Distributed Simulations of Physical Phenomena Based on Windows Machines

Elvis Popović, Nikola Ivković*
Faculty of Organization and Informatics, University of Zagreb, Pavlinska 2, HR 42000 Varaždin, Croatia.

* Corresponding author. Tel.: +385 42 390872; email: nikola.ivkovic@foi.hr
Manuscript submitted October 15, 2017; accepted January 8, 2018.
doi: 10.17706/jcp.13.8.963-970

Abstract: Simulating physical phenomena can be computationally very demanding, making the distributed approach attractive. Scientific computing and other computationally complex tasks are often performed on dedicated cluster or grid of computers usually running some distribution of Linux operating system. In this article we propose a simulation framework that can utilize general purpose computers running Windows operating system to share available computational resources. The proposed system is composed of simulation client, the network of distributors, and workers. The presented framework has architecture and design that provides a flexible and efficient system with good scalability and fault tolerance properties. The performed tests on the prototype application show that our approach is viable.

Key words: Distributed architecture, distributed simulation, network application, non-blocking socket, physical simulation.

1. Introduction

Computers were used for simulations in physics from early beginnings of electronic computers. There are many reasons for simulating physics phenomena. By using simulations it is possible to predict the future events and the example that is known to almost everyone is weather forecast. Simulations are also used for reconstructing some events that already occurred but some details or circumstances related to that event remained unknown, or to find what could be done differently to change the course of action. It is also possible to obtain some new knowledge by using simulation and simulations can play important role in education.

A problem with simulation of physical phenomena is associated with the mathematical complexity of models. Many models are such that simple exact solutions cannot be obtained and computationally intensive numerical methods are a reasonable approach. Often it is necessary to solve partial differential equations, perform integration, find roots of a nonlinear equation or solve a system of nonlinear equations etc. These systems are sometimes susceptible to phenomena from chaotic theory and controlling errors is computationally very demanding.

To overcome the computationally intensive calculations in physical simulations and other forms of scientific computing various forms of concurrent computing is used. Often simulations of physical phenomena are performed on dedicated supercomputers, cluster of computers or dedicated grid of computers. Machines in these clusters and grids usually have multicore processors and run some distribution of Linux operating system.

Some projects use volunteer computing where distributed software is installed on volunteers' personal computers and use part of their computational resources (CPU, memory, etc.). Project SETI@home [1] is based at UC Berkeley and uses volunteer computing to analyze radio telescope data in a search for extraterrestrial intelligence. University of Wisconsin–Milwaukee and Max Planck Institute use volunteer computing in Einstein@Home [2] to analyze radio signals and gravitational wave data in a search for pulsars. In the field of particle physics the European Organization for Nuclear Research (CERN) has projects ATLAS@Home [3] and several projects in the group of LHC@home [4].

Our objective is to design simulation environment to allow simple adding of simulation tasks into distributed system based on Windows operating systems to efficiently use available computational resources. The intended system should be flexible and efficient, simple to use and to provide good scalability and fault tolerance.

There are many interesting scenarios where such simulation environment would be useful including simulations performed for high school education and faculty laboratories, hobbyists that are willing to share computational resources or real engineering and scientific applications.

2. Application Requirements

Basically the proposed application should be comprised of three parts shown in Fig. 1: a simulation client, distributed task and resource management layer (DTRML) and available computational resource layer (ACRL). In this system a simulation client would initiate some distributed simulation, collect the results and present them to the user. This simulation would be performed on Windows machines inside available computational resources. The simulation client communicates only with DTRML that is responsible to efficiently use available computational resources. Distributed tasks and resource management layer is also responsible for fault recovery, storing data about performed simulation, system configuration, and the current state of the system. These data are stored in available computational resources.

DTRML is used to hide, from the simulation client, all the complexity of dealing with a network of computers that need to be managed to cooperatively solve hard computational problems. It has to provide scalability thus it is possible to use, not only a system with a few working computers but also a system with a very large number of machines. In the case that some machines from ACRL become unresponsive their subtasks have to be rescheduled to other computers from available computational resources. Sharing available computational resources to perform multiple simulations required by different users is desirable. It should be possible to add additional simulations while other simulations are already running.

