JERA: A Scalable Web Server

A. Belloum, A.J.H. Peddemors and L.O. Hertzberger
Computer Architecture and Parallel Systems Group
Dept. of Computer Science
Universiteit van Amsterdam
Kruislaan 403 NL 1098 SJ Amsterdam
Email: {adam,arjan,bob}@wins.uva.nl

Abstract *The immense growth of the number of Web Servers and the pressure on the existing infrastructure forces the Internet community to investigate new models for the exchange of information. The traditional web server architecture running an HTTP daemon reading the data from the file-system does not fit well the client server paradigm. In this web server architecture the data management is becoming harder and harder. New solutions, in the area of parallel processing and efficient data management, are needed to enable the processing of the huge number of requested sent to a web cache server. In this paper, a distributed web cache server based on Object Oriented Database technology is proposed to sustain the high throughput of requests introduced by the web caching application running on it.*

Keywords: Web server technology, Web caching, Distributed Database Systems

1 Introduction

The Internet is becoming a victim of its own success due to the rapid increase of Internet imposing more and more pressure on the infrastructure. Several web characterization studies have shown that a large number of the Internet traffic results from repeated access to the same documents. It has been outlined that only a small percentage of the files available on a given server are requested a disproportionate number of time by the web clients. [7]. Bestavros confirms this result in [2], he shows that the more popular the server, the smaller the fraction of the documents that account for most of its accesses. Besides, Gwertzman shows that the most popular files are the one that are less frequently updated [5]. Furthermore, Braun indicates in the analysis of the NCSA[1] access log files that the most popular files are under 10 kbytes [3]. From the previous studies, it is possible to identify a class of documents that greatly affect the network traffic. These documents can be characterized by the following features:

- They represent a small percentage of the files at a given server.
- They are responsible for a large percentage of the accesses to the server.
- They have a relatively small size.
- They are less frequently updated.

This class of files seems to be worth to be considered in the Internet traffic problem. The characteristics listed above show that they are really suitable for a caching mechanism. Therefore, Web Caching emerges as a potential solution to the Internet traffic jam. Web caching mechanisms introduce a new category of web servers, which act as an intermediate between the user and the origin server. Roughly speaking a web cache server receives requests from geographically close users seeking for documents distributed throughout the Internet.

[1]NCSA: National Center Super-computing Application

Basically, the web cache server keeps a copy of frequently requested documents, if the user's request matches with one of these documents, he receives a copy of the cached document without going back to the remote original server. This mechanism reduces the amount of Internet traffic and the delays of getting the documents. To reach this goal, the web cache servers should have enough resources to handle the continuous increasing number of requests. The need for high performance, scalable and fault tolerant web cache servers is becoming urgent, many examples have already shown that web cache servers could be the new bottleneck for the web caching systems. The term *high performance* concerns both a better resource management and the appropriate hardware that support scalability and fault tolerance.

Currently, the usage of frameworks supporting the implementation of distributed object oriented Internet applications, such as the Common Object Request Broker Architecture (CORBA), and the Java's Remote Method Invocation (RMI), is becoming more and more accepted. Such a frameworks would enable a web client to connect to, for instance, database server objects, thereby bypassing the Common Gateway Interface (CGI) mechanisms allowing dynamic data generation and database access. The latter is often described as stateless, not scalable and having poor performances [10]. Given the previous considerations, an object oriented, high performance, scalable and fault tolerant database system embedded in the web server architecture may dissolve some of the inefficiencies associated with the traditional web server, especially the lack of scalability and data management.

The rest of the paper is organized as follows: In Section 2, the main tasks involved in web caching are discussed. In Section 3, a description of the JERA web server is given as well as the web cache simulation model. In Section 4, some simulation and implementation results are discussed. Finally, Section 5 concludes the paper.

2 Web caching

In this section, we discuss in more detail the web application to be run on top of the JERA web server. The basic idea behind caching is to keep copies of the documents which have recently been fetched. On the web, caching typically involves the browser itself remembering the documents which were returned when the URLs (URL: Uniform Resource Locator) were fetched, by storing them on the disk, and possibly sending its request via a *cache server* rather than using the URL to directly access the remote server. The *cache server* allows large number of users to share a pool of cached documents.

The *cache server* takes the URL of the object required by the web browser and examines a database of a local copies that it has already cached. If the object associated with the given URL is held in its database, the *cache server* returns its local copy of the object straight back to the browser. If not, it uses the URL to retrieve the object from the remote server, passes one copy of the object to the end-user's browser and keeps another copy of the object into its database.

