Distributed Applications from Scratch: Using GridMD Workflow Patterns

I. Morozov[1,2] and I. Valuev[1]

[1] Joint Institute for High Temperatures of Russian Academy of Sciences, Izhorskaya 13/19, Moscow, 125412, Russia
[2] Moscow Institute of Physics and Technology (State University), Institutskii per., 9, Moscow Region 141700, Russia
valuev@ihed.ras.ru

Abstract. A new approach is proposed to generate workflow scenarios of scientific applications such as Molecular Dynamics and Monte-Carlo simulations in a distributed environment. The approach is based on embedding all workflow elements into the source (C++) code of the application as external library (GridMD) function calls. Thus the compiled executable is used both to generate the scenario and to perform computations related to the individual scenario elements. Having the scenario, its execution may be delegated to any resource manager which supports workflows.

1 Introduction

Scientific applications such as simulations using Molecular Dynamics (MD) and Monte-Carlo (MC) methods often require substantial computational resources. Available MD/MC packages (such as NAMD [2] or LAMMPS [1]) provide efficient serial and parallel algorithms for simple execution scenarios, such as to take the system of particles at some initial state and to propagate it through the chain of other states. At higher level the numerical experiment includes statistical averaging [3], parameter sweep, optimal parameter search, etc. While distributed environment looks very promising for these tasks, the researchers face the problem of constructing complicated scenarios and workflows.

Various tools for Grid-enabled workflow management are being developed [4, 5, 7, 8]. They provide both visual interfaces based on direct acyclic graphs [4, 5] and specialized languages [7, 8] for definition of the workflow. However in all cases this definition is to be made by the user of application who must be aware of the execution scenario and the workflow definition software.

Recently GridMD [9] C++ class library was proposed, which is designed for the simulations developers, the programmers who create computational applications. Researchers may use these applications in production runs to obtain physical results. We emphasize here the distinction between users and developers, which for scientific applications are often the same people, to stress the GridMD strategy of delegating the workflow specification entirely to the application developer. The workflow elements are specified in easy and portable way

Y. Shi et al. (Eds.): ICCS 2007, Part III, LNCS 4489, pp. 199–203, 2007.

by the special GridMD function calls inside the application C++ code. Being embedded, the scenario information can be extracted from the executable and passed to the workflow management system at run-time with minimal user intervention. The aim of the library is also to provide specialized mechanisms of submitting the GridMD application jobs to different workflow management systems. The detailed knowledge of these submission mechanisms is not required neither for the application developer nor for the application user. Between the submission mechanisms, the one called "local" where all jobs are executed one-by-one on the same computer is always accessible by any GridMD application. It may be used for testing the distributed application without even having the actual access to the Grid and workflow software. In the following we will analyze the mechanisms of experiment scenario specification in GridMD in more detail.

2 GridMD Workflow Specification Patterns

GridMD has the usual notion of workflow, which may be represented by directed graph consisting of nodes and links (edges of workflow graph). The links represent dependence between nodes, the nodes represent some actions of the program. Unlike other workflow systems, GridMD does not require that *all* actions in the code should be wrapped in workflow elements. Instead, only the parts of the code most important for distributed execution may be specified as nodes.

There are several types of links in GridMD: *logical* or *hard* link from node A to node B means that node B may be executed only by the same process as node A AND only after node A; *data* link between A and B means that either A and B must be executed consecutively by the same process OR B may be executed without executing A provided that the *result data* of node A, associated with this data link, is accessible by the process executing B; *process* link between A and B is a special case of *hard* link assuming that some calculations which are initialized at node A are taking place in node B. Although being a limited subset of possible node dependencies, used by different workflow systems, this system of links provides a sufficient basis for iterative simulation experiments and can be easily extended in the future. Data links are the main sources of distributed capabilities of the code in general. The node result data associated with a link is represented in C++ code as an object of some data type.

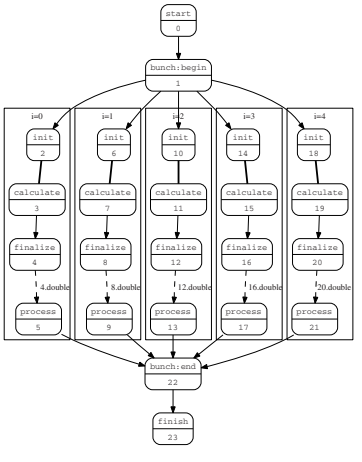
The workflow nodes are defined and executed by the same GridMD application, so this application may proceed in two execution modes. *Construction or manager mode* is used for the assembly of the workflow graph and to its subsequent analysis. *Worker mode* is used to execute some subset (subgraph) of the nodes through the invocation of the same application with command line parameters uniquely specifying the subgraph for execution.

3 Example of the Distributed Application

In GridMD, the nodes and links can be created *explicitly*, by associating them with C++ virtual functions and linking them in the graph, or *implicitly*. By

the example in Fig. 1 the use of the implicit mechanism to easy redesigning the existing applications for distributed execution is illustrated. The initial program (Listing 1) is complemented by GridMD calls, and the workflow graph is produced automatically. As it is seen from Listing 2, the implicit node markup patterns have a form of conditional operators with GridMD functions as arguments. All these functions (`bunch.begin()`, `bunch.node_define()`, `bunch.node_process()`, `bunch.end()`, `gmdExperiment.execute()`) return zero in the manager mode and are used simultaneously to specify the nodes and to conditionally bypass the time consuming parts of node execution. The bypassed parts are in fact the actions, associated with defined nodes. They are performed only in worker mode and only if corresponding GridMD functions return nonzero, i.e. when the execution of specific node is requested.

The `gmdSweep` object implements branching where a data link can be put in the middle of the branch (see Fig 1). This data link is bound with an object of some data type (this type is a C++ template parameter, double for the example). The object for the current branch may be accessed by `node_result()` function both in the source node of the link (marked by `node_define`) and in the target node (marked by `node_process`). If these nodes are executed by different processes, the data transfer according to this link (including file creation) is managed by GridMD through type-specific I/O operations.



Listing 1:

```
1  double s=0;
2  for(int i=0;i<5;i++){
3    s+=long_calculation(i);
4  }
5  printf("The result is: %lf\n",s);
```

Listing 2:

```
1  gmdExperiment.init(argc,argv);
2  gmdSweep<double> bunch("bunch");
3  bunch.mark_begin();
4  double s=0;
5  for(int i=0;i<5;i++){
6    if(bunch.mark_node_define(strfmt("i=%d",i)))
7      bunch.node_result()=long_calculation(i);
8    if(bunch.mark_node_process())
9      s+=bunch.node_result();
10 }
11 bunch.mark_end();
12 if(gmdExperiment.execute())
13   printf("The result is: %lf\n",s);
```

Fig. 1. Execution graph (left part) automatically generated from the code presented in Listing 2 (right part). Logical links are represented by solid lines with arrows, data links by dashed lines with names of transferred files indicated, process links are shown by thick lines. The numbers are node unique identifiers which are assigned to the nodes in the order of creation. Visualized by graphviz.

After the workflow graph is created, an iterative graph analysis algorithm is used to determine which branches (subgraphs) may be used concurrently. In order for the workflow defined by GridMD markings to be consistent, there must

be no implicit logical links between nodes except those known from the workflow graph. This has to be checked by the application code developer, because GridMD is unable to analyze this in advance in construction mode and will only report an error in worker mode when the actual execution does not conform to the defined graph.

Distributed execution can be controlled by the GridMD application itself or it can be delegated to the external execution manager which supports workflows (NIMROD [10], Kepler [4]). In the later case GridMD application first generates the workflow definition file (plan file or script). The script (external Perl submission) for starting all the tasks of example in Fig. 1 is shown below:

```
# Generator: appl.exe -plscript -engine=submit.pl
require "submit.pl";
# distributed subgraph submission
submit("appl.exe -w0-4","","4.double");
submit("appl.exe -w0-1:6-8","","8.double");
submit("appl.exe -w0-1:10-12","","12.double");
submit("appl.exe -w0-1:14-16","","16.double");
submit("appl.exe -w0-1:18-20","","20.double");
wait_for_queue();    # wait till all subgraph tasks finished
# distributed subgraph submission
submit("appl.exe -w5:9:13:17:21-23","4.double 8.double 12.double 16.double 20.double","");
wait_for_queue();    # wait till all subgraph tasks finished
```

Acknowledgements

This work is supported by the program of fundamental research of the Russian Academy of Sciences #15 and Dutch-Russian Project "High Performance simulation on the grid" NWO-047.016.007/ RFBR-04-01-89006.

References

1. http://www.ks.uiuc.edu/Research/namd/
2. http://lammps.sandia.gov/
3. Kuksin, A.Yu., Morozov, I.V., Norman, G.E., Stegailov, V.V., Valuev, I.A.: Standards for Molecular Dynamics Modelling and Simulation of Relaxation. Molecular Simulation **31** (2005) 1005–1017
4. http://www.kepler-project.org
5. http://pegasus.isi.edu
6. Pytlinski, J., Skorwider, L., Benedyczak, K., Wronski, M., Bala, P., Huber, V.: Uniform Access to the Distributed Resources for the Computational Chemistry Using UNICORE. In: Sloot, P.M.A., et al (eds.): Lecture Notes in Computer Science, Vol. 2658. Springer-Verlag, Berlin Heidelberg New York (2003) 307–315
7. W.M.P. van der Aalst: The Application of Petri Nets to Workflow Management: The Journal of Circuits, Systems and Computers, Vol. 8, No. 1 (1998) 21–66.
8. Lee, E.A., Parks, T.M.: Dataflow process networks. Proc. of the IEEE **83** (1995) 773–799

9. Valuev, I.: GridMD: Program Architecture for Distributed Molecular Simulation. In: Hobbs, M., et al (eds.): Lecture Notes in Computer Science, Vol. 3719. Springer-Verlag, Berlin Heidelberg New York (2005) 307–315
10. Sudholt, W., Baldridge, K., Abramson, D., Enticott, C., Garic, S.: Parameter Scan of an Effective Group Difference Pseudopotential Using Grid Computing: New Generation Computing **22** (2004) 125–135