Cluster Computing Framework Based on Transparent Parallelizing Technology

Viktor Burdeinyi

This paper is devoted to the transparent parallelizing technology and describes the problems that arise while implementing a cluster computing framework based on this technology. These problems are discussed and possible approaches for solving them are given. Architecture of such framework is shown and functionality of its parts is described. Efficiency of the implemented framework is shown by parallelizing a sample problem and running tests on a cluster of 1, 2, 3, 5 and 10 computers.

Introduction

Parallel computing is the subject of many researches nowadays [1], which is caused by big number of computational problems that cannot be solved fast enough on single processor (for instance, those are methods of identification of models of nonlinear dynamic systems, based on Volterra series [2]). Also modern processors are often multi-core, so applications have to use parallel computing to utilize all power of modern processors.

There are many approaches used for parallel applications development, such as manual parallelizing (for example, by enhancing traditional imperative programming languages with communication libraries), automatic and semi-automatic parallelizing. The technology of transparent parallelizing has been proposed in earlier papers. This paper is devoted to implementing it as a cluster computing framework.

Transparent parallelizing technology

Transparent parallelizing technology is based on splitting of parallel algorithms and the means of their parallelizing. Its main idea is finding large groups of algorithms that can be parallelized in similar way, introducing template of parallel algorithm and implementing parallel version of the template. It has to be done only once, so we can implement some advanced features in such parallel algorithm, such as handling communication failures, monitoring tools, tools for adding computers to cluster while computations are in progress and removing them and so on. Once such parallel template is implemented we can run any algorithm that fits the template.

One algorithm template has been implemented. It is based on two principles. Let's assume that we have selected some procedures in the program. Each procedure should pass parameters by value and should not modify any data during execution except values of parameters and temporary data structures.

The first principle introduces the concept of an order as the minimal unit of work that should be executed on one computer and cannot be split into smaller parts. Such a unit of work is defined as execution of one procedure without execution of procedures it calls. Each procedure call creates a new order that should be executed by some computer of cluster (let's call such call making of order). One of selected procedures should be marked as main one to define program entry point.

A lot of algorithms contain intervals of time between moments of getting some values computed and moments of first usage of these values. If we perform procedure call in common programming languages, caller procedure continues its execution only after called one is over. We can say that caller procedure starts waiting for output parameters of called procedure in the moment of call and stops waiting in the moment when called procedure finishes its execution. The second principle proposes to continue execution of caller procedure after a call and to start waiting only in the moment of first request to output parameters of called procedure. If called

procedure execution is already over in the moment of such request we should not start waiting at all.

Please refer to [3] for more information about transparent parallelizing technology.

Creating parallel computing framework

Architecture of the framework implementing the technology is shown on figure 1.

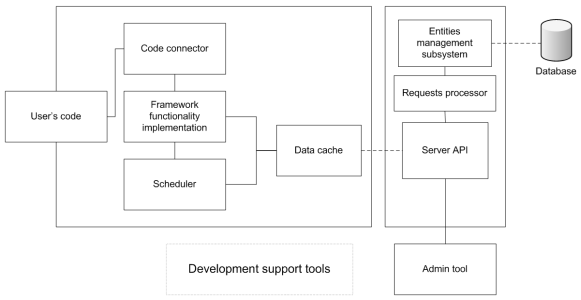

Figure 1. Architecture of the framework

It has been implemented on Java and consists of server part responsible for storing intermediate data and providing tasks to clients and clients that are responsible for performing computations. The framework works with the following entities:

- Data identifier. It identifies a value of input or output parameter of a method. Actual value of the parameter can be obtained by the identifier. If some data identifier is used in multiple calls we can avoid sending actual value of parameter multiple times as it can be cached on clients.
- Unready data identifier. Procedures can use it as a "future" value that will be bound with data identifier or other unready data identifier later. Each time when a new order is created an unready data identifier is created for each of its output parameters. Parent order puts these identifiers in its local parameters and child order binds these identifiers after its execution is over.
- Order. It is a formal description of an order that includes information about method that should be executed, its input parameters (data identifiers or unready data identifiers) and output parameters (unready data identifiers).

