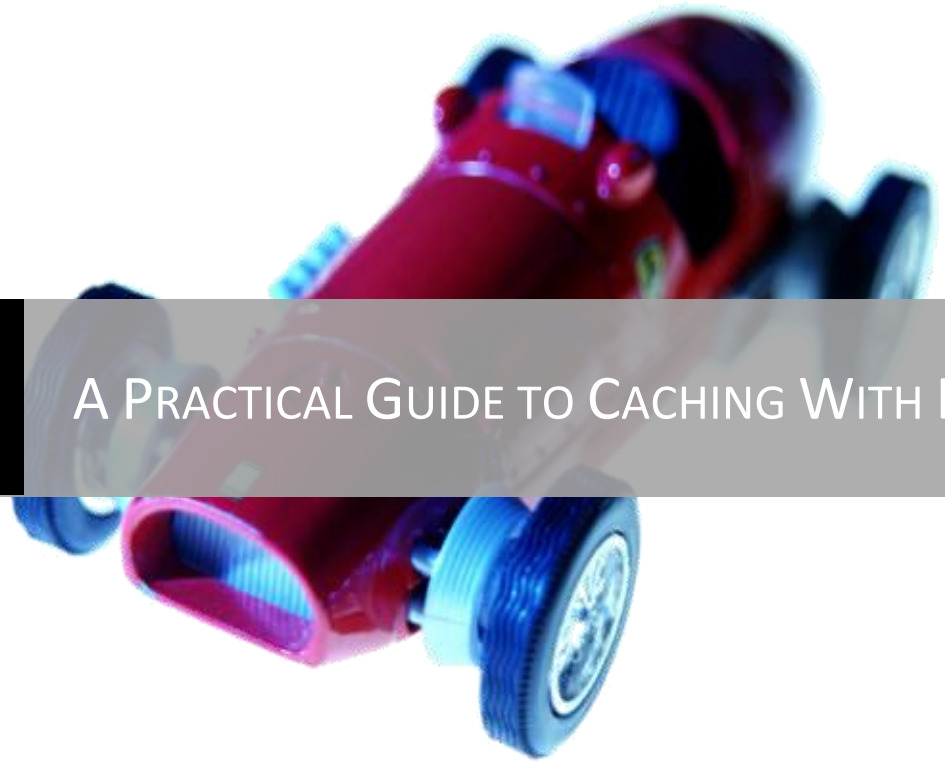
open kod
Get it Done!



OPEN KOD SDN BHD

A Practical Guide to Caching With Durio

Contents

Introduction

Cache as per computer science is defined as a component that transparently stores data so that future requests for that data can be served faster. The data that is stored within a cache might be values that have been computed earlier or duplicates of original values that are stored elsewhere. If requested data is contained in the cache (**cache hit**), this request can be served by simply reading the cache, which is comparatively faster. Otherwise (**cache miss**), the data has to be recomputed or fetched from its original storage location, which is comparatively slower. Hence, the greater the number of requests that can be served from the cache, the faster the overall system performance becomes.

Let's take a look at this figure for better understanding.

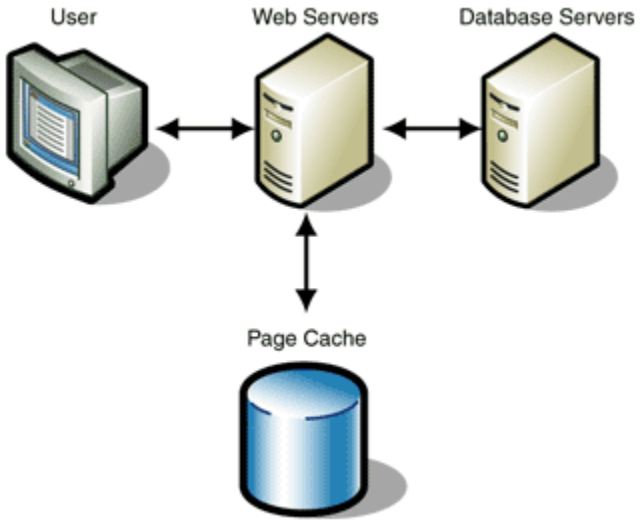


Figure 1: Basic cache configuration

Durio can be categorized and used for caching in these five areas:

1. Web Cache
2. File Transfer Protocol Cache
3. Post Office Protocol 3 Cache
4. Simple Mail Transfer Protocol Cache
5. Domain Name System Cache

For this particular time, we will discuss mainly on web cache and some of the other features.

