분산 및 병렬 알고리즘 시뮬레이터

o
서영진, R.S.Ramakrishna
광주과학기술원 정보통신공학과
peacemak@geguri.kjist.ac.kr, rsr@kjist.ac.kr

Distributed/parallel Algorithm Simulator

o
Yung-Jin Suh and R.S.Ramakrishna
Department of Information and Communications
Kwang-Ju Institute of Science and Technology(K-JIST)
peacemak@geguri.kjist.ac.kr, rsr@kjist.ac.kr

ABSTACT

A new distributed/parallel algorithm simulator, DASim(Distributed Algorithm Simulator), is proposed in this paper. The idea is to ease the task of design, analysis and implementation of distributed algorithms. A small high level language has been proposed for the purpose. Through this non-language specific high level language, the users are spared from the tedious details about how to program distributed or parallel algorithms. Further, visualization of these algorithms are pretty helpful to understand behaviors of these algorithms.

1. Instruction

When designing, debugging, and understanding computational algorithms, it is quite helpful to visualize the results of algorithms. Although a lot of work has already been done in this research area[1,2,3], there are not many effective simulators. A distributed/parallel algorithm simulator comes with attractive attributes. Programming on simulator can be free from any specific programming language. Users do not need to know the whole bothersome syntax of a language or functions in the library. Even users who do not know how to program a distributed or parallel algorithm, can simulate these algorithms with the help of higher level concepts: No knowledge of network programming is needed. A small language for describing algorithms that runs on an integrated simulator, named DASim has been proposed in this paper.

2. A small Language for Distributed/parallel Algorithm Description.

To simulate distributed/parallel algorithms we employ a small language. This front-end affords easy to debugging and use. Fig.1 shows the structure of proposed language.
With an eye on economy, only basic mathematical and message passing operations are provided. It is however, possible to define extraneous macros through the *imports* part. The system has been developed using Java. Hence, all the pre-defined Java classes are available for use in this part. Some

```
imports :: ...
topology :: ...
algorithm :: ...
variables :: ...
messages :: ...
initiator :: ...
{...}
noninitiator ::
{...}
```

Fig. 1. small language template

constructs for sending a message, receiving a message, broadcasting a message, and decision making have already been included for convenience. Any topology can be simulated on DASim. *Topology* part only contains ascii-file name or predefined topology name. This file contains information about links and starting configurations such as initiator, weight etc.

Algorithm is used just to specify the name of program. As DASim is basically simulates message passing algorithms, *variables* part declares variables located in individual processors. *Messages* part is used to define messages passed among processors. *Initiator* and *noninitiator* parts are the cores of the template. They contains the algorithm which users want to execute. *Initiator* part is run by all initiators, while *noninitiator* part is run by all non-initiators. We have tried to make it as easy and simple as possible. As we mentioned previous, user can employ pre-defined functions to pass messages and to terminate a program. To implement this part, JLex[5] and Java_Cup[6] have been employed. JLex DFA based lexers are usually faster than hand written lexers and LALR(1) parser generated by CUP is usually faster than PCCTS LL(k) parsers[6]. The most important reason for choosing them is both are available in source code.

3. DASim

Distributed/parallel algorithm simulator (DASim) is very flexible and customizable. Any distributed algorithm can be simulated on any topology. DASim consists of three main parts, configuration, distributed algorithm execution, and visualization part.

3.1 Configuration part

The configuration part indicates the network configuration such as topology, nodes, and links, the execution configuration, and visualization configuration. In particular, the simulator accommodates flexibility at compile time. The topology is fixed during simulation, however. Plans are afoot to incorporate run time flexibility. Only one link between two nodes is permitted. Links can be either bi-directional or uni-directional, and can be weighted.

3.2 Execution part

Distributed algorithm execution part is based on transition system with message passing. Fig. 2. shows its structure. Each node has a message queue. This queue serves messages basically according to a FIFO(first-in-first-out) strategy. The Node_set controls all of the nodes that participate in an algorithm. It creates all the processes and indicates initiators to begin an algorithm. Watcher serve as terminator. In distributed system there is no real global clock to check when an action such as message passing occurred. We use a logical clock[4] to record all the events that occur during simulation. Messages passed between nodes can be of any type and size(subject to certain constraints). Asynchronous messaging is assumed throughout.

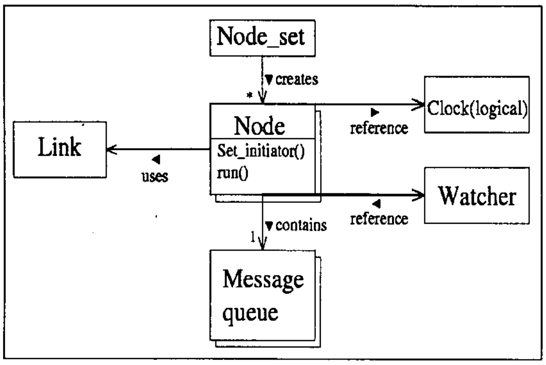


Fig. 2. structure of execution part

3.3 Visualization part

The visualization part exhibits the data and the messages. Basically DASim provides source code animation view, Lamport view[4], and message passing view. Fig. 3. is an example of Lamport view.

Visualization part is divided into two modules. One of them is a modeler and the other is a renderer. The main role of the modeler is to collect all the information about what to display and analyse them. Tenderer actually shows the data and the messages. The views provide a precise and an exact portrait. As the number of processors increases, higher display abstraction would be required. DASim

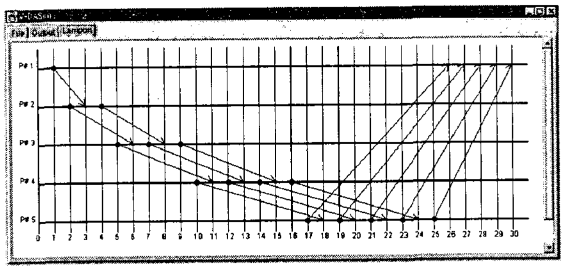
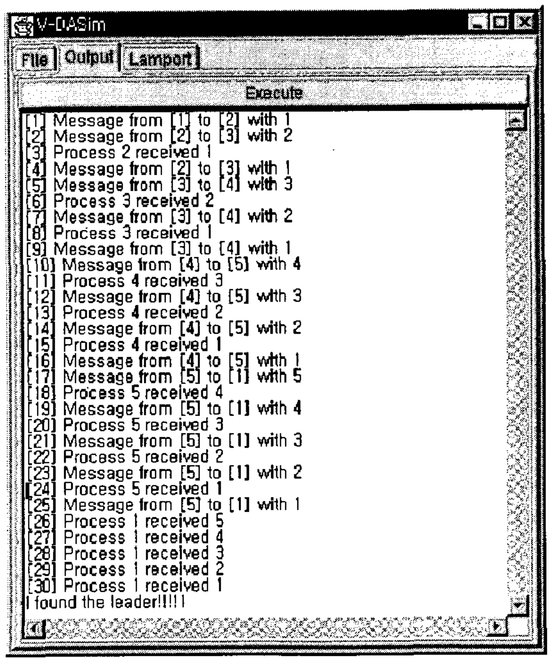


Fig. 3. Lamport view

provides a kind of level abstraction and scrolling. We also tried to separate visualization codes from algorithm codes, so that users can concentrate on programming without being disturbed by visualization. The visualization configuration can be set up not only before execution but also during execution. Graphic and tex outputs have been provided. Fig. 4. is an example of text output.

4. Conclusion and Future Work

Using text output alone to understand the behavior of systems with multiple concurrent flows of control is tedious and hard. A new simulator has been proposed to ease the burden for distributed/parallel algorithms. To test distributed algorithms and to watch the progresses, DASim would provide invaluable assistance. DASim just simulates asynchronous algorithms at present. It will be possible to simulate synchronous ones on it in the near future. DASim simulates only one algorithm at any given time, but running a couple of algorithms at the same time to compare the performance and behavior of different algorithms would be valuable.



```
[1] Message from [1] to [2] with 1
[2] Message from [2] to [3] with 2
[3] Process 2 received 1
[4] Message from [2] to [3] with 1
[5] Message from [3] to [4] with 3
[6] Process 3 received 2
[7] Message from [3] to [4] with 2
[8] Process 3 received 1
[9] Message from [3] to [4] with 1
[10] Message from [4] to [5] with 4
[11] Process 4 received 3
[12] Message from [4] to [5] with 3
[13] Process 4 received 2
[14] Message from [4] to [5] with 2
[15] Process 4 received 1
[16] Message from [4] to [5] with 1
[17] Message from [5] to [1] with 5
[18] Process 5 received 4
[19] Message from [5] to [1] with 4
[20] Process 5 received 3
[21] Message from [5] to [1] with 3
[22] Process 5 received 2
[23] Message from [5] to [1] with 2
[24] Process 5 received 1
[25] Message from [5] to [1] with 1
[26] Process 1 received 5
[27] Process 1 received 4
[28] Process 1 received 3
[29] Process 1 received 2
[30] Process 1 received 1
I found the leader!!!!!
```

Fig. 4. Example of text output

References

[1] M. H. Brown and R. Sedgewick. A System for Algorithm Animation. Computer Graphics, 18(3):177-186, July 1984.

[2] J. A. Kohl and G. A. Geist. The PVM 3.4 Tracing Facility and XPVM 1.1. In H. El-Rewini and B. D. Shriver, editors, Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences, vol. 1, pages 290-299, 1996.

[3] J. T. Stasko and E. Kraemer. A Methodology for Building Application Specific Visualizations of Parallel Programs. Journal of Parallel and Distributed Computing, 18:258-264, 1993.

[4] L. Lamport. Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21, 558-564, 1978.

[5] Elliot Berk. JLex: A lexical analyzer generator for Java(TM), 1997.

[6] Scott E. Hudson. CUP parser generator for Java, 1997.