CDN-Metric Estimation

A requirement of CDNs is that they need **to make a trade-off between many aspects** when it comes to hosting replicated content.

For example, **access time**s for a document may be optimal if a document is massively replicated, but at the same time this incurs a **financial cost**, as well as a cost in terms of **bandwidth usage** for disseminating updates.

First, there are **latency metrics**, by which the time is measured for an action. For example, fetching a document, to take place. **Estimating latencies becomes difficult** when, for example, a process deciding on the placement of replicas needs to know the delay between a client and some remote server. Typically, an algorithm globally positioning nodes as discussed in Chap.7 will need to be deployed.

Instead of estimating latency, it may be more important to measure the **available bandwidth between two nodes**. This information is particularly important when large documents need to be transferred, as in that case the responsiveness of the system is largely dictated by the time that a document can be transferred. There are various tools for measuring available bandwidth, but in all cases it turns out that accurate measurements can be difficult to attain. Further information can be found in Strauss et al. (2003).

CDN-Metric Estimation

Next class consists of **spatial metrics** which mainly consist of measuring the **distance between nodes in terms of the number of network-level routing hops**, or **hops between autonomous systems**. Again, determining the number of hops between two arbitrary nodes can be very difficult, and may also not even correlate with latency (Huffaker et al., 2002).

Moreover, simply looking at routing tables is not going to work when low-level techniques such as Multi-Protocol Label Switching (MPLS) are deployed. MPLS circumvents network-level routing by using **virtual-circuit** techniques to immediately and efficiently forward packets to their destination [see also Guichard et al. (2005)]. Packets may thus follow completely different routes than advertised in the tables of network-level routers.

CDN-Metrics

Another class is formed by **network usage metrics** which most often entails **consumed bandwidth**. Computing consumed bandwidth in terms of the number of bytes to transfer is generally easy. However, to do this correctly, we need to take into account how often the document is read, how often it is updated, and how often it is replicated.

Still another class; **Consistency metrics;** tell us to what extent a replica is deviating from its master copy. We already discussed extensively how consistency can be measured inthe context of continuous consistency in Chap. 7 (Yu and Vahdat, 2002).

CDN-Metrics

Finally, **financial metrics** form **fourth class** for measuring how well a CDN is doing. Although not technical at all, considering that most CDN operate on a commercial basis, it is clear that in many cases financial metrics will be decisive.

Moreover, the financial metrics are closely related to the actual infrastructure of the Internet. For example, most commercial CDNs place servers at the edge of the Internet, meaning that they hire capacity from ISPs directly servicing end users.

At this point**, business models** become intertwined with **technological issues**, an area that is not at all well understood. There is only **few material available on the relation between financial performance and technological issues** (Janiga et al.,2001).

From these examples it should become clear that **simply measuring the performance of a CDN, or even estimating its performance may by itself be an extremely complex task**. In practice, for commercial CDNs the issue that really counts is whether they can meet the **service-level agreements (SLA)** that have been made with customers.

CDN-Adaptation Triggering

Another question that needs to be addressed is **when and how adaptations are to be triggered**. A simple model is to **periodically estimate metrics and subsequently take measures as needed**. This approach is often seen in practice. Special processes located at the servers collect information and periodically check for changes.

A **major drawback** of **periodic evaluation** is that **sudden changes may be missed**. One type of sudden change that is receiving considerable attention is that of **flash crowds**. **A flash crowd is a sudden burst in requests for a specific Web document.**

In many cases, these type of bursts can bring down an entire service, in turn causing a cascade of service outages as witnessed during several events in the recent history of the Internet.

Handling flash crowds is difficult. **A very expensive solution is to massively replicate a Web site and as soon as request rates start to rapidly increase, requests should be redirected to the replicas to off-load the master copy**. This type of over provisioning is obviously not the way to go. Instead, what is needed is a **flash crowd predictor** that will provide a server enough time to dynamically install replicas of Web documents, after which it can redirect requests when the going gets tough. **One of the problems** with attempting to predict flash crowds is that they can be so very different.