How Do I Get Started?

I am working with a web-based application that presents dynamic information for users. I have observed that many users access a specific page without the dynamic information changing. How can I improve the response time of dynamically generated web pages that are requested frequently but consume a large amount of system resources to construct?

Solution

One obvious approach to making systems faster is to buy more hardware. This option may be appealing because hardware is cheap (or so the vendors say) and the program does not have to change. On the other hand, more hardware only helps until you reach its physical limitations. Network limitations, such as data transfer rates, or latency make these physical limitations more apparent.

A second approach to making systems faster is to do less (processing) work. This approach requires more effort from developers, but can provide enormous increases in performance.

Then came the Durio solution, but first let's take a look at this diagram.

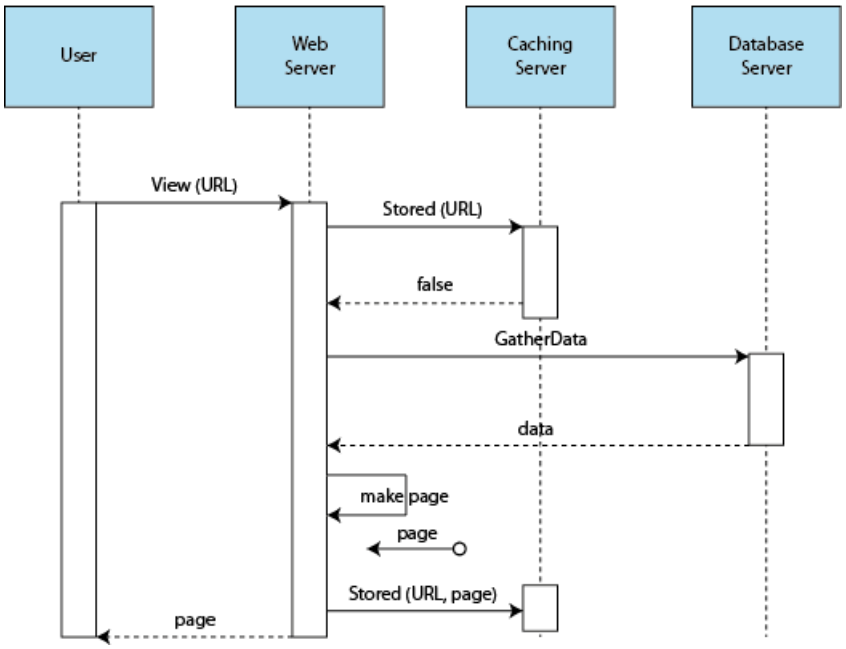


Figure 2: Sequence for a cache miss (when the page is not in the cache)

As shown in the above diagram, this is a normal connection type that involves a user requesting a web page (or application in this matter) from the internet. But this will only works if you already have a caching server (web cache) installed and working. Consequently, a working caching server will only cache a static information or data that less or unchanged at all.

But what is static data?

A range of data that's prepares the system to the daily use. It can be a background, buttons, fields, colors, texts and anything that don't change much within the web or application.

Now, let's look at this diagram.

