Brief Contributions

Period-Based Load Partitioning and Assignment for Large Real-Time Applications

Tarek F. Abdelzaher, *Member*, *IEEE*, and
Kang G. Shin, *Fellow*, *IEEE*

Abstract—We propose a new approach to the problem of workload partitioning and assignment for very large distributed real-time systems, in which software components are typically organized hierarchically, and hardware components potentially span several shared and/or dedicated links. Existing approaches for load partitioning and assignment are based on either *schedulability* or *communication*. The first category attempts to construct a feasible schedule for various assignments and chooses the one that minimizes task lateness (or other similar criteria), while the second category partitions the workload heuristically in accordance with the amount of intertask communication. We propose, and argue for, a (new) third category based on task *periods*, which, among others, combines the ability of handling heterogeneity with excellent scalability. Our algorithm is a recursive invocation of two stages: *clustering* and *assignment*. The clustering stage partitions tasks and processors into clusters. The assignment stage maps task clusters to processor clusters. A later scheduling stage will compute a feasible schedule, if any, when the size of processor clusters reduces to one at the bottom of the recursion tree. We introduce a new clustering heuristic and evaluate elements of the period-based approach using simulations to verify its suitability for large real-time applications. Also presented is an example application drawn from the field of command and control that has the potential to benefit significantly from the proposed approach.

Index Terms—Task allocation, task partitioning, inhomogeneous networks, real-time scheduling.

1 Introduction

Distributed embedded real-time systems are becoming increasingly larger and more complex as the scope of real-time computing extends to more demanding and challenging application domains. Air defense, command and control, battle control, and space exploration, for example, tend to be very large in problem size and potentially extend over a large number of heterogeneous physically dispersed computers. Since violation of timing constraints in these applications may have dire consequences, static workload (usually periodic) must be assigned to processors before run-time and its schedulability verified. The main workload of the above-mentioned applications is periodic in nature. Sporadic and aperiodic tasks can be treated as periodic by allotting them a periodically-replenished execution budget, e.g., a deferrable server [1] or a processor capacity reserve[1] [2]. Thus, we assume that all tasks are periodic.

We propose a new workload partitioning and assignment algorithm for periodic tasks in large heterogeneous real-time systems which attempts to find an assignment of tasks to processors that results in a feasible schedule. The contributions of our approach lie in:

1. its scalability to very large systems by taking advantage of recursive clustering,
2. its ability of handling arbitrary-topology heterogeneous systems,
3. its independence from the run-time scheduling policy, and
4. its use of a period-based clustering heuristic which tends to increase feasible processor utilization bounds and minimize total preemption overhead.

Our approach is built around solving a variant of the minimum k-way cut problem [3] on a set of graphs defined in the subsequent sections.[2] We developed a new $O(n^2\ lg\ n)$ heuristic for solving this variant, where n is the number of vertices in the graph. For all the graphs for which we were able to compute an optimal solution to the minimum k-way cut problem using exhaustive search, our heuristic solutions, on average, were found to be within 3 percent of the optimal. Moreover, in 74 percent of these cases, our heuristic algorithm yielded an optimal solution.

We propose grouping tasks by period. In embedded real-time systems, the number of different task periods among critical tasks (those with hard deadlines) is relatively small, with many tasks running at each of these periods. This is because the entire software hierarchy composing the application is driven by the rates at which the system interacts with the environment through a relatively small number of sensors and actuators. Having each processor run tasks of the same or harmonically related periods has the potential to increase its feasible utilization bound (under fixed-priority scheduling) [4]. Moreover, such a task assignment is likely to reduce the least common multiple (LCM) of task periods on each machine (compared to an assignment that does not group tasks by harmonic periods). Hence, it simplifies schedulability analysis and requires a smaller amount of memory space for storing a precomputed schedule. This is an important practical consideration when one implements table-driven scheduling. Grouping modules by period may also reduce consumed communication bandwidth. Communication occurs mostly among modules of harmonically related periods (typically the same period). For example, in a process control or robotics application, communicating elements of the control loop [5] would typically run at the same period (e.g., the sampling interval), while unrelated control loops might run at unrelated periods. Another consideration is preemption overhead. Having each processor run tasks of comparable periods reduces total preemption overhead, which accumulates quickly when a short-period task keeps preempting a larger-period one, and may reduce blocking due to synchronization conflicts. To summarize, we propose that modules should be grouped whenever possible if their periods are comparable and harmonically related.

The rest of the paper is organized as follows: The next section describes related work putting the proposed load partitioning and task assignment in a comparative perspective. Section 3 describes the system model. Section 4 presents the proposed partitioning and assignment algorithm. Section 5 describes an example application. Finally, Section 6 presents the conclusion of the paper and directions for future work.

1. Reserves can be used for periodic tasks as well.

2. The minimum k-way cut in graph is a partitioning of the graph into k subgraphs (clusters) such that the cost of edges among the resulting clusters is minimized, which is an NP-hard problem [3].

