Query Analysis

Based on the pt-query-digest report sorted by most time consuming queries, the top 6 queries accounted for **85%** of your total execution time. During the 3 hours sampled using a long query time of 0 (capturing all queries), your system was running **111 queries/sec**. Here is the breakdown of your top 6 queries and I've included the full report for your review as well

```
# 329.8s user time, 290ms system time, 46.25M rss, 113.91M vsz
# Current date: Thu Jun 21 13:34:25 2012
# Hostname: bango
# Files: diagnostics/mysqld.slow.query.log.pt-digest_0s
# Overall: 1.16M total, 545 unique, 111.69 QPS, 0.92x concurrency ________
# Time range: 2012-06-21 10:32:32 to 13:25:55
# Attribute    total     min     max     avg     95%   stddev  median
============ ======= ======= ======= ======= =======  ======= =======
Exec time      9590s     1us     21s     8ms     4ms     79ms    27us
Lock time        96s       0      3s    82us    93us      8ms       0
Rows sent      1.47M       0  19.50k    1.33    1.96    87.22       0
Rows examine   1.41G       0 614.84k   1.27k  329.68   12.74k       0
Query size   222.54M       8  15.34k  200.84  363.48   122.66  223.14

# Profile
# Rank Query ID           Response time   Calls  R/Call Apdx V/M   Item
# ==== ================== =============== ====== ====== ==== ===== =======
#    1 0x966BF1AF355882FB 5228.4144 54.5% 23934  0.2185 1.00 0.14  SELECT zizzle_bp_activity zizzle_users
zizzle_bp_xprofile_data
#    2 0xC7A3C85C9B35EC3B 924.6605 9.6%   1703   0.5430 0.96 0.83  SELECT zizzle_bp_activity zizzle_users
#    3 0x3D78916179C76C82 683.7524 7.1%   215    3.1802 0.48 0.50  SELECT zizzle_bp_activity zizzle_users
#    4 0x215C9A763D786F93 552.2501 5.8%   524    1.0539 0.87 0.96  SELECT zizzle_bp_activity zizzle_users
#    5 0xC601E80703B62833 509.4502 5.3%   1785   0.2854 1.00 0.01  UPDATE zizzle_posts
#    6 0x7009AAC9672C67AE 318.8803 3.3%   1407   0.2266 1.00 0.04  SELECT zizzle_bp_activity
```

Query #1

This was by far your most time consuming query as it accounted for over **54%** of your total execution time. Looking at the query, the first thing that jumps out is high number of rows examined compared to the actual number of rows sent. Here, you can see that in the 95th percentile, you were reading **~60k rows** to return 7 rows:

```
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count          2   23934
# Exec time     54   5228s    24us      3s   218ms   356ms   174ms   241ms
# Lock time     10     10s       0      2s   404us   159us    22ms   131us
# Rows sent      3  47.18k       0      73    2.02    6.98    2.54    0.99
# Rows examine  61 885.38M       0  66.89k  37.88k  59.57k  22.54k  49.01k
# Query size     3   8.31M     361     366  364.26  363.48    1.52  363.48
```

Looking at the explain plan, you can see that the **type** index was chosen even though the **item_id** was much more selective:

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: a
   partitions: NULL
         type: ref
possible_keys: user_id,item_id,type,mptt_left
          key: type
      key_len: 227
          ref: const
         rows: 100280
        Extra: Using where; Using filesort

*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: u
   partitions: NULL
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 8
          ref: turtlejelly_dev.a.user_id
         rows: 1
        Extra: Using where

*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: pd
   partitions: NULL
         type: ref
```

```
possible_keys: field_id,user_id
          key: user_id
      key_len: 8
          ref: turtlejelly_dev.a.user_id
         rows: 2
        Extra: Using where
```

And here are the index statistics in question:

```
mysql> show index from zizzle_bp_activity where Key_name = "item_id" or Key_name = "type"\G
*************************** 1. row ***************************
       Table: zizzle_bp_activity
  Non_unique: 1
    Key_name: item_id
Seq_in_index: 1
 Column_name: item_id
   Collation: A
 Cardinality: 553707
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:

*************************** 2. row ***************************
       Table: zizzle_bp_activity
  Non_unique: 1
    Key_name: type
Seq_in_index: 1
 Column_name: type
   Collation: A
 Cardinality: 21
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
2 rows in set (0.01 sec)
```

The reason MySQL chose the type index over the item_id index was due to the fact that item_id passed was not a string literal:

```
SELECT a.*,
u.user_email,
u.user_nicename,
u.user_login,
u.display_name,
pd.value as user_fullname
FROM zizzle_bp_activity a,
zizzle_users u,
zizzle_bp_xprofile_data pd
WHERE u.ID = a.user_id
AND pd.user_id = a.user_id
AND pd.field_id = 1
AND a.type = 'activity_comment'
AND a.item_id = 715988
AND a.mptt_left BETWEEN 1 AND 10
ORDER BY a.date_recorded ASC
```

By changing item_id to be a string literal, **item_id = '715988'**, you allow MySQL to use the selective index and table 1 of the explain plan now looks like this:

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: a
   partitions: NULL
         type: ref
possible_keys: user_id,item_id,type,mptt_left
          key: item_id
      key_len: 227
          ref: const
         rows: 4
        Extra: Using where; Using filesort
```

