Сценарии

Дата теста 30.01.2024

1. Эндпоинт /user/search 10 пользователей по 10 запросов каждый.
2. Эндпоинт /user/search 100 пользователей по 10 запросов каждый.
3. Эндпоинт /user/search 1000 пользователей по 10 запросов каждый.
4. Эндпоинт /user/search 10 000 пользователей по 10 запросов каждый.
5. Эндпоинт /user/search 200 000 пользователей по 10 запросов каждый.
6. Эндпоинт /user/search 250 000 пользователей по 10 запросов каждый.
7. Эндпоинт /user/search 500 000 пользователей по 10 запросов каждый.

Результаты тестов

Без ORM и индекса сильное падение производительности наблюдается при 10 000 пользователях по 10 запросов каждый. Максимальное время отклика достигает 4800 мс при среднем значении 370 мс и составляет 3250м для персентиля 99.

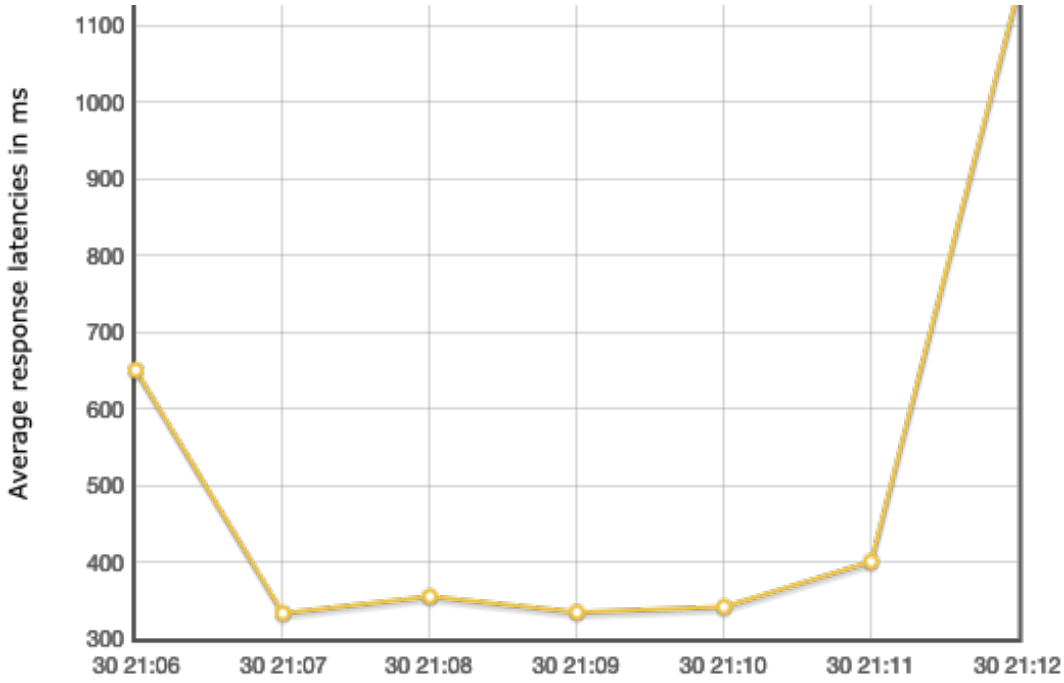

График latency при 10 000 пользователях

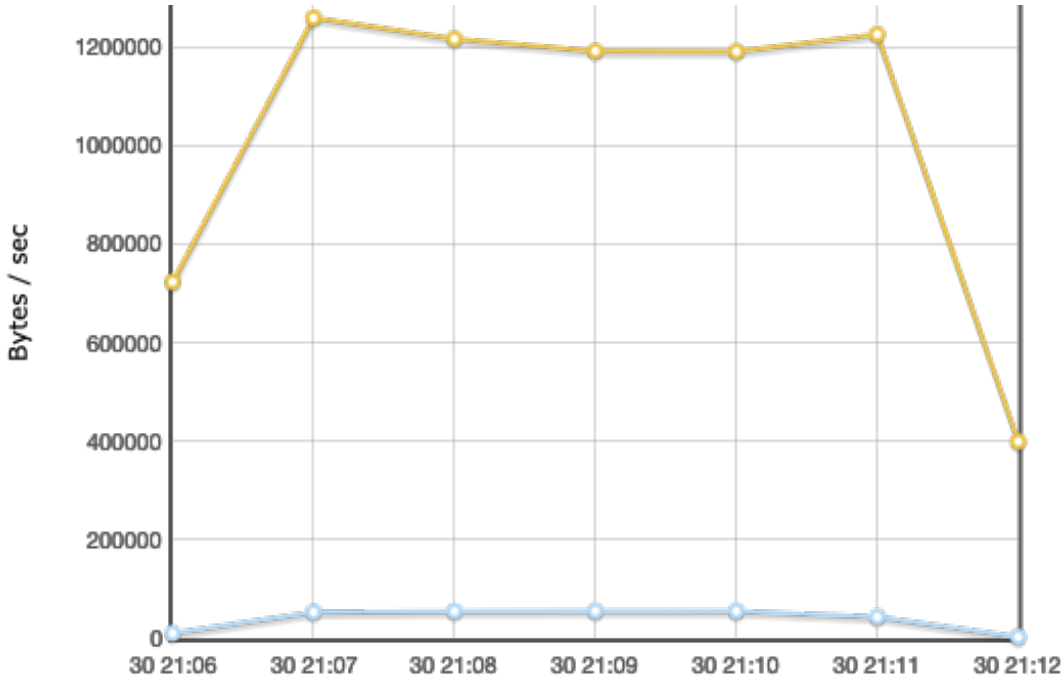

График Throughput при 10 000 пользователях

Для улучшения производительности было решено добавить индекс типа btree для полей first_name и last_name с помощью команды `CREATE INDEX IF NOT EXISTS first_name_last_name_index ON users_tbl(first_name varchar_pattern_ops, last_name varchar_pattern_ops);`