- *T.F. Abdelzaher is with the Department of Computer Science, The University of Virginia, Charlottesville, VA 22903. E-mail: zaher@cs.virginia.edu.*
- *K.G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109. E-mail: kgshin@eecs.umich.edu.*

Manuscript received 1 Oct. 1996; accepted 6 Jan. 1999.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 102119.

```
solve (processor_set, module_set)
{
  /* clustering phase */
  module_clusters = compute_module_clusters (module_set)
  processor_clusters = compute_processor_clusters (processor_set)

  /* assignment phase */
  run_assignment_algorithm (module_clusters, processor_clusters)

  /* do recursion */
  for each processor_cluster in processor_clusters do
    if not (size(processor_cluster) = 1)
    {
      module_cluster = modules_allocated_to (processor_cluster)
      solve (processor_cluster, module_cluster)
    }
}
```

Fig. 1. The assignment algorithm.

2 Related Work

Optimal algorithms [6], [7], [8], [9], [10] have been proposed for different variants of the hard real-time task assignment and scheduling problem, for both homogeneous [9] and heterogeneous [8], [10] systems. For example, in [6], [7], [8], different task-assignment methods are presented to minimize the sum of task execution and communication costs using an implicit enumeration branch-and-bound (B&B) method. The optimal algorithm in [9], while utilizing an elegant approach to reduce the complexity of the search space, makes a fundamental assumption necessary for its correct performance and optimality, namely that task computation and communication delays remain constant, regardless of the particular assignment. Thus, it can be used only for homogeneous systems. The optimal algorithm in [10] allows heterogeneity, but has a scalability limitation. In general, scalability is a common concern with optimal solutions to task allocation since the problem is NP-hard.

To overcome the scalability limitation, heuristic approaches [11], [12], [13], [14], [15], [16] have been proposed for larger instances of the problem. Based on their performance measures, these approaches can be classified as schedulability-based [12], [13], [14] or communication-based [15], [16]. One common way to reduce the allocation search space is to cluster tasks into larger units of allocation, then allocate the resulting task clusters, *not* individual tasks, to available processors. Different flavors of this are proposed in [9], [15], [16]. For the special case of fixed-priority scheduling on homogeneous multiprocessor systems, rate-monotonic analysis has been used to derive near-optimal task clustering policies taking into consideration task periods [17], [18]. In general, clustering heuristics, such as those in [9], [15], typically require the knowledge of module execution times and intermodule communication overhead. Computing these values, which depend on processor speed and link bandwidth, requires a priori knowledge of task-to-processor assignment. Since the assignment is not known in advance, these heuristics are usually applicable only to homogeneous systems. For large distributed applications, parts of which may span several heterogeneous platforms, this is a serious limitation. Our approach differs from other clustering approaches in two respects. First, while, in existing approaches, clustering is done only once, followed by the allocation stage, we use a more scalable recursive approach which iteratively refines the solution. Second, our clustering algorithm can handle heterogeneous systems efficiently.

3 System Model and Problem Formulation

We assume the workload is composed of a set of periodic tasks. Each task T_i is characterized by a set of modules $M_j \in T_i$, an invocation period P_i, and a deadline D_i relative to the start of the invocation period. In this paper, we assume that $D_i = P_i$, which is true of most periodic tasks, since each periodic task invocation must usually complete execution before the next invocation (i.e., it must complete execution within its designated period P_i). Each module M_j has a worst-case computation requirement C_j measured in processor cycles (or other units independent of processor speed). We assume that all invocations of the same module run on the same processor, i.e., task reassignment is not considered here. A module M_j may exchange messages with another module M_k in the same or a different task. Modules M_j and M_k are said to be independent if the start times and finish times of their invocations can be arbitrarily interleaved. Otherwise, the modules are said to have a *synchronization constraint*. The hardware platform on which the application is to be executed is an arbitrary-topology network, possibly composed of several dedicated and shared links. Each processor N_i on the network provides service at rate μ_i, measured in number of processor cycles per unit time. Each link l has bandwidth B_l. Links may be dedicated (point-to-point) or multiple access (e.g., an FDDI ring). A processor may have access to more than one link.

We want to find an assignment of modules to processors, in a distributed system, for which a feasible schedule is likely to be found. As described in Section 4, this problem is reduced to that of graph partitioning, where finding a suitable clustering and assignment corresponds to finding a minimum k-way cut in a set of graphs. Our simulation results have shown the proposed graph partitioning heuristic to work very well.

