A Hierarchical Protocol for Decentralizing Information Dissemination in Distributed Systems

JOHN G. VAUGHAN

Department of Computer Science, University College Cork, Cork, Ireland

Systems of distributed processors connected by a physical network may have a virtual structure imposed on them to facilitate parallel cooperative system actions. This paper describes the Multiple Virtual Rings organization which is an instance of this approach. System processors are partitioned into groups, structured as virtual rings, which operate in parallel to accumulate local information and make decisions. A hierarchy of rings is formed, at the pinnacle of which is one ring which joins the system together and enables overall cooperation. The configuration of the virtual ring hierarchy is explained and issues which are independent of the ring topology are discussed. A simulation study of a large distributed system is presented which investigates the operation of load balancing algorithms in a hierarchical processor organization.

Received November 8, 1993, revised January 19, 1995

1. INTRODUCTION

In a distributed system of workstations connected by a network, the entire system may be applied in a cooperative manner to the solution of common problems. A prime example of such an application is the collective processing of computing load by network processors. When this processing is scheduled in order to optimize a system-wide performance criterion such as mean job response time, the resulting activity involves exchanging jobs between processors and is known as load sharing or load balancing. In order to make sensible scheduling decisions, some form of information distribution must be organized in the network. Information distribution can be triggered on demand, when a system-wide scheduling decision has to be made at a processor, or it can occur periodically, in which case the scheduling decision is based on the latest locally-available information. In addition, the area from which information is gathered can be limited to within a local neighbourhood of the decision maker or it can include the entire set of system processors. Whatever policy is adopted in this regard, it must facilitate rapid action by harnessing the parallelism inherent in the system structure. The approach taken in this paper is to organize the processors in a virtual structure so that individual processors are grouped into neighbourhoods and information gathering within a neighbourhood occurs in parallel with that in other neighbourhoods. The neighbourhoods themselves are in turn grouped into higher-level neighbourhoods and the pattern continues until at the highest level there is just one neighbourhood which completes the virtual interconnection of all the network processors. The hierarchical nature of the structure implies that the age, nature and precision of the information distributed may vary between levels.

We present here an instance of this philosophy called the Multiple Virtual Rings (MVR) protocol. MVR structures each neighbourhood in the hierarchy as a virtual ring in which an information-gathering token circulates. The structure can be set up either statically or dynamically. The nature of the configuration algorithm allows each node to calculate its own position in the structure, allowing for both local and global reconfiguration on processor failure and repair. The remainder of this section describes the context in which our work on information dissemination is performed. Related work on hierarchical processor organizations is surveyed in Section 2. The MVR configuration algorithm is described in Section 3. Strategies are outlined for coping with failure in Section 4, which also discusses issues encountered in the application of MVR to load-balancing which must be addressed in all similar hierarchical organizations. Section 5 explores the importance of neighbourhood size and gives a method for theoretically determining the virtual ring size which is most appropriate to certain synchronization and transmission delay assumptions. Section 6 evaluates the hierarchical organization of distributed systems by testing load balancing algorithm performance in a system of 500 processors.

The background to our work lies in the investigation of information dissemination for distributed load balancing. One of the main components of a load balancing algorithm is its information policy. This specifies the amount of load information to be used in arriving at a scheduling decision and the way in which it is to be distributed. Information provided by the information policy is used by a placement policy to identify suitable hosts for the execution of jobs which are eligible for remote scheduling. Issues which have to be

addressed in the design of the information policy of a distributed scheduling algorithm are as follows:

1. *Centralization.* Should information be exchanged between every pair of processors in the network or should it be sent to and distributed from a central coordinator?
2. *Periodicity.* Should information be requested only when needed, sent only on change of state or distributed periodically? In the case of periodic distribution, what period should be used?
3. *Quantity.* How much information is necessary to describe a processor's load state and from what subset of the system's processors should it be gathered at a particular network node?
4. *Quality.* Information describing dynamic system state grows irrelevant with age. The degree to which it is useful depends on the frequency with which it is gathered as well as the underlying message transmission delays inherent in the network.

The degree to which a distributed algorithm should be decentralized is one of the basic information-related questions in load balancing. Zhou (1988) investigates several algorithms and concludes that a centralized approach to both information gathering and decision making is best, due mainly to a reduction in message traffic over the distributed approach. He also concludes that there is little difference in performance between periodic and non-periodic information policies. Stankovic (1989) observes that most decentralized algorithms will have 'points of autonomy' and 'points of coordination' and comments on the tradeoff between expending effort on information exchange and advancing the speed of decision making. He identifies the solution of this tradeoff as being an algorithm design decision. The MVR protocol presented in this paper attempts to reconcile the centralized and distributed philosophies by having centralized actions within each base-level neighbourhood, distributing the neighbourhoods and centralizing the information summary at the highest level. MVR is designed for centralized periodic information gathering within each neighbourhood.

The issues of periodicity, information quality and information quantity need to be determined for each individual system by performing sensitivity tests. The indications are that detailed information is not required, but that a small amount of information must be distributed in a timely manner in order to be useful. The usefulness of gathering an increased quantity of information is investigated by Casavant and Kuhl (1987). They conclude that although the overhead resulting from information gathering to detect the global system state is too great for such an effort to be profitable, it may be beneficial to gather and use information about a small subset of the system. Eager *et al.* (1986) advocate the use of adaptive load-balancing policies which use very little information regarding the state of the system. They show that such an approach considerably improves performance over the case where load balancing is not used, and is nearly as good as approaches using more information.

The information quality issue is examined in Dikshit *et al.* (1989) and Shin and Chang (1989) with respect to frequency of collection, and in Mirchandaney *et al.* (1989) with regard to the effect of transmission delays. Dikshit *et al.* (1989), reporting on a testbed system for the evaluation of load-balancing schemes using process migration, conclude that poorer scheduling decisions result from decreasing the frequency of information distribution, and that load balancing is harmful if the frequency falls below a certain level. Shin and Chang (1989) comment on the difficulty of exchanging information sufficiently often while maintaining low network traffic overhead. Their solution is to have each node broadcast an information update within a small radius of itself whenever it experiences a state change. Mirchandaney *et al.* (1989) discuss the performance degradation caused by delays in information distribution and job transfer. They find that when high delays exist (greater than or equal to 10 times the mean job service time), employing non-local information is non-productive. In the presence of such delays, load-balancing itself is useful only at high system loads.

2. RELATED WORK

Hierarchical processor organisations have been described for both multiprocessing and distributed processing configurations. Multiprocessor hierarchies are presented in Ahmad and Ghafoor (1991), Horton (1993), Maples (1985) and van Tilborg and Wittie (1984). A multi-level approach to dynamic load balancing in multiprocessor systems is examined by Horton (1993). The algorithm is intended for use in the parallel solution of partial differential equations, an application in which each calculation phase is followed by a synchronization phase during which the load-balancing algorithm is applied. The algorithm bisects the set of processors, balances the load between the two subsets and is applied recursively to each subset until the newly-created subset is indivisible. This approach concentrates on scheduling rather than information distribution and is applied in a quasi-static manner during synchronization. It also differs from MVR in that its neighbourhood size is fixed at 2, and scheduling action at a higher level must initiate and terminate completely before scheduling action can begin at a descendent lower level containing a subset of the nodes considered at the higher level.

Maples (1985) reports on the MIDAS project, which organizes the processors of a multiprocessor system in a multi-level pyramidal fashion, so that in an L-level system, the number of processors at the base of the pyramid is $2 * 8^{L-1}$. At the base level, processors are grouped in clusters of 16. At the next level, a secondary processor oversees the operation of each cluster. These secondary processors are in turn clustered and placed

under the control of another supervisory processor, and the hierarchical organization continues until at the apex of the pyramid there is just one supervisory processor. Control directives, information and task delegation flows vertically in the pyramidal structure. Processors within a cluster are connected by a crossbar switch, and communication and synchronization information flows horizontally in the structure. The paper comments on the performance of a two-level machine and describes the expansion to machines of three and four levels, the latter machine having 1024 processors at the base level. The scheme reported in Maples (1985) differs from the present work in that MIDAS imposes a physical organization on the components of a multiprocessor system, whereas MVR structures the components of a distributed system in a logical fashion. In addition, MIDAS forms a physical hierarchy of processors by dedicating special processors to supervision. This contrasts with MVR's virtual hierarchy in which processors may be active at several levels and a processor's position in the structure may change due to re-initialization or re-configuration.