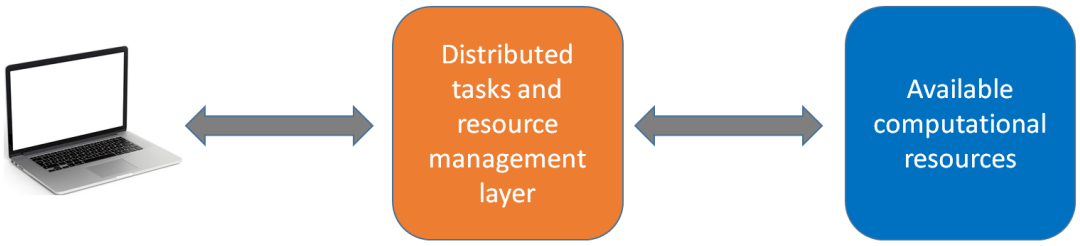

Fig. 1. Organization of the proposed simulation framework.

3. Architecture of Distributed System

The architecture of distributed system for simulation of physical phenomena is presented in Fig. 2. There are many workers running on Windows machines in available computational resource layer that can be used for running simulations. DTRML can have one or many distributors. Fig. 2. shows default configuration

with few distributors (D_1 ... D_m) – forming a sort of a peer-to-peer network where for particular simulation one distributor is the primary distributor and the others are secondary distributors, although other configurations are also possible. The workers are connected in the way that each worker has transport layer connection to one distributor that is the home distributor for that worker. The simulation client is connected with transport layer connection only to his primary distributor.

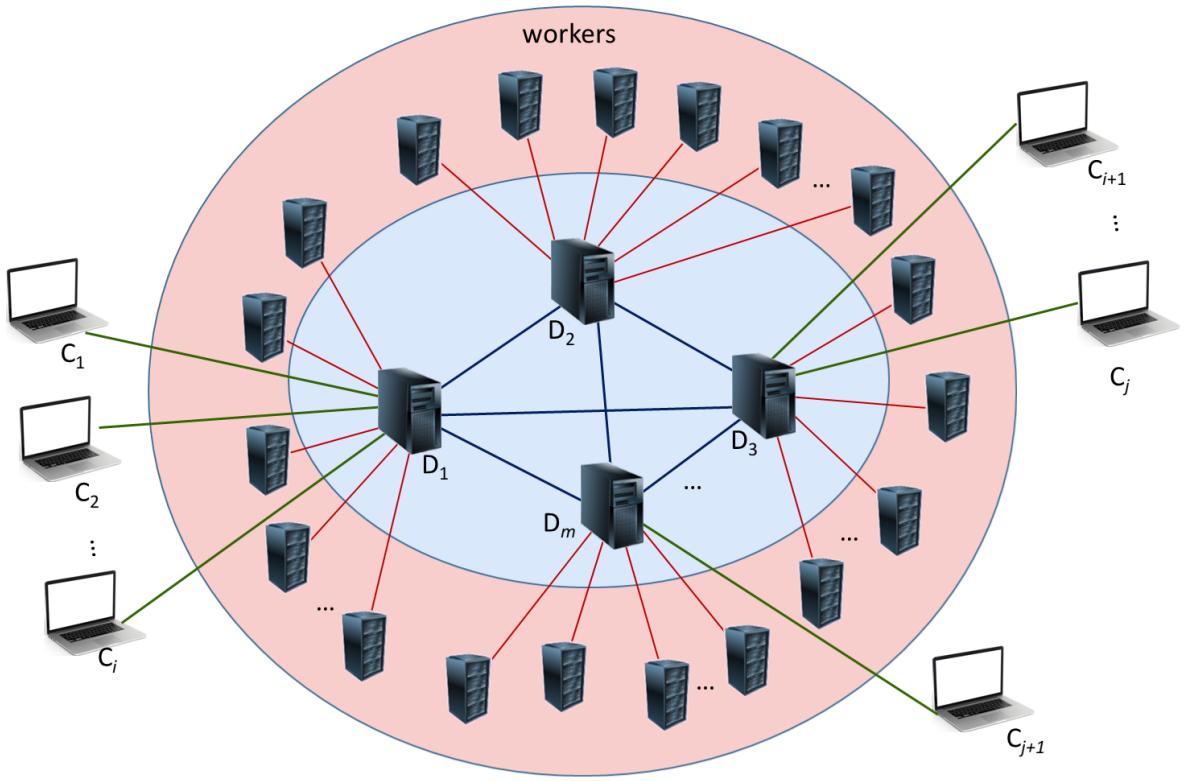

Fig. 2. Three-layer architecture of the proposed distributed system.

3.1. Workers

The workers are computer processes on top of Windows operating system that are part of the distributed system. After the start up, the worker process makes TCP connection toward a distributor and initiate application level communication The list of potential distributors is loaded from a configuration file, entered manually or fetched/updated from an online source. If one distributor is not available, then the worker process can try with another distributor until it finds its home distributor.

When the connection between the worker process and its home distributor is established, the worker process sends READY_TO_WORK message. This message can have additional attributes about processing capabilities of working machine like a number of thread processing cores or current and maximal allowed CPUs load. An important attribute is the cryptographic hash value of loaded binary executable file intended to perform subtask of the simulation (if such file is already loaded into worker). At the beginning the worker does not have the loaded binary executable file which will be provided by its home distributor, directly or by link, along with input parameters for the subtask. After the worker process does same processing and creates results of the subtask, it sends the message to the distributor with output data.

When the results become available the subtask might finish but the binary executable file remains loaded. Then the worker process sends READY_TO_WORK message to distributer in order to get input parameters for another subtask and starts the cycle again.

The worker machines can become available or unavailable in the middle of distributed simulation. Similarly the transport connections can be broken. These issues have to be considered and appropriately addressed in order to create fault tolerant system that can remain to execute the distributed simulation.

3.2. Distributors

The distributed task and resource management layer (DTRML) is composed by one or many distributors. One distributor might be enough to perform distributed simulations in simple scenarios, but normally multiple distributors promote better scalability, availability, and fault tolerance. Each distributor has to be capable of handling many simultaneous transport layer connections. One distributor should be able to handle all transport connections when there are smaller or moderate number of available workers, but additional distributors can increase availability and fault tolerance. Multiple distributors might be necessary for a large number of workers and they can form a two-level hierarchy where the primary distributor is on the first level and all the other (secondary) distributors are on the second level. This way if there are in total $w = n \times m$ worker machines, n worker machines per each distributor, and m distributors, then the number of transport layer connection for particular distributor stays $O(n + m)$ in contrast to $O(n \times m) = O(w)$ which would be required if all the worker machines are connected to only one distributor. For a large number of workers w, the number of distributors can be chosen as $m = \sqrt{w}$ to minimize the number of connections to $O(\sqrt{w}) = O(m)$. For an extremely large number of worker machines, if $O(\sqrt{w})$ becomes too large for a distributor to handle, the distributors could be organized into a tree structure where the primary distributor is the root of the tree.

The simulation clients and workers initiate transport layer connections toward the distributors. After the connections are successfully established, the distributors become the primary distributors for the simulation clients and the home distributors for workers. Upon receiving a request from a worker or a simulation client, the distributor may send REDIRECT message with the address of another distributor in order to perform load balancing or because the distributor is scheduled for shutdown due to planned maintenance and only waits for running simulations to finish. Distributors also communicate with each other to advertise the available workers that are connected to them.

3.3. Simulation Clients

The simulation client starts the simulation. It has to divide simulation on as much as possible subtasks, which are sent to the primary distributor to be relayed to the workers. The results of subtasks need to be combined into total simulation results and presented to the user by the simulation client. The simulation client does not deal with workers directly not only because of scalability, fault tolerance, and availability problems, but also because the simulation client and workers usually do not have permanent IP address and often are on the private address side of network address translation (NAT) devices [5], [6]. In the case when the simulation has to be safeguarded against deliberate false results of subtasks reported by workers, e.g. due to pranks or malicious intentions, security mechanisms need to be employed by simulation clients.

4. Design Choices and Implementation Details

4.1. Efficiency and Flexibility

In order to use computational and communicational resources in an efficient way it is necessary to take special care in designing all parts of the system, the total architecture and particular implementations of distributed simulation. The goal is to minimize overall execution time for particular simulations, to minimize energy consumption, and to leave as much as possible computational (CPU, memory, storage) and communicational (available throughput, small queuing delays) resources to other users or tasks.

It is not possible to reinstall the workers' software from the fresh for every new simulation because of flexibility requirements. Very high flexibility would be gained if workers would dynamically interpret simulation subtasks written in some high level programming language, but the efficiency of such solution would be very low, especially because the simulations of physical phenomena are almost always CPU intensive. To satisfy simultaneously efficiency and flexibility requirements, the software on worker machines should consist of optimized preinstalled binary machine code and dynamically loaded binary executable code. Since the worker machines are running Windows operating systems, the dynamic-link libraries (DLLs) seems the best possible choice [7]-[9].

In most cases the workers are expected to run on multicore computers thus to utilize available processing power there are few design choices. One is to allow multiple simulation subtasks on the same machine either by having multiple instances of workers that run in separate processes or one multithreaded worker process. The separate process instances, where each process run exactly one simulation subtask, have an advantage of simplicity, security, and natural fault tolerance but a downside is larger overhead. One process with multiple threads, each thread per simulation subtask, spends fewer resources and can share TCP connection(s) to promote scalability by lowering a number of connections toward home distributors. But this approach is more complex to implement correctly, more prone to bugs and failures where one thread could potentially bring the whole worker process down. The other possibility is to make worker process as simple as possible by using only one singlethreaded worker process per worker's machine and let the simulation subtasks use multithreaded algorithms to take advantage of multicore machines. This approach has few drawbacks and one is that this complicates the design of simulations that are already distributed (which is not trivial to accomplish) by imposing multithreading on workers which are heterogeneous (e.g. different worker machines have a different number of available CPU cores). It is also easier to parallelize worker algorithm once rather than parallelizing each distributed simulation that will use these workers. This design does not allow various distributed simulations to share the same worker machine. Of course, running multiple subtasks (possibly from different simulations) on one machine does not restrict simulation subtask to be singlethreaded and balancing with a number of threads is not too hard to achieve.

4.2. Scalability

A distributed simulation framework should ensure that simulation is efficiently executed whether there is a few available computers or very large number of computers. By the architecture design, the simulation client is insensitive to a number of workers and all the communication burden is put on DTRML. The internal architecture of DTRML, as explained in Section 3.2, allows sublinear growth of communication complexity regarding the number of workers. The simple design of distributors, where for each TCP connection there is one thread with blocking socket functions, have some limitations and might not scale well to hundreds of TCP connections. The classical nonblocking designs with select() and poll() functions would allow much better scalability of distributors. But for very large number of connections, e.g. 10000 connections [10], this approach could become inefficient considering that complexity of this approach is $O(t)$, where t is the number of TCP connections. The greatest scalability can be achieved by specialized mechanisms like epoll (provided by Linux kernel) [11], kqueue (available on OS X) [12] and Input/output completion ports [13].

Input/output completion port (IOCP) is implemented in Windows kernel and exposed by WinAPI [14]. The complexity of this approach is O(1) making it very viable for use in distributors. Fig. 3 shows a sequence of events in the case of using IOCP with two working threads, two sockets, and two clients. This is an asynchronous technique that uses FIFO and LIFO queues and all the messages are processed by a pool of threads. The number of threads in the pool should be such that all the processing cores can be used, but at

the same time it should not be too large to cause too many context switches. A suitable number of threads could, for example, be a twice the number of processing cores.

4.3. Fault Tolerance

The distributed simulation system is composed of many computers and transport layer connections that might fail without warning. For long-time running distributed simulations and with the increased number of computers this issue becomes more significant. The simulation client can shut down unexpectedly, e.g. due to the power loss, therefore the state of the running simulation is backed up on a disk. After the reboot, the simulation client reconstructs simulation state and tries to re-establish TCP connection(s) with the primary distributor. If the primary distributor is not available, then the simulation client tries with another distributor. A similar but little simpler recovery happens when TCP connection(s) breaks between worker and the distributor.

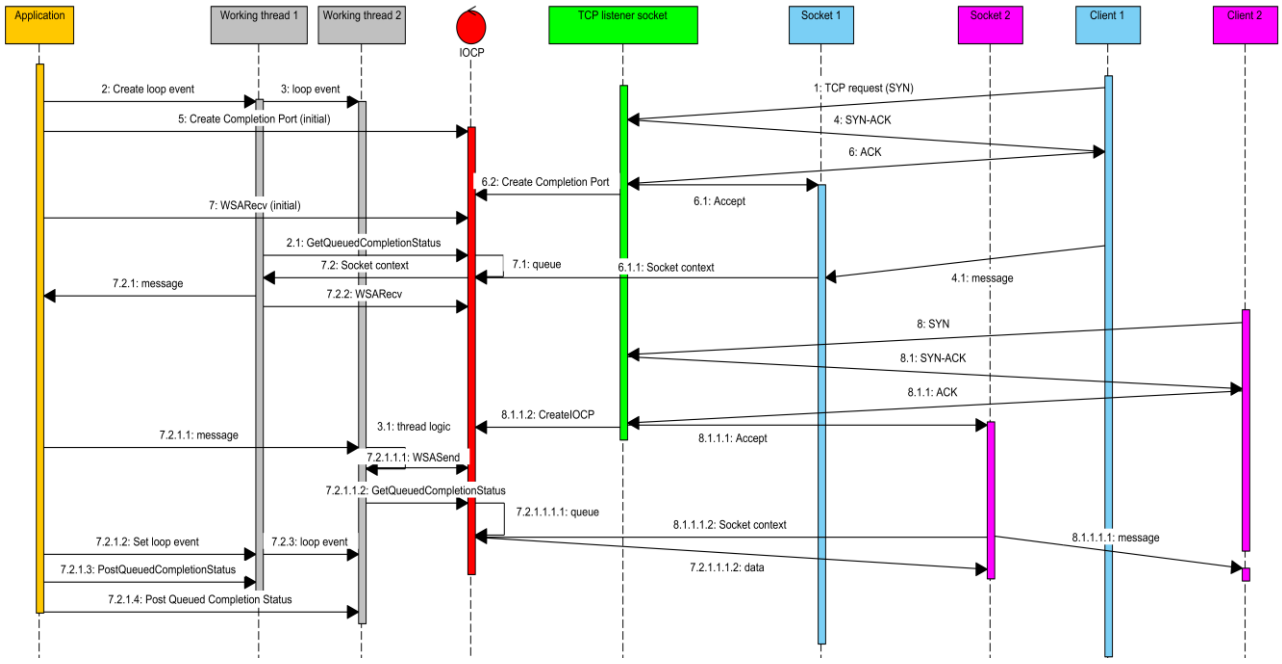

Fig. 3. Asynchronous overlapped communication over two sockets with ICOP mechanism.

In order to uniquely identify running simulation it is necessary to use the identifiers, which generates the simulation client by using a hash function like SHA functions [15] on subtask DLLs, public IP address, port number and the starting time. While the simulation client is unavailable, the distributor keeps communicating with workers and stores intermediate results until the simulation distributor becomes available.

When distributor becomes unavailable workers can continue to operate until they finish with subtasks. Then if the worker cannot reestablish the connection with its home distributor it will try to connect with other distributor and report the results together with simulation ID and the subtask instance ID. The simulation client waits for some time and tries to reestablish the connection with its primary distributor, but after a predefined number of tryouts and elapsed time, the client simulation will find another distributor that will become the primary distributor. A worker can become unavailable temporarily or permanently and the simulation client can reschedule the subtask to another worker. Depending on simulation client preference it is possible to have each subtask run with two copies on different workers making it much more unlikely that both instances of subtask will fail.

4.4. Security

When the distributed simulations are performed in a trusted environment, e.g. in the faculty laboratory for the purpose of teaching and learning, there might not be necessary to use special security measures. To prevent pranks or malicious attacks versatile security measures can be applied. Running the same instance of subtask on different workers allow the simulation client to detect false results. Attacks on distributors can have greater impact thus distributors have to be specially guarded with standard security measures for servers. Cryptography can be used to authenticate and authorize users and devices and possibly to make communication private and to preserve integrity. Well-known protocols like TLS can be used for communication between the simulation clients and distributors and also for communication among distributors.

5. Conclusion

A framework consisting of simulation client, the network of distributors, and workers running Windows OS for performing distributed simulations of physical phenomena is proposed in this paper. The system is designed to be flexible, efficient, and to promote scalability and fault tolerance. After running two distributed simulations, calculations of Julia sets that exhibit chaotic behavior and phase portrait of simple gravity pendulum by using fourth-order Runge-Kutta method, our approach proved to be viable. In the future work we plan to test our framework with other distributed simulations like magnetic field of permanent magnets, electrostatic fields with initial-boundary conditions, Kármán vortex streets from fluid dynamics, etc. The proposed framework can be used for other complex calculations other than simulations of physical phenomena but for some of such applications that can benefit from space-time trade, by caching some intermediate calculations, certain smaller changes in the presented framework should be made.

References

[1] SETI@Home. (2017). Retrieved from https://setiathome.berkeley.edu

[2] Einstein@Home. (2017). Retrieved from https://einsteinathome.org

[3] Aad, G., *et al*. (2008). The ATLAS experiment at the CERN large hadron collider. *Journal of Instrumentation*, *3(8).*

[4] LHC@Home: Volunteer computing for the LHC. (2017). Retrieved from http://lhcathome.web.cern.ch

[5] Srisuresh, P., & Egevang, K. (2001). Traditional IP network address translator (traditional NAT), *RFC 3022*, 2001.

[6] Zhang, L. (2997). A retrospective view of NAT. *The IETF Journal*, *3(2),* 2007.

[7] Hart, J. M. (2010). *Windows System Programming* (4th ed.). Addison-Wesley Professional.

[8] What is a DLL? (2017). Retrieved from https://support.microsoft.com/en-us/help/815065/what-is-a-dll

[9] Dynamic-Link Libraries. (2017). Retrieved from website https://msdn.microsoft.com/en-us/library/ms682589.aspx

[10] Kegel, D. (2014). *The C10K Problem*. Retrieved from http://www.kegel.com/c10k.html

[11] Love, R. (2007). Linux system programming. *O'Reilly,* 83-94.

[12] Singh, A. (2006). Mac OS X internals: A systems approach. *Pearson Education,* 1191-1196.

[13] Lucovsky, M. H., Vert, J. D., Cutler, D. N., Havens, D. E., & Wood, S. R. G. K. (2001). Input/output completion port queue data structures and methods for using same. *US Patent 6, 223, 207, 24.*

[14] Russinovich, M., Solomon, D. A., & Ionescu, A. (2012). *Windows Internals Part 2* (6th ed.).

[15] Chaves, R., Sousa, L., Sklavos, N., Fournaris, A. P., Kalogeridou, G., Kitsos, P., & Sheikh, F. (2016). Secure hashing: SHA-1, SHA-2, and SHA-3. *Circuits and Systems for Security and Privacy*.

Elvis Popović was born in Zagreb, Croatia. He received a bachelor of informatics degree from Faculty of Organization and Informatics, University of Zagreb where he is currently perusing a master degree in information and software engineering. Previously he has studied physics on Faculty of science, University of Zagreb. His professional experience is mostly in software engineering, 2D and 3D computer graphics and algorithms for visualization in science and physical process, three-dimensional and multidimensional Euclidian and non-Euclidian geometry. His research interests are in computer networks, distributed systems, and low-level algorithms.

Nikola Ivković received the M.S degree in computer engineering and the Ph.D degree in computer science from the Faculty of Electrical Engineering and Computing, University of Zagreb. He is a member of the research and teaching staff at the Department of Information Technologies and Computing of the Faculty of Organization and Informatics, University of Zagreb. His research interests include computer networks, formal methods, computational intelligence and optimization, operating systems, and parallel programming.