4 Solution Algorithm

The proposed partitioning and assignment algorithm starts by invoking the **solve** function described in Fig. 1 with the total processor set and module set as the input parameters. This function executes both the clustering and assignment phases, then calls itself recursively. Section 4.1 addresses processor clustering. It deals with issues of processor heterogeneity (each processor is assumed to have a different speed) and network topology. Section 4.2 addresses task clustering. Section 4.3 details and evaluates the minimum k-way cut heuristic used to solve the clustering problem. For generality, the minimum k-way cut

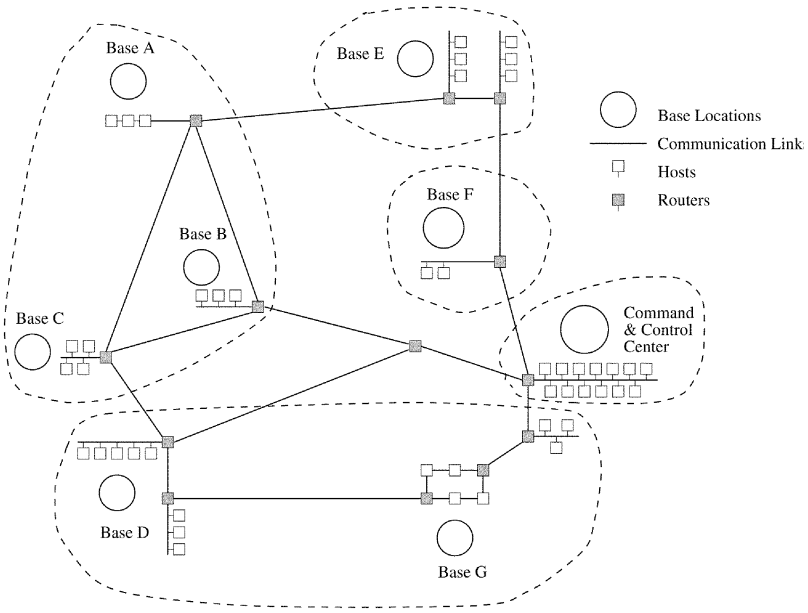

Fig. 2. The ground base resource network.

heuristic is intended to be independent of the particular way the input graph is computed. Finally, Section 4.4 deals with assignment of task clusters to processor clusters. It addresses issues of processor utilization and presents aspects of the algorithm that attempt to bound the utilization such that deadlines are not missed.

4.1 Processor Clustering

The problem of processor clustering is solved by finding a minimum k-way cut in a graph called the *processor attraction graph*. In the processor attraction graph, each vertex represents a processor and each edge represents a communication link connecting two processors. Processor clustering attempts to identify subgroups (clusters) of processors to be treated as single processing units in the subsequent assignment phase. Workload will be allocated to these units as a whole and the details of particular module assignment to individual processors within a cluster (unit) will be determined during later iterations. In general, given two processors N_i and N_j of speeds μ_i and μ_j, interconnected by a link of bandwidth B_{ij}, the larger the ratio, $B_{ij}/(\mu_i + \mu_j)$, of their network interconnection bandwidth to their computing bandwidth, the more desirable it is to group them in the same cluster. We let this ratio be the attraction force (edge cost) in the processor attraction graph.

If more than two processors share the same communication link, its capacity must be partitioned among these processors. In most real-time communication technologies such as ATM or FDDI, there exist means for communication bandwidth reservation. Thus, if the communication bandwidth reserved for N_i and N_j is b_i and b_j, respectively, then $B_{ij} = b_i + b_j$. The approach does not apply to technologies with no communication bandwidth reservation capability such as Ethernet. We also do not explicitly consider multicast and broadcast capabilities. Each message is assumed to have a single destination, even when communicated over a broadcast-supporting medium.

Processors on different LANs are typically interconnected via routers. Although routers are not a target for task assignment, they must be considered in the processor attraction graph because they affect communication delays. Consider a leaf router N_r connecting some subnet N to an internetwork. All communication to/from subnet N passes through router N_r. We expect the communication volume through N_r to be proportional to $\sum_i \mu_i$. Thus, we set $\mu_r = \sum_i \mu_i$, where the summation is carried over all processors N_i in subnet N. In general, if a subnet N has n routers interfacing it to other networks, then, assuming balanced traffic, μ_r for each router is given by $\mu_r = (\sum_i \mu_i)/n$. Finally, routers that are connected to other routers inherit (the average of) their μ_r values.

Fig. 2 demonstrates the result of clustering 59 nodes (of which 13 are routers and 46 are processors), distributed over eight ground defense bases, into five clusters. We used the heuristic described in Section 4.3 to find a minimum 5-way cut in the processor attraction graph. For simplicity of presentation, let all LANs be 100 Mbps FDDI rings, all dedicated links be of T1-link type (1.544 Mbps bandwidth), and all processors be identical, of speed μ, except processors at Base A whose speed is 3μ and processors at Base F whose speed is 2μ. Finally, let link bandwidth be partitioned among processors sharing the same FDDI ring in proportion to their speed. Edge costs are then computed for the processor attraction graph. For example, edge costs between any two processors N_i and N_j on Base B is $B_{ij}/(\mu_i + \mu_j) = (100/3Mbps)/(2\mu)$. Similarly, recalling the definition of μ_r (for routers), the attraction force between subnet routers at Base A and Base C is $1.544Mbps/(9\mu + 4\mu)$. The minimum 5-way cut resulting from assigning edge costs as described above is shown in Fig. 2. We can observe the following advantageous properties:

- Cluster boundaries coincide with LAN boundaries, which is intuitively appealing since, in our example, processors on a LAN are connected by higher bandwidth links (100 Mbps) than those separated by intermediate routers (1.544 Mbps). Cluster boundaries should run across slower links.
- The LANs at Base A, B, and C were merged into one cluster, which is a good choice, because these LANs are interconnected by a complete graph (each is directly connected to the other two). This rich interconnection makes it less likely that any of the three point-to-point links in the cluster will be congested and, thus. enables the cluster to run more tightly-coupled tasks.
- The larger LAN at the Command and Control Center has not been merged with any neighboring LANs because, by virtue of its size, it tends to create a bottleneck at its router. Putting this LAN in a separate cluster makes the load

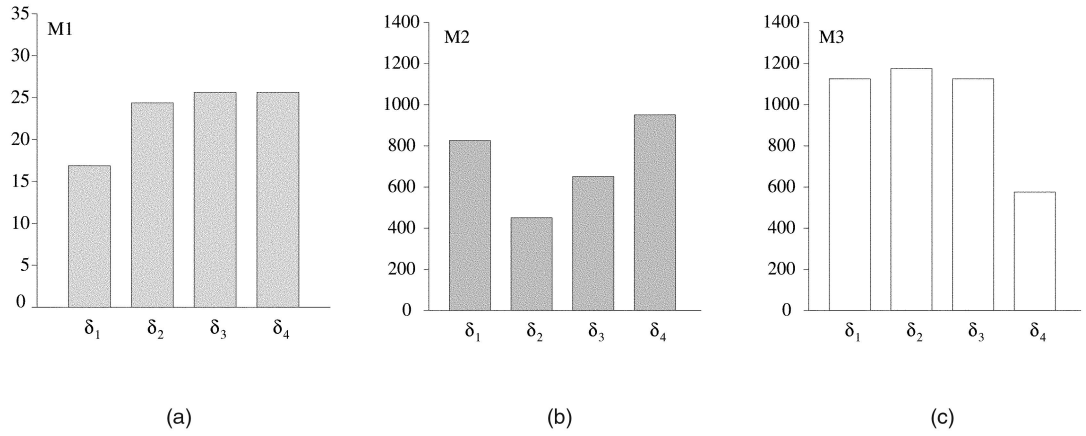

Fig. 3. Comparing allocation heuristics for random task sets. (a) Preemption cost, M1. (b) Normalized LCM, M2. (c) Bandwidth, M3.

assigned to it relatively independent of that assigned to other clusters, thus alleviating the potential bottleneck problem.

- The LANs at Base G and D have been merged with their neighboring ones because they have a small processor per router ratio, which makes their routers less of a bottleneck. The same argument applies to merging the two LANs at Base E.

4.2 Task Clustering

We now describe how modules are clustered for the subsequent assignment purpose. We construct a module attraction graph, where each vertex represents a module. An edge exists between two modules if and only if they either communicate or share a synchronization constraint. Clustering such modules together attempts to reduce synchronization overhead and consumed network bandwidth. Let the total number of modules be M. We would like to find a set of clusters that, if assigned to different processor clusters, will minimize the following metrics:

M1. Normalized Preemption Cost:

$$\sum_k \{ \sum_{ij|P_j>P_i, M_i, M_j \in Cluster\ k} \lceil P_j/P_i \rceil \}/M.$$

This cost refers to the worst-case number of preemptions per module. Note that, for two modules M_i, M_j with $P_i \leq P_j$, M_i may preempt a single invocation of M_j no more than $\lceil P_j/P_i \rceil$ times.

M2. Normalized LCM:

$$Avg_k\{LCM_{i|M_i \in Cluster\ k} P_i / Avg_{i|M_i \in Cluster\ k} P_i\}.$$

It refers to the average ratio (over all clusters) between the LCM of periods of all modules in each cluster divided by the average period of the modules in that cluster. Ideally, this average should be 1, meaning that each cluster contains modules of the same period only.

M3. Intercluster communication.

Without loss of generality, let P_i and P_j be the periods of two communicating modules M_i and M_j, respectively, with $P_i \leq P_j$ (recall that periods and deadlines are the same in our model). Let V_{ij} be the average communication volume between them (in bytes per second). The following edge cost expressions were compared:

- $\delta_1 = P_i/P_j$. This expression attempts to cluster modules with closer periods (to reduce the number of preemptions).
- $\delta_2 = P_i/LCM(P_i, P_j)$. This expression attempts to cluster modules with similar periods if they are harmonic multiples (in which case $LCM(P_i, P_j) = P_j$). Unlike δ_1, it penalizes module pairs with non-harmonic periods (where $LCM(P_i, P_j) > P_j$). Clustering modules of harmonic periods increases the feasible processor utilization bound [4].
- $\delta_3 = P_j/LCM(P_i, P_j)$. This expression attempts to cluster modules with periods which are harmonic multiples. Unlike δ_1 and δ_2, it does not penalize modules for having significantly different periods as long as they are harmonic.
- $\delta_4 = V_{ij}/P_i$. This expression attempts to cluster modules which consume a higher communication bandwidth. This metric is added for comparison only.

We conducted several experiments to evaluate the edge cost expressions $\delta_1, \ldots, \delta_4$, in terms of performance metrics $M_1, \ldots, M_3$. To eliminate bias for a particular input generation process, we conducted two sets of texperiments that differ in the way task graphs are generated.

In the first set of experiments, 60 task graphs of 100 modules and 300 edges each were generated. Module periods were generated at random with a uniform distribution between 10 and 100. Messages sizes were chosen randomly with a discrete uniform distribution between 1 and 1,000 bytes. Edge costs were computed for the corresponding module attraction graph using each of $\delta_1, \ldots, \delta_4$ and the minimum 10-way cut was found in each case for each task graph. In the second set, we repeated the experiments after modifying the task graph generation process. In this set, only a small number of different module periods were used and messages were communicated with a higher probability between modules of the same period.

Figs. 3 and 4 compare the performance of $\delta_1 \ldots \delta_4$ in terms of our three performance metrics, namely, the normalized preemption cost (M1), normalized LCM (M2), and consumed communication bandwidths (M3) in the two sets of experiments. Each data point is the average of 60 experiments. We can observe that the LCM in Fig. 3 is much higher than that in Fig. 4. This is because periods of individual modules in the first set of experiments have been generated at random. The LCM is significantly reduced for systems where module periods tend to be related, or the number of different periods is small.

Period-based heuristics yielded a slightly better normalized preemption cost and a larger improvement in normalized LCM over the communication-based heuristic. In both figures, δ_2 minimizes LCM. It also performs favorably with respect to the number of preemptions (which depend on the range of module

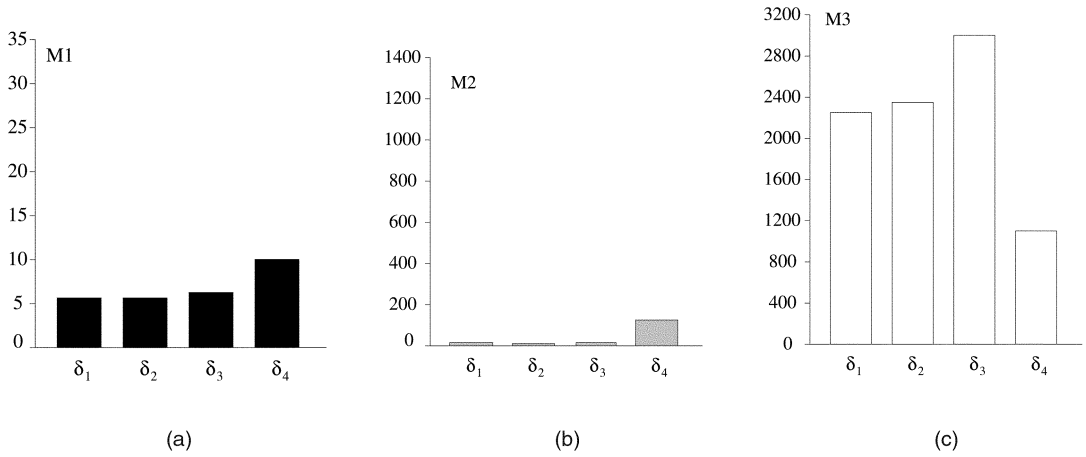

Fig. 4. Comparing allocation heuristics for harmonic task sets. (a) Preemption cost, M1. (b) Normalized LCM, M2. (c) Bandwidth, M3.

periods within a cluster), although δ_1 performs slightly better in this regard. This was expected since δ_2 is sensitive to the increase of period ratio as well as that of LCM, while δ_1 is concerned with the period ratio only. On the other hand, δ_4 results in less communication bandwidth consumption since it creates higher attraction between more heavily communicating modules, thus making them more likely to be coallocated. The above results were obtained for systems where intertask communication volume was not dominant, which is the model we consider in this paper (i.e., only small messages were exchanged among tasks). We use δ_2 for computing edge costs. Unlike δ_1, it tends to separate modules with non-harmonic periods into different clusters, even if those periods are close. Thus, it is more likely to increase feasible utilization bounds [4]. Note that if all periods allocated to the same processor are made harmonic, the module set will be schedulable even at 100 percent processor utilization. For applications where intertask communication is large, δ_4 should be used for computing edge-costs since it performs best in terms of minimizing communication cost.

