Ministry of Education of Republic of Moldova
Technical University of Moldova
CIM Faculty
Anglophone Department

Report

On Distributed Aplications Programming
Laboratory Work №4-6

Performed by: st. gr. FAF-131 **Boldescu A.**
Verified by: conf. univ. dr. **Ciorba D.**

Chisinau, 2016

Laboratory Work №4-6

Topic:

Studiul protocoalelor de transport TCP/IP în contextul dezvoltării unei aplicații conținând colecții distribuite de date. Studiul modelelor pentru procesarea datelor XML (DOM/ SAX) și JSON la distribuirea acestora. Studiul protocolului HTTP în contextul distribuirii datelor. Utilizarea metodelor HTTP în implementarea interacțiunii dintre client și un server de aplicații. Studiul metodelor de caching și load-balancing aplicate la crearea unui serviciu proxy.

Objectives:

- Aplicarea protocolului UDP în transmisiuni unicast și multicast;
- Aplicarea protocolului TCP în transmisiuni de date;
- Procesarea colecțiilor de obiecte;
- Dezvoltarea unui sistem cu date distribuite eterogene și utilizarea unui mediator pentru accesarea acestora;
- Crearea unui modul de validare a datelor de format XML ce parvin centralizat de la nodul de mediere, Maven, spre client;
- Elaborarea unui server cu procesare concurentă a cererilor HTTP;
- Realizarea unei aplicații de intermediere a accesului la nodurile warehouse.

Used technologies:

requests

As the creators of this library put it: "Requests allows you to send organic, grass-fed HTTP/1.1 requests, without the need for manual labor. There's no need to manually add query strings to your URLs, or to form-encode your POST data. Keep-alive and HTTP connection pooling are 100% automatic, powered by urllib3, which is embedded within Requests."

BaseHTTPServer

A standard Python library used to create HTTP servers. Can handle multiple HTTP requests at a time.

json / xmltodict / dicttoxml

Standard and non-standard libraries used to serialize-deserialize the data in JSON and XML.

lxml

The lxml XML toolkit is a Pythonic binding for the C libraries libxml2 and libxslt. It is unique in that it combines the speed and XML feature completeness of these libraries with the simplicity of a native Python API, mostly compatible but superior to the well-known ElementTree API. Used to validate the XML documents with an XML schema.

The Task:

Distributed Database Systems

A distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network. This definition leads to some implicit assumptions like:

- Data stored at a number of sites, each site logically consists of a single processor;
- Processors at different sites are interconnected by a computer network (we do not consider multiprocessors in DDBMS, cf. parallel systems);
- DDBS is a database, not a collection of files (cf. relational data model). Placement and query of data is impacted by the access patterns of the user;
- DDBMS is a collections of DBMSs (not a remote file system).

One may consider the use of Distributed Database Systems because they can deliver the following advantages:

- Higher reliability - no single points of failure as well as distributed transaction processing guaranteeing the consistency of the database and concurrency;
- Improved performance - parallelism in execution and proximity of data to it's points of use (reduces access delays);
- Easier system expansion - because database scaling was always an issue;
- Transparency of distributed and replicated data - hiding the implementation details from the user.

Nevertheless one must also consider the possible Complicating Factors that implementing a DDB may bring:

- Complexity of the DDB compared to a non distributed DB;
- Cost of implementing a DDB;
- Ensuring security of the DDB;
- More difficult integrity control;
- Lack of standards;
- Lack of experience;
- More complex database design.

Semi-Structured data

Semi-structured data is a form of structured data that does not conform with the formal structure of data models associated with relational databases or other forms of data tables, but nonetheless contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data. Therefore, it is also known as self-describing structure. The entities belonging to the same class may have different attributes even though they are grouped together, and the attributes' order is not important. Semi-structured data are increasingly occurring since the advent of the Internet where full-text documents and databases

are not the only forms of data anymore, and different applications need a medium for exchanging information. 2 very popular types of semi-structured data are XML and JSON.

XML stands for eXtensible Markup Language and was designed to store and transport data. XML was designed to be both human- and machine-readable. It is defined by the W3C's XML 1.0 Specification and by several other related specifications, all of which are free open standards.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.