2.1 Replacement strategies

On a cache miss, if the cache is full, one of the entries of the cache must be replaced to store the missing document. Several document replacement strategies have been described in the literature. The document replacement strategy is defined as the policy that should be followed to remove cached documents in order to make room for a new referenced one. The removal process is started once the web cache is full or the remaining amount of memory is not enough to store the new incoming document. However, other approaches exist, which remove documents after a certain amount of time units (known as periodic removal policies), or policies removing documents as soon as a predefined threshold is reached [11]. The argument for using periodic removal and threshold based strategies is that if a cache is nearly 100%

full, then running the removal policy only on-demand (when no more space is available) will invoke the removal policy on nearly all document requests. If removal is time consuming, it might generate a significant overhead. On the other hand, periodic removal and threshold based strategies reduce the hit rate because documents are removed earlier than needed.

2.2 Document coherency

Unfortunately, caching mechanisms lead to consistency problems: the copies of the documents cached at the cache server should be kept up-to-date with the original ones. Conventional coherency technique such as the *invalidation protocol* or client *polling every-time* have been considered expensive, the amount of control messages exchanged between the client and the server have had a bad impact on the Internet traffic. The caching server always sends a "conditional GET"[2] to the next higher cache or server to verify if the cached copy is still up to date. Such a mechanism would result in a message to the document's origin server for each one of the copied documents. Thus the home servers of popular documents would be overwhelmed with "conditional GET" requests.

Another category of methods emerges and introduced a new concept of coherency: *the weak coherency.* Several web designers claim that the strong document coherency is not necessary for the Web, since the major documents that cross the network everyday have a quite large life cycle (less than 2% are updated every day [2]). Often, these documents are texts, images and sound/video files. Instead of checking at each new request, each category of documents is assigned a *Time-To-Live "ttl"* and checking is performed only when this time elapses. The *ttl-tag* is an a priori estimate of a document's lifetime that is used to determine how long cached data remain valid. Each document is assigned a *ttl-tag*, and is marked invalid when the *ttl* elapses [5]. *ttl-tag*'s are simple to implement in HTTP using the optional "expires" header field. The challenge in supporting *ttl-tag* lies in selecting the appropriate time out value. If the latter is too small, data may be reloaded unnecessarily, if it's too long stale data may be returned.

[2]Conditional GET is a request header defined by HTTP protocol

3 JERA web server

3.1 Architecture Description

The web server proposed in this paper is based on heterogeneous hardware architecture composed of an SMP (Symmetric Multi-Processor) entry-point machine and an MPP (Massively Parallel Processor) machine used as a database server. The reason for having a single entry point is based on the fact the web sites are associated with a single IP address. There are two main reasons for having a distributed database server as a part of the web server. The first reason is the easy way to support the consistency among cross references inherently included in the HTML pages nowadays. The second is to prepare the web server to face high performance web application, which require a use of an efficient database. The data management within the MPP machine is performed by the "Matisse Database System" [8]. This database management system is based on the client-server paradigm: client applications con-

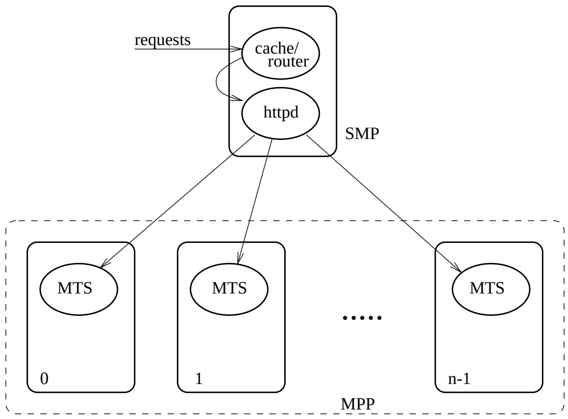

Figure 1: JERA web server architecture

nect via a network to a server, where the data is stored, in order to perform actions like object search, data retrieve, data updates, etc. The server, called MTS, supports these actions with a relatively simple set of possible operations on the limited number of server data structures, as defined by the Server Engine Services (SES). A client application may need services at a higher level, for instance services which allow for Object Oriented database access. The services are provided by a number of Matisse client libraries and association APIs. The most important one is the Object oriented Services (OOS) C API, extended with C++ binding and an SQL interface. The client libraries use the low-level server interface to implement their services.

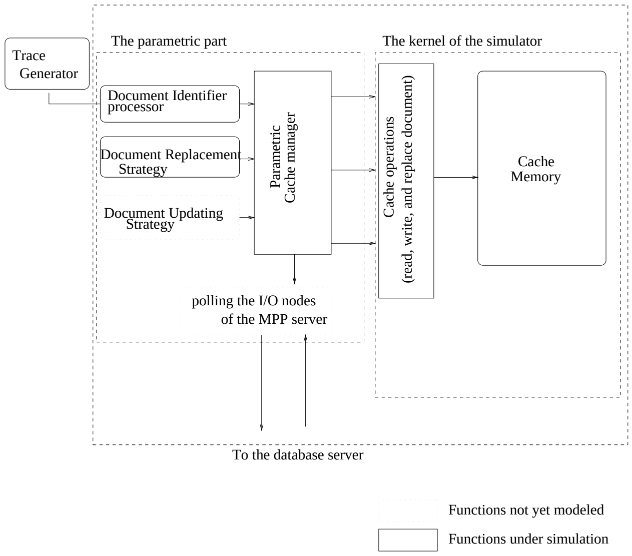

Figure 2: The web server Simulation model

3.2 Web cache simulation model

To investigate web caching problems, we have built a simulation model that allows us to compare several web cache strategies. The major problem in building simulators is the fact that they have to be flexible enough to allow a wide range of simulations. To implement a web cache simulator, we have divided the main tasks of the cache web server into two sets. The first set includes the basic tasks necessary for the cache functioning but which are not under the focus within the web cache community such as: determining whether or not the current requested document is cached, providing the requested document, and so on. The second involves all the tasks that are supposed to have a great impact on the web cache performance. To address these two types of tasks, we have divided the simulator architecture in two parts: the kernel of the simulator which involves the simulation independent set of tasks, and the parametric part of the simulator which involves the tasks that have a direct interaction with the currently performed simulation (Figure 2). The parametric part of the simulator is implemented in such a way that it reduces the number of modifications when moving from one simulation to another.

3.2.1 WCS architecture

To simulate the basic functions of the web cache server such as the document updating strategy and the replacement strategy, we start by implementing a model that manages the cache memory using a parameter called "document priority". Each document is assigned a priority, which value depends on the caching method being simulated, the set of documents contained in the cache are then sorted according to the "document priority" values. To do so, we define the organization of both the cached documents and the memory cache itself.For each document stored in the cache we store the following data:

- Document signature: a set of bytes used to identify the document.
- Document size: this field contains the document size (in bytes).
- Document entry time: this field contains the time the document was referenced.
- Document validation flag: this flag is set only when the document is up to date.

- Document priority: this field contains the priority associated with the document. The priority can be assigned to the document according to different strategies as mentioned before.

- Document dummy priority: this is a second priority level used to sort the cached documents when it is necessary (used for the sorting process initialization described in [1]).

The memory cache organization is as follows:

- The communication buffer: used to store newly referenced documents before they are moved to their correct position in the memory cache.

- The cache: contains the document previously referenced and which are ordered according to the priority field of each document.

3.3 The database server

The design of the distributed database system is depicted in figure 3. A client is granted full database access by connecting to a single server, the so called entry server. Each sever in the system performs the role of entry server for the clients that are currently connected to it. The client connects to an Application specific extension (Extension), this interface involves functional requirements specific for each application using the distributed database. The communication handler (CH) takes care of the efficient communication between the nodes. It support broadcasting requests and gathering of replies over the high speed network connecting the nodes. Form an entry server point of view, all the other servers are called internal servers. This means that each server in the system also performs the role of internal server.

Suppose a server defines an object read operation and an object may reside on any node in the system. The following action take place on the entry server when a client reads an object:

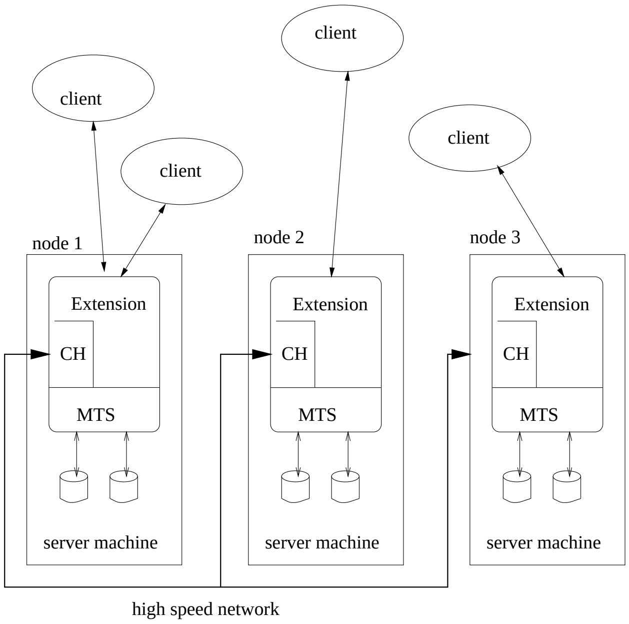

Figure 3: The distributed database system: clients connects to an application specific extension to the Matisse server (MTS)

- The client connects to the entry server.

- The client send the request to the entry server.

- The request is broadcast to the other nodes.

- Reply data is gathered by the entry server using the communication handler.

- The client sends the next request or close the connection.

The internal server uses a dispatch mechanism offered by the communication handler to process incoming broadcast requests from entry servers.

Obviously, multiple clients may read objects at the same time connecting to any server in the system. This means that the network between the nodes is used for multiple reads actions at the same time. Therefore, the capacity for the internal network imposes a maximum throughput restriction. From the throughput