Exploiting the Concept of Activity for Dynamic Reconfiguration of Distributed Simulation

Ming Zhang\textsuperscript{1}, Bernard P. Zeigler\textsuperscript{2}, Azzedine Boukerche\textsuperscript{1}

\textsuperscript{1}Paradise Research Lab
School of Information Technology and Engineering (SITE)
University of Ottawa
Ottawa, Ontario Canada, K1N 6N5
\{mizhang, boukerch\}@site.ottawa.ca

\textsuperscript{2}Arizona Center of Integrative Modeling & Simulation
The Department of Electrical and Computer Engineering
University of Arizona, Tucson, AZ 85721
zeigler@ece.arizona.edu

Abstract

In this paper, we specialize the concept of “activity”, defined in earlier work, to measure the heterogeneity of model behavior using the temporal-spatial distribution of its local transitions in a 2D cellular space. We then show how to employ this “activity” metric to balance the computation load using the dynamic reconfiguration of the distributed simulation. We also show how the degree of improvement depends on the heterogeneity of the activity's distribution. That is, high concentrations of activity in space that change relatively slowly during simulation can be exploited to reduce execution time significantly within an appropriate infrastructure for dynamic reconfiguration in a DEVS based distributed simulation framework. In contrast to other dynamic load balancing approaches, the activity-based approach discussed here exploits model properties directly rather than relying on resource-based measurements as the basis for its reconfigurations.

1. Introduction

With the increased demand for computing resources from modern simulation applications, parallel and distributed simulations are attracting more and more attention from researchers in this area. In more and more cases, these types of simulations are becoming a necessity for solving large-scale simulation applications rather than just a potentially performance improving approach. However, effective model partitioning is the key to determining the overall simulation execution performance when parallel and distributed simulation techniques are used.

In this paper, we focus our study on model “activity” based dynamic repartition in a distributed simulation environment. We exploit our idea through a highly dynamic discrete event model represented in DEVS, and then experiment with this model in a flexible distributed simulation framework. In particular, we are interested in how the distribution of “activity” in a simulation model determines the computing workload distribution, and therefore affects the simulation execution performance in a distributed computing environment. Moreover, we focus our implementation on a DEVS based distributed simulation framework that uses “activity” as a measure of the computing workload [1][2]. We will show how to exploit this “activity” metric to improve the distributed simulation performance by applying it to a run-time repartitioning approach.

Discrete event simulations are characterized by asynchronous and irregular, random, or data dependent behavior [3], which requires a highly dynamic and strict modeling and simulation framework. In contrast to other modeling and simulation methodologies, DEVS [4] provides a theoretical foundation for the modeling and simulation of discrete event system (DES), continuous systems, and hybrid systems. DEVS based modeling and simulation frameworks are generally flexible, rigorous, and conform to modern

\textsuperscript{1} This work is partially supported by Canada Research Chair Program, NSERC Ontario Distinguished Researcher Award and Ontario Early Researcher Award.
software engineering standards. At the present time, DEVS based frameworks have also been verified as being able to solve large-scale and highly dynamic simulation models effectively. In particular, distributed DEVS frameworks such as ADEVS [5], DEVS/Grid [6], DEVS/P2P [7], and DEVS/RMI [8], have opened us a wide area of research on parallel and distributed simulation using DEVS.

With regards to distributed simulation performance, model partition algorithms are worth investigating, particularly dynamic partition or repartition techniques. In general, the run-time behaviors of a simulation model play an important role in determining the optimal model partition schema. However, it is difficult to predict the runtime behaviors of a highly dynamic simulation model and, therefore, a well predefined model partition plan is not easy to obtain in practice when running the model in a distributed fashion. Also, a parallel or distributed simulation framework that supports dynamic reconfiguration is needed to properly support the dynamic repartitioning of simulation models on clusters of machines.

In this paper, we present a dynamic reconfiguration mechanism that uses a run-time gathered “activity” metric to repartition a simulation model in order to improve the overall performance of a distributed simulation. We exemplified the concept of “activity” through the re-implementation of a time-stepped valley fever model [9]. This implementation uses a DEVS-based asynchronous approach to represent the behavior of cells, called “patches”, in a 2D cellular space. In the following sections, general model partitioning techniques are reviewed briefly, followed by a section that describes how the dynamic reconfiguration capability is implemented in a distributed simulation framework to be used for later experimentations. The re-implemented DEVS based valley fever model is then tested and discussed to demonstrate the effectiveness of using an “activity” based model reparation on improving the performance of a distributed simulation. In the last sections, a conclusion and suggestions for future work are presented.

2. Model Partitioning in A Distributed Simulation

In this section, we review some of the major model partitioning concepts used in distributed simulations [17-20]. We will provide some basic background information with regard to general model partitioning techniques and, in particular, we focus on what “activity” is and how it is used in model partitioning in a distributed simulation environment.

