

Caching Best Practices with Pitney Bowes Business Insight Tile Servers

WHITEPAPER:

LOCATION INTELLIGENCE

Alex Lee • Product Evangelist, Pitney Bowes Business Insight
Craig Goettsche • Product Architect, Pitney Bowes Business Insight
Tom Citriniti • Product Evangelist, Pitney Bowes Business Insight



Caching Best Practices with Pitney Bowes Business Insight Tile Servers

2

ABSTRACT

THE POSITIVE RESULTS AND BENEFITS OF CASHING IN ON CACHING BEST PRACTICES ARE FELT THROUGHOUT THE ENTERPRISE.

THIS PITNEY BOWES BUSINESS INSIGHT WHITE PAPER, THE FIRST IN A TWO-PART SERIES, HAS A THREE-FOLD MISSION.

FIRST, DISTRIBUTED APPLICATIONS, CONSISTING OF LARGE USER BASES THAT DEPEND ON REQUESTS FOR INFORMATION OR STATIC PAGES, MUST EFFECTIVELY REDUCE THE WORK NEEDED TO SERVICE REQUESTS. CACHING THE RESULTS FROM LONG RUNNING REQUESTS WILL INCREASE RESPONSE TIME AND REDUCE USER WAIT TIME. THIS WHITE PAPER PRESENTS THE BENEFITS OF USING A CACHING ARCHITECTURE IN A MULTI-TIER ENTERPRISE APPLICATION, AND DESCRIBES HOW IT CAN BE EFFECTIVELY INTEGRATED WITH A PITNEY BOWES BUSINESS INSIGHT TILE SERVER.

SECOND, WEB CACHING IS A MECHANISM USED IN VARIOUS WAYS AND IN MANY DIFFERENT SCENARIOS. THIS WHITE PAPER DESCRIBES HOW TO INTEGRATE PITNEY BOWES BUSINESS INSIGHT PRODUCTS INTO VARIOUS WEB CACHING MECHANISMS USING STANDARD HTTP CACHE HEADERS AND FOLLOWING THE STANDARDS SET FORTH BY THE W3C.

FINALLY, PITNEY BOWES BUSINESS INSIGHT MAP BASED APPLICATIONS WERE TRADITIONALLY DESIGNED TO RENDER MAPS FOR EACH AND EVERY REQUEST. THIS PRACTICE IS VERY TAXING ON SERVERS AND, WHEN COMBINED WITH BUSINESS LOGIC, LEADS TO SEVERE PERFORMANCE CHALLENGES. THIS WHITE PAPER PROPOSES A WAY TO CACHE CERTAIN ASPECTS OF THE MAPPING APPLICATION AND COMBINE IT WITH OTHER MAP REQUESTS. THIS FUNCTIONALITY ALLOWS THE CACHING OF STATIC PARTS OF THE MAP, WHILE RENDERING DYNAMIC PIECES ONLY WHEN NECESSARY. THIS DIVISION OF LABOR LENDS ITSELF WELL TO HIGHLY SCALABLE SYSTEMS THAT CAN TRULY SCALE ACROSS THE ENTERPRISE.

Caching Basics

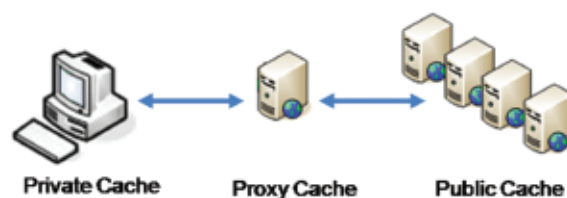
Web caching is the process of storing a copy of a web request in a cache for quick retrieval at a later time. Web caches are deployed in various ways and implement different methodologies. There are three basic types of web caching—private caches, proxy caches, and gateway caches. The implementation of one, two, or a combination of all three web caches, in most cases, reduces server load, lessens bandwidth requirements, and eliminates the perceived lag in page load times. Each cache store can save copies of requests from the originating site. Subsequent requests may be satisfied from the cache, provided the correct conditions are met.