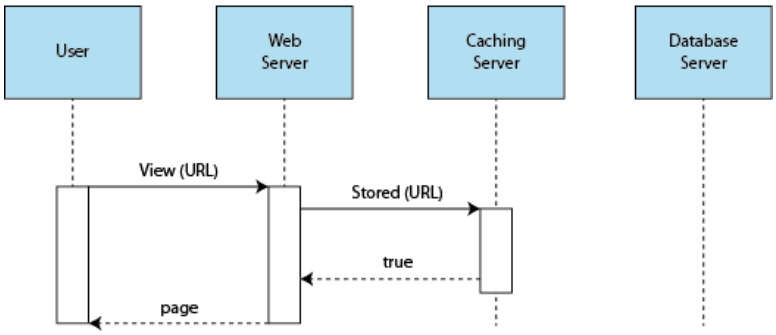


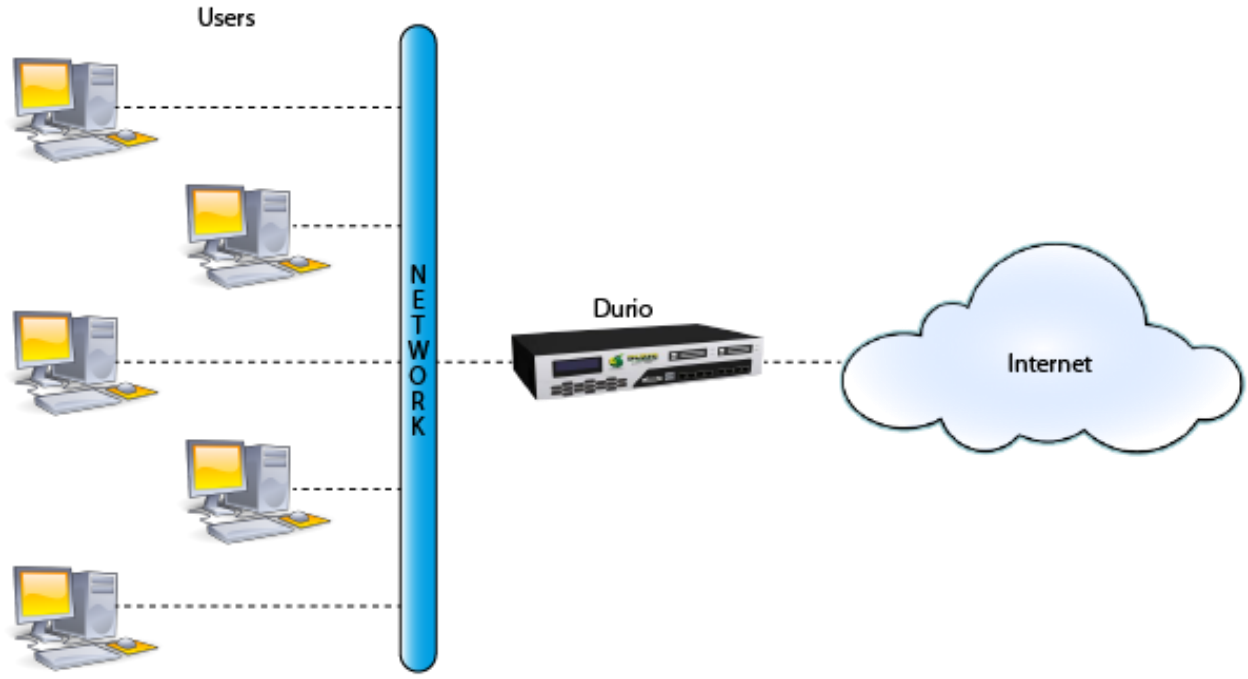
Figure 3: Sequence for a cache hit (when the page is in the cache)

In the cache hit scenario shown in Figure 3, the page is already in the cache. A cache hit saves cycles by skipping the database access, the page rendering, and the storing of the page.

That's mean you can save up to 60 – 95% bandwidth consumption thus the need to add more hardware is terminate.

How Do I Decide What To Cache?

That is simple. Durio is built with the latest intelligent agent to justify every data whether it's static or dynamic and automatically cache every data by the first time user request for it. Saying that you have 1,000 users and concurrently all of them are accessing the same data.



Durio will not only cache the data but subsequently filter all the incoming traffic from the internet to protect and safeguard your users. Apart from saving you from expanding your budget to buy more

hardware, Durio can also be used as an antivirus, anti-spam (should you have a mail server) and that's sound like saving you more money.

What Do I Cache?

The smartest approach to caching would be to look for the slowest spots in your network environment, and the best way to do that would be using Durio monitoring and logging tools. You can use those to detect the real bottlenecks in your network, and then proceed to eradicate them by applying the caching techniques.

Another point you should keep in mind is that a 10% speedup on a page which accounts for 80% of your traffic (such as home page on some application) is usually way more effective than a 70% speedup on a page which accounts for 2% of your traffic. You should always start with the most popular application in your collection.

To reach the most compromise between speed and size, you must be careful about which application you cache. Some application will be accessed much more frequently than others. So, ideally, you should cache only the popular application and forget about the rarely used ones. This decision is not always easy to make, because usage patterns tend to vary. Many cache implement strategies such as Least Frequently Used (LFU) to remove application that have been used frequently since being stored. Other caching schemes let the user specify the caching strategy for each individual application.

The next most important decision is how big the pieces in the cache should be. Storing complete application enables quick page display after a page hit, because the system retrieves the page from the cache and immediately sends it to the user without any other action. However, if some portions of the application change frequently and others do not (for example, a page with weather and stock prices), storing complete application could lead to a lot of extra storage. Storing smaller pieces improves the chances of a page hit, but also required more storage overhead (there are more pieces to index) and more CPU consumption.

How Long Do I Need To Cache For?

How long the system keeps items in the cache is also important. Storing pages for a fixed amount of time is the simplest methods. This method may not always be sufficient, however.

We'll be using weather for an example, if an unusual weather pattern such as a hurricane is approaching a major city, you may want to update every 15 minutes. You can resolve these issues by tying the caching duration to external events. You may choose to flush the cache when an external event arrives, forcing the page to be re-rendered when the next request arrives.

Some caching strategies try to re-render pages during low-traffic periods. This approach can be very effective if you have predictable traffic patterns and you can hold pages long enough to avoid refreshing during peak traffic times.

Durio with its intelligent agent actually plan for your storage. At first when we configure it, we allow the utilization of 100% of storage space for storing data. Saying that you have a 1TB of storage and 2% of it was used by the system. That's mean the whole 98% storage space can be occupy to be caching storage. Durio automatically calculate your remaining storage space and fully utilize it to be caching storage. Take a look at this diagram.

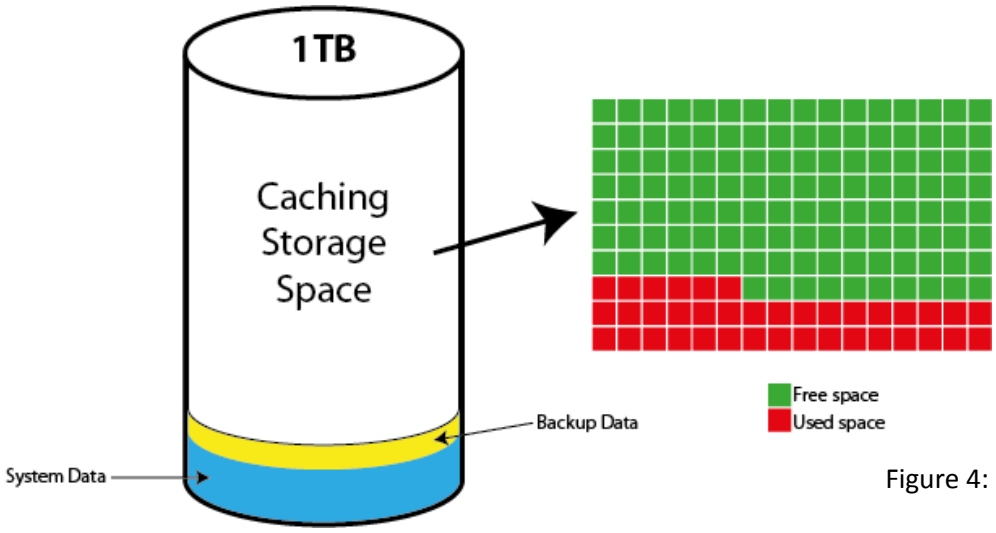


Figure 4: Durio Intelligent Agent

Data stored in the storage will be analyzed every 5 seconds to check if the need to re-render the data occurs. Durio Intelligent Agent will arrange the data automatically by checking whether the cache hit is dynamically change or static. It also will use the remaining free space till it's full.

What happen when it full?