CDN-Adaptation Triggering

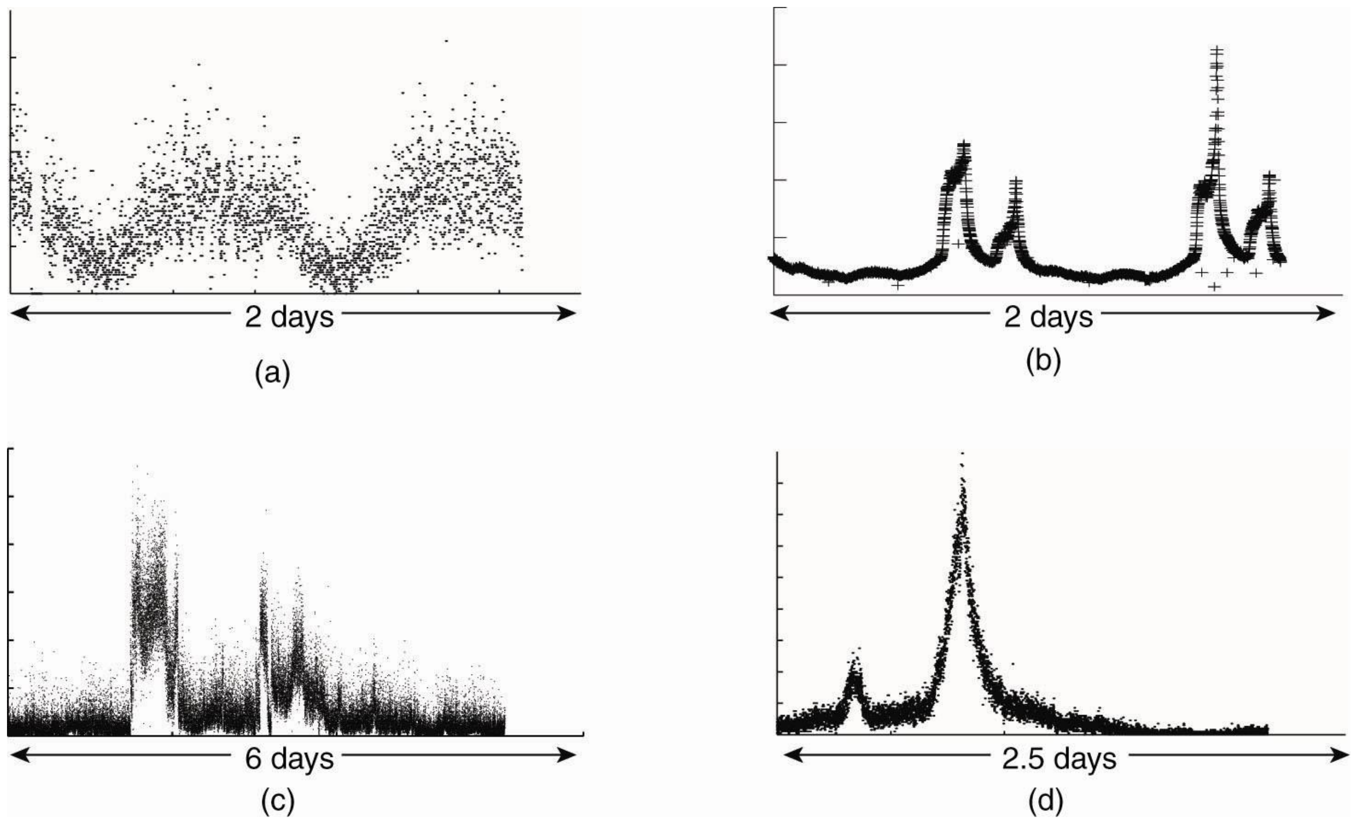


Figure 12-19. One normal and three different access patterns reflecting flash-crowd behavior (adapted from Baryshnikov et al., 2005).

CDN-Adjustment Measures

There are essentially only three (related) measures that can be taken to change the behavior of a Web hosting service:

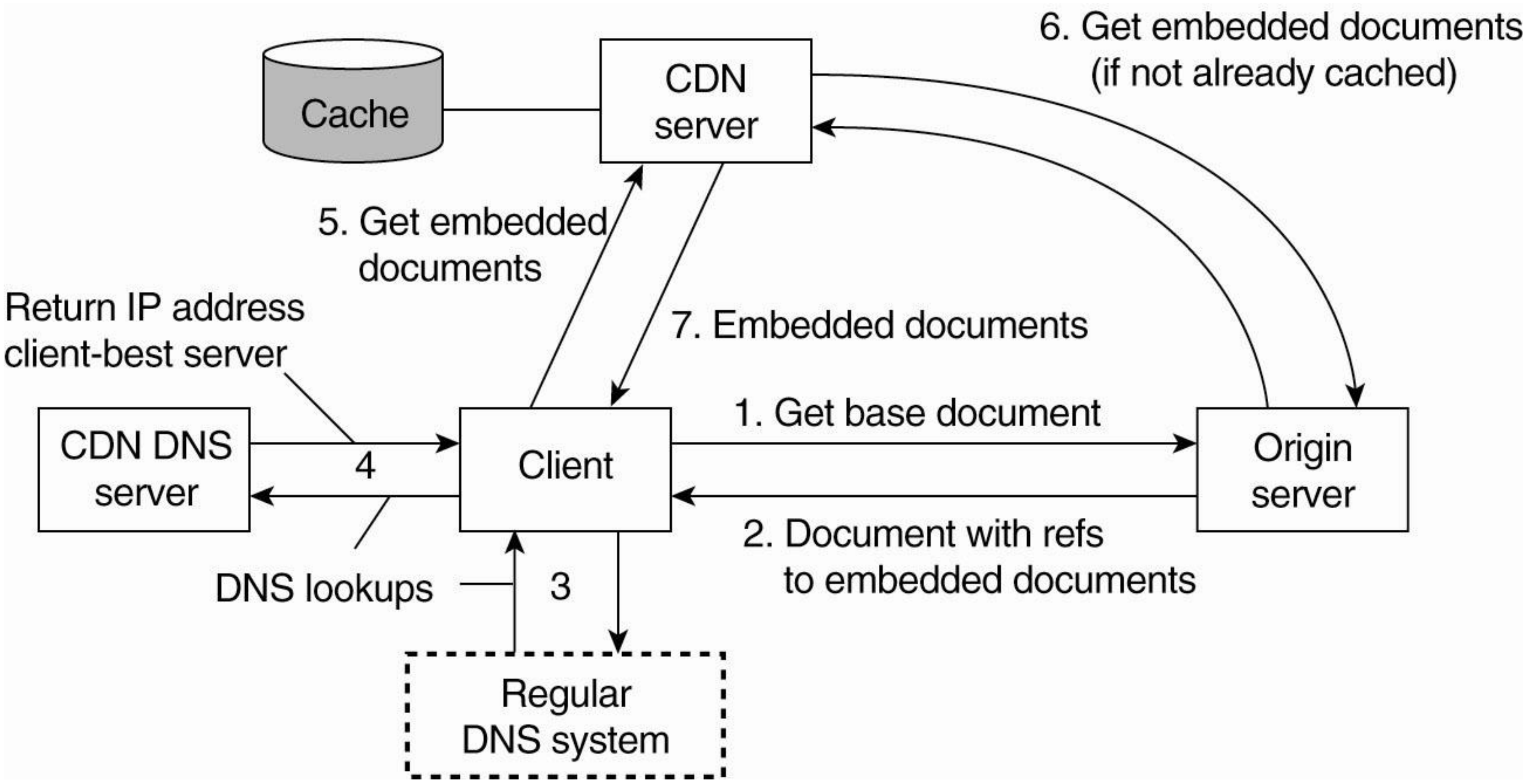
- Changing the placement of replicas -> Already discussed in Chapter 7
- Changing consistency enforcement -> Already discussed in Chapter 7
- Deciding on how and when to redirect client requests.