4.3 The k-Way Cut Heuristic

In this section, we describe the heuristic used for clustering both the processor attraction and module attraction graphs whose edge costs have been computed as described in Sections 4.1 and 4.2, respectively. This is analogous to the minimum k-way cut problem. Several algorithms have been proposed to solve it, e.g., [19], [20], [21] for the special case of $k = 2$, [22], which is specific to VLSI design, and [23], [24], which use randomized algorithms that take a long time to converge to a good solution. We propose a simple heuristic that was developed with our hierarchical system model in mind. Let us define the cost of a vertex in a graph as the sum of the costs of all edges incident to the vertex. It can be easily shown that, in any graph, the sum of vertex costs defined this way is exactly twice the sum of edge costs. If each vertex represents a cluster, the sum of edge costs is also the cost of the cut. Thus, to minimize cut cost, we can minimize the sum of vertex costs. Vertices are sorted in decreasing order of cost in a heap. The highest-cost vertex is popped off the stack and merged with the neighbor in the graph along the highest-cost incident edge. The resulting vertex is reinserted in the heap with a single O (lg V) operation, where V is the number of vertices. Merging is repeated until the desired number of clusters remains. For example, applying this heuristic to the graph in Fig. 5a, we obtain the graphs in Fig. 5b, Fig. 5c, and Fig. 5d, yielding an optimal solution.

To demonstrate the efficiency of the proposed heuristic, we conducted two sets of experiments. In the first experiment, we compare the relative performances of different clustering heuristics (including ours) with that of an optimal algorithm. The optimal algorithm uses exhaustive search for the minimum-cut solution. Since this problem is NP-hard, we limit ourselves to the special case of a two-way cut and consider graphs of 10 to 25 vertices with, on average, four to six incident edges per vertex. Graphs are generated by first constructing the required number of vertices, then randomly generating the required number of edges among them. Edge costs are uniformly distributed. The following heuristics were considered:

K and L: Kernighan and Lin bisection algorithm [19]. It starts with a random cut then swaps vertices to improve cut cost until no further improvement can be made.

Best Neighbor: Select vertices at random. Find the highest-cost edge incident to the selected vertex. Join the two endpoints of that edge. Repeat this until only two clusters remain.

The Proposed Heuristic.

Optimal Algorithm: Uses exhaustive search.

Random: Randomly cuts the graph into two subgraphs.

Table 1 summarizes comparison results for 120 different graphs. Column 2 of the table gives the average performance of each heuristic. It is measured by the ratio of the optimal solution to the solution found by the heuristic, averaged over all 120 cases. Column 3 gives the percentage of 120 cases for which the given heuristic was able to find the exact optimal-cost solution.

As can be seen from Table 1, the proposed heuristic finds the optimal solution an order of magnitude more often than the best of the other heuristics compared. On average, the cost achieved by our heuristic is within 3 percent of the optimal. The run-times of all heuristics were found insignificant (of the order of milliseconds)

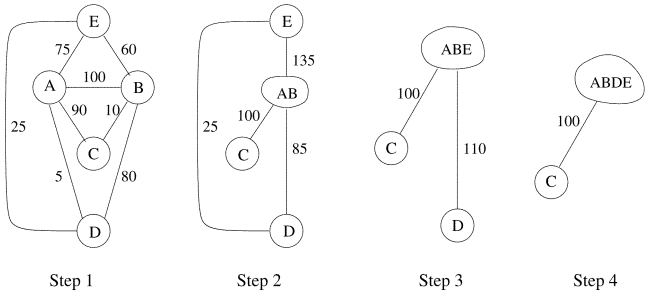

Fig. 5. A clustering example. (a) Step 1. (b) Step 2. (c) Step 3. (d) Step 4.

TABLE 1
Comparison of Clustering Methods

Algorithm	Average Performance	Optimal Solutions
Random	13%	0%
K and L	22%	0%
Best Neighbor	59%	7%
Our Heuristic	97%	74%
Optimal	100%	100%

and, hence, not measured. Given the off-line nature of our partitioning problem, run-times of such magnitude will not constitute an important basis for choosing one of the compared heuristics.

In order to assess the scalability of the approach to very large system sizes, we conducted 30 other experiments on systems of 10,000 vertices. Each system was organized into a three-layer hierarchy. The attraction forces within a layer were set, on average, to be an order of magnitude higher than the attraction force across successive layers. Excessive computational requirements make it impossible to use an optimal algorithm for systems of such size. We compared the minimum-cut cost obtained from our heuristic to that obtained from the best neighbor heuristic and the random cut. Costs were averaged over all conducted experiments, and normalized with respect to that of our heuristic. Table 2 shows the approximate relative figures of cut cost obtained. It can be seen that the cost achieved using our heuristic is far superior to the other costs in the table. The run-time of the slowest of all compared heuristics was of the order of seconds for this problem size, and as such was dismissed as an insignificant factor in choosing the heuristic to use.

