Algon: A Framework for Supporting Comparison of Distributed Algorithm Performance

Karen Renaud
Department of Computing Science
University of Glasgow
karen@dcs.gla.ac.uk

Johnny Lo Judith Bishop
Pieter van Zyl Basil Worrall
Department of Computer Science
University of Pretoria

Abstract

Programmers often need to use distributed algorithms to add non-functional behaviour, such as mutual exclusion, deadlock detection and termination, to a distributed application. They find the selection and implementation of these algorithms daunting. Consequently they have no idea which algorithm will be best for their particular application. To address this difficulty the Algon framework provides a set of pre-coded distributed algorithms for programmers to choose from, and provides a special performance display tool to support choice between algorithms. The performance tool is discussed in this paper.

The developer of a distributed application will be able to observe the performance of each of the available algorithms according to a set of of widely accepted and easily-understandable performance metrics and compare and contrast the behaviour of the algorithms to support an informed choice. The strength of the Algon framework is that it does not require a working knowledge of algorithmic theory or functionality in order for the developer to use the algorithms.

1 Introduction

Many programmers find developing distributed applications an overly complex task [14]. In addition to the normal intensively demanding cognitive activity involved in working with abstract tasks during programming, they have extra concerns to deal with such as the non-determinism, contention and synchronisation issues of distributed systems [18].

One of essential issues that many programmers find daunting is the use of distributed algorithms in their systems. For example, distributed systems often need to ensure that distributed mutual exclusion is maintained with respect to a particular resource, or need to incorporate a distributed deadlock detection element into the system. Distributed algorithms can be categorised according to functionality, and within each category various tried and tested algorithms are widely available and have been tailored to distribution-specific problems [26, 27]. Each algorithm achieves the expected result in a different way and with different performance characteristics. Whilst programmers may understand the superficial functionality of the algorithm, a far deeper understanding is required in order to critically evaluate these algorithms and to make an informed decision about the correct algorithm to use for any particular system.

Another problem is the fact that programmers may not sufficiently understand the nature and needs of the distributed system itself. An application may need to employ distributed mutual-exclusion, snapshots or deadlock-detection algorithms but the algorithms in everyday use are often centralised and inappropriate for distributed systems [26].

Initiating all programmers into the mysteries of algorithms is neither viable nor tenable. A far better alternative is to provide the algorithms as reusable software components, and then to provide *support for choice* between algorithms offering the same functionality. The use of software components in this way allows us to separate the algorithm from the application so that the algorithm can be independently tested, configured, upgraded or replaced. This approach is in stark contrast to the usual practice of interspersing the algorithm with the application code, which does not give the programmer this degree of flexibility.

The Algon[1] system provides a framework for incorporating algorithmic software components into a distributed system. It reduces the complexity related to distributed algorithmic concerns in a distributed system, and supports programmers in finding the best possible algorithm for their system without the arduous and time-consuming task of mastering the algorithms. Algon includes:

- a library of algorithms be to be used as and when re-

[1] Algon stands for *Algorithms On the Net.*

quired;

- a tool for selecting;
- a framework for incorporating said algorithms into the system;
- a tool for depicting the performance of an algorithm and for comparing this performance to that of another algorithm in the same family.

Many programming languages routinely provide libraries of mathematical functions to simplify the programmer's task. Following the same principle Algon provides programmers with a store of *algorithms* so that they can experiment and thereby arrive at the algorithm with the best performance for their particular application.

We previously reported on the structure and rationale of the Algon framework [4], and on the role of Java in facilitating the framework [5]. This paper focuses on further work which has been done in order to extend Algon by including a performance visualisation aspect to support a choice between algorithms. Section 2 discusses issues related to reporting on algorithm performance in order to support choice between algorithms. Section 3 briefly reviews the Algon structure to demonstrate how the software components are accommodated. Section 4 discusses aspects relating to the visualisation of algorithm metric measurement. Section 5 gives details of the implementation of the performance measurement and display tool. Section 6 compares Algon to related work in the field. Section 7 concludes.

2 Performance Measurement and Reporting

The measurement of software performance *by and for experts* is a well-known task [12, 25] and dedicated algorithmicists have developed techniques to judge the performance of algorithms [23, 2]. One technique, competitive analysis, compares an algorithm to a powerful adversary on a worst-case input [17]. Koutsoupias and Papadimitriou report that this technique is less than optimal since it fails to discriminate and does not suggest good approaches. Clearly there is a need for a better way to compare algorithms. In distributed systems it becomes even more difficult due to the non-deterministic nature of the systems. Konkin *et al.* [16] claim that there has been very little progress in providing programmers with tools to tune their applications in the last 15 years. This implies that tuning mechanisms have not kept up with developments in the distributed domain.

Algorithmic comparison techniques and results, while adequately recorded in academic publications, appear to be inaccessible to the average developer. They do not understand the various performance implications and expecting them to do so is unrealistic and unnecessary. Much complexity is already successfully hidden from developers and they are none the worse for it. Up to a few years ago the average developer had no need to be concerned about complicated distributed algorithms but with more and more developers entering the distributed arena it is no longer possible to shield them entirely.

Algon was developed with the express aim of making algorithms more accessible to the multitude of uninformed developers currently struggling with these concepts. Algon does this by building a bridge between the intensely theoretical world of algorithmicists and the pressured practical world developers inhabit. Algorithmicists readily admit that their experiments with algorithms and their performance are often unnatural and impractical [17]. Aspnes and Waarts [1] review the extensive work done on optimality of algorithms given certain conditions such as a particular sequence of messages or a particular sequence of failure events. Such assumptions allow academics to study particular problems but may not be relevant in a real-life setting.

Algon aims to give developers an insight into the differing performance of algorithms in a particular system — without requiring a deep understanding of the algorithms themselves. We therefore provide a visualisation of performance measurement data in such a way that a non-expert can easily understand and interpret it and we thereby support the making of an informed decision with respect to the use of one algorithm or another. This type of performance visualisation can provide information that cannot be obtained in any other way since it reports on algorithms in action, in a real-life application, and it may be that algorithmicists would also benefit from using this visualisation.

3 The Algon Concept

Middleware frameworks have been shown to be viable for the deployment of software components on the Internet [8]. However, they tend to require a certain amount of tailoring of the application and the introduction of new methods. The problem with the wide-scale deployment of components over the Internet is that the frameworks and wrapper interfaces each have to be custom-made for a given system. This is wasteful in terms of time and money and is doomed to failure if the programmers do not understand the functioning of the legacy software.

The objective of Algon is to produce a library of pre-coded distributed algorithms which can be incorporated into a distributed component-based application while maintaining a clear separation between components. Having decided to treat distributed algorithms as a candidate separate concern, the next step is to develop a mechanism for structuring the base code so as to enable this separation [21]. Algon's algorithms have been implemented in such a way that the code is split into logical processes, enveloped within a distributed system and augmented with a pre-coded distributed

algorithm. The resulting component can then be added to a distributed system with the minimum of changes to the original code of the application. To illustrate the concept, consider an application, P, as shown Figure 1(a), using a classic algorithm, C, to achieve some required behaviour. Example applications are shown in Figure 1(b) & (c). In example (b) there is a resource, RS, which requires mutual exclusive access. In (c) there are two devices whose usage has to be coordinated.

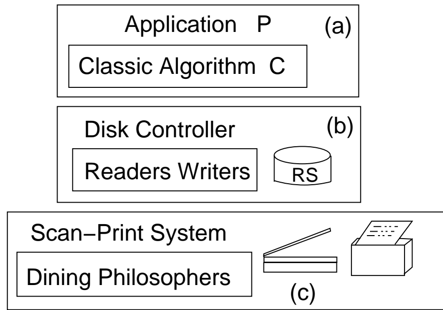

Figure 1. *Centralised View of an Application*

Consider, for example, the situation where there is a need to *distribute* the behaviour previously provided by the algorithm C. The programmer would traditionally have to re-code the algorithm and incorporate it into the application *in place of* C. Algon provides a mechanism whereby the distribution features can be *added* to the application in the form of a component.

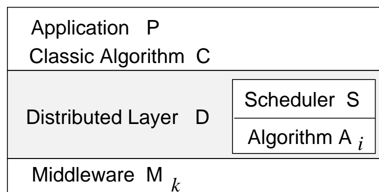

Figure 2. *Distributed View of the Application using an Algon component.*

The proposed architecture for an application system is shown in Figure 2(a). The application-specific code, P, and classic algorithm, C, remain unchanged. Figure 2(b) illustrates how this architecture might be implemented on a system with three nodes. In order to distribute C's behaviour, the system is extended by adding:

1. a distribution layer, D, which consists of:

   (a) a scheduler S, and

   (b) an algorithm A_i.

   The distribution layer, D, is selected specifically to match the algorithm, C.

2. a middleware backbone M_k, which facilitates communication with other participants. It can use any suitable communication structure such as Java RMI, CORBA IIOP, DCOM or .NET.

Our intention is to provide developers with a range of distributed algorithms for any particular problem. To make these algorithms easily interchangeable, a standard interface is implemented for specific types of algorithms. For a specific classic problem i, the interface I_i is used by the scheduler to interact with all algorithms implementing that interface. This makes it easier to introduce new algorithms and to specify, at runtime, the algorithm that should be used. An example of this is the ME (Mutual-Exclusion Group of Algorithms) interface being implemented by the Ricart-Agrawala mutual exclusion algorithm [26] (Figure 3). Another example of this could be where the DD (Deadlock-Detection Group of Algorithms) interface being implemented by the Chandy-Misra OR model deadlock-detection algorithm [7]. The developer can choose the algorithm to be used by means of a graphical user interface, and also indicate the identities of the nodes involved in the distributed system. This makes the algorithms interchangeable without any need for recompilation.

To illustrate how Algon works, consider the classic problem of a disk-controller application with various readers and writers. Assume that it is necessary to allow distributed readers and writers to access the disk contents simultaneously — hence requiring the services of a distributed mutual-exclusion algorithm. For the purposes of this discussion we will use the Ricart-Agrawala algorithm [24].

An example of the Algon approach is shown in Figure 3. Two nodes have readers, while a third has a writer, with the third node holding the shared resource. The reader or writer will invoke methods on the scheduler, S, in order to acquire permission to read or write. The scheduler holds a reference to the algorithm, RA, which implements the ME interface. The algorithm will use the middleware backbone to communicate with other readers and writers as required.

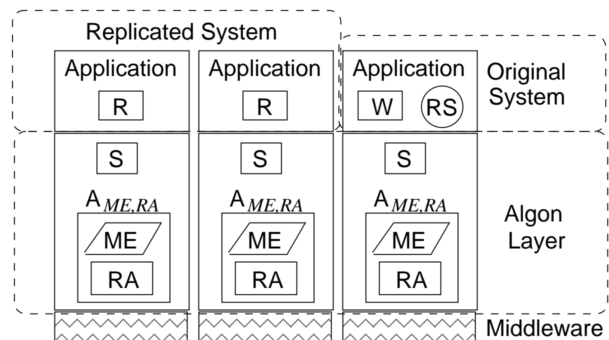

Figure 3. *Using Algon to add a distributed mutual exclusion algorithm*

This section has given a brief overview of Algon. The next section will discuss issues pertaining to the measurement and visualisation of algorithm performance so that developer choice can be facilitated.

4 Performance Metrics

Portraying the performance of algorithms *to the uninitiated* is not trivial. For example, the Ricart-Agrawala algorithm for mutual exclusion [24], which will be used to illustrate our approach in this paper, involves distributed sites sending requests and replies to other participating sites, comparing timestamps, and keeping queues of waiting sites. Other algorithms provide better performance with even greater complexity. Systems *do* exist for illustrating the functioning of algorithms [3, 6, 15] but the primary function of these systems is in supporting education and research and they are not intended to be components for development.

It is therefore desirable to quantify certain aspects of algorithmic performance so as to be able to compare algorithms — and not attempt to compare actual *mechanisms* for achieveing their purpose. Various metrics can be used to measure performance of algorithms in distributed systems [1, 12, 19, 26], for example: (1) response or waiting time; (2) synchronisation delay; (3) number of messages exchanged; (4) throughput; (5) communication delay; (6) node fairness; (7) CPU cycle usage; or (8) memory usage.

Selected performance metrics need to be generic enough to be applied to a variety of algorithms, and straightforward enough to be understood by developers. We need to ensure that the developer can obtain metrics which reflect the needs, priorities and workloads of the particular distributed system [22], since no single metric can be optimal for all applications [11].

For the initial prototype we chose to measure performance using only the first four metrics. These were most suited to specifically measuring *algorithm* performance. The fifth metric is more dependent on network load than a specific algorithm, the sixth difficult to quantify and the seventh and eighth produce measurements of debatable merit in judging algorithm efficacy. Experience may suggest other metrics and the interface has therefore been designed in such a way that the inclusion of additional metrics is trivial.

Visualisation is a powerful tool which is being used increasingly to enhance understanding in a variety of different application areas [28]. Visualising algorithm performance in a simple and understandable manner merits some consideration. To provide a visualization of this data one has to consider the *granularity* and *nature* of the data to be depicted. One could choose to collect and display immediate *up-to-the-minute* data, showing process state as it enters and leaves critical sections, for example. On the other hand, it may be more useful to collect *cumulative data* which tracks the progress of the system and depicts it in a form that shows the variations in the system's performance. In terms of supporting comparisons, *averaged data* over a whole time period — perhaps for a session, or a certain time period, could conceivably be the most useful type of data. The cumulative and averaged displays can depict either one algorithm's performance or differences between the performance of algorithms in a visual format thus supporting and facilitating an informed decision-making process.

Each of these data types contribute something different to the decision-making process and we are loath to deprive the developer of any so we provide all of them. While many tools concentrate on providing *one* of the three perspectives few if any provide all at the same time, as Algon does. The following visualisations were employed:

1. Dynamic data about the state of the nodes participating in the distributed algorithm is displayed in the form of a table with a row for each node participating in the distributed algorithm, and a column for each metric.

2. Cumulative data portraying, in the form of a graph, data about performance during a user-defined interval, which helps the developer to identify deviations.

3. Averaged retrospective data depicts performance for a session, day or specified time period. For this we used an established visualisation technique called *parallel coordinates* [13] This allows us to view relations in multivariate data. The advantage of this technique is that it is extensible enough to allow comparisons of multiple algorithms and that algorithms can be compared on many different axes at the same time.

The following section discusses implementation details and demonstrates the performance display.

5 The Performance Display Tool

The performance comparison support tool was developed to work with the algorithms already implemented in Algon. During the development some key implementation issues emerged:

1. *How to measure the performance according to the various metrics?* If we wanted to monitor method invocations or other externally observable behaviour it could be done non-invasively [16, 30]. However, since some of the metrics require the participation of the algorithm in recording data it must be achieved invasively by inserting code into the algorithm so that it records things like response times and messages sent and received. There are two ways to keep the tool informed of this data:

(a) Allow the tool to query the processes.

(b) Require the processes to inform the tool of their measurements.

We decided on the second option since it will be difficult for the tool to query processes that need to be blocked or idle for much of the time. It is better for the process to inform the tool after each measurement has been made. The process can detect the presence of the tool and in its absence simply does not measure or report.

The algorithm processes thus measure their own performance and invoke a method on the tool to inform it of the measurements. The tool assists in storing this information in its own data structures. Various operations are performed on the data, such as calculating averages, for example. The performance tool stores the data for a specific session so that it can be used for comparison purposes later.

2. *How to allow the user to specify the metrics to be recorded?* We wanted this to be dynamic so it is done by means of a configuration file which can be set using a graphical user interface prior to running the system.

3. *What additional non-essential functionality should be incorporated?* We decided to allow the users to save the graphs and to print them, and to import the pre-recorded performance results of various system runs in order to support comparison. Navigation is often allowed in visualisations [29] but it is not suitable for our tool and was therefore not implemented.

Figure 4 demonstrates the dynamic system snapshot. The response time of all processes participating in a system using the Maekawa disrtibuted mutual exclusion algorithm is shown in Figure 5. The comparison of the response time of a particular node using the Maekawa and Ricart-Agrawala algorithms over a time period of seconds is shown in Figure 6. Figure 7 displays information about average metric values for the Maekawa and Ricart-Agrawala algorithms. From this graph it is clear that, for the particular distributed system being evaluated, the Maekawa algorithm performs far better than the Ricart-Agrawala algorithm in terms of response time. If throughput is the most important factor in the system, however, the Ricart-Agrawala algorithm might be a better choice.

6 Related Work

System performance visualisation has not been neglected by researchers in this area. The research ranges from techniques for monitoring behaviour to techniques for visualising the collected data. Monitoring can be done either invasively or non-invasively. For example, Konkin *et al.* [16] propose a mechanism for inserting performance measurement components into a system. Welch and Stroud's Kava tool non-invasively reflects on the behaviour of systems [30]. Researchers working in visualisation often specialise their visualisations for particular application areas. For example, Dahlberg and Subramanian [9] report on a visualisation of real-time survivability in mobile networks. Mellor-Crummey and Whalley [20] have developed a tool for performance tuning which measures performance and allows programmers to compare performance on different architectures, for example. Their tool is intended for source-code tuning and is not suitable for Algon's purposes. Some tools have concentrated on visualising the behaviour of parallel systems [10, 11, 19].

Three factors make these and other similar tools unsuitable for use by Algon:

1. Most of these performance visualisation tools perform a post-mortem visualisation — the measurements are pre-recorded and then used to provide a visualisation based on analysis of the entire set of measurements. While this is useful it is sometimes not flexible enough to satisfy the developers' needs in understanding the performance of the system [29].

2. They visualise the behaviour of the entire system. Algon needs to isolate and depict the behaviour of the algorithms in isolation.

3. All visualisation tools intended for use with real-life systems are, to date, intended for use by experts. The main requirement of Algon's algorithmic comparison visualisation is that it should present information in such a way that it can be easily understood, by non-experts, and used, both dynamically and retrospectively, to support a choice between two or more algorithms [20].

7 Conclusion

We have reported on the development of a performance measurement and visualisation tool which supports the comparison of algorithms based on performance and behaviour. This is a significant advance in the use of distributed algorithms by non-experts. Future work will consider how different algorithms in a system may need to interact with one another. An example of this is the way the deadlock-detection and deadlock-resolution algorithms need to work cooperatively in order to deal with deadlock situations. This handling of interacting concerns, and the visualisation thereof, needs to be investigated.

This work was partially funded by grant NRF194 from the National Research Foundation of South Africa. We would like to acknowledge Nigel Bishop for the usage of

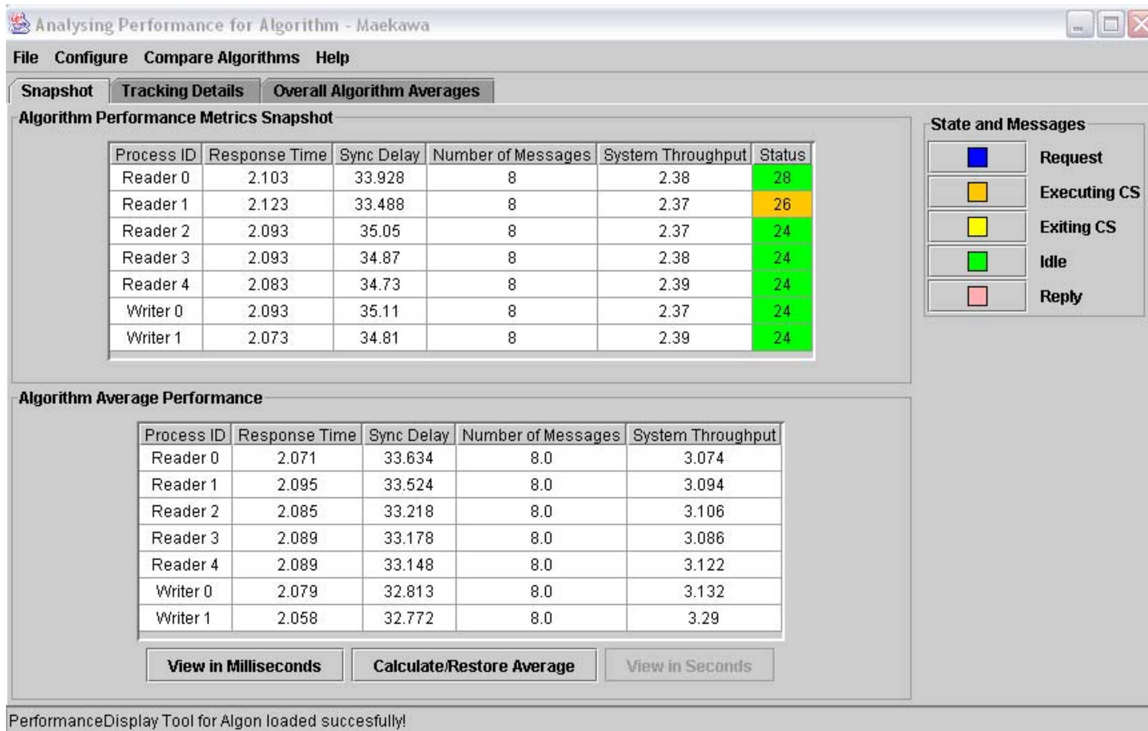

Figure 4. *A Snapshot View*

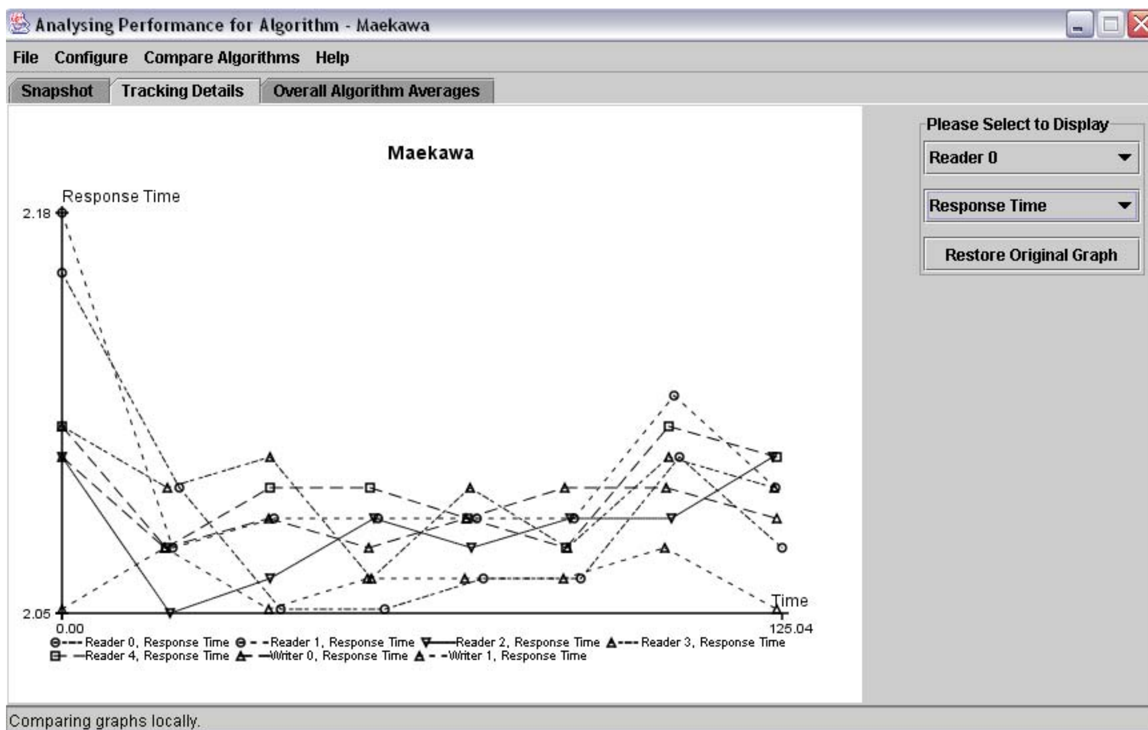

Figure 5. *Comparing the Performance of Processes participating in Maekawa*

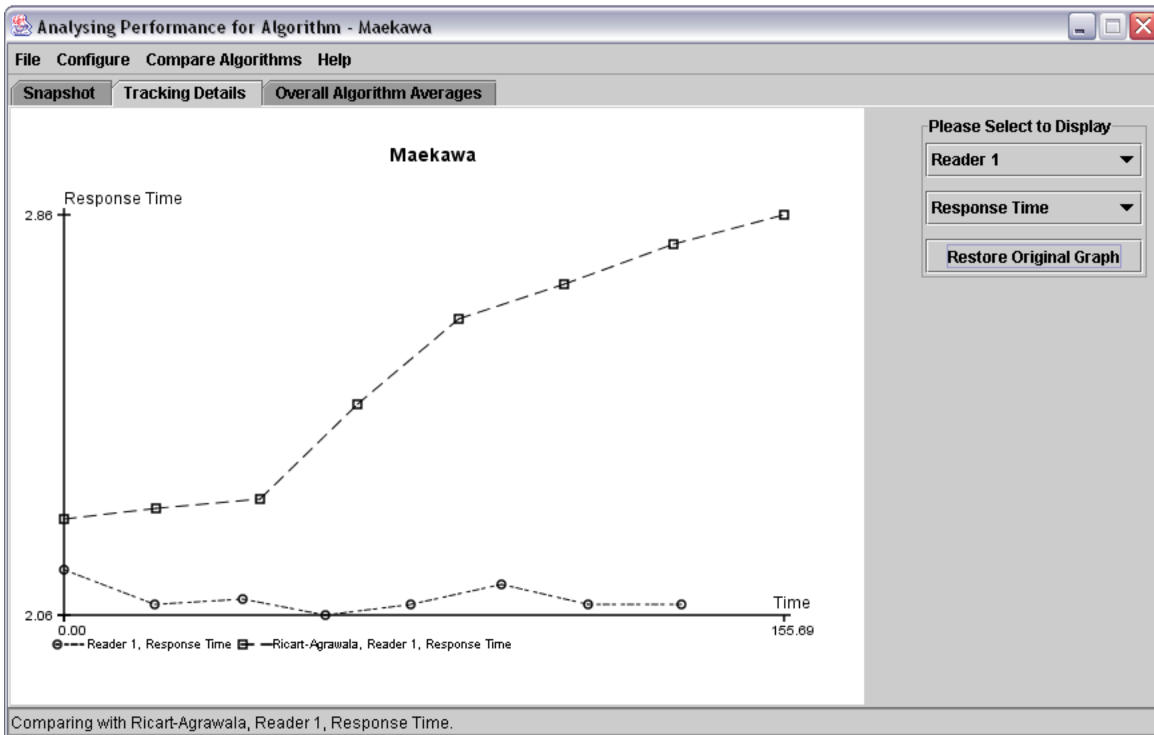

Figure 6. *Comparing the Performance of Nodes using different Algorithms*

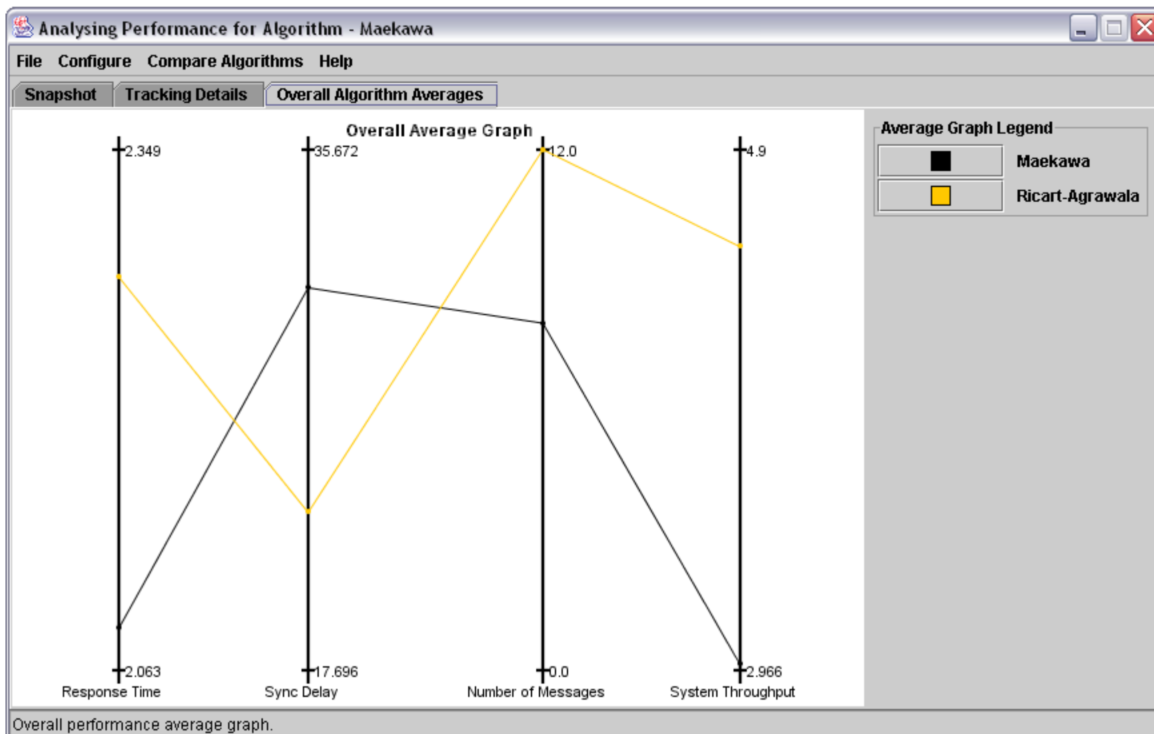

Figure 7. *Viewing Average Performance across Various Metrics*

the Graph class source code from his book: "Java Gently for Engineers and Scientists" Addison Wesley 2000.

References

[1] J. Aspnes and O. Waarts. Modular competitiveness for distributed algorithms. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 237–246. ACM Press, 1996.

[2] A. Bain and P. Key. Modelling the performance of distributed admission control for adaptive applications. *ACM SIGMETRICS Performance Evaluation Review*, 29(3):21–22, 2001.

[3] M. Ben-Ari. Interactive Execution of Distributed Algorithms. *ACM Journal of Educational Resources in Computing*, 1(2es), Summer 2001.

[4] J. M. Bishop, K. V. Renaud, and B. Worrall. Composition of Distributed Software with Algon — Concepts and Possibilities. In *Workshop on Software Composition. SC 2002.*, Grenoble, France, April 6-14 2002. ETAPS.

[5] J. M. Bishop, B. Worrall, K. V. Renaud, and J. Lo. Java and distribution of applications requiring mutual exclusion and deadlock detection. Submitted for Review, June 2002.

[6] S. Burdette, T. Camp, and B. Bynum. Distributed BACI: A Toolkit for Distributed Applications. *Concurrency and Computation: Practice and Experience*, 12(1):35–52, January 2000.

[7] K. M. Chandy, J. Misra, and L. M. Haas. Distributed Deadlock Detection. *ACM Transactions on Computer Systems*, 1(2):144–156, May 1983.

[8] S. M. Coetzee and J. M. Bishop. A New Way to query GIS on the web. *IEEE Computer*, 15(3):31–45, May–June 1998.

[9] T. A. Dahlberg and K. R. Subramanian. Visualization of real-time survivability metrics for mobile networks. In *Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 113–118. ACM Press, 2000.

[10] M. T. Heath and J. A. Etheridge. Visualising the Performance of Parallel Programs. *IEEE Software*, 8(5):29–39, September 1991.

[11] J. K. Hollingsworth and B. P. Miller. Dynamic control of performance monitoring on large scale parallel systems. In *Proceedings of the 7th international conference on Supercomputing*, pages 185–194. ACM Press, 1993.

[12] H. Hsieh and R. Sivakumar. Performance comparison of cellular and multi-hop wireless networks: A quantitative study. *ACM Sigmetrics Performance Evaluation Review*, 29(1):113–122, June 2001.

[13] A. Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(2):69–92, Oct. 1985.

[14] N. Kaveh and W. Emmerich. Deadlock Detection in Distributed Object Systems. In V. Gruhn, editor, *Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9)*, Vienna, Austria, Sept. 2001.

[15] B. Kolehofe, M. Papatriantifilou, and P. Tsigas. Distributed Algorithm Visualisation for Educational Purposes. In *4th SIGCSE/SIGCUE Conference. ITiCSE'99*, Cracow, Poland, June 1999.

[16] D. P. Konkin, G. M. Oster, and R. B. Bunt. Exploiting software interfaces for performance measurement. In *Proceedings of the first international workshop on Software and performance*, pages 208–218. ACM Press, 1998.

[17] E. Koutsoupias and C. H. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):394–400, 2000.

[18] J. Kramer. Distributed Software Engineering. In *16th ICSE Conference*, Sorrento, Italy, May 1994. Invited State of the Art Report.

[19] W. Meira, T. J. LeBlanc, and A. Poulos. Waiting time analysis and performance visualisation in carnival. In *Symposium on Parallel and Distributed Tools. SIGMETRICS*, pages 1–10, Philadelphia, Pennsylvania, 1996.

[20] J. Mellor-Crummey, R. Fowler, and D. Whalley. Tools for application-oriented performance tuning. In *Proceedings of the 15th international conference on Supercomputing*, pages 154–165. ACM Press, 2001.

[21] G. Murphy, A. Lai, R. J. Walker, and M. P. Robillard. Separating Features in Source Code: An Exploratory Study. In *23rd International Conference on Software Engineering*, Toronto, Canada, May 12–19 2001.

[22] B. A. Nixon. Managing performance requirements for information systems. In *Proceedings of the first international workshop on Software and performance*, pages 131–144. ACM Press, 1998.

[23] R. Perlman and G. Varghese. Pitfalls in the design of distributed routing algorithms. In *Symposium proceedings on Communications architectures and protocols*, pages 43–54. ACM Press, 1988.

[24] G. Ricart and A. K. Agrawala. An Optimal Algorithm for Mutual Exclusion Algorithms. *Communications of the ACM*, 24(1):9–17, Jan. 1981.

[25] C. Shousha, D. Petriu, A. Jalnapurkar, and K. Ngo. Applying performance modelling to a telecommunication system. In *Proceedings of the First International Workshop on Software and Performance*, pages 1–6, 1998.

[26] M. Singhal and N. G. Shivaratri. *Advanced Concepts in Operating Systems*. McGraw Hill, 1994.

[27] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2nd edition, 2000.

[28] L. Tweedie. Characterizing interactive externalizations. In *Conference proceedings on Human factors in computing systems*, pages 375–382. ACM Press, 1997.

[29] R. J. Walker, G. C. Murphy, B. N. Freeman-Benson, D. Wright, D. Swanson, and J. Isaak. Visualizing dynamic software system information through high-level models. In *Conference on Object-Oriented*, pages 271–283, 1998.

[30] I. Welch and R. J. Stroud. Kava — Using Byte code Rewriting to add Behavioural Reflection to Java. In *COOTS '01 — Proceedings of USENIX Conference on Object-Oriented Technology*, San Antonio, Texas, USA, January 29 – February 2 2001.