UTILIZING DAG SCHEDULING ALGORITHMS FOR ENTITY-LEVEL SIMULATIONS

Alan Su* Henri Casanova†* Fran Berman†*

* Computer Science and Engineering Department,
University of California, San Diego.
alsu@cs.ucsd.edu

† San Diego Supercomputer Center,
University of California, San Diego.
{casanova,berman}@sdsc.edu

Keywords: entity-level models, individual-based models, DAG scheduling algorithms, heterogeneous computing

Abstract

The continuing deployment of high-performance network technology enables the development of computing platforms that aggregate widely distribute hardware resources. The vision for such a *Computational Grid* promises computational platforms of unprecedented power for scientific applications. However, application developers need to rethink implementation paradigms in order to realize this potential. In this paper, we identify a class of increasingly important applications, *entity-level simulations*, which currently cannot use large-scale computing platforms effectively. We will show how careful application-aware scheduling can enable such applications to utilize large distributed heterogeneous platforms. Our initial approach is to exploit the structure of entity-level applications and leverage existing Directed Acyclic Graph (DAG) scheduling techniques. We validate our approach by simulating a realistic application scenario on several synthetic platforms, including a representative Computational Grid testbed.

I. INTRODUCTION

A common approach to studying large systems is to express the aggregate behavior of groups of individuals as a mathematical function. The interaction between multiple such phenomena can be characterized by solving for the steady-state behavior of a system of simultaneous partial differential equations. By contrast, the goal of entity-level models is typically to study the behavior of complex systems at the level of interacting entities, rather than the level of aggregate behaviors. In implementations of such models, each application task typically represents an entity in the system and encapsulates the necessary computation and communication operations that task must perform. This approach has been used in many fields, including traffic studies [6, 15], population studies [5, 9], and ecology [10]. These efforts have found that the behavior of complex processes can often be accurately predicted by the emergent behavior of entities in entity-level models.

However, efforts to utilize entity-based implementations have been limited to simple, homogeneous computing platforms. Noting the dramatic improvement in computing technology, application scientists are now considering entity-based implementations in large-scale distributed computing environments. In particular, the *Computational Grid* [7, 8], or *Grid*, seeks to enable ensembles of distributed resources to be used as a unified computing platform. To realize the potential of such platforms for entity-level applications, the principal challenge is the scheduling problem: assigning application tasks to available resources in a manner which promotes performance.

Section II presents an application model which captures the resource requirements of an entity-level application. Section III describes a process to express instances of this application model as directed acyclic graphs (DAGs). We conducted experiments to determine the efficacy of applying DAG scheduling algorithms to entity-level applications; results of these efforts are shown in Section IV. Finally, we conclude with a discussion of future work in Section V.

II. APPLICATION MODEL

A. Formal Model Definition

Entity-level model implementations are iterative applications composed of a set of entities with associated state in-

This research was supported in part by NASA Graduate Student Research Grant #NGT-2-52251. Equipment used in this research was supported in part by the UCSD Active Web Project, NSF Research Infrastructure Grant Number 9802219.

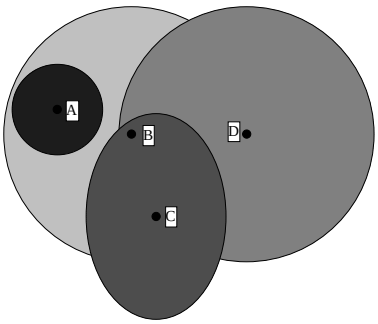

Figure 1. Entity-level Application Example

formation and behavior models. The *state* encapsulates static and dynamic properties of the entity in the application domain. An entity's *behavior models* govern interactions with other entities and state changes. An entity's *interest set* (the set of all other entities which influence that entity's behavior) can be derived from its behavior models. At every iteration, each entity takes into account its own state and the states of entities in its interest set, and evaluates its behavior models. In entity-level applications, this process results in some internal computation, communication with other application tasks, and state update operations. Once all entities have completed this process, the application is ready to proceed to the next iteration. Formally, our application model is composed of

- E, a set of entities;
- c_i, the amount of computation work for entity i in the current iteration, derived from the computational behavior model for entity i;
- $t_{i,j}$ the amount of data entity i must transfer to entity j in the current iteration, derived from the communication behavior model for entity i (and may be equal to 0 if entity j is not in the interest set of entity i);

B. Entity-level Application Example

To illustrate our model, we present a simple example of a four-entity system. In this system, the state of each entity is a set of coordinates in some Cartesian coordinate system. Without loss of generality, we assume a two-dimensional space.

