Grid Services for Adaptive Content Delivery

Guillaume Pierre
Computer Science Dept., Vrije Universiteit, Amsterdam
gpierre@cs.vu.nl

Abstract

Content Delivery Networks must adapt their configuration continuously to maintain acceptable performance in the presence of large variations in their request load characteristics. To automate and systematize these necessary continuous adaptations, we propose to structure a CDN along a grid service-oriented architecture. Two techniques can be used to simplify the development and maintenance of such a system: first, present a group of machines offering a given service as a single stable node, even though the concerned servers may be located worldwide and the membership change often; second, modeling the performance of internal elements of a web service allows to make informed decisions on the dimensioning of the platform to host it. We expect that these two techniques will allow us to build services that continuously adapt their capacity to the demand that they are facing.

1. Introduction

The goal of any content delivery network is to deliver Web content with guaranteed quality of service regarding client-perceived download performance, operational costs and service availability. To reach this goal in the presence of large variations in the request load issued by the clients, CDNs must continuously measure their own performance and adapt their configuration when necessary. Such adaptation may take several forms: change the placement of replicas holding the content to have sufficient serving capacity located close to the clients requesting it; change the way replicas are created and maintained consistent in the presence of updates; and change the way clients are redirected to one of the replicas [11].

In a certain sense, CDNs share many characteristics with Grid services. Grid services are defined as Web services that make full use of Grid computing technology to manage transient service instances. For example, *"in a Web serving environment, service instances might be instantiated dynamically to provide for consistent user response time by managing application workload through dynamically added capacity. [...] Transience has significant implications for how services are managed, named, discovered, and used"* [4]. This shared vision that services can dynamically change the set of servers on which they run motivates this paper. More precisely, we investigate the potential benefits and issues that would arise from structuring a content delivery network as a number of Grid services interacting with each other.

Structuring a CDN as a Grid service can provide multiple benefits. First, such a system could make use of standardized features that Grids usually provide such as system and application monitoring, dynamically allocating or releasing machines within the Grid to match the demand, and security features such as authentication and authorization for the use of resources. Importantly, hosting a CDN on a Grid infrastructure would allow any user who has access to a Grid to deploy a content delivery service on resources contributed by external partners, while retaining full control on the system. This could alleviate the trust and security difficulties which originate from building collaborative content delivery networks where multiple people or organizations share their resources for everyone's benefit [3, 8]. Finally, structuring a CDN as a number of Web services requesting each other can simplify the task of adapting the system configuration, as we will discuss later in this paper.

Reaching this goal requires that a number of tasks are carried out. First, one needs to design a CDN as a group of services interacting with each other. Second, each of these services must be built such that it can dynamically adapt its own configuration to the demand that it receives. Third, as each service would be hosted on a potentially large and frequently changing set of servers, establishing bindings between servers in different layers can become increasingly difficult.

We propose two different techniques to deal with these issues. First, we show how a service can be implemented such that it continuously determines the best number of servers to reach the expected quality of service. Second, we propose to use TCP handoff techniques based on mobile IPv6 to give each layer a single abstract IP address.

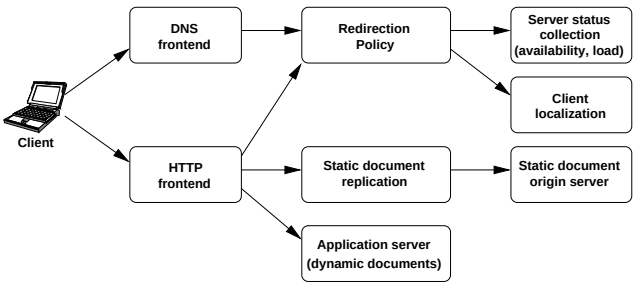

Figure 1. Modeling a CDN as a Service-Oriented Application (call graph)

Regardless of changes in the servers taking part in a given layer, this single contact address can be built such that it always allows to reach one of the servers taking part in the addressed group of machines.