4.4 Cluster Allocation

This section describes the algorithm which assigns module clusters to processor clusters. It is invoked after the clustering phase. We cast this problem into that of finding a minimum k-way cut in a graph, called an *assignment graph*. Each node in the assignment graph represents either a processor cluster or a module cluster. Attraction forces are set as described below.

Processor-processor cluster force: This force is set to $-\infty$ to prevent processor clusters from being merged in the process of assignment.

Module-module cluster force: We use δ_2 as the attraction force between module clusters. In other words, the cost of an edge between two module clusters in the assignment graph is equal to the cost of the edge between these two clusters in the reduced module attraction graph (the graph that results from module clustering).

Module-processor cluster force: The force between processor cluster N and module cluster M determines the desirability of assigning M to N. It should account for real-time constraints and differences in processor (cluster) speed in a heterogeneous system, as well as be sensitive to load distribution to avoid processing bottlenecks. Processor utilization captures both processing speed and applied load. For schedulability, this utilization needs to be lower than a specified bound U_N. The derivation of U_N has been adequately addressed in the real-time scheduling literature. In general, it depends on the real-time scheduling policy used. For example, for the case of independent tasks and EDF scheduling, $U_N = 1$, whereas, for rate monotonic scheduling, $U_N = 0.69$. We choose the module-processor cluster force to be $U_N - U$, where U is the total utilization of the processor cluster, should the module cluster be assigned to it. The utilization is computed by summing the contributions of all modules already assigned to N and all modules contained in M, i.e., $U = \sum_i C_i/\mu P_i,\ M_i \in M \cup N$ (where $M_i \in N$ if M_i has already been assigned to N) and $\mu = \sum_j \mu_j,\ N_j \in N$. Note how the speed of the cluster is the summation of its individual processor speeds, which accounts for heterogeneity among different processors. This cost expression tends to attract module clusters to less utilized processors, keeping processor utilization below the schedulable bound if possible. We assume that module deadlines coincide with their periods. Thus, deadlines are implicitly taken into consideration in the above expression. Reducing the utilization, as defined above, below the schedulable bound will mean that all deadlines will be satisfied.

5 An Application Example

We tested the algorithm using a command and control application where a set of sensors detect incoming threats (e.g., enemy planes or missiles) in a battle scenario. The details of the application are omitted for space limitations. Table 3 compares the values of the performance metrics for task assignments generated by two algorithms, the one proposed in this paper and the algorithm in [15]. Each table entry gives the ratio of the metric value obtained by the latter algorithm to that obtained by ours. Two cases are presented. In Case 1, nonharmonic task periods chosen (LCM >> maximum period). Fifty percent of all communication was generated between modules of the same period. In Case 2, mostly harmonic task periods were used (LCM = twice the maximum period). Ninety percent of communication was generated between modules of the same period.

TABLE 2
Normalized Cut Costs

Algorithm	Normalized cut cost
Our Heuristic	1
Best Neighbor	100
Random	50000

TABLE 3
Relative Performance

	M_1 ratio	M_2 ratio	M_3 ratio
Case 1	9.7	8.5	17
Case 2	10.1	1.8	26

From Table 3, note that period-based clustering resulted in roughly an order of magnitude improvement in estimated preemption overhead because it tends to coallocate modules of the same period. The improvement in the LCM metric was high in Case 1, where the LCM of all task periods is very high. On the other hand, it was low in Case 2, where the LCM of all task periods is low, precluding further reduction. The most surprising result, however, is the great improvement in consumed communication bandwidth when using our heuristic. We believe this is because the approach of [15] requires the number of vertices in each partition to be *fixed*. Ours, on the other hand, allows this number to vary, thus gaining an extra degree of freedom in searching for optimal cuts. The improvement in consumed communication bandwidth is greater in Case 2 because, in Case 2, there is a tighter correlation between intermodule communication and module periods than in Case 1.

6 Conclusion

In this paper, we presented a new approach to partitioning and assignment of large real-time applications, which groups tasks by period. The main appeal of the approach is its ability to provide both potential for scalability and support for system heterogeneity. Scalability is achieved by utilizing a recursive divide-and-conquer technique which groups modules and processors into respective clusters, then maps module clusters to processor clusters and refines the resulting assignment recursively. Both clustering and assignment are based around the finding of a minimum k-way cut in appropriately defined graphs. Another contribution of this work is a new efficient heuristic for solving the k-way cut problem. We have shown that for the case of 2-way cut with 10-25 vertices and four to six incident edges per vertex, our algorithm finds on average a solution within 3 percent of the optimal and find the true optimal solution 74 percent of the time. For systems of 10,000 modules, it performed at least two orders of magnitude better than other heuristics. In Section 5, we presented a comparison with a different approach performed on application data drawn from the field of command and control. The comparison shows the promise of our approach.

Acknowledgments

The work reported in this paper was supported in part by the Defense Advanced Research Projects Agency, monitored by the U.S. Air Force Rome Laboratory under Grant F30602-95-1-0044. Any opinions, findings, and conclusions or recommendations are those of the authors and do not necessarily reflect the views of the funding agency.

