A distributed architecture for gracefully degradable Web-based services

Mauro Andreolini
Department of Information Engineering
University of Modena and Reggio Emilia
andreolini@unimore.it

Sara Casolari
Department of Information Engineering
University of Modena and Reggio Emilia
casolari.sara@unimore.it

Michele Colajanni
Department of Information Engineering
University of Modena and Reggio Emilia
colajanni@unimore.it

Abstract

Modern Web sites provide multiple services that are often deployed through distributed architectures. The importance and the economic impact of Web-based services introduces significant requirements in terms of performance and quality of service. In this paper, we describe the necessary load monitoring, dispatching and access control mechanisms that allow the architecture to achieve graceful degradation even in the case of unpredictable and overwhelming user request loads. The implemented access control strategy aims to favor the completion of already initiated user sessions, with respect to requests pertaining to new sessions.

1 Introduction

Over the years, the Web has evolved from a simple container of static information to the main interface for sophisticated and dynamic network services, such as information portals, e-commerce and home banking [2], whose level of performance is often dictated by *Service Level Agreements*. The workload subject to modern Web-based information systems has become increasingly unpredictable and difficult to reproduce in benchmarking environments. Besides the usual concepts of user sessions, think times among page requests, self-similarity [4], modern services employ the concept of different “user categories” (guest, member, gold) which further complicate the analysis through the adoption of many workloads. Nowadays, the typical infrastructure for supporting Web-based services is based on a multi-tier logical architecture that tends to separate the three main functions of service delivery: HTTP interface, application (or business) logic and information repository [2]. We present a variation of this architecture in Figure 1. The *front-end node* (also called *Web switch*) acts as a representative interface for the Web site.The *dynamic content provider* layer is a cluster of application server nodes. Its primary task is to receive inbound requests for dynamic Web pages and build the appropriate responses. Requests for static objects are routed to one of the Apache Web servers of the *static content provider* layer, instead. The *information repository* is responsible for the retrieval of the information needed to build dynamic Web pages. It is built up of a cluster of database servers.

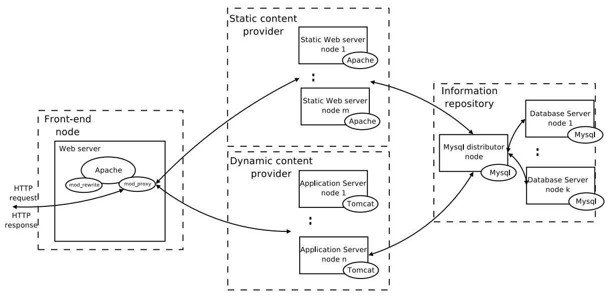

Figure 1. Proposed system architecture

Answering to a request reaching a multi-tier Web-based system is a complex task that triggers the creation and interaction of several processes, which are deeply correlated and may push the hardware and software infrastructure bejond their maximum capacity, thus causing a rapid degradation of performance. This system behavior has many negative consequences. The session may be suddenly aborted with a bad publicity for the Web site owner. An appropriate and fast reaction to service degradation in a short time interval is almost impossible. The reason behind the performance collapse is the exhaustion of suddenly degrading system resources (being them hardware or software) at one or more nodes of the architecture. This kind of resources are ''token-based", which means that only a finite number of them is available and can be assigned to requesting processes. When the available tokens are exhausted, additional requests are queued for an unpredictable time (i.e., until a token becomes free). A connection request may fail because the time-out deadline is passed or because it cannot be stored in the finite-length waiting queue of the service node. The concept of gracefully degradable is different from its traditional meaning, which is more related to the fault tolerance and the recoverability of single nodes after a hardware or software failure. By "graceful degradation", we mean a controlled request refusal and a possible increase of the response time (instead of a sudden peak as in traditional overloaded systems) to keep the load within the system capacity. In this paper,we propose load monitoring, dispatching and access control mechanisms that allow the architecture to achieve graceful degradation when subject to request loads exceeding the maximum capacity of the system. The implemented access control strategy aims to favor the completion of already started user sessions, with respect to requests pertaining to new sessions.