You can see that it is now picking the item_id index with great selectivity. On the your production server, you can see the difference in both execution time and handler statistics when switching this field to be a string literal:

```
Current:
4 rows in set (0.26 sec)
mysql> show status like "ha%";
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
```

```
| Handler_commit             | 1     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 0     |
| Handler_read_key           | 12    |
| Handler_read_next          | 58108 |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 4     |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
| Handler_savepoint          | 0     |
| Handler_savepoint_rollback | 0     |
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
String Literal Version:
4 rows in set (0.02 sec)
mysql> show status like "ha%";
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 0     |
| Handler_read_key           | 12    |
| Handler_read_next          | 63    |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 4     |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
| Handler_savepoint          | 0     |
| Handler_savepoint_rollback | 0     |
| Handler_update             | 0     |
| Handler_write              | 0     |
|                            |       |
```

Based on these numbers, you are looking at a **99.9%** reduction in the number of rows examined to execute the same query with a **93%** reduction in execution time. Seeing as this query accounted for over half of your execution time and over 60% of the total rows examined, you should see a huge performance boost to the server by making this change.

Queries 2, 4, and 6

These queries all follow the same breakdown as query #1. Each uses the item_id not as a string literal and as a result, a much less selective index is chosen.

Query #2

Here is the query form:

```
SELECT a.*,
      u.user_email,
      u.user_nicename,
      u.user_login,
      u.display_name
FROM zizzle_bp_activity a
      LEFT JOIN zizzle_users u ON a.user_id = u.ID
WHERE a.component IN ( 'groups' )
      AND a.type IN ( 'new_blog_post','new_forum_topic','activity_update','joined_group' )
      AND a.item_id IN ( 3036 )
      AND a.hide_sitewide = 0
      AND a.type != 'activity_comment'
ORDER BY a.date_recorded DESC
LIMIT 0, 20
```

The issue is in the **a.item_id IN (3036)** clause with the id not being a string literal. As is, the query does an index scan of the date_recorded index (to sort in descending order). By changing that clause to **a.item_id IN ('3036')**, MySQL chooses the item_id index correctly and query execution time drops from **0.72 seconds to 0.02 seconds.**

Query #4

This is very similar to query #2 with the form of:

```
SELECT a.*,
      u.user_email,
      u.user_nicename,
      u.user_login,
      u.display_name
FROM zizzle_bp_activity a
      LEFT JOIN zizzle_users u ON a.user_id = u.ID
```

```
WHERE a.component IN ( 'groups' )
      AND a.item_id IN ( 496 )
      AND a.hide_sitewide = 0
      AND a.type != 'activity_comment'
ORDER BY a.date_recorded DESC
LIMIT 0, 20
```

Similary, the issue is in the **a.item_id IN (496)** clause with the id not being a string literal. As is, the query does an index scan of the date_recorded index (to sort in descending order). By changing that clause to **a.item_id IN ('496')**, MySQL chooses the item_id index correctly and query execution time drops from **0.70 seconds to 0.03 seconds.**

Query #6

While this query is a different form selecting a different field, it follows the same pattern as above. The **secondary_item_id** index is again on a varchar field, but the query is submitted without the string literal:

```
SELECT id
FROM zizzle_bp_activity
WHERE type = 'activity_comment'
      AND secondary_item_id = 720826
```

Like the other queries, as the data type doesn't match, it can't use the selective index and must resort tousing the type index. Changing this to also be a string literal dropped execution time from **0.21 seconds to under 0.00 seconds.**

Queries 3 and 5

These queries share a different issue in that MySQL can't use an index when performing an infix text search:

```
WHERE a.content LIKE '%@nrestakhri<%'
```

In order to make this run faster, I would recommend to restrict this type of search to a prefix search, **WHERE a.content LIKE '@nrestakhri<%'**, or use an external fulltext search indexer like Sphinx.

In this case, however, I would recommend going the external index route. Since you are running your LIKE clause against a text field, you can't add an index to be hit anyway. If this were a varchar field withan index, restricting the search to be a prefix match would help.

Another option if using an external indexer isn't an option, would be to add another indexed field to the table that is the prefix of the content

field and use it for your search. When each record is populated, you can grab the first few characters of the text and populate a field, maybe called content_search, with this values. Again, this will only work for a prefix search. In the above example, your where clause could then become:

```
WHERE a.content_search = '@nrestakhri<'
```

In general, when doing fulltext searches of this nature, it would be most beneficial to implement a full text search engine and then you won't have to change the business logic of your application.