Distributed Simulation Environment for Heterogeneous Computer Clusters

Vadim Engelson, Peter Fritzson
Dept. of Computer Science, Linköpings Universitet, S-58183, Linköping, Sweden,
Tel.: +46 13 281979, Fax: +46 13 284499, vaden@ida.liu.se, petfr@ida.liu.se,
http://www.ida.liu.se/~modelica

1. ABSTRACT

Computational resources can be used with higher efficiency if the CPU load is distributed on many computers. A flexible system for task distribution has been designed. Multiple tasks are organized into a queue, and when possible, are sent for execution to idle computers on the network. The CORBA standard is used for control and data sending over the network. An API for the system makes possible to use the system from a program or attach arbitrary driving system with user interface.

2. INTRODUCTION

Optimisation of model parameters is a major goal of many simulation activities. Often it assumes that large amount of simulation cases with slightly varying input parameters are prepared, performed and analysed. Usually, many of these simulation cases can be performed independently, in a parallel fashion.

In many organizations average CPU use on personal workstations is less than one percent, since the computers are not used for computations. Many computers are used just for text editing, web browsing or other "slow" operations that do not occupy CPU. Some computers stay idle most of the non-office time since nobody is logged in. Cluster computing is emerging research and development area that includes methods and tools for efficient utilizing of idle CPUs of multiple computers connected in a high performance local area network.

Computational resources necessary for a single simulation vary from one second on a customary workstation to several days on a parallel supercomputer. Using computational power of supercomputers is not affordable for many cases; therefore a cheaper alternative in form of clusters of ordinary computers is used. These computers can be specially dedicated for high performance simulation (e.g. Beowulf Linux clusters, a collection of 10-100 computers with extremely high communication performance, suitable for parallel programming). The cheapest solution is to use existing set-up of computers in a local area network, where many independent processes can be coordinated and distributed. In this case cluster computing shares CPU with normal use of workstations, and preference should be given for the latter.

3. DIFFERENCE FROM OTHER APPROACHES

There are several commercial and research systems for cluster computations for the UNIX world. The review /4/ written in 1996 explains the differences between several alternatives. Some systems allow using Windows-based clusters only /7/. Just one system /8, 12/ operates on mixed heterogeneous clusters where both Windows and UNIX workstations participate. However capabilities of Windows version are limited. The main reason is that Windows has not been designed for remote control. Also, capabilities for process monitoring, terminating and inter-process communications are primarily developed for UNIX.