2.1. General Model Partition Techniques

In general, partitioning techniques can be classified into the following: random partitioning, partitioning improvement, simulated annealing, and heuristic partitioning [10][11]. Random partitioning randomly aggregates models to a set of partition blocks and then maps the partition blocks to the processors. The Partitioning improvement algorithm modifies the partitioning results during the process of partitioning [12][13]. Simulated annealing [14-16] uses statistical methods to develop the process of model partitioning, and, finally, heuristic partitioning is an algorithm that uses domain-specific knowledge or a particular optimization technique to achieve a better partitioning results.

Hierarchical model partitioning is a technique that applies a general model partitioning technique, such as graph partitioning, to the hierarchical model structure of a distributed simulation. It is a process of constructing partition blocks by decomposing a hierarchical model structure based on certain decision-making criteria. Hierarchical model partitioning is especially important for DEVS based distributed simulation environments because the model structure in most DEVS implementation uses such a hierarchical, modular model structure to represent a system for simulations. General hierarchical model partitioning techniques include the flattening, deepening, and heuristic. Flattening transforms a hierarchical structure into a non-hierarchical structure. Deepening, sometime called hierarchical clustering, is a technique that in reverse transforms non-hierarchical structures into hierarchical structures. The heuristic technique uses heuristic functions to analyze the nodes in a hierarchical model tree to determine the partitioning policies. In fact, hierarchical model partitioning is the basis of the “cost” or “activity” based model partition that we will discuss in the following section.

2.2. “Activity” Based Model Partition

In this sub-section, we define and discuss some of the fundamental concepts related to the “activity”. In general, “activity” is a term to define how “active” a component/role is in a system in terms of some predefined rules. For instance, “activity” concept is introduced in [23] using DEVS quantization theory, which defines the “activity” as: “ A cell is said to be
In this paper, we are particularly interested in investigating how the model's intrinsic properties or “activity” determine the run-time computing workload distribution of a simulation model. Such key information is crucial for obtaining an optimal model partition and/or repartition plan for a distributed simulation. As we know, a flexible and dynamically reconfigurable distributed simulation framework is required to support generic static model partitioning as well as more advanced functionalities such as model dynamic repartitioning. In this section, we will introduce a DEVS based framework called DEVS/RMI [8], which has the capabilities needed for the study of the model repartition mechanism proposed and implemented in this paper.

For a short summary of its key attributes, DEVS/RMI is a distributed DEVS that is able to support the seamless distribution of simulation entities across network nodes. It supports model continuity in a distributed environment, which means that a model system can be developed and tested in a single machine, and then be mapped to the distributed computing nodes without any changes aside from adding attributes for the simulation controller to know where to situate the models. Furthermore, DEVS/RMI can provide a simulation application with a dynamically re-configurable run-time infrastructure in order to handle load balancing and fault tolerance in a distributed simulation environment.

In DEVS/RMI, model partitions are described in the model definition and configuration layer, which is separate from the simulation layer. Such model partitions are implemented in the model construction phase and then manipulated by the corresponding simulators. In this way, any atomic model or coupled sub-model can be assigned to any computing node during the initialization phase of a simulation. This means that random partition is directly supported in DEVS/RMI. However, in order to reduce the communication overhead, regrouping the models into sub-domains is commonly used before assigning the partitioned sub-models to computing nodes.

Now, we will introduce the dynamic model repartition in DEVS/RMI, which distinguishes itself
from most other distributed simulation frameworks in term of having this capability. The idea of the dynamic reconfiguration of the DEVS model was initially proposed by Hu [21], which supports the evolution of the model structure during the simulation's run-time. DEVS/RMI extends this idea and implements the model repartition capability in a distributed simulation environment. Such dynamic reconfiguration capability is easily implemented in DEVS/RMI, and takes the advantages of Java Remote Object technology. As shown in the illustrated example in Figure 1, the figure in the top half shows the initial model partition in two sub-domains, while the bottom part of the figure shows that “cell 13” and “cell 23” in “sub-domain 2” migrated to “sub-domain 1” during run-time. Such a process is accomplished by decoupling “cell 13” and “cell 23” from their neighbor cells and sub-domain boundary (i.e. the digraph to which they belong), and then migrating them by a RMI call at the simulation controller such as RMICoordinator. After such model migrations, new couplings need to be added to maintain the overall coupling relationship between cells in the cell-space.

As we have seen in the above discussion, DEVS/RMI provides an ideal solution for implementing “activity” based model partition and repartition with the native support of dynamic model reconfiguration. In the following section, we will test our ideas with a focus on the simulation performance evaluation in a computer cluster environment.