The remaining of this paper is structured as follows. Section 2 briefly presents a CDN architecture based on a service oriented design. Section 3 shows how each service can be made adaptive to changes in its workload. Section 4 explains how Mobile IPv6 can be exploited to give a single IP address to a frequently changing set of servers. Finally, Section 5 draws conclusions of this paper.

2. Modeling a CDN as a Grid Service

Most CDNs are built nowadays following an ad-hoc architecture, meaning that the software architecture is tailored according to the specific demands of the application. We take here a different approach, and explore to which extent a CDN can be built following a service-oriented architecture. We believe that, although this approach imposes a number of constraints, it also enables the use of generic hosting technologies which simplify the development and allow for automatic adaptation.

At first sight, the only function of a CDN is to replicate and deliver content. However, a more detailed analysis shows that in fact many components can be abstracted and implemented as separate services. Remember that a Grid service is assumed to be implemented by any number of physical servers used to process the incoming requests. Each service is also associated with its own adaptation component which monitors the performance of the service and adjusts the number and location of servers taking part in the service accordingly.

Figure 1 shows one possible design of a CDN structured as a service-oriented application. Client requests are received by a DNS or HTTP frontend. The DNS frontend is used to redirect clients to the HTTP frontend most suited to answer this client. Deciding which HTTP frontend should treat each client's requests is done by the redirection policy service, which in turn needs information about the identity, availability and load of the HTTP frontend servers, and about the localization of the client. The server status collection is in charge of receiving information about the identity of servers taking part in the system, and continuously monitoring their availability and load status. Finally, the client localization service is used to convert a client's IP address into a location that can be used for placement. This service can for example be implemented following the principles discussed in [13].

The HTTP frontend is in charge of distributing the incoming requests to the relevant components: requests for dynamically-generated documents should be forwarded to the appropriate application servers; requests for static documents should be forwarded to the static document replication service; finally, if the CDN wants to use HTTP-based redirection, then redirection requests require calling the redirection policy to obtain the identity of the frontend to which the requests should be redirected.

The static document replication is the heart of the system. It is implemented by any number of servers which are in charge of creating copies of the documents present in the origin server, and maintain them consistent as the original documents get updated. Any replication policy can be used here. We expect that simple policies such as caching with leases or invalidation can easily be implemented in the proposed architecture.

For space reasons we do not detail the application server part here. However, it should be clear that most dynamic Web applications are implemented along a client-server that can easily be mapped to a service-oriented architecture.

3. Service Dimensioning

To achieve the requested performance, each service may need to be deployed across multiple machines. Deploying a service usually involves replicating its code to a number of application servers and its data to an array of data store machines. Furthermore, different caching layers such as for service response caching and database caching can be deployed to improve performance. Although these techniques have been introduced independently from (and often in opposition to) each other [1, 6, 7, 9, 14], we believe that they rather complement each other and may need to co-exist to obtain the best performance. We consider that different techniques are best suited for different kinds of services. For example, if requests to a service exhibit high request locality, then service caching might be beneficial. On the other hand, if the bottleneck lies in the underlying data retrieval, then database caching or replication might be useful depending on the temporal locality of database queries. Sometimes, a combination of these techniques might be needed to achieve a certain quality of service.

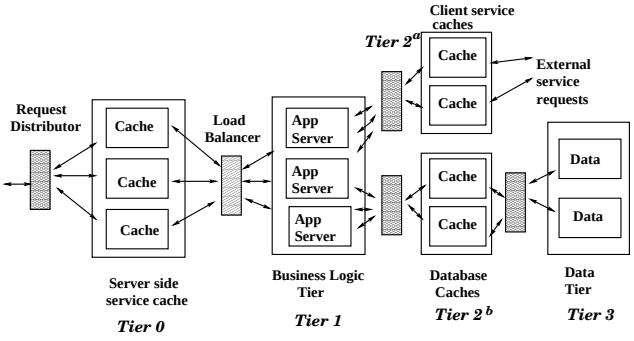

Figure 2. Internal Organization of a Service

In a previous paper, we have presented the internal architecture of an adaptive system to host service-oriented applications [10]. We describe the proposed architecture briefly here; readers are invited to refer to this paper for more information.