2 Requirements of a gracefully degradable architecture

Access control. An access control process is needed to allow the Web-based service to meet the predictive service targets or, at least, to mitigate the effects of system overload due to excessively high request rates. This process consists of two steps: *declaration* and *access*. Declaration has the purpose of estimating the amount of resources needed to fulfill the requirements of an entire user session (through the assignment of proper "weights"). After service classification, the access control mechanism checks whether there are sufficient resources to satisfy the entire user session, and to decide about allowing or refusing the request. Admission control is essential to avoid overload at the server nodes, because this causes a significant loss of throughput [3].

Classification of a Web session. A mechanism for Web session *classification* is needed to allow the Web cluster to take any kind of operation on a request, such as to recognize a starting session, to dispatch a request pertaining to an existing user session to the same servers. The classification process consists of two steps: *identification* and *prioritization*. The identification process tracks each request with the purpose of identifying starting sessions. This function is vital to the dispatcher that has to keep track of active user sessions. The prioritization process understands the type of service from the past requests, with the goal of assigning "priorities" to the most important sessions. The latter task is crucial, since it is impossible to understand the service type from the very first request (usually the home page).

Session-based admission control. Given the session-like nature of present Web workloads, it is mandatory that the admission control mechanism operates on the basis of the entire user session. Indeed, admission on the basis of single requests is an error because, based on the load of the server nodes, the system risks serving low-priority requests and refusing high-priority requests. Ideally, a session-based admission control strategy denies access to new requests while favoring those related to already initiated sessions.

Session-aware request dispatcher. The infrastructure of modern, dynamic Web-based systems supports the concept of *user session*. Most services offered to users use state information which is preserved across requests. There are different mechanisms for storing the data pertaining to a session; some store information at the clients (*cookies*), others store information at the server nodes (*session beans*). The dispatcher must be aware of these mechanisms and its primary goal is to preserve state information across requests. This implies deploying a locality-aware dispatching mechanism, that assigns requests for documents of the same session to the same application server node. The dispatching module, while trying its best to serve all requests of a user session, can decide to refuse requests for less critical services, to allow graceful degradatation.

Evaluation of server state. The admission controller and the dispatching algorithm must use some information related to the back end server nodes to perform a sub-optimal assignment or, at least, to minimize the risks of load unbalance. Server load information tends to become obsolete quickly [5]; thus, rather then relying upon time-synchronous updates every n seconds, it is preferable to process it, in order to achieve more meaningful figures of the system state. Typical examples of statistical techniques include weighted and moving averages of the past k performance samples, and smoothing [1].

3 Design of the prototype

Figure 2 shows the high level design of the proposed system. As soon as an HTTP request reaches the Web switch, the *admission control* module tries to associate it to an existing user session. If the request pertains to an active user session, it is immediately admitted into the system, unless the system is so overloaded that a request refusal is forced. If the request belongs to a brand new user session, the admission control module checks whether there is a sufficient amount of available system resources and, based on the recent behavior of system load, decides to admit into or to deny access to the system. In the former case, the request is answered with a canonical *Internal Server Error* message. In the latter case, the request is passed to the *dispatcher* module, which has the task of selecting the "best" application server to serve it. If the request belongs to an active session, it is immediately dispatched to the application server handling the appropriate session. If the request is the first of a new user session, an application server is chosen according to a locality-aware policy.

The *access control* mechanism includes two of the most important functions, which are tightly coupled: *admission control* and *request dispatching*. Admission control selectively enables users to enter the system, based on the load conditions of the server nodes. This is the primary mechanism to avoid overload and abrupt degradation. Request dispatching is necessary to assign

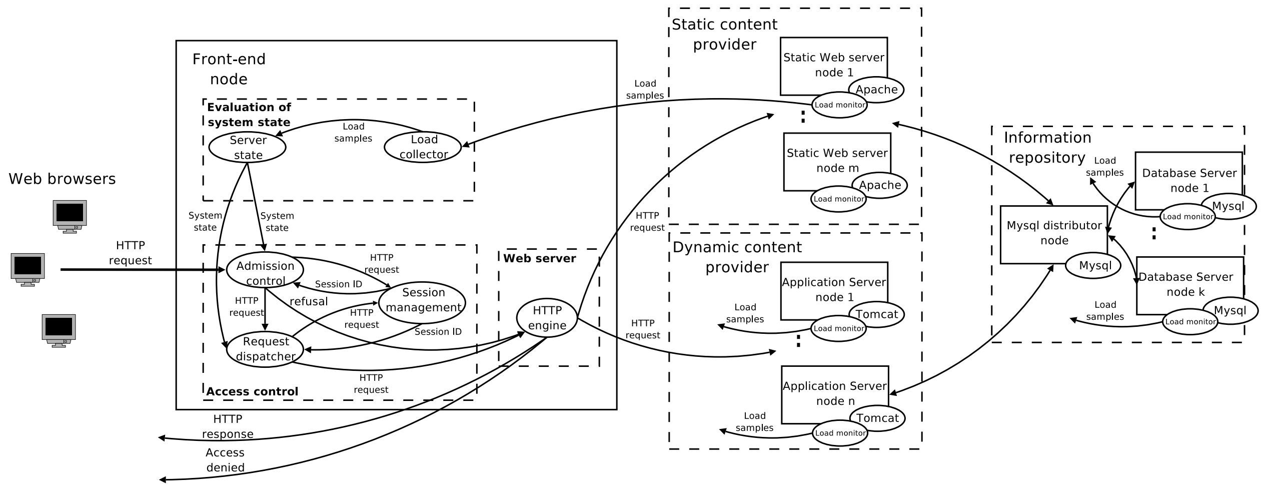

Figure 2. High-level design of the front end node

admitted requests to the proper Web server or application server. Information about the infrastructure nodes, computed by distributed *load monitor* processes (one for each node), is collected by the *load collector* process at the front-end node. The architecture shown in Figure 2 tries to accomplish the following main requirements. The *session management* module stores the identificators of all active sessions for efficient retrieval by the admission controller and dispatcher. The load information, which is used to process the state of the entire system, is computed on a per-node basis through *load monitor* modules executed at each node. These modules collect information about the resource usage (for example CPU, disk, network interface, open sockets) and send it to the *load collector* module at regular time intervals. Here, the various samples are collected and sent to the *system load state* module, which processes them to build a representative view of the system state. Typical operations include the weighted (or exponential) mean of past sample values, or even more advanced processing which involves the trend of recent/past performance samples. The load collector makes the resulting system state available to the admission control and dispatcher modules.

References

[1] Y. Barishnikov, E. Coffman, G. Coffman, G. Pierre, D. Rubinstein, M. Squillante, and T. Yimwadsana. Predictability of web server traffic congestion. In *Proc. of the 10th Int'l Conf. on Web content Caching and Distribution (WCW2005)*, sep 2005.

[2] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, and W. Zwaenepoel. Performance comparison of middleware architectures for generating dynamic web content. In *Proc. of the ACM/IFIP/USENIX Int'l Middleware Conference (MIDDLEWARE2003)*, June 2003.

[3] L. Cherkasova and P. Phaal. Session based admission control: a mechanism for improving performance of commercial web sites. In *Proc. of Int'l Workshop on Quality of Service*, June 1999.

[4] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Trans. on Networking*, 5(6):835–846, Dec. 1997.

[5] M. Dahlin. Interpreting stale load information. *IEEE Transactions on Parallel and Distributed Systems*, 11(10), 2000.