Figure 1. Dynamic Model Repartition
4. An Example

In this section, a 2D DEVS valley fever model is exemplified to show how to exploit the concept of “activity” to obtain improved model partition plans for dynamic reconfiguration. The advantage of such a model “activity” based partition plan can be verified by a comparison experiment using a distributed simulation. Our experiment aims to provide a clear picture of how the model “activity” metric can help with obtaining an improved partition plan during a simulation's run-time. In this example, a Linux Beowulf cluster is used to run the valley fever model in distributed computing nodes.

4.1 Valley Fever Model

The agent-based valley fever model is a 2D dynamic cell space model used to represent how the fungal spores grow in a patch of field over a long period of time with given environmental conditions including wind, rain, and moisture, etc. This model is initially a time-stepped model, and is then re-implemented in DEVS to run on an ADEVS [22] C++ platform. Furthermore, the ADEVS valley fever model is translated into Java for distributed execution using DEVS/RMI. As shown in Figure 2, the Java based valley fever model consists of several components: the wind model, rainfall model, coupling control model, and patch model. All these components are DEVS atomic models except for the patch model, which is a DEVS coupled model consisting of an atomic model called “sporingProcess” and another atomic model called “environment”. To run this model in a distributed DEVS such as DEVSTM/RMI, particular attention is given to the “patches” models, which sit in a 2D cell space. The “wind” and “rainfall” models are both statistic models that generate wind data and rain data periodically. Their outputs are then sent to the “coupling control” model to determine the dynamic coupling of the “rain” model with the “patches” as well as the dynamic coupling between “patches”. This valley fever model is a highly dynamic one that changes its structure after every step of the simulation.

4.2 Static Blind Model Partition of Valley Fever Model

In order to create a comparison baseline for dynamic model repartition, static blind model partition is used to map the “patch” models to computing nodes. In this setting, the “wind”, “rain” and “coupling_control” models are all arranged at the “head” node, and the “patch” cells are evenly divided between other computing nodes in a “blind” fashion, i.e., without regard to their measured activities. For example, for a 4 * 4 cell space to run on four computing nodes, each column of cells is assigned to one computing node, resulting in an even distribution of four cells to each computing node.

4.3 Dynamic Reconfiguration Using “Activity”

The static blind model partition described in section 4.2 does not consider the imbalance of the workload on each individual cell. Some cells may have less “activity” than others, and are therefore subject to less computing workload. Partitioning the cells blindly results in an imbalance of the workload of computing nodes, which cannot benefit the overall performance of the simulation. However, when given a highly dynamic simulation model such as that for valley fever, it is generally difficult to predict the model run-time behavior.

Fortunately, in the valley fever model, the production of spores is the main driver of activity in the patches. “Sporing” is largely determined by the strength and direction of the wind, which is an external input to the model. New “sporing” patches are typically highly concentrated in the direction of the wind. The fact that wind regimes change relatively infrequently allows us to obtain stable activity distributions using a simulation on a single machine for each such regime. The activity at each cell in a given period is measured as the total number of internal transitions that the cell undergoes during that period. Experimentally, we verify that this number is closely related to the
computational intensity required to simulate a cell during such a period.

Given the activity's dependence on wind regimes, we can measure model “activity” by executing the model (on a single machine or in a distributed fashion) for the desired wind regimes and gathering the model “activity” metric through such a run. This information can then be applied to obtain a model partition plan for a distributed run of the same model configuration. Since the wind regime is controlled externally to the model, we can monitor the wind generator and apply the partition plan that is optimal for a regime whenever the wind changes. We measure the model activity by counting the internal transitions of the ‘sporingProcess’ to see how a dynamic model repartition using such information can benefit the distributed simulation's performance. In this example, a simplified method is used to determine a subset of cells called high-activity cells.

Firstly, the average internal transition count of all the cells in the cell space is obtained by running the model in the head node for a given wind regime. At the end of this run, each cell compares its own count with the average, if it is larger than the average, the cell’s id is added to a linked list data structure for high-activity cells. Figure 3 illustrates how high-activity cells are selected from the cell space.

After the high-activity sets are obtained for each wind regime, the following process occurs within a single simulation run. The “RMICoordinator” in the head node creates a new valley fever model and partitions it according to the initial wind regime. Every time a new wind regime is detected, the “RMICoordinator” creates a new valley fever model, and partitions its cells so that the high activity cells for that regime are granted more computing power than are the remaining cells. Finally, the cells are dynamically loaded to the computing nodes and the distributed simulation is then restarted from the state that the model was in before the repartition.

In the following test, we discuss one iteration of this process in which all the low activity cells are assigned to one computing node, and all the high activity cells are then evenly distributed between the other available computing nodes. Some test results are presented in the next section.

4.4 Test Environment and Results

In this experiment, a 40 node Linux cluster is used, in which each node has an AMD Athlon XP 2400+ with 2GHz CPU and 512M physical memory. All the computing nodes are inter-connected by 100M Ethernet switches, and the operating system of each node is GNU/Linux 2.4.20 with Java Runtime 1.4.1-01 installed.