As shown in Figure 2, one given service can be implemented as a series of tiers. The most important tiers are the business logic tier where the application code is executed, and the data tier where the data are stored. In addition, a number of caching tiers can be introduced: a server-side service cache can store the responses generated by the service for future reuse; a database cache can be used to reduce the number of queries addressed to the database tier; finally, client-side service caches can be introduced to store the responses received in response to external service calls. Each of these tiers can be implemented by 0 or more servers (1 or more for the business logic and data tiers). Finally, load balancers are introduced to distribute requests among servers belonging to a given tier. We discuss a possible implementation of these load balancers in Section 4.

Not shown in the figure is the adaptation component of the service. This component is in charge of monitoring the quality of service offered by the service, as well as the performance of each internal component. The adaptation component executes a performance model of the service which can infer the performance that the service would have if it was running with a different configuration. When the performance of the service drops below a given acceptable level, the adaptation component decides what needs to be done to restore the performance level back to normal. Adaptation usually consists of adding or removing servers from one or more tiers.

In a CDN, the location of servers participating in a given service is crucial to the overall performance. To this end, the adaptation component of each service should query the client location service for the location of the components which address requests to them. This allows the adaptation component to select the right locations where new servers should be introduced to host the concerned service[1].

We expect that letting services adapt their own configuration independently from each other may result in efficient adaptations of the CDN as a whole. For example, if the HTTP frontend service notices a drop in the quality of service that it offers to its clients, this may be due to an inadequate location of its servers, which influences the average network latency between the clients and the frontend servers. The frontend service may then decide to add servers in locations from which many requests originate. It then simply needs to inform the server status collection service of the newly introduced frontend servers such that clients get redirected to these new servers. In turn, the static document replication service may notice that, to efficiently respond to queries originating from the new frontend servers, it also needs to add more servers in the same vicinity. As a result of these cascading effects, we believe that the CDN as a whole can soon find a new equilibrium point, until the next time that a change in workload is detected.

4. Ad-Hoc Servers for Transparent and Highly Available Services

Structuring a CDN as a group of grid services requesting each other can create a new problem, as we can expect a high churn of servers frequently joining and leaving each service. In this context, it may become difficult to establish the necessary bindings between different layers in the system. For example, if the list of servers taking part in the HTTP frontend and the static document replication service change all the time, how can we make sure that each server participating in the HTTP frontend knows the identity of at least one server taking part in the document replication service at all times?

Such bindings between layers must be realized by the load balancers shown in Figure 1. Good bindings should ideally exhibit the following properties: (i) it should always be possible to reach one of the servers taking part in a given service, even if the list of these servers change over time; (ii) contacting any server out of the list is usually not good enough; for performance reasons, we should be able to *direct* connections initiating from a given client to the closest available instance of the requested service; (iii) as the instances of a given service will be located worldwide according to the demand, it is not acceptable to implement a load balancer as a frontend machine as is commonly done in cluster computing [2]: this would imply that traffic must follow a triangular route that would seriously impact the overall system performance.

[1] We assume that functionalities such as dynamically allocating and releasing servers based on location characteristics are part of the basic features provided by Grids.

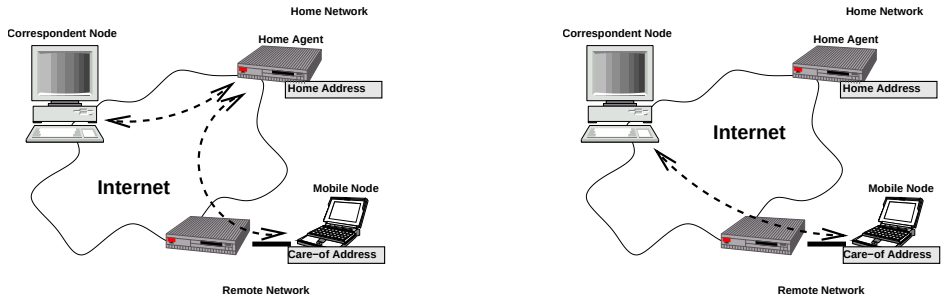

Figure 3. Communication in MIPv6: (a) tunneling, and (b) route optimization

In a previous paper, we have proposed a new technique to build so-called "ad hoc distributed servers" [12]. An ad hoc distributed server is a group of machines potentially located worldwide that cooperate to create the illusion of being a single high-performance server. Such an ad hoc server has a single IP address which can always be used to contact one of its members, regardless of changes in the ad hoc server's membership. Members of an ad hoc distributed server can also handoff connections to each other, effectively allowing for implementing location-based bindings between any client and server's member node closest to that client. It also allows for fine-grained load balancing of connections addressed to the ad hoc server by handing off individual client requests among servers, not only at the moment when the request is being received, but also while it is being serviced.

Our implementation of ad hoc distributed servers is based on the Mobile IPv6 (MIPv6) protocol. MIPv6 consists of a set of extensions to the IPv6 protocol [5]. It has been proposed to enable any IPv6 *mobile node* to be reached by any other *correspondent nodes*, even if the mobile node is temporarily away from its usual location.

To allow one to reach a mobile node while it is away from its home network and connected to some visited network, MIPv6 distinguishes between two types of addresses that are assigned to mobile nodes. The *home address* (HoA) never changes, and identifies a mobile node in its home network. The home address defines the *identity* of the mobile node. A mobile node can always be reached at its HoA.

A mobile node can also have a *care-of address* (CoA), which is obtained from a visited network when the mobile node moves to that network. The CoA represents the current physical network attachment of the mobile node and can change as the mobile node moves among various networks.

The goal of MIPv6 is to ensure uninterrupted communication with mobile nodes via their HoA, independently of their current network attachment. To this end, MIPv6 implements two mechanisms, as shown in Figure 3. Tunneling allows all traffic to be sent to the HoA of the mobile node, from which it is forwarded to its current CoA. Route optimization, in turn, consists of revealing the current CoA to the MIPv6 layer of the correspondent node so that direct communication can take place between the correspondent and the mobile node.

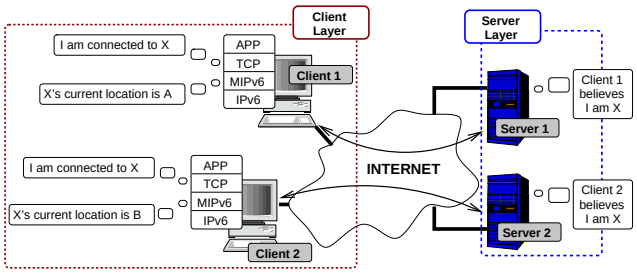

Figure 4. Using Mobile IPv6 to Bind Software Layers (the server layer has a single IP address X).

In our architecture, an ad hoc distributed server is perceived by its clients as a *single mobile node*. This means that the ad hoc server's HoA is shared between all its members. Each member node address, in turn, is considered as a potential CoA of that fictive mobile node. By disclosing different CoAs to each client, the server can convince different clients that it has moved to different locations. This allows the server to service its clients via its single contact address using many member nodes, just like a mobile node can communicate with its clients using its HoA at many different locations.

The general model of communication between an ad hoc distributed server and its clients is depicted in Figure 4. All clients nodes access the server layer using its unique contact address X. The MIPv6 layers at different clients may have a different idea about the server location and therefore communicate with different member nodes. Still, the client's higher (transport and application) layers are unaware of the distribution of the server layer and retain the illusion that they communicate with a single server X.

In our technical report [12] we show how the ad hoc server can carefully mimics the signaling of a mobile node performing route optimization to convince clients that is has moved to the location where the client is supposed to send its requests. This technique also allows to transfer entire TCP connections from one member node of the ad hoc server to another such that the handoff is totally transparent to the client-side application. We expect that this can be useful for load balancing or when a member node must be removed from the ad hoc server. It can then handoff its remaining connections to another server to be treated there, without disturbing the client.

5. Conclusion

Structuring a CDN as a group of Grid services is attractive because it allows to use standard Grid features to dynamically adapt the system to varying client request demands. We have shown how such a system could potentially be built. First, one should structure a CDN as a group of adaptive services requesting each other. Each of these services should be built such that it can dynamically adapt its configuration to the demand it receives, by way of adding and removing servers. Finally, to take the expected high churn into account, we propose to give each layer of the system a single IP address that can always be used to contact a member of the addressed layer. This organization provides separation of concerns as the client layers can remain totally unaware of the internal organization of the service they request, while benefiting from the performance of direct communication with the requested service.

Building such a system will certainly still require a lot of work. First, building an adaptive service hosting architecture as discussed in Section 3 constitutes a non-trivial piece of engineering. Second, building adaptive Grid services also requires to build good performance models of distributed service architectures such that the adaptation components can predict the performance that would result from any given change of configuration. Finally, our implementation of ad hoc distributed servers currently supports only relatively simple applications. More developments will be needed to support feature-rich applications such as a Grid service.

References

[1] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan. DBProxy: a dynamic data cache for web applications. In *Proc. Intl. Conf. on Data Engineering*, pages 821–831, 2003.

[2] V. Cardellini, E. Casalicchio, M. Colajanni, and P. Yu. The State of the Art in Locally Distributed Web-Server Systems. *ACM Computing Surveys*, 34(2), June 2002.

[3] CoralCDN frequently asked questions. Can I run a CoralCDN node? http://wiki.coralcdn.org/wiki.php/Main/FAQ#runnode.

[4] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, June 2002. `http://www.globus.org/alliance/publications/papers/ogsa.pdf`.

[5] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775, June 2004.

[6] B. Kemme and G. Alonso. A suite of database replication protocols based on group communication primitives. In *Proc. 18th Intl. Conf. on Distributed Computing Systems (ICDCS)*, page 156, Washington, DC, USA, 1998.

[7] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. In *Proc. 9th Intl. conf. on Architectural support for programming languages and operating systems*, pages 190–201, 2000.

[8] G. Pierre and M. van Steen. Globule: a collaborative content delivery network. Submitted for publication, Nov. 2005.

[9] S. Sivasubramanian, G. Alonso, G. Pierre, and M. van Steen. GlobeDB: Autonomic data replication for web applications. In *Proc. 14th Intl. World-Wide Web Conference*, Chiba, Japan, may 2005.

[10] S. Sivasubramanian, G. Pierre, and M. van Steen. Towards autonomic hosting of multi-tier internet applications. Submitted for publication, Mar. 2006.

[11] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. van Steen. Replication for web hosting systems. *ACM Computing Surveys*, 36(3):291–334, 2004.

[12] M. Szymaniak, G. Pierre, M. Simons-Nikolova, and M. van Steen. A single-homed ad hoc distributed server. Technical Report IR-CS-013, Vrije Universiteit, Amsterdam, The Netherlands, Mar. 2005. `http://www.globule.org/publi/ASHAHDS_ircs013.html`.

[13] M. Szymaniak, G. Pierre, and M. van Steen. Scalable cooperative latency estimation. Newport Beach, CA, July 2004.

[14] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Database replication techniques: a three parameter classification. In *Proc. 19th IEEE Symposium on Reliable Distributed Systems*, 2000.