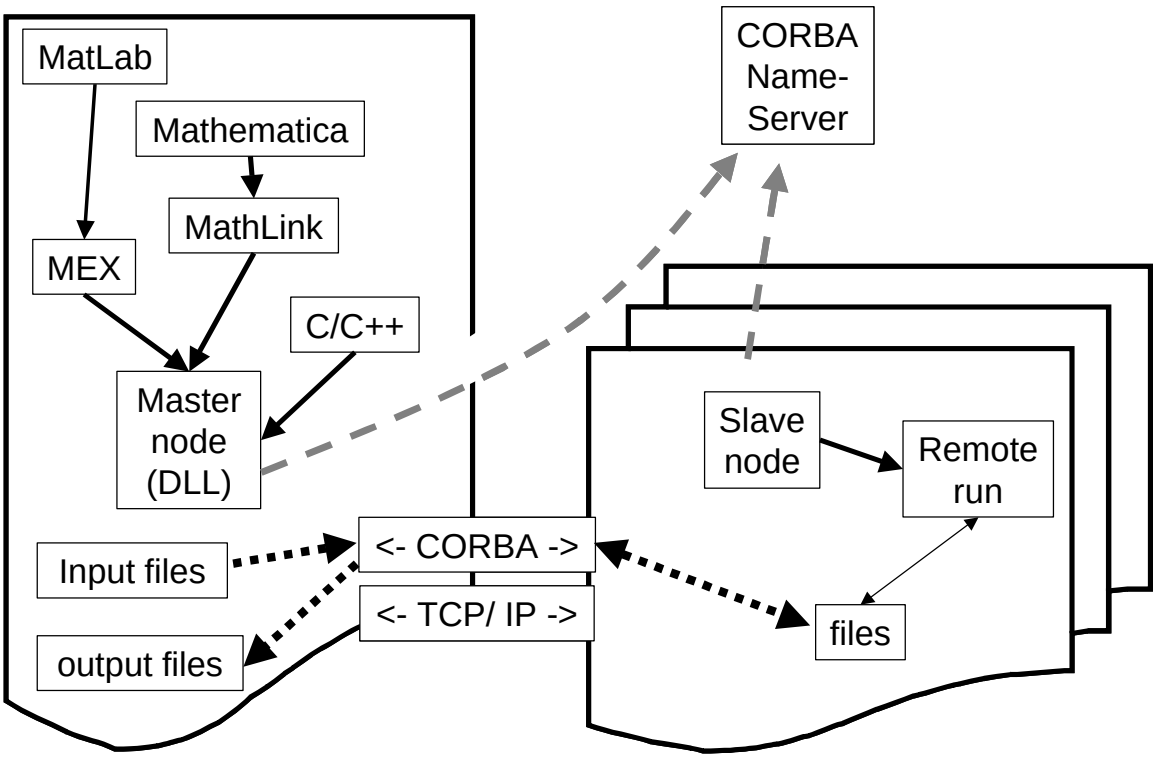
On many sites only large amount of Windows-based computers are available. Often multiple Linux, Sun and Windows workstations are connected to the network. We develop a system that is able to run on such heterogeneous clusters of computers.

Existing commercial and research system have no flexible, interactive and dynamic control over set of simulations. Typically a cluster configuration file as well as all necessary data should be supplied to the system before the simulations start. The results can be first analysed when all simulations finished. We develop a system that aim to provide more flexibility to the cluster computations via application programming interface (API).

4. MULTIPLE FRONT-ENDS

The API allows both automating configuration and making pre-processing and post-processing of simulations more flexible. In particular, a queue of simulations can be manipulated during execution: new tasks can be added, and ready tasks can be immediately post-processed, e.g. results can be visualized. The API allows attaching various front-ends to the system. Particularly important are

- MatLab front-end, which makes possible to interactively affect simulation by calling MatLab commands implemented as MEX-functions.
- Mathematica front-end, which makes possible to call special functions, implemented via MathLink inter-process communication standard
- C front-end, which can be used independently or can be attached to any other environment.
- Tcl/Tk front-end, which is convenient for scripting and rapid prototyping of graphical user interfaces.



In addition to interactive front-ends it is possible to use special purpose optimisation software. In particular the Robotics institute of German Aerospace Agency uses the MOPS tool /1/, based on MatLab for optimisation of robot and aircraft parameters /11/.

5. HETEROGENEOUS IMPLEMENTATION

The kernel of cluster computation API is implemented as dynamically linked library, which can be used with several different front-ends as mentioned above. CORBA is used for communication between the master and slave nodes. The master performs configuration, distribution and monitoring of the system. The slave nodes perform actual simulation work as well as monitoring of CPU and keyboard activities on the remote computers.

API allows specifying tasks to be performed. Each task contains information about the executable (in format matching the operating system), as well as a list of input and output files. In addition post- and pre-processing actions for master node can be specified. The master node stores the queue of the tasks.

The slave nodes monitor local CPU and keyboard activity. If CPU activity is lower than a threshold, and keys and mouse were not active for some time, the node is marked as available for computations. Also node is available if no users are currently logged in. Tasks are fetched from the queue on first-in/first-out basis and executed on the first available node.

When the task is starting, pre-processing actions are performed on the master node, necessary input files and executable files are sent from the master to slave node, executable is running, and output files are delivered from slave to master node. This file can be processed by the post-processing actions mentioned above. Instead of sending the files, shared network system can be used.

Normally, there is no need to specially re-design the executable for work with this environment. However, the executable should be able to run in non-interactive mode, i.e. as a command-line application, without opening any windows.

API includes options to monitor CPU time of each executed task. In addition it is possible to monitor the internal status of executed simulation, assuming that it communicates with the operating environment by writing status files.

6. EXAMPLE

Several computers on the local area network run their slave nodes. Each slave is a Windows service that waits for CORBA commands from the master node. A Windows service is running even if nobody is logged on to the workstation.
The MATLAB script below contains definitions for several tasks. For each task we specify input and output files (and directories, on the master node), as well as executable name and command line parameters. The preparation of the input files and analysis of output files usually requires some separate tools, which is not part of the distributed simulation system. The sample script illustrates operation with Modelica simulations /2,12/ using the simulation executables compiled by Dymola tool /9/.
The command **enqueue** appends the recently defined task to the queue.
The command **queue_go** actually starts the processing of the queue, displays the progress of queue execution and returns the control to the MATLAB user when all tasks in the queue are terminated (or aborted due to timeout).

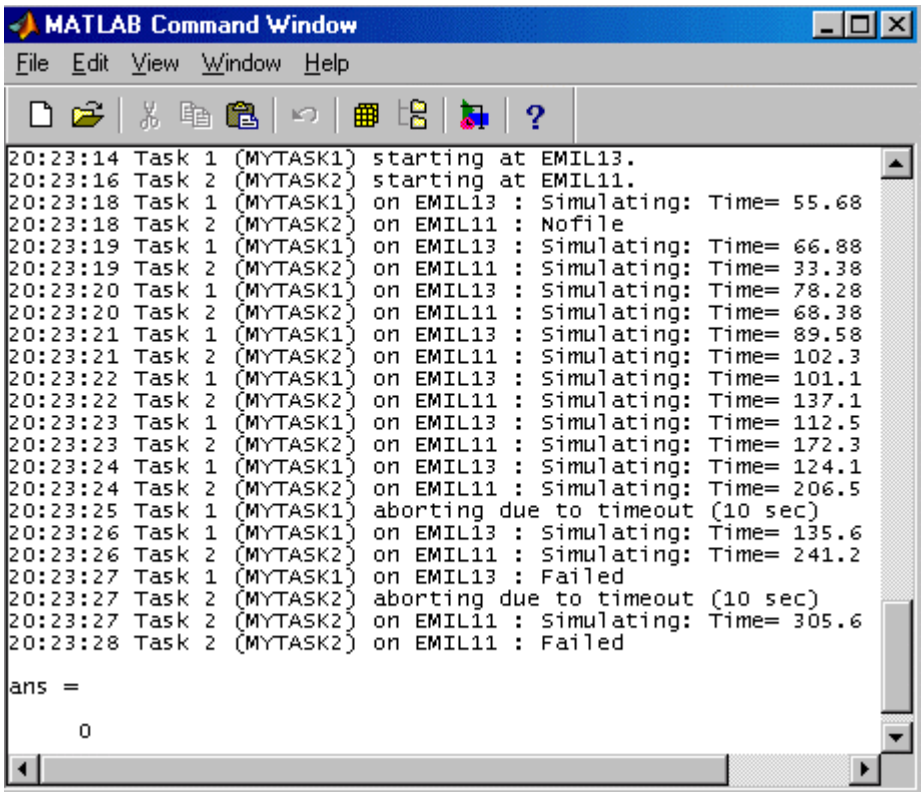
```
dissim input_files 'dsin.txt';
dissim input_directory 'c:\test\dymorun';
dissim exe_file 'dymosim.exe';
dissim exe_directory 'c:\test\dymorun';
dissim output_files 'dsres.mat' 'dsfinal.txt' 'status';
dissim output_directory 'c:\test\dymoout';
dissim (time_limit,10);
dissim enqueue 'MYTASK1';

dissim input_directory 'c:\test\dymorun2';
dissim output_directory 'c:\test\dymoout2';
dissim enqueue 'MYTASK2';

dissim queue_go
```

Initially the CORBA Name Server should start on an arbitrary node on the network. Each slave node starts and registers itself at the name server. These activities are performed as Windows Services or Unix Daemons and the processes are active even when nobody is logged in, and even when the master node is not active.
Two similar tasks were supplied to the queue on the master node. The master node determined that two slave nodes are available for execution. Tasks are assigned to the free nodes. The resulting display of queue progress is given below. Messages for the process monitoring are displayed in the MATLAB window. These messages contain task name, slave node name, and simulated time, i.e. internal progress of the simulation. It would be difficult to display percentage of ready simulation because often the differential equation solvers use adaptive steps and simulation speed greatly varies during simulation.
The tasks specification explicitly contains timeout (10 seconds). Therefore the tasks are aborted after 10 seconds. In order to be able to abort simulation in a nice way (free system resources and close output files) the simulation

executable (in this case, **dymosim.exe**) should provide such capabilities. On UNIX machines it is possible to define signal handlers for that. On Windows the safest solution is that simulation executable checks whether a special request file exists. If such file exists, it can instruct the simulation executable to terminate.



7. DISCUSSION ON MULTITHREADING

The chosen implementation solution does not utilize multithreading capabilities of CORBA. Instead, the queue manager on the master node gathers information from all slaves, determines their state. If some slave reported that the simulation run finished, then result files are fetched, and new task is launched on the free node. After that the master sleeps (e.g. for one second, as can be seen above). On slave nodes the simulation executables are launched as detached processes.

A multithreading solution would allow reducing waiting time. On the master node a separate thread should be created for each slave node. Such thread would wait until simulation executable terminates. Separate thread should be used for monitoring and for aborting simulations.

The queue control algorithm can run in a separate thread. In this case it is possible to return control to the MatLab prompt immediately after the queue is launched; furthermore, the user is able to add new tasks to the queue after it has started. A special monitoring tool is needed, since messages from the queue manager should not appearv in the MatLab command window.

8. CONCLUSIONS

The tool is intended for improving use of currently wasted computational resources. It appears that organizing non-interactive simulations is a relevant application for this purpose, especially when many independent simulations are prepared for running at once. Model parameter optimisation is one such application, but there can be many other cases as well.

The tool is currently in the prototype phase, allowing to communicate between several Windows and UNIX (Linux and Solaris) workstations using OmniORB (/3/, an open CORBA implementation) and MatLab (as a front-end). It will be used for optimisation of robot models, which are modelled in the Modelica language /2,11/. The current state of the project can be monitored at /5/.

This work is supported by European Commission IST Programme project RealSim. /6/

9. REFERENCES

1. Multi-Objective Parameter Optimisation (MOPS) tool, http://www.robotic.dlr.de/control/mops/
2. Modelica, a language for modelling and simulation of physical systems, www.modelica.org
3. OmniORB, high performance CORBA 2 ORB, www.omniorb.org
4. Cluster Computing Overview, http://www.cs.wayne.edu/~haj/load/survey/overview.html
5. Distributed simulation of Modelica models, available via http://www.ida.liu.se/~pelab/modelica
6. RealSim, Real-time simulation, http://www.ida.liu.se/~pelab/realsim
7. Windows Clusters Resource Centre http://www.windowsclusters.org
8. Condor, high throughput computing environment, http://www.cs.wisc.edu/condor/
9. Dymola with Modelica support, Dynasim AB, Sweden, http://www.dynasim.se
10. Joos, H.-D., Looye, G., Moormann, D.: Design of Robust Dynamic Inversion Control Laws using Multi-objective Optimization. AIAA Guidance, Navigation and Control Conference, Montreal, Canada, 2001.
11. Fritzson, P. and Bunus, P., Modelica, A General Object-Oriented Language for Continuous and Discrete-Event System Modeling and Simulation, in Proceedings of the 35th Annual Simulation Symposium, San Diego, April 14-18, 2002.
12. Basney, J., Livny, M., "Deploying a High Throughput Computing Cluster", *High Performance Cluster Computing*, Rajkumar Buyya, Editor, Vol. 1, Chapter 5, Prentice Hall PTR, May 1999.