Figure 1 depicts a snapshot of an instance of this model. In this example, entities have constant computational costs and communication costs determined by a "neighborhood" area in the application space. For example, entity task D computes c_D units of work and transfers $t_{D,B}$ and $t_{D,C}$ units of data to the tasks corresponding to entities B and C, respectively. Figure 2 summarizes the computation and communication requirements of this example.

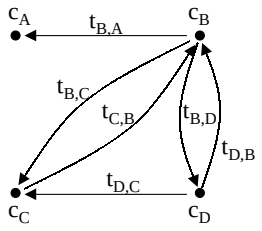

Figure 2. Computation and Communication Tasks

C. Scheduling

Given this application model, the job of a scheduler is to assign each application task to an available computing resource. The scheduling problem is known to be NP-complete [18], with exponential complexity in the number of computing resources; this is clearly infeasible for realistic platform sizes. Moreover, the number of entities and their computation and communication requirements are not known until run-time, making it difficult to employ compile-time heuristics.

Our goal is to develop computationally tractable run-time scheduling heuristics which yield good application performance. In the following sections, we describe a process to transform instances of the entity-level application model into DAGs and show results of scheduling the resulting DAGs with classical DAG scheduling algorithms.

III. DAG Transformation for Entity-level Applications

Little work has been done to develop scalable and general scheduling strategies for parallel implementations of entity-level applications. However, the problem of scheduling ***D**irected **A**cyclic task precedence **G**raphs*, known as DAGs, on distributed computational platforms has been deeply explored [11, 19, 16, 20, 14]. Tasks in a parallel application are represented as nodes, with node costs indicating the magnitude of processing that task requires. Edges represent the precedence relationships between tasks, with edge weights corresponding to the magnitude of inter-task communication. To enable DAG scheduling algorithms for entity-level applications, we propose a transformation to express instances of an entity-level model as a DAGs. The basic elements of this construction are:

- **Root and end nodes**: Our construction creates a single root node of which all other nodes are descendents, and a single end node which is the only node without children.
- **Sentinel nodes**: We define a set of nodes which map directly to the set of entities in the entity-level application model. The root node is connected to each sentinel node with a zero-weight edge (signifying precedence but no

data transfer). These are the only children of the root node and represent the mapping of entities to Grid resources at the beginning of the iteration. A set of sentinel nodes are connected to the end node in a similar fashion, representing the mapping of entities to Grid resources for the following iteration. All sentinel nodes have a zero cost.

- **Infinite-weight edges**: The DAG transformation creates several nodes in the DAG to represent the same entity in the entity-level model. In order to prevent the DAG scheduling algorithms from producing impossible schedules, we introduce the notion of an infinite-weight edge in the DAG to represent scheduling constraints.
- **Computation and communication sub-graphs**: The core of the DAG transformation algorithm converts each task (as depicted in Figure 2) into a "strand" of DAG nodes. These strands are then connected with other strands and the sentinel nodes to form the complete DAG. The strands that are constructed for computation and data transfer tasks are shown in Figure 3.