Client-request redirection deserves some more attention. Before we discuss some of the trade-offs, let us first consider **how consistency and replication are dealt within a practical setting** by considering the Akamai situation (Leighton and Lewin, 2000; and Dilley et al., 2002).

The basic idea is that each Web document consists of a main HTML (or XML) page in which several other documents such as images, video, and audio have been embedded. To display the entire document, it is necessary that the embedded documents are fetched by the user's browser as well. **The assumption is that these embedded documents rarely change, for which reason it makes sense to cache or replicate them.**

Each embedded document is normally referenced through a URL. However, in Akamai's CDN, such a URL is modified such that it refers to a **virtual host (virtual ghost)**, which is a reference to an actual server in the CDN. The URL also contains the host name of the origin server for reasons we explain next. The modified URL is resolved as follows, as is also shown in Fig. 12-20.

Adjustment Measures



6. Get embedded documents (if not already cached)
Cache
CDN server
5. Get embedded documents
Return IP address client-best server
7. Embedded documents
CDN DNS server
4
Client
1. Get base document
Origin server
2. Document with refs to embedded documents
DNS lookups
3
Regular DNS system

Figure 12-20. The principal working of the Akamai CDN.

CDN-Adjustment Measure

The name of the virtual ghost includes a DNS name such as ***ghosting. com***, which is resolved by the regular DNS naming system to a CDN DNS server (the result of step 3). Each such DNS server keeps track of servers close to the client.

To this end, any of the **proximity metrics** we have discussed previously could be used. In effect, the CDN DNS servers redirects the client to a replica server best for that client (step 4), which could mean **the closest one**, **the least-loaded one**, or a combination of several such metrics (the actual redirection policy is proprietary).

Finally, the client forwards the request for the embedded document to the selected CDN server. If this server does not yet have the document, it fetches it from the original Web server (shown as step 6), caches it locally, and subsequently passes it to the client. If the document was already in the CDN server's cache, it can be returned forthwith. Note that in order to fetch the embedded document, the replica server must be able to send a request to the origin server, for which reason its host name is also contained in the embedded document's URL.

CDN-Adjustment Measure

An interesting aspect of this scheme is the simplicity by which consistency of documents can be enforced. Clearly, whenever a main document is changed, a client will always be able to fetch it from the origin server.

In the case of **embedded documents**, a different approach needs to be followed as these documents are, in principle, fetched from a nearby replica server. To this end, a URL for an embedded document not only refers to a special host name that eventually leads to a CDN DNS server, but also contains a unique identifier that is changed every time the embedded document changes. In effect, **this identifier changes the name of the embedded document**. As a consequence, when the client is redirected to a specific CDN server, that **server will not find the named document in its cache and will thus fetch it from the origin server**. The old document will eventually be evicted from the server's cache as it is no longer referenced.

In principle, by properly redirecting clients, a CDN can stay in control when it comes to client-perceived performance, but also taking into account global system performance by, for example, avoiding that requests are sent to heavily loaded servers. These so-called **adaptive redirection policies** can be applied when information on the system's current behavior is provided to the processes that take redirection decisions.

CDN-Adjustment Measure

Besides the different policies, an important issue is whether request **redirection is transparent** to the client or not. In essence, there are only three redirection techniques:

- **TCP handoff,**
- **DNS redirection,**
- **HTTP redirection**

We already discussed **TCP handoff**. This technique is applicable only for server clusters and does not scale to wide-area networks.

DNS redirection is a transparent mechanism by which the client can be kept completely unaware of where documents are located. Akamai's two-level redirection is one example of this technique. We can also directly deploy DNS to return one of several addresses as we discussed before. Note, however, that DNS redirection can be applied only to an entire site: the name of individual documents does not fit into the DNS name space.

HTTP redirection, finally, is a non-transparent mechanism. When a client requests a specific document, it may be given an alternative URL as part of an HTTP response message to which it is then redirected. An important observation is that this URL is visible to the client's browser. In fact, the user may decide to bookmark the referral URL, potentially rendering the redirection policy useless.