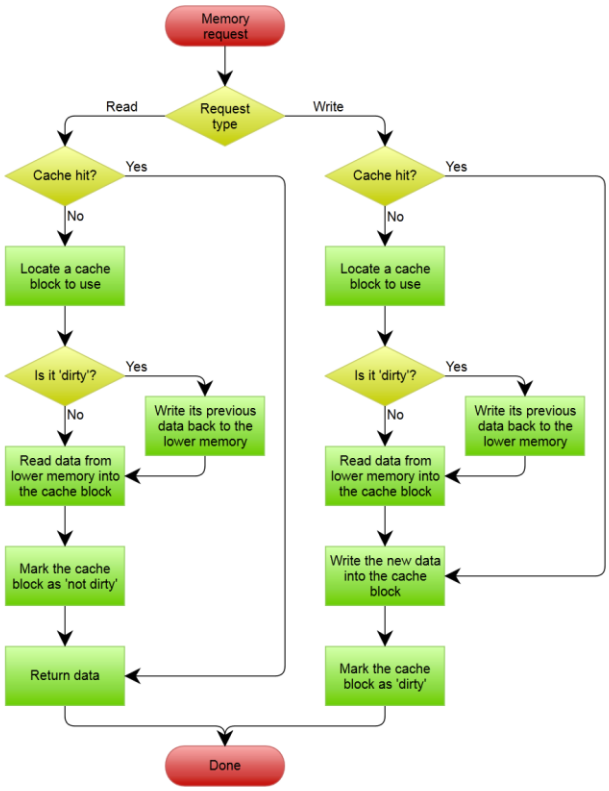
The Least Frequently Used data will be flushed out automatically and replaced with a new one. Ideally a new storage space is only needed when the capability to process is maxed out by the user. An increasing number of users need more storage space.

The Durio Solution!

Durio allows organizations to save on their bandwidth through content caching. Cached content means data is served locally and users will see this through faster download speeds with frequently-used content. Thus avoiding spending large amounts of money on upgrading core equipment and transit links to cope with ever-demanding content growth. It also allows organizations to prioritize and control certain web content types where dictated by technical or economic reasons.

Durio optimizes the data flow between client and server to improve performance and caches frequently-used content to save bandwidth. Durio can also route content requests to servers in a wide variety of ways to build cache server hierarchies which optimize network throughput.

Durio can reduce server load and improve delivery speed to anyone. Durio can also be used to deliver content from around the world – copying only the content being used, rather than inefficiently copying everything. Durio's advanced content routing configuration allows administrators to build content clusters to route and load balance requests via a variety of web servers.



The infinity time period for data stored in Durio is also a winning statement for organization who wishes to expand their budget on other IT product rather than to buy more equipment meant only for storage.

Summary

Caching results in the following benefits and liabilities:

1. **Eliminates unnecessary round-trips to the database or other external data sources.** This benefit is particularly important, because these external sources usually provide only a limited number of concurrent connections that must be shared by all concurrent page requests in a resource pool. Frequent access to external data sources can quickly bring a web server to an abrupt halt due to resource contention.

2. **Conserves user connections.** Each concurrent connection from a user to the web server consumes limited resources. The longer it takes to process a page request, the longer the connection resource is tied up.

3. **Enables concurrent access by many page requests.** Because a page cache is primarily a read-only resource, it can be multithread rather easily. Therefore, it prevents resource contention that can occur when the system accesses external data sources. The only portion that must be synchronized is the cache update, so the considerations around frequency of update are most critical to good performance.

4. **Increases the availability of the application.** If the system accesses an external data source to render pages, it depends on the data source being available. Page caching enables the system to deliver cached pages to the users even when the external source becomes unavailable; the data may not be current, but it is likely better than no data at all.

5. **Display information that is not current.** If the cache refresh mechanism in configured incorrectly, the web site could display invalid data, which could be confusing or even harmful.

6. **Requires additional security considerations.** This implication of caching is often overlooked. When a web server is processing concurrent requests for confidential information from multiple users, it is important to avoid crossover between these requests. Because the page cache is a global entity, an improperly configured page cache may deliver a page to the browser that was originally rendered for another user.

7. **Can produce dramatically inconsistent response times.** Although delivering pages quickly in 99 percent of the cases is surely better than delivering slow pages every time, a caching strategy that is over-optimized for cache hits and under-optimized for cache misses can cause sporadic timeouts. This concern is particularly relevant for web services as opposed to simple HTML pages.

Combining a good, quality and local product with its intelligent agent is another way of saying ‘bon voyage’ to your caching misery.