Love the Functionality, Hate the Speed, Now What?
My WordPress Experience

• Built Pagely’s Hosting Stack
• Worked on our devops team
• Solved many performance problems
I Built a wordpress site

Hello Wordcamp

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Its performance didn’t suck
Then I made the site a bit cooler
Its performance isn’t so great now.
Now What?
Full Page Caching to the Rescue
Maybe Not

- Low cache hit rate
- Traffic from bots
- Logged in requests
- Full cache purge
Red Flag

Page rendering time over 2 seconds
Let's do some math

- Server has 2 cores
- Each core can render 4 WordPress pages at once
- Each page takes 2 seconds to complete

\[
\text{RPS Max} = \frac{2 \times 4}{2} = 4
\]
Now What?
Let's find what's slow
Slow is normally IO

Slow Database query
Slow http request

It's sometimes doing tons of fast things
It's less commonly doing something slow in PHP land
Revert changes one at a time to pinpoint a slow plugin.
1. APM like New Relic
2. Query Monitor, etc
3. P3 profiler
4. PHP profilers, Xdebug
Tools help pinpoint a slow line of code
Everything you can see
Digging in, found the slow
Now What?
Let's fix it.

If it's not your code

FORK IT.
Plugins hosted in github makes it simple

At a minimum rename the folder, update comments

Goal is to get your fixes upstream
CAUTION

Running patched plugins without renaming is a good way to have auto updates ruin your day
Fixing a slow API call

You can’t make the API faster, so limit its effects

Options:
• Caching around the API call
• Use a work queue
Caching using transients

Transients are the generic WordPress caching API
https://codex.wordpress.org/Transients_API

Stores data in object cache if available or wp_options
public function render()
{
    if (is_singular('post'))
    {
        return "<div class='ContentDesirability'>".$this->callApi(get_permalink())."</div>";
    }
    return "";
}

public function callApi($permalink)
{
    $score = wp_remote_get('http://cd.pagely.com?url='.urlencode($permalink));
    return $score['body'];
public function render()
{
    if (is_singular('post'))
    {
        return "<div class='ContentDesirability'>".$this->callApiCached(get_permalink())."</div>";
    }
    return "";
}

public function callApiCached($permalink)
{
    // transient keys have a max length of 172 chars, so md5 to get a constant length
    $key = 'Content_Desirability-'.md5($permalink);
    $content = get_transient($key);
    if ($content === false)
    {
        $content = $this->callApi($permalink);
        set_transient($key, $content);
    }
    return $content;
}
Things to think about

Default to `set_transient` is no expiration

Often we are getting timely data

Set expirations as long as possible

Dynamic expirations are often the solution
public function callApiCached($permalink)
{
    // transient keys have a max length of 172 chars, so md5 to get a constant length
    $key = 'Content_Desirability-'.md5($permalink);
    $content = get_transient($key);
    if ($content === false)
    {
        $content = $this->callApi($permalink);
        $expiration_seconds = 4 * 60 * 60;
        // after a week set expiration to 24 hours
        if (get_the_date('U') < (time()-86400*7))
            $expiration_seconds = 24 * 60 * 60;
        set_transient($key, $content, $expiration_seconds);
    }

    return $content;
}
Problem Solved?

Only if we can get a high enough cache hit rate
Long tail content
Bots crawling the site
What if the api gets slower
Moving the API call out

1. If you have the data, display it
2. If you don’t queue a request to load it, Make API request outside of the normal flow
3. Update the data in transients
Some projects to look at

WP Minions: https://github.com/10up/WP-Minions

WP Background Processing: https://github.com/A5hleyRich/wp-background-processing
Let's build a basic queue

Store jobs in an option

Process them using wp-cron
Limitations

Keep the max queue size relatively small

Will have a low max processing rate

Will be prone to race conditions
add_filter('cron_schedules', array($this, 'cron_schedules'));
if (!wp_next_scheduled('content_desirability_cron'))
    wp_schedule_event( time(), 'five_minutes', 'content_desirability_cron' );
}

public function cron_schedules($schedules)
{
    $schedules['five_minutes'] = [
        'interval' => 300,
        'display' => esc_html__( 'Every Five Minutes' ),
    ];
    return $schedules;
}

function content_desirability_cron()
{
    $plugin = new Content_Desirability();
    $plugin->apiCron();
}
public function render()
{
    if (is_singular('post'))
    {
        $score = $this->callApiCachedQueued(get_permalink());
        if ($score)
            return "<div class='ContentDesirability'>$score</div>";
    }
    return "";
}
public function callApiCachedQueued($permalink)
{
    $key = 'Content_Desirability-'.md5($permalink);
    $content = get_transient($key);
    if ($content === false) {
        $queue = get_option('cd_queue');
        if (!$queue)
            $queue = [];
        $queue[$key] = $permalink;
        if (count($queue) > 100)
            $queue = array_slice($queue, 0, 100);
        update_option('cd_queue', $queue, false);
    }
    return $content;
}
public function apiCron()
{
    $start = microtime(true);
    $queue = get_transient('cd_queue');
    if (!$queue)
        return;

    $expiration_seconds = 24 * 60 * 60;
    foreach($queue as $key => $permalink)
    {
        $content = $this->callApi($permalink);
        set_transient($key, $content, $expiration_seconds);
        $this->removeFromQueue($key);

        // after 10 seconds stop
        if ((microtime(true)-$start > 10)
            break;
    
}
public function removeFromQueue($key)
{
    // reduce races, by dropping the in-memory object cache
    // most likely won't be compatible with drop-in object-cache.php (memcache, redis etc)
    global $wp_object_cache;
    $wp_object_cache->cache = []; 

    $queue = get_option('cd_queue');
    unset($queue[$key]);
    update_option('cd_queue', $queue, false);
}
Queue Results

- 10 users: Concurrent Users
- 525 ms: Response Time
- 720 rpm: Transaction Rate
- 24Mb/s: Network Throughput
- 204 ms: Latency

Graph showing the relationship between users and response time, transaction rate, and network throughput over time.
Problem Solved?

As long as our queue can keep up

High rate sites will need to use something like WP Minions
The fixes we tried

- Added a full page cache
- Patched the plugin
- Caching around API calls
- Moving API calls to a work queue
What I normally do

Use full page caching and Transient Caching

For really high traffic sites use WP Minions
Thanks 😊