Code, written by user, interacts only with the "code connector" layer. This layer implements semantics of orders and their parameters. On one hand, it provides user code with a small set of methods of framework (creating an order, obtaining data by unready data identifier and some utility methods); on the other hand, it represents user code to the rest of framework as a "black box" that can execute orders. Functionality for accessing data handlers is provided indirectly by the means of DataHandler class. Classes of parameters of orders should meet the following requirements: they should extend DataHandler class; they should implement methods for saving data to stream and restoring data from stream (standard mechanism of serialization cannot be used as it creates new objects during deserialization and cannot update existing objects in place); they should provide access to their content only by the means of getters and setters that should call inherited methods preRead(), preWrite() and preReplace() before accessing data.

Framework functionality implements logic of threads management and provides functionality for management of data identifiers, unready data identifiers and orders, remote logging, accessing remote files and reporting failures to code connector. The scheduler is a block responsible for choosing orders to be loaded from server or a ready order to be woken up. At the moment there's only one scheduler based on naive planning heuristic that prefers continuation of execution of ready order to getting a new one from server. The data cache stores information about data identifiers and unready data identifiers to reduce traffic between client and server.

Communication between server and client is implemented by the means of RMI technology. The administrative tool provides functionality for creating orders, viewing their status and accessing computations result. Requests processor and entities management subsystem implement processing of requests from clients and administrative tool and provide methods that are used for scheduling. Data can be stored either in database or in memory.

The framework also includes a tool that generates classes for sending orders according to description provided by user.

Testing of the framework

A solution of the problem of determination of diagnostic value of formed diagnostic features has been implemented for testing the efficiency of created framework. It has been run on clusters with Intel Pentium 1.7 GHz processors, connected via Fast Ethernet, for on clusters of different sizes for problems dimensions 23, 24 and 25. The dependence of execution time from the problem dimension and the number of computers is shown on figure 2. Result of multiplication of execution time by number of processors grows by not more than 1.13% when using 2, 3 or 5 computers instead of one, and by not more than 3.25% when using 10 computers instead of one.

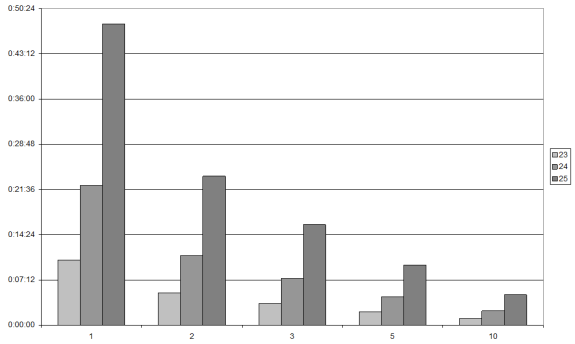

Figure 2. Results of experimental testing of efficiency of framework

Conclusion

This paper describes some problems that arise while implementing a framework that implements transparent parallelizing technology and proposes ways to solve them. Such framework has been developed and it consists of server part, a client part, administrative tool and tools for development support. Its efficiency has been shown by solving a sample problem.

References

[1] Voyevodin V.V., Voyevodin Vl.V., Parallel computations, Saint Petersburg: BHV-Petersburg, 2002 (in Russian)

[2] Kolding T. E., Larsen T., High Order Volterra Series Analysis Using Parallel Computing, http://citeseer.ist.psu.edu/242948.html

[3] Pavlenko V.D., Burdejnyj V.V., Cluster Computing Using Orders Based Transparent Parallelizing, Proceedings 6th International Conference Information Systems Technology and its Applications ISTA'2007, Lecture Notes in Informatics (LNI), Series of the Gesellschaft fur Informatik (GI), Kharkiv, Ukraine, May 23-25, 2007, Vol. P-107, Bonn 2007, pp.152-163.

Authors

Viktor Viktorovych Burdeinyi — postgraduate student, computer-based control systems faculty, Odessa National Polytechnic University, Odessa, Ukraine; E-mail: *vburdejny@gmail.com*