Semi-structured data are known to have the following advantages:

- Programmers persisting objects from their application to a database do not need to worry about object-relational impedance mismatch, but can often serialize objects via a light-weight library.
- Support for nested or hierarchical data often simplifies data models representing complex relationships between entities.
- Support for lists of objects simplifies data models by avoiding messy translations of lists into a relational data model.

Also, one must consider the disadvantages of semi-structured data:

- The traditional relational data model has a popular and ready-made query language, SQL.
- Prone to "garbage in, garbage out"; by removing restraints from the data model, there is less fore-thought that is necessary to operate a data application.

HTTP and REST

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World-Wide Web global information initiative since 1990. The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the Internet. HTTP/1.0, as defined by RFC 1945 , improved the protocol by allowing messages to be in the format of MIME-like messages, containing meta-information about the data transferred and modifiers on the request/response semantics. However, HTTP/1.0 does not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections, or virtual hosts. In addition, the proliferation of incompletely-implemented applications calling themselves "HTTP/1.0" has necessitated a protocol version change in order for two communicating applications to determine each other's true capabilities.

This specification defines the protocol referred to as "HTTP/1.1". This protocol includes more stringent requirements than HTTP/1.0 in order to ensure reliable implementation of its features.

Practical information systems require more functionality than simple retrieval, including search, front-end update, and annotation. HTTP allows an open-ended set of methods and headers that indicate the purpose of a request . It builds on the discipline of reference provided by the Uniform Resource Identifier (URI) , as a location (URL) or name (URN) , for indicating the resource to which a method is to be applied. Messages are passed in a format similar to that used by Internet mail as defined by the Multipurpose Internet Mail Extensions (MIME).

HTTP is also used as a generic protocol for communication between user agents and proxies/gateways to other Internet systems, including those supported by the SMTP, NNTP, FTP, Gopher, and WAIS protocols. In this way, HTTP allows basic hypermedia access to resources available from diverse applications.

REST is the underlying architectural principle of the web. The amazing thing about the web is the fact that clients (browsers) and servers can interact in complex ways without the client knowing anything beforehand about the server and the resources it hosts. The key constraint is that the server and client must both agree on the media used, which in the case of the web is HTML.

An API that adheres to the principles of REST does not require the client to know anything about the structure of the API. Rather, the server needs to provide whatever information the client needs to interact with the service. An HTML form is an example of this: The server specifies the location of the resource, and the required fields. The browser doesn't know in advance where to submit the information, and it doesn't know in advance what information to submit. Both forms of information are entirely supplied by the server.

So, how does this apply to HTTP, and how can it be implemented in practice? HTTP is oriented around verbs and resources. The two verbs in mainstream usage are GET and POST, which I think everyone will recognize. However, the HTTP standard defines several others such as PUT and DELETE. These verbs are then applied to resources, according to the instructions provided by the server.

Proxy

A proxy server is a server (a computer system or an application) that acts as an intermediary for requests from clients seeking resources from other servers.A client connects to the proxy server, requesting some service, such as a file, connection, web page, or other resource available from a different server and the proxy server evaluates the request as a way to simplify and control its complexity. Proxies were invented to add structure and encapsulation to distributed systems. Today, most proxies are web proxies, facilitating access to content on the World Wide Web and providing anonymity. There can be many uses for a proxy, such as:

- Monitoring and filtering;
- Improving performance;
- Translation;
- Accessing services anonymously;
- Security.

An interesting type of proxies is the reverse proxy. A reverse proxy (or surrogate) is a proxy server that appears to clients to be an ordinary server. Requests are forwarded to one or more proxy servers which handle the request. The response from the proxy server is returned as if it came directly from the original server, leaving the client no knowledge of the origin servers. Reverse proxies are installed in the neighborhood of one or more web servers. All traffic coming from the Internet and with a destination of one of the neighborhood's web servers goes through the proxy server. The use of "reverse" originates in its counterpart "forward proxy" since the reverse proxy sits closer to the web server and serves only a restricted set of websites.

There are several reasons for installing reverse proxy servers:

- Encryption / SSL acceleration;
- Load balancing;
- Spoon feeding;
- Security;
- Extranet Publishing.

The Solution: Distributed database

Structure

The main actors in the database are two types of nodes: the distributed databases (called Nodes) which can function as an usual non-distributed database, and a master node (called Maven) whose job is to request data from the Nodes and combine it into one response, then send it to the client.

Discovery

On startup and until shutdown the Nodes have a thread dedicated for broadcasting their ip address, on a predefined port. The Maven has a thread as well, dedicated for listening on the predefined port. When a new ip address is learned, the Maven saves the address in a list thus able to access it from now on.

This protocol allows for correct learning of the node's ip address, whichever Node / Maven starts first.

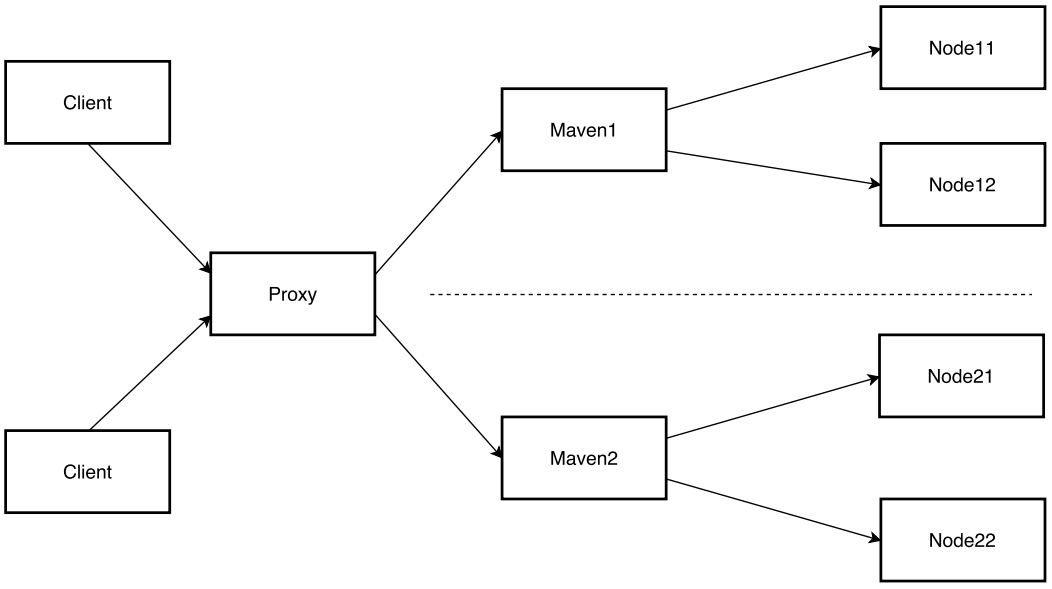

Figure 1: Black box representation of the main components of the database

Further improvements

- Ever dough the database implements the *delete_one* method, none on the elements of the system (Maven, Node) can handle delete requests. An improvement would be to implement handling of this HTTP predicate as well as adding the possibility to update an entry in the database, to totally satisfy the CRUD principles;
- Implement some higher order functions (Filter / Sort / Group) at the client level, for better processing of the data;
- Account for unexpected shutdowns of some Nodes (either by creating a "dummy" Node at the address of the shutdown Node or by pinging the Node in case it stopped broadcasting).

The Solution: JSON / XML

Because the actors in this project may well be implemented on totally different languages / operation systems, it is imperative that they communicate with messages they all understand. The semi-structured data are a good choice because it can easily be deserialized in something that a No-SQL database can handle.

Nodes and Maven

Thus the objective for this part of the project states that the communication between the Nodes and Maven must be implemented using JSON formatted data.

Maven and Client

The communication between the Maven and the Client must be implemented using XML formatted data.

XML Schema

On the client side there exists the possibility of validation of the incoming XML documents using an XML Schema. The XML Schema Definition is a recommendation of the World Wide Web Consortium that specifies how to formally describe the elements in an Extensible Markup Language document. It can be used by to verify each piece of item content in a document.

The Solution: HTTP

Even more important is that the actors of the system communicate using the same set of rules. The task was to use the HTTP protocol.

Requests

The HTTP requests between the elements of the system are performed using the *requests* library, which provide an easy interface.

Servers

The HTTP servers the system are implemented using the *BaseHTTPServer* library, which provide an easy interface and is a standard Python lybrary.

Thread per Request

The *BaseHTTPServer* library allows, with some minor tweaking, to process the requests that come in a parallel manner, by using the thread per request concept. This was used in the Maven server to allow for fasted processing of the requests of several users, but was not used at the Nodes servers, to avoid anomalies / conflicts while updating the database.

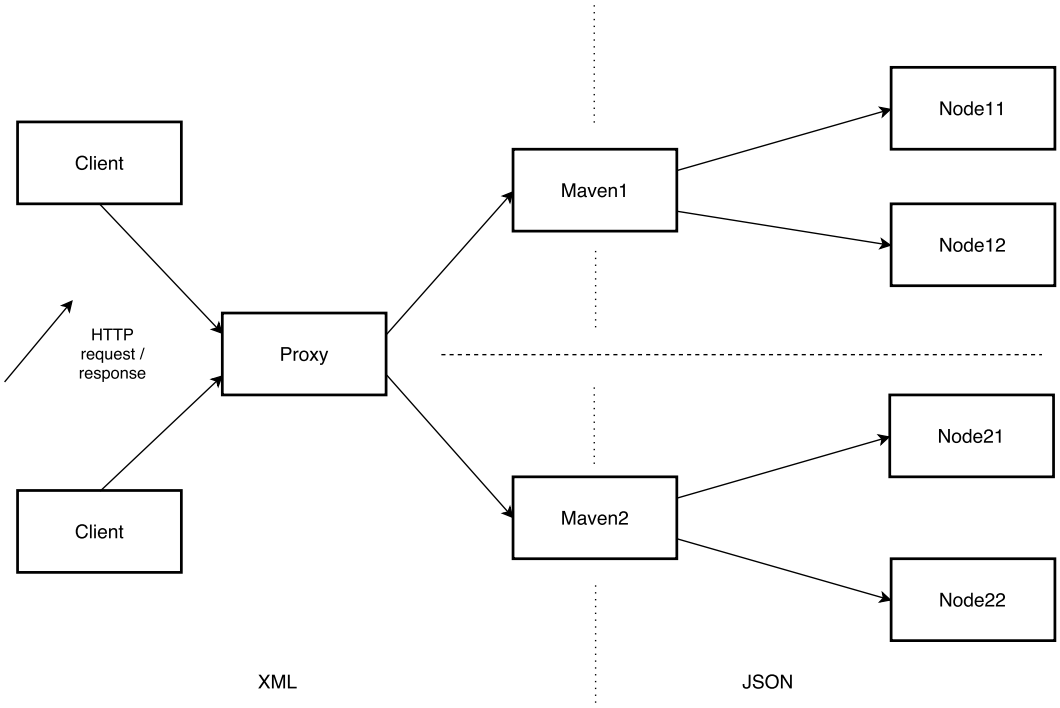

Figure 2: Black box representation of the communication between the main components of the database

Further improvements

- Although the Maven can process in parallel several client requests, it performs the Node requests sequentially. An improvement would be to implement this process in a parallel manner (using threads to request, then join the results).

The Solution: Proxy

The implemented proxy is a reverse proxy. It means that for the client the proxy IS the database - everything it does seems to be done by a database. In fact, the proxy just passes through itself the client's requests. The main reason for implementing this tipe of proxy is to manage some internal aspects of the database.

Load Balancing

Load balancing is performed between two identical copied of the distributed database. If one of them has at least one more connection than the other, the proxy redirects the request to the other copy. The connection count is performed by tracking the number of threads, names of which are saved in a list each time a request is sent to one of the database copies.

Caching

Caching is performed only on *get_one* requests so that if someone requested an item with an id, the next time this item is requested, the proxy will return the saved answer, without bothering the databases.

Smart Proxy

The proxy can handle requests from several client and will answer each request, knowing where it came from.

Further improvements

- If the *delete_one* request would be implemented, the caching needs to be updated. The same goes for *update_one* requests. Also, one may want to cache *head* requests or *even* put requests.

Conclusion:

In this laboratory work, a distributed database system, consisting of Nodes and a Maven, with a proxy was created. A distributed database is needed when the volume of data is too big for a single, non-distributed database, or is speed and redundancy is key. The components of the application become less interconnected, communicating over a network. With the addition of a proxy, the possibility for load balancing and caching appeared.

References:

http://www.inf.unibz.it/dis/teaching/DDB/ln/ddb01.pdf
http://www.json.org/
http://www.w3schools.com/xml/
https://www.w3.org/Protocols/rfc2616/rfc2616-sec1.html#sec1.1
http://stackoverflow.com/questions/671118/what-exactly-is-restful-programming
http://lxml.de/