References

[1] J.K. Strosnider, J.P. Lehoczky, and L. Sha, "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *IEEE Trans. Computers,* vol. 44, no. 1, pp. 73-91, Jan. 1995.

[2] C.W. Mercer, S. Savage, and H. Tokuda, "Processor Capacity Reserves: An Abstraction for Managing Processor Usage," *WWOS,* pp. 129-134, 1993.

[3] W.-P. Chiang, "Optimal Graph Clustering Problems with Applications to Information System Design (Partitioning, Database," PhD thesis, Univ. of Michigan, 1984.

[4] L. Sha, R. Rajkumar, and S.S. Sathaye, "Generalized Rate Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems," *Proc. IEEE,* vol. 82, no. 1, pp. 68-82, Jan. 1994.

[5] M. Alfano, A. Di-Stefano, L. Lo-Bello, O. Mirabella, and J.H. Stewman, "An Expert System for Planning Real-Time Distributed Task Allocation," *Proc. Florida AI Research Symp.,* Key West, Fla., May 1996.

[6] C.C. Shen and W.H. Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion," *IEEE Trans. Computers,* vol. 34, no. 3, pp. 197-203, Mar. 1985.

[7] J.B. Sinclair, "Efficient Computation of Optimal Assignments for Distributed Tasks," *J. Parallel and Distributed Computing,* vol. 4, pp. 342-362, 1987.

[8] P.Y.R. Ma et al., "A Task Allocation Model for Distributed Computing Systems," *IEEE Trans. Computers,* vol. 31, no. 1, pp. 41-47, Jan. 1982.

[9] K. Ramamritham, "Allocation and Scheduling of Precedence-Related Periodic Tasks," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, no. 4, pp. 412-420, Apr. 1995.

[10] D.-T. Peng and K.G. Shin, "Static Allocation of Periodic Tasks with Precedence," *Proc. Int'l Conf. Distributed Computing Systems,* pp. 190-198, June 1989.

[11] C.-J. Hou and K.G. Shin, "Replication and Allocation of Task Modules in Distributed Real-Time Systems," *Proc. 24th IEEE Symp. Fault-Tolerant Computing Systems,* pp. 26-35, June 1994.

[12] S.B. Shukla and D.P. Agrawal, "A Framework for Mapping Periodic Real-Time Applications on Multicomputers," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 7, pp. 778-784, July 1994.

[13] E. Wells and C.C. Caroll, "An Augmented Approach to Task Allocation: Combining Simulated Annealing with List-Based Heuristics," *Proc. Euromicro Workshop,* pp. 508-515, 1993.

[14] Y. Oh and S.H. Son, "Scheduling Hard Real-Time Tasks with Tolerance to Multiple Processor Failures," *Multiprocessing and Multiprogramming,* vol. 40, pp. 193-206, 1994.

[15] T.-S. Tia and J. W.-S. Liu, "Assigning Real-Time Tasks and Resources to Distributed Systems," *Int'l J. Minim and Microcomputers,* vol. 17, no. 1, pp. 18-25, 1995.

[16] S.S. Wu and D. Sweeping, "Heuristic Algorithms for Task Assignment and Scheduling in a Processor Network," *Parallel Computing,* vol. 20, pp. 1-14, 1994.

[17] A. Burchad, J. Liebeherr, Y. Oh, and S. Son, "Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems," *IEEE Trans. Computers,* vol. 44, no. 12, pp. 1,429-1,442, Dec. 1995.

[18] Y. Oh and S. Son, "Allocating Fixed-Priority Periodic Tasks on Multiprocessor Systems," *J. Real-Time Systems,* vol. 9, no. 3, Sept. 1995.

[19] W. Kernighan and S. Lin, "An Efficient Heuristci Procedure for Partitioning Graphs," *Bell System Technical J.,* vol. 49, pp. 291-307, 1970.

[20] D.G. Schweikert and B. Kernighan, "A Proper Model for Partitioning of Electrical Circuits," *Proc. Ninth Design Automation Workshop,* pp. 57-62, 1972.

[21] C.M. Fiduccia and R.M. Mettheyses, "A Linear Time Heuristics for Improving Network Partitions," *Proc. 19th Design Automation Conf.,* pp. 175-181, 1982.

[22] C. Kring and A.R. Newton, "A Cell-Replicating Approach to Mincut-Based Circuit Partitioning," *Proc. IEEE Int'l Conf. Computer-Aided Design,* pp. 2-5, Nov. 1991.

[23] S. Kirkpatrick, C. Gallet, and M.P. Vecchi, "Optimization by Simulated Annealing," *Science,* vol. 220, pp. 671-680, May 1983.

[24] J. Cohoon and W. Paris, "Genetic Placement," *Proc. IEEE Int'l Conf. Commputer-Aided Design,* pp. 422-425, 1986.