Ahmad and Ghafoor (1991) propose a two-level organization of a multiprocessor system for load-balancing purposes. At the lower level, processors are grouped in *spheres*, and the information-collection and scheduling functions within each sphere are centralized at a pre-identified control node called a *scheduler*. At the upper level, each scheduler collects information from remote spheres and makes remote scheduling decisions based on its local view of the global situation. Simulation of this organization for a variety of multiprocessor structures indicates good performance with respect to response time, resource utilization and control overhead.

Van Tilborg and Wittie (1984) describe a technique for scheduling competing task forces in multicomputers which they call *wave scheduling*. This technique is based on the organization of processors in a virtual hierarchy which is not directly related to the physical connections between nodes of the multi-computer. At the lowest level are worker nodes which execute user tasks. Processors at higher levels are designated as managers, and are responsible for task force scheduling and maintaining communications integrity. Managers may themselves be organized under other managers at higher levels in the virtual hierarchy. Direct exchange of control information occurs only between adjacent hierarchy levels. At the topmost level there are several managers which exchange control and status information among themselves. Having many managers at a level facilitates reallocation of function if a manager fails.

Hierarchical organizations of distributed processors have been described in Evans and Butt (1994), Kim *et al.* (1993) and Zhou *et al.* (1993). Kim *et al.* (1993) advocate a hierarchical structure in order to reduce communications costs and increase fault tolerance. They describe a two-level framework and discuss its application to the problems of distributed mutual exclusion and majority consensus. They believe this approach to be particularly appropriate in an internetwork environment.

Evans and Butt (1994) describe network partitioning techniques to be used in large distributed systems in order to reduce the communication costs of load-balancing algorithms. Their approach is to form groups of processors according to geographical criteria and apply load balancing algorithms at two levels in the resulting system, i.e. within groups and between groups. Their study is based on a simulated 16-processor system and is largely concerned with the effect of group membership (i.e. to which group each processor belongs) on load-balancing performance. It examines static group membership (with and without inter-group load balancing), dynamic group membership and joint group membership.

Zhou *et al.* (1993) describe the Utopia load sharing facility for large distributed systems. Utopia uses a two-level structure which organizes processors into clusters in order to promote scalability. Within a cluster, the Central and Global load balancing algorithms (Zhou, 1988) are provided as options. Information distribution between clusters is subject to a directed graph organization of the clusters in the case where clusters with extra computing resources, called *widely-sharable* hosts, can be readily identified. When the widely-sharable hosts are not concentrated in a few clusters, the technique of *virtual clustering* is used to form a sink node in the directed graph. The digraph allows load information to be disseminated with low overhead among those clusters to which it is most useful.

3. MVR CONFIGURATION

A processor which is connected to a network and wishes to cooperate with other network nodes to form a hierarchy under MVR needs to know several items of static information. These are, the total number of nodes in the system (denoted by **m**), the logical i.d. of the processor (denoted by **N**) and the maximum number of processors to be connected in a single virtual ring (denoted by **p**). The latter quantity is called the *nominal ring size* and rings which connect exactly **p** processors are referred to as *complete* rings. Due to the way in which the MVR structure is built, some rings may contain fewer than **p** processors and these are called *incomplete* rings. The logical processor i.d. is particular to MVR and determines a processor's position within the structure. It is calculated on each configuration and reconfiguration and does not persist when a processor leaves the structure, albeit temporarily.

The MVR scheme does not identify particular processors as having a uniquely supervisory function. Rather, all processors function at the base of the structure, while some processors may perform duties at several levels in the hierarchy. The number of levels in the hierarchy is a function of the total number of processors to be connected, **m**, and of the nominal ring size, **p**. For

convenience in the pseudocode which follows, we declare a limit on the maximum number of levels. The number of levels which actually occurs in a particular MVR configuration is then calculated as follows:

number_of_levels := ceil ($\log_{\mathbf{p}}\mathbf{m}$);

Levels are numbered from 1 to number_of_levels such that the base level is level number 1. Each processor maintains a local configuration table comprising a record of its MVR connections at each level:

```
const maxlevels = 10;
type
  node_id = 1..maxint;
  levelrecord = record
    connected, {connected is true if the node is
                connected in a ring at level L}
    firstnode, {firstnode is true if the node is a
                firstnode}

    lastnode: boolean; {lastnode is true if the node is a
                        lastnode}

    predecessor,       {if the node is not a firstnode,
                        then predecessor gives the
                        logical i.d. of node N's
                        predecessor in the ring}
    successor,         {if lastnode is false, then
                        successor gives the logical i.d.
                        of node N's successor in the
                        ring }
    first_id : node_id {if lastnode is true, then first_id
                        gives the logical i.d. of the
                        ring's firstnode}
    end;

  configuration_table = array[1..maxlevels] of
  levelrecord;

var
  configtab : configuration_table;
```

A virtual ring is designed to be traversed by a token which originates at a particular node of that ring, called the *firstnode*, passes through intermediate nodes and ends its traversal at a preidentified node called the *lastnode*. The token is created at the firstnode and is destroyed on arrival at the lastnode. Token creation may be periodic or event-driven, but up to now we have used periodic token creation.

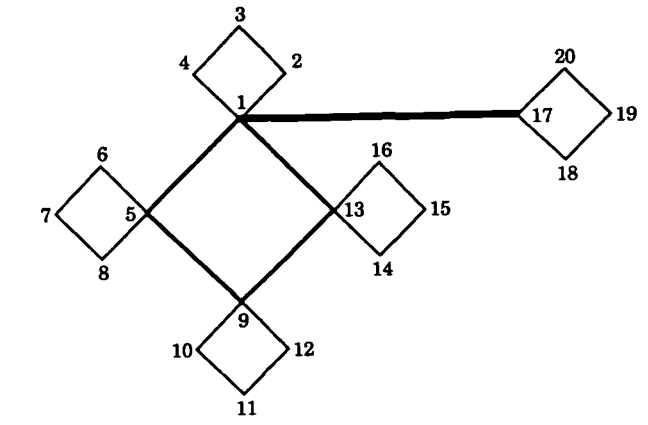
As the token travels through the nodes of the ring, it accumulates information, so that when it arrives at the lastnode, information has been gathered from all nodes of the ring. In order for all nodes to be aware of each others' status, the token should circulate once again in an information dissemination phase. Thus in a **p**-node ring, the token makes $\mathbf{p} - 1$ internode hops to bring complete ring information to the lastnode. This could be followed by a further $\mathbf{p} - 1$ hops to distribute the complete information as far as the node preceding the lastnode. This means that a total of $2(\mathbf{p} - 1)$ internode hops are needed for full dissemination of information within the ring. However, since the lastnode possesses complete ring information following one token traversal, it can make an immediate central decision for the ring, and this is the approach that we advocate.

When the number of system processors exceeds the nominal ring size, more than one ring will exist at level 1. Each of these rings will have its own circulating token, so that information-gathering within the rings is concurrent. To facilitate system-wide information migration, the level 1 rings are connected by a level 2 ring which has its own circulating token. Since a token is passed from node to node, certain processors of the level 1 rings must also serve as processors in the level 2 ring. These processors are called *Interlevel Contact Points* (ICPs), and are responsible for transmitting information from one level to another. The idea of an ICP is not only applicable in MVR, but may also be applied in any hierarchical information-distribution protocol which does not appoint special supervisory processors.

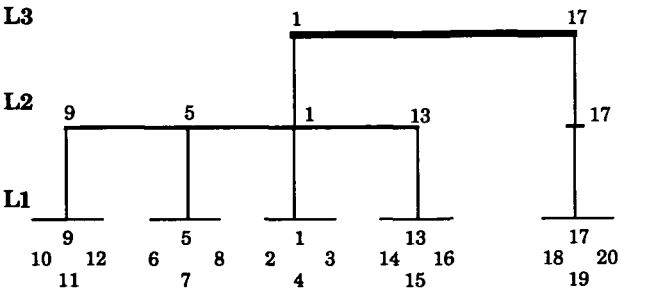
A token circulating in a ring at level 1 terminates at the lastnode of that ring. The information gathered by the token must be sent to the ring's firstnode, since that node is the ICP for communication to higher-level rings. For this reason, a processor which is the lastnode of a particular ring maintains the logical i.d. of the firstnode of that ring in its corresponding levelrecord.

For a given nominal ring size, the number of levels in an MVR structure depends on the total number of processors to be connected. For many MVR structures, all of the virtual rings in the structure will be *complete*, i.e. no processor will exist at any level in a ring which connects fewer than the number of nodes specified by the nominal ring size. However, for certain combinations of system size **m** and nominal ring size **p**, some of the virtual rings may connect fewer than **p** elements. Thus for $\mathbf{p} = 10$ and $\mathbf{m} = 6$, there is only one ring, which we call *incomplete*. The gathering of information within incomplete rings takes place in the same manner as for complete rings. However, the configuration algorithm has to recognize that a ring is incomplete and adjust the levelrecord entries accordingly.

Figure 1 shows a system of 20 processors connected by an MVR structure using a nominal ring size of $\mathbf{p} = 4$ processors. There are five complete rings at level 1, only one complete ring at level 2 and the sole level 3 ring is incomplete. There are three levels, comprising a total of seven virtual rings and therefore seven asynchronously circulating tokens. Five of these rings are at the base level, one at level 2 and one at level 3. Parallelism occurs both in information gathering and decision making due to asynchronous operation of the rings. Consider the level 1 ring connecting logical nodes 1, 2, 3 and 4. Node 1 is the firstnode and node 4 is the lastnode. A token will originate at node 1, pass through nodes 2 and 3 and terminate at node 4. All the nodes are connected at this



(a)



(b)

FIGURE 1. MVR interconnection of 20 processors with a nominal ring size of $p = 4$ processors. (a) Plan view showing complete and incomplete rings. (b) Elevation showing connection hierarchy.

level. To communicate information between the level 1 rings with firstnodes 1, 5, 9 and 13, a level 2 ring is formed. The level 2 token circulates among the processors from which level 1 tokens originate. Thus the level 2 token begins its traversal at node 1, passes through nodes 5 and 9, and terminates at node 13. Information is collected and used within the level 2 ring in the same manner as in the level 1 rings. In the 20-node scheme of Figure 1, the level 1 lastnodes are 4, 8, 12, 16 and 20. When tokens arrive at these nodes, the completed ring information is sent to nodes 1, 5, 9, 13 and 17, respectively. This information is collected at level 2 and in due course complete level 2 information is returned to node 1 (the level 2 firstnode) from node 13 (the level 2 lastnode). Level 3 information is collected and returned to node 1 (the level 3 firstnode) by a token circulating between nodes 1 and 17.

Information moves vertically between adjacent levels of the hierarchy through the ICPs. The hierarchical structure of MVR is emphasized in Figure 1(b) by giving an elevation view which shows the vertical communication paths from level 1 to level 2 at nodes 1, 5, 9, 13 and 17, and from level 2 to level 3 at nodes 1 and 17. The situation at level 2 is interesting, since as well as the complete ring connecting nodes 1, 5, 9 and 13, there is an incomplete ring which contains just one element, node 17. In this latter ring, node 17 is both the firstnode and the lastnode. Thus it is not necessary to have a token circulating within this virtual ring. At level 3, there is one ring which connects just nodes 1 and 17. Node 1 is the firstnode and node 17 is the lastnode. Consequently, node 17, although connected at all three levels, only handles a token at levels 1 and 3. In contrast, node 1 which is also connected at the three levels, handles a token for each level.

To join an MVR structure, a processor must complete an appropriate number of levelrecords in its configuration table. In order to accomplish this, the processor needs to know the total number of processors in the system as well as its own MVR-specific logical i.d. Furthermore, all processors need to agree on the values of these items. The MVR configuration phase is decentralised and proceeds according to the following steps:

1. Configuration begins when all processors exchange messages to form a distributed queue. Once the queue is formed, all processors are aware of the system size, **m**. A processor can find the nominal ring size, **p**, by consulting a previously-constructed local table. As we discuss later, the best nominal ring size to use is predictable and depends on the system size. From this

```
procedure configure (var configtab : configuration_table;
                     N,   { logical i.d. of this node }
                     m,   { number of nodes in system }
                     p,   { nominal ring size }    : node_id);
  var
  number_of_levels : 1..maxlevels;
  i, L : 1..maxint;
  first : boolean;
  begin
  number_of_levels := ceil(log_p m);
  for L := 1 to number_of_levels do
    with configtab[L].levelrecord do
      begin
      connected := ((N - 1) mod p^(L-1) = 0);
      if connected then
        begin
        firstnode := ((N - 1) mod p^L = 0);
        lastnode := ((N - 1 + p^(L-1)) mod p^L = 0);
        if not firstnode then
          predecessor := N - p^(L-1);
        if not lastnode then
          if (N + p^(L-1)) ≤ m then
            successor := N + p^(L-1)
          else
            lastnode := true;
        if lastnode then
          begin
          i := N;
          first := firstnode;
          while not first do
            begin
            i := i - p^(L-1);
            first := (((i - 1) mod p^L) = 0);
            end;
          first_id := i
          end
        end
      end
  end; { procedure configure }
```

FIGURE 2. MVR configuration procedure called by each processor.

information, the processor can calculate the number of levels in the hierarchy.
2. The processor at the head of the queue takes logical i.d. 1 and informs its successor which then takes logical i.d. 2, continuing in this manner until the processor at the tail of the queue takes logical i.d. **m**. In order to maintain a physical foundation for the MVR structure, processors subsequently broadcast their (physical_id, logical_id) pairs. Thereafter each processor maintains a table which enables it to associate a physical network address with a given logical i.d.
3. Each node executes the configuration algorithm shown in Figure 2 in order to complete its configuration table. This algorithm is called only when the conditions $\mathbf{p} > 1$ and $\mathbf{m} \geqslant \mathbf{p}$ are true.

This configuration procedure assumes that a processor's position in the logical structure does not affect its performance due to the topology of the physical network. If this assumption does not hold, steps 1 and 2 of the configuration procedure must be modified accordingly. If a processor attempts to form a distributed queue when one already exists, it is joined to the end of the existing queue and informed of the i.d. of its predecessor, from which it can calculate its own logical i.d. and the system size. Its attempt to form the queue alerts the other processors to its existence and enables them to respond by sending it their (physical_id, logical_id pairs).

4. EXPERIENCE WITH THE APPLICATION OF MVR

Experience in the application of MVR to distributed load balancing has highlighted some interesting situations. Some of these may arise due to the MVR structure itself, but many are caused by the basic idea of a hierarchy of cooperating neighbourhoods and must be addressed by any instance of this abstraction.

The first case concerns how MVR should react to certain types of failure. If a logical link failure between processors P_i and P_j occurs, this is detected at processor P_i as a failure to find a route to P_j (and vice versa). P_i may regard this situation as equivalent to failure of processor P_j and hence link failure is dealt with in the same manner as processor failure. Processor failure not only necessitates a restructuring of MVR links, but can also give rise to token loss.

Although MVR generally assumes an underlying reliable message delivery subsystem, loss of the information-gathering token can still occur when a token sender node believes that the next node in the ring is active and transmits the token, subsequently discovering that the destination has failed. Special recovery actions to cope with token loss are unnecessary since MVR's normal actions make it resilient to such a failure. Within each virtual ring, a new token is periodically created by the firstnode and destroyed when it reaches the lastnode. Thus when a token is lost, a new token will be created in any case at the beginning of the next information-collection period.

The short-term approach to reforming the MVR structure following failure of processor P_j is to re-calculate the logical links of all processors previously directly connected to P_j at some level in the MVR hierarchy. If the failed processor is a firstnode or a lastnode in some ring, then the surviving processors in that ring must agree on a new firstnode or lastnode. This may cause partitioning of the MVR structure (even though the physical network may not be partitioned), since loss of a firstnode at level L implies that there is no route for information exchange with levels $L + 1$ and above. Note also that a firstnode at level L is also a firstnode at levels $L - 1$ down to 1. Figure 3(b) shows the 27-processor structure of Figure 3(a) after failure of node 1. Note that the structure now exists as three separate parts whose highest levels are:

1. An incomplete level 1 ring connecting nodes 2 (the new firstnode) and 3.
2. An incomplete level 2 ring connecting nodes 4 (the new firstnode) and 7.
3. An incomplete level 3 ring connecting nodes 10 (the new firstnode) and 19.

The lower-level rings connected by the level 2 and 3 incomplete rings are themselves complete. This gives rise to a need for a medium-term solution to processor failure, which is reconfiguration. Reconfiguration may be initiated by a node which discovers that one of the original members of a ring in which it operates has been out of operation for an extended period of time. When a processor receives a reconfiguration message, it is obliged to yield priority to the request and take place in a new configuration phase. Thereafter, reconfiguration proceeds in the same manner as the initial configuration procedure described in the previous section. Logical node i.d.s are reassigned and the MVR structure configured for the new system size. This can mean the use of a different nominal ring size **p**, although a policy decision can be taken to maintain the original ring size on reconfiguration since the failed processors may be expected to rejoin the system at some future stage. The dynamic assignment of logical node i.d.s means that saving the structure of the original configuration for use when the failed processors are repaired is of no value. Thus, in Figure 3(c), which shows the partitioned system of Figure 3(b) after reconfiguration, it must be understood that the processors connected as ring nodes may have entirely different physical addresses to the processors at the same positions in Figure 3(a). When a repaired processor wishes to join an existing MVR structure, it broadcasts its request to the other system processors and is duly informed (by the processor with highest existing logical i.d.) of its new logical i.d., the number of system processors and the nominal ring size in use. In this way the repaired processor may calculate its

position in the structure and current structure members are informed of the new arrival.

The second case arises from a restriction on job migration possibilities due to the MVR structure. If a scheduling action at level L seeks to balance load among level $L-1$ rings, the jobs which are available for direct transfer from these rings are those resident at the $L-1$ ICPs. Thus a scheduling decision at level L may be futile if no transferable job exists at the ICP in question. The solution that we have adopted is to bias job transfers within the level $L-1$ rings so that transferable jobs are always migrated to the ICPs even if such a transfer is not immediately profitable from a scheduling point of view.

The third case concerns the attachment of appropriate significance to information collected from different levels of the hierarchy. The problem arises in MVR because of the likely existence of incomplete rings. However, this is a problem that occurs in any hierarchical information-collection structure, since no matter how the groups of processors are formed, there is always a possibility that some groups will be larger than others. To illustrate the situation, suppose that the objective of a load-balancing algorithm is to try to schedule equal numbers of jobs on all processors. Consider the nineteen-processor system of Figure 4 which uses a nominal ring size of $p = 3$. Processor 19, although it is active at all three levels, only communicates with other processors at level 3. Suppose the system is perfectly balanced with, say, two jobs at each processor. The job counts reported by the processors at each active level are shown in Table 1. The count reported at level 3 by processor 19 is the same as its level 1 count. Thus system load appears to be

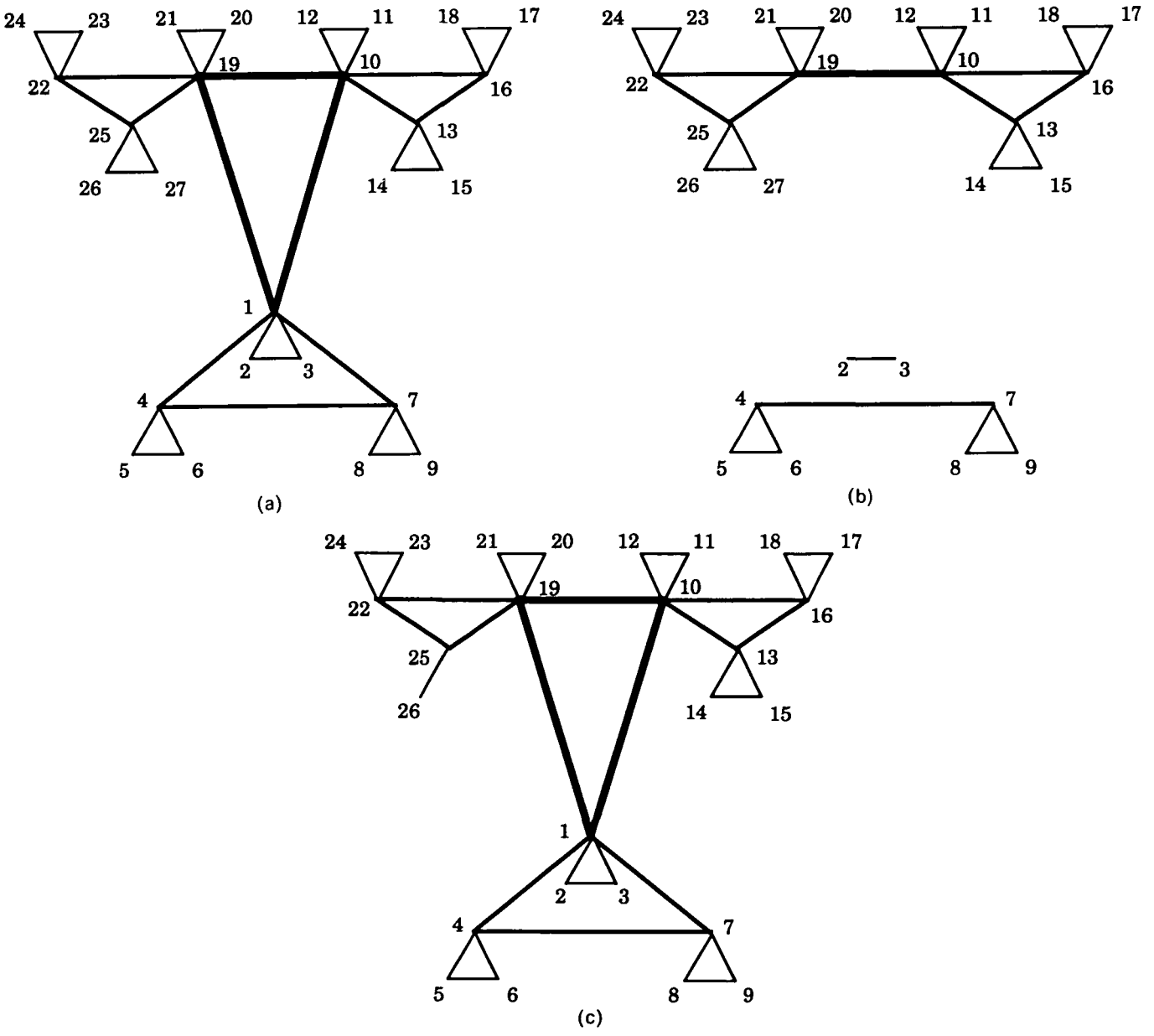


FIGURE 3. (a) MVR interconnection of 27 processors with a nominal ring size of $p = 3$ processors. (b) Interconnection pattern after failure of node 1. (c) MVR reconfiguration of (b) with a nominal ring size of $p = 3$ processors.

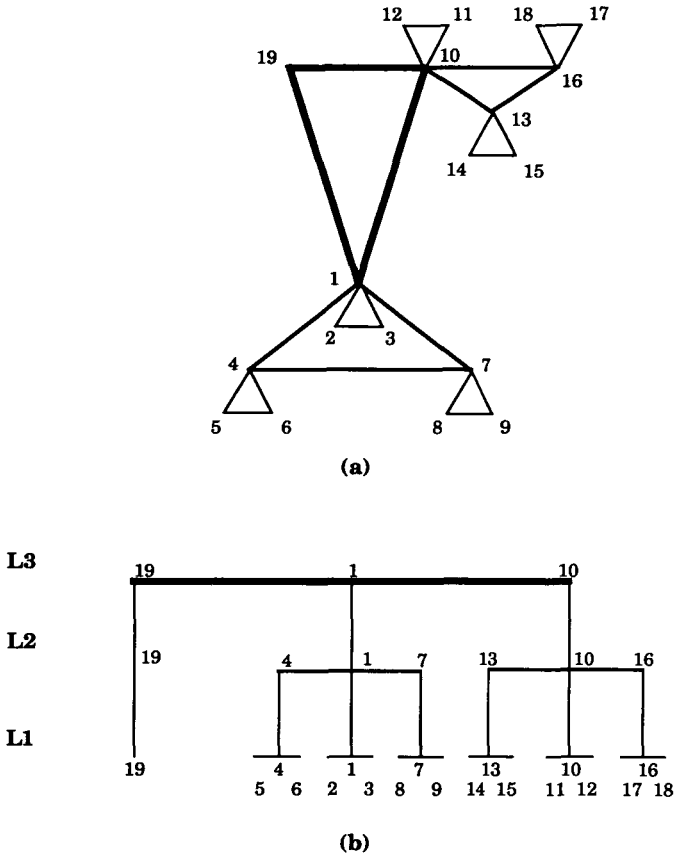


FIGURE 4. MVR interconnection of 19 processors with a nominal ring size of $p = 3$ processors. (a) Plan view showing virtual ring structure. (b) Elevation showing three-level connection hierarchy.

unbalanced at level 3. To compensate for this type of information-reporting error, we use a weighting factor W_L at each level L of every ICP such that

$$W_L = \frac{p}{r} \geqslant 1$$

where p is the nominal ring size and r is the actual number of processors connected to this ring.

Then the load information reported to level $L+1$ from level L is multiplied by W_L. Processor 19 in Figure 4 has W_L values at levels 1, 2 and 3 of 3, 3 and 1, respectively. When these weighting factors are applied to the job counts, the values shown in Table 2 ensue. As a result, system load no longer appears to be unbalanced at level 3.

The fourth case is caused by interference between decision-making activities at different levels in the virtual ring hierarchy. In the load-balancing context, the participation of an ICP processor in a job scheduling decision at level L affects information-gathering and decision making in rings for which the processor is also an ICP below level L and may delay decision implementation in such rings at levels above L. To resolve potential inter-level conflict, a tristate flag, termed the C-flag, is used at each active level of an MVR node. The flag takes the values 'changed', 'unchanged' and 'deciding'. A level L C-flag is initialized with the value 'unchanged', indicating that a job transfer is not being considered at levels above L. If the flag is set to 'changed' then a job transfer involving the node has recently been decided at a higher level, implying that any current information-gathering or decision-making effort should be cancelled and restarted. If the flag is set to 'deciding', a job transfer is under consideration at a level above L but it is not definite that the transfer will take place. The C-flag at level L is checked when an MVR-connected processor P begins to be involved in a scheduling decision at that level. If the flag indicates 'changed', its value is set to 'unchanged', the current decision-making phase is aborted and a new information-gathering phase is initiated. If the flag is set to 'deciding', the decision phase is suspended until the flag again changes state, upon which its new value is checked. If the level L C-flag has the value 'unchanged', then processor P checks its C-flags at all levels below L. If any of these is not set to 'unchanged', the level L decision is postponed until all lower-level C-flags indicate 'unchanged'. Once the level L decision is allowed to proceed, processor P sets its C-flags at level L and all lower levels to 'deciding' and begins the decision phase, cooperating

TABLE 1. Job counts reported at each level by the processors of Figure 9: unweighted job counts

	Processor i.d.																		
Level	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*
1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	6			6			6			6			6			6			2
3	18									18									2

TABLE 2. Job counts reported at each level by the processors of Figure 9: weighted job counts

	Processor i.d.																		
Level	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*
1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	6			6			6			6			6			6			6
3	18									18									18

with other nodes in its level L ring if necessary. When the decision is made, processor P sets its C-flags at level L and all lower levels to 'changed' or 'unchanged', depending on the outcome of the decision.

5. SELECTING A NOMINAL RING SIZE

The MVR structure facilitates parallel decision-making due to simultaneous activities in rings at the same level. The degree of parallelism increases in proportion to the number of rings and in inverse proportion to the nominal ring size. The speed of making a system-wise decision may be expected to increase in proportion to the degree of parallelism. However, higher parallelism implies a greater number of levels in the MVR hierarchy and interlevel synchronization delays retard system-wide decision speed. Consequently, if the number of levels is too great, synchronization overhead will outweigh the advantage of parallel action. It follows that, for a given system size of **m** processors, there is a nominal ring size $\mathbf{p}_{opt}$ which is optimal in the sense of yielding the fastest decision speed. In this section we describe a function we have used to describe algebraically the structure imposed on a system of processors by the MVR protocol. The function is called a path function and it may be used to predict $\mathbf{p}_{opt}$ theoretically. In this way the local table of best nominal ring size versus system size which is used by each processor in step 1 of the MVR configuration procedure can be constructed beforehand.

The path function is denoted by $\mathcal{P} = f(s)$, where s is a variable used to reflect synchronizations between levels in the MVR hierarchy. Coefficients of s^0 describe transmission delays on the virtual paths between nodes. Coefficients of s^i, $i \geqslant 1$, describe synchronizations between levels. The following rules may be used to construct the path function for a particular system size and nominal ring size:

Rule 1: The value of $\mathcal{P}(s)$ is initially 0. For each internode path not having a parallel path that has already been taken into consideration, a coefficient of s^0 is added to $\mathcal{P}(s)$.

Rule 2: Each synchronization between levels L and $L-1$ at an ICP adds a coefficient of s to $\mathcal{P}(s)$. Synchronizations which may occur in parallel MVR substructures are reckoned for only one of those substructures. If $L = 2$, the coefficient of s is the value of the path function for the level 1 ring linked to the ICP. If $L > 2$, the coefficient is formed by deleting the s^0 term from the path function describing the substructure connected to the ICP below level L.

Figure 5 shows sample configurations to be used in the illustration of these rules. Figure 5(a and b) shows the simple application of Rule 1. In Figure 5(a), a level 1 token travels from node 1 to node 2 and complete ring information is returned to node 1 from node 2. Thus the number of paths traversed is two, which becomes the coefficient of s^0, so the path function $\mathcal{P}$ is $\mathcal{P} = 2$. Similarly, in Figure 5(b), three paths are traversed yielding $\mathcal{P} = 3$.

The s^0 coefficient for Figure 5(c) is calculated as follows. Three paths are traversed at level 1, that is, from node 1 to node 2 to node 3 as in Figure 5(b). In addition, a level 2 token travels from node 1 to node 4 and information is returned to node 1 from node 4, giving an extra two path traversals. Thus the coefficient of s^0 is 5. The s^1 coefficient is determined according to Rule 2 by noting that there is one interlevel synchronization, which occurs between levels 2 and 1 at node 1. The synchronization is represented by s and is multiplied by the path function of the level 1 structure with which level 2 synchronizes. The level 1 path function is 3 as in Figure 5(b), so the path function·component due to synchronization is $3s$. The complete path function is formed by adding the separate components together, so $\mathcal{P} = 5 + 3s$.

In Figure 5(d) the s^0 coefficient is still 5, since the three additional paths between nodes 4, 5 and 6 are in parallel to the paths connecting nodes 1, 2 and 3. There are two interlevel synchronizations, each adding a component of value $3s$ to the path function. Thus $\mathcal{P} = 5 + 3s + 3s = 5 + 6s$.

In Figure 5(e), the three level 1 paths joining nodes 1, 2 and 3 are paralleled by those of rings 4–5–6 and 7–8–9, so the level 1 s^0 coefficient contribution is 3. At level 2, there are 3 paths in the ring connecting nodes 1, 4 and 7, so the level 2 s^0 coefficient is 3. There is an interlevel synchronization at nodes 1, 4 and 7, each contributing a term of value $3s$. Thus the path function is $\mathcal{P} = 3 + 3 + 3s + 3s + 3s = 6 + 9s$.

Figure 5(f) adds a level 3 connection between nodes 1 and 10 which contributes an extra 2 units to the s^0 coefficient. This is added to the path function of Figure 5(e) to give $8 + 9s$. At node 1 there is a synchronization of three levels. The synchronization expression corresponding to the levels up to and including level 2 for which node 1 is an ICP is $9s$. This is multiplied by s to indicate the extra level 3 synchronization, giving a path function component of $9s^2$. The complete path function for this structure is therefore $\mathcal{P} = 8 + 9s + 9s^2$.

Figure 5(g) extends the MVR structure by the addition of extra nodes whose interconnections operate in parallel to those already existing at levels 1 and 2. Thus the path function component corresponding to those levels continues to be $8 + 9s$. For level 3, there is an additional synchronization at node 10 which contributes an extra $9s^2$ to the path function. Thus the complete path function is $\mathcal{P} = 8 + 9s + 9s^2 + 9s^2 = 8 + 9s + 18s^2$.

Table 3 lists path functions for system sizes of four to 20 nodes and nominal ring sizes of three to six nodes. To interpret the path function values, we recall that coefficients of s^0 in $\mathcal{P}(s)$ account for transmission delays, coefficients of s^1 correspond to synchronizations between pairs of levels, coefficients of s^2 describe synchronization between three levels, and so on. For the purpose of interpretation, the powers of s in $\mathcal{P}(s)$ may

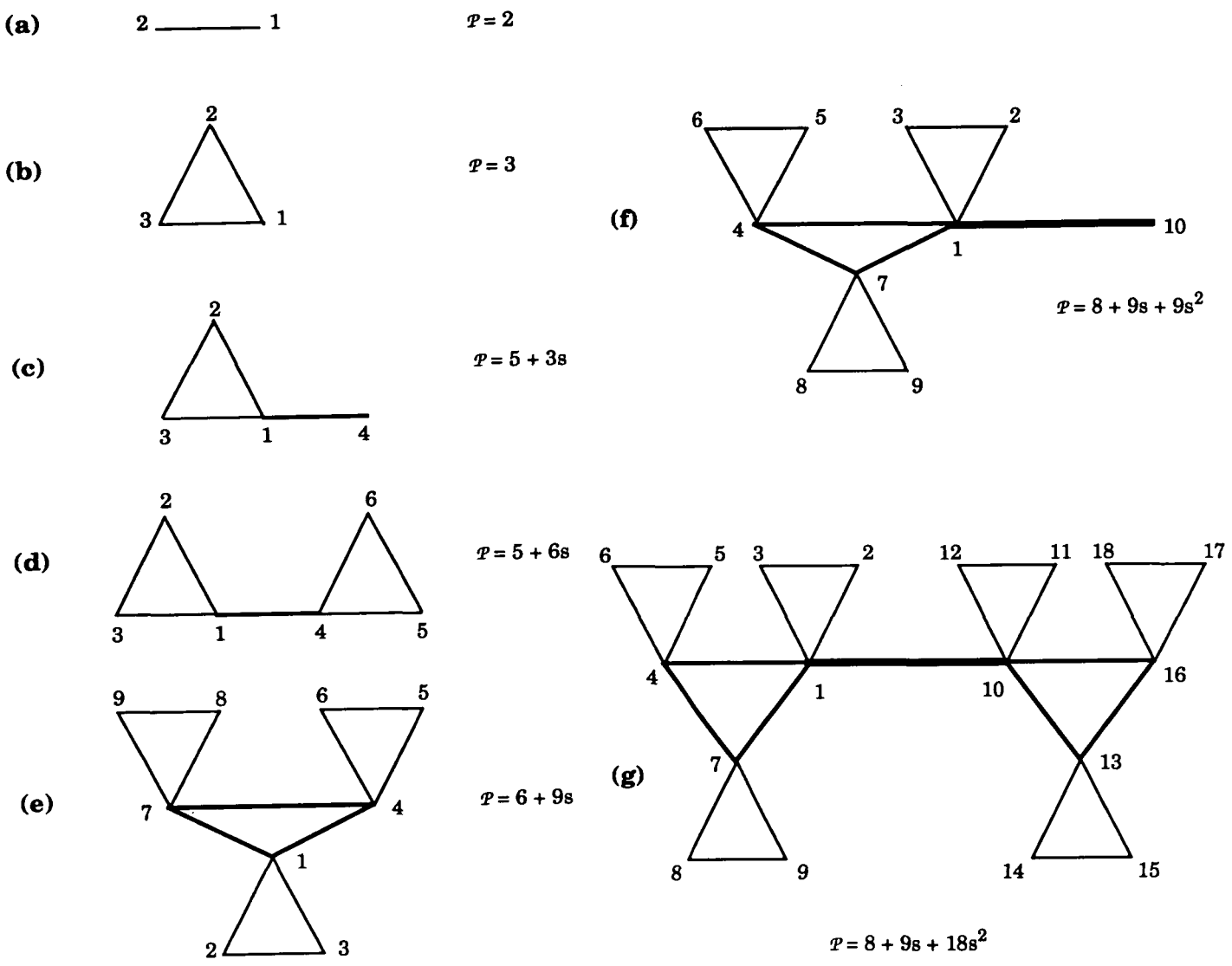


FIGURE 5. MVR structures with their corresponding path functions.

TABLE 3. Path functions corresponding to a range of system sizes and nominal ring sizes

System size	*Nominal ring size* 3	4	5	6
4	$5 + 3s$	4		
5	$5 + 5s$	$6 + 4s$	5	
6	$5 + 6s$	$6 + 6s$	$7 + 5s$	6
7	$6 + 6s$	$6 + 7s$	$7 + 7s$	$8 + 6s$
8	$6 + 8s$	$6 + 8s$	$7 + 8s$	$8 + 8s$
9	$6 + 9s$	$7 + 8s$	$7 + 9s$	$8 + 9s$
10	$8 + 9s + 9s^2$	$7 + 10s$	$7 + 10s$	$8 + 10s$
11	$8 + 11s + 9s^2$	$7 + 11s$	$8 + 10s$	$8 + 11s$
12	$8 + 12s + 9s^2$	$7 + 12s$	$8 + 12s$	$8 + 12s$
13	$8 + 9s + 12s^2$	$8 + 12s$	$8 + 13s$	$9 + 12s$
14	$8 + 9s + 14s^2$	$8 + 14s$	$8 + 14s$	$9 + 14s$
15	$8 + 9s + 15s^2$	$8 + 15s$	$8 + 15s$	$9 + 15s$
16	$8 + 9s + 15s^2$	$8 + 16s$	$9 + 15s$	$9 + 16s$
17	$8 + 9s + 17s^2$	$10 + 16s + 16s^2$	$9 + 17s$	$9 + 17s$
18	$8 + 9s + 18s^2$	$10 + 18s + 16s^2$	$9 + 18s$	$9 + 18s$
19	$9 + 9s + 18s^2$	$10 + 19s + 16s^2$	$9 + 19s$	$10 + 18s$
20	$9 + 11s + 18s^2$	$10 + 20s + 16s^2$	$9 + 20s$	$10 + 20s$

be ranked in any order, depending on the relative magnitude of transmission and synchronization delays. One order we have investigated has ranked transmission delays as being more significant than synchronization between pairs of levels and less significant than synchronizations between three or more levels. Ordering the powers of s by increasing delay contribution we therefore get

$$s^1 < s^0 < s^2 < s^3 < \cdots < s^n,$$

where $n+1$ is the number of levels in a given configuration. We apply this ordering to the path functions of Table 3 and assume in doing so that the coefficients of less significant powers of s are negligible when compared with those of the more significant powers. As our objective is to increase decision-making speed, and therefore to minimize delays, the following best nominal ring sizes are predicted. For system sizes, **m**, from 2 to 5, the best predicted ring size is equal to the system size; in other words a single-ring structure is best. For systems with six to nine nodes, the best nominal ring

size is 3, giving two-level structures. For 10 to 16 nodes the best ring size is 4 and for systems with 17 to 20 nodes $p_{opt} = 5$. In this way, the path function can be used to predict an appropriate nominal ring size for a particular system size. The predicted value can be considered an initial approximation which can be subsequently refined by experiment.

6. SIMULATION EXPERIMENTS

This section presents an evaluation of hierarchical information distribution in distributed systems. The problem we have selected is that of load balancing in a large distributed system comprising a group of processors situated at each of five geographically remote sites. At each site, interprocessor communication occurs over an ethernet with a transmission rate of 10 Mbit/s, which is modelled according to the description provided in Schwartz (1987). It is assumed that each group has its own file server, so that job migration arising from load balancing within each group is by means of command line transfer. The transmission delay due to local job migration is modelled as the time to transfer one minimum-sized ethernet message, that is, 72 bytes, 46 of which are data. The objective of load balancing at a site is to reduce the mean job response time over all the processors at that site.

It is well known (Zhou, 1988) that load balancing performance improves with increasing number of processors, up to a certain limit. Accordingly, we performed preliminary simulation tests which showed that, under the conditions specified above, there was no improvement in system response time for more than 100 processors per site. Since we wished to investigate the conditions under which load balancing might be worthwhile in large systems, we set the number of processors at each site to 100, making a total of 500 processors in the overall system.

The interconnection pattern of the resulting distributed system is shown in Figure 6. The five group file servers are directly connected to each other by point-to-point links with transmission speeds lower than the ethernet rate. Job migration between processor groups must be implemented by migrating the associated job files. We assume that each job has a single associated file whose size is generated from the empirical distribution published for the Unix system by Mullender and Tanenbaum (1984). Transmission delays between groups are modelled as multiples of the time it would take to transfer the job files across an ethernet. In this calculation the maximum amount of data which can be held in an ethernet frame (1500 bytes) is taken into account, the multiplication factor is 10^3 and jobs are transmitted in several frames if necessary. The model incorporates the CPU overhead of message transmission and reception as follows: each time a message is transmitted to or received from a processor, a system

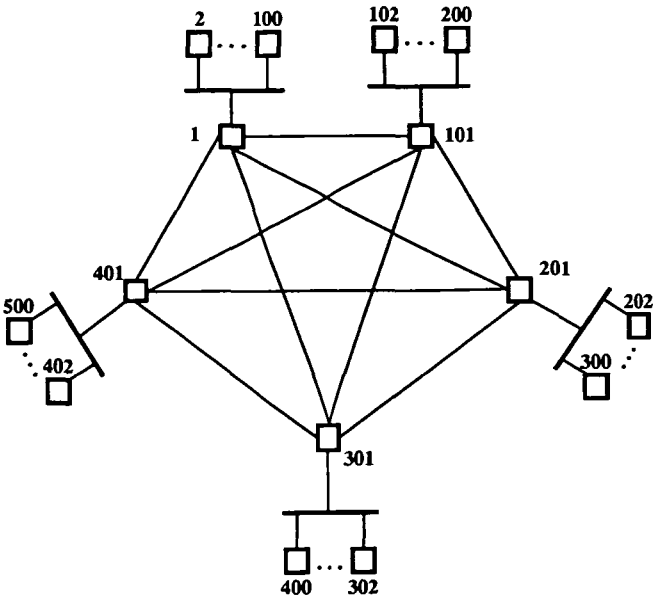


FIGURE 6. Interconnection pattern of the 500-processor simulated system.

process is started at that processor with an execution time of 20 ms.

Our investigation of load balancing in this large distributed system is based on the following reasoning. There is no advantage to balancing load between more than 100 processors. However, if there is no load balancing between groups, a group with a higher average load than that of the other groups has no means of relief. Experiments performed by Zhou (1988) identify centralized load balancing as yielding the best performance. A logical disadvantage of centralized system-wide load balancing is that load balancing is neutralized if the load-balancing processor fails. A compromise is to apply centralized load balancing within each group of processors. Load balancing between groups then allows system-wide scheduling while minimizing performance degradation if one of the load balancing processors fails.

We performed a discrete-event simulation of the system shown in Figure 6, using the following additional parameters: CPU scheduling within each processor is round-robin with a time quantum of 100 ms. Job execution times are generated from an exponential distribution with a mean $1/\mu$ of 1.492 s, which is the execution time mean used in Zhou (1988). Job interarrival times are exponential with a mean $1/\lambda$ which changes in order to vary the offered system load $\rho = \lambda/\mu$.

Three load-balancing algorithms were simulated. For all these algorithms, the load index, which indicates the busyness of each processor, is the number of jobs waiting to use the processor. The performance index is the mean job response time and the objective is to minimize this quantity. The algorithms have the following characteristics in common: The load-balancing decision is initiated by the arrival of a job and the arriving job itself is the one selected for possible remote execution.

The job is eligible for transfer if its execution time is less than a threshold T_{CPU} (set to 100 ms for these tests) and the current length of the CPU queue is greater than a threshold T_1 (equal to 1 for these tests). Algorithmic stability is assured by allowing jobs to be transferred to a remote site at most once before being executed.

The CENTRAL algorithm (Zhou, 1988; Zhou *et al.*, 1993) is well-known and is used here as a basis for comparing the value of partially-distributed load balancing to completely centralized load balancing. The DistCent algorithm is a distributed application of the CENTRAL algorithm based on the two-level structure shown in Figure 6. Since it is known from the preliminary tests that load balancing between more than 100 processors does not improve performance, the Disjoint-Central algorithm, which operates independently in each 100-processor group, provides a benchmark against which the value of including greater numbers of processors in the load balancing activity can be assessed. The information and location policies of these algorithms are as follows:

6.1. CENTRAL

Every P_1 seconds ($P_1 = 3s$ here), processor 1, acting as the *load information centre* (LIC), receives load updates from all the other processors and assembles them into a load vector. If the load of a processor is the same as that sent out the last time, no update needs to be sent to the LIC. The LIC acts as a central scheduler for all the processors. When a job is eligible for load balancing, the processor sends a request to the LIC, together with the current value of its load. The location policy of CENTRAL searches the load vector for the processor with the lowest load and if this load is lower than that of the job's current location by a relative transfer threshold Δ_1 (set to 1 for CENTRAL and DistCent) or more, the job is sent to that processor. Otherwise, the job is to be executed at its current location. If there are several processors with the same shortest queue length, one of them is selected randomly. The originating processor is informed of the execution site and the load vector is updated to reflect this decision.

6.2. DistCent

The network is logically partitioned in accordance with the physical boundaries shown in Figure 6. The ethernet-connected processors form five local groups at level 1. The groups contain processors 1 to 100, 101 to 200, 201 to 300, 301 to 400 and 401 to 500, the LICs for these groups being processors 1, 101, 201, 301 and 401, respectively. Within each group, the CENTRAL algorithm is applied with information period P_1 ($P_1 = 3s$) and relative transfer threshold Δ_1 ($\Delta_1 = 1$). The LICs of the level 1 groups are connected at level 2 and the CENTRAL algorithm applied between these with information period P_2 ($P_2 = 10s$) and relative transfer threshold Δ_2 ($\Delta_2 = 100$), using processor 1 as the coordinator or 'superLIC'. Information distribution occurs in parallel in the five level 1 groups and the single level 2 group. The information collected and distributed for each group at level 2 is the sum of the load indices of all the processors belonging to that group. When a job arrives at a processor it triggers a request to the local level 1 LIC. This in turn consults the superLIC which decides the best group in which the job should execute. If this is not the job's current group, the job is sent to a remote level 1 LIC which then schedules the job for execution within the remote group. Otherwise, the job is executed in its current group at a site determined by the local LIC.

6.3. Disjoint-Central

The network is logically partitioned in accordance with the physical boundaries shown in Figure 6. The ethernet-connected processors form five local groups. The groups contain processors 1 to 100, 101 to 200, 201 to 300, 301 to 400 and 401 to 500, the LICs for these groups being processors 1, 101, 201, 301 and 401 respectively. Within each group, the CENTRAL algorithm is applied with information period P_1 ($P_1 = 3s$) and relative transfer threshold Δ_1 ($\Delta_1 = 1$). There is, however, no load balancing activity between the five groups.

Two types of experiments were carried out. In the first type, the load levels are homogeneous, i.e. the offered load is the same at every processor in every group. In the second type, load levels are homogeneous within groups but heterogeneous between groups, so that one group has a higher offered load than the other four. In all experiments, independent replications were performed to ensure that the results presented are within 5% of the mean value at the 95% confidence level.

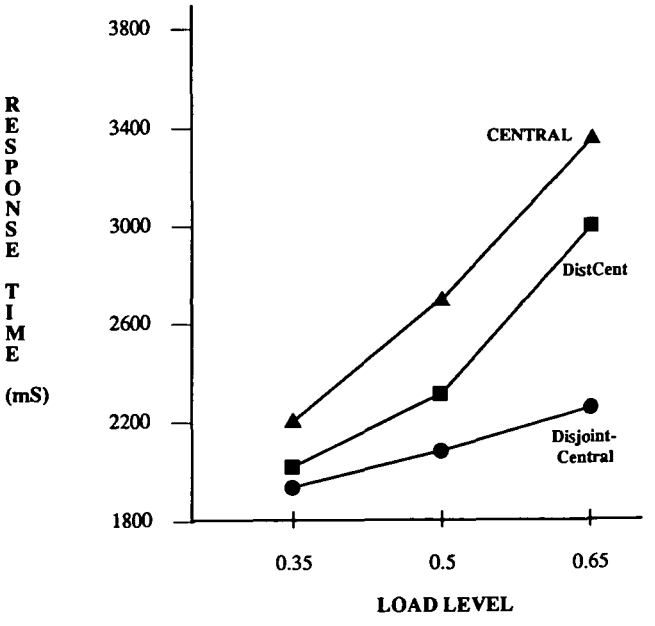


FIGURE 7. System-wide mean job response times plotted against load level for the three load balancing algorithms.

6.4. Experiment 1

This experiment examines the performance of the algorithms under homogeneous loading conditions for different load levels. Figure 7 shows the system response time plotted against load level for the three algorithms. It can be seen that the worst performance is obtained from CENTRAL, which is not surprising since this algorithm performs best for fewer processors. The best results are obtained for the Disjoint-Central algorithm, again supporting the argument against load balancing among large numbers of processors. The DisCent algorithm gives a response time which lies between those of the other two algorithms. Thus, system-wide load balancing is not justified in large distributed systems under homogeneous loading conditions.

6.5. Experiment 2

In this experiment, one group experiences a very high load level ($\rho = 0.8$), while the load level in the other groups is lighter and the same in each of the four groups. The mean system-wide job response times are displayed in Table 4(a) for light load levels of $\rho = 0.35$ and $\rho = 0.5$. As in the case of homogeneous loading, the Disjoint-Central algorithm yields the best performance, followed by the DistCent and CENTRAL algorithms. However, the situation within the heavily-loaded group is different, as shown in Table 4(b). Here, Disjoint-Central gives a very poor performance owing to the impossibility of migrating jobs from this group to relieve the heavy load level within it. The results from the CENTRAL algorithm are better, while the best results are delivered by DistCent.

It can be concluded that, under the stated operational conditions, load balancing among large numbers of processors always results in a cost to the system by way of degraded mean system-wide job response time. However, having no system-wide load balancing may give rise to unacceptable local response times in subsystems. The two-level partially distributed DistCent algorithm consistently performed better than the completely centralised CENTRAL algorithm, while giving better local response in heavily-loaded subsystems than the localized Disjoint-Central algorithm. This validates the hierarchical approach to information distribution. The real value of DistCent lies in the economic advantage of applying it in systems where loading is heterogeneous rather than purchasing extra equipment to deal with localized high load levels.

TABLE 4. Response times in a heterogeneously-loaded system: four groups operate with $\rho = \rho_{light}$ and the remaining group with $\rho = 0.8$

	(a) Systemwide response times (ms)		*(b) Heavily-loaded group response times (ms)*	
Algorithm	$\rho_{light} = 0.35$	$\rho_{light} = 0.5$	$\rho_{light} = 0.35$	$\rho_{light} = 0.5$
Central	2659	2951	2348	2676
DistCent	2489	2713	2147	2150
Disjoint-Central	2323	2413	2987	3244

7. SUMMARY

We have presented an organisation of a distributed system which allows information to be gathered in a parallel, structured fashion. The organization imposes a virtual interconnection pattern on the system such that system processors are grouped in rings which in turn are connected to higher-level rings in a multilevel hierarchical structure. The rings are traversed by information-gathering tokens which terminate at a processor which is potentially a centralized decision maker. These rings can be configured dynamically and patched or reconfigured on the occurrence of processor or link failure and repair. The nominal ring size is an important parameter of the configuration algorithm, and we have shown how an algebraic description of the transmission and synchronization delays in the MVR structure may be formulated in order to determine the virtual ring size which is most appropriate to the delay characteristics of the structure.

Although the organization has been presented here as a hierarchy of rings, many of the points which have been discussed are applicable to a general hierarchical organization where each group of processors in the hierarchy is linked together by a connection pattern which is not necessarily a ring. The configuration algorithm needs only slight modification to substitute another virtual topology for the ring. Vertical information-flow paths must exist in any such structure, so the function of the ICPs will have to be continued in some manner. The problem of weighting information appropriately at each level will exist no matter what topology is used. Likewise, the resolution of conflict between levels is a topology-independent issue.

We have explored the application of a load balancing algorithm in a large distributed system organized as a two-level hierarchy. The performance of the hierarchical algorithm, DistCent, has been compared with that of a centralized algorithm applied to the overall system (Central) and to disjoint subsystems (Disjoint-Central). The hierarchical algorithm has functioned successfully, yielding better performance than the completely centralized algorithm for the large system which we have simulated. It has also been shown to provide relief for heavily-loaded subsystems under asymmetric loading conditions. Comparison of DistCent with Disjoint-Central does, however, indicate that system-wide scheduling in large distributed systems is justified by taking advantage of geographical heterogeneity in system load rather than on the basis of improving overall system response.

ACKNOWLEDGEMENT

The author wishes to thank Mark O'Connor for his assistance with the simulation experiments.

REFERENCES

Ahmad, I. and Ghafoor, A. (1991) Semi-distributed load balancing for massively parallel multicomputer systems. *IEEE Trans. Software Eng.*, **17,** 987–1004.

Casavant, T. L. and Kuhl, J. G. (1987) Analysis of three dynamic distributed load balancing strategies with varying global information requirements. In *Proc. 7th Int. Conf. on Distributed Computing Systems*, pp. 185–192. IEEE Computer Society Press, New York.

Dikshit, P., Tripathi, S. K. and Jalote, P. (1989) SAHAYOG: a test bed for evaluating dynamic load-sharing policies. *Software—Practice and Experience*, **19,** 411–435.

Eager, D. L., Lazowska, E. D. and Zahorjan, J. (1986) Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. Software Eng.*, **12,** 662–675.

Evans, D. J. and Butt, W. U. N. (1994) Load balancing with network partitioning using host groups. *Parallel Comput.*, **20,** 325–345.

Horton, G. (1993) A multi-level diffusion method for dynamic load balancing. *Parallel Comput.*, **19,** 209–218.

Kim, C., Lee, J.-Y. and Park, C.-M. (1993) Hierarchical decision structure for distributed algorithms. In *Proc. 4th Workshop on Future Trends of Distributed Computing Systems*, pp. 203–207. IEEE Computer Society Press, New York.

Maples, C. (1985) Pyramids, crossbars and thousands of processors. In *Proc. 1985 Int. Conf. on Parallel Processing*, pp. 681–688. IEEE Computer Society Press, New York.

Mirchandaney, R., Towsley, D. and Stankovic, J. A. (1989) Analysis of the effects of delays on load sharing. *IEEE Trans. Comp.*, **38,** 1513–1525.

Mullender, S. J. and Tanenbaum, A. S. (1984) Immediate files. *Software—Practice and Experience*, **14,** 365–368.

Schwartz, M. (1987) *Telecommunications Networks: Protocols, Modeling and Analysis.* Addison-Wesley, Reading, MA.

Shin, K. G. and Chang, Y.-C. (1989) Load sharing in distributed real-time systems with state-change broadcasts. *IEEE Trans. Comp.*, **38,** 1124–1142.

Stankovic, J. A. (1989) Decentralized decision making for task reallocation in a hard real-time system. *IEEE Trans. Comp.*, **38,** 341–355.

van Tilborg, A. M. and Wittie, L. D. (1984) Wave scheduling—decentralized scheduling of task forces in multicomputers. *IEEE Trans. Comp.*, **33,** 835–844.

Zhou, S. (1988) A trace-driven simulation study of dynamic load balancing. *IEEE Trans. Software Eng.*, **14,** 1327–1341.

Zhou, S., Zheng, X., Wang, J. and Deslisle, P. (1993) Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Software—Practice and Experience*, **23,** 1305–1336.