Development of Grid Services using Erlang

Emanouil, Atanassov
Institute of Information and Communication Technologies
-BAS
Sofia, Bulgaria
e-mail: emanouil@parallel.bas.bg

ABSTRACT
The establishment of production-level Grid infrastructure in Europe motivates the development of flexible frameworks for Grid services. The Erlang programming language is well known for its emphasis on concurrency, robustness and distributed computing. Since the authentication and authorization mechanisms that are used in Grids have specific requirements and APIs that are supported "out-of-the box" in JAVA and C/C++, there is a need for software to enable servers, written in Erlang, to implement these mechanisms. In this work we present our approach for developing grid services using Erlang, ensuring authentication with X.509 certificates and authorization based on the VOMS attributes. We also show testing results for three different implementations.

Keywords
Grid computing, Erlang

1. INTRODUCTION
The Erlang language is a functional language that achieved certain popularity due to its ability to handle concurrent execution of thousands of light-weight processes, thus exploiting the capabilities of modern multi-core architectures. The Open Telecom Platform (OTP) is a de-facto standard distribution of Erlang, which provides tools and libraries and supports particular models for building applications that foster reliability and robustness, for example through the use of supervisor processes that monitor and restart worker processes as needed. The Open Telecom Platform also provides protocal for communication between different nodes on a network running Erlang so that they can form a distributed system. More information about the Erlang language and the OTP platform can be found in [1], [2] and [3]. The computational grids are giant distributed systems utilizing a variety of protocols and services. The computational and storage resources that form Grids are usually heterogeneous as hardware, geographically and institutionally dispersed. In our view the advantages of Erlang may be used for implementing new or old protocols and developing reliable services that use efficiently available hardware. In order to achieve this one must ensure that authentication and authorization mechanisms that are used in Grids are followed.

1.1 Mechanisms for authentication and authorization in production Grids
The computational Grids are an established way of sharing heterogeneous hardware resources that are administratively controlled by multiple institutions accross the network. That is the main difference from Clouds where administrative control is ultimately retained by the Cloud provider. In the context of the European e-Infrastructures for research the computational and data resources of European countries are organized in the European Grid Infrastructure (EGI), which comprises currently of more than 200 clusters and provides access to more than 20 Petabytes of disk storage. More information about this infrastructure can be found in [4]. Because of the size of this infrastructure and the fact that it is of production quality, i.e., it is available and usable 24/7, we concentrated our development efforts on software that can be used on it specifically. The EGI resources are provided to scientists that are members of Virtual Organizations, corresponding to the various scientific domains. Within each Virtual Organization there are roles and groups, which provide a means for finer-grained control and delegation of responsibilities with the Virtual Organization.

In EGI the information about roles and groups in the VOs is maintained in a database accessible through the Virtual Organization Management Server (VOMS). The authentication in EGI as well as in many other Grid infrastructures is ensured through the use of X.509 certificates, issued by Certification Authorities (CA) that are part of International Grid Trust Federation (IGTF). Researchers obtain digital certificates, signed by one of these CA. However, in order to limit the possible harm from a stolen certificate a level of indirection is introduced by utilizing proxy certificates, that are signed by user's certificate. When the users need to prove their membership in a VO they obtain through the use of VOMS proxy certificates that include extensions, digitally signed by VOMS, which certify their VO membership and their groups, roles and capabilities information.

The European Middleware Initiative (EMI) provides software packages that can handle Grid authentication and authorization. The EMI trustmanager can be used to write Java-based clients and servers. The EMI CANL is a module, written in **C**, which can be used to test the authenticity of proxies and certificates. For the authorization part one can use the VOMS APIs, which are also provided by EMI as part of the package `voms-api-c`. Our idea is to interface these two packages with the Erlang runtime environment.

1.2 Background on Erlang
Erlang is a functional language, developed initially by Ericsson, which open-sourced its Open Telecom Platform implementation in order to foster the general adoption of the language. The Erlang runtime environment features a Virtual Machine (VM), which enables spawning and running of lightweight Erlang processes, that are similar to the so-called "green threads" in Java. The common approach in Erlang to utilize efficiently the available cores in the modern multi-core

CPUs is to launch hundreds and even thousands of active processes in the Erlang VM. The immutability of variables and the "share-nothing" approach facilitate good scalability and avoidance of bottlenecks. The pattern-matching features and support for binaries as language objects enable rapid implementation of protocols.

The emphasis on reliability and robustness of Erlang is important for its use in telecommunications. A sophisticated system of supervisors for processes enables restarting of failed server processes or other appropriate reactions to failures. It can be expected that subroutines written in C are a potential source of failures. The preferred way of interfacing with such subroutines is through ports, where the communication happens over the standard input/output. Alternatively, the subroutines may be compiled into functions that can be called direct, but then there is the risk that failures in the C subroutine, for example in memory management, may bring down the Erlang VM, which is otherwise highly resilient.

2. MODIFICATION OF THE SSL IMPLEMENTATION IN ERLANG

The preferred way to implement authentication and authorization in EGI is to use the SSL/TLS. The whole chain of certificates, including any proxy certificates, is checked for consistency by the peer and the authorization information is extracted from there.

The Erlang SSL implementation is available as a module, part of the standard distribution of Erlang. Since the new implementation of SSL is implemented in Erlang, its sources are easy to read and not so difficult to modify in order to add some necessary functionality. In order to implement Grid authorization, there are two important peculiarities. First of all, the certification authorities accepted in Grids are provided as a set of rpms by the IGTF. The certification authorities accepted in Grids also provide Certification Revocation Lists (CRLs). Since the Erlang SSL implementation provides a callback, through the option `verify_fun`, for replacing the default checks with user-defined ones. Thus, we developed a function that in turn calls a port, where the authentication mechanism provided by EMI is invoked. In order to avoid bottlenecks and better use multi-core CPUs, we actually start several Erlang processes, each connected with the corresponding port, and rely on the function get_closest_pid to perform load-balancing among this group of processes.

For the authorization part we found out that we can use the function
`ssl::peercert(Socket)`, which provides access to the X.509 certificate, presented by the peer for an SSL session, but it is not enough for our purposes, because a chain of proxies may be involved and, thus, the VOMS attributes may be available further down the chain, instead of the top. That is why we need to obtain access to the whole chain of certificates. We decided to add it as a field in the SSL socket structure and provide a function called ssl::peerchain that gives access to the whole chain as a list of DER-encoded certificates.

This approach has the drawback that we are patching parts of the Erlang distribution and we are increasing the size of the `sslsocket` structure. However, it has the obvious advantage that is easier to integrate with existing frameworks, since whenever we have access to the socket for an SSL session we can also recover authorization information. Otherwise, we can obtain this information from inside the user-supplied verification function, that is under our control, and forward it somehow. For example, we can associate the chain with the peer certificate in an associative ETS table, but this approach will break if the same certificate may be on top of two different chains. Although this is not a common case, we can not ignore it.

Our tests have shown that the addition of the certificate chain information in `sslsocket` results in decreased performance, but it is acceptable and seems necessary for proper support of the Grid authentication and authorization mechanisms.

3. IMPLEMENTING GRID SERVICES IN ERLANG

In this section we are going to describe how the Grid authentication and authorization mechanisms described above can be integrated to the existing and new frameworks for SSL-secured TCP or HTTP servers in Erlang. From the previous chapter one can conclude that the Grid authentication is performed during the establishment of the SSL session and the authorization information can be obtained on-demand from the `sslsocket` structure associated with a connection.

There exists a plethora of frameworks for development of TCP or HTTP servers in Erlang. One natural way of developing Grid services will be to add the Grid authentication and authorization mechanisms to a framework that can provide HTTP server functionality. In order to test this approach we used `mochi-web` [5], which is a lightweight http server, written in Erlang. In order to enable Grid authentication, we patched the code related to connection establishment to add the necessary options, the main one being the `verify fun`. Since the `sslsocket` structure, associated with a request, is available as a part of the context during the processing of the request, it is straight-forward to add logic for the Grid-aware authorization during the handling of the request. Taking into account that the same connection can be re-used for many requests, one can optimize by computing the VOMS FQAN for a connection only once and adding it to the request structure, but we didn't pursue this line of work further, because it is rather straightforward and we were more interested in the performance when each request corresponds to a new connection.

Another established framework is the so-called `erlang-cowboy`, which works in tandem with the `ranch` socket framework. As a part of the standard distribution of `cowboy` one can find examples of different styles of request processing, including REST and web-services, provided over HTTP or HTTPS. Compared with `mochi-web`, the `cowboy` framework seems more complex, employing a more layered approach to request handling. A certain disadvantage of `cowboy` was the lack of direct access to the `sslsocket` information associated with the SSL session from the user modules. In order to compensate for that we implemented a mechanism that can forward this information to the user modules.

As a third approach we developed a TCP server that is secured with SSL, using the approach outlined in [6]. However, because of using SSL we dropped the parts that were related to ensuring asynchronous operations in the original codes. These parts use some undocumented features of the Erlang TCP implementation that could not be used directly in our work and are less relevant because of the additional SSL layer.

4. TEST RESULTS

Our efforts were concentrated on stress-testing the system to uncover if it breaks under high load. On the client side we used standard, widely used packages and software for testing with security bits disabled since our focus is on the server.

Since all of the frameworks that we modified for providing Grid services have in common that they may be used to provide services over HTTPS protocol, we made them produce simple reply stating the user's Distinguished Name (obtained from the X.509 certificate), Virtual Organization membership and voms information, similar to what can be obtained from `voms-proxy-info` command.

In the sequel we denote by CWR the implementation using the combination of cowboy and ranch for providing web-services, by MOCH the `mochi-web` implementation and by GTCP the generic TCP implementation, which in principle can handle any type of requests over TCP, but in this case was restricted to answer simple HTTPS requests in the same way as the other two are doing it. Each of these frameworks was run on a dedicated server HP ProLiant SL 390S with 96 GB RAM and dual CPU Intel Xeon E5649 @ 2.53GHz. The operating system is Scientific Linux 5.6. In order to make it possible to handle the high number of connection attempts during the tests, which may appear as flooding, we had to issue the commands

```
echo 0 > /proc/sys/net/ipv4/tcp_syncookies
echo 20 > /proc/sys/net/ipv4/tcp_fin_timeout
sysctl -w net.ipv4.netfilter.ip_conntrack_max=240000
```

Traditionally we also have high limits for a number of open files (300000), which were useful in this case. The version of Erlang that we used was R16B, deployed from source. Since we found that enabling kernel polling did not improve the performance, all tests are run with kernel polling disabled. The clients started on 8 blade nodes of our cluster, which are HP ProLiant BL 280c with two processors dual Intel Xeon X5560 @ 2.80GHz and 24 GB of RAM, running Sci-entific Linux 6.3. The interconnect between these nodes is non-blocking Infiniband DDR, running at 20 Gbps. By using the IP-over-Infiniband emulation we had sufficient bandwidth and because of the short messages we didn't observe any network bottlenecks.

We tested different ways to generate the loads. For example, the `tsung` framework is an erlang-based generic framework for stress testing HTTP, LDAP and other types of servers. More information about its usage can be found in [8]. It was capable of generating load for our tests using the client nodes as a part of a in an erlang distributed system. How- ever, the request rates that we observed were lower than what we obtained with the `siege` software. The reason for that is probably the larger number of aborted or refused connections, which still wasted server's resources. One important observation which we draw from our `tsung` testing was the performance gradually decreases with time and after one or two minutes becomes flat. The explanation for that is the gradual increase of the number of transactions that the server cannot handle in the allotted time, which still takes some CPU cycles to process. This problem was more visible when using `tsung` for the testing. We also used it for testing connection reuse, where we found that we have to change some kernel parameter as shown above and observed that other types of errors started to occur due to the high number of open connections, for example "address already in use".

In this paper we present the benchmarking results, obtained with the software `siege`, version 3.0.0, which can be obtained from [7]. We set the certificate and key in `siegerc` to a user's voms proxy certificate and used the options `-b -c 400 -t 120s` launching one `siege` process on each of the blades dedicated for testing. This software was efficient in testing and with the concurrency option of 400 one could achieve results close to the maximum rate of opening of connections even using just one client blade. However, with more nodes the results are slightly higher. We only tested one request per connection, thus putting most load on connection establishment. In all cases we notice that when the load increases substantial portion of the CPU time gets spent in the processing of Grid authentication and authorization information, which is done in separate OS processes. An easy way to all but eliminate this load is to introduce caching in the computations. The VOMS-related information is essentially static unless there is some change in the accepted VOMS servers and VOs, which is quite a rare event. In case of such even a general trigger can invalidate all the cache with only temporary performance impact. Since we are processing hundreds of messages per second in our tests and we used only one client certificate, the impact of caching is high even for 1 second caching. The expiration times of the certificates in the chain are also to be considered, but in most cases these are much more easier to compute than to perform all the file I/O required for full evaluation.

In both testing frameworks the same phenomenon occurs - there is some limit on the number of connections that can be handled without error and after that timeouts start to occur. In most cases the client sees the timeout while it would be better to start aborting connections at the server side after certain threshold has been reached. Since we could implement this option only in the GTCP implementation, where we have better control of the code, we did not pursue this idea further. With the `siege` we noticed much lower error rates, since the load is adapting to the server capabilities.

The results in Table 1 show the rate of connections achieved when testing the three implemented frameworks with `siege`, showing the total request rate achieved from all the clients. In this experiment one connection is used for one request only.

Table 1: Maximum sustained number of new connections and requests per second with caching enabled

Framework	Requests per second
CRW	288
MOCH	223
GTCP	399

In the next table we show how the results change if caching is fully disabled.

Table 2: Maximum sustained number of new connections and requests per second with caching disabled

Framework	Requests per second
CRW	197
MOCH	221
GTCP	279

One important parameter of the Erlang runtime environment that we found to have an impact was the number of asynchronous threads, which we set at 10 (the default value), when no caching is used and 12 for the caching version. The

reason is that the number of physical cores that we have on the server machine is 12 and the computation of authentication and authorization information takes some CPU power.

Our implementation of Grid authentication and authorization also benefits from reusing of the connection for performing many requests. In such case the performance will mostly depend on the HTTP or other server being used.

In general the benefit of caching is non-negligible, but introduces small delay in propagating information about revoked certificates. Since the Certificate Revocation Lists are usually downloaded only once every six hours, we consider this acceptable.

The difference in performance of the three framework seems to be in favor of our GTCP implementation, but this may be due to its relative simplicity. We were satisfied to see that adding Grid authentication and authorization capability to popular generic open-source servers, written in Erlang, can be achieved without lots of effort and with acceptable performance.

5. CONCLUSIONS AND FUTURE WORK

Our implementation of the Grid authentication and authorization mechanisms used in EGI opens up the possibilities of developing reliable, robust and scalable Grid services using the Erlang programming language. The tests show that our implementation can perform even under utilization levels that are rarely found in practice. Our approach was implemented as a plug-in to two popular Erlang frameworks for HTTP servers and in a simpler system that is also usable as a generic TCP server with SSL security. In this way we enable actual application servers to be built by our team or other researchers and we plan to do so, firstly by making some other popular servers, written in Erlang, grid-enabled. The performance can be improved further by some fine-tuning, but we believe we have reached sufficient performance levels and with the capabilities of Erlang to build distributed systems we can scale horizontally as required.

In our future work we plan to develop a clean Grid interface to a mnesia database, as well as one that provides access to purely computational capabilities, using our NVIDIA M2090 GPU cards. The research work reported in the paper is partly supported by the project AComIn "Advanced Computing for Innovation", grant 316087, funded by the FP7 Capacity Programme (Research Potential of Convergence Regions), and by the Bulgarian NSF grant DVCP02/1 CoE Supper CA++.

REFERENCES

[1] Joe Armstrong, Programming Erlang: Software for a Concurrent World, 2007.

[2] Franchesco Cesarini, Simon Thompson, O'Reilly media, Sebastopol, CA, 2009.

[3] Martin Logan, Eric Merritt, Richard Carlsson, Erlang and OTP in Action, 2010

[4] http://www.egi.eu/

[5] http://www.mochi-web.org

[6] Building a Non-blocking TCP server using OTP principles http://www.trapexit.org/Building_a_Non-blocking_TCP_server_using_OTP_principles

[7] http://www.joedog.org/pub/siege/

[8] http://tsung.erlang-projects.org/