Distributed Algorithms Visualisation for Educational Purposes

Boris Koldehofe
Computer Science Department
Saarlandes Un., 66123 Saarbrücken, Germany
& Chalmers Un., 412 96 Göteborg, Sweden
Email: khofer@cs.chalmers.se

Marina Papatriantafilou Philippas Tsigas*
Computing Science Department
Chalmers Un. of Technology and Göteborg Un.
S-412 96 Göteborg, Sweden
Email: (ptrianta,tsigas)@cs.chalmers.se

Abstract

We present our work on building interactive continuous visualisations of distributed algorithms for educational purposes. The animations are comprised by a set of visualisation windows. The visualisation windows are designed so that they demonstrate i) the different behaviours of the algorithms while running in different systems, ii) the different behaviours that the algorithms exhibit under different timing and workload of the system iii) the time and space complexities of the algorithms and iv) the "key ideas" of the functionality of the algorithms. Visualisations have been written for a set of 10 algorithms that are tought in a Distributed Algorithms advanced undergraduate course.

1. Introduction

Distributed algorithms are by nature complicated to understand and to teach. Threads of control compete for resources, try to synchronise and dynamically change execution behaviour. Each execution involves many processes, a large amount of data of processes' local state to describe the system state and an even larger amount of date to describe complex interactions between the processes. Moreover, different executions of the same algorithm, even if they start from the same initial system configuration, may not result in the same output, due to asynchrony.

Animation of distributed algorithms graphically shows an execution of the distributed algorithm given. It can assist the teacher to illuminate the description of a distributed algorithm (including its time and communication complexity analysis) and to graphically show material that she/he has explained on the board.

This paper is on our work in building animations of distributed algorithms to demonstrate *(i)* the "key ideas" of the functionality of the algorithms, *(ii)* their behaviour under different timing and workload of the system, *(iii)* their communication and time complexities. Each visualisation is comprised by a set of visualisation windows, each one of them demonstrating a specific aspect of the execution of the algorithm; we call these visualisation windows *views*. The visualisation takes as input any possible execution trace of the respective algorithm, so that students (users) can view it in any possible execution that they can select. We propose the use of a set of views, which also take into account two inherent difficulties in understanding distributed algorithms executions. These difficulties stem from the absence of *global time* in the system, which implies:

- that processes need to rely on their knowledge of *causal relations* among events in the system,
- and that in order to measure the length of an execution in time, we need to employ some mechanism related to the *dependencies* induced by each algorithm.

In the next section we describe the set of views that we provide for each animation and also motivate our decisions, by explaining the role each one plays in assisting the understanding of the algorithms. The code for all but one ("special") view is modularly used by all algorithms, as they are to assist in understanding issues which are common in all distributed algorithms. The idea behind the "special" view is to illustrate the *special concepts* for each algorithm (therefore the view needs to be different for each algorithm).

For our animation programs we use the Polka library [9], which is highly portable, friendly to use and has very good features for visualisation, including possibility for multiple views, speed tuning, step-by-step execution and callback events to assist interactive animation.

This effort is within our project Lydian, which involves the development and optimisation of an integrated environment to enable *(i)* development and maintainance of an archive of distributed algorithms and protocols (together with their animation programs), *(ii)* development and main-

*Contact author, phone: +46 31 7725409, fax: +46 31 16 56 55

tenance of an archive of system specification/configuration files, to specify different types of systems to be used for the simulation and animation of the algorithms, and *(iii)* simulation of the algorithms in appropriate systems, in order to produce traces of a variety of executions, which can be used for the animation and for the evaluation of the algorithms.

2 Related Work

To the best of our knowledge, there has been only one attempt towards a a set of animations of distributed protocols for educational purposes, ZADA [8], on the animation package Zeus, a Modula-3 based system for specialised platforms, —not as highly portable. The effort resulted in a small archive of protocols, for each of which the set of views is fixed and the implementation is the same program as the animation (this implies essentially fixed timing, workload, etc). Of relevance is also a nice work described by Ben-Ari in [1], where the focus is on the choice of a language to be used for implementations of distributed algorithms for demonstration and laboratory exploration.

3 The Animation Views

We have decided to offer the same views to the user, we think that this not only helps the user to get familiarity with the tool but the views that are offered are essential for almost all distributed algorithms that we know. All views evolve continuously as the execution of the algorithm evolves (continuous motion). The user can decide on-line which views he/she would like to see at any time. The views can be selected by a menu window. Also a further control window enables the user to change the speed or even pause animations in order to watch interesting parts or skip uninteresting parts of the algorithm's execution. Moreover the user has the possibility to zoom into interesting parts of the animations as he/she can move to any area of a view. This is important since by nature some animations will not be able to take place in a bounded window frame because the animator does not have any previous knowledge of further executions of the algorithm. With exception of the basic view, in which an individual animation for each algorithm was developed, the offered views were designed such that they are transferable for any distributed algorithm for a message passing system although they allow some specifications. Thus further development will have to concentrate only on the main ideas of algorithms. Below we describe in detail the views that we have decided to offer.

The accompanying figures illustrate a snapshot of the animation of an execution of the ECHO algorithm (broadcast with acknowledgements) [11]. The problem and the algorithm are as follows: One process(or) needs to broadcast a message to all the others and to also know when all have received it. It can only communicate with its neighbours in the network, so it sends the message to them. Each process, upon receiving the broadcast message for the first time, it propagates it to its other neighbours and it waits to receive acknowledgements from all of them before sending its own acknowledgement to the one where it got the message from for the first time. Any process receiving the broadcast message again acknowledges immediately to that sender and does not propagate it again.

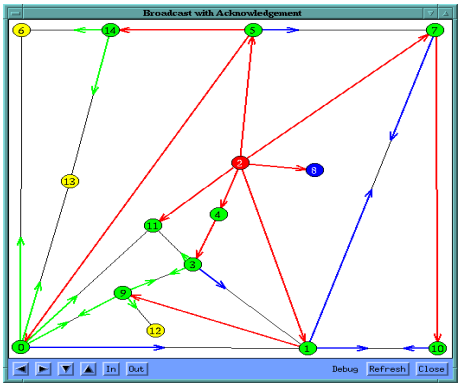

Figure 1. Basic view of the broadcast-with-acknowledgements algorithm animation

Basic View (cf. Fig 1) It illustrates the basic idea of the algorithm, hence it can look quite different for different algorithms. However, for many algorithms it is of interest to see the state of processes and messages which are sent along links. This can be achieved by showing the communication network and coloring its nodes (processes) according to their state and showing resizing arrows which are coloured according the kind of message sent along an edge (link). In the particular algorithm it shows the communication network, the propagation of the broadcast and the acknowledgement messages (arrows in green and blue respectively) and colors (green or blue) the nodes (processes) that have received the broadcast message and/or the acknowledgements, accordingly. Initially all nodes are yellow, except from the one that initiates the broadcast, which is always shown in red. As the algorithm execution evolves, *waiting chains* are formed among processes (each process in the chain waits for an acknowledgement from its next one in the chain); these chains also determine the *time complexity* of the algorithm. The edges that involved processes in the chains are marked in red (in this particular algorithm they also form a spanning tree of the network at the end).

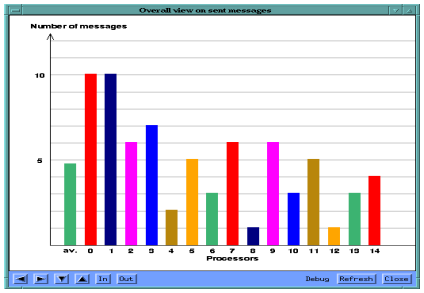

Figure 2. View showing the communication induced by each process(or) and the average figure

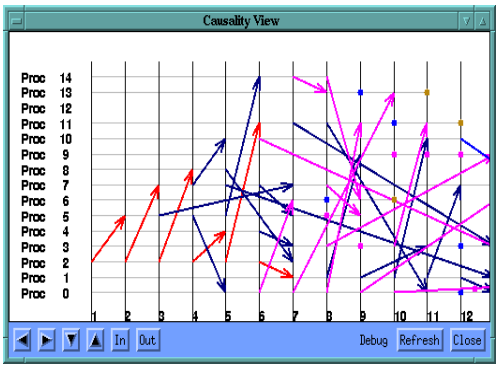

Figure 3. View showing the causality and logical times (e.g. as would be seen by a monitoring process)

Communication View (cf. Fig. 2) This view assists in measuring the communication complexity of the algorithm and is often helpful in finding relationships between communication complexity and the structure of the communication graph. It shows the contribution of each process(or) in the traffic (messages) induced by the algorithm's execution and it also shows the average number of messages per process(or) during the execution. The number of messages are displayed in a bar chart where bars grow online with the number of messages sent by a process(or). In this example it is easy to observe that the amount of traffic that each process(or) is responsible for is proportional to its degree in the communication graph (shown in figure 1).

For some algorithms it is also of interest to have a measure of the bit complexity of messages. The actual known maximum size of a message (represented in bits) is displayed below every processes bar. The size of a message is represented by a circle whose area content is proportional to its message size. As the message size increases online, the user is able to observe how fast message sizes are increasing. In our example algorithm the bit complexity of a message was constant so that the bit complexity is not of any interest.

Causality View (cf. Fig 3) It illustrates the causal relation between events in the system execution (arrows represent message transmission). It also shows how the processes logical clocks are incremented during the execution. Even though logical clocks are not used in all algorithms, the view is always available; its purpose is to show how would a monitoring process view the execution, based on traces as would be given by each process separately. This is important, as the processes in a distributed system do not have global knowledge of time. Besides, as consecutive causally related events change color, overlapping arrows with different colors visualise the degree of asynchrony in the execution. It should be noted that showing the maximum directed path in the resulting graph shows the length of the execution in units of message transmission times.

Naturally, only part of the whole view can be shown in the window, but the user is able to go back and return (as well as to zoom in and out), like with all other views.

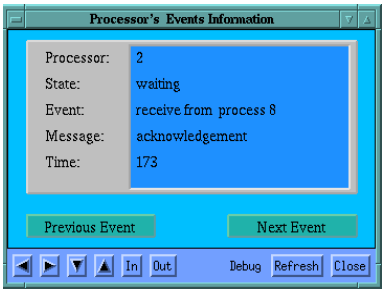

Figure 4. Process(or) step view

Process Step View (cf. Fig. 4) This gives the user the possibility to click on any node in the basic view window, to get information about its status (state, last event processed, last message received/sent, ...) at any point during the animation (or even after it has completed). It is also possible to view *interactively* the whole execution of the selected process. This can be done for any process in the system.

Process Occupation View (cf. Fig. 5) It shows in actual times (i.e. as given by the simulation trace) the period that each process is kept busy by the algorithm during the animated execution. If it is required by the algorithm it is also possible to distinguish how long a process was kept busy in a certain state. Therefore a user may come to a better understanding of the algorithm's time complexity by retracing

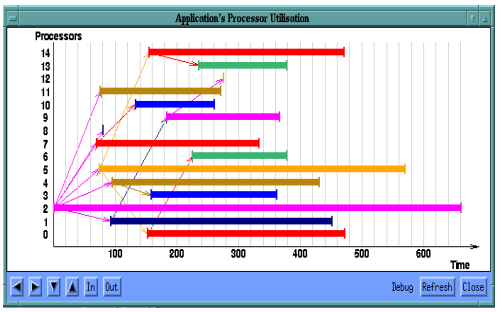

Figure 5. Process(or) occupation view

e.g. with the Process Step View why a specific process was kept busy for a long time. In this example, it can be easily observed that the initiator of the broadcast is the first to start and the last to finish; by getting the acknowledgements from its neighbours —i.e. its children in the induced spanning tree— it knows that the broadcast message reached everybody, hence it terminates.

4 The Implemented Visualisations

Visualisation programs (each one comprised of the above mentioned views) have been written for ten distributed algorithms:

- Broadcast algorithm [11, 7]
- Broadcast ECHO algorithm [11]
- Ricard-Agrawala's mutual exclusion [10]
- Chandy-Misra's dinning and drinking philosophers [2]
- Luby's maximal independent set [6]
- Choy-Singh's three resource allocation algorithms [3]
- Gallager-Humblet-Spira's minimum weight spanning trees [5]
- A periodic counting network isomorphic to the Dowd-Perl-Rudolph-Saks' network [4]

The above mentioned algorithms cover one big part of what it is taught in a distributed algorithm course. We expect to double the number of algorithms that are being visualized in the near future. The resource allocation algorithms are taught in operating systems courses as well as in distributed systems courses.

Conclusion

In this paper we provide an overview of progress that has been made in designing homogeneously interactive visualisations of distributed algorithms that will help the students to visualize the time and space complexities of the algorithms and at the same time will show the "key ideas" of the functionality of each algorithm. In section 2 we summarised the approach that we followed. We will use the animations this year in class together with the lab assignments. Furthermore, we expect that the feedback from the students will help us improve the visualiztions at the basic view parts of the algorithms.

References

[1] M. Ben-Ari "Distributed Algorithms in Java" *ACM Conference on Integrating Technology into Computer Science Education – ITiCSE '97*, p. 62-64, 1997.

[2] K.M. Chandy and J. Misra. "The Drinking Philosophers Problem" *ACM TOPLAS*, Vol. 6, No. 4, pp. 632-646, Oct. 1984.

[3] M. Choy and A. Singh. "Efficient Fault Tolerant Algorithms for Resource Allocation in Distributed Systems." *ACM TOPLAS*, Vol. 17, No. 3, pp. 535-559, May 1995. (Also in *Proc. of ACM STOC 1992*, pp. 593-602).

[4] M. Dowd, Y. Perl, M. Saks and L. Rudolph "The balanced sorting network" *JACM*, 1991.

[5] R.G. Gallager, P.A. Humblet and P.M. Spira "A Distributed Algorithm for Minimum Weight Spanning Trees" in *ACM TOPLAS*, Vol.5, No.1, January 1983, pp. 67–77.

[6] M. Luby "A simple parallel algorithm for for maximal idependent set problem" *SIAM Journal of Computing*, **15(4)**:1036-1053, November 1986.

[7] N. Lynch "Distributed Algorithms" *Morgan Kaufmann*, 1996.

[8] A. Mester, P. Herrmann, D. Jager, V. Mattick, M. Sensken, R. Kukasch, A. Ritter, S. Bunemann, P. Unflath, M. Bernhard, F. Austel, T. Alders, A. Rohrbach "ZADA: Zeus-based animations of distributed algorithms and communication protocols" *T.R. Universität Dortmund*, 1995.

[9] John Stasko "POLKA Animation Designer's Package" *Technical Report*, Georgia Institute of Technology, 1995.

[10] G. Ricart, A. K. Agrawala "An Optimal Algorithm for Mutual Exclusion in Computer Networks" *Communications of the ACM*, January 1981, Volume 24, Number 1, pp. 9-17.

[11] G. Tel "Introduction to Distributed Algorithms" *Cambridge University Press*, 1994.