Private caches are used only by the local user on the client machine. Web browsers utilize this private cache to store local copies of requests and, when allowed, use this local copy instead of making requests back to the server.

Proxy caches are deployed locally to cache frequently visited sites. A company or school, for example, may employ a proxy cache to locally store frequent sites that a group of employees or students visit as a way to reduce bandwidth for their entire site. These caches can be explicitly configured by the users, or can be used without user awareness.

Gateway caches are used at a provider's site and act as public caches allowing access to all users. Gateway caches transparently act as accelerators or in-line caching systems by storing copies of requests on disk or in memory. These caches respond with the stored copy when allowed, or pass the request on to the server when needed. The client is not aware of these caches and no configuration on the client machine is necessary.

Combining the three different types of caches, based on site and user needs, provide many options and possible configurations. Choosing the correct settings is critical to obtaining the ultimate in performance and reliability.



Tile server applications make the most of these cache stores. These applications assist in improving performance and gaining control of cached information by enforcing dynamic rules of tiles for each user.

Consider the scenario in which an application has two tile layers: one base tile reference map layer and one overlay theme layer. The base layer is the same for all users, while the overlay layer is specific to an individual user based on their data preferences. In this example, the base layer is set to cache at all stores (public, proxy, and private), while the dynamic layer may only be cached at the private store. This scenario provides individual clients with a private cache of their specific dynamic content while preventing other users from accessing those tiles. All base tiles stored in a public or proxy cache are shared among all users.

HTTP Headers

HTTP headers are information tags sent in both HTTP requests and as additional information in server responses. The cache-control header tag is the mechanism used to control where content is stored with respect to a specific cache store. These headers contain tags that specify the content type, date created, server, and other web related information. Since these headers are sent separately from the content, adding them to the application's HTML head section does not carry over to proxies or client browsers. Therefore, it is recommended that the request handler APIs of the server hosting the request be used to control these tags.

Caching Best Practices with Pitney Bowes Business Insight Tile Servers

4

Web content can be cached without sending requests back to the server by specifying the Expires tag. This allows content to be cached without requiring network traffic back to the server. Expiring cache content is used for application data that is updated on a regular basis.

This information is taken from the RFC 2616 Section 13.2.1, which states:

HTTP caching works best when caches can entirely avoid making requests to the origin server. The primary mechanism for avoiding requests is for an origin server to provide an explicit expiration time in the future, indicating that a response MAY be used to satisfy subsequent requests. In other words, a cache can return a fresh response without first contacting the server.

For instance to indicate an Expires tag for cached web content, it is recommended that the absolute date and time be used as defined by the HTTP specification RFC 1123. For example:

Expires: Thu, 01 Dec 1994 16:00:00 GMT

This specifies that a cache can return the requested content from the cache until the listed date without contacting the original server.

There are many rules regarding how the cached content is handled. For more information on HTTP caching tags, review RFC 2616 Section 14.21 of the HTTP 1.1 specifications at <http://tools.ietf.org/html/rfc2616>.

In addition, multiple web sites are available that describe the details of using HTTP headers to control cache settings. The following is a list of three sites providing helpful information.

A simple start:

http://www.mnot.net/cache_docs/

Dedicated to cache:

<http://www.web-caching.com/>

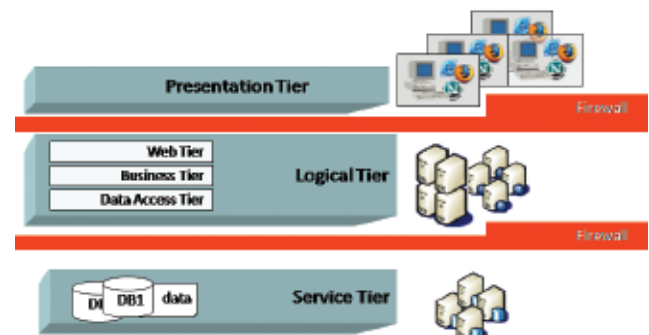
The W3C RFC:

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html>

Web Tier Caching

When using an N-Tier web architecture, it is best to use a caching methodology that exists at the web tier. The web tier is the first logical site where all requests are handled and forwarded on to other subsystems. When the requests are cached at this tier, any request made of the server can be cached, regardless of the backend system from which it is retrieved. For instance, some requests are handled at a middle tier application server, while stateless requests are handled by a services tier. Either scenario can have the appropriate HTTP headers to control where the item is cached.

The key to using caching efficiently is to set the appropriate HTTP cache header to identify if and where the request is cached. Setting the cache headers incorrectly leads to unexpected results or behavior. For instance, if the content is dynamic for each client, then caching at the web tier, or public cache, causes requests to be cached and allows all clients to share the same content. When dealing with private or sensitive data, making the content available to all users is certainly not the desired outcome.



A methodology is needed that specifies the location of the cached content by the type of required response. Either the content is dynamic for each user, or it is static and can be shared by all users. Each request must be defined in these terms and the appropriate header applied.

Tile Server Caching

Tile servers are popular for web mapping because they allow the application to pre-render parts of the map and store them as images. These image tiles are requested by the client and pieced together at the user's computer. Tiling has the following two major advantages over rendering each individual request:

- Tile server based applications utilize the resources of the client computer to organize and compose the tiles, making tiling a truly scalable option
- Client requests require the same information, allowing a public pre-rendered cache to serve all user tile requests without taxing backend servers

Proxy Caches

A proxy cache is used to create an intermediate location that allows caching outside of the web application. Proxy caches store requests that are passed through the caches, and follows the caching rules set in the HTTP header. Proxy caches are not hosting applications, they only accept requests from clients and either return the cached result if it exists, or pass the request on to the application server.

A good example is the Squid web proxy cache (<http://www.squid-cache.org/>). Squid is used as a proxy cache at sites to forward requests to any tile server and cache the resultant image tile. This allows the separation of the rendering of tiles from the process of caching the results. These results can then be spread over multiple servers to distribute the requests for tiles to multiple machines and drastically increase scalability.

Our Pitney Bowes Business Insight testing and preliminary work using Squid added a reverse proxy to the Squid configuration file specific to our site, as well as a refresh pattern to identify the content returned from the tile handler. The details are included as part of the performance document for caching.

These settings are specific to the Squid proxy cache and are different for each caching system. To optimize response times and take full advantage of the enterprise environment, users must thoroughly understand the concepts of caching and take the necessary time to plan and configure cache subsystems.

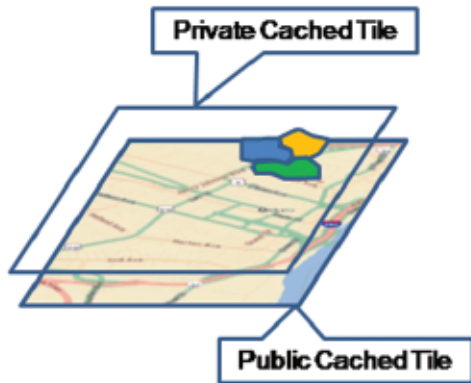
Pitney Bowes Business Insight offers a performance document that explains both how to configure our products and how to configure basic cache settings. This document also shows our results of direct tile rendering versus cached tile responses.

Caching in Use

As mentioned previously, the cache settings are the key to a well behaved and predictable application. The rules governing the cache are important in defining how the response to a request is serviced. This is both a problem and a benefit. If not set correctly, items to be cached may not be cached, and items not to be cached may, in fact, be cached. A web mapping application that uses tiles to serve up the base map and a user specified overlay should use both cached and non-cached tiles in the same map.

In this scenario, the same base map tiles are shared by all users, since all users of the application desire the same underlying map. The overlay tiles differ from user to user, and are based on the user choices in the application. These choices may include the creation of a theme layer or other user defined criteria.

Caching Best Practices with Pitney Bowes Business Insight Tile Servers



The base map tiles contain an Expires header element that defines when the content becomes stale. This header defines how long a response can be cached. Once this time expires, the request must go back to the originating server to generate a new tile. This type of marked tile can be cached at any level along the request pipeline.

The overlay tiles are based on user interaction with the application, and are reflective of the specific user's requests. These tiles cannot be cached at the public or proxy level as they would not necessarily reflect the user's current session state. When another user asks for the same tile, they receive a tile based on their particular information and session state. To control this, the response headers contain a tag specifying that the request can only be cached at the private cache. This guarantees that the user only has access to their individual cached tiles.

An example of this is an application that creates a thematic map based on a query of user-specific customers from a table of all customers. This subset is different for each user, as they are only interested in their own customers, not all customers in the table. There are two choices for the map generation. First, the application creates a single map of

all layers including the individual user's data. Second, the application takes a tiled approach and separates the public tiles from the individual tiles.

A tiled approach requires the creation of a tile handler that handles an individual user's state. This tile handler is only concerned with the individual tiles that are based on the customer data for each user. Additionally, the application requires a method of dividing the base map, the map that is constant for all users, from the user specific map. To accomplish this, a tile handler is constructed for the base map tiles, and a second tile handler serves the user specific map.

The base map handler simply serves the requested tile, while making sure the appropriate HTTP headers are in place to enable caching at all levels. Since all users require the same background, these tiles can be safely cached in public, proxy, and private stores.

The custom tile handler for the overlay map must have the ability to store and apply the user's current state. When a tile is requested, the returned tile properly reflects the correct subset of the customer table. When a request is made to this handler, the user's state is applied to the map, and the tile is rendered with a transparent background. The tile is then returned to the client where it is overlaid on the base map. The HTTP cache header needs to specify that this tile may only be cached in the private cache of each user.

Using the two tile handler approach, the application has the ability to now properly cache public tiles that all users will share, while caching the private tiles that are specific to each user. This scheme maximizes the caching benefits of all tiers while reducing the amount of work done at the server and distributing the load of map organization to the client system.

The following are the four key points inherent to a tile handler approach.

- Most tiling frameworks allow the application to overlay point and vector data on the client side. While this may work for simple point sets, overlaying a large number of points or vector objects using JavaScript severely degrades the client performance. Creating a tile overlay to handle large amounts of map data is a better option.
- Client side programming APIs, using JavaScript, have a simple rendering model. To achieve better cartographic quality, overlay tiles must be generated.
- HTTP headers controlling the cache must be set correctly to ensure proper caching and expiration dates.
- Requests for the dynamic overlay tiles are sent back to the handler for every tile request if the user does not have a private or local browser cache.

A tile server that not only serves requests but also allows the specification of how to cache the response is the proper course of action. This allows the use of any level cache to achieve the required behavior while allowing the freedom to serve static requests or dynamic tiles based on an individual user's state.

Conclusion

A caching strategy greatly improves performance for a distributed application, and a well planned out scheme is a key part of application architecture. Adding a caching scheme to an application increases resource requirements to host the application. These activities must be well planned, taking into consideration all requirements and proper metrics. The result of this planning is a well designed architecture that seamlessly integrates a cache mechanism while increasing site performance. These systems also reduce the load on backend servers and increase the ability to serve customers efficiently and effectively.

The next Pitney Bowes Business Insight white paper, the second in the two-part series, focuses on the performance of our tile servers and details best practices in application sizing and planning.



UNITED STATES

One Global View
Troy, NY 12180
main: 518.285.6000
1.800.327.8627
fax: 518.285.6070
pbbi.sales@pb.com
www.pbinsight.com

CANADA

26 Wellington Street East
Suite 500
Toronto, Ontario
M5E 1S2
main: 416.594.5200
fax: 416.594.5201
pbbi.canada.sales@pb.com
www.pbinsight.ca