DESIGN OF ALGORITHMS FOR SHARED RESOURCE ALLOCATION AND CERTAIN SELECTED PROBLEMS IN DISTRIBUTED SYSTEMS

Synopsis of
the thesis to be submitted
in partial fulfilment for
the award of the Degree of

DOCTOR OF PHILOSOPHY

by

R.Rajendra Prasath

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF MADRAS
CHENNAI - 600 005

AUGUST 2003

DESIGN OF ALGORITHMS FOR SHARED RESOURCE ALLOCATION AND CERTAIN SELECTED PROBLEMS IN DISTRIBUTED SYSTEMS

Synopsis

The message passing style of programming is widely used in almost all parallel computers. The primitives to *send* and *receive* messages hide low level architectural details and are ideal for programming many large applications. While message passing systems have been in use for over a decade, relatively few results concerning the complexity of message passing protocols are available. This discrepancy is due to the lack of theoretical models that appropriately capture issues related to communication.

Underneath the message passing strategy, a message passes through several intermediate nodes before reaching the destination. During each phase, the message buffer goes through the network interface connecting the processor to the network. After occupying the buffer at the network interface of the destination, the message is either processed or removed. Whenever a message can not get the critical resource it needs, it must wait. When messages wait for a long time then communication delays can cause processor idling, thereby reducing overall performance greatly. Also the programmer must carefully arrange *send/receive* instruction pairs to avoid deadlock[Liu *et al*, 2001].

An algorithm is said to be *distributed* if it operates on a physically or logically distributed computing architecture. The idea of distributed algorithms centers around messages, synchronizing shared use of scarce resources and casual dependencies of actions for some particular computing architecture. The decisive properties of each distributed algorithm include aspects of safety and liveness, intuitively characterized as *nothing bad will ever happen* and *eventu-*

ally something good will happen, respectively. Distributed systems often consist of subsystems which share scarce resources. Such a resource (e.g., a shared variable) is accessible by at most one component simultaneously[Lynch, 1996].

To co-operate to solve a problem, processors in a distributed computing system must communicate among themselves. For both large computer networks and Very Large Scale Integrated (VLSI) architectures, the inclusion of a shared memory to facilitate inter processor communication is usually infeasible[Loui, 1984]. The processors in distributed systems can communicate only by sending messages via a network. An effective algorithm for distributed systems should minimize the message traffic in order to minimize the computation time. The communication model is assumed to be asynchronous, requires decentralized control, admits no shared memory and permits data transfers only on a communication network.

Usually in distributed systems, communication among autonomous sites or processors takes place exclusively by sending messages. The problem is how to guarantee the integrity of a shared resource. The preferred method to accomplish mutual exclusion for this resource is by restricting its use to only one site at a time. An effort to guarantee exclusive access to a shared resource is one of the first problems encountered in parallel programming. The term *Critical Section* is used to characterize a structural program abstraction for concurrent access to a shared resource[Maekawa, 1985; Makki, 1994]. Regarding the critical section and its exclusive access to it, the following conditions must be satisfied.

i) Any request from a process to enter the critical section will be granted in finite time.

ii) Any process concurrently in the critical section will exit in finite time.

iii) At any given time, only one process can enter the critical section.

In distributed systems, we often encounter two critical aspects namely *deadlock* and *starvation*. A *deadlock* is defined as a state in a distributed system where a site which desires to enter the critical section is never allowed entry into the critical section. *Starvation* occurs when a site which is requesting for shared resource, never receives. Adding the properties of correctness, freedom from deadlock, freedom from starvation, fairness and fault-tolerance to the solutions of problems which occur in distributed systems attract more attention to the mutual exclusion problem. In distributed systems, there are two types of algorithms namely token based algorithms and non-token based algorithms. We have, in this thesis, concentrated particularly on token based algorithms for controlling the allocation of single shared resource and sorting a set of elements among the processors in distributed systems.

In token based control algorithms[Lamport, 1978; Andrews and Schulz,1982; Suzuki and Kasami, 1985; Raynal, 1988; Raymond, 1989; Lodya and Kshemkalyani, 2000 and Wu and Shu, 2002], the *token* is a unique and singular message, also known as the *previlege*[Suzuki and Kasami, 1985] which is circulated among the sites. Only the node which has the token may enter the critical section. Different token based algorithms are distinguished by their method for determining how a site obtains the token and where the site sends the token immediately after finishing the critical section(assuming that there is a pending request). Further a site may not release the token until it has completed the critical section.

In order to measure the efficiency of token based algorithms, the following measures have been used[Feuerstein *et al*, 1998; Tarjan, 1985]. The first measure is the *average number of message exchanges per request* necessary to satisfy a sequence of requests, i.e., the worst case ratio between the total number of (token and request) messages and the number of requests in the sequence. The second measure is the *service traffic*, defined as the worst case number of messages that

are exchanged in the network between the time in which a processor sends a request and the time in which it gets served. To obtain a feasible solution, the service traffic should be finite. Another measure used is the number of message exchanges per critical section.

LeLann [LeLann, 1977] has proposed a token based algorithm in which token is circulated among processors in a logical ring. Any site(or node) which desires to enter the critical section simply waits for the token arrival. After completing the critical section, the token is forwarded along the logical ring. Even though this approach is simple, it is easy to notice the inefficiency of this method as it can unnecessarily utilize network bandwidth in a lightly loaded system by continuously and needlessly passing the token. This weak condition mandates no upper bound for the number of message exchanges per critical section.

Ricart and Agrawala (1981) have proposed a mutual exclusion algorithm that requires $2(N-1)$ message exchanges for each critical section invocation in a computer network whose nodes communicate only by messages and do not share memory, while Suzuki and Kasami's distributed mutual exclusion algorithm [Suzuki and Kasami, 1985] requires at most N messages per critical section. Maekawa's algorithm[Maekawa, 1985], for mutual exclusion in a computer network, further reduces the number of message exchanges per critical section entry to $O(\sqrt{N})$. Then Raymond proposed a tree-based distributed mutual exclusion algorithm that depends on the precise topology of the network spanning tree used and the average number of messages required per critical section is $O(\log N)$[Raymond, 1989].

Makki *et al*[Makki, 2000] have presented a token-based distributed mutual exclusion algorithm for a fully connected (complete) graph with N sites. All connections are assumed to be full duplex connections. Further it is assumed that the message transmission time among sites is unpredictable and that no global clock exists. The proposed Makki's algorithm offers a higher degree

of connectivity[Makki, 1994] and requires $(N + 1)/2$ message exchanges per request in the worst case. The characteristic is highly desirable as it does not unnecessarily burden the network with frivolous message traffic. Also token always circulates in a counter clockwise direction and request messages always circulate in clockwise direction.

Feuerstein *et al*(1988) have presented a solution to the token distribution problem by means of a permission(control) token. A *free* token is passed from one processor to the next processor in the ring according to a predefined set of rules understood and adhered by all processors. To access the shared resource, a processor must first obtain a free token and convert it into a *busy* token; once the use of the resource is terminated, the busy token is destroyed and a new free token is again released for further allocation. This type of circular token based control mechanism was the primary motivation for the study of the election problem[LeLann,1978], mutual exclusion problem[Raynal, 1988] and hub polling systems[Lynch, 1996]. Each processor can generate a request for the resource at any time and neither the location nor the time of requests is known in *priori*.

In order to minimize the number of messages exchanged that may be unbounded even for a finite set of requests, the request message based token control strategy for a ring network was presented by Feuerstein *et al*(1998). This request message based strategy allows the token to circulate only if it is informed of a processor asking the resource and avoids an extensively large amount of communication that might be spent to manage a seldomly used resource. They have proposed two protocols in which every request will eventually reach the token and *push* it till the next processor with a pending request.

The fairness of the proposed algorithm can be caused by the order in which requesting sites are serviced with the token relative to the order in which the sites actually requested the token. However due to the lack of a global clock,

there is no way for a distributed algorithm to know absolutely the order in which requests are generated. A system which exhibits fairness should receive requests in the order that they are generated as closely as possible[Makki, 2000]. A system which offers fault-tolerance at the transport layer as well as at the algorithm level is clearly preferable to an algorithm which relies on the transport layer to provide fault tolerance.

In the design and analysis of algorithms, sorting problem is one of the most fundamental problem. It has been extensively investigated in distributed contexts. The *distributed sorting* problem is defined as follows: At the initial state, each processor P_i has an element u_i for sorting. Then the position of each element is assigned to satisfy the condition, $\forall i$, $1 \leq i \leq n$, $u_i \leq u_{i+1}$ at the final state. There exists several models for distributed sorting in various topologies. Loui[Loui, 1984] has presented a simple sorting problem on rings. Gerstel and Zaks [Gerstel and Zaks, 1997] have proved a lower bound of the bit complexity for distributed sorting on a rooted tree. Hofstee *et al* [Hofstee *et al*, 1990] have designed a time-optimal algorithm for a sorting problem on a line network with restricted local memory. However each processor has to store at least two elements and this algorithm fails when each processor has exactly one element. Therefore Sasaki has designed a time-optimal distributed sorting algorithm with a strict lower bound of $(n-1)$ rounds on a line network by creating copies of the elements at the intermediate processors [Sasaki, 2002]. But Sasaki's algorithm does not ensure that each processor always has two elements of the same value at the final round. We propose an algorithm that reduces the number of elements needed for sorting without the creation of copies of elements at the intermediate processors. The proposed algorithm reduces the time complexity to $(n-1)$ rounds.

Due to the importance of shared resource allocation problem and its close association with mutual exclusion property in real time applications, the token

based control strategy for solving this single shared resource allocation problem in various distributed systems attract more attention. Also in distributed environments, sorting adds up enough processing capabilities applicable to various parallel machines with restricted local memory.

Motivated by the above considerations, in this dissertation, the problem of controlling the allocation of single shared resource, among a set of processors that are arranged in various interconnection networks, using request based token passing strategy and the problem of distributed sorting on a line and general networks are investigated. In a distributed computing environment, we assume that there is no global clock, lack of source and destination identities of request messages as well as token and an unknown but finite message delay.

This dissertation consists of seven chapters.

In chapter I, we outline briefly the recent developments of message routing strategies for shared resource allocation, sorting and other proposed problems used in distributed systems and the present work. Also we briefly present the salient features[Tarjan, 1985; Raynal, 1988] that estimate the performance of distributed algorithms for message passing systems and its applications to various real time problems.

In chapter II, we present token based control strategy for the allocation of a single shared resource among n processors that are arranged in ring network and linear array. The first algorithm D' deals with the allocation of single shared resource in a bidirectional ring network in which token moves in anticlockwise direction and requests move in clockwise direction. In this algorithm D', the additional check tour is not required as in the algorithm D of Feuerstein *et al*(1998). It satisfies all requirements of processors and requires atmost $2(n-1)$messages per request and its service traffic is bounded by $3(n-1)$. Then we have presented an algorithm $L1$ for shared resource allocation in a linear array - a ring with a faulty node. In this algorithm $L1$, token starts serving the

processors along one direction serving all pending requests until there exists a pending request; otherwise, the token changes its direction if there is a pending request in the other direction. This algorithm requires $2(n-1)$ messages per request and service traffic is $3(n-1)$.

In chapter III, we consider request-message-based token passing strategy to control single shared resource allocation in the ring extension topologies. First we consider a touching rings network in which two rings that touch at only one common processor through which routing of the request messages as well as token from one ring to another takes place either in unidirectional or in bidirectional way. Here the role of the central processor is significant as it can appropriately switch the token as well as request messages according to the traversal of the token as well as request messages. The algorithm for unidirectional touching rings requires atmost $(n-1)$ messages per request and $\frac{(5n-2)}{2}$ service traffic. In bidirectional touching rings, the central processor balances the traffic by keeping track of request messages received from both rings concurrently. The algorithm for bidirectional touching rings requires $(n-1)$ messages per request and $(2n-1)$ service traffic in the worst case. Also the proposed algorithm can be generalized to multi-touching rings network. Then we deal with fault-free bidirectional intersecting rings in which processors are arranged in two rings with different total number of processors that intersect at any two arbitrary common processors through which the routing of messages takes place. During the indexing process, one common processor is counted in the first ring and the other in the second ring. This algorithm requires n messages per request and service traffic is bounded by $2n$.

In chapter IV, we deal with the single shared resource allocation in two interconnected rings, single side wrap around mesh and regular mesh. In an interconnected rings network, there are $\frac{n}{2}$ processors in each ring, n is the total number of processors(assumed to be even). Here each processor $P_{(i+\frac{n}{2})}$ in second

ring communicates with the processor P_i in the first ring. Whenever a request is generated for the shared resource by the processor in the second ring, then it will be forwarded to the directly connected processor in the first ring and the processor in the first ring forwards the received request to its adjacent processor. Token maintains a counter to recognize the receipt of request messages. The algorithm for interconnected rings requires $(n-1)$ messages per request and service traffic is bounded by $(3n-2)$.

In a single side wrap around mesh, n processors are arranged in k rings with m processors that are connected by a bidirectional channel. We assume unidirectional movements over the ring edges and bidirectional movements over the edges connecting processors in the adjacent rings. In this algorithm, token makes a check tour in the base ring to learn and serve all the pending requests. The proposed algorithm requires $(n-1)$ messages per request and service traffic is bounded by $(2n+m-1)$. Then we consider bidirectional token as well as request movements in a regular 2−dimensional mesh for single shared resource allocation. In this network, token moves from base row processor to column processors. After serving the request in that column, the token is passed to a processor in the adjacent column. From this column processor, token reaches the base row processor only after serving all pending requests. In this protocol, we do not need the token counter. The algorithm for regular meshes needs $(n-1)$ messages per request and service traffic is $(2n+1)$, in the worst case.

In chapter V, we propose two token based single shared resource allocation algorithms for general networks. The first algorithm, for the indexed structure of an undirected, fault-free, connected network, consists of two phases. The given network is converted into depth(or breadth) first search order. Then in the first phase, we consider the allocation of the shared resource among the leaf processors (degree of the leaf processor is 1) and in the second phase, we describe an allocation procedure for controlling the movements of the token

and request messages at a central processor whose degree is greater than 1. During the service, token keeps track of the entries of pending requests and the request indicator, which moves along the token, pushes the token towards the processor with a pending request. This algorithm serves all requests, requires $2(n-1)$ messages per request and its service traffic is bounded by $3(n-1)$ in the worst case. Then we have described an allocation algorithm using depth first search ordered nodes of a network in the form of a linear array. Here the depth first search ordered nodes shall be embedded into the linear array in such a away that the indices of the nodes shall be mapped into the indices of the linear array. This algorithm needs $2(2n-1)$ messages per request and its service traffic is bounded by $3(2n-1)$. The proposed algorithms can also be modified for breadth first search ordering of nodes of the general network.

In chapter VI, we consider the problem of sorting n elements distributed over a number of communication nodes on a line network. We have derived a fast efficient algorithm focused on *median based exchanges* with the worst case lower bound of $(n-1)$ rounds for distributed sorting, where n is the number of processors in a line network. The proposed distributed sorting algorithm improves the performance of each processor without creating copies of $(n-2)$ elements at the intermediate processors and reduces the execution time of Sasaki's time-optimal algorithm [Sasaki, 2002]. However the algorithm assumes the knowledge of global location of each processor, where as Sasaki's algorithm and Hofstee *et al*'s algorithm assume no knowledge of global position of each processor. But, as in Hofstee *et* al's algorithm [Hofstee *et al*, 1990] in which each processor communicates with both neighbors simultaneously, we do communicate with both neighbors simultaneously only at the *median* processors and not at all non-median processors. In the proposed algorithm, all processors do not necessarily perform the disjoint comparison - exchange operations as in parallel sorting in a linear array. The idea is different from the approach of odd-even transposi-

tion sort and the simulation results show that the proposed algorithm is faster than the Sasaki's time-optimal sorting algorithm. Even though the proposed algorithm takes $(n-1)$ rounds in the worst case, it is still fast and robust. Also we have proposed a distributed sorting algorithm for general networks which is different from the strongly sequentialized approach of Gerstel and Zaks for a tree network [Gerstel and Zaks, 1997].

The final chapter ends up with concluding remarks and future directions.

References

Andrews.D and Schulz.G., A token-ring architecture for local area network: an update, in: Proc. IEEE COMPCON, 1982.

Feuerstein.E, Leonardi.S, Marchetti-Spaccamela.A and Santoro.N., Efficient token-based control in rings, *Inform. Process. Lett.* 66(1998) 175-180.

Gerstel.O and Zaks.S., The bit complexity of distributed sorting, *Algorithmica*, 18 (1997) 405-416.

Hofstee.H.P, Martin.A.J. and Van De Snepscheut.J.L.A., Distributed sorting, *Sci. Comput. Programming*, 15 (1990) 119-133.

Lamport.L., Time clocks and ordering of events in distributed systems, *Comm. of the ACM*, 21 (1978) 558-565.

LeLann.G., Motivation, Objective and Characteristics of distributed systems, Springer-Verlag, 1 (1978) 1-9.

LeLann.G., Distributed Systems − Towards a formal approach, in: B.Gilchrist(Ed.), *Inform. Processing*, 77 (1977) 155-160.

Liu.P, Aiello.W. and Bhatt.S., Tree search on an atomic model for message passing, *SIAM J. Comput.*, 31 (2001) 67-85.

Lodya.S and Kshemkalyani.A., A fair distributed mutual exclusion algorithm, *IEEE Trans. on Parallel and Distrib. Syst.*, 11 (2000) 537-549.

Loui.M.C., The complexity of sorting on distributed systems, *Inform. and Control*, 60 (1984) 70-85.

Lynch.N.A., Distributed Algorithms, Morgan Kaufmann Pub., San Francisco, 1996.

Maekawa.M., An $\sqrt{N}$ algorithm for mututal exclusion in decentralized systems, *ACM Trans. on Comput. Syst.*, 3 (1985) 145-159.

Makki.K., An efficient token based distributed mutual exclusion algorithm, *J. Computer and Software Engineering*, 2 (1994) 401-416.

Makki.K, Delt.J., Pissinou.N., Melody Moh.W. and Xiaohua Jia, Using logical rings to solve the distributed mutual exclusion problem with fault tolerance issues, *The Journal of Supercomputing*, 16 (2000) 117-132.

Raymond.K., A tree based algorithm for distributed mutual exclusion, *ACM Trans. on Computer Systems*, 7 (1989) 61-77.

Raynal.M., Distributed Algorithms and Protocols, John Wiley and Sons, NY, 1988.

Ricart.G and Agrawala.A.K., An optimal algorithm for mututal exclusion in computer networks, *Commn. of the ACM*, 24 (1981) 9-17.

Sasaki.A, A time-optimal distributed sorting algorithm on a line network, *Inform. Process. Lett.* 83 (2002) 21-26.

Suzuki.I, and Kasami.T., A distributed mutual exclusion algorithm, *ACM Trans. on Computer Systems*, 3 (1985) 344-349.

Tarjan.R.E., Amortized Computational Complexity, *SIAM J. Alg. Discrete Math.*, 6 (1985) 306-318.

Wu.M.Y. and W.Shu, An efficient distributed token-based mutual exclusion algorithm with central coordinator, *J. Parallel and Distrib. Comput.*, 62 (2002) 1602-1613.