The transformation begins by instantiating the distinguished root and end nodes. A set of start sentinel nodes, one corresponding to each entity, is constructed. A set of end sentinel nodes is similarly constructed. Finally, each computation and communication task is transformed into an appropriate DAG strand. To compose the DAG from the sentinel nodes and strands, we use a combination of zero-weight edges, to indicate task precedence (but no real data transfers), and infinite-weight edges to reflect entity-based scheduling constraints. The basic methodology follows these steps:

1. Connect the root node to each start sentinel with a zero-weight edge.
2. Connect each end sentinel with a zero-weight edge to the end node.
3. Using an infinite-weight edge, connect each start sentinel node to the head of the computation strand derived from the same entity, as indicated in Figure 3. Similarly, connect the last node in the strand with the end sentinel node. This ensures that the computational task is scheduled on the correct Grid resource.
4. For each communication strand, connect the sentinel node corresponding to the source entity task to the head of the strand. Connect the last node in the strand with the end sentinel node corresponding to the target of the communication. Both edges are infinite-weight edges. This ensures the communication will occur on the appropriate network link.
5. Finally, for every entity with multiple communication strands, the communication strands must be connected in order to express serialization of communication. This is done by connecting the trailing node of each communication strand (except the last) to the leading node of the subsequent communication strand.

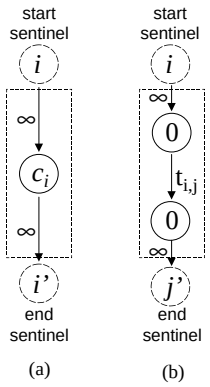

Figure 3. *Computation and Communication Strands*: (a) the computation strand resulting from the transformation of an entity computation task of magnitude c_i and (b) the communication strand resulting from the transformation of an entity communication task of magnitude $t_{i,j}$ from entity i to entity j.

Applying this transformation to the entity-level model in Figure 1 results in the DAG shown in Figure 4. Under this transformation, an n-entity system in which each entity communicates with an average of c other entities results in a DAG containing $2 + n(3 + 2c)$ vertices. Since each entity communicates with at most $n - 1$ other entities, the maximum size of a DAG generated by this transformation is $2 + n(3 + 2(n - 1)) = O(n^2)$. Our experimental results indicate that the overhead of applying DAG scheduling algorithms to DAGs generated from fairly large entity-level applications is quite minimal. However, we realize that an $O(n^2)$ growth in DAG size may be a significant limitation to scalability as we consider larger application problem sizes and more sophisticated DAG scheduling algorithms.

IV. RESULTS

Our intent is to test and validate our approach in the context of real applications. We derive our application model from IMMSIM, a simulation package for the human immune system [13, 12]. IMMSIM is a basic entity-level application targeted for homogeneous clusters of workstations. From our analysis of the software, we created a simple entity-level model for a small subset of functionality available in the IMMSIM simulator. Specifically, we model T-cell and B-cell movements and interactions. Using our simplified entity model, we instantiated environments with a realistic mixture of B- and T-cells and computed the corresponding initial task graphs. Using those graphs, we generated DAGs by applying the transformation described in Section III. From initial task graphs of 110 nodes, we obtained DAGs consisting of approximately 6500 vertices.

In the remainder of this section, we present results of sim-

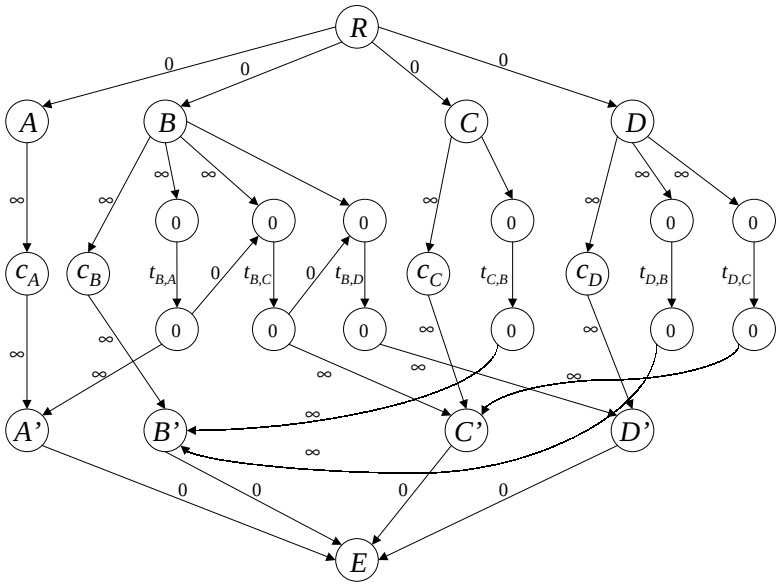

Figure 4. DAG Equivalent of the Application Model in Figure 1

ulations based on instances of our entity-level application model. The initial simulations were conducted for a collection of synthetic computational platform models composed of homogeneous resources. Our experimental results are discussed in Section IV-A. To investigate the possibility of using a Grid platform, we studied simulated application performance based on a model of an existing Grid testbed. These results are presented in Section IV-B.

A. Preliminary Results

To evaluate the utility of the DAG transformation, we based our initial study on instantiations of 11 basic computing platforms. The first 5 models describe single clusters with 2, 3, 4, 8, and 16 homogeneous computing nodes and a shared low-bandwidth local network. The next 5 models describe clusters with shared high-bandwidth local networks (with capacity 3 orders of magnitudes greater than that of the low-bandwidth networks).

Our results are based on a simulator built for those platform models using the Simgrid toolkit [2, 17]. In that simulator, we implemented numerous well-studied DAG scheduling algorithms (ETF [11], HLFET [16], HDLFET [16], and DLS [16]). We also implemented RAND, a scheduling algorithm which randomly assigns computational tasks to Grid resources, and RR, a scheduling algorithm which assigns tasks to Grid resources in a round-robin fashion. For each Grid model, we generated 60 randomized instantiations of our application model. We then simulated runs of each application instance for 1000 iterations with each scheduling algorithm. For each run and for each scheduling algorithm we compute a standard metric: the percentage degradation from the application execution time achieved by the best scheduling algorithm for that run [14]. A degradation of 0% means that the algorithm was best for that run. Table 1 shows average degradation from best over 60 runs for each platform model and for each scheduling algorithm.

A few key observations can be made from Table 1. First, the performance results obtained with the DLS and ETF algorithms are identical. A similar observation can be made for HDLFET and HLFET (except for 2-node clusters). This is due to the fact that our experiments use homogeneous computing resources. We opted for that approach as a first step in order to understand behaviors of the algorithms in simple environments. Another key observation is that both the RAND and RR algorithms are impractical in any scenario where more than 2 compute resources are available. This is due to the fact that they do not account for communication costs adequately, as can be observed in Table 1(a) for low-bandwidth networks. Naturally, that effect is not as pronounced for high-bandwidth environments as performance is not as heavily dependent on communication costs. The final observation is that the DLS and ETF algorithms seem to be the most appropriate in almost all scenarios.

These results show that (i) DAG scheduling algorithms yield a tremendous improvement over scheduling techniques that do not take into account the structure of the entity-to-entity communication patterns; (ii) among those algorithms DLS and ETF seem to be the most promising. In addition,

Table 1. Percentage degradation from the best application execution time (1000 iterations)

Cluster size	2-node	3-node	4-node	8-node	16-node
DLS/ETF	571.8	4.0	0.1	0.0	0.0
HDLFET	579.3	12.6	28.7	39.3	43.8
HLFET	4.5	12.6	28.7	39.3	43.8
RAND	2554.2	1124.5	1288.3	1680.3	2050.5
RR	2575.7	1133.0	1298.9	1696.3	2070.0

(a) Low-bandwidth cluster configurations

Cluster size	2-node	3-node	4-node	8-node	16-node
DLS/ETF	30.8	3.2	0.9	2.2	1.3
HDLFET	30.3	3.8	0.2	0.7	0.0
HLFET	39.7	4.1	0.2	0.7	0.0
RAND	0.6	7.6	20.7	43.2	44.6
RR	1.3	8.7	21.9	44.4	45.8

(b) High-bandwidth cluster configurations

these results justify our DAG transformation algorithm as it enables the use of numerous heuristics developed in the extensive DAG scheduling literature.

B. Results for GrADS Testbed Model

To evaluate the potential of our approach in a Computational Grid setting, we extended our simulation framework to include a model based on the testbed used by the Grid Application Development Software (GrADS) Project [1].[1] The GrADS resources included in our model are

- a cluster of eight Pentium III 550 MHz nodes, connected with a 100 Mb/s switch, located at the University of Tennessee, Knoxville;
- a cluster of 16 Pentium III 500 MHz nodes, connected with a 1 Gb/s switch, also located at the University of Tennessee, Knoxville;
- a cluster of 3 Athlon 700 MHz and 3 Pentium III 400 MHz nodes, connected with a 100 Mb/s switch, located at the University of California, San Diego; and
- a cluster of 8 Pentium II 266 MHz nodes, connected with a 100 Mb/s switch, located at the University of Illinois, Urbana-Champaign.

Based on our inspection of data transfer behavior between the two clusters hosted at the University of Tennessee, we concluded the network linking these clusters had a capacity of 100 Mb/s. We modeled the remaining links between the three sites using a snapshot of network bandwidth measurements

[1] Although the GrADS testbed contains some nodes with more than one processor, our model treats each of these as a single-processor node.

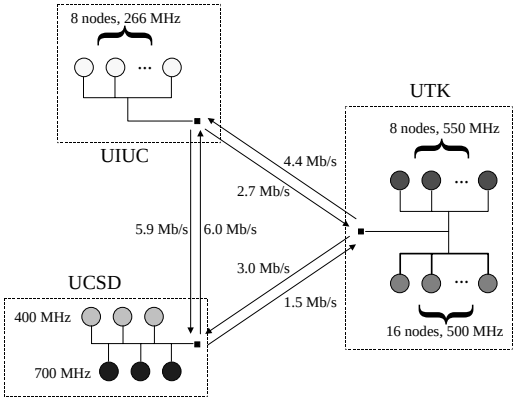

Figure 5. GrADS testbed platform model

corresponding to typical network conditions [3]. A depiction of the model we used is shown in Figure 5.

To evaluate the effect of using a distributed and heterogeneous computing platform, we modified the Simgrid-based simulation framework described in Section IV-A to use the GrADS testbed platform model instead of the 11 synthetic models previously used. The same set of six scheduling heuristics were considered: four DAG scheduling heuristics (ETF, HLFET, HDLFET, and DLS), RAND, and RR. We generated 500 randomized instantiations of our application model and simulated the application performance achieved with each of the scheduling heuristics. Table 2 presents re-

Table 2. Percentage degradation from the best application execution time for GrADS testbed model (1000 iterations)

Algorithm	Avg. % Degradation
DLS	0.0
ETF	19.9
HDLFET	18.0
HLFET	18.0
RAND	40.5
RR	19.9

sults of these experiments.

These results demonstrate the limitations of three of the four DAG scheduling algorithms considered in this paper. These algorithms assume that computational nodes are homogeneous and dedicated. Although our experimental results for the GrADS testbed do not address the issue of potential ambient computational load on processors, the computational capacities of nodes in this model vary. DLS was the only DAG scheduling algorithm designed to account for heterogeneity among computational resources, and was thus able to significantly outperform the other scheduling heuristics presented in this study. These results demonstrate that DAG scheduling algorithms which take into account platform heterogeneity are important if the DAG transformation strategy described in this paper is applied in a Grid environment. Moreover, we surmise that adapting such algorithms to also incorporate dynamic resource availability information may dramatically improve the quality of the resulting application schedules by accounting for the dynamic nature of Grid resources.

V. Conclusion and Future Work

Our initial experiments have proven encouraging, indicating that sophisticated scheduling heuristics may yield substantial performance gains for entity-level applications on the Computational Grid. In particular, our DAG transformation allowed us to leverage proven DAG scheduling heuristics, resulting in dramatically reduced execution times.

Our initial results suggest several directions for future work. Although strict synchronization at the end of every application iteration is a common requirement, the overhead costs of synchronization grow significantly as the number of entities increases. The potential for improved performance by relaxing synchronization constraints is great, despite the added complexity to the simulation model. For example, the DaSSF project [4] has recently proposed *composite synchronization* as a technique to synchronize less frequently based on application-specific characteristics.

We also plan to investigate other scheduling techniques which we can apply to entity-level applications. In particular, genetic algorithms have been shown to deal well with problems which are computationally difficult. To leverage this work, we plan to design a representation for entity-level application schedules as "chromosomes" for use in existing genetic models. We believe that this approach may yield effective application schedules at reasonable scheduling cost.

References

[1] Berman, F., Chien, A., Cooper, K., Dongarra, J., Foster, I., Gannon, D., Johnsson, L., Kennedy, K., Kesselman, C., Mellor-Crummey, J., Reid, D., Torczon, L., and Wolski, R. The GrADS Project: Software Support for High-level Grid Application Development. *International Journal of Supercomputer Applications 15*, 4 (2001), 327–344.

[2] Casanova, H. Simgrid: A Toolkit for the Simulation of Application Scheduling. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid* (May 2001).

[3] Dail, H. A Modular Framework for Adaptive Scheduling in Grid Application Development Environments. Master's thesis, University of California at San Diego, March 2002. Available as UCSD Tech. Report CS2002-0698.

[4] DaSSF webpage at `http://www.cs.dartmouth.edu/~jasonliu/projects/ssf/docs.html`, 2002.

[5] Epstein, J., and Axtell, R. Artificial Societies and Generative Social Science. In *Proceedings of the First International Symposium on Artificial Life and Robotics* (Feb 1996).

[6] Erol, K., Levy, R., and Wentworth, J. Application of Agent Technology to Traffic Simulation, `http://www.tfhrc.gov/advanc/agent.htm`, 1998.

[7] Foster, I., and Kesselman, C. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.

[8] Foster, I., Kesselman, C., and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications* (2001). to appear.

[9] Gilbert, N. Simulation: an Emergent Perspective, `http://www.soc.surrey.ac.uk/research/simsoc/tutorial.html`, 1996.

[10] Grimm, V. Ten years of individual-based modelling in ecology: What have we learned, and what could we learn in the future? *Ecological Modelling 115* (1999), 129–148.

[11] Hwang, J.-J., Chow, Y.-C., Anger, F. D., and Lee, C.-Y. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal on Computing 18*, 2 (Apr 1989), 244–257.

[12] IMMSIM webpage at `http://www.cs.princeton.edu/immsim/`, 1999.

[13] Kleinstein, S. H., and Seiden, P. E. Simulating the Immune System. *Computing in Science and Engineering* (Jul–Aug 2000), 69–77.

[14] Kwok, Y., and Ahmad, I. Benchmarking and Comparison of the Task Graph Scheduling Algorithms . *Journal of Parallel and Distributed Computing 59*, 3 (1999), 381–422.

[15] Oh, J.-S., Cortets, C., Jayakrishnan, R., and Lee, D.-H. Microscopic Simulation with Large-network Path Dynamics for Advance Traffic Management and Information Systems. In *Proceedings of the 6th ASCE International Conference on Applications of Advanced Technologies in Transportation Engineering* (Jun 2000).

[16] Sih, G. C., and Lee, E. A. A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. *IEEE Transactions on Parallel and Distributed Systems 4*, 2 (Feb 1993), 175–187.

[17] Simgrid webpage at `http://grail.sdsc.edu/projects/simgrid/`, 2002.

[18] Ullman, J. NP-complete scheduling problems. *Journal of Computer and System Sciences 10* (1975), 434–439.

[19] Wu, M.-Y., and Gajski, D. Hypertool: A Programming Aid for Message-Passing Systems. 330–343.

[20] Yang, T., and Gerasoulis, A. DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors. *IEEE Transactions on Parallel and Distributed Systems 5*, 9 (Sep 1994), 951–967.