A Java-based secure platform for distributed computation

Artém Glebov
Belarusian State University, e-mail: aglebov@acm.org

We present a platform for distributed computation that employs Java RMI as the underlying communication technology, making use of its remote class-loading features. The platform permits dynamic connection of executor nodes and guarantees that executors are properly released. Security is reinforced by authentication based on X.509 certificates and SSL encryption. We show that the overhead associated with authentication and encryption is negligible in practically important cases.

Keywords and phrases: distributed computations, Java RMI, remote method invocation security, authentication and encryption overhead.

Introduction
The volatile character of modern computer networks and the risks of unauthorized access require that fault-tolerance, authentication and confidentiality be ensured. These requirements are commonly overlooked by the creators of distributed computation toolkits since these toolkits are designed to work in controlled environments. However, as the idea of volunteer-based computation gains popularity, more and more computation will be done over uncontrolled open networks, such as the Internet, which will demand computational systems compliant with the aforementioned requirements. The goal of this work is to propose a simple platform for distributed computation that conforms to the needs of open networks.

In particular, development of the proposed platform was motivated by the needs of parallel genetic algorithms. Despite their computational ingenuity, genetic algorithms (GAs) often require a significant amount of time to find a satisfactory solution. This is especially the case if the evaluation of the objective function is based on imitative modeling, a typical situation in many engineering problems. Parallelization of the algorithm becomes a natural approach in this case, accomplished by delegating the calculation of fitness to slave machines, while maintaining the core GA cycle on the master computer.

A number of works [1–7] have been devoted to developing libraries for parallel GAs. These libraries either have limited portability [1–2], which makes them unsuitable for use in heterogeneous computer networks, or prove to be insufficiently flexible [3–4]. For instance, the socket-based implementation used in JGAP requires that the nodes be run in a specific order, and an unexpected shutdown of any node may make the whole algorithm crash. Consequently, this implementation does not meet the requirements of open networks.

Several researchers [5–7] independently came up with the idea of employing the Java RMI technology for distributed computations to take advantage of its proven reliability and flexibility. Maciura and Setlak [5] presented a rather narrow implementation of the island model that lacked configuration opportunities. A more powerful framework was introduced in [6]; however, its use is also restricted to the island model of parallel GAs. A recent paper by Strykeleu [7] suggested using the dynamic class-loading features of Java RMI to execute custom client tasks remotely.

Approach
As in the well-known master-slave scheme, our platform assumes a single manager node and several executor nodes. The key difference is that the manager node is responsible only for registering and managing executors
A Java-based secure platform for distributed computation

while the client issues tasks and controls the execution process.

After startup, each executor registers with the manager by calling a remote method and passing it a proxy object of the management interface, which is later used by the manager to control the executor’s state.

Upon a request from the client demanding a certain number of executors, the manager generates a random long integer that will serve as a ticket permitting the client’s access to the assigned executors. The manager selects the requested number of executors and notifies them by passing them the generated ticket. During the course of this process, the manager also tests the availability of the executors and updates the list of available nodes accordingly. Thus, every executor is tested before it is assigned to a client, which reduces the likelihood of providing the client with down executors.

The executors assigned to a certain client are marked busy and not considered for reservation again until the client releases them by calling the respective remote method on the manager, and passing it the corresponding ticket. In order to avoid a situation in which a client disconnects without having released the executors, we require that each client periodically prolong its reservation; otherwise, the manager orders the executors to reset the ticket and cancel the client’s tasks.

In order to limit the number of executors that a single client may reserve, we implement an auxiliary authentication service whose only purpose is to tell the client its unique identifier, if the client has been successfully authenticated in the course of SSL handshake. The client then uses the identifier for reservations.

The key features of our platform that distinguish it from its analogues are the following:

- adaptiveness, which permits dynamic connection and disconnection of executors;
- a guarantee that executors are released when they are no longer used;
- a use of authentication and encryption.

Experimental results

Experimental investigation of the proposed platform was undertaken in order to determine the performance penalty associated with authentication and encryption. All experiments were conducted in a laboratory consisting of 14 computers with 2.3 GHz Athlon 64 X2 processors and 2048 Mb of memory connected in a LAN with DLink DES-1024D 10/10 Fast Ethernet Switch. To eliminate the effect of fluctuations, for each combination of parameters ten measurements were made and average values were calculated.

The first test aimed to determine the speedup (i.e. the time rate of sequential to parallel execution) achieved by parallel execution of a certain amount of tasks without authentication and encryption. The number of tasks in a batch was chosen equal to 101 in order to avoid the undesired effects arising when the number of executors divides the number of tasks evenly. Each task consisted of 20 Kb of payload and contained a single wait instruction (1 s), included to simulate the length of computation. The results are shown in Fig. 1.

The speedup values achieved are close to the theoretical maximum values. The deviation is less than 9% when using one thread and is more significant (up to 20%) for two threads, which can be attributed to the burden of additional synchronization and internal distribution of tasks between the threads.

The effect of introducing authentication and encryption was studied in the next series of experiments. Figure 2 presents the dependency of the speedup on the execution time of a single task.

The overhead associated with the use of SSL comes from two sources (leading to the presence of the average and the worst cases):

- symmetric bulk encryption of messages sent between parties, which is fast and does not degrade the performance significantly;
- authentication involving validation of certificates, based on slower asymmetric encryption methods and causing a significant drop in speed.

However, as the Fig. 2 suggests, when the task execution time reaches 5 s the speedup approaches the values achieved without encryption and authentication. Thus, for times above 5 s the use of SSL does not cause a serious
deterioration of performance, making it possible to use authentication and encryption in remote simulation applications, where the typical task execution time is 30 s and more.

Conclusion

Our experimental results suggest that the overhead due to encryption is negligible compared to the typical amount of time required by imitational modelling, which means that the proposed platform may be used in place of existing computational systems. The flexible design of the platform permits its application to a wide range of distributed algorithms. Finally, our platform will hopefully promote the idea of secure and reliable computation in open networks.

References