Применение индекса типа btree является оптимальным при данном сценарии использования, так как фактически поиск производится по диапазону значений и позволяет уменьшить среднее время отклика до 10.8 мс, максимальное до 62 мс при 18 мс для 99 персентиля.

```
otus_sn_db=# EXPLAIN ANALYZE SELECT * FROM users_tbl WHERE first_name LIKE 'Ap%' AND last_name LIKE 'Ar%' ORDER BY id;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------
 Gather Merge  (cost=31254.72..31285.75 rows=266 width=159) (actual time=57.457..59.123 rows=164 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Sort  (cost=30254.69..30255.02 rows=133 width=159) (actual time=52.933..52.936 rows=55 loops=3)
         Sort Key: id
         Sort Method: quicksort  Memory: 34kB
         Worker 0:  Sort Method: quicksort  Memory: 39kB
         Worker 1:  Sort Method: quicksort  Memory: 30kB
         ->  Parallel Seq Scan on users_tbl  (cost=0.00..30250.00 rows=133 width=159) (actual time=21.612..52.663 rows=55 loops=3)
               Filter: (((first_name)::text ~~ 'Ap%'::text) AND ((last_name)::text ~~ 'Ar%'::text))
               Rows Removed by Filter: 333279
 Planning Time: 1.382 ms
 Execution Time: 59.294 ms
(13 rows)
```

Explain без индекса

```
otus_sn_db=# EXPLAIN ANALYZE SELECT * FROM users_tbl WHERE first_name LIKE 'Ap%' AND last_name LIKE 'Ar%' ORDER BY id;
                                                                                QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
 Sort  (cost=2123.41..2124.20 rows=313 width=159) (actual time=1.280..1.291 rows=164 loops=1)
   Sort Key: id
   Sort Method: quicksort  Memory: 55kB
   ->  Bitmap Heap Scan on users_tbl  (cost=989.00..2110.44 rows=313 width=159) (actual time=0.627..1.100 rows=164 loops=1)
         Filter: (((first_name)::text ~~ 'Ap%'::text) AND ((last_name)::text ~~ 'Ar%'::text))
         Heap Blocks: exact=66
         ->  Bitmap Index Scan on first_name_last_name_index  (cost=0.00..988.92 rows=306 width=0) (actual time=0.604..0.604 rows=164 loops
=1)
               Index Cond: (((first_name)::text ~>=~ 'Ap'::text) AND ((first_name)::text ~<~ 'Ac'::text) AND ((last_name)::text ~>=~ 'Ar'::
text) AND ((last_name)::text ~<~ 'Ад'::text))
 Planning Time: 0.851 ms
 Execution Time: 1.357 ms
(10 rows)
```

Explain с индексом btree на полях first_name и last_name

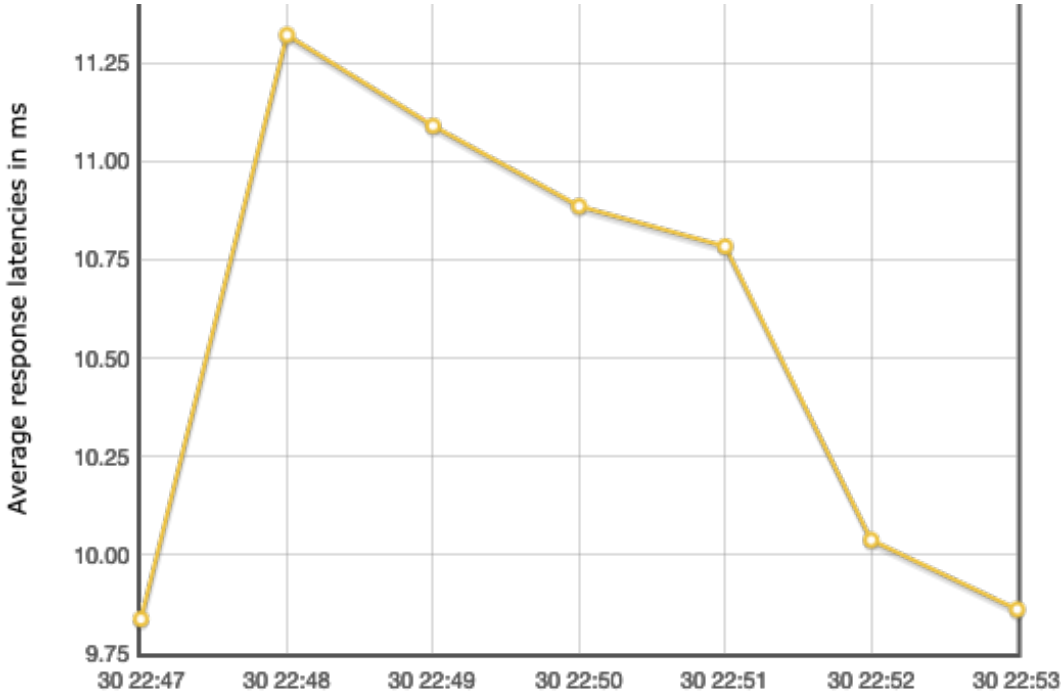

График latency при 10 000 с индексом

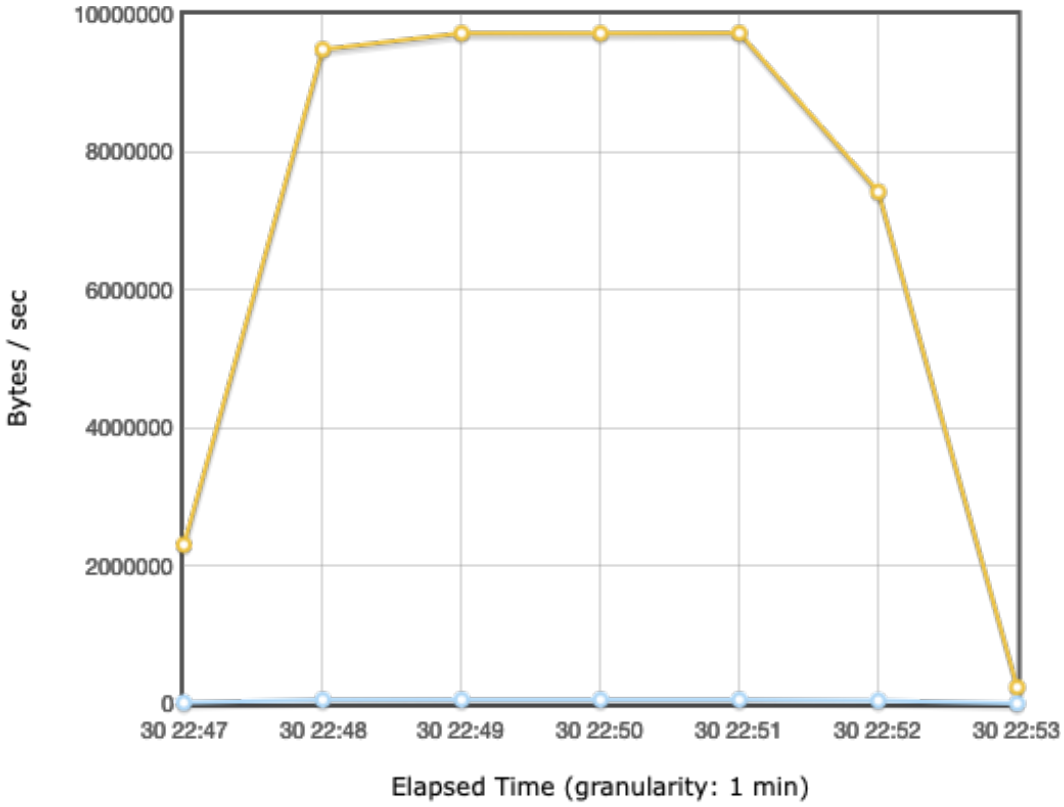

График Throughput при 10 000 с индексом

Максимальной производительности удалось добиться при совместном использовании ORM и индекса. Среднее время отклика составило 2.6 мс, максимальное 43 мс и 6 мс для персентиля 99.

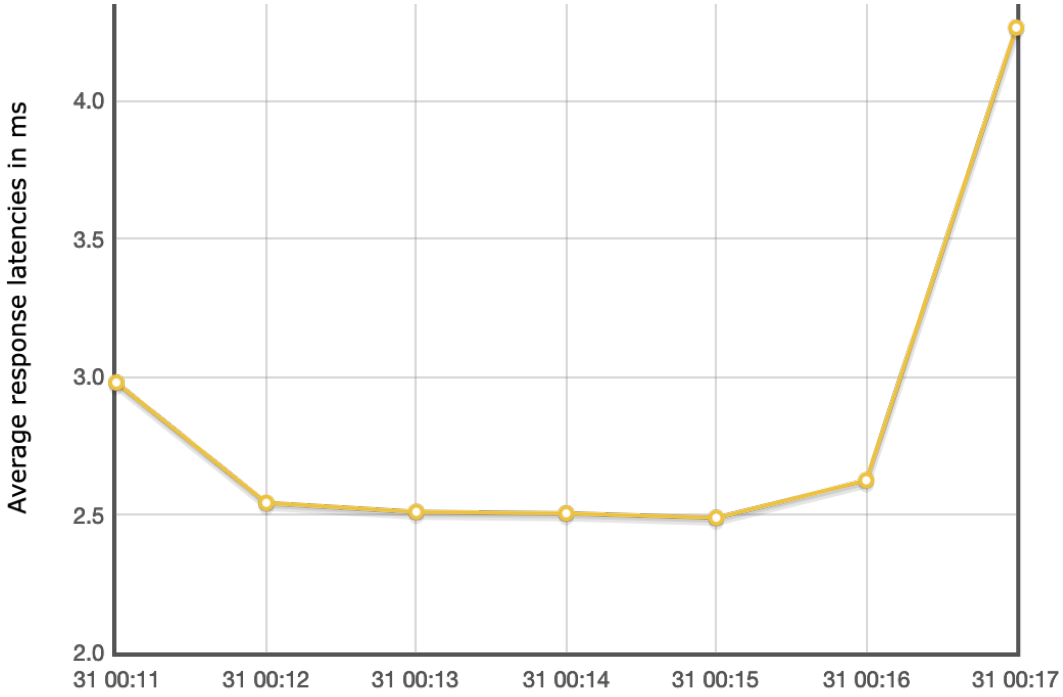

График latency при 10 000 пользователях с индексом и ORM

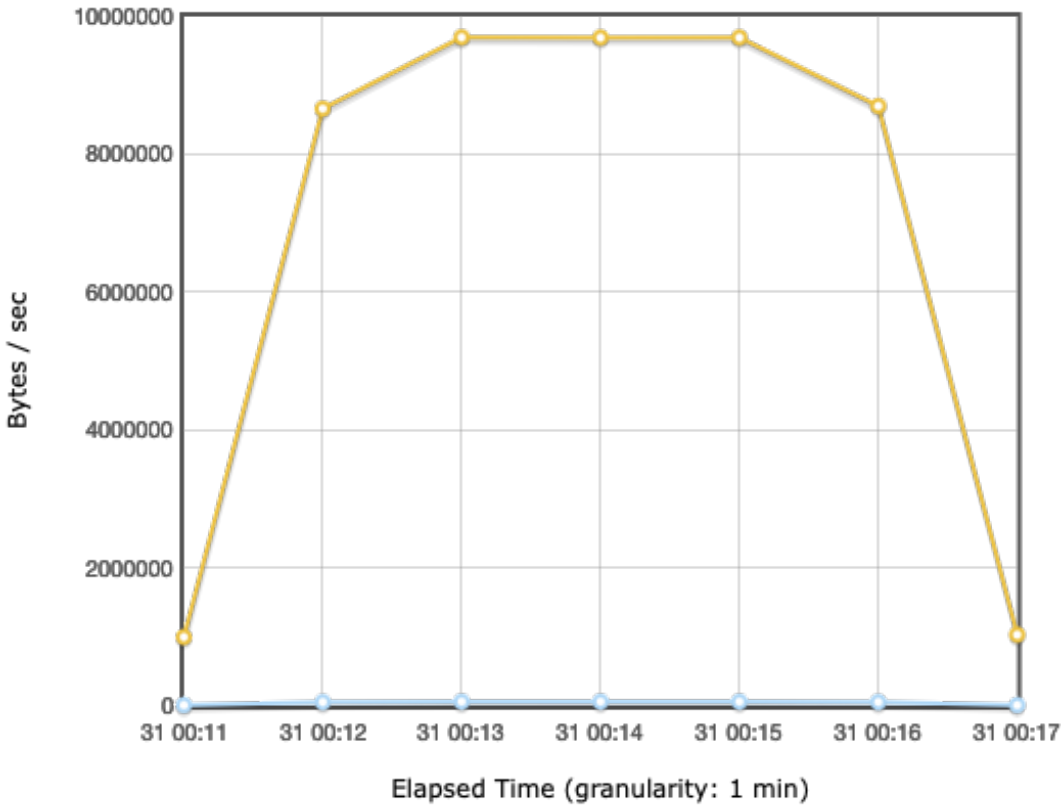

График Throughput при 10 000 пользователях с индексом и ORN

Влияние ORM на производительность

При отсутствии индекса использование ORM приводит с существенному падению производительности. Всего при 10 000 пользователей, среднее время ответа достигает боле чем 100 000 мс, 33% запросов завершаются ошибкой. Однако, как видно из предыдущих графиков, при задействовании индекса ситуациия кардинально меняется. Использование индекса совместно с ORM позволяет добиться максимальной производительности. Среднее время отклика уменьшилось в 4 а

максимальное в 1.5 раза. При оптимальной загрузке CPU на уровне 80% количество пользователей может достигать 200 000.

Заключение

Во всех тестах, производительность ограничивалась CPU. Оптимальной производительности удалось добиться при одновременном использовании ORM и задействовании индекса. При 200 000 пользователях средняя загрузка CPU составила примерно 80% Среднее время отклика составило 4.4 мс, максимальное 236 мс и 4.9 для персентиля 99. Увеличение пользователей до 250 000 тысяч привело к резкому снижению производительности. Среднее время отклика составило 1120 мс, максимальное больше 20 тысяч. Загрузка CPU приблизилась к 100%

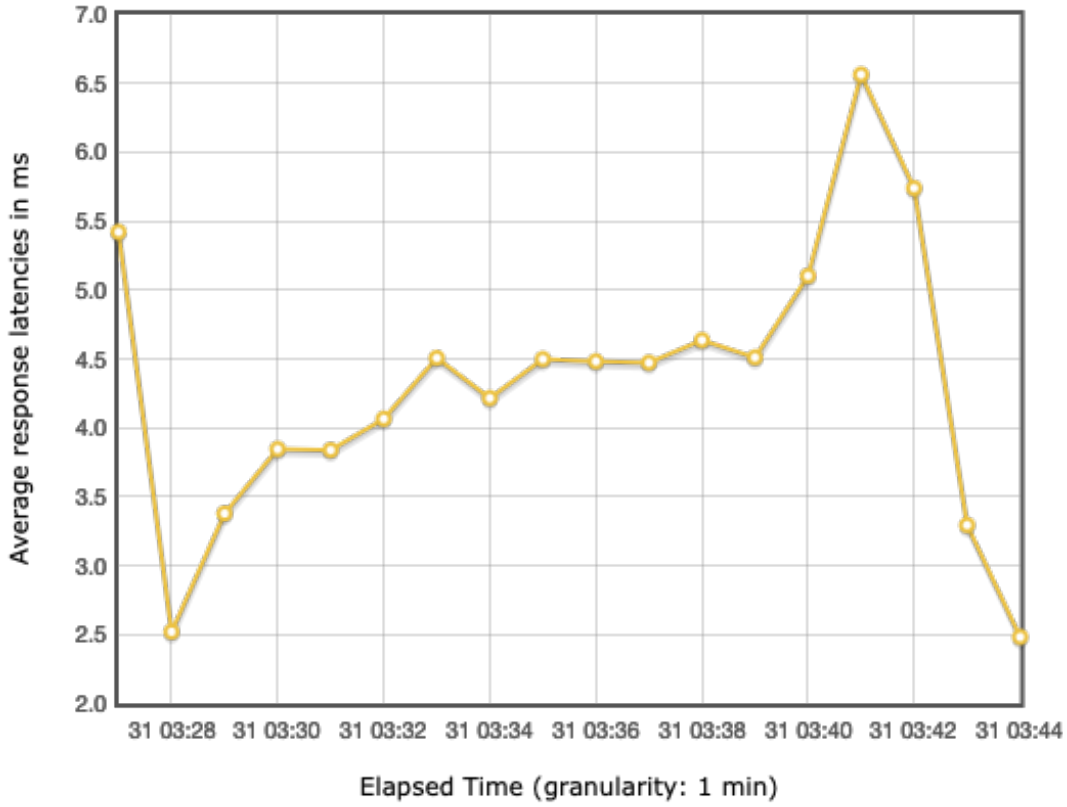

График latency при 200 00 пользователей с индексом и ORM

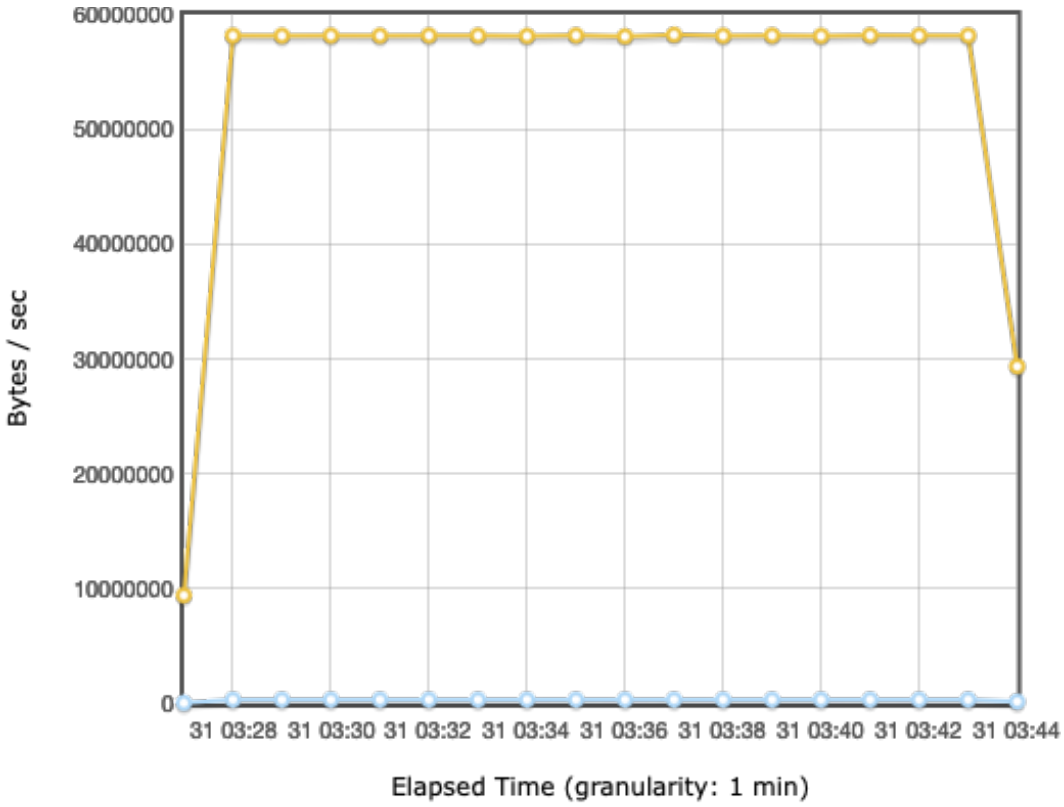

График Throughput при 200 000 пользователей с индексом и ORM

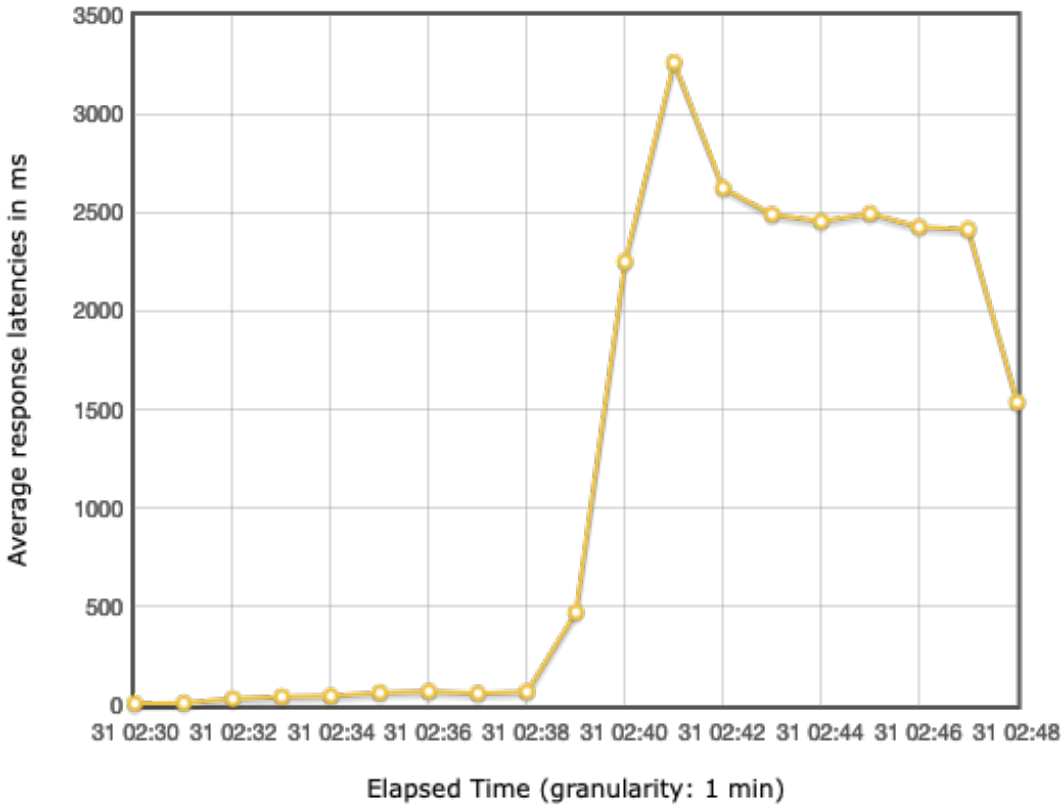

График latency при 250 000 пользователях с индексом и ORM

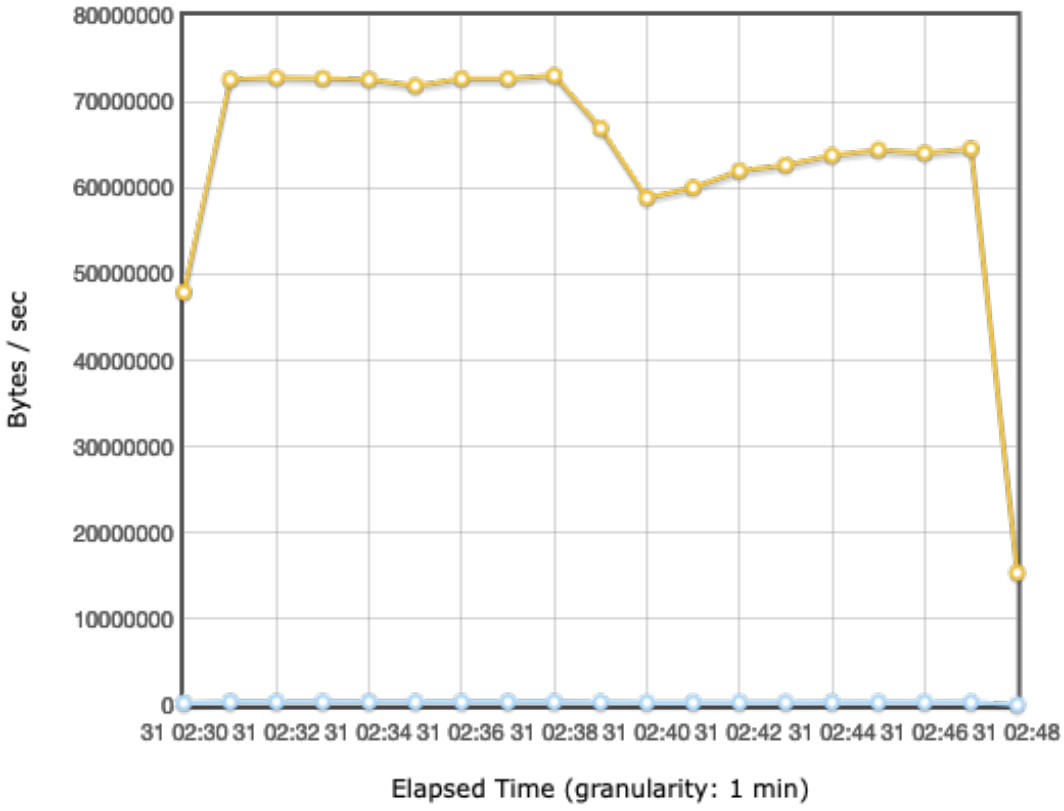

График Throughput при 250 000 пользователях с индексом и ORM