Scaling Backend API

(Reducing Response Times)

Software Engineering
Technical Presentation Milestone

Mingxuan(Matthew) Du

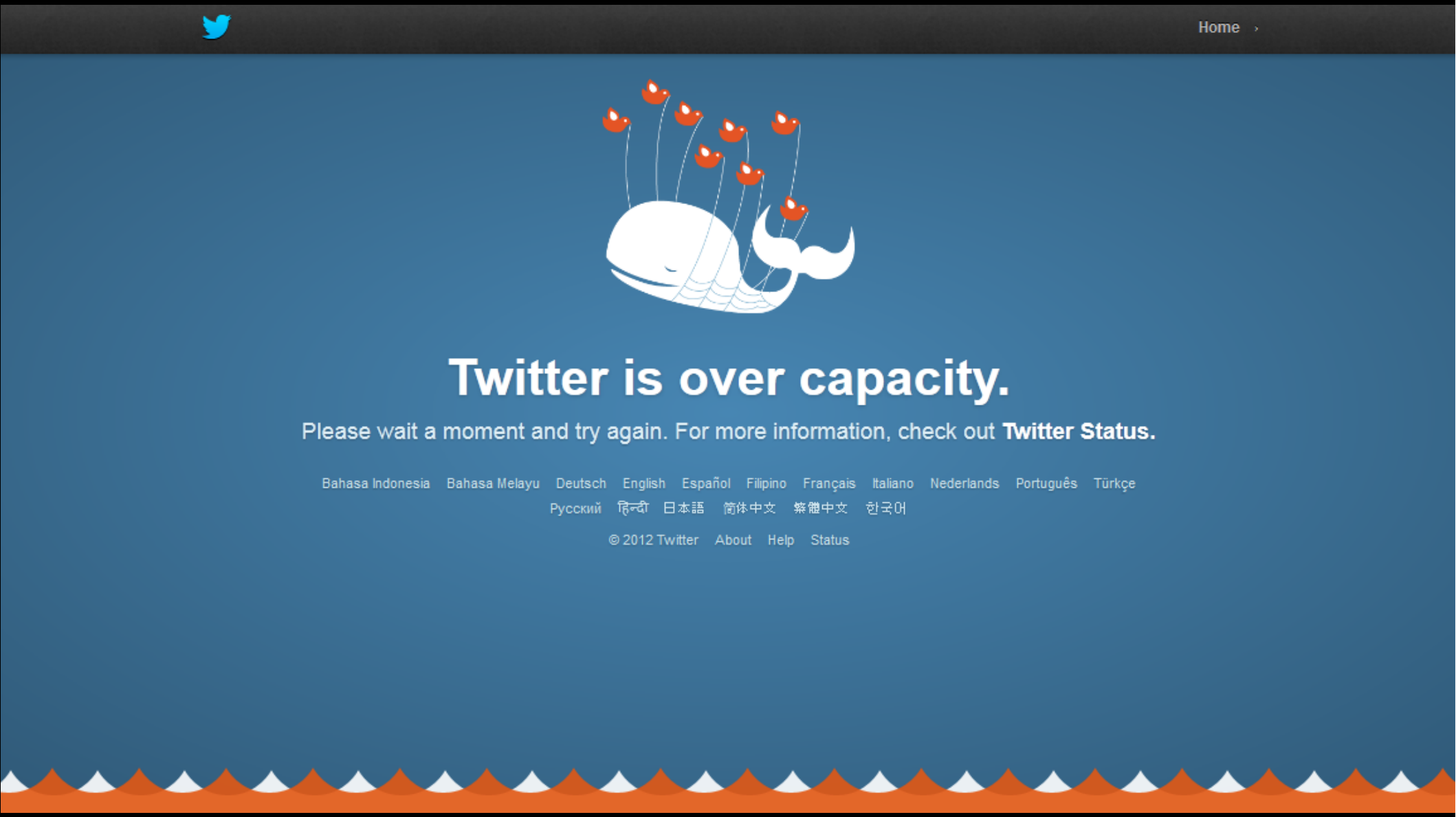
June 2, 2015

Overview

- What is scaling? Why scale?
- Optimizing queries
- Caching
- Queuing

https://www.thesleuthjournal.com/wp-content/uploads/2015/01/empty-store-shelves1.jpg

https://leagueofmentalmen.files.wordpress.com/2014/11/xs02.jpg



http://www.tsupdate.com/wp-content/uploads/2013/01/t.png

Optimizing Queries

- Make your database queries fast
- Do what you can to make external queries fast
- Benchmark

Optimize Database Queries

Suppose you wanted to figure out how many of your customers are under 21

```
users = User.all
under_21 = 0
users.each do |i|
  if i.age < 21
    under_21 += 1
  end
end
```

```
users = User.where("age < ?", 21).count
```

Indexing

- Indices speed up searching

```
class AddIndex < ActiveRecord::Migration
  def change
    add_index :users, :email
  end
end
```

- Add indices whenever you have “WHERE” statements

You belong to multiple organizations. Each of which have multiple events. And each event has multiple customers. Find all unique customers with in the last month.

```php
public function getNewUsers() {

  $input = Input::all();

      $rules = [
          'event_id' => 'required|integer'
      ];

      $validator = Validator::make($input, $rules);

  $user = Auth::user();

  if(!$user) {
    return Response::json([
          "error_message" => "user authentication failed"
        ], 401);
  }

      if($validator->passes()) {

        $event_id = Input::get('event_id');
    $event = EventListing::find($event_id);

    $total_number_of_users_within_range = DB::select("SELECT COUNT(DISTINCT user_id) as count FROM purchases WHERE canceled != 1 AND event_listing_id = " . $event_id)[0]->count;

    $user_organization_ids = $user->organizations()->select('organizations.id')->lists('id');

    // $user_organization_ids = implode(",", $user_organization_ids);

    $user_event_ids = DB::table('event_listings')
              ->select('event_listings.id')
              ->join('organizations', 'event_listings.organization_id', '=', 'organizations.id')
              ->whereIn('organizations.id', $user_organization_ids)
              ->lists('event_listings.id');
            // ->where(DB::raw('SELECT organization_id IN (SELECT organization_id FROM user_organization_role WHERE user_id = ' . Auth::user()->id . ')' ))->lists('event_listings.id');

    $user_event_ids = implode(",", $user_event_ids);

    $number_of_users_that_intersect = DB::select(DB::raw("SELECT COUNT(*) as count FROM (SELECT DISTINCT user_id FROM purchases WHERE canceled != 1 AND created_at BETWEEN '" . $event->created_at . "' AND '" .
                    JOIN (SELECT DISTINCT user_id from purchases WHERE canceled != 1 AND DATE(created_at) < (SELECT DATE(created_at) FROM event_listings WHERE event_listings.id = " . $event_id . ")) AS p2
                    ON p1.user_id = p2.user_id"))[0]->count;

    $number_of_recurring_users = DB::select(DB::raw("SELECT COUNT(*) as count FROM (SELECT DISTINCT user_id FROM purchases WHERE canceled != 1 AND event_listing_id = " . $event_id . "  ) AS p1
                    JOIN (SELECT DISTINCT user_id from purchases WHERE canceled != 1 AND DATE(created_at) < (SELECT DATE(created_at) FROM event_listings WHERE event_listings.id = " . $event_id . ") AND event
                    ON p1.user_id = p2.user_id"))[0]->count;

    // Count the number of unclaimed purchases as well, these people are for sure new users
    $number_of_new_users_from_unclaimed_purchases = UnclaimedPurchase::where('event_listing_id', $event_id)
                                  ->where('canceled', '!=', 1)
                                  ->count();


    $total_number_of_users_within_range += $number_of_new_users_from_unclaimed_purchases;
    $total_new_customers = $total_number_of_users_within_range - $number_of_users_that_intersect;

    return Response::json([
      'total_new_customers' => $total_new_customers,
      "total_recurring_customers" => $number_of_recurring_users
    ], 200);

  } else {
    return Response::json([
        'error_message' => 'validation failed!'
      ], 400);
  }
}
```

```ruby
def new_customers(first_date = Time.zone.now.to_date - 30, last_date = Time.zone.now.to_date)
  query = %Q[
    SELECT COUNT(DISTINCT(p1.user_id)) as new_users
    FROM (
      SELECT user_id
      FROM purchases
      WHERE ticket_id IN (
        SELECT id
        FROM tickets
        WHERE event_listing_id IN (
          SELECT id
          FROM event_listings
          WHERE organization_id IN (
            SELECT organization_id
            FROM user_organization_role
            WHERE user_id=#{@user.id}
          )
        )
      )
      AND canceled=0
      AND DATE(created_at) BETWEEN '#{first_date}' AND '#{last_date}'
    ) p1
    LEFT JOIN (
      SELECT user_id
      FROM purchases
      WHERE ticket_id IN (
        SELECT id
        FROM tickets
        WHERE event_listing_id IN (
          SELECT id
          FROM event_listings
          WHERE organization_id IN (
            SELECT organization_id
            FROM user_organization_role
            WHERE user_id=#{@user.id}
          )
        )
      )
      AND canceled=0
      AND DATE(created_at) < '#{first_date}'
    ) p2
    ON p1.user_id = p2.user_id
    WHERE p2.user_id IS NULL;
  ]
  result = ActiveRecord::Base.connection.execute(query, 'SQL').to_a.first.first.to_i
end
```

Eager Loading

Solve the n+1 queries problem.

```
user = User.find(1)
user.organizations.each do |organization|
  puts organization.address.street
end

user = User.includes(:organizations).find(1)
user.organizations.each do |organization|
  puts organization.address.street
end
```

Messages Timeline Exceptions Views 4 Route Queries 1450 Mails Request

1450 statements were executed, 103 of which were duplicated

```
select * from `users` where `id` = '77' and `remember_token` = 'vALDx8OMqrbEMPRronw8pfhi4hA9RiUPNe5E1esORx976PEOdIRfbIptkDNl' limit 1
select * from `organizations` where `organizations`.`id` = '92' limit 1
select * from `groups` where `groups`.`id` = '3' limit 1
select * from `organizations` where `organizations`.`id` = '92' limit 1
select * from `event_listings` where `event_listings`.`deleted_at` is null and `event_listings`.`id` = '93' limit 1
select * from `event_listings` where `event_listings`.`deleted_at` is null and `event_listings`.`id` = '93' limit 1
select * from `event_settings` where `event_settings`.`event_id` = '93' limit 1
select * from `organizations` where `organizations`.`id` = '28' limit 1
select `users`.*, `user_organization_role`.`organization_id` as `pivot_organization_id`, `user_organization_role`.`user_id` as `pivot_u
as `pivot_created_at`, `user_organization_role`.`updated_at` as `pivot_updated_at` from `users` inner join `user_organization_role` on
'28'
select * from `tickets` where `tickets`.`deleted_at` is null and `tickets`.`event_listing_id` = '93'
select * from `purchases` where `purchases`.`ticket_id` = '133' and `canceled` != '1'
select * from `purchase_addons` where `purchase_addons`.`purchase_id` = '2707' and `canceled` != '1' and `canceled` != '1'
select * from `users` where `users`.`id` = '2420' limit 1
select * from `orders` where `orders`.`id` = '3918' limit 1
select * from `purchase_addons` where `purchase_addons`.`purchase_id` = '3620' and `canceled` != '1' and `canceled` != '1'
```

Messages Timeline Exceptions Views 4 Route Queries 25 Mails Request

25 statements were executed, 3 of which were duplicated

```
select * from `users` where `id` = '77' and `remember_token` = 'vALDx8OMqrbEMPRronw8pfhi4hA9RiUPNe5E1esORx976PEOdIRfbIptkDNl' limit 1
select * from `organizations` where `organizations`.`id` = '92' limit 1
select * from `groups` where `groups`.`id` = '3' limit 1
select * from `organizations` where `organizations`.`id` = '92' limit 1
select * from `event_listings` where `event_listings`.`deleted_at` is null and `event_listings`.`id` = '93' limit 1
select * from `event_listings` where `event_listings`.`deleted_at` is null and `event_listings`.`id` = '93' limit 1
select * from `event_settings` where `event_settings`.`event_id` = '93' limit 1
select * from `organizations` where `organizations`.`id` = '28' limit 1
select `users`.*, `user_organization_role`.`organization_id` as `pivot_organization_id`, `user_organization_role`.`user_id` as `pivot_u
as `pivot_created_at`, `user_organization_role`.`updated_at` as `pivot_updated_at` from `users` inner join `user_organization_role` on
'28'
select * from `tickets` where `tickets`.`deleted_at` is null and `tickets`.`event_listing_id` = '93'
select * from `purchases` where `purchases`.`ticket_id` = '133' and `canceled` != '1' and `canceled` = '0'
select * from `users` where `users`.`id` in ('2420', '1481', '3105', '3107', '3112', '2015', '3115', '3116', '3118', '3120', '3122', '3
'3196', '3199', '3216', '3218', '3219', '3225', '3229', '3232', '3234', '3248', '3253', '3255', '3256', '3257', '208', '2582', '3271',
'3430', '3442', '3468', '3471', '3476', '3480', '3487', '3499', '3505', '3506', '3507', '2977', '3561', '3562', '3565', '3566', '3567',
'3602', '3603', '3608', '3616', '3617', '3630', '3631', '3632', '3633', '3634', '3635', '3638', '3637', '3640', '3641', '3642', '3643',
'3656', '3657', '3658', '3659', '3660', '3661', '3662', '3663', '3664', '3665', '3667', '3668', '3669', '3670', '3671', '3672', '3687',
'3737', '3738', '3739', '3740', '3744', '3745', '3746',
```

javascript:

Optimize External Queries

- External services provide additional features, ex Google Analytics
- They are very slow compared to internal services.
- Each request has an HTTP overhead
- Combine requests to reduce overhead.

Results showing: **7** | Total results found: **7** | Contains sampled data: **No** | ↓ Jump to bottom

Date	Pageviews	Unique Pageviews
1 Feb 2015	11	5
2 Feb 2015	540	86
3 Feb 2015	305	105
4 Feb 2015	570	156
5 Feb 2015	676	105
6 Feb 2015	465	103
7 Feb 2015	134	77

Caching

- Don't do the same thing twice
- When users make requests, check cache before doing anything
- Save computation time

What you need to start caching

- A cache store
- Some conditionals to choose between using cache or computing
- Way to warm/populate cache

Cache Stores

Redis

- More datatypes
- Persistence to disk

Memcached

- Small memory usage comparatively

http://manuel.manuelles.nl/images/posts/redis.png
http://no-cache.appdynamics-static.com/appsphere/logos/memcache_350.png

```ruby
def revenue
  Rails.cache.fetch("#{event_id}/revenue", :expires_in => 1.day) do
    query = %Q[
      SELECT SUM(amount_in_cents) as revenue_in_cents
      FROM (
        SELECT amount_in_cents
        FROM purchases
        WHERE ticket_id IN (
          SELECT id FROM tickets
          WHERE event_listing_id = #{@event.id}
        )
        AND canceled != 1

        UNION ALL

        SELECT amount_in_cents
        FROM unclaimed_purchases
        WHERE ticket_id IN (
          SELECT id FROM tickets
          WHERE event_listing_id = #{@event.id}
        )
        AND canceled != 1
      ) as amount_in_cents_union
    ]
    ActiveRecord::Base.connection.exec_query(query).rows.first.first
  end
end
```

```ruby
def pageviews_by_date
  Rails.cache.fetch("#{event_id}/pageviews_by_date", :expires_in => 1.day) do
    results = Analytics::Google::Client.execute(
      :parameters => {
        'start-date' => @first_date.to_s,
        'end-date'   => @last_date.to_s,
        'dimensions' => 'ga:date',
        'metrics'    => 'ga:pageviews,ga:uniquePageViews',
        'filters'    => "ga:pagePath=~^/event/#{@event.id}(\\D|$);ga:pagePath!~^/event/#{@event.id}/checkout/"
      }
    ).data.rows

    results.map { |row| [Date.parse(row[0]), row[1].to_i, row[2].to_i] }
  end
end
```

```ruby
Rails.cache.fetch("analytics/users/#{@user.id}/report", :expires_in => 1.day) do
  {
    :revenue => self.revenue,
    :revenue_by_date => self.revenue_by_date,
    :revenue_by_date_cumulative => self.revenue_by_date_cumulative,
    :tickets_sold => self.tickets_sold,
    :tickets_sold_by_date => self.tickets_sold_by_date,
    :tickets_sold_by_date_cumulative => self.tickets_sold_by_date_cumulative,
    :new_customers => self.new_customers
  }
end
```

How to populate cache

- The default implementation. If the cache key does not exist, populate the cache
- Pre-populate the cache with ALL relevant information
- Update cache when related records are updated
- Write some complex machine learning algorithm to decide when to populate/refresh cache

A simple 'parallel' method for warming cache

```
namespace :analytics do
  desc "cache user analytics report"
  task :cache_users => :environment do
    org_ids_with_events = Organization.where(:id => Event.all.pluck(:organization_id)).pluck(:id)
    user_ids = UserOrganizationRole.where(:organization_id => org_ids_with_events).uniq.pluck(:user_id)
    user_ids.shuffle!
    user_ids.each do |user_id|
      Analytics::User.new(user_id).report
    end
  end
end
```

- Each running server would run this task
- The ids that need caching is shuffled
- Each server would cache different ids

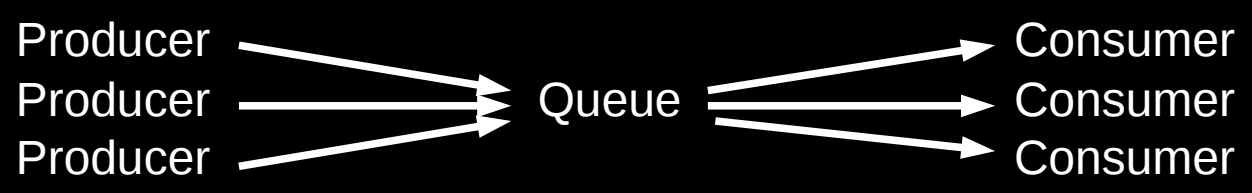
Caching Expiration/Eviction

“There are only two hard things in CS: cache invalidation and naming things.”

- Phil Karlton

Queues

- Long running process shouldn't clog the user interface
- Queue tasks into a queue
- Process them later



When to use queues?

- Emails
- Image processing/Video encoding
- Communication between servers
- Updating cache

```ruby
def sign_up
  @user = User.new(user_params)
  if @user.save
    SendReminderEmail.perform_later(@user.id)
    flash[:success] = "#{@user.name} created."
    redirect_to users_path
  else
    render 'new'
  end
end
```

Implementing a Queue

- Using Redis
- Using a database table
- Using a third-party service like Amazon SQS, or IronMQ

Summary

- Optimize your backend queries. Try what you can to optimize third-party queries.
- Cache. Cache. CACHE!
- Queue long running tasks.

Conclusion