayer = database_component.db.user_activity_layer
```

drew2a 2 weeks ago — Member

The name `self.database_manager` is misleading as it is not a manager but a layer:

Suggested change

```
51 -         self.database_manager: UserActivityLayer = database_component.db.user_activity_layer
51 +         self.user_activity_layer: UserActivityLayer = database_component.db.user_activity_layer
```

Commit suggestion | Add suggestion to batch

Reply...

Resolve conversation

src/tribler/core/components/database/db/layers/user_activity_layer.py

```
67 +         # Update or create a new database entry
68 +         with db_session:
69 +             existing = self.Query.get(query=query)
70 +             if existing is not None:
```

drew2a 2 weeks ago — Member

"flat is better than nested" regarding to the Zen of Python. To decrease nesting in your code you can simply use `get_or_create` function from `pony_utils`:

```
            ...

        if existing.infohashes and infohash in weights:
            weights[infohash] = self.update_weight_new
```

Next nesting level could be removed by using this trick:

```
with db_session:
    existing = get_or_create(self.Query, query=query)
    known_infohashes = (i for i in existing.infohashes if i.infohash in weights)
    unknown_infohashes = (i for i in existing.infohashes if i.infohash not in weights)

    for old_infohash_preference in unknown_infohashes:
        ...

    for old_infohash_preference in known_infohashes:
        ...
```

Also, "readability counts" and "sparse is better than dense." Therefore, two-line formulas could be rewritten as follows:

```
        for infohash_preference in known_infohashes:
            weight = weights.pop(infohash_preference.infohash)
            new_weight = infohash_preference.preference * self.update_weight_old + weight * self.update_weight_new
            infohash_preference.preference = new_weight
```

Therefore, we can significantly simplify the code while retaining the same logic.

I'll add the assembled example with all improvements as a code suggestion.

Reply...

Resolve conversation

src/tribler/core/components/database/db/layers/user_activity_layer.py

Comment on lines +68 to +90

```
68  +         with db_session:
69  +             existing = self.Query.get(query=query)
70  +             if existing is not None:
71  +                 for old_infohash_preference in existing.infohashes:
72  +                     if old_infohash_preference.infohash in weights:
73  +                         new_weight = (old_infohash_preference.preference * self.update_weight_old
74  +                                       + weights.pop(old_infohash_preference.infohash, 0.0) * self.update_weight_new)
75  +                         old_infohash_preference.preference = new_weight
76  +                     else:
77  +                         # This infohash did not pop up, candidate for deletion
78  +                         new_weight = old_infohash_preference.preference * self.update_weight_old
79  +                         if new_weight < self.e:
80  +                             old_infohash_preference.delete()
81  +                         else:
82  +                             old_infohash_preference.preference = new_weight
83  +                 if infohash in weights:
84  +                     weights[infohash] = self.update_weight_new
85  +             else:
86  +                 existing = self.Query(query=query, infohashes=set())
87  +
88  +             for new_infohash, weight in weights.items():
89  +                 existing.infohashes.add(self.InfohashPreference(infohash=new_infohash, preference=weight,
90  +                                                                 parent_query=existing))
```

drew2a 2 weeks ago — Member

Simplified code block:

Suggested change

```
68  -         with db_session:
```

```
-                for old_infohash_preference in existing.infohashes:
-                    if old_infohash_preference.infohash in weights:
-                        new_weight = (old_infohash_preference.preference * self.update_weight_old
-                                      + weights.pop(old_infohash_preference.infohash, 0.0) * self.update_weight_new)
-                        old_infohash_preference.preference = new_weight
-                    else:
-                        # This infohash did not pop up, candidate for deletion
-                        new_weight = old_infohash_preference.preference * self.update_weight_old
-                        if new_weight < self.e:
-                            old_infohash_preference.delete()
-                        else:
-                            old_infohash_preference.preference = new_weight
-                if infohash in weights:
-                    weights[infohash] = self.update_weight_new
-            else:
-                existing = self.Query(query=query, infohashes=set())
-
-            for new_infohash, weight in weights.items():
-                existing.infohashes.add(self.InfohashPreference(infohash=new_infohash, preference=weight,
-                                                                parent_query=existing))
+        with db_session:
+            existing = get_or_create(self.Query, query=query)
+            related_infohashes = (i for i in existing.infohashes if i.infohash in weights)
+            unrelated_infohashes = (i for i in existing.infohashes if i.infohash not in weights)
+
+            for infohash_preference in unrelated_infohashes:
+                # This infohash did not pop up, candidate for deletion
+                new_weight = infohash_preference.preference * self.update_weight_old
+                if new_weight < self.e:
+                    infohash_preference.delete()
+                else:
+                    infohash_preference.preference = new_weight
+
+            for infohash_preference in related_infohashes:
+                weight = weights.pop(infohash_preference.infohash)
+                new_weight = infohash_preference.preference * self.update_weight_old + weight * self.update_weight_new
+                infohash_preference.preference = new_weight
+
+            if existing.infohashes and infohash in weights:
+                weights[infohash] = self.update_weight_new
+
+            for new_infohash, weight in weights.items():
+                existing.infohashes.add(self.InfohashPreference(infohash=new_infohash, preference=weight,
+                                                                parent_query=existing))
```

Commit suggestion | Add suggestion to batch

Reply...

Resolve conversation

src/tribler/core/components/user_activity/user_activity_component.py

```
19 + from tribler.core.sentry_reporter.sentry_reporter import SentryReporter
20 +
21 +
22 + class UserActivityComponent(Component):
```

drew2a last week — Member

NIT: From a design perspective, it would be beneficial to keep the `UserActivityComponent` more declarative at a high level and minimize specific implementation details by extracting them into a separate class (for example, `PreferableChecker` (I don't like the name, it is just an example)). Then the Component code will look neater, and it will be independent of `PreferableChecker` implementation changes.

This approach would make it easier to write and conduct tests separately for the `UserActivityComponent` (to test its composition) and the `PreferableChecker` (to test its logic).

```python
class UserActivityComponent(Component):
    preferable_checker = None

    async def run(self) -> None:
        await super().run()

        # Wait for dependencies
        await self.require_component(ContentDiscoveryComponent)  # remote_query_results notification
        await self.require_component(LibtorrentComponent)  # torrent_finished notification

        database_component = await self.require_component(DatabaseComponent)  # local_query_results notification
        torrent_checker_component = await self.require_component(TorrentCheckerComponent)

        self.preferable_checker = PreferableChecker(
            max_query_history=self.session.config.user_activity.max_query_history,
            torrent_checker=torrent_checker_component.torrent_checker,
            user_activity_layer=database_component.db.user_activity_layer
        )

    async def shutdown(self) -> None:
        await super().shutdown()
        if self.preferable_checker:
            self.preferable_checker.shutdown()
```

qstokkink yesterday — Member, Author

Agreed. I had this in my original design as well (#7632 (comment)) However, the impelementation was so small and trivial that I moved the code here. Note that this code is not any less testable due to this (100% coverage).

Reply...

Resolve conversation

src/tribler/core/components/database/db/layers/user_activity_layer.py

Comment on lines +42 to +50

```
42 +         class Query(database.Entity):
43 +             query = orm.PrimaryKey(str)
44 +             infohashes = orm.Set("InfohashPreference")
45 +
46 +         class InfohashPreference(database.Entity):
47 +             infohash = orm.Required(bytes)
48 +             preference = orm.Required(float)
49 +             parent_query = orm.Required(Query)
50 +             orm.PrimaryKey(infohash, parent_query)
```

drew2a last week — Member

The database does not appear to be normalized. As **@kozlovsky** is coming back from vacation at the same time as you, I summon him to review the database structure.

To me, the database structure looks normalized. It is possible to link `InfohashPreference` with the `Resource` entity, but it is actually not necessary and complicates the database schema a bit.

But I'd like to use integer fields instead of floats, like:

```
class Query(database.Entity):
    query = orm.PrimaryKey(str)
    searched_counter = orm.Required(int, default=1)
    infohashes = orm.Set("InfohashPreference")

class InfohashPreference(database.Entity):
    infohash = orm.Required(bytes)
    parent_query = orm.Required(Query)
    chosen_counter = orm.Required(int, default=0)
    ignored_counter = orm.Required(int, default=0)
    orm.PrimaryKey(infohash, parent_query)
```

This way, changing the formula on how `preference` is calculated becomes possible.

kozlovsky yesterday Member

After some additional thought, I agree with **@drew2a** that it may be better to link the `InfohashPreference` entity with the `Resource` entity in the same way as the `Tracker` entity of the `HealthDataAccessLayer` is linked with the `Resource` entity via the `Tracker.torrents` / `Resource.trackers` relationships.

It has the following benefits:

1. The primary key of `InfohashPreference` becomes shorter, as it now will use resource id instead of infohash bytes.
2. With the current approach of `TriblerDatabase`, the torrent metadata is just a kind of `Resource`, and having `Resource` directly linked with `InfohashPreference` can simplify some future queries.

The drawback is when we want to search `InfohashPreference` knowing the specific info hash, we first need to find the `Resource` and then use it instead of the info hash value.

With this change, the code will be like:

```
class UserActivityLayer:

    def __init__(self, knowledge_layer: KnowledgeDataAccessLayer) -> None:
        self.database = knowledge_layer.instance
        self.Resource = knowledge_layer.Resource

        class Query(self.database.Entity):
            query = orm.PrimaryKey(str)
            searched_counter = orm.Required(int, default=1)
            infohashes = orm.Set("InfohashPreference")

        class InfohashPreference(self.database.Entity):
            torrent = orm.Required(self.Resource)
            query = orm.Required(Query)
            chosen_counter = orm.Required(int, default=0)
            ignored_counter = orm.Required(int, default=0)
            orm.PrimaryKey(torrent, query)

        self.Query = Query
        self.UserPreference = InfohashPreference
```

(In this example, I renamed `parent_query` to `query`, as `parent_` prefix looks unnecessary)

And, in the `Resource` entity of the `KnowledgeDataAccessLayer` we will have:

```
infohash_preferences = orm.Set(lambda: db.InfohashPreference, reverse="torrent")
```

qstokkink yesterday Member Author

I don't see how you could implement decay of previous search and results for the same (infohash, query) with this database format. Is it still possible? Because that is a requirement.

Indeed, it is not enough to implement the proper decay; I missed that requirement. But it leads me to some additional thoughts.

The current scheme implemented in this PR is single-user. I don't think decay is important when the statistics are accumulated only for a single user. But if we aggregate anonymized query-result-preference statistics from thousands of users, the decay indeed makes sense.

But then we have a new question on spreading and accumulating these statistics. It probably should be signed by a second key when gossiping to prevent spam. However, we cannot sign the dynamically decaying `preference` value of the `float` type. We can sign some discrete information that at the moment T, an anonymous user U performed the query Q and clicked on the infohash H.

So, if the goal is to aggregate decaying anonymous user-clicks-at-query-results statistics across multiple users, the discrete signed piece of information should probably be (T, U, Q, H). Then, the decay can be implemented by taking the time stamps into account - the weight of the user's "vote" can be inversely proportional to the vote's age.

In that case, the entity attributes might be something like:

```
class InfohashPreference(self.database.Entity):
    user = orm.Required(User)  # some new entity
    query = orm.Required(Query)
    torrent = orm.Required(self.Resource)
    last_clicked_at = orm.Required(datetime)
    signature = orm.Optional(bytes)  # for remotely received gossips
    # for the next field see https://github.com/Tribler/tribler/pull/7786#discussion_r1439578570
    successfully_downloaded = orm.Required(bool, default=False)  # to ignore local unfinished downloads
    orm.PrimaryKey(user, query, torrent)
```

What do you think?

qstokkink yesterday Member Author

That is close to what I had in mind for the long term, in a different PR. I would prefer we discuss the grand design in the linked issue, not on this PR.

Just to touch on it, in short, the plan for now is to use emergent behavior, as follows:

1. Torrents that are "preferred" have their health checked more frequently locally (this PR).
2. Torrents that have their health checked recently (locally) are more frequently gossiped in the popularity community (already exists).
3. Torrents that are gossiped more by others have a higher chance of appearing in search results remotely (already exists).
4. *Effect:* search results that are gossiped to more users are more likely to be downloaded. For actually popular content, that is downloaded, this forms a feedback loop: back to step 2.

In summary, this PR creates a soft bias and, therefore, an emergent effect that boosts the popularity of content that is searched for and downloaded.

Establishing shadow identities and more aggressively gossiping - while preventing spam - is something I'll leave for a follow-up PR. Ideally, we don't need to gossip preference directly and we can somehow merge gossiped ML models. However, this should only be implemented after careful experimentation. For now, this PR gives each user a local ranking to start the ML experimentation.

kozlovsky 3 hours ago Member

Thanks for the explanation; now I understand your approach better. Initially, I was misguided by the picture in #7632 with a "store signed" label, as my understanding was that it is only possible to sign discrete facts, not float values. If gossiping about individual provable facts is not the intention, then storing calculated preferences is probably fine.

Still, you can consider using `torrent = orm.Required(self.Resource)` instead of `infohash = orm.Required(bytes)` in the `InfohashPreference` entity to reduce the data size.

qstokkink 2 hours ago Member Author

Sure, thank you for the suggestion. Once the initial prototype has been merged, we can look at refactoring and optimizations and I will definitely keep your suggestion in mind. That said, once this has been merged, this PR is also no longer my sole responsibility and others can also contribute their excellent suggestions to the communal code. We can grow the code over time.

I do realize, now, that I left `enabled = True` as the default setting. I'll make sure to keep this disabled by default so we still have the freedom to change things like the database format in future PRs.

Resolve conversation

src/tribler/core/components/user_activity/user_activity_component.py

```
68 +             self.infohash_to_queries[infohash].append(query)
69 +         self.queries[query] = results | self.queries.get(query, set())
70 +
71 +         if len(self.queries) > self.max_query_history:
```

drew2a last week — Member

Perhaps for this purpose (to store a limited amount of data), you could use a dedicated data structure. This would make it possible to better cover it with tests and to use it in other parts of Tribler's code. As a beneficial side effect, it would be easier to understand the logic of the dedicated data structure and of the current method.

We can extend this draft:

```python
class LimitedSizeDict(OrderedDict):
    def __init__(self, *args, size_limit=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.size_limit = size_limit

    def __setitem__(self, key, value):
        super().__setitem__(key, value)
        self._check_size_limit()

    def _check_size_limit(self):
        if self.size_limit is None:
            return
        while len(self) > self.size_limit:
            self.popitem(last=False)
```

qstokkink yesterday — Member, Author

Sure, we can implement such a feature once we have a need for it. However, that is not the feature this PR is implementing and it is therefore best left to another PR.

Reply...

Resolve conversation

src/tribler/core/components/user_activity/user_activity_component.py Outdated

Comment on lines 87 to 99

```
87 +         b_infohash = InfoHash(unhexlify(infohash))
88 +         queries = self.infohash_to_queries[b_infohash]
89 +         for query in queries:
90 +             losing_infohashes = self.queries[query] - {b_infohash}
91 +             fut = get_running_loop().run_in_executor(None,
92 +                                                      self.database_manager.store,
93 +                                                      query, b_infohash, losing_infohashes)
94 +             self.task_manager.register_task("Store query", functools.partial(UserActivityComponent._fut_to_task, fut))
```

drew2a last week — Member

Two points:

1. This method seems overly complicated for its simple purpose — to call the `self.database_manager.store` function.
2. I'm uncertain if the logic will work correctly in cases where the user downloads two or more torrents from the query results.

qstokkink yesterday — Member, Author

This is solving a real issue: it is related to #4320 and #7784.

qstokkink 1 hour ago — Member, Author

I did manage to shrink this a little bit (see conversation with @kozlovsky).

Reply...

Resolve conversation

src/tribler/core/components/user_activity/user_activity_component.py

Comment on lines +104 to +105

```
104 +         for infohash in random_infohashes:
105 +             self.task_manager.register_anonymous_task("Check preferable torrent",
106 +                                                       self.torrent_checker.check_torrent_health, infohash)
```

drew2a last week — Member

Here's a simplified version: This version avoids a for loop for a set of just a single item and avoids an unnecessary intermediate call for `register_anonymous_task`.

If you really want queuing for `check_torrent_health`, it would be better to implement it on the `check_torrent_health` side.

Suggested change

```
103 -         for infohash in random_infohashes:
104 -             self.task_manager.register_anonymous_task("Check preferable torrent",
105 -                                                       self.torrent_checker.check_torrent_health, infohash)
103 +         if not random_infohashes:
104 +             return
105 +
106 +         infohash = random_infohashes.pop()
107 +         await self.torrent_checker.check_torrent_health(infohash)
```

Commit suggestion | Add suggestion to batch

As a side note, if you never use `get_random` with a limit different from 1 (excluding tests), then it is probably a sign that the function should not return a set but a single item. See: YAGNI on Wikipedia.

Reply...

Resolve conversation

drew2a requested a review from **kozlovsky** last week

kozlovsky requested changes yesterday

View reviewed changes

kozlovsky left a comment — Member

I like the idea of this PR! It is crucial for Tribler to understand which torrent users prefer for a specific query because it is tough to properly rank torrents for a particular query without anonymized user feedback.

The system that analyzes user preferences and shares the anonymized aggregated results looks to me like a cornerstone of a future Tribler's search system.

Initially, I thought we needed to gather that information on the UI side, but this PR shows it is possible to do it entirely on the Core side, which brings simplicity.

I mean, it is possible to keep the number of times it was searched for each Query and for each info hash that appeared in the query results to keep the number of times it was chosen or ignored. This information should be enough to calculate the rank and allow for changing the ranking formula in the future without the data being lost.

So what this PR literally does? Right now, it just adds repeated health checks for torrents that the user downloaded. This does not look very useful and can be implemented much more simply; we can have a table with a list of locally downloaded info hashes and randomly check the health of its items. With the current PopularityCommunity protocol, it will not help much, and if this is the end goal of the current PR, then a significant part of it is unnecessary. But I believe that in the future, we should spread not only info hash health info but also should gossip anonymized user preferences for query results (in the form "for query Q, the info hash H with title T was chosen by someone"), and this PR lays down a foundation for this.

src/tribler/core/components/database/db/layers/user_activity_layer.py

Comment on lines +42 to +50

```
42 +         class Query(database.Entity):
43 +             query = orm.PrimaryKey(str)
44 +             infohashes = orm.Set("InfohashPreference")
45 +
46 +         class InfohashPreference(database.Entity):
47 +             infohash = orm.Required(bytes)
48 +             preference = orm.Required(float)
49 +             parent_query = orm.Required(Query)
50 +             orm.PrimaryKey(infohash, parent_query)
```

kozlovsky yesterday — Member

To me, the database structure looks normalized. It is possible to link `InfohashPreference` with the `Resource` entity, but it is actually not necessary and complicates the database schema a bit.

But I'd like to use integer fields instead of floats, like:

```
class Query(database.Entity):
    query = orm.PrimaryKey(str)
    searched_counter = orm.Required(int, default=1)
    infohashes = orm.Set("InfohashPreference")

class InfohashPreference(database.Entity):
    infohash = orm.Required(bytes)
    parent_query = orm.Required(Query)
    chosen_counter = orm.Required(int, default=0)
    ignored_counter = orm.Required(int, default=0)
    orm.PrimaryKey(infohash, parent_query)
```

This way, changing the formula on how `preference` is calculated becomes possible.

src/tribler/core/components/user_activity/user_activity_component.py

```
25 +         super().__init__(reporter)
26 +
27 +         self.infohash_to_queries: dict[InfoHash, list[str]] = defaultdict(list)
28 +         self.queries: OrderedDict[str, typing.Set[InfoHash]] = OrderedDict()
```

kozlovsky yesterday — Member

With the current implementation, the `self.queries` dict is stored in memory. That means (if I understand correctly) that if a user performs the search, starts the download, closes Tribler, and starts it again, the finished torrent will not be matched with the corresponding search query. It looks more correct if, upon torrent completion, we get queries from the database and do not store them in the memory in a separate dictionary. Another approach is to pre-load the dictionary at the Tribler startup, but it probably overcomplicates the code compared to just storing objects in the database.

qstokkink yesterday — Member, Author

Correct. By design, I only consider downloads in the current session.

kozlovsky yesterday — Member

As I just wrote here, I think that we should add the timestamp field to the `InfohashPreference`. Then, we can update this field in the database if the user clicks again at the same infohash.

In that case, the pure-db implementation looks relatively straightforward. When the user clicks on the search result, we add the `InfohashPreference` object with the current timestamp to the database and start the download. When the download is finished, we mark the corresponding `InfohashPreference` as successful. We are gossiping only successfully downloaded `InfohashPreference` objects. Still, the timestamp is recorded at the time when the user clicks on the torrent and not at the time when the download is complete. We update the timestamp if the user clicks again at the same torrent in the new query with the same query string.

qstokkink yesterday — Member, Author

Sure, I do not doubt the feasibility of adding this. My main concern is this:

> I don't think this should be in this first prototype.

We can add features - like you suggested - later. This is not the goal of this initial prototype.

kozlovsky 3 hours ago — Member

Ok, I agree that we can switch to the pure-db implementation later if necessary. Let's hope Tribler restarts will not significantly skew the preference statistics about big torrents.

Reply...

Resolve conversation

`src/tribler/core/components/user_activity/user_activity_component.py` Outdated — Show resolved

`src/tribler/core/components/user_activity/user_activity_component.py` Outdated — Show resolved

qstokkink changed the title ~~**READY: Store preferable infohashes for queries**~~ **WIP: Store preferable infohashes for queries** yesterday

qstokkink marked this pull request as draft yesterday

qstokkink force-pushed the `add_user_activity` branch 4 times, most recently from `09b16aa` to `7186d73` 1 hour ago — Compare

`Store preferable infohashes for queries` ✕ `8049b92`

qstokkink force-pushed the `add_user_activity` branch from `7186d73` to `8049b92` 1 hour ago — Compare

Reviewers

- **kozlovsky**
- **egbertbouman**
- **drew2a**
- **reviewers**

Requested changes must be addressed to merge this pull request.

Assignees

No one—assign yourself

Projects

None yet

Milestone

No milestone

Development

Successfully merging this pull request may close these issues.

None yet

4 participants

✓ Maintainers are allowed to edit this pull request.