In this test, static blind partitioning is compared to dynamic reconfiguration in terms of the simulation’s execution time, and 4 * 4 and 8 * 8 cell spaces are used and executed with 400 and 2000 simulation steps. The purpose of this test is to verify the advantage of using “activity” based dynamic repartition over static “blind” model partitioning. Such verification will also prove that model “activity” is a more accurate indicator for computation workload of the examined cells in a cell space.

As shown in Table 1, for a 4 * 4 cell space, there is no noticeable difference between using dynamic reconfiguration and static blind partitioning. However, for a 8 * 8 cell space, dynamic partitioning using model “activity” improves the simulation performance in a noticeable manner. This is because, for a 4 * 4 cell space, there is only a difference of a few cells between each computing node, and these cells cannot contribute enough to make the difference in the workload distribution. It could be expected that for a larger cell space with long simulation execution steps, model “activity” would play an increasingly important role in

Figure 3. Selecting High-Activity Cells

[Diagram showing the process of selecting high-activity cells]

All Cells
Counting number of internal transitions of each cell
Calculate Average of internal transitions
Compare each cell with average
Else
If this cell’s number of internal transitions average
Add this cell to low-activity list
Add this cell to high-activity list

In this example, a simplified method is used to determine a subset of cells called high-activity cells.
Table 1 Distributed Simulation Execution Time for Static Blind Partition and Dynamic Reconfiguration Using “Activity”—5 nodes.

<table>
<thead>
<tr>
<th>Using 5 computing nodes including 1 head node.</th>
<th>Static Blind Partition not considering model activities</th>
<th>Dynamic reconfiguration using “activity”</th>
<th>Performance increase by percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 * 4 cells with 400 simulation steps</td>
<td>28.124s</td>
<td>27.566s</td>
<td>1.98%</td>
</tr>
<tr>
<td>4 * 4 cells with 2000 simulation steps</td>
<td>113.977s</td>
<td>114.968s</td>
<td>-0.87%</td>
</tr>
<tr>
<td>8 * 8 cells with 400 simulation steps</td>
<td>256.49s</td>
<td>248.644s</td>
<td>3.06%</td>
</tr>
<tr>
<td>8 * 8 cells with 2000 simulation steps</td>
<td>1238.479s</td>
<td>1216.97s</td>
<td>1.73%</td>
</tr>
</tbody>
</table>

Table 2. Distributed Simulation Execution Time for Static Blind Partition and Dynamic Reconfiguration Using “Activity”—9 Nodes.

<table>
<thead>
<tr>
<th>Using 9 computing nodes including 1 head node.</th>
<th>Static Blind Partition not considering model activities</th>
<th>Dynamic reconfiguration using “activity”</th>
<th>Performance increase by percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 * 4 cells with 2000 simulation steps</td>
<td>134.74s</td>
<td>110.49s</td>
<td>18%</td>
</tr>
<tr>
<td>8 * 8 cells with 2000 simulation steps</td>
<td>1348.17s</td>
<td>1199.87s</td>
<td>11%</td>
</tr>
</tbody>
</table>

affecting the distributed simulation performance. Table 2 does indeed verify our expectations for performance improvement when more computing nodes are used for high-activity cells. In this test, we can see a significant performance increase when using “activity” based model repartition compared to static blind model partitioning.

The test results suggest that it is worth further investigating the concept of model “activity” in more detail and developing model partition plans that exploit activity distributions in a more precise way.

5. Conclusion

In this paper, we present and demonstrate how a DEVS “activity” based model repartition can influence the performance of a distributed simulation. As is known, dynamic model reconfiguration plays a very important role in distributed simulation performance, especially when running large-scale models exhibiting highly asynchronous and irregular behavior. Here, we have shown that dynamic model reconfiguration using the “activity” metric can improve distributed simulation performance significantly. It is worth noting that workload distribution is crucial for optimizing the performance of large-scale distributed simulation applications, while the model “activity” metric is the key to obtaining critical workload distribution information. We have shown that DEVS/RMI provides the flexibility necessary to exploit model-intrinsic properties to order to direct dynamic repartition. Such environments will play increasingly important roles in future distributed simulation applications.

6. Future Work

For future work, we propose to further investigate the concept of “activity” to better understand how intrinsic model dynamic behaviors determine the run-time workload distribution. We will do more experiments on larger cell space sizes and larger number of processors to obtain a clearer picture on how “activity” affects the performance of a
distributed simulation. We are also interested in studying model repartition algorithms in a distributed simulation environment with the support of a flexible framework such as DEVS/RMI. How to effectively monitor the model's run-time “activity” is also a topic that